EC325 Lecture Notes

EC325: Econometrics | Colby College

Prof. Caraher

March 16, 2026

Table of contents

Preface

Welcome to EC 325: Econometrics at Colby College! This booklet contains the lecture notes, interactive exercises, and R tutorials that will guide you through your first course in econometrics.

What is Econometrics?

Econometrics is the application of statistical methods to economic data with a particular emphasis on *causal inference*—figuring out whether one thing actually *causes* another, rather than merely being correlated with it. This distinction matters enormously for policy: if we want to know whether raising the minimum wage reduces employment, whether health insurance improves health, or whether education increases earnings, we need tools that go beyond simple correlations.

This course will teach you those tools. By the end of the semester, you'll be able to read and critically evaluate empirical research in economics, conduct your own quantitative analyses, and—most importantly—think carefully about when statistical evidence can and cannot support causal claims.

About These Notes

These notes are designed to be a *companion* to the course, not a replacement for the textbook or class participation. They aim to:

- Present core concepts in accessible, narrative form
- Provide extensive R code examples you can run yourself
- Include interactive exercises to test your understanding
- Emphasize intuition through simulations before introducing formulas

The content draws heavily from the following texts, which are required or recommended for this course:

- Wooldridge (2019) — The primary textbook, providing rigorous coverage of econometric methods

- Angrist and Pischke (2014) — An accessible introduction to causal inference and the "credibility revolution" in economics
- Bailey (2020) — A practical guide to doing econometrics in R
- Wickham, Çetinkaya-Rundel, and Grolemund (2023) — The definitive guide to data science in R with the tidyverse

Structure of the Course

The course is organized into three modules:

Module 1: Correlation, Causation, and Regression We begin by establishing why correlation doesn't imply causation and what additional assumptions we need to make causal claims. We then develop the core tool of econometrics—Ordinary Least Squares (OLS) regression—first in its simple bivariate form, then with multiple control variables.

Module 2: Statistical Inference and Real-World Regression With the mechanics of OLS in hand, we turn to quantifying uncertainty. How confident should we be in our estimates? We develop the tools of hypothesis testing, confidence intervals, and statistical significance. We then explore practical issues like functional form, dummy variables, and interactions.

Module 3: Causal Inference Methods The final module covers the major identification strategies economists use to make causal claims from observational data: instrumental variables, difference-in-differences, and regression discontinuity designs.

Roadmap Through the Midterm

The first half of the course (through the midterm) covers the foundations:

- **Chapter 1**: We begin with the central question—does correlation imply causation?—using the example of health insurance and health outcomes. You'll see why simple comparisons can be misleading and why we need econometric tools.
- **Chapter 2**: Before we move onto econometric theory, we cover best practices for data best practices and reproducible research in R
- **Chapter 3**: We introduce the "gold standard" of causal inference—Randomized Control Trials (RCTs)—and the Potential Outcomes Framework, which provides the mathematical foundation for thinking about causality.
- **Chapter 4**: We develop bivariate OLS regression, deriving the estimator, understanding when it gives us causal estimates, and diagnosing when it fails due to omitted variable bias.

- **Chapter 5**: We extend OLS to multiple independent variables, learning how to "control for" confounders and interpret coefficients in the multivariate setting.
- **Chapter 6**: We tackle statistical inference—hypothesis testing, t-statistics, p-values, and confidence intervals—learning to distinguish real effects from statistical noise.

Statistical Software

Modern econometrics is done with statistical software that you interact with through code. In this course, we use **R**, a free and open-source programming language widely used in academia, government, and industry.

Why R? Three reasons:

1. **It's free.** Unlike Stata or SAS, you can install R on any computer and use it forever at no cost.
2. **It's powerful.** R can handle everything from simple regressions to cutting-edge machine learning, beautiful data visualizations, and reproducible research documents (these notes were entirely written in R!).
3. **It's in demand.** R skills are valued by employers in consulting, tech, finance, public policy, and academic research. It regularly ranks among the most widely used programming languages.

Don't worry if you've never programmed before. We'll start from the basics. See Appendix A for instructions on getting set up.

A Note on Learning Econometrics

Econometrics is challenging. It combines economic intuition, statistical theory, and programming skills, often all at once. If you find yourself struggling (and you will), that's normal. Here's my advice:

- **Run the code yourself.** Don't just read the examples—type them out, run them, break them, fix them. You learn programming by doing.
- **Focus on intuition first.** Before memorizing formulas, make sure you understand *why* a method works.
- **Ask questions.** Come to office hours, post on the discussion board, work with classmates.
- **Be patient with yourself.** The skills you're learning will serve you well beyond this course, whether you pursue graduate study, policy work, or a career in the private sector.

About the Instructor

I am a recent PhD from UMass Amherst. My research focuses on the relationship between public policy and maternal, infant, and child health, with particular attention to racial and ethnic health disparities. In my work, I use many of the methods you'll learn in this course.

You can find out more about me and my research at www.raymondcaraher.com.

These notes are a work in progress. If you spot errors or have suggestions for improvement, please let me know!

1 Introduction

Key Questions

- What is the difference between correlation and causation?
- What does an econometrics research project entail?
- What are the main types of data formats we will work with?
- What is statistical software and why is it useful?

Suggested Readings

- Wooldridge (2019), Ch. 1
- Wickham, Çetinkaya-Rundel, and Grolemund (2023), Ch. 1

1.1 What is this Class About?

This class is obviously dedicated to the study of econometrics (endearingly shortened 'metrics). But what exactly does econometrics entail? What differentiates it from broader terms such as "statistical analysis" or "data analysis"?

From Wooldridge (2019), we get a classic definition of econometrics:

> "...statistical methods for estimating economic relationships, testing economic theories, and evaluating and implementing government and business policy."

However, my view of econometrics is at the same time, broader and more narrow:

Definition of Econometrics

The practice of using statistical, quantitative methods to study social phenomena and provide insights into public policy. More specifically, modern econometrics has focused on identifying the **causal effects** of social phenomena/policy as opposed to **correlations**.

In other words, I view the practice of econometrics as the use of quantitative methods to understand the social world around us, with a special (though not exclusive) lens focusing on separating correlation from causation.

1.2 Does Having Health Insurance *Cause* You to be Healthier?

Let's think about a critical question in public policy: *does having health insurance make someone healthier*? This is especially an important question in the context of the United States, which does not have a public health insurance system. Instead, it is a predominantly private system entangled with a complex net of social safety nets meant to provide a patchwork of coverage to the most marginalized.

However, as of 2023, 9.5% of U.S. adults do not have any health insurance coverage.[1] Would the US as a whole be better off guarantee health insurance coverage for everyone? If health insurance makes people healthier, then there may be a strong case to do so. If not, the benefits of such a system would become, at the very least, more opaque.

Let's take a look at life expectancy in the US (a course but important measure of population health), relative to similar high-income, developed countries. Figure 1.1 shows the average life expectancy from 1990 to 2021 for the US, UK, Canada, Australia, and a number of other high-income countries. As can be seen, the US performs considerably worse in terms of life expectancy, and this divergence between peer countries has gotten *larger* since the 1990s, and especially during the COVID-19 pandemic.

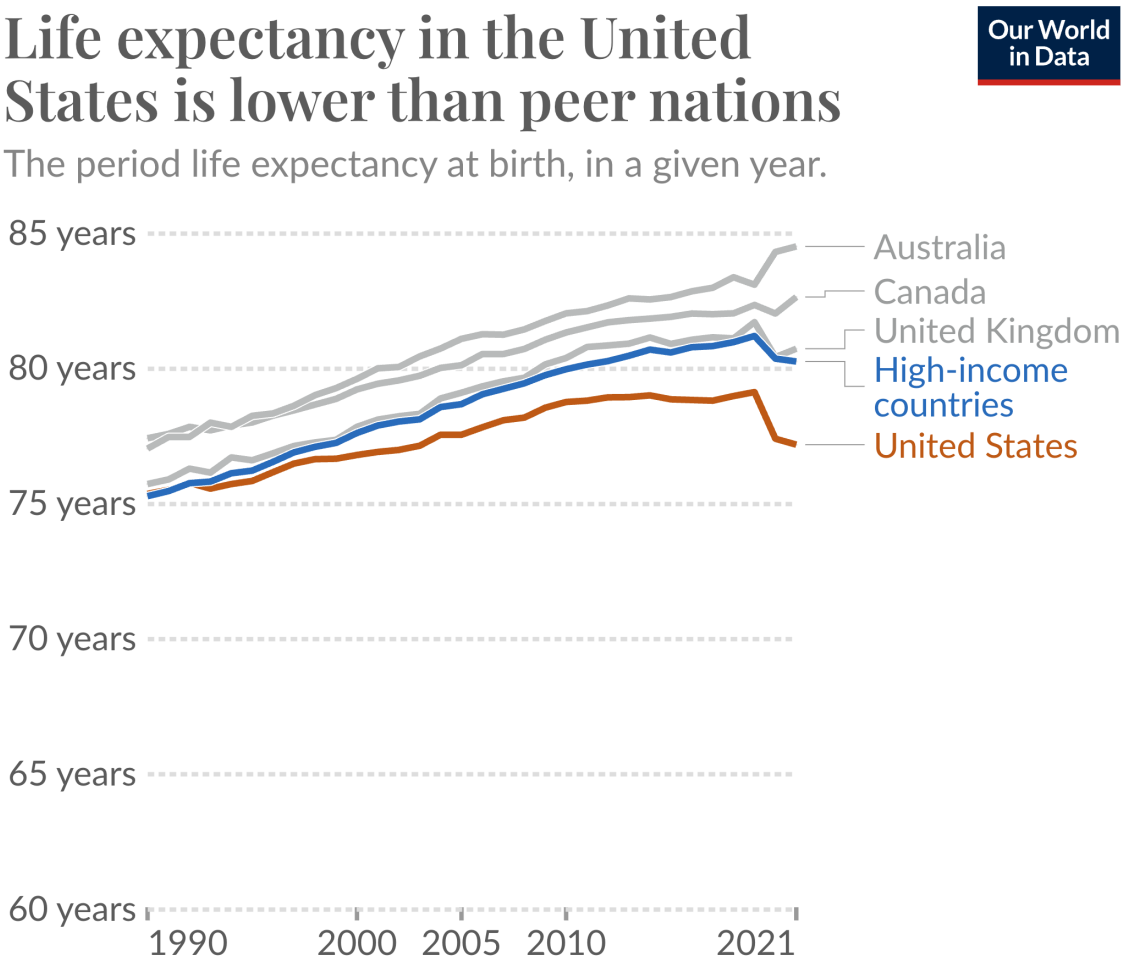
Critically—with the exception of the US—all of the countries in the peer group have some form of universal health care. In other words, all developed countries which are healthier than the US have policies which guarantee the uninsured rate is zero.

This then leads to a critical question: did universal health insurance *cause* these countries to have better health outcomes? Or is this a correlation between policy and outcomes? Certainly, one could make an argument that having health insurance does cause better health outcomes. On the other hand, there are lots of differences between the US and other high-income countries which may also explain the pattern seen in Figure 1.1. For example, maybe these countries have different regulations regarding the food quality, which may explain part of the difference in life expectancy. Or maybe because relatively more people commute to work via bicycle or walking, which may have health benefits compared to driving to work.

These other differences between groups which may *confound* our comparison between countries with universal health insurance and those without also extends to individuals. In the US at least, those with health insurance may be more likely to have a high-paying job which includes insurance as a benefit. Therefore, a comparison between those who have insurance and those who do not will be plagued with the fact that those who have insurance *also are more likely*

[1] Source: KFF

Figure 1.1: Life expectancy in the US vs. peer countries.



Source: Our World in Data

to have a high income job, who may have had better health outcomes regardless of if they had insurance or not. In other words, we may be confusing *correlation* for **causation*.

> **i** Correlation vs. Causation
>
> The **correlation** between two variables describes how they move together. It means two things are *associated*. **Causation** describes a direct relationship where a change in one variable causes a change in another. It means one thing *makes the other happen.*

A key part of the content in this course is to think about when our comparisons are describing a **correlative** relationship or a **causal** relationship. We will study techniques which—if one is convinced that the requisite assumptions hold—will actually be able to recover the *causal effect* of social phenomena, and bring important insights into essential public policy questions like: what is the effect of health insurance on health outcomes? Does going to college cause you to have higher wages? Do immigrants reduce native wages? and does policing cause decreases in crime?

1.3 Broad Steps of Econometric Analysis

Let's say we want to know what the effect of on-the-job training has on your wage. What steps would we need to take to answer this question?

1.3.1 Step 1: Write the Model

Our first step would be to write an *econometric model,* with the goal of using data and econometric tools to *estimate the parameters of the model.* This model can sometimes be derived from a formal model in economics (i.e., the profit maximization of firms), but it is at least based on some intuitive logic which mathematically relates some variable (the **dependent variable**) to some **independent variable**(s).

An example of the type of econometric model you may want to run to study the effect of on-the-job training is:

$$wage = \beta_0 + \beta_1(training) + \mu, \tag{1.1}$$

where *wage* is the *outcome, left-hand side (LHS), predicted, or dependent variable* (these are synonyms for the same thing), and *training* is the *explanatory, right-hand side (RHS), predictor, or independent variable* (again, synonyms for the same thing). The β_0 and β_1 variables are the parameters that we want to *estimate the value of* using econometric methods, and they tell us *how* our *independent* variable is related mathematically to our dependent variable.

1.3.2 Step 2: Find and Explore the Data

The next thing you have to do—after we figure out the model we want to estimate, such as in Equation 1.1—is to get the data. There is a *huge* range of datasets out there, and we are especially lucky that the data-gathering infrastructure in the United States is incredibly robust. As we go through the semester, we will work with as many different datasets that we can, which cover a range of topics: from sports to health, crime, and economic development. While different datasets usually can their own quirks, broadly speaking there are three different types of data we will work with in this class:

1. **Cross-sectional data:** This is data which contains information on units (e.g., individuals, countries, firms, households, etc.) at a *single point in time.* This data is the most common, and also is generally the simplest to work with, so we will start out by working mostly with cross-sectional data.
2. **Time-series data:** This data consists of observations on a single unit (e.g., a country's Gross Domestic Product (GDP), a specific company's stock price, or the national unemployment rate) across multiple time periods (e.g., years, quarters, months, or even minutes). In time series data, a crucial characteristic is the temporal ordering of the observations. This structure is often used for forecasting and modeling how a variable's past values influence its present and future values.
3. **Panel (or Longitudinal) data**: Panel data combines both cross-sectional and time-series dimensions. It consists of observations on multiple units (e.g., individuals, firms, or countries) over multiple time periods. It essentially tracks the same set of cross-sectional units over time. This gives it more information, more variability, and allows researchers to use a rich set of methods to really hone in on the causal effects of whatever social phenomena being studied.

In all of these cases, econometric data is organized in what is called a **matrix** or a **dataframe**. It should look roughly similar to an Excel Spreadsheet, where each column represents a different variable (which is usually labelled, sometimes not), and each row represents the values that a specific unit has across these variables.

For example, the data we use to estimate the model in Equation 1.1 may look like Table 1.1,

where the **person_name** column is (obviously) each person's name, **person_id** column corresponds the unique numerical identifier associated with each person (this will be more common to see in datasets since names are usually anonymous), the **wage** column shows each person's hourly wage in dollars, and the **yrs_training** column shows the number of years each person received on-the-job training. This row-column structure of organizing data is so fundamental that the word *column* is often interchanged with the word *variable*.

When you first look at some data, it is really important to figure out what each row represents. In the simple example shown in Table 1.1, this is straightforward since each row represents a different person. However, in more complex datasets, it may not be immediately obvious what

Table 1.1: Relationship between training and wages for 10 workers

person_name	person_id	wage	yrs_training
Paul	1	18.50	1.00
Ringo	2	22.00	3.00
George	3	17.65	2.50
John	4	27.95	8.20
Linda	5	19.20	0.50
Yoko	6	24.80	6.80
Pattie	7	20.50	5.50
Maureen	8	23.75	4.20
Cynthia	9	16.40	0.80
Barbara	10	28.50	9.10

each row represents. For example, in panel data, each row may represent a specific person at a specific point in time. So row 1 may be Paul in year 2020, row 2 may be Paul in year 2021, row 3 may be Ringo in year 2020, and so on. Figuring out what each row represents is *critical* to understanding the data, as well as how to appropriately analyze it.

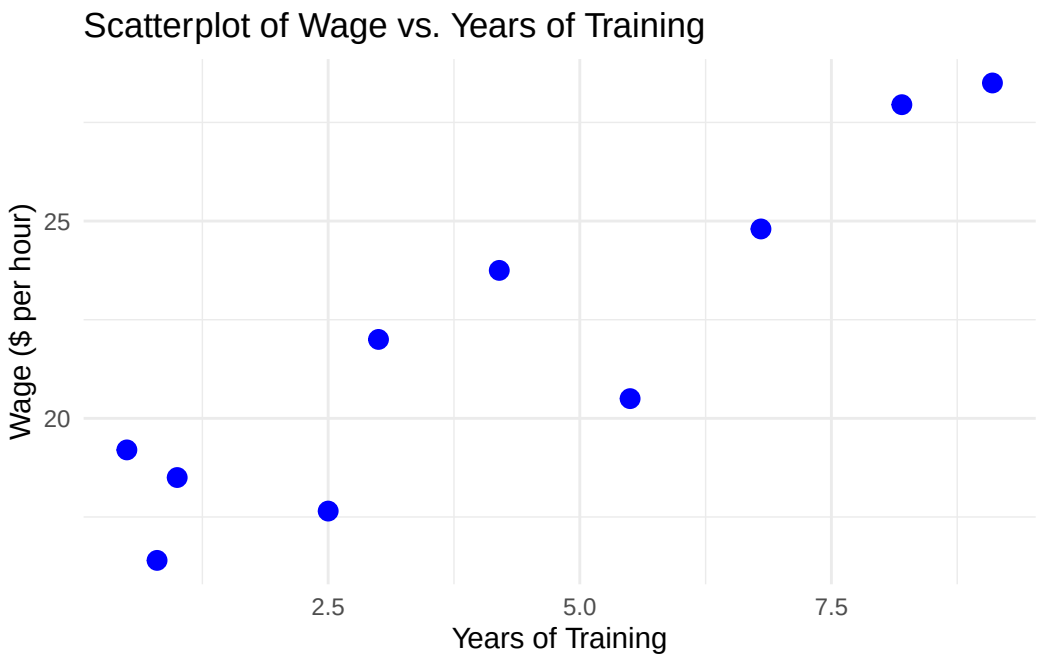
Once we have the data in hand, then we may need to do some data cleaning and organization. This may involve: - Removing missing values - Creating new variables - Changing the format of variables (e.g., from string to numeric) - Merging datasets together - Filtering the data to only include certain observations - And many other tasks.

These are all tasks that in many cases take up the bulk of the time in an econometric research project, and while not the most glamorous part of econometrics, it is absolutely essential to get right.

Before moving on to estimation, it is also important to do some **exploratory data analysis**, (which should be considered step 2.5 in the econometric research process). This involves looking at summary statistics of the data (means, medians, standard deviations, etc.), as well as visualizations (histograms, scatter plots, etc.). Exploratory data analysis is important for understanding the data, as well as for identifying potential issues with the data (e.g., outliers, missing values, etc.).

One of the most important parts of exploratory data analysis is to look at the relationship between the dependent variable and independent variable(s) in your model. One way this can be done is using a **scatterplot**, which plots each observation in the dataset using the independent and dependent variables as coordinates, with the independent variable on the x-axis and the dependent variable on the y-axis.

Figure 1.2: Relationship between training and wages for 10 workers



In Figure 1.2, we see a scatterplot of the relationship between years of training and wages for the 10 workers shown in Table 1.1. From this plot, we can see that there is a positive relationship between years of training and wages: as years of training increases, wages also tend to increase. While not good enough to draw any strong conclusions, one can see how this type of exploratory data analysis is useful for understanding the data and the relationship between variables.

1.3.3 Step 3: Estimate the Model

Once we have a model in mind as well as the appropriate data, we can then use our econometric methods to estimate the relationship. The bulk of this class is dedicated to this step in the econometric research process, but the other stages are no less important! The most common way that econometric models are estimated is with **Ordinary Least Squares** (OLS), which you may remember from your pre-req. In addition to estimating the actually value of the parameters in the model, we also want some idea of how *uncertain* we are about the estimated parameter value. This is where notions of statistical significance and confidence intervals come in.

1.3.4 Step 4: Interpret the Estimate

Once we have an estimated number for the parameter in our model, we then have to interpret the number. Here, the phrase "interpret the estimate" means two parallel things. First, there is the literal, mathematical interpretation of the model. This will change depending on the functional form we use to estimate the model, and we will spend lots of time learning about this. The second part of "interpreting" the model is about what the results mean in the context of your broader research question. Is your estimate big or small? Is it economically important? Do you actually think it reflects a causal effect? These are all types of interpretation which are a critical part of econometrics research, but for which there is not usually a formula we can use. A large part of the "art" working on an econometrics project is exactly this type of interpretation, and how you form it into a compelling narrative. While there are some tools we will cover which will help in this regard, in my opinion, nothing will help you more with this than just reading lots of good, well-written econometrics papers.

1.4 Statistical Software

Much of the econometric process is done using specialized statistical software. We will use this software to: "clean" our data and organize in the appropriate row-column format to do our econometric estimations; explore our data using descriptive statistics (means, medians, etc.) and visualizations (an increasingly important part of a compelling econometric project); estimate our model and conduct analyses of statistical uncertainty; and output our results to easy-to-interpret tables and figures.

There are a few commonly used econometrics software:

- **Stata:** Proprietary software purpose-built for econometrics
- **R:** Open-source programming language with a focus on statistical analysis
- **Python:** Open-source programming language with a focus on data science more broadly
- **EViews:** Proprietary software with a focus on forecasting and other time series applications

Each one of these software of their own pros and cons, and it is worth knowing several quite well. For the purposes of this class, we will learn the **R** programming language to do econometrics. If you haven't take the time to review Appendix A to learn more about R, how to install it, and to run your first chunk of code!

1.5 Summary and Conclusion

This chapter introduced the fundamental framework that will guide our study of econometrics throughout this course. We defined econometrics as the use of quantitative methods to un-

derstand social phenomena and inform public policy, with a particular focus on distinguishing between correlation and causation. Through the example of health insurance and health outcomes, we saw how patterns in data can be misleading. While the United States lags behind peer countries in life expectancy, and those peer countries all have universal health insurance, we cannot immediately conclude that universal coverage causes better health outcomes. The tools you will learn in this course are designed to help you identify causal relationships rather than mere associations.

1.5.1 Key Takeaways:

- **Econometrics vs. correlation:** Modern econometrics focuses on identifying causal effects, not just correlations between variables
- **Four steps of econometric research:**
    1. Write an econometric model
    2. Find and explore the appropriate data
    3. Estimate the model parameters
    4. Interpret the results
- **Three main data types:** Cross-sectional (single point in time), time-series (one unit over time), and panel data (multiple units over time)
- **Statistical software:** We will use **R** to clean data, estimate models, and present results

As we move forward, remember that econometrics is as much an art as it is a science. The mathematical tools we will learn are powerful, but they are only as good as the questions we ask and the care with which we interpret our results.

1.6 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

2 Good Data Practices

Key Questions

- Why do data errors matter for econometric analysis?
- What are descriptive statistics and why should we examine them?
- How can data visualization help us detect problems?
- What is replication and why is it fundamental to credible research?

Suggested Readings

- Bailey (2020), Ch. 2

Before we dive into sophisticated econometric methods, we need to establish good habits for working with data. Even the most advanced statistical techniques cannot rescue an analysis built on flawed or misunderstood data. This chapter focuses on the essential first steps: understanding our data through descriptive statistics and visualization, and documenting our work so that others (including our future selves) can verify and build upon it.

2.1 When Good Economists Use Bad Data

In 2010, economists Carmen Reinhart and Kenneth Rogoff published influential research on government debt and economic growth. Their analysis covered many annual observations from countries around the world, and their key finding was striking: when a country's government debt exceeded 90% of GDP, economic growth dropped dramatically.

The policy implications seemed clear. Governments should be cautious about using deficit spending to fight unemployment. This finding influenced policy debates across the world during a period when many countries were considering how aggressively to respond to the Great Recession. There was one problem with this analysis, however: the data didn't quite say what Reinhart and Rogoff thought it did.

In 2014, a graduate student named Thomas Herndon was trying to replicate their results for a homework assignment. He couldn't get the numbers to match. After obtaining the original Excel spreadsheet from Reinhart and Rogoff, Herndon and his advisors discovered three issues: some observations had been mistakenly excluded from key calculations, observations were

weighed strangely, and some cells in the Excel spreadsheet had simply not been included in the formula calculating the average.

When the data were corrected, the dramatic "cliff" in economic growth disappeared. While growth did decline somewhat as debt increased, the relationship was much more gradual than originally reported. There was no "cliff" that was so central to the original analysis. The corrected analysis told a very different policy story.

This experience teaches us that good data practices are one of the most fundamental tenants of credible research. No amount of sophisticated econometric technique can compensate for errors and misunderstandings in the underlying data. This chapter introduces practices that help us avoid such problems, and will be useful throughout your career as an econometrician.

2.2 Know Your Data

The first rule of data analysis is simple: **know your data**. This means examining it carefully before conducting any formal analysis. We want to understand what we're working with, detect potential errors, and develop intuition about the relationships we might find.

2.2.1 Descriptive Statistics

For every variable in our dataset, we should know several key statistics: the number of observations, the mean, the standard deviation, and the minimum and maximum values (refer to Appendix B for a review of what exactly these calculations mean). These statistics give us a feel for the data and help us spot problems.

Let's look at a simple example. Suppose we're studying the relationship between education and wages. Here's simulated data for 100 individuals:

```
# Set seed for reproducibility
set.seed(42)

# Generate simulated wage and education data
n <- 100
education <- round(rnorm(n, mean = 14, sd = 2.5))
education <- pmax(8, pmin(education, 20))  # Constrain between 8 and 20 years

# Generate wages with some relationship to education
wages <- 10 + 2.5 * education + rnorm(n, mean = 0, sd = 5)
wages <- pmax(wages, 8)  # Minimum wage floor

# Create age variable
```

	Unique	Missing Pct.	Mean	SD	Min	Median	Max
education	12	0	\num{14.1}	\num{2.6}	\num{8.0}	\num{14.0}	\num{20.0}
wage	100	0	\num{44.9}	\num{7.9}	\num{23.2}	\num{45.6}	\num{60.8}
age	33	0	\num{35.3}	\num{9.4}	\num{22.0}	\num{35.0}	\num{60.0}

	Unique	Missing Pct.	Mean	SD	Min	Median	Max
education	13	0	\num{13.9}	\num{3.2}	\num{-5.0}	\num{14.0}	\num{20.0}
wage	100	0	\num{52.8}	\num{80.9}	\num{23.2}	\num{45.6}	\num{850.0}
age	34	0	\num{37.0}	\num{19.0}	\num{22.0}	\num{35.0}	\num{200.0}

```
age <- round(rnorm(n, mean = 35, sd = 10))
age <- pmax(age, 22)  # Minimum age

# Create a tibble
wage_data <- tibble(
  id = 1:n,
  education = education,
  wage = wages,
  age = age
)

# Display descriptive statistics
datasummary_skim(wage_data |> select(education, wage, age),
                 title = "Descriptive Statistics: Education and Wages",
                 histogram = FALSE)
```

These statistics immediately tell us several things. We have 100 observations with no missing data (N = 100 for all variables). Education ranges from 8 to 20 years, with a mean around 14 years—this seems reasonable for a sample of working adults. Wages range from about $8 to $63 per hour, with substantial variation (standard deviation of about $10).

But what if our summary statistics looked like this instead?

Now we have a problem! The maximum wage of $850 per hour seems suspicious—is this real or a data entry error? The minimum education value of -5 years is impossible. And an age of 200 is clearly wrong. These obvious errors suggest we need to investigate our data more carefully.

2.2.2 Computing Descriptive Statistics in R

While summary tables like those above are useful for reports, you often need to calculate specific statistics yourself. R provides simple functions for all the common descriptive statistics.

2.2.2.1 Basic Summary Functions

The most commonly used functions for numeric variables are:

```
# Mean (average)
mean(wage_data$wage)
```

```
[1] 44.91258
```

```
# Standard deviation
sd(wage_data$wage)
```

```
[1] 7.9215
```

```
# Median (middle value)
median(wage_data$wage)
```

```
[1] 45.60513
```

```
# Minimum and maximum
min(wage_data$wage)
```

```
[1] 23.18942
```

```
max(wage_data$wage)
```

```
[1] 60.83585
```

You can also get several statistics at once using `summary()`:

```
summary(wage_data$wage)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  23.19   40.26   45.61   44.91   50.36   60.84
```

2.2.2.2 Handling Missing Values

Real-world data often contains missing values, represented as `NA` in R. Most summary functions will return `NA` if the data contains any missing values:

```
# Create a vector with a missing value
wages_with_na <- c(15, 22, NA, 18, 25)
mean(wages_with_na)
```

```
[1] NA
```

To calculate statistics while ignoring missing values, use the `na.rm = TRUE` argument:

```
mean(wages_with_na, na.rm = TRUE)
```

```
[1] 20
```

```
sd(wages_with_na, na.rm = TRUE)
```

```
[1] 4.396969
```

> ⚠ Always Check for Missing Data
>
> Before calculating any statistics, check how many missing values you have. The `sum(is.na())` function counts missing values:
>
> ```
> sum(is.na(wages_with_na))
> ```
>
> ```
> [1] 1
> ```
>
> If a large proportion of observations are missing, your statistics may not be representative of the full sample.

2.2.2.3 Counting and Frequency Tables

For categorical variables, we want to count how many observations fall into each category. The `count()` function from the `tidyverse` makes this easy. We are also going to be using the pipe `|>` in order to string our data operations together like a sentence. For more info on the `tidyverse` and the pipe operator, refer to Appendix A.

First, let's create some categorical variables to work with:

```
# Create categorical variables for demonstration
wage_data <- wage_data |>
  mutate(
    # College degree: 1 if 16+ years of education
    college = ifelse(education >= 16, 1, 0),
    # Age groups
    age_group = case_when(
      age < 30 ~ "Under 30",
      age < 45 ~ "30-44",
      TRUE ~ "45+"
    )
  )
```

Now we can count observations in each category:

```
# Count observations by college status
wage_data |>
  count(college)
```

```
# A tibble: 2 x 2
  college     n
    <dbl> <int>
1       0    67
2       1    33
```

You can count by multiple variables to create cross-tabulations:

```
# Count by two variables
wage_data |>
  count(college, age_group)
```

```
# A tibble: 6 x 3
  college age_group     n
    <dbl> <chr>     <int>
1       0 30-44        39
2       0 45+          14
3       0 Under 30     14
4       1 30-44        12
5       1 45+           5
6       1 Under 30     16
```

To add percentages, you can extend the `count()` output:

```
wage_data |>
  count(college) |>
  mutate(percent = 100 * n / sum(n))
```

```
# A tibble: 2 x 3
  college     n percent
    <dbl> <int>   <dbl>
1       0    67      67
2       1    33      33
```

2.2.2.4 Group-wise Statistics

Often we want to compare statistics across groups. The `group_by()` and `summarize()` functions work together for this:

```
# Calculate mean wage by college status
wage_data |>
  group_by(college) |>
  summarize(
    mean_wage = mean(wage),
    sd_wage = sd(wage),
    n = n()
  )
```

```
# A tibble: 2 x 4
  college mean_wage sd_wage     n
    <dbl>     <dbl>   <dbl> <int>
1       0      41.7    6.92    67
2       1      51.4    5.57    33
```

This is incredibly useful for exploring how outcomes differ across groups—exactly what we'll be doing in regression analysis.

```
# Compare wages across age groups
wage_data |>
  group_by(age_group) |>
  summarize(
    mean_wage = mean(wage),
    median_wage = median(wage),
    min_wage = min(wage),
    max_wage = max(wage),
    count = n()
  )
```

```
# A tibble: 3 x 6
  age_group mean_wage median_wage min_wage max_wage count
  <chr>         <dbl>       <dbl>    <dbl>    <dbl> <int>
1 30-44          42.9        43.0     23.2     57.5    51
2 45+            45.4        42.7     37.5     60.5    19
3 Under 30       48.1        47.6     27.3     60.8    30
```

Quick Reference: Common Summary Functions

Table 2.1: Common functions for descriptive statistics

Function	What it calculates
`mean()`	Average
`sd()`	Standard deviation
`median()`	Middle value
`min()`, `max()`	Minimum, maximum
`sum()`	Total
`n()`	Count (inside summarize)
`count()`	Frequency table
`summary()`	Multiple statistics at once

2.2.3 Examining Distributions

When a variable takes on a limited number of values, it's helpful to examine the frequency distribution. Recall that we created a variable indicating whether someone has a college degree (1 = yes, 0 = no). Let's look at its distribution:

```
# Create frequency table
wage_data |>
  count(college) |>
  mutate(percent = n / sum(n) * 100) |>
  knitr::kable(
    col.names = c("College Degree", "Count", "Percent"),
    digits = 1,
    caption = "Distribution of College Degree"
  )
```

Table 2.2: Distribution of College Degree

College Degree	Count	Percent
0	67	67
1	33	33

This tells us that 29% of our sample has a college degree. But imagine our frequency table looked like this:

Table 2.3: A Problematic Distribution

College Degree	Count	Percent
0	30	58.8
1	20	39.2
100	1	2.0

We have a value of 100 for the college degree variable. Either we have someone with 100 college degrees (unlikely), or—much more probably—we have a coding error. This is the kind of problem that descriptive statistics help us catch.

2.3 Visualizing Data

Numbers are useful, but graphs often reveal patterns and problems that summary statistics miss. Effective data visualization is both a diagnostic tool and a way to understand relation-

ships in our data. In econometrics, we rely heavily on visualizations to explore data, diagnose problems, and communicate results.

> **!** Why Visualization Matters
>
> A good data visualization translates complex data into a clear visual format, revealing patterns, trends, and relationships that would be difficult to see in tables of numbers. The goal is to tell an accurate and compelling story that allows your audience to quickly grasp key insights.

2.3.1 The Grammar of Graphics: ggplot2

We'll use `ggplot2` (part of the tidyverse) for creating visualizations. The "gg" stands for "grammar of graphics"—a systematic way of thinking about how to build plots.

Every ggplot has two main components:

1. **The scaffolding**: titles, axis labels, legends, sources
2. **The content**: the actual data encoded as visual elements (points, lines, bars, etc.)

The key insight of ggplot2 is that we build plots **layer by layer**, adding elements one at a time until we have a complete visualization.

2.3.2 Getting Started: The Palmer Penguins

Let's use real data to learn ggplot2. The Palmer Penguins dataset contains measurements of 344 penguins from three species collected near Palmer Station, Antarctica.

```
# Load the penguins data
library(palmerpenguins)

# Take a look at the data
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
```

```
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

The data includes species, island location, bill measurements, flipper length, body mass, sex, and year of observation.

2.3.3 Building a Plot: Step by Step

Let's answer a research question: **Do penguins with longer flippers weigh more?**

We start by creating an empty plot object:

```
ggplot(data = penguins)
```

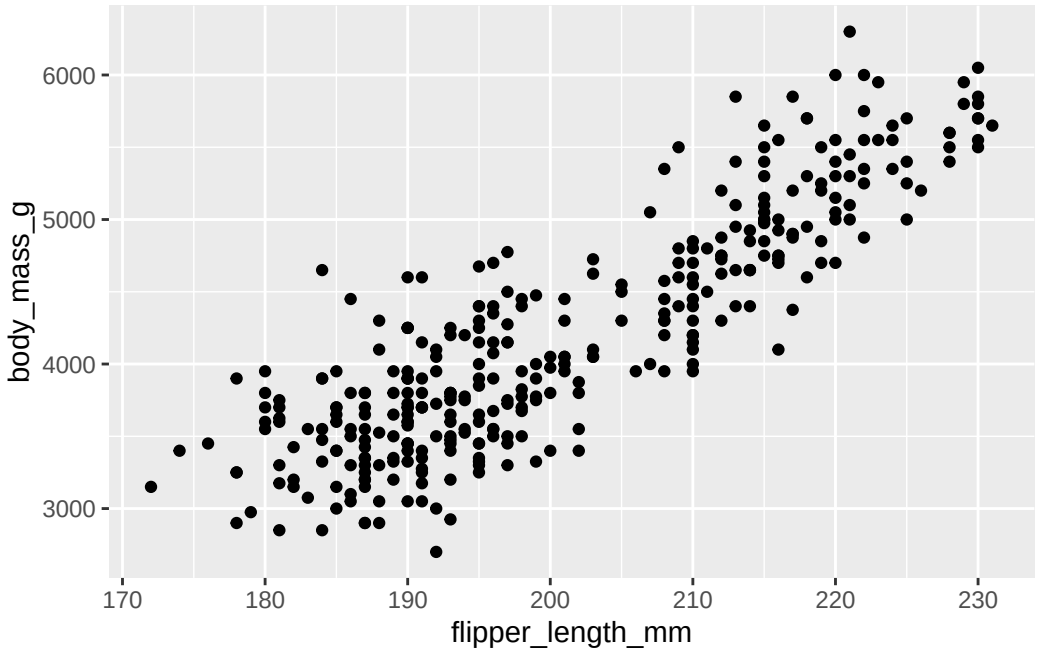
Figure 2.1: An empty ggplot object with no layers.

This creates the plotting area but doesn't show any data yet.

Now let's add a layer with points. We need to specify: - The **geometry** (what shape): `geom_point()` for a scatterplot - The **aesthetic mapping** (which variables go where): `aes(x = ..., y = ...)`

```
ggplot(data = penguins) +
  geom_point(aes(x = flipper_length_mm, y = body_mass_g))
```

Figure 2.2: A basic scatterplot showing flipper length versus body mass.



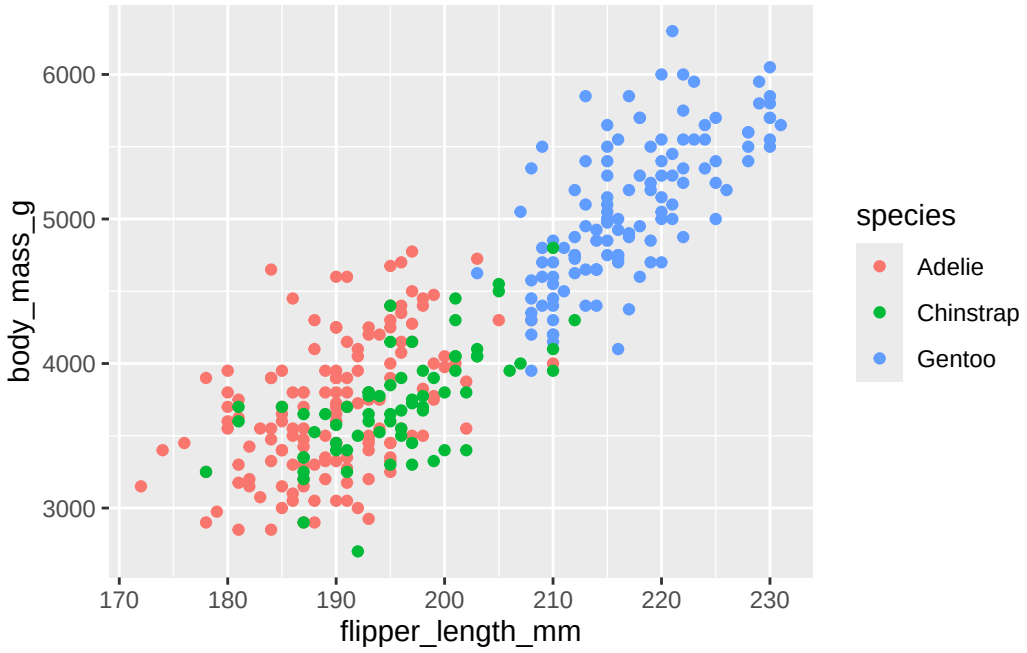
The plot clearly shows a positive relationship: penguins with longer flippers tend to be heavier.

2.3.4 Adding More Information: Color and Shape

But wait—could this relationship be driven by different penguin species? Maybe larger species have both longer flippers and greater body mass. We can *encode* species information using color:

```
ggplot(data = penguins) +
  geom_point(aes(x = flipper_length_mm, y = body_mass_g, color = species))
```

Figure 2.3: Scatterplot with species indicated by color. The positive relationship holds within each species.



Now we can see that:

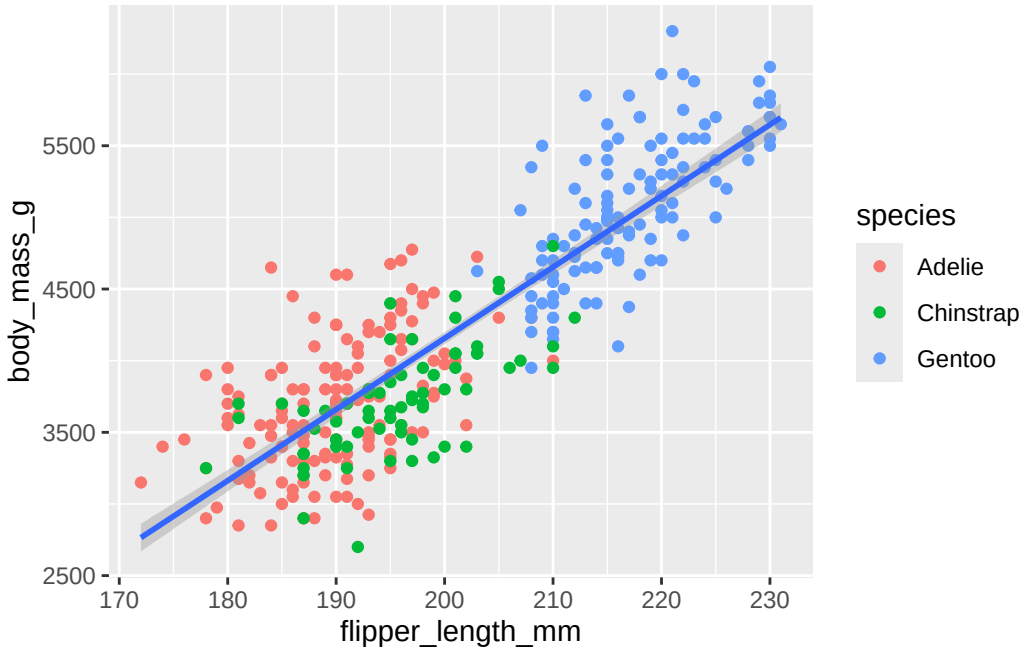
- Gentoo penguins (blue) are generally larger
- The positive relationship between flipper length and body mass exists **within each species**
- This suggests the relationship isn't just due to species differences

2.3.5 Adding a Line-of-Best Fit

Let's add a line-of-best fit (i.e., a regression line) to visualize the overall trend. This requires a new **layer** (not a new aesthetic):

```
ggplot(data = penguins) +
  geom_point(aes(x = flipper_length_mm, y = body_mass_g, color = species)) +
  geom_smooth(aes(x = flipper_length_mm, y = body_mass_g), method = "lm")
```

Figure 2.4: Scatterplot with regression line. The blue line shows the estimated linear relationship.



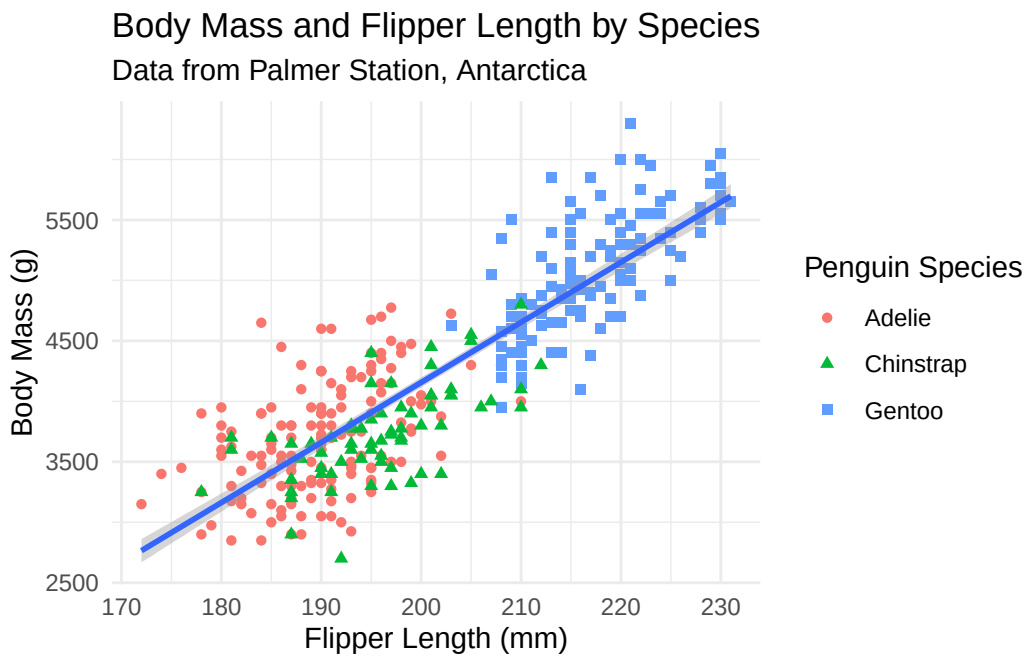
The `geom_smooth()` layer adds a line of best fit using linear regression (`method = "lm"`).

2.3.6 Proper Labeling

Our plot conveys information well, but the axis labels and title need improvement. Good labels are essential for clear communication:

```
ggplot(data = penguins) +
  geom_point(aes(x = flipper_length_mm, y = body_mass_g, color = species, shape = species)) +
  geom_smooth(aes(x = flipper_length_mm, y = body_mass_g), method = "lm") +
  labs(
    title = "Body Mass and Flipper Length by Species",
    subtitle = "Data from Palmer Station, Antarctica",
    x = "Flipper Length (mm)",
    y = "Body Mass (g)",
    color = "Penguin Species",
    shape = "Penguin Species"
  ) +
  theme_minimal()
```

Figure 2.5: A properly labeled scatterplot with clear titles and axis labels.



Notice how much clearer this is! The plot now includes:

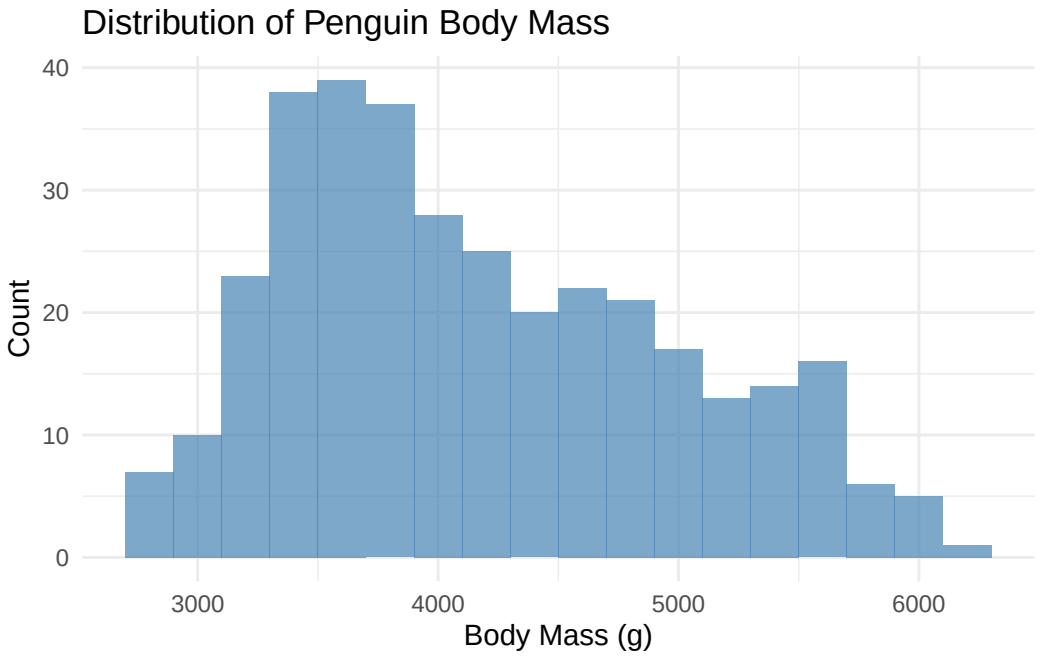
- A descriptive title
- Proper axis labels with units
- A data source in the subtitle
- Readable legend labels

2.3.7 Histograms for Distributions

Histograms show the distribution of a continuous variable by dividing it into bins:

```
ggplot(penguins) +
  geom_histogram(aes(x = body_mass_g), binwidth = 200, fill = "steelblue", alpha = 0.7) +
  labs(
    title = "Distribution of Penguin Body Mass",
    x = "Body Mass (g)",
    y = "Count"
  ) +
  theme_minimal()
```

Figure 2.6: Histogram showing the distribution of penguin body mass.



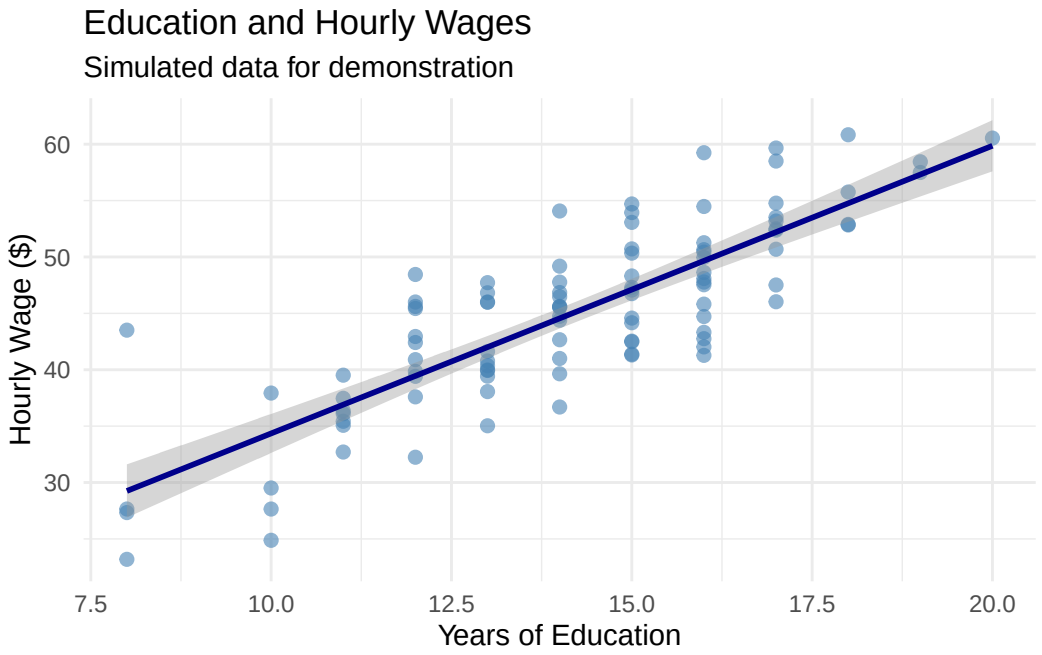
The `binwidth` parameter controls how wide each bar is. Smaller binwidths create more bars and show more detail.

2.3.8 A Simple Example with Our Wage Data

Let's apply these principles to our wage and education data:

```
ggplot(wage_data, aes(x = education, y = wage)) +
  geom_point(alpha = 0.6, size = 2, color = "steelblue") +
  geom_smooth(method = "lm", se = TRUE, color = "darkblue") +
  labs(
    title = "Education and Hourly Wages",
    subtitle = "Simulated data for demonstration",
    x = "Years of Education",
    y = "Hourly Wage ($)"
  ) +
  theme_minimal()
```

Figure 2.7: The relationship between education and wages with proper labeling and styling.



Building Intuition

The layered approach of ggplot2 mirrors how we think about data analysis: - Start with the data - Add visual representations - Refine with labels and formatting Experiment with different geometries, aesthetics, and layers to find the visualization that best tells your data's story.

2.4 Replication: The Foundation of Credible Research

Science depends on replication. If other researchers cannot reproduce our results using the information we provide, then our findings cannot be verified or built upon. This is not just an abstract principle—it's a practical necessity.

The Replication Standard

Research meets the **replication standard** when an independent researcher can exactly duplicate the results using only the information provided.

2.4.1 Why Replication Matters

Replication serves several crucial purposes. First, it allows others to check our work. The Reinhart-Rogoff example shows why this matters: only because they shared their data could Thomas Herndon discover the errors. Without replication materials, the mistakes might never have been found.

Second, replication enables others to probe our analysis. Statistical results often hinge on seemingly small decisions—which observations to include, how to handle missing data, which variables to control for. Skeptical readers (which should be everyone!) want to see whether reasonable alternative approaches produce similar conclusions. If a certain choice substantially changes our results, we need to pay attention.

Third, committing to a replication standard keeps our work honest. If we know others will check our work, we're less likely to make choices based on getting the "right" answer rather than statistical merit.

Finally, replication helps our future selves. Even a moderately complex project can involve hundreds of decisions over weeks or months. What seemed obvious at the time can become completely opaque later. Good documentation means we can return to a project after a break and immediately understand what we did.

2.4.2 What Makes Good Replication Materials?

Effective replication files have two components: a **codebook** that documents the data, and **analysis code** that shows exactly how results were produced.

The codebook describes where each variable comes from and any transformations applied. For example:

- **Variable name**: `wage`
- **Description**: Hourly wage in dollars, calculated as annual salary divided by hours worked in past year
- **Source**: National Longitudinal Survey of Youth, 1996 wave
- **Notes**: Wages above $200/hour were excluded as likely data errors

The analysis code shows the exact steps used to produce each result. This means providing the actual R code, not just descriptions of what we did. And crucially, the code should include comments explaining *why* we made certain choices:

```
# Estimate relationship between education and wages
# We control for age because it affects both education and wages
# (older workers have had more time to complete education and gain experience)
```

```
model <- lm(wage ~ education + age, data = wage_data)
summary(model)
```

2.4.3 Good Coding Practices

As we write code for our analysis, several practices make our work easier to understand and verify:

Use descriptive variable names. Compare `x1` versus `years_education`—the second is much clearer.

Comment extensively. Explain what each section of code does and why you made certain choices. Your future self will thank you.

Organize your code logically. Start with data loading and cleaning, then move to analysis, then to creating tables and figures.

Make your code reproducible. Use `set.seed()` before any random operations so results don't change each time you run the code.

Here's an example of well-documented code:

```
# ============================================================================
# Analysis of Education and Wages
# Author: Your Name
# Date: December 2024
# ============================================================================

# Load required packages
library(tidyverse)

# Set seed for reproducibility
set.seed(42)

# ----------------------------------------------------------------------------
# 1. LOAD AND CLEAN DATA
# ----------------------------------------------------------------------------

# Load data from National Longitudinal Survey
# Downloaded from: https://www.nlsinfo.org/content/cohorts/nlsy79
# Download date: December 1, 2024
wage_data <- read_csv("nlsy_wages.csv")

# Remove observations with missing wage data
```

```r
# This affects 23 observations (2.3% of sample)
wage_data <- wage_data |>
  filter(!is.na(wage))

# Create college degree indicator
# Defined as 16+ years of education (bachelor's degree)
wage_data <- wage_data |>
  mutate(college = ifelse(education >= 16, 1, 0))

# ------------------------------------------------------------------------------
# 2. DESCRIPTIVE STATISTICS
# ------------------------------------------------------------------------------

# Summary statistics for main variables
summary(wage_data |> select(wage, education, age))

# ------------------------------------------------------------------------------
# 3. MAIN ANALYSIS
# ------------------------------------------------------------------------------

# Estimate basic relationship between education and wages
# Simple regression without controls
model1 <- lm(wage ~ education, data = wage_data)

# Add age as control variable
# Age affects both education completion and wage levels
model2 <- lm(wage ~ education + age, data = wage_data)
```

2.5 A Quick Example: State Crime Data

Let's practice these principles with a small example. We'll look at violent crime rates across U.S. states and potential correlates.

```r
# Create simulated state crime data for demonstration
set.seed(123)
n_states <- 50

state_data <- tibble(
  state = state.abb,
  violent_crime = rnorm(n_states, mean = 400, sd = 150),
```

	Unique	Missing Pct.	Mean	SD	Min	Median	Ma
violent_crime	50	0	\num{405.2}	\num{138.9}	\num{105.0}	\num{389.1}	\n
percent_urban	49	0	\num{72.1}	\num{13.4}	\num{35.4}	\num{72.3}	\n
percent_poverty	50	0	\num{13.2}	\num{3.0}	\num{8.0}	\num{13.1}	\n

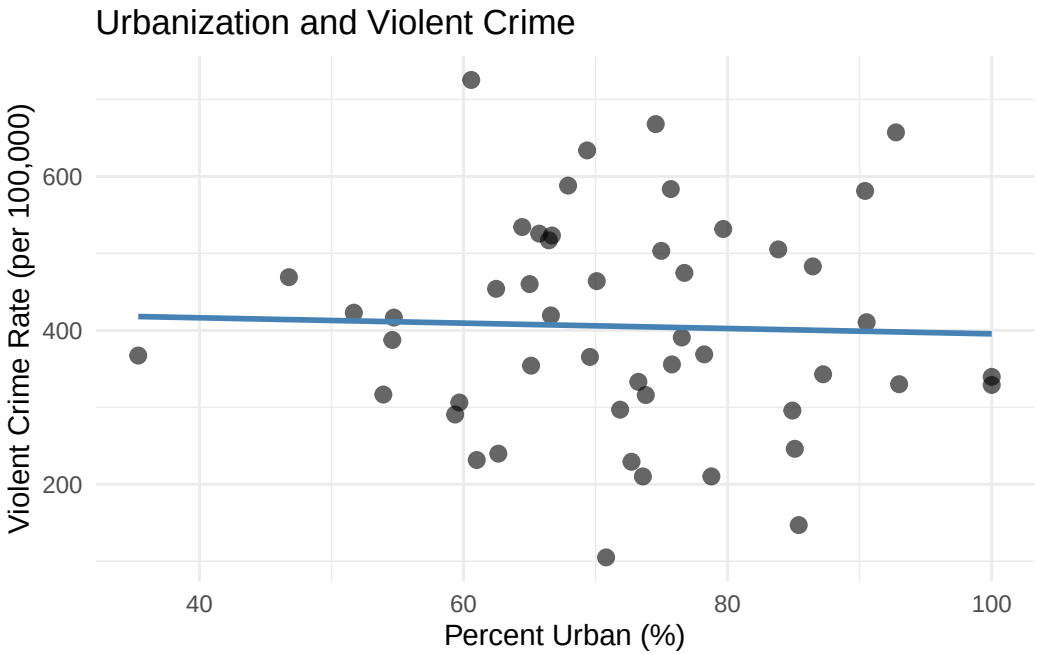
```
  percent_urban = rnorm(n_states, mean = 70, sd = 15),
  percent_poverty = rnorm(n_states, mean = 14, sd = 3)
) |>
  mutate(
    violent_crime = pmax(violent_crime, 100),  # Keep positive
    percent_urban = pmax(20, pmin(percent_urban, 100)),  # Constrain to 0-100
    percent_poverty = pmax(8, pmin(percent_poverty, 22))  # Reasonable range
  )

# Display descriptive statistics
datasummary_skim(state_data |> select(-state),
                 title = "State Crime Data: Descriptive Statistics",
                 histogram = FALSE)
```

The descriptive statistics look reasonable. Crime rates range from about 150 to 750 per 100,000 people. The urbanization variable is measured on a 0-100 scale, while poverty is measured as a percentage (8% to 22%). Understanding these scales is important for interpreting results.

Now let's visualize the relationship between urbanization and crime:

```
ggplot(state_data, aes(x = percent_urban, y = violent_crime)) +
  geom_point(alpha = 0.6, size = 2.5) +
  geom_smooth(method = "lm", se = FALSE, color = "steelblue") +
  labs(
    x = "Percent Urban (%)",
    y = "Violent Crime Rate (per 100,000)",
    title = "Urbanization and Violent Crime"
  ) +
  theme_minimal()
```

Figure 2.8: Relationship between urbanization and violent crime rates across U.S. states.



The scatterplot suggests a positive relationship: more urbanized states tend to have higher violent crime rates. But this is just correlation, not causation. Many factors differ between urban and rural states beyond just population density, and any of these could be driving the relationship we observe. This is precisely the kind of endogeneity problem that motivates the econometric methods we'll study in later chapters.

2.6 Summary

Good data practices are the foundation of credible econometric analysis. Before conducting any formal statistical tests, we must understand our data through descriptive statistics and visualization. These diagnostic tools help us catch errors, understand variable scales and distributions, and develop intuition about relationships in the data.

Equally important is documenting our work so that others can replicate our results. A replication file should include a codebook describing data sources and transformations, plus the complete analysis code with clear comments explaining our choices. This commitment to transparency keeps our work honest, enables others to verify and build on our findings, and helps our future selves understand what we did months or years later.

The Reinhart-Rogoff case reminds us that even accomplished researchers make data mistakes. The best defense is establishing good habits: examine your data carefully, document everything,

and write code that your future self (and others) can understand.

2.7 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

3 Causality and Randomized Control Trials

Key Questions

- Why can't we interpret correlations as causal effects?
- What is selection bias and how does it arise?
- How do randomized control trials eliminate selection bias?
- What is the potential outcomes framework and how does it formalize causal inference?

Suggested Readings

- Angrist and Pischke (2014), Ch. 1

In Chapter 1, we discussed how econometrics focuses on identifying **causal effects** rather than mere correlations. We used the example of health insurance: while people with health insurance tend to be healthier than people without insurance, we cannot immediately conclude that health insurance *causes* better health. This chapter develops the conceptual and mathematical tools we need to think rigorously about causality, and introduces the randomized control trial (RCT) as the gold standard for causal inference.

3.1 The Problem with Simple Comparisons

Let's return to our health insurance example. People with health insurance are, on average, healthier than people without it. This correlation is undeniable. But can we interpret this as evidence that insurance *causes* better health?

The problem is that people who have health insurance differ from those who don't in many ways beyond just their insurance status. Those with insurance tend to be wealthier, more educated, and have more stable employment. All of these factors independently affect health outcomes. When we compare the health of insured and uninsured individuals, we're not just comparing the effect of insurance—we're comparing people who differ along many dimensions.

Selection Bias

Selection bias occurs when we compare groups that differ systematically in ways beyond just the treatment we're studying. In other words, we are comparing **apples to oranges**.

The comparison we *want* to make is between two otherwise identical people who differ only in whether they have health insurance. In other words, we want an **apples to apples** comparison. This is the essence of the *ceteris paribus* condition: **all else being equal**, the difference between treated and untreated units reveals the causal effect of treatment.

3.2 The Multiverse of Possibilities

To build intuition for causal inference, consider the television show *Rick and Morty*. In this show, the character Rick possesses a "portal gun" that allows him to travel across infinite parallel universes. In each universe, there exists a version of his grandson Morty who is identical in every way except for one characteristic—Cowboy Morty, Evil Morty, Hammer Morty, and so on. Each Morty has the same home, same parents, same sister, same *everything else*.

This multiverse provides the perfect setting for causal inference! Comparisons between different Mortys are truly *ceteris paribus* comparisons because everything except the one characteristic of interest is held constant.

3.2.1 The Causal Effect of College

Suppose we want to know the causal effect of attending college on wages. We can't simply compare the wages of college graduates to non-graduates because these groups likely differ in many unobserved ways (motivation, family background, innate ability, etc.).

But imagine we could observe two versions of Morty from parallel universes: Morty C-137 (who didn't attend college) and College Morty (who did). These two Mortys are identical in every way—same intelligence, same family, same opportunities—except that one went to college and the other didn't.

The difference in wages between College Morty and Morty C-137 would be a genuine *ceteris paribus* difference—the true causal effect of college on wages.

Figure 3.1: Rick and Morty: A show about infinite parallel universes.

Figure 3.2: Morty C-137 (left) and College Morty (right). The wage difference between them represents the true causal effect of college.

(a) Morty C-137

(b) College Morty

3.2.2 The Fundamental Problem

Of course, the fundamental problem of causal inference is that **we don't have a portal gun**. We only observe one version of reality, not parallel universes. We can never observe what would have happened to a college graduate had they *not* gone to college, or what would have happened to a non-graduate had they attended. This unobserved "what if" scenario is called the **counterfactual**.

3.3 Randomized Control Trials: The Next Best Thing

Since we can't travel between parallel universes, we need another way to make valid causal comparisons. The solution that comes closest to achieving *ceteris paribus* conditions in the real world is the **randomized control trial** (RCT).

Consider how a pharmaceutical company tests whether a new drug works. They don't just give the drug to sick people and see if they get better—that would be subject to all sorts of selection bias. Instead, they conduct an RCT:

1. **Sample** a group of volunteers from the population of interest
2. **Randomly assign** some volunteers to receive the drug (the "treatment" group) and others to receive a placebo (the "control" group)
3. **Compare** the average outcomes between the treatment and control groups

The key insight is that **random assignment eliminates selection bias**. When treatment is assigned randomly, there is no systematic relationship between who receives treatment and their underlying characteristics. On average, the treatment group and control group will be identical in terms of age, health status, income, motivation, and every other characteristic—both observed and unobserved.

> **!** Why Randomization Works
>
> Random assignment ensures that, on average, the treatment and control groups have identical characteristics. Any difference in outcomes between the groups can therefore be attributed to the treatment itself, not to pre-existing differences between the groups.

3.4 The Law of Large Numbers

Why exactly does randomization eliminate selection bias? The answer lies in a fundamental result from probability theory: the **Law of Large Numbers** (LLN).

The LLN states that as the sample size increases, the sample average converges to the population average. Formally, if $Y_1, Y_2, ..., Y_n$ are independent, identically distributed random variables with mean μ, then:

$$\text{plim}(\bar{Y}_n) = \mu$$

where $\bar{Y}_n$ is the sample average and plim denotes the probability limit (what the value converges to as $n \to \infty$).

3.4.1 Seeing the LLN in Action: Rolling Dice

To see the Law of Large Numbers at work, consider a simple example: rolling a fair six-sided die. The expected value (average outcome) of a single roll is:

$$\mu = \frac{1+2+3+4+5+6}{6} = 3.5$$

If we roll a die just once or twice, we might get values far from 3.5 (like a 1 or a 6). But what happens if we roll it many, many times? Let's see what happens!

First, we will "pretend" rolling a dice by using the `sample()` function. This function will randomly select `n` numbers from a vector, here the numbers 1-6.

Let's first roll the dice once. We will set the `size` argument in the `sample()` function to `size = 1` to generate a single random number from our vector.

```
# Set seed for reproducibility
set.seed(1248)

# Create a vector which represents the possible outcomes of a dice roll
dice <- c(1, 2, 3, 4, 5, 6)

# Now "roll" the dice using the sample() function

roll_1 <- sample(dice, size = 1, replace = TRUE)

# Print the result of the dice roll
roll_1
```

```
[1] 2
```

With 1 dice roll, we get a value of 2. Obviously pretty far from the expected value $\mu = 3.5$.

Now, let's roll it 5 times and take the average of our 5 different rolls using the `mean()` function:

```
roll_5 <- sample(dice, size = 5, replace = TRUE)

# Print the 5 different dice rolls
roll_5
```

```
[1] 1 5 2 5 1
```

```
# Find the mean of the 5 different dice rolls
mean(roll_5)
```

```
[1] 2.8
```

With just 5 rolls, our average is a little closer to the expected value. Now, let's try to roll it 25 times, 50 times, and 100 times and find the average values of our dice roll!

```
# Continue rolling to get 10 total rolls
rolls_25 <- sample(dice, size = 10, replace = TRUE)
mean(rolls_25)
```

```
[1] 2.3
```

```
# Keep going to 50 rolls
rolls_50 <- sample(dice, size = 50, replace = TRUE)
mean(rolls_50)
```

```
[1] 3.68
```

```
# And finally 100 rolls
rolls_100 <- sample(1:6, size = 100, replace = TRUE)
mean(rolls_100)
```

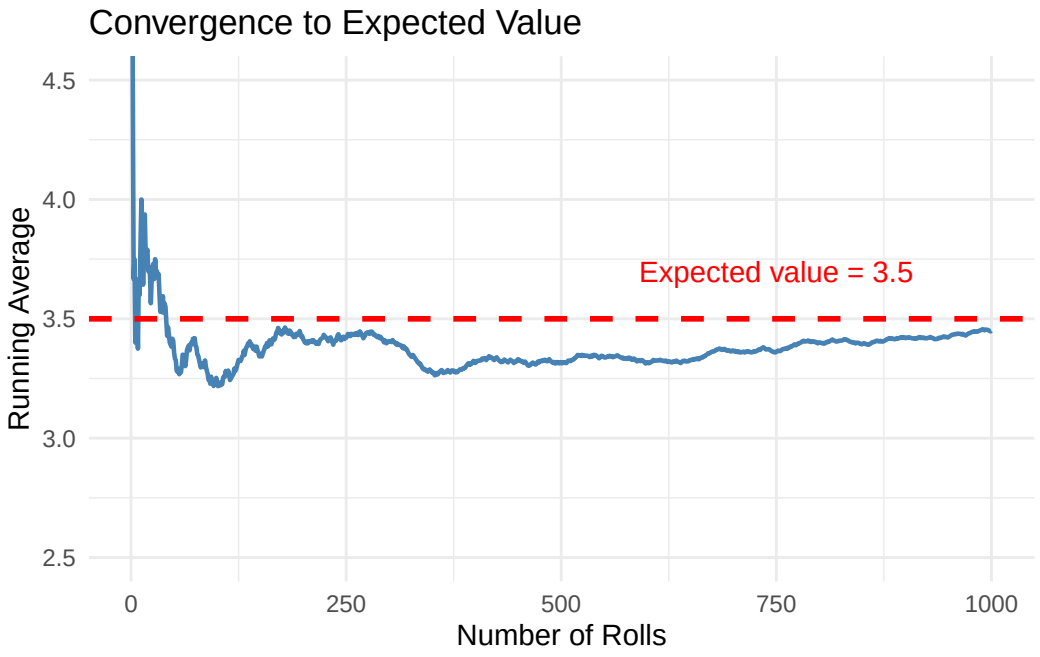
```
[1] 3.35
```

Notice how the average tends to get closer to 3.5 as we increase the number of rolls. This isn't guaranteed on any single attempt (randomness still plays a role), but the general tendency is clear.

Now let's see what happens when we roll the die many times and track the running average at each step. In Figure 3.3, I plot the *sample average* of our dice rolls on the y-axis, and the number of times I roll the dice on the x-axis. The population mean, 3.5, is plotted with a dashed red line.

```
# Set seed for reproducibility
set.seed(12345)

# Number of dice rolls to simulate
n_rolls <- 1000

# Simulate rolling a six-sided die n_rolls times
dice_rolls <- sample(1:6, size = n_rolls, replace = TRUE)

# Calculate the running average after each roll
running_avg <- cumsum(dice_rolls) / 1:n_rolls

# Create a data frame for plotting
dice_data <- tibble(
  roll_number = 1:n_rolls,
  running_average = running_avg
)

# Plot the running average
ggplot(dice_data, aes(x = roll_number, y = running_average)) +
  geom_line(color = "steelblue", linewidth = 0.8) +
  geom_hline(yintercept = 3.5, linetype = "dashed", color = "red", linewidth = 1) +
  annotate("text", x = 750, y = 3.7, label = "Expected value = 3.5",
           color = "red", size = 4) +
  labs(
    x = "Number of Rolls",
    y = "Running Average",
    title = "Convergence to Expected Value"
  ) +
  theme_minimal() +
  coord_cartesian(ylim = c(2.5, 4.5))
```

Figure 3.3: The Law of Large Numbers in action. As we roll a die more times, the average of all rolls converges to the expected value of 3.5.



Notice several key features about Figure 3.3:

- **Early volatility**: In the first few rolls, the running average bounces around wildly. After just 10 rolls, we might be at 4.2 or 2.9.
- **Gradual stabilization**: As we accumulate more rolls (say, 100 or 200), the average starts to settle down closer to 3.5.
- **Long-run convergence**: By the time we've rolled 1,000 times, the average is very close to the theoretical expected value of 3.5.

This is the Law of Large Numbers in action: **with enough observations, sample averages converge to population averages**. The key word is "enough"—we need a sufficiently large sample for the convergence to occur.

3.4.2 How the LLN Enables Causal Inference

When we randomly assign individuals to treatment and control groups, we are essentially drawing random samples from the same underlying population. The LLN guarantees that, with a large enough sample, the average characteristics of each group will converge to the population average—and therefore, to each other.

This is why randomization eliminates selection bias: the treatment and control groups become statistically identical in all characteristics—both observed and unobserved—as the sample size grows.

3.5 How Randomization Eliminates Selection Bias: A Simulation

What is a Simulation?

A **simulation** is a computer-based experiment where we generate artificial data according to rules we specify. Unlike real-world data, we know exactly how the simulated data was created—including the true causal effects, which variables influence which outcomes, and how treatment was assigned.
Simulations are powerful learning tools because they let us see what *should* happen under ideal conditions. If we set the true causal effect to zero and then compare treated and untreated groups, any difference we observe must be due to something other than a real treatment effect (like selection bias or random chance). By manipulating features of the simulation—such as sample size or how treatment is assigned—we can build intuition for how these factors affect our ability to recover causal effects. Think of simulations as a laboratory where we can run controlled experiments on statistical methods themselves.

Let's look at a more complex version of how the LLN works in the context of a randomized control trial. Consider a simulation where we have a population with three observable characteristics: light-colored eyes, dark hair, and being tall. Suppose these characteristics are correlated with both treatment status and outcomes in the observational data (i.e., without random assignment). Specifically, having light-colored eyes will be *strongly positively correlated* with the treatment and the outcome; having dark hair will be *moderately positively correlated* with the treatment and the outcome, and being tall will be *slightly negatively correlated* with the treatment and the outcome. Critically, we set the true causal effect of treatment on the outcome to **zero**.

A Note on the Code Below

The R code in the following sections is more advanced than what we've covered so far in the course. Don't worry if you don't understand every line! The code is provided for those who are curious and want to see how simulations work "under the hood." Even if the syntax is unfamiliar, reading through the comments can help you understand the logic of what we're doing. As you progress through the course, more of this code will start to make sense.

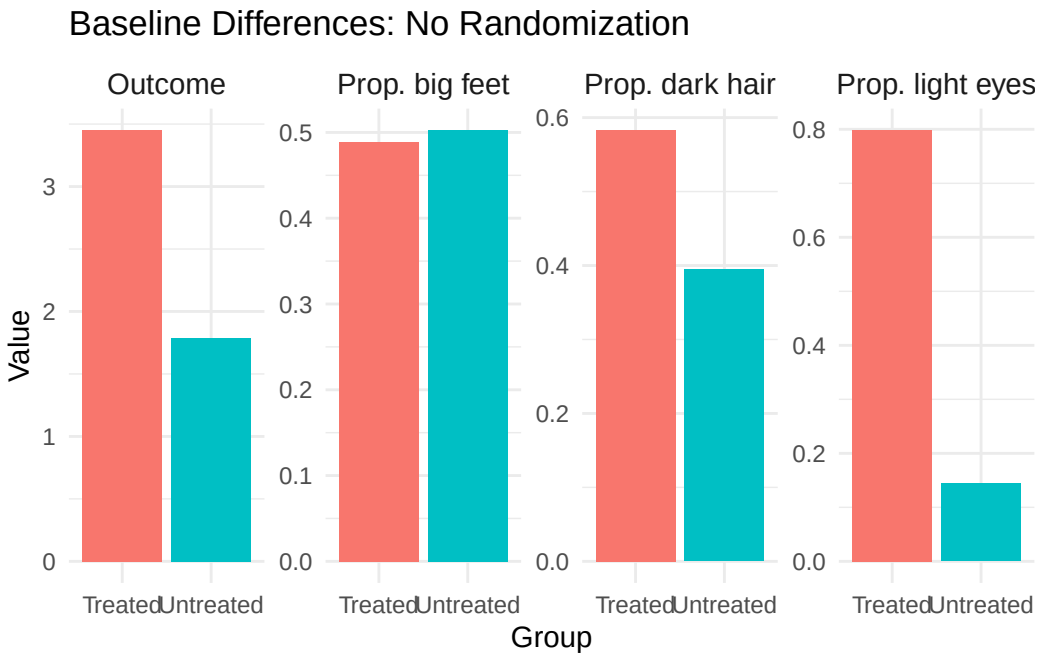
```r
# Load the tidyverse package for data manipulation and plotting
library(tidyverse)

# Set a "seed" for reproducibility
# This ensures we get the same random numbers each time we run the code
set.seed(181247)

# Define the TRUE causal effect of treatment
# We're setting this to ZERO so we know any observed difference is bias!
causal_effect <- 0

# Create a population of 10,000 individuals
n_population <- 10000

# Generate three random characteristics for each person
# Each characteristic is randomly 0 or 1 (like a coin flip)
eye_light <- sample(x = c(1, 0), size = n_population, replace = TRUE)
hair_dark <- sample(x = c(1, 0), size = n_population, replace = TRUE)
big_feet <- sample(x = c(1, 0), size = n_population, replace = TRUE)

# Generate random "noise" - unexplained variation in outcomes
err <- rnorm(n_population)

# Create an underlying "propensity" for treatment
# Notice: people with light eyes are MUCH more likely to be treated (coefficient = 2.1)
# People with dark hair are moderately more likely (coefficient = 0.83)
# Big feet has a small negative effect (coefficient = -0.11)
treatment_propensity <- 1 + 2.1 * eye_light + 0.83 * hair_dark - 0.11 * big_feet + err

# Assign treatment based on this propensity (treated if propensity > 2.25)
# This creates SELECTION BIAS: treatment is correlated with characteristics
treated <- ifelse(treatment_propensity > 2.25, 1, 0)

# Create the outcome variable
# The outcome depends on the same characteristics that predict treatment!
# But notice: we add "treated * causal_effect" which equals zero
# So treatment has NO real effect on the outcome
outcome <- 1 + 2.2 * eye_light + 1.2 * hair_dark - 0.01 * big_feet + treated * causal_effect

# Combine everything into a data frame
dat <- tibble(
  outcome = outcome,
```

```
  treated = treated,
  eye_light = eye_light,
  hair_dark = hair_dark,
  big_feet = big_feet
)
```

```
# Calculate the mean of each variable for treated vs untreated groups
baseline_diffs <- dat |>
  # Group the data by treatment status
  group_by(treated) |>
  # Calculate summary statistics for each group
  summarise(
    Outcome = mean(outcome),
    `Prop. light eyes` = mean(eye_light),
    `Prop. dark hair` = mean(hair_dark),
    `Prop. big feet` = mean(big_feet)
  ) |>
  # Reshape data from "wide" to "long" format for plotting
  pivot_longer(
    cols = Outcome:`Prop. big feet`,
    names_to = "variable",
    values_to = "value"
  ) |>
  # Create nice labels for the treatment groups

  mutate(group = ifelse(treated == 1, "Treated", "Untreated"))

# Create the bar plot
ggplot(baseline_diffs) +
  geom_bar(
    aes(x = group, y = value, fill = group),
    stat = "identity"
  ) +
  # Create separate panels for each variable
  facet_wrap(~variable, nrow = 1, scales = "free_y") +
  labs(
    x = "Group",
    y = "Value",
    title = "Baseline Differences: No Randomization"
  ) +
  theme_minimal() +
  theme(
```

```
    legend.position = "none",
    strip.text = element_text(size = 11)
  )
```

Figure 3.4: Baseline differences between treated and untreated groups without randomization. Despite the true causal effect being zero, we observe large differences in both characteristics and outcomes.



Notice in Figure 3.4 how the treated and untreated groups look very different! The treated group has a higher proportion of people with light eyes and dark hair. Most importantly, the treated group has a higher mean outcome—even though **we know the true causal effect is zero**. This is selection bias in action: people who "select into" treatment have characteristics that also lead to better outcomes.

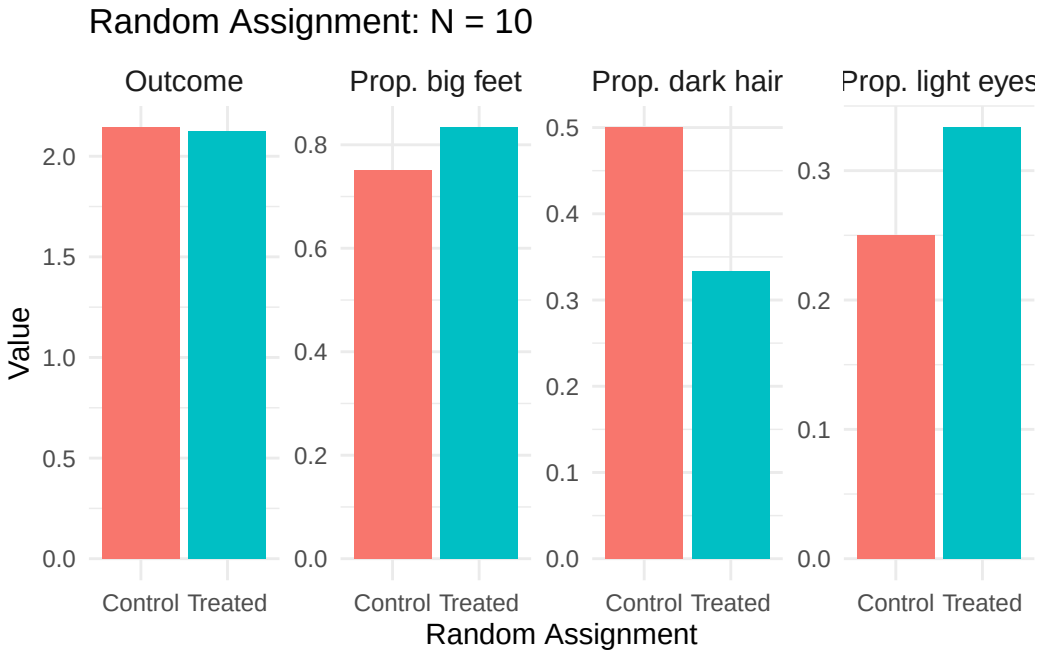
```
# Function to run the random assignment simulation for a given sample size
run_random_assignment <- function(data, sample_size, seed = 181247) {

  # Set seed for reproducibility
  set.seed(seed)

  # Take a random sample from the untreated population
  # (We use untreated only to avoid any prior treatment effects)
  sample_data <- data |>
```

```
    filter(treated == 0) |>
    slice_sample(n = sample_size)

  # RANDOMLY assign treatment - this is the key step!
  # Each person has a 50% chance of being assigned to treatment
  # This assignment is INDEPENDENT of their characteristics
  random_treatment <- sample(c(1, 0), size = sample_size, replace = TRUE)

  # Add the random treatment assignment to our sample
  sample_data <- sample_data |>
    mutate(rand_treat = random_treatment)

  # Calculate means for each randomly assigned group
  sample_data |>
    group_by(rand_treat) |>
    summarise(
      Outcome = mean(outcome),
      `Prop. light eyes` = mean(eye_light),
      `Prop. dark hair` = mean(hair_dark),
      `Prop. big feet` = mean(big_feet),
      .groups = "drop"
    ) |>
    mutate(sample_size = sample_size)
}

# Run the simulation for different sample sizes
results_n10 <- run_random_assignment(dat, sample_size = 10)
results_n25 <- run_random_assignment(dat, sample_size = 25)
results_n50 <- run_random_assignment(dat, sample_size = 50)
results_n100 <- run_random_assignment(dat, sample_size = 100)
results_n500 <- run_random_assignment(dat, sample_size = 500)


# Combine all results
all_results <- bind_rows(results_n10, results_n25, results_n50,
  results_n100, results_n500)

# Reshape for plotting
all_results_long <- all_results |>
  pivot_longer(
    cols = Outcome:`Prop. big feet`,
    names_to = "variable",
```

```
    values_to = "value"
  ) |>
  mutate(
    group = ifelse(rand_treat == 1, "Treated", "Control"),
    sample_size_label = paste("N =", sample_size)
  )
```

Now let's see what happens when we randomly assign treatment. Starting with a small sample (N=10), we see in Figure 3.5 that random assignment reduces but doesn't eliminate the differences between groups. With such a small sample, random chance can still produce imbalanced groups. With only 10 people, we still see noticeable differences between the randomly assigned treatment and control groups.

Figure 3.5: Random assignment with N=10. Some imbalance remains due to small sample size.



As we increase the sample size to N=25 and N=50, as shown in Figure 3.6 and Figure 3.7, the differences between treatment and control groups shrink. The LLN is at work: larger samples produce more balanced groups in terms of their background characteristics.

By N=500, as show in Figure 3.8, the treatment and control groups are nearly identical across all characteristics. The difference in outcomes between groups is now close to zero—the true causal effect! Any remaining difference is just random variation, not systematic selection bias.

Figure 3.6: Random assignment with N=25.

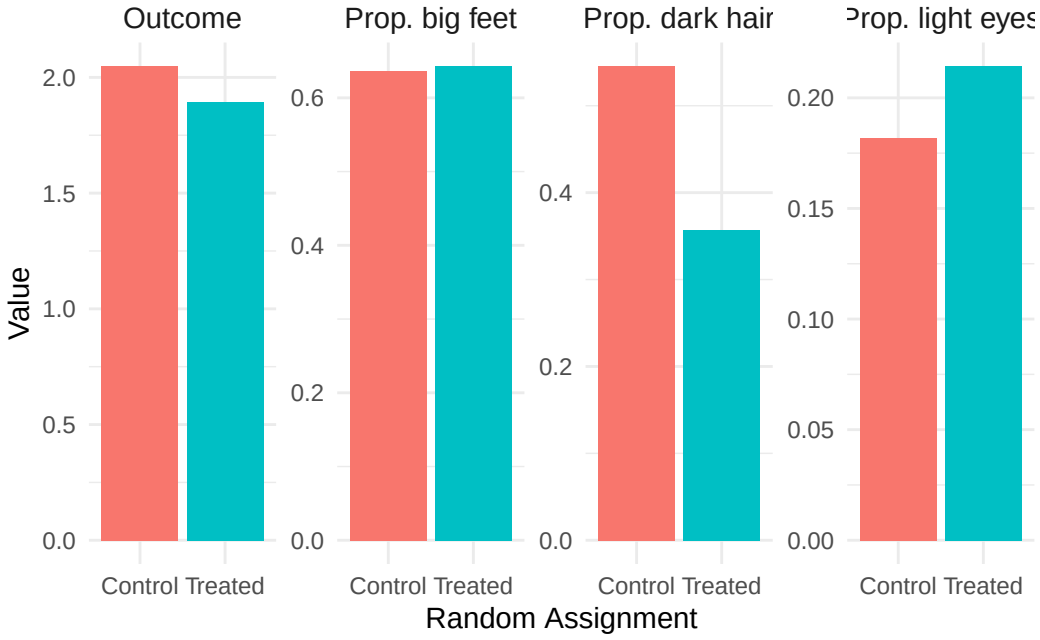


Figure 3.7: Random assignment with N=50.

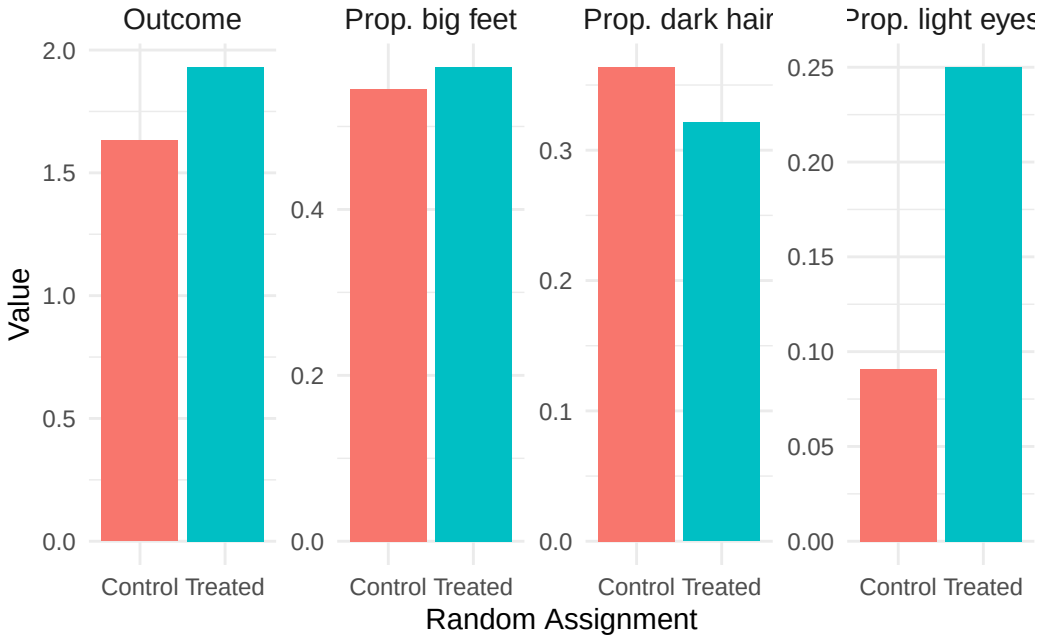
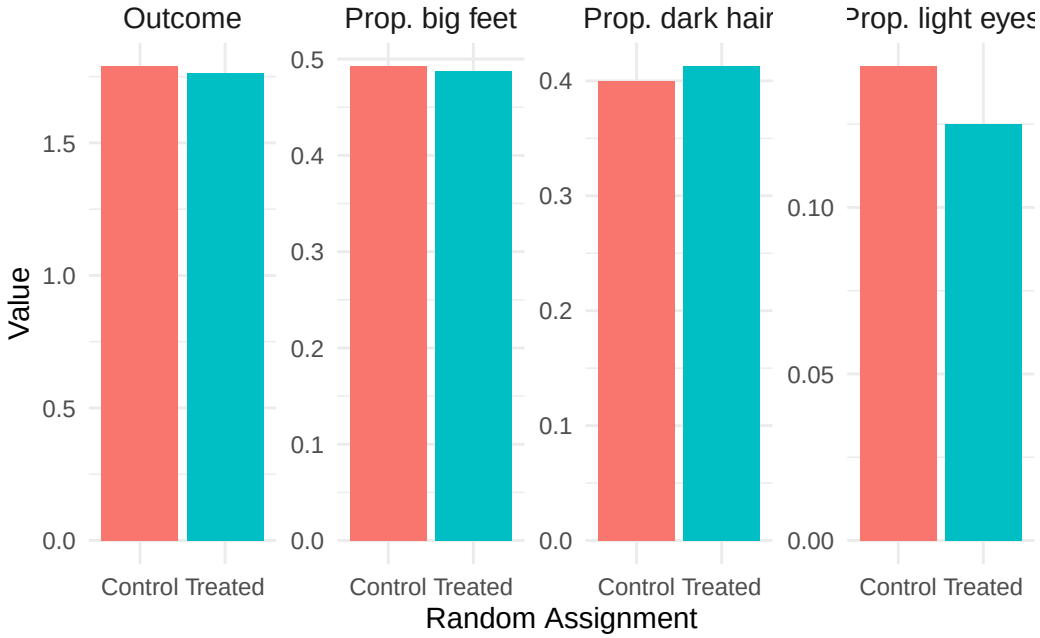


Figure 3.8: Random assignment with N=500. The treatment and control groups are now nearly identical in their characteristics, and the difference in outcomes is close to the true causal effect of zero.



3.6 The Potential Outcomes Framework

Now that we intuitively understand why simple comparisons can be misleading and how randomization addresses this problem, let's formalize these concepts mathematically using the **Potential Outcomes Framework**.

3.6.1 Defining Potential Outcomes

Consider our example of estimating the causal effect of college on wages. For any individual i, we can define two **potential outcomes**:

- Y_{1i}: The wage individual i would earn **if they attend college**
- Y_{0i}: The wage individual i would earn **if they do not attend college**

The causal effect of college for individual i is simply the difference between these potential outcomes:

$$\kappa_i = Y_{1i} - Y_{0i}$$

This framework captures the "multiverse" intuition: Y_{1i} is College Morty's wage, Y_{0i} is Morty C-137's wage, and κ_i is the causal effect of college for Morty.

3.6.2 The Fundamental Problem of Causal Inference

The fundamental problem is that we only observe **one** potential outcome for each individual. If person i attends college, we observe Y_{1i} but never see Y_{0i}. If they don't attend, we observe Y_{0i} but never see Y_{1i}. We never observe both potential outcomes for the same individual.

Let D_i be a treatment indicator:

$$D_i = \begin{cases} 1 & \text{if individual } i \text{ is treated (attends college)} \\ 0 & \text{otherwise} \end{cases}$$

What we actually observe is:

$$Y_i = D_i \cdot Y_{1i} + (1 - D_i) \cdot Y_{0i}$$

That is, we observe the treated outcome for treated individuals and the untreated outcome for untreated individuals—but never both.

3.6.3 Selection Bias Formalized

Now we can formally see why simple comparisons are problematic. Suppose we compare average outcomes between those who are treated ($D_i = 1$) and those who are untreated ($D_i = 0$). The difference in group means is:

$$E[Y_i|D_i = 1] - E[Y_i|D_i = 0]$$

Since we only observe Y_{1i} for the treated and Y_{0i} for the untreated, this becomes:

$$E[Y_{1i}|D_i = 1] - E[Y_{0i}|D_i = 0]$$

Here's the key insight. Let's add and subtract a term that represents the counterfactual—what the treated group's outcome would have been without treatment:

$$E[Y_{1i}|D_i = 1] - E[Y_{0i}|D_i = 0] + \underbrace{\{E[Y_{0i}|D_i = 1] - E[Y_{0i}|D_i = 1]\}}_{=0}$$

Rearranging terms:

$$\underbrace{E[Y_{1i}|D_i = 1] - E[Y_{0i}|D_i = 1]}_{\text{Causal effect for the treated}} + \underbrace{\{E[Y_{0i}|D_i = 1] - E[Y_{0i}|D_i = 0]\}}_{\text{Selection bias}}$$

! The Selection Bias Decomposition

A simple difference in group means equals the causal effect **plus** selection bias:

$$\underbrace{E[Y_i|D_i = 1] - E[Y_i|D_i = 0]}_{\text{Observed difference}} = \underbrace{\kappa}_{\text{Causal effect}} + \underbrace{\{E[Y_{0i}|D_i = 1] - E[Y_{0i}|D_i = 0]\}}_{\text{Selection bias}}$$

The selection bias term compares the **untreated potential outcomes** of the treated group to the untreated potential outcomes of the control group. In the health insurance example, this captures differences in baseline health (absent insurance) between those who choose to get insurance and those who don't. If people who get insurance would be healthier even without insurance (perhaps because they're wealthier or more health-conscious), then selection bias is positive, and the simple comparison overstates the causal effect.

3.6.4 How Randomization Eliminates Selection Bias

When treatment is randomly assigned, the treatment and control groups are drawn from the same population. By the Law of Large Numbers, their average characteristics—including their average potential outcomes—converge to each other. Formally:

$$E[Y_{0i}|D_i = 1] = E[Y_{0i}|D_i = 0]$$

This says that the average untreated potential outcome is the same for both groups. Plugging this into our decomposition:

$$E[Y_i|D_i = 1] - E[Y_i|D_i = 0] = \kappa + \{E[Y_{0i}|D_i = 1] - E[Y_{0i}|D_i = 0]\} = \kappa + 0 = \kappa$$

With random assignment, the simple difference in means equals the causal effect. Selection bias disappears.

3.7 The Oregon Health Insurance Experiment

Most econometric research cannot rely on randomization to identify causal effects. We typically work with observational data where treatment is not randomly assigned, and much of this course focuses on methods for addressing selection bias in such settings. However, there have been a few landmark studies that used randomization to study important policy questions. One of the most influential is the **Oregon Health Insurance Experiment**.

3.7.1 Context: Medicaid Expansion

Throughout the 2000s, many U.S. states expanded their Medicaid programs to cover more low-income residents. Most states simply extended coverage to everyone who became newly eligible. But in 2008, Oregon did something unique: facing limited resources, they offered Medicaid coverage through a public lottery.

3.7.2 The Lottery

Oregon's approach created a natural randomized experiment. People who wanted coverage could sign up for the lottery. Winners received the opportunity to apply for the Oregon Health Plan (the state's Medicaid program), while losers remained uninsured. To be eligible, applicants had to be legal Oregon residents aged 19-64, uninsured for the past six months, with income below the federal poverty line, and not otherwise eligible for health insurance.

Because lottery winners were chosen randomly, the comparison between winners and losers provides a valid causal estimate of the effect of health insurance—free from the selection bias that plagues observational studies.

3.7.3 Results

Researchers followed lottery participants and found that lottery winners (relative to losers) experienced:

1. **Increased take-up of Medicaid:** The lottery actually changed insurance status, confirming the treatment "worked"
2. **Increased use of medical resources:** Winners used more hospital and outpatient services
3. **Small improvements in overall health:** Physical health improvements were modest, though mental health and financial security improved substantially

These findings are important because they come from a rigorous RCT rather than observational comparisons. The study provides some of the cleanest evidence we have on what health insurance actually *causes*, as opposed to what it merely correlates with.

Discussion Question

The Oregon experiment found modest effects of insurance on physical health. Does this mean expanding health insurance is not worthwhile? What other outcomes might matter for policy decisions?

3.8 Summary and Conclusion

This chapter introduced the fundamental concepts underlying causal inference in econometrics. We began with the intuition that simple comparisons between treated and untreated groups can be misleading due to selection bias—the groups may differ in ways beyond just the treatment. The "multiverse" thought experiment illustrated what we really want: *ceteris paribus* comparisons where everything except the treatment is held constant.

Randomized control trials approximate this ideal by randomly assigning treatment, which ensures that treatment and control groups are comparable on average. The Law of Large Numbers guarantees that with large enough samples, random assignment eliminates selection bias.

We then formalized these ideas using the potential outcomes framework, which defines causal effects as the difference between what would happen with and without treatment. The fundamental problem is that we only observe one potential outcome per individual. Simple comparisons of means capture both the causal effect and selection bias, but random assignment makes selection bias vanish.

3.8.1 Key Takeaways

- **Selection bias** arises when treated and untreated groups differ systematically beyond just the treatment—we're comparing apples to oranges
- **Ceteris paribus** ("all else equal") comparisons isolate the causal effect by holding everything constant except the treatment
- **Randomized control trials** eliminate selection bias by ensuring treatment and control groups are drawn from the same population
- The **Law of Large Numbers** guarantees that with large samples, randomly assigned groups have identical average characteristics

- The **potential outcomes framework** formalizes causal effects as $\kappa = Y_{1i} - Y_{0i}$, but we only observe one potential outcome per individual
- Simple differences in means equal: Causal effect + Selection bias. Randomization sets selection bias to zero.

3.9 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

4 Bivariate Regression and Ordinary Least Squares

Key Questions

- How do we mathematically model the relationship between two variables?
- What is the population regression model and how do we interpret its parameters?
- How do we derive the OLS estimator?
- What assumptions are required for OLS to be unbiased?
- What is omitted variable bias and when does it arise?
- When can we interpret OLS estimates as causal effects?
- How do we measure the precision of our estimates?

Suggested Readings

- Wooldridge (2019), Ch. 2

In Chapter 3, we learned that randomized control trials provide the gold standard for causal inference by eliminating selection bias through random assignment. But most economic questions cannot be answered with experiments. We typically work with observational data where "treatment" is not randomly assigned. This chapter introduces **Ordinary Least Squares (OLS)** regression, the workhorse tool of econometrics. We focus here on the **bivariate** case—a single independent variable—to build intuition before extending to multiple variables.

4.1 Modeling Relationships Between Variables

Suppose we have two variables, y and x. In econometrics, we are often interested in one of several related goals: estimating the **causal effect** of x on y, explaining variation in y using x, or simply describing how y varies with x. OLS is a tool that can help with all of these—though as we'll see, whether we can interpret results causally depends on assumptions that may or may not hold.

But what do we even *mean* when we say we want to study "how x is related to y"? We need a precise, mathematical way to operationalize this question. The most straightforward approach is to ask: what is the *average* value of y for each different value that x can take?

Consider some examples. How can we quantify the gender pay gap? We compare the *average* wage for men versus women. How do we measure the returns to education? We compare the *average* wage of people with different levels of education. In both cases, we are asking about how the average of one variable changes across values of another.

4.1.1 Conditional Expectations

This idea of finding "the average value of y for a given value of x" is exactly what we call the **conditional expectation**. Formally, the conditional expectation of y given x is written as:

$$E[Y|X = x]$$

This notation asks: if we knew that X took on some specific value x, what would we expect Y to be, on average? The conditional expectation is often the central quantity we want to model and estimate in econometrics.

4.1.2 Example: Education and Wages

Suppose X represents education level, which can take values $X = \{\text{hsdeg}, \text{colldeg}, \text{profdeg}\}$ (high school degree, college degree, professional degree), and Y represents hourly wage.

For each education level, we can compute the average wage among people in that group. This will gives us a good idea of how wages are related to education.

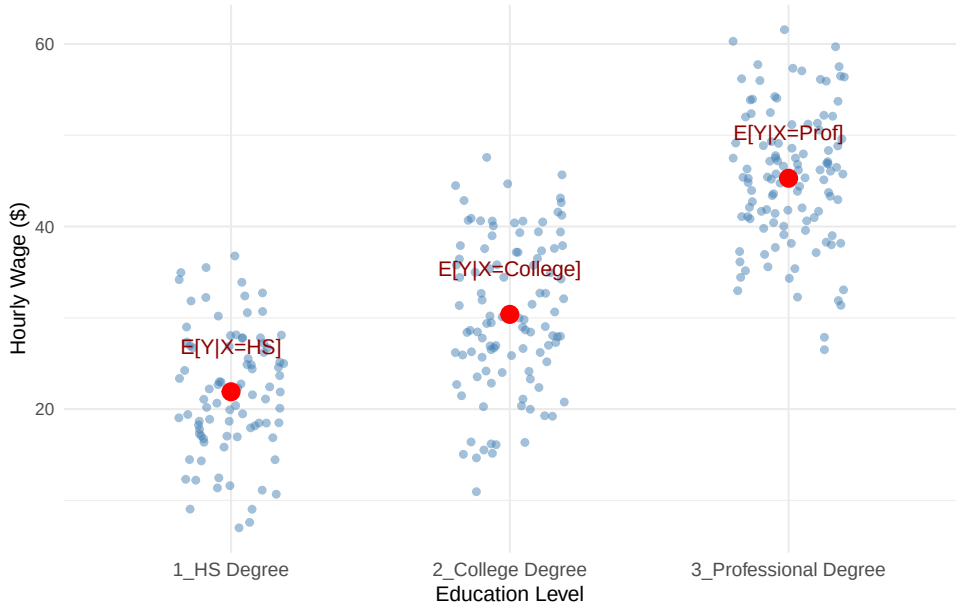
Figure 4.1 is a scatterplot with an indiviudal's wage on the y-axis and their education level on the x-axis, with each point plotted in blue. The red points in Figure 4.1 show the conditional expectations—the average wage for individuals *within each education category,* i.e., $E[Y|X = HS], E[Y|X = College], \text{and} E[Y|X = Prof]$.

These averages tell us how we can expect wages to change when we increase education, which is precisely what we set out to measure.

4.2 The Population Regression Model

Now that we understand what we want to estimate (the conditional expectation), we need a framework for *how* to estimate it. Before we can estimate anything, we must first describe the underlying relationship we believe exists in the population.

Figure 4.1: Wages by education level. Each point represents an individual. Red points show the conditional expectation (average wage) for each education group.



4.2.1 The Bivariate Model

The simplest specification is called the **bivariate** (or simple) regression model:

$$y = \beta_0 + \beta_1 x + \mu \tag{4.1}$$

where:

- y is the **dependent variable** (also called the explained, response, or outcome variable)
- x is the **independent variable** (also called the explanatory, control, or predictor variable)
- μ is the **error term**, which represents all unobserved factors that affect y
- β_0 and β_1 are **population parameters**—the "true" values describing the relationship between x and y in the population

The Error Term

The error term μ is crucial. It captures everything that affects y other than x. In a wage equation, μ might include ability, motivation, family connections, luck, and countless other factors we cannot observe or measure.

4.2.2 Connecting to Conditional Expectations

How does this regression model relate to our goal of estimating $E[Y|X]$? If we assume that the error term has mean zero conditional on x—that is, $E[\mu|x] = 0$—then:

$$E[y|x] = E[\beta_0 + \beta_1 x + \mu|x] = \beta_0 + \beta_1 x + E[\mu|x] = \beta_0 + \beta_1 x$$

This is exactly the conditional expectation we set out to estimate! The regression line $\beta_0 + \beta_1 x$ traces out the average value of y for each value of x.

4.2.3 Interpreting the Coefficients

The coefficients in Equation 4.1 have specific interpretations:

- β_0 **(the intercept):** The predicted value of y when $x = 0$. Depending on the context, this may or may not have a meaningful interpretation.
- β_1 **(the slope):** The **marginal effect** of x on y. Mathematically, it is the derivative:

$$\frac{dy}{dx} = \beta_1$$

This tells us: when x increases by one unit, y changes by β_1 units, on average.

4.2.4 Example: Education and Wages

A regression model relating years of education to hourly wages might be:

$$wage = \beta_0 + \beta_1 \cdot educ + \mu$$

where *wage* is dollars per hour, *educ* is years of education, and μ captures all other determinants of wages (experience, ability, etc.).

If $\beta_1 = 2.50$, this would mean that each additional year of education is associated with \$2.50 higher hourly wages, on average.

4.3 The Regression Line: Predicted Values and Residuals

The entire goal of linear regression is to use our *sample* of the data from the wider *population* to estimate the values of β_0 and β_1 that are the best possible estimates we can generate. We call these estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, and the method we use to calculate these estimates the **estimator**. We can never actually get the *true* population values β_0 and β_1; we have to rely on estimates!

Before we derive estimators for β_0 and β_1, let's introduce some key concepts that will guide our approach.

4.3.1 Fitted Values and the Regression Line

Once we have estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, we can construct the **OLS regression line** (or fitted line):

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

For any value of x, this line gives us $\hat{y}$, the **predicted value** (or **fitted value**) of y. The regression line represents our best guess for the average value of y at each level of x.

4.3.2 Residuals: The Prediction Errors

Of course, our predictions won't be perfect. The **residual** for observation i is the difference between the actual value and the predicted value:

$$\hat{\mu}_i = y_i - \hat{y}_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

The residual measures how much our prediction "misses" for each observation. Positive residuals mean we underpredicted; negative residuals mean we overpredicted.

4.3.3 A Numerical Example

Let's see these concepts with actual numbers. The table below shows the first six observations from a regression of MPG on horsepower, along with the predicted values and residuals:

Table 4.1: Example regression data showing actual values, predicted values, and residuals.

	x (Horsepower)	y (MPG)	ŷ (Predicted)	û (Residual)
Mazda RX4	110	21.0	22.59	-1.59

Table 4.1: Example regression data showing actual values, predicted values, and residuals.

	x (Horsepower)	y (MPG)	$\hat{y}$ (Predicted)	$\hat{u}$ (Residual)
Mazda RX4 Wag	110	21.0	22.59	-1.59
Datsun 710	93	22.8	23.75	-0.95
Hornet 4 Drive	110	21.4	22.59	-1.19
Hornet Sportabout	175	18.7	18.16	0.54
Valiant	105	18.1	22.93	-4.83

Notice how the residual column equals the actual value minus the predicted value. For the first observation, the car has 110 horsepower and gets 21 MPG. Our regression predicts it should get about 22.59 MPG, so the residual is $21 - 22.59 = -1.59$. The negative residual means we *overpredicted*—the car actually gets slightly worse fuel economy than our model predicted.

4.3.4 The Goal: Minimize the Residuals

Here's the crucial idea: we want to choose $\hat{\beta}_0$ and $\hat{\beta}_1$ so that our predictions are as good as possible. In other words, we want to make the residuals as small as possible.

But we can't simply minimize the sum of residuals $\sum \hat{\mu}_i$, because positive and negative residuals would cancel out. Instead, we minimize the **Sum of Squared Residuals (SSR)**:

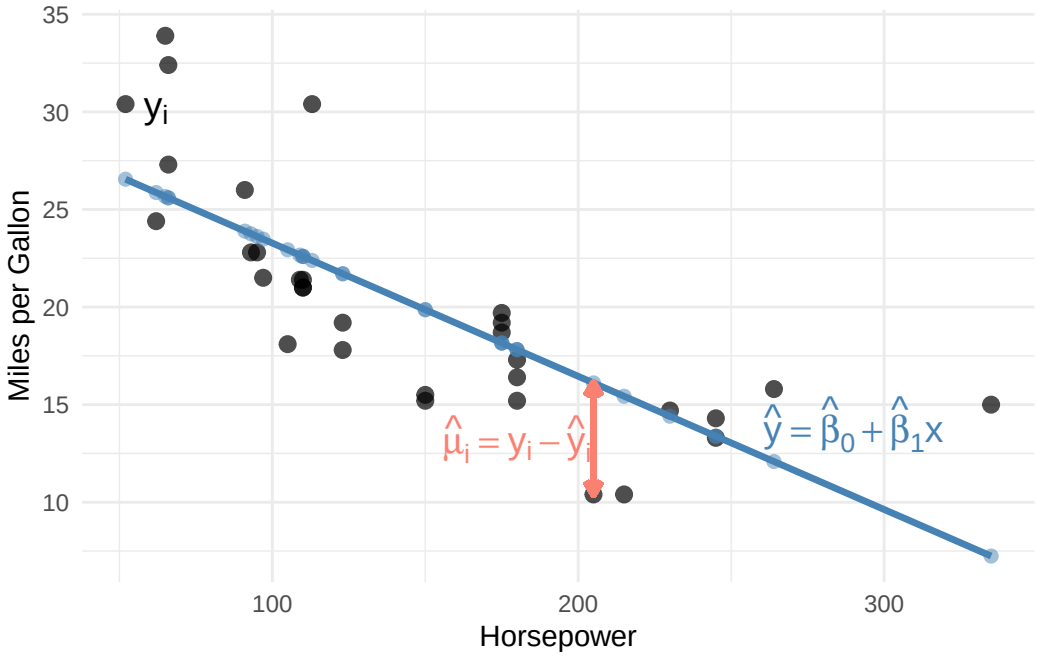
$$SSR = \sum_{i=1}^{n} \hat{\mu}_i^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

This is exactly where the name "Ordinary Least Squares" comes from: we find the estimates that make the sum of squared residuals as small as possible.

4.4 Deriving the OLS Estimator

Now we turn to the mechanics of estimation. The **population** regression model describes the true relationship, but we never observe the entire population or the true values of β_0 and β_1. Instead, we have a **sample** of data and must *estimate* the population parameters.

Figure 4.2: Components of the regression line. Black points show actual values, the blue line shows predicted values, and the salmon segment illustrates a residual.



4.4.1 The Minimization Problem

Suppose we have a sample of n observations, $\{(x_i, y_i) : i = 1, 2, ..., n\}$. As discussed above, we want to find the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the sum of squared residuals:

$$\min_{\hat{\beta}_0, \hat{\beta}_1} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

This is a calculus optimization problem! The SSR is a function of two unknowns ($\hat{\beta}_0$ and $\hat{\beta}_1$), and we want to find the values that minimize it.

4.4.2 First-Order Conditions

To find the minimum, we take partial derivatives with respect to each parameter, set them equal to zero, and solve. These are called the **first-order conditions**.

Partial derivative with respect to $\hat{\beta}_0$:

$$\frac{\partial SSR}{\partial \hat{\beta}_0} = -2\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

Dividing by -2 and rearranging:

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

Partial derivative with respect to $\hat{\beta}_1$:

$$\frac{\partial SSR}{\partial \hat{\beta}_1} = -2\sum_{i=1}^{n} x_i(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

Dividing by -2:

$$\sum_{i=1}^{n} x_i(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

4.4.3 Solving for the Intercept

From the first condition, distributing the summation:

$$\sum_{i=1}^{n} y_i - n\hat{\beta}_0 - \hat{\beta}_1 \sum_{i=1}^{n} x_i = 0$$

Dividing by n and recognizing sample means:

$$\bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}$$

Solving for the intercept:

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{4.2}$$

This tells us that the regression line always passes through the point $(\bar{x}, \bar{y})$—the "center" of the data.

4.4.4 Solving for the Slope

Substituting Equation 4.2 into the second first-order condition and working through the algebra (using summation properties from Appendix B):

$$\sum_{i=1}^{n} x_i[y_i - (\bar{y} - \hat{\beta}_1\bar{x}) - \hat{\beta}_1 x_i] = 0$$

After distributing and rearranging:

$$\sum_{i=1}^{n} x_i(y_i - \bar{y}) = \hat{\beta}_1 \sum_{i=1}^{n} x_i(x_i - \bar{x})$$

Using the summation properties $\sum x_i(y_i - \bar{y}) = \sum(x_i - \bar{x})(y_i - \bar{y})$ and $\sum x_i(x_i - \bar{x}) = \sum(x_i - \bar{x})^2$[1]:

$$\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \hat{\beta}_1 \sum_{i=1}^{n}(x_i - \bar{x})^2$$

Solving for $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.3}$$

4.4.5 Confirming It's a Minimum

How do we know this critical point is a minimum rather than a maximum? We check the second-order conditions. The second partial derivative of SSR with respect to $\hat{\beta}_1$ is:

$$\frac{\partial^2 SSR}{\partial \hat{\beta}_1^2} = 2\sum_{i=1}^{n} x_i^2 > 0$$

Since this is positive (as long as there's variation in x), the SSR function is convex, and our solution is indeed a minimum.

[1] Why can we replace $\sum x_i(y_i - \bar{y})$ with $\sum(x_i - \bar{x})(y_i - \bar{y})$? Expand the second expression: $\sum(x_i - \bar{x})(y_i - \bar{y}) = \sum x_i(y_i - \bar{y}) - \bar{x}\sum(y_i - \bar{y})$. But from Result 1 in Appendix B, deviations from the mean always sum to zero, so $\sum(y_i - \bar{y}) = 0$. The second term vanishes and the two expressions are equal. The same logic applies to $\sum x_i(x_i - \bar{x}) = \sum(x_i - \bar{x})^2$.

! The OLS Slope Formula

The OLS slope estimate equals the sample covariance of x and y divided by the sample variance of x:

$$\hat{\beta}_1 = \frac{Cov(x, y)}{Var(x)} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

This can also be written in terms of the correlation coefficient:

$$\hat{\beta}_1 = \rho_{xy} \frac{\sigma_y}{\sigma_x}$$

This last expression reveals something important: bivariate OLS is essentially a scaled version of the correlation between x and y. And as we discussed in Chapter 1, correlation does not imply causation! We must be very careful about interpreting bivariate OLS results causally.

4.5 A Worked Example by Hand

Before we let R do the heavy lifting, let's apply the OLS formulas to a tiny dataset to make sure we understand what's going on. Suppose we survey 5 workers and record their years of education (x) and hourly wage in dollars (y):

Worker	x_i (Years of Education)	y_i (Hourly Wage)
1	10	12
2	12	15
3	14	17
4	16	22
5	18	24

4.5.1 Step 1: Compute the Means

$$\bar{x} = \frac{10 + 12 + 14 + 16 + 18}{5} = \frac{70}{5} = 14$$

$$\bar{y} = \frac{12 + 15 + 17 + 22 + 24}{5} = \frac{90}{5} = 18$$

4.5.2 Step 2: Compute the Deviations

We need $(x_i - \bar{x})$, $(y_i - \bar{y})$, their product, and $(x_i - \bar{x})^2$:

x_i	y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
10	12	-4	-6	24	16
12	15	-2	-3	6	4
14	17	0	-1	0	0
16	22	2	4	8	4
18	24	4	6	24	16
			Sum	**62**	**40**

Notice that the deviations $(x_i - \bar{x})$ sum to zero, just as Result 1 in Appendix B promises.

4.5.3 Step 3: Compute the Slope

Applying Equation 4.3:

$$\hat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = \frac{62}{40} = 1.55$$

Each additional year of education is associated with \$1.55 higher hourly wages in our sample.

4.5.4 Step 4: Compute the Intercept

Applying Equation 4.2:

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} = 18 - 1.55 \times 14 = 18 - 21.7 = -3.7$$

Our estimated regression line is:

$$\hat{y}_i = -3.7 + 1.55\, x_i$$

The intercept (-3.7) is the predicted wage for someone with zero years of education. This doesn't have a meaningful real-world interpretation here—nobody in our data has zero years of education—but the intercept is still necessary for the line to fit the data correctly.

4.5.5 Step 5: Check Fitted Values and Residuals

Let's plug each x_i into our equation and compute the residuals $\hat{\mu}_i = y_i - \hat{y}_i$:

x_i	y_i	$\hat{y}_i$	$\hat{\mu}_i = y_i - \hat{y}_i$
10	12	11.8	0.2
12	15	14.9	0.1
14	17	18.0	-1.0
16	22	21.1	0.9
18	24	24.2	-0.2
		Sum of residuals	**0.0**

As expected, the residuals sum to zero. This isn't a coincidence—it's guaranteed by the first-order condition from Section 4.4.2, which requires $\sum(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$.

> **Try It Yourself**
>
> Before moving on, try reproducing this example on paper or in a calculator. Getting comfortable with the mechanics of these formulas will make everything that follows easier to understand.

4.6 Computing OLS in R

While we derived the formulas by hand, in practice we use software to compute OLS estimates. In R, the `lm()` function (for "linear model") performs OLS regression.

4.6.1 Basic Syntax

```r
# General syntax
lm(y ~ x, data = your_data)
```

The formula `y ~ x` specifies that y is the dependent variable and x is the independent variable. An intercept is included by default.

4.6.2 Example: MPG and Horsepower

Let's estimate the relationship between fuel efficiency (MPG) and engine horsepower using the built-in `mtcars` dataset:

```
# Load the data
data(mtcars)

# Estimate the regression
lm_mpg <- lm(mpg ~ hp, data = mtcars)

# View the results
summary(lm_mpg)
```

```
Call:
lm(formula = mpg ~ hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7121 -2.1122 -0.8854  1.5819  8.2360

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
hp          -0.06823    0.01012  -6.742 1.79e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024,    Adjusted R-squared:  0.5892
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

The output shows our estimates: $\hat{\beta}_0 = 30.10$ and $\hat{\beta}_1 = -0.07$. The slope suggests that each additional unit of horsepower is associated with about 0.07 fewer miles per gallon. Similarly, according to this estimate, a car with 50 additional horsepower is associated with about 3.5 (50×0.07) fewer miles per gallon.

A one-unit increase isn't always the most meaningful way to think about a coefficient. When the units of x are hard to interpret on their own, it can be helpful to consider a **one standard deviation** increase instead. In this dataset, the standard deviation of horsepower is about 68.6. So a one standard deviation increase in horsepower is associated with a change in fuel

economy of roughly $68.6 \times (-0.07) \approx 4.8$ fewer miles per gallon. This gives a better sense of what a "typical" difference in horsepower implies for fuel economy.

4.7 Goodness-of-Fit: R-Squared

How well does our regression line fit the data? The **R-squared** (R^2) provides one measure of goodness-of-fit.

Define:

- **Total Sum of Squares (SST):** $SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$ — total variation in y
- **Explained Sum of Squares (SSE):** $SSE = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$ — variation explained by the regression
- **Residual Sum of Squares (SSR):** $SSR = \sum_{i=1}^{n}\hat{\mu}_i^2$ — unexplained variation

The R-squared is:

$$R^2 = \frac{SSE}{SST} = 1 - \frac{SSR}{SST}$$

This measures the fraction of the total variation in y that is explained by the regression. R^2 ranges from 0 (no explanatory power) to 1 (perfect fit). We usually interpret the calculated value as a percent. For example if the $R^2 = 0.386$, you might say something the model *explains about 39% of the variation in y.

> ⚠ Interpreting R-Squared
>
> Be careful interpreting R^2, especially in the context of causal inference. A high R^2 does not mean we have a causal estimate—it just means x is correlated with y. Conversely, a low R^2 does not mean our estimate is wrong or unimportant. In cross-sectional data, R^2 values of 0.1 to 0.3 are common and can still represent economically meaningful relationships.

4.8 The Gauss-Markov Assumptions

We've derived the OLS formulas, but how do we know if the estimates are any good? The formulas will always give us *some* numbers, but whether those numbers have useful statistical properties depends on certain conditions being met.

The **Gauss-Markov assumptions** are a set of conditions under which OLS has desirable properties. Understanding these assumptions is crucial: they tell us when we can trust our estimates and when we should be skeptical.

4.8.1 Assumption 1: Linear in Parameters

> **i** Gauss-Markov Assumption 1: Linear in Parameters
>
> The population model is: $y = \beta_0 + \beta_1 x + \mu$

This assumption states that the dependent variable y can be written as a linear function of the parameters β_0 and β_1, plus an error term.

What this does NOT mean: This assumption is less restrictive than it might first appear. It does *not* require that y and x have a linear relationship. We can model many non-linear relationships by transforming the variables. For example:

- $y = \beta_0 + \beta_1 x^2 + \mu$ is linear in parameters (we can define $z = x^2$ and estimate $y = \beta_0 + \beta_1 z + \mu$)
- $\log(y) = \beta_0 + \beta_1 x + \mu$ is linear in parameters
- $y = \beta_0 + \beta_1 \log(x) + \mu$ is linear in parameters

What would violate this assumption: Models where parameters enter non-linearly, such as $y = \beta_0 + x^{\beta_1} + \mu$ or $y = \frac{1}{1+e^{-\beta_0 - \beta_1 x}} + \mu$ (a logistic function). These require different estimation techniques.

4.8.2 Assumption 2: Random Sampling

> **i** Gauss-Markov Assumption 2: Random Sampling
>
> We have a random sample of size n from the population: $\{(x_i, y_i) : i = 1, ..., n\}$

This assumption states that each observation in our dataset is drawn randomly and independently from the population of interest. Each unit in the population has an equal chance of being selected, and the selection of one unit doesn't affect the probability of selecting another.

Why it matters: Random sampling ensures that our sample is representative of the population. If certain types of observations are more likely to be included, our estimates may not generalize to the full population.

When this might be violated:

- **Self-selection:** If people choose whether to participate in a survey, those who respond may differ systematically from those who don't. For example, a wage survey might oversample high earners if low-wage workers are less likely to respond.
- **Convenience sampling:** Surveying only people who are easy to reach (e.g., college students for a study about the general population).
- **Time series data:** Observations collected over time are often correlated with each other. Today's stock price depends on yesterday's stock price, violating independence.
- **Clustered data:** Students within the same classroom, or workers within the same firm, may be more similar to each other than to randomly selected individuals.

4.8.3 Assumption 3: Variation in the Independent Variable

> **i** Gauss-Markov Assumption 3: Sample Variation in x
>
> The independent variable has some variation in the sample: $\sum(x_i - \bar{x})^2 > 0$

This assumption simply requires that x is not constant in our sample. If everyone in our sample has the same value of x, we cannot estimate how changes in x relate to changes in y.

Why it matters: Look back at our formula for $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

If all x_i equal $\bar{x}$, the denominator is zero, and the formula is undefined. Intuitively, if everyone has 12 years of education, we have no way to estimate what would happen with 13 years of education.

In practice: This assumption is almost always satisfied. It would only fail in unusual circumstances, like if you accidentally collected a sample where everyone happened to have the exact same value of x. However, having *little* variation in x (even if not zero) can lead to imprecise estimates, as we'll see when discussing variance.

4.8.4 Assumption 4: Zero Conditional Mean

> **i** Gauss-Markov Assumption 4: Zero Conditional Mean
>
> The expected value of the error term is zero for any value of x: $E[\mu|x] = 0$

This is the most important and most frequently violated assumption. It states that the average value of all the unobserved factors affecting y is the same regardless of the value of x.

Understanding the assumption: Let's break this down carefully. The error term μ contains everything that affects y other than x. The zero conditional mean assumption says that if we could collect all people with $x = 10$ and compute the average of their μ values, it would be zero. And if we collected all people with $x = 20$ and averaged their μ values, that would also be zero. The average of unobservables is the same at every level of x.

Example: Education and Wages

Consider the regression:

$$wage = \beta_0 + \beta_1 \cdot educ + \mu$$

The error term μ includes ability, motivation, family connections, and other unobserved factors. The zero conditional mean assumption requires:

$$E[\mu|educ = 8] = E[\mu|educ = 12] = E[\mu|educ = 16] = 0$$

Is this plausible? Probably not. People with higher ability tend to get more education (because school is easier for them, or because they enjoy learning). So among people with 16 years of education, average ability is probably higher than among people with 8 years. This means $E[\mu|educ = 16] > E[\mu|educ = 8]$, violating the assumption.

Why it matters: When this assumption fails, OLS is **biased**. The estimate $\hat{\beta}_1$ will systematically overshoot or undershoot the true β_1. In the education example, if high-ability people get more education, our estimate of the "return to education" will partly capture the "return to ability," making education look more valuable than it really is.

This is the essence of **omitted variable bias**, which we'll explore in more detail in Chapter 5.

4.8.5 Assumption 5: Homoskedasticity

> **i** Gauss-Markov Assumption 5: Homoskedasticity
>
> The variance of the error term is constant across all values of x: $Var(\mu|x) = \sigma^2$

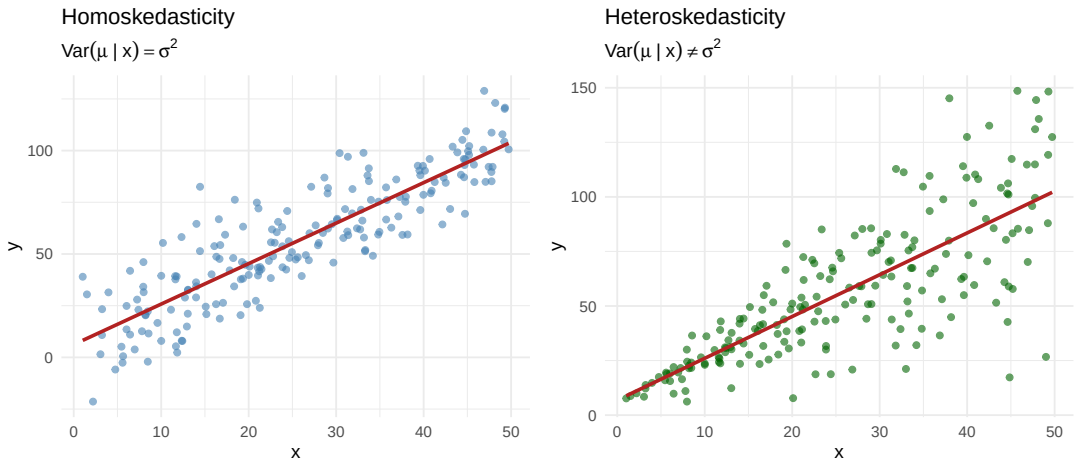
This assumption states that the spread of the error term is the same regardless of the value of x. The "noise" in our data is equally noisy everywhere.

Understanding the assumption: While Assumption 4 concerns the *mean* of μ at different values of x, Assumption 5 concerns the *variance*. Homoskedasticity (from Greek: "same scatter") means the scatter of points around the regression line is constant. **Heteroskedasticity** (different scatter) occurs when the variance of μ depends on x.

Example: Income and Spending

Consider a regression of consumption spending on income. For low-income households, spending is relatively constrained—there's not much room for variation because most income goes to necessities. For high-income households, there's much more discretion in spending patterns, leading to greater variation. This would create heteroskedasticity: $Var(\mu|income)$ increases with income.

Figure 4.3: Left: Homoskedasticity—the spread of points around the line is constant. Right: Heteroskedasticity—the spread increases with x.



Why it matters: Unlike Assumption 4, violating homoskedasticity does **not** make OLS biased. Your estimate of β_1 is still centered on the true value. However, heteroskedasticity affects the *variance* and *standard errors* of the estimator. The usual standard error formulas become incorrect, which can lead to invalid hypothesis tests and confidence intervals.

We'll discuss how to handle heteroskedasticity later in the course using **robust standard errors**.

4.8.6 Summary: Which Assumptions Matter for What?

Assumptions 1–4 (linearity in parameters, random sampling, variation in x, and zero conditional mean) are all required for OLS to be **unbiased**—that is, for the estimator to hit the true parameter values on average. If any one of these assumptions fails, OLS may systematically over- or under-estimate the true β_0 and β_1.

Assumption 5 (homoskedasticity) is *not* needed for unbiasedness. Even if this fails, OLS still gets the right answer on average. However, homoskedasticity *is* required for an additional property:

BLUE

If all five Gauss-Markov assumptions hold, OLS is the **Best Linear Unbiased Estimator (BLUE)**. The Gauss–Markov theorem tells us that when all five assumptions hold, OLS has the smallest variance among *all* estimators that are (a) linear functions of the outcome variable y and (b) unbiased. In other words, no other linear unbiased estimator can produce more precise estimates than OLS. When assumption 5 fails, OLS is still unbiased, but it is no longer the most efficient—other estimators could do better.

4.9 Unbiasedness of OLS

4.9.1 Population Parameters vs. Sample Estimates

Before discussing unbiasedness, recall an important distinction we introduced in Appendix B: the difference between the **population** and the **sample**.

The **population** represents the entire group we're interested in studying. In our wage-education example, the population might be all workers in the U.S. economy. The population regression model describes the *true* relationship in this population:

$$y = \beta_0 + \beta_1 x + \mu$$

The parameters β_0 and β_1 are **population parameters**—fixed, unknown constants that describe how x and y are related for everyone in the population. We never observe these true values directly.

Instead, we collect a **sample**—a subset of observations from the population. Using this sample, we compute **estimates** $\hat{\beta}_0$ and $\hat{\beta}_1$. These estimates are our best guesses for the unknown population parameters.

Critically, *if we drew a different sample, we would get different estimates.* Your classmate analyzing a different random sample of workers would compute different values of $\hat{\beta}_0$ and $\hat{\beta}_1$ than you would, even though the population and regression model are the same. This randomness in the estimates is called **sampling variability**.

4.9.2 What Does Unbiasedness Mean?

Given that estimates vary from sample to sample, how do we know if our specific estimation is any good? One desirable property is **unbiasedness**.

An estimator $\hat{\theta}$ is **unbiased** if its expected value equals the true population parameter:

$$E[\hat{\theta}] = \theta$$

Unbiasedness does *not* mean that any particular estimate exactly equals the truth. Rather, it means that if we could:

1. Draw a random sample from the population
2. Compute $\hat{\beta}_1$ from that sample
3. Repeat steps 1-2 many, many times
4. Average all the $\hat{\beta}_1$ values we computed

…then that average would equal the true β_1.

In other words, our estimation procedure doesn't systematically overshoot or undershoot the target. Individual estimates might be too high or too low, but on average, we get it right.

4.9.3 Proof Sketch

We start from the OLS slope formula (Equation 4.3):

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Now substitute the true population model $y_i = \beta_0 + \beta_1 x_i + \mu_i$ into the numerator. Since $\bar{y} = \beta_0 + \beta_1 \bar{x} + \bar{\mu}$, we get $y_i - \bar{y} = \beta_1(x_i - \bar{x}) + (\mu_i - \bar{\mu})$. Plugging this in:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})[\beta_1(x_i - \bar{x}) + (\mu_i - \bar{\mu})]}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Distributing the numerator and splitting the fraction:

$$\hat{\beta}_1 = \beta_1 \frac{\sum(x_i - \bar{x})^2}{\sum(x_i - \bar{x})^2} + \frac{\sum(x_i - \bar{x})(\mu_i - \bar{\mu})}{\sum(x_i - \bar{x})^2}$$

The first fraction equals 1, so:

$$\hat{\beta}_1 = \beta_1 + \frac{\sum_{i=1}^{n}(x_i - \bar{x})\mu_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.4}$$

where the $\bar{\mu}$ term drops out because $\sum(x_i - \bar{x}) = 0$ (Result 1 from Appendix B).

This is a powerful decomposition. It says: **our estimate = the truth + estimation error**. The estimation error depends on how the unobserved errors μ_i happen to line up with the x_i values in our particular sample.

Now take expectations. Treating the x_i as fixed (Assumption 2), the only random part is the μ_i terms. Under the zero conditional mean assumption ($E[\mu_i|x_i] = 0$):

$$E[\hat{\beta}_1] = \beta_1 + \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \cdot 0}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \beta_1$$

Thus, OLS is unbiased when the Gauss-Markov assumptions 1–4 hold.

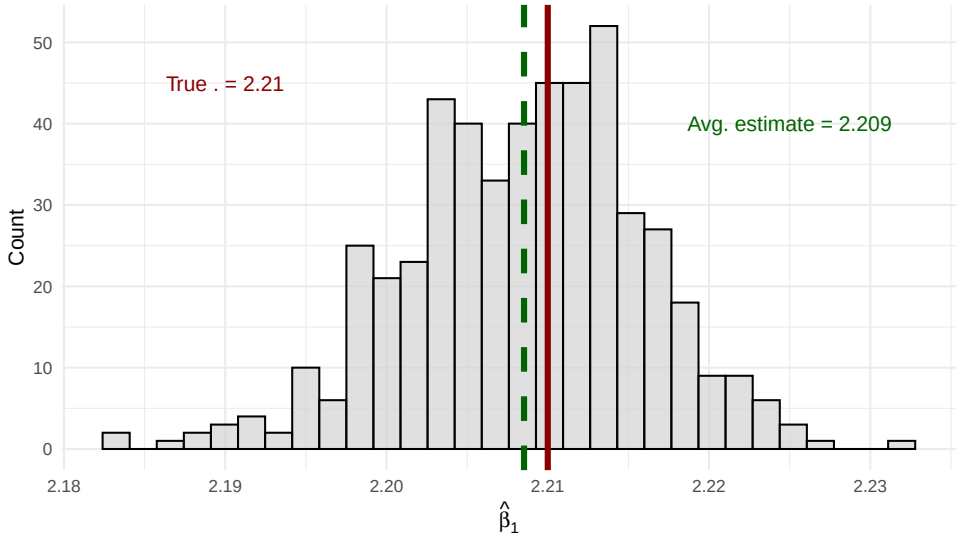
4.9.4 Simulation: Unbiasedness in Action

Show Code: Unbiasedness Simulation

```
# Create a "population" with known parameters
set.seed(1248)
n_pop <- 1000
x_pop <- sample(1:100, size = n_pop, replace = TRUE)
err_pop <- rnorm(n_pop)

# True model: y = 3.5 + 2.21*x + error
pop_data <- tibble(
  x = x_pop,
  y = 3.5 + 2.21 * x + err_pop
)

# Draw many samples and estimate beta_1 each time
set.seed(812476)
n_simulations <- 500
sample_size <- 25

sim_results <- map_dfr(1:n_simulations, function(i) {
  # Draw a random sample
  sample_data <- slice_sample(pop_data, n = sample_size, replace = TRUE)

  # Estimate OLS
  reg <- lm(y ~ x, data = sample_data)

  tibble(
    iteration = i,
    b1_hat = coef(reg)[2]
  )
})

# Calculate the average of all estimates
mean_b1 <- mean(sim_results$b1_hat)
```

The simulation in Figure 4.4 confirms the theory. Each of the 500 samples of size 25 produces a different estimate of β_1—some too high, some too low—because each sample is a different random draw from the population. This spread is **sampling variability**, and it is completely normal. The key thing about unbiasedness is that these estimates are centered on the true value of 2.21: the overestimates and underestimates cancel out on average, so the mean across all 500 estimates lands right on the truth.

Figure 4.4: Distribution of OLS slope estimates across 500 samples. The green line shows the average of all estimates, which equals the true value (2.21), demonstrating unbiasedness.



4.9.5 When Can We Interpret $\hat{\beta}_1$ Causally?

Consider our wage regression:

$$wage = \beta_0 + \beta_1 \cdot educ + \mu$$

When $E[\mu|educ] = 0$, our OLS estimate $\hat{\beta}_1$ is unbiased for β_1. But what does β_1 actually represent?

If the zero conditional mean assumption holds, then β_1 represents the **causal effect** of education on wages: how much wages would change if we could take a person and give them one more year of education, holding everything else constant. In this case, $\hat{\beta}_1$ is an unbiased estimate of this causal effect.

But if $E[\mu|educ] \neq 0$—say, because high-ability people get more education—then β_1 no longer represents a pure causal effect. It conflates the effect of education with the effect of ability. Our estimate $\hat{\beta}_1$ is biased, and we cannot interpret it as the causal effect of education. In other words, our estimate $\hat{\beta}_1$ reflects some **selection bias**, in that there is some systemic differences groups at different values of x.

4.9.6 Correlation vs. Causation, Revisited

If our estimate $\hat{\beta}_1$ can't be interpreted causally, what is it?

Recall that the OLS slope can be written as:

$$\hat{\beta}_1 = \rho_{xy}\frac{\sigma_y}{\sigma_x}$$

Bivariate OLS is fundamentally measuring the *correlation* between x and y, scaled by their relative standard deviations. Correlation becomes causation **only when the Gauss-Markov assumptions hold**—especially the zero conditional mean assumption.

This is why we emphasize these assumptions so heavily. Without them:

- OLS still gives you numbers
- Those numbers still describe the correlation in your data
- But you **cannot** interpret those numbers as causal effects

4.10 Variance of the OLS Estimator

As discussed above, our OLS estimates will always–biased or unbiased–vary from sample to sample. The **variance** of $\hat{\beta}_1$ tells us how much variation to expect amoung our different estimates.

In this section we derive the formula step by step.

4.10.1 Starting Point: Decomposing the Estimator

Recall from the unbiasedness proof that we can write:

$$\hat{\beta}_1 = \beta_1 + \frac{\sum_{i=1}^{n}(x_i - \bar{x})\mu_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Since β_1 is a constant, the variance of $\hat{\beta}_1$ comes entirely from the second term:

$$Var(\hat{\beta}_1) = Var\left(\frac{\sum_{i=1}^{n}(x_i - \bar{x})\mu_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)$$

4.10.2 Treating the Denominator as a Constant

Under Assumptions 2 and 3 (random sampling and variation in x), we treat the x_i values as fixed (or condition on them). This means $\sum(x_i - \bar{x})^2 = SST_x$ is a constant. Using the Variance Property $Var(cX) = c^2 Var(X)$, we can pull this term out of the variance:

$$Var(\hat{\beta}_1) = \frac{1}{SST_x^2} \cdot Var\left(\sum_{i=1}^{n}(x_i - \bar{x})\mu_i\right).$$

Similarly, we can also pull the $\sum(x_i - \bar{x})\mu_i$ term out of the expression using the same property:

$$Var(\hat{\beta}_1) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{SST_x^2} \cdot Var(\mu_i) = \frac{SST_x}{SST_x^2} \cdot Var(\mu_i) = \frac{1}{SST_x} \cdot Var(\mu_i).$$

4.10.3 Applying the Homoskedasticity Assumption

Under Assumption 5 (homoskedasticity), the error variance is the same for every observation: $Var(\mu_i) = \sigma^2$ for all i. This lets us factor σ^2 out of the variance term:

$$Var(\hat{\beta}_1) = \frac{1}{SST_x} \cdot Var(\mu_i) = \frac{1}{SST_x} \cdot \sigma^2$$

4.10.4 Putting It All Together

Thus, we get our expression for the variance of the OLS estimate of our population parameter:

$$Var(\hat{\beta}_1) = \frac{\sigma^2}{SST_x}$$

where $\sigma^2 = Var(\mu)$ is the error variance and $SST_x = \sum(x_i - \bar{x})^2$ is the total variation in x.

4.10.5 What Does This Tell Us?

This formula reveals three important things about the precision of our estimates:

1. **More noise (σ^2) means more variance:** If there's a lot of unexplained variation in y, our estimates are less precise. Unfortunately, we typically can't control this.
2. **More variation in x means less variance:** Spreading out our observations of x gives us more information about the relationship and more precise estimates.
3. **Larger samples help:** More observations mean more total variation in x (larger SST_x), which pushes the variance down. This is one reason why larger samples are better.

> **Which Assumptions Did We Use?**
>
> Notice that the variance formula requires *all five* Gauss-Markov assumptions: Assumptions 1–4 for unbiasedness (so the decomposition in Step 1 holds), and Assumption 5 (homoskedasticity) to factor σ^2 out of the sum. This is why the variance formula σ^2/SST_x is only valid under homoskedasticity.

4.11 Standard Errors

The **standard error** of $\hat{\beta}_1$ is the square root of the variance:

$$se(\hat{\beta}_1) = \frac{\hat{\sigma}}{\sqrt{SST_x}}$$

where $\hat{\sigma} = \sqrt{\frac{SSR}{n-2}}$ is the estimated standard deviation of the errors (more about this in Chapter 5).

Standard errors quantify the precision of our estimates and are essential for hypothesis testing and confidence intervals (which we'll cover in Chapter 6).

```
# R reports standard errors automatically
summary(lm_mpg)
```

```
Call:
lm(formula = mpg ~ hp, data = mtcars)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-5.7121 -2.1122 -0.8854  1.5819  8.2360

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
hp          -0.06823    0.01012  -6.742 1.79e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024,    Adjusted R-squared:  0.5892
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

The standard error of $\widehat{\beta}_1$ is 0.01, shown in the `Std. Error` column.

4.12 Summary and Conclusion

This chapter introduced Ordinary Least Squares regression, the foundational tool of econometrics. We started with the goal of modeling how the conditional expectation $E[Y|X]$ varies with x, and showed how the linear regression model $y = \beta_0 + \beta_1 x + \mu$ provides a framework for this.

We derived the OLS estimator by minimizing the sum of squared residuals, learned to compute regressions in R, and explored the properties of the OLS regression line. We then turned to the statistical properties of OLS, introducing the Gauss-Markov assumptions and proving that OLS is unbiased when these assumptions hold. When the zero conditional mean assumption fails—typically due to omitted variables—our estimates become biased.

Crucially, we connected unbiasedness to causality: the zero conditional mean assumption is what allows us to interpret OLS coefficients as causal effects rather than mere correlations. Without this assumption, OLS still produces numbers, but those numbers cannot be given a causal interpretation.

Finally, we discussed the variance of OLS estimates and the conditions under which OLS is the best linear unbiased estimator.

4.12.1 Key Takeaways

- **The goal** is to estimate the conditional expectation $E[Y|X] = \beta_0 + \beta_1 x$

- **The OLS slope** equals the covariance of x and y divided by the variance of x:

$$\hat{\beta}_1 = \frac{Cov(x, y)}{Var(x)}$$

- **Bivariate OLS is essentially correlation**—be very careful about causal interpretation
- **The Gauss-Markov assumptions** (especially zero conditional mean) are required for unbiasedness
- **Causality requires unbiasedness**—only when $E[\mu|x] = 0$ can we interpret $\hat{\beta}_1$ as a causal effect
- **OLS is BLUE** under the Gauss-Markov assumptions—best among linear unbiased estimators

4.13 Check Your Understanding

4.13.1 Practice: Interpreting Regression Output

The following questions present regression results for you to interpret.

Show Explanation

i. The slope coefficient β_1 is the marginal effect—it tells us how much y changes when x increases by one unit. The intercept β_0 gives the predicted value when $x = 0$.

ii. The zero conditional mean assumption $E[\mu|x] = 0$ states that the expected value of the error term is zero for any given value of x. This is crucial for unbiasedness.

iii. The OLS slope equals the sample covariance of x and y divided by the sample variance of x. This follows directly from the derivation.

iv. Unbiasedness means that if we could repeat the sampling process many times, the *average* of our estimates would equal the true parameter. Individual estimates can still miss the target.

v. Omitted variable bias occurs when we leave out a variable that belongs in the model and is correlated with our included independent variable. This causes $E[\mu|x] \neq 0$.

vi. The zero conditional mean assumption ($E[\mu|x] = 0$) is needed to prove OLS is unbiased, but it is not needed to derive the OLS formulas themselves. The calculus minimization approach works regardless of this assumption—it just tells us which values minimize squared residuals.

vii. As sample size increases, we get more variation in x, so $SST = \sum(x_i - \bar{x})^2$ increases. Since this appears in the denominator, the variance of our estimate decreases. Larger samples give more precise estimates.

Regression Interpretation Questions:

viii. The coefficient on `str` is -2.28, meaning each additional student per teacher is associated with 2.28 fewer points. For 5 additional students: $5 \times (-2.28) = -11.4$ points.

ix. Using the regression equation: $\hat{y} = 698.93 + (-2.28)(20) = 698.93 - 45.6 = 653.33$

x. R-squared measures the fraction of variance in y explained by the regression. Here, 5.1% of test score variation is explained by class size. A low R-squared doesn't make the coefficient meaningless—it just means other factors also matter.

xi. The coefficient is -1.04, so a 10-cent increase predicts: $10 \times (-1.04) = -10.4$ packs per capita.

xii. The negative relationship between price and quantity is exactly what economic theory predicts—the law of demand.

xiii. States with strong anti-smoking policies might have both higher cigarette taxes (raising prices) AND other policies that reduce smoking. This would create omitted variable bias, making price appear more effective than it actually is at reducing consumption.

::::

Note: This section contains interactive content only available in the HTML version.

5 Multivariate Regression

Key Questions

- Why do we need multivariate regression?
- How do we interpret coefficients when there are multiple independent variables?
- What is the zero conditional mean assumption in the multivariate case?
- How does multivariate OLS function as a "matching" estimator?
- What is the omitted variable bias formula and how do we use it?
- What is the bias-variance tradeoff when deciding which controls to include?

Suggested Readings

- Wooldridge (2019), Ch. 3

In Chapter 4, we introduced bivariate OLS and saw that causal interpretation requires the zero conditional mean assumption: $E[\mu|x] = 0$. We also saw that this assumption is *very* difficult to justify.

This chapter introduces **multivariate OLS**, which allows us to explicitly control for confounding variables. By including additional variables on the right-hand side of our regression, we can make the zero conditional mean assumption more plausible and get closer to causal estimates.

5.1 The Problem with Bivariate OLS

Let's revisit why bivariate OLS often fails to give us causal estimates.

Consider a regression of wages on education:

$$wage = \beta_0 + \beta_1 \cdot educ + \mu$$

For $\hat{\beta}_1$ to be an unbiased estimate of the causal effect of education, we need $E[\mu|educ] = 0$. But the error term μ contains everything else that affects wages: ability, family connections, motivation, and countless other factors.

Many of these factors are likely correlated with education. For example, people from wealthier families tend to get more education (better schools, tutors, college is affordable) *and* earn higher wages (family connections, inherited wealth). If family income is in μ and correlated with education, then $E[\mu|educ] \neq 0$, and our estimate is biased.

> **!** The Core Problem
>
> With bivariate OLS, the zero conditional mean assumption requires that *all* factors affecting y (other than x) be uncorrelated with x. This is almost never plausible in observational data.

5.2 The Multivariate Solution

Multivariate OLS offers a solution: **explicitly control** for variables that might be confounders.

If we're worried that family income confounds the education-wage relationship, we can include it in our population regression model by simply adding additional right-hand side, independent variables:

$$wage = \beta_0 + \beta_1 \cdot educ + \beta_2 \cdot family\_income + \mu$$

Now β_1 represents the effect of education on wages **holding family income constant**. The error term μ no longer contains family income, so that specific source of bias is eliminated.

5.2.1 The General Multivariate Model

If we are worried about additional sources of omitted variable bias (as we probably should be), it is easy to expand our regression model and include even more control variables.

The general multivariate linear regression model with k independent variables is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \mu \tag{5.1}$$

where:

- y is the dependent variable
- $x_1, x_2, \ldots, x_k$ are the independent/control variables
- β_0 is the intercept
- β_j is the coefficient on x_j (for $j = 1, 2, \ldots, k$)
- μ is the error term containing all factors not included in the model

5.2.2 Interpreting Coefficients

In multivariate regression, each coefficient has a **ceteris paribus** interpretation:

$$\beta_j = \frac{\partial y}{\partial x_j}$$

This is the change in y associated with a one-unit increase in x_j, **holding all other independent variables constant**.

For example, in the model:

$$wage = \beta_0 + \beta_1 \cdot educ + \beta_2 \cdot family\_income + \mu$$

- β_1: The change in wages for one additional year of education, **holding family income constant**
- β_2: The change in wages for one additional dollar of family income, **holding education constant**

5.2.3 The Zero Conditional Mean Assumption

In multivariate OLS, the zero conditional mean assumption becomes:

$$E[\mu|x_1, x_2, \dots, x_k] = 0$$

This states that the average of unobserved factors is zero for *all unique combinations* of the independent variables. In other words, everything in μ must be uncorrelated with *all* the variables we've included.

This assumption is much more plausible than its bivariate counterpart. By including relevant control variables, we've removed them from μ, making it more likely that what remains is uncorrelated with our variables of interest.

5.3 Deriving the Multivariate OLS Estimator

The derivation follows the same logic as bivariate OLS: we minimize the sum of squared residuals.

5.3.1 The Minimization Problem

Given a sample of n observations, we want to find values $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k$ that minimize:

$$SSR = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \hat{\beta}_2 x_{2i} - \cdots - \hat{\beta}_k x_{ki})^2$$

5.3.2 First-Order Conditions

Taking partial derivatives with respect to each $\hat{\beta}_j$ and setting them equal to zero yields $k+1$ first-order conditions:

$$\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \cdots - \hat{\beta}_k x_{ki}) = 0$$

$$\sum_{i=1}^{n} x_{1i}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \cdots - \hat{\beta}_k x_{ki}) = 0$$

$$\vdots$$

$$\sum_{i=1}^{n} x_{ki}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \cdots - \hat{\beta}_k x_{ki}) = 0$$

Solving this system of equations yields the OLS estimates. The algebra is tedious (and typically handled with matrix methods), so we won't derive the explicit formulas here. Fortunately, your computer does this effortlessly.

5.4 Estimating Multivariate OLS in R

In R, we estimate multivariate regression using the same `lm()` function, simply adding variables with the `+` symbol.

5.4.1 Example: MPG, Horsepower, and Weight

Let's estimate the effect of horsepower on fuel efficiency, first without and then with a control for vehicle weight.

```
# Load data
data(mtcars)

# Bivariate regression
lm_bivariate <- lm(mpg ~ hp, data = mtcars)

# Multivariate regression (adding weight as control)
lm_multivariate <- lm(mpg ~ hp + wt, data = mtcars)
```

Bivariate results:

```
summary(lm_bivariate)
```

```
Call:
lm(formula = mpg ~ hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7121 -2.1122 -0.8854  1.5819  8.2360

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
hp          -0.06823    0.01012  -6.742 1.79e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6024,    Adjusted R-squared:  0.5892
F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

Multivariate results:

```
summary(lm_multivariate)
```

```
Call:
lm(formula = mpg ~ hp + wt, data = mtcars)

Residuals:
   Min     1Q Median     3Q    Max
-3.941 -1.600 -0.182  1.050  5.854

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 **
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.593 on 29 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8148
F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

5.4.2 Interpreting the Results

In the **bivariate** regression, the coefficient on horsepower is -0.068: each additional unit of horsepower is associated with 0.068 fewer miles per gallon.

In the **multivariate** regression (controlling for weight), the coefficient on horsepower drops to -0.032—less than half the size!

What happened? Horsepower and weight are positively correlated (more powerful engines tend to be in heavier vehicles), and weight independently reduces fuel efficiency. The bivariate estimate was conflating the effect of horsepower with the effect of weight. Once we control for weight, we isolate the effect of horsepower alone.

> **What Changed?**
>
> The coefficient on `hp` dropped from -0.068 to -0.032 when we added `wt`. This suggests that much of what appeared to be the "effect" of horsepower was actually the effect of weight. This is precisely what controlling for confounders accomplishes.

5.5 Multivariate OLS as Matching

What is actually happening "under the hood" when we say we want to "control" for a given variable?

There's an intuitive way to understand what multivariate OLS does: it's essentially a **matching** estimator. When we "control for" a variable, we're comparing observations (i.e., values of x_1) that have similar values of a control variable (ie., x_2).

5.5.1 The Returns to Education Example

Suppose we want to estimate the effect of education on wages. A simple comparison of average wages across education levels might be misleading because people with more education tend to work in higher-paying industries. Part of the apparent "return to education" might actually be an "industry premium."

Let's work through this with simulated data to see exactly how controlling for industry works.

Specifically, let's create an imaginary population where:

1. Individuals have either 12. 14, or 16 years of education
2. Those with 12, 14, and 16 years of education have a 0.20, 0.40, and 0.65 chance of working in a high-paying industry
3. The "true" population regression model is:

$$wage = 25000 + 3000 \cdot \text{education} + 15000 \cdot \text{professional industry}$$

So, the "true" effect of an additional year of education on wages is thus 3000.

```
# Simulate wage data
set.seed(12345)
n <- 5000

# Create data where:
# - More educated workers are more likely to be in high-paying industries
# - Industry independently affects wages
wage_data <- tibble(
  # Education: 1 = HS, 2 = Some College, 3 = Bachelor's+
  educ_years = sample(c(12, 14, 16), n, replace = TRUE, prob = c(0.4, 0.3, 0.3)),

  # Higher education -> more likely in professional services (high pay industry)
  ind_professional = rbinom(n, 1, prob = case_when(
```

```
    educ_years == 12 ~ 0.20,
    educ_years == 14 ~ 0.40,
    educ_years == 16 ~ 0.65
  )),

  # Wages depend on education AND industry
  wage = 25000 +
    3000 * (educ_years - 12) +     # True education effect: $3000 per year
    15000 * ind_professional +      # Industry premium: $15000
    rnorm(n, 0, 8000)
) |>
  mutate(
    educ_cat = case_when(
      educ_years == 12 ~ "HS Only",
      educ_years == 14 ~ "Some College",
      educ_years == 16 ~ "Bachelor's+"
    ),
    industry = ifelse(ind_professional == 1, "Professional", "Other")
  )
```

5.5.2 The Naive Comparison

First, let's compute average wages by education level without any controls:

```
naive_means <- wage_data |>
  group_by(educ_cat, educ_years) |>
  summarise(avg_wage = mean(wage), n = n(), .groups = "drop") |>
  arrange(educ_years)

naive_means |>
  knitr::kable(
    col.names = c("Education", "Years", "Avg Wage ($)", "N"),
    digits = 0
  )
```

Education	Years	Avg Wage ($)	N
HS Only	12	28060	1994
Some College	14	36499	1508
Bachelor's+	16	47005	1498

The naive comparison suggests that going from high school to some college is associated with about $8,439 higher wages, and going from high school to a bachelor's degree is associated with about $18,945 higher wages (i.e., the difference between average wages for those with only 12 years of education vs. 14 years, and so on).

But wait! The true effect of education in our simulation is only $3,000 per year (so $6,000 for 2 extra years and $12,000 for 4 extra years). The naive estimates are too large!

We can also see this in the results if we run a simple bivariate regression:

```
naive_ols <- lm(wage ~ educ_years, data = wage_data)

summary(naive_ols)
```

```
Call:
lm(formula = wage ~ educ_years, data = wage_data)

Residuals:
   Min     1Q Median     3Q    Max
-31072  -7558   -354   7555  34592

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -28775.40    1250.54  -23.01   <2e-16 ***
educ_years    4713.69      89.96   52.40   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10560 on 4998 degrees of freedom
Multiple R-squared:  0.3546,    Adjusted R-squared:  0.3544
F-statistic:  2745 on 1 and 4998 DF,  p-value: < 2.2e-16
```

The coefficient on `educ_years` is way bigger than the true effect!

This is because more educated workers are more likely to be in high-paying industries. By failing to account that some industries just pay higher than others, our simple

5.5.3 Step 1: Compute Within-Industry Differences

To control for industry, we first compute average wages separately for each education-industry combination:

```r
cell_means <- wage_data |>
  group_by(industry, educ_cat, educ_years) |>
  summarise(avg_wage = mean(wage), n = n(), .groups = "drop") |>
  arrange(industry, educ_years)

cell_means |>
  select(industry, educ_cat, avg_wage, n) |>
  knitr::kable(
    col.names = c("Industry", "Education", "Avg Wage ($)", "N"),
    digits = 0
  )
```

Table 5.2: Average wages by education and industry

Industry	Education	Avg Wage ($)	N
Other	HS Only	24925	1583
Other	Some College	30793	939
Other	Bachelor's+	37385	515
Professional	HS Only	40136	411
Professional	Some College	45915	569
Professional	Bachelor's+	52044	983

Now let's compute the wage differences **within** each industry:

```r
within_industry <- cell_means |>
  group_by(industry) |>
  mutate(
    diff_from_hs = avg_wage - avg_wage[educ_years == 12]
  ) |>
  filter(educ_years > 12) |>
  select(industry, educ_cat, diff_from_hs, n)

within_industry |>
  knitr::kable(
    col.names = c("Industry", "Education", "Wage Diff vs HS ($)", "N in Cell"),
    digits = 0
  )
```

Industry	Education	Wage Diff vs HS ($)	N in Cell
Other	Some College	5868	939
Other	Bachelor's+	12461	515
Professional	Some College	5779	569
Professional	Bachelor's+	11908	983

Notice that the within-industry differences are much closer to the true values ($6,000 for Some College, $12,000 for Bachelor's+).

5.5.4 Step 2: Compute Weights

To combine these within-industry estimates into a single number, we take a **weighted average**. The weights are based on the number of observations in each industry:

```
# Total workers in each industry
industry_totals <- wage_data |>
  count(industry) |>
  mutate(weight = n / sum(n))

industry_totals |>
  knitr::kable(
    col.names = c("Industry", "N", "Weight"),
    digits = 3
  )
```

Industry	N	Weight
Other	3037	0.607
Professional	1963	0.393

5.5.5 Step 3: Compute Weighted Average

Now we combine the within-industry differences using these weights. Let's do this for the Bachelor's+ vs HS comparison:

```
# Get the within-industry differences for Bachelor's+
ba_diffs <- within_industry |>
  filter(educ_cat == "Bachelor's+") |>
  left_join(industry_totals, by = "industry")
```

```r
ba_diffs |>
  select(industry, diff_from_hs, weight) |>
  knitr::kable(
    col.names = c("Industry", "Within-Industry Diff ($)", "Weight"),
    digits = c(0, 0, 3)
  )
```

Industry	Within-Industry Diff ($)	Weight
Other	12461	0.607
Professional	11908	0.393

```r
# Weighted average
weighted_avg <- sum(ba_diffs$diff_from_hs * ba_diffs$weight)
```

The weighted average is:

$$(1.2461 \times 10^4 \times 0.607) + (1.1908 \times 10^4 \times 0.393) = 12,244$$

This is much closer to the true effect of $12,000!

5.5.6 The Regression Equivalent

Now let's see what regression gives us:

```r
# Create numeric education variable (years beyond HS - just to get easier to interpret numbe
wage_data <- wage_data |>
  mutate(educ_beyond_hs = educ_years - 12)

# Regression controlling for industry
lm_controlled <- lm(wage ~ educ_beyond_hs + ind_professional, data = wage_data)
summary(lm_controlled)
```

```
Call:
lm(formula = wage ~ educ_beyond_hs + ind_professional, data = wage_data)

Residuals:
     Min       1Q   Median       3Q      Max
```

```
-28099.0  -5366.5    -53.9   5311.7  26941.5

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      24907.19     175.67  141.78   <2e-16 ***
educ_beyond_hs    3042.33      74.46   40.86   <2e-16 ***
ind_professional 15009.59     253.05   59.32   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8088 on 4997 degrees of freedom
Multiple R-squared:  0.6212,    Adjusted R-squared:  0.6211
F-statistic:  4098 on 2 and 4997 DF,  p-value: < 2.2e-16
```

The coefficient on `educ_beyond_hs` is approximately $3,000 per year—almost exactly the true value we built into the simulation!

Compare this to the naive regression without the industry control:

```r
lm_naive <- lm(wage ~ educ_beyond_hs, data = wage_data)
coef(lm_naive)["educ_beyond_hs"]
```

```
educ_beyond_hs
      4713.686
```

The naive estimate (4,714) is inflated because it conflates the education effect with the industry premium.

Multivariate OLS as Matching

When you include control variables, OLS is effectively:

1. **Grouping** observations by similar values of the control variables
2. **Computing** the relationship between x and y within each group
3. **Taking a weighted average** across groups, where weights reflect group sizes

This is why multivariate regression can sometimes be thought of as a "matching" estimator—we're matching people who work in the same industry and then comparing their wages based on education. The same logic applies if we have multiple control variables, although the grouping gets complicated fast.

5.6 Goodness-of-Fit: Adjusted R-Squared

In bivariate OLS, we used R^2 to measure how much variation in y our model explains. The same concept applies to multivariate OLS, but with an important caveat.

5.6.1 The Problem with R-Squared

The R^2 can never decrease when you add more variables—it can only increase or stay the same. This is because adding variables can only improve (or maintain) the fit, even if those variables are irrelevant.

This creates a problem: R^2 mechanically increases as you add more variables, potentially making a bloated model look better than a parsimonious one.

5.6.2 The Adjusted R-Squared

The **adjusted R-squared** penalizes for the number of parameters:

$$\bar{R}^2 = 1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}$$

where n is the sample size and k is the number of independent variables.

The adjusted R^2:

- Is always smaller than the regular R^2
- Can decrease when you add irrelevant variables
- Provides a better comparison between models with different numbers of variables

```
# Compare R-squared and Adjusted R-squared
c(
  "Bivariate R-sq" = summary(lm_bivariate)$r.squared,
  "Bivariate Adj R-sq" = summary(lm_bivariate)$adj.r.squared,
  "Multivariate R-sq" = summary(lm_multivariate)$r.squared,
  "Multivariate Adj R-sq" = summary(lm_multivariate)$adj.r.squared
)
```

```
       Bivariate R-sq    Bivariate Adj R-sq     Multivariate R-sq
            0.6024373             0.5891853             0.8267855
Multivariate Adj R-sq
            0.8148396
```

5.7 Gauss-Markov Assumptions for Multivariate OLS

The Gauss-Markov assumptions extend naturally to the multivariate case, with one important addition.

5.7.1 Assumption 1: Linear in Parameters

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \mu$$

As in the bivariate case, we can transform variables to model non-linear relationships while maintaining linearity in parameters.

5.7.2 Assumption 2: Random Sampling

We have a random sample $\{(x_{1i}, x_{2i}, \ldots, x_{ki}, y_i) : i = 1, \ldots, n\}$ from the population.

5.7.3 Assumption 3: No Perfect Multicollinearity

This assumption has two parts:

1. **No constant variables:** Each independent variable must have some variation in the sample
2. **No exact linear relationships:** No independent variable can be a perfect linear function of other independent variables

The second part is new and important in multivariate regression.

⚠ Perfect Multicollinearity

Perfect multicollinearity occurs when one independent variable is an exact linear combination of others. Examples:

- Including both `family_income` and `family_income_thousands` (one is just the other divided by 1000)
- Including `parent1_income`, `parent2_income`, and `family_income` when `family_income = parent1_income + parent2_income`
- Including dummy variables for all categories of a categorical variable plus an intercept

When perfect multicollinearity exists, the OLS formulas break down (the system of equations has no unique solution). R will typically drop one of the collinear variables automatically and issue a warning.

5.7.4 Assumption 4: Zero Conditional Mean

$$E[\mu|x_1, x_2, \dots, x_k] = 0$$

The error term has mean zero for all combinations of the independent variables. This can still fail due to:

- Omitted variables correlated with included variables
- Wrong functional form
- Measurement error in independent variables
- Simultaneous causality

5.7.5 Assumption 5: Homoskedasticity

$$Var(\mu|x_1, x_2, \dots, x_k) = \sigma^2$$

The variance of the error term is constant across all combinations of independent variable values.

5.8 Unbiasedness of Multivariate OLS

Just like with bivariate OLS, if Gauss-Markov assumptions 1–4 hold, then our multivariate OLS estimates are **unbiased:**

$$E[\hat{\beta}_j] = \beta_j, j = 0, 1, ..., k.$$

We can then interpret our estimates as *ceteris paribus* comparisons, or apples-to-apples comparisons *conditional on control variables.*

5.8.1 Observed vs. Unobserved Variables

However, just because we are able to control for lots of variables doesn't mean we have a causal estimate. In many cases, if we have lots of *observed* correlation between control variables and our independent variable of interest, we also probably have lots of **unobserved** correlation between variables.

For example, there could be important components of μ that we simply cannot measure sufficiently–inherent motivation in a wage regression, for example. Labor market surveys generally do not have an effective way to capture this, so this important determinant of wages

is considered *unobserved*, and we thus have cannot include it in our multivariate regression model and we will therefore end up with a biased estimate.

But this is not the end of the world. Throughout the course, we are going to cover some methods which allow us to deal with some of these issues.

5.9 The Omitted Variable Bias Formula

Even when we can't include an omitted variable (perhaps because it's unobservable), we can still reason about the **direction** of the bias.

5.9.1 Setup

Let's say our independent variable of interest is $beta_1$ (e.g., education).

Suppose the true population regression model is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \mu$$

In an ideal world, we would be able to estimate an unbiased model:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x.$$

But we can't observe x_2 (it may be motivation or ability, for example), so instead we estimate the misspecified model:

$$\tilde{y} = \tilde{\beta}_0 + \tilde{\beta}_1 x_1$$

where the tilde ($\tilde{}$) indicates estimates from the misspecified model.

5.9.2 The OVB Formula

It turns out there is a simple relationship between our biased estimate from the misspecified model, $\tilde{\beta}_1$, and our unbiased, correctly specified model $\hat{\beta}_1$:

It can be shown that:

$$\tilde{\beta}_1 = \hat{\beta}_1 + \hat{\beta}_2 \tilde{\delta}_1 \tag{5.2}$$

where $\tilde{\delta}_1$ is the coefficient from regressing x_2 on x_1:

$$x_2 = \tilde{\delta}_0 + \tilde{\delta}_1 x_1 + \text{error}.$$

Taking expectations:

$$E[\tilde{\beta}_1] = \beta_1 + \beta_2 \tilde{\delta}_1$$

Therefore, the **bias** is:

$$\text{Bias}(\tilde{\beta}_1) = \beta_2 \cdot \tilde{\delta}_1 \tag{5.3}$$

5.9.3 Interpreting the Bias Formula

The bias depends on two factors:

1. β_2**:** The effect of the omitted variable on y
2. $\tilde{\delta}_1$**:** The correlation between the omitted variable (x_2) and the included variable (x_1)

If either factor is zero, there is no bias. But if both are non-zero, bias exists, and its direction depends on the signs:

	$Corr(x_1, x_2) > 0$	$Corr(x_1, x_2) < 0$
$\beta_2 > 0$	Positive bias (overestimate)	Negative bias (underestimate)
$\beta_2 < 0$	Negative bias (underestimate)	Positive bias (overestimate)

5.9.4 Example: Education, Wages, and Ability

Consider estimating the return to education while omitting ability:

- β_2 **(effect of ability on wages):** Positive (more able people earn more)
- $\tilde{\delta}_1$ **(correlation of ability with education):** Positive (more able people get more education)

Therefore:

$$\text{Bias} = (+)(+) = \text{Positive}$$

Our estimate of the return to education is **too large**—it includes part of the return to ability.

5.9.5 Example: Policing, Crime, and Poverty

Consider estimating the effect of policing on crime while omitting local poverty:

- **Effect of poverty on crime (β_2):** Positive (higher poverty $\rightarrow$ more crime)
- **Correlation of poverty with policing ($\tilde{\delta}_1$):** Positive (high-poverty areas get more policing)

Therefore:

$$\text{Bias} = (+)(+) = \text{Positive}$$

Our estimate suggests policing *increases* crime more than it actually does (or reduces it less than it actually does). This is why naive correlations between police presence and crime can be misleading.

5.9.6 Example: Study Hours, Course Difficulty, and GPA

Consider estimating the effect of study hours on GPA while omitting course difficulty:

- **Effect of course difficulty on GPA (β_2):** Negative (harder courses $\rightarrow$ lower GPA)
- **Correlation of course difficulty with study hours ($\tilde{\delta}_1$):** Positive (harder courses $\rightarrow$ students study more)

Therefore:

$$\text{Bias} = (-)(+) = \text{Negative}$$

Our estimate understates the true benefit of studying. Students who study the most tend to be taking harder courses that drag down their GPAs, so the naive estimate makes studying look *less* effective than it actually is.

5.10 Variance in Multivariate OLS

Under the Gauss-Markov assumptions, the variance of $\hat{\beta}_j$ is:

$$Var(\hat{\beta}_j) = \frac{\sigma^2}{SST_j(1 - R_j^2)} \tag{5.4}$$

where:

- σ^2 is the error variance
- $SST_j = \sum(x_{ji} - \bar{x}_j)^2$ is the total variation in x_j
- R_j^2 is the R-squared from regressing x_j on all other independent variables

5.10.1 What Increases Variance?

From Equation 5.4, the variance of $\hat{\beta}_j$ increases when:

1. **More noise (σ^2):** Higher error variance means less precise estimates
2. **Less variation in x_j (SST_j):** Less information about how x_j relates to y
3. **Higher R_j^2:** More correlation between x_j and other independent variables

The third point is crucial: when your variable of interest is highly correlated with control variables, your estimate becomes less precise. This is sometimes called **high multicollinearity** or the **multicollinearity problem**.

Note this is different from **perfect multicolinearity**, which makes it *impossible* to get an estimate. Here, we still will be able to estimate our β_j successfully, but it will have a high variance and be "noisy".

5.10.2 Estimating σ^2 from the Data

Look carefully at the variance formula above. The denominator contains quantities we can compute directly from our sample: SST_j (the total variation in x_j) and R_j^2 (how much of x_j is explained by the other regressors). But the numerator contains σ^2—the **variance of the population error term**. This is a population parameter that we cannot observe directly.

So how do we estimate σ^2?

Recall from the homoskedasticity assumption (GM.5) that $Var(\mu_i|\mathbf{x}_i) = \sigma^2$ for all observations i. Combined with the zero conditional mean assumption (GM.4), which gives us $E[\mu_i|\mathbf{x}_i] = 0$, we can show that $\sigma^2 = E[\mu_i^2]$. Let's work through why.

From Appendix B, the definition of variance is $Var(X) = E[(X - \mu)^2]$—the expected squared deviation from the mean. Applying this to μ_i and using the fact that $E[\mu_i] = 0$ from GM.4:

$$Var(\mu_i) = E[(\mu_i - 0)^2] = E[\mu_i^2]$$

And from GM.5, $Var(\mu_i) = \sigma^2$, giving us:

$$\sigma^2 = E[\mu_i^2]$$

In other words, σ^2 is simply the expected value of the squared error terms. If we could observe the true errors μ_i, we'd just compute their average squared value. But we can't observe the true errors—they're the difference between the actual outcome y_i and the **population** regression line, which we don't know.

What we *can* compute are the **residuals**:

$$\hat{\mu}_i = y_i - \hat{y}_i = y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \cdots - \hat{\beta}_k x_{ki}$$

The residuals are the sample analog of the population errors. They measure the difference between the actual outcome and the **estimated** regression line. Since the residuals approximate the true errors, we can use them to estimate σ^2.

It turns out that the average squared residual, adjusted for **degrees of freedom**, is an unbiased estimator of σ^2:

i Estimating σ^2

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} \hat{\mu}_i^2}{n - k - 1} = \frac{SSR}{n - k - 1}$$

where $SSR = \sum_{i=1}^{n} \hat{\mu}_i^2$ is the sum of squared residuals and $n - k - 1$ is the degrees of freedom (sample size minus the number of estimated parameters).

Why do we divide by $n - k - 1$ instead of n? Because we used $k + 1$ estimated coefficients $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_k)$ to compute the residuals. Each estimated coefficient "uses up" one degree of freedom, so we need to correct for this to get an unbiased estimate. This is the same logic behind the degrees-of-freedom correction in the adjusted R^2.

The square root $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$ is sometimes called the **standard error of the regression** (SER) or the **root mean squared error** (RMSE). In R output, this appears as **Residual standard error**.

5.10.3 From $\hat{\sigma}^2$ to the Standard Error

Now we can plug our estimate $\hat{\sigma}^2$ into the variance formula. The **standard error** of $\hat{\beta}_j$ is:

$$se(\hat{\beta}_j) = \frac{\hat{\sigma}}{\sqrt{SST_j(1 - R_j^2)}}$$

The standard error is the estimated standard deviation of the sampling distribution of $\hat{\beta}_j$. It tells us how much our estimate would typically vary across repeated samples. Every time you see $se(\hat{\beta}_j)$ in a regression table, a formula, or R output, this is what's being computed behind the scenes.

> The Big Picture
>
> $$Var(\hat{\beta}_j) = \frac{\sigma^2}{SST_j(1 - R_j^2)} \xrightarrow{\text{estimate } \sigma^2 \text{ with residuals}} se(\hat{\beta}_j) = \frac{\hat{\sigma}}{\sqrt{SST_j(1 - R_j^2)}}$$
>
> We go from a formula involving an unknown population parameter (σ^2) to a fully computable **standard error** by replacing σ^2 with its sample estimate $\hat{\sigma}^2 = SSR/(n-k-1)$. The standard error is the foundation of statistical inference: t-statistics, p-values, and confidence intervals (covered in the next unit) all depend on it.

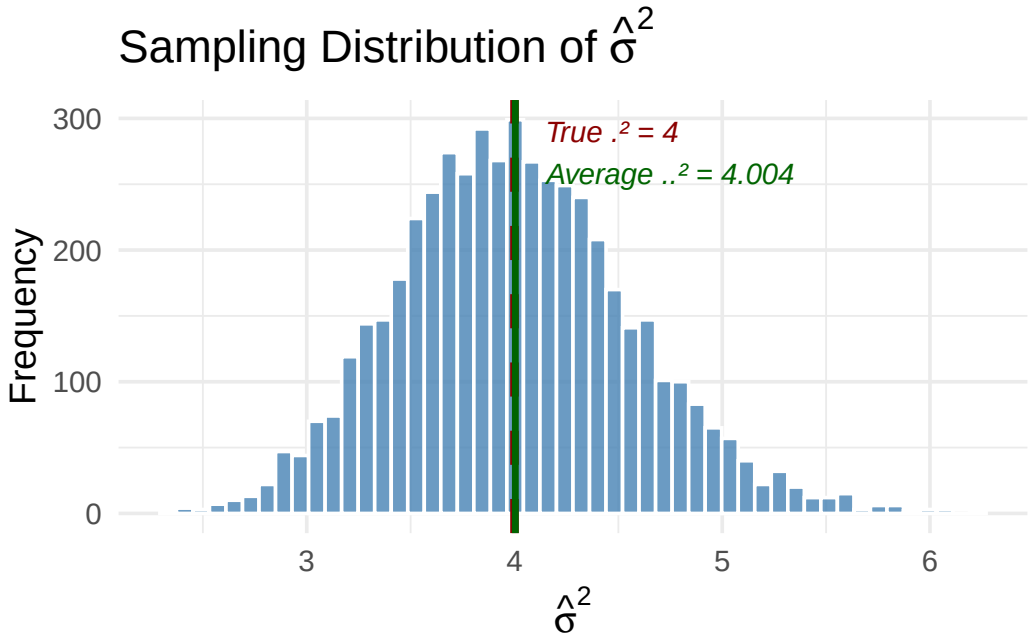
5.10.4 Simulation: Does $\hat{\sigma}^2$ Actually Work?

Let's verify that $\hat{\sigma}^2 = SSR/(n-k-1)$ is a good estimator of σ^2 with a simulation. We'll set up a known population where we *choose* the true σ^2, then check whether our estimator recovers it.

```
set.seed(325)

# --- Population parameters ---
n <- 100          # sample size
true_sigma2 <- 4  # the TRUE error variance ( ² = 4)
n_sims <- 5000    # number of simulations

# --- Run the simulation ---
sigma2_hat <- numeric(n_sims)

for (i in 1:n_sims) {
  # Generate data from a known DGP: y = 2 + 3*x1 + 1*x2 +
```

```r
  x1 <- rnorm(n)
  x2 <- rnorm(n)
  mu <- rnorm(n, mean = 0, sd = sqrt(true_sigma2))  # errors with  ² = 4
  y <- 2 + 3 * x1 + 1 * x2 + mu

  # Estimate the model
  reg <- lm(y ~ x1 + x2)

  # Compute ^² = SSR / (n - k - 1)
  residuals_sq <- residuals(reg)^2
  sigma2_hat[i] <- sum(residuals_sq) / (n - 2 - 1)  # n - k - 1 = 100 - 2 - 1 = 97
}

# --- Visualize the results ---
ggplot(tibble(sigma2_hat), aes(x = sigma2_hat)) +
  geom_histogram(fill = "steelblue", color = "white", bins = 50, alpha = 0.8) +
  geom_vline(xintercept = true_sigma2, color = "darkred", linewidth = 1.5, linetype = "dashed
  geom_vline(xintercept = mean(sigma2_hat), color = "darkgreen", linewidth = 1.2) +
  annotate("text", x = true_sigma2 + 0.15, y = Inf, vjust = 2,
           label = paste0("True  ² = ", true_sigma2),
           color = "darkred", size = 4, hjust = 0, fontface = "italic") +
  annotate("text", x = mean(sigma2_hat) + 0.15, y = Inf, vjust = 4,
           label = paste0("Average ^² = ", round(mean(sigma2_hat), 3)),
           color = "darkgreen", size = 4, hjust = 0, fontface = "italic") +
  labs(x = expression(hat(sigma)^2),
       y = "Frequency",
       title = expression(paste("Sampling Distribution of ", hat(sigma)^2))) +
  theme_minimal(base_size = 14)
```

Figure 5.1: Distribution of $\hat{\sigma}^2$ across 5,000 simulated samples. The estimator is centered on the true $\sigma^2 = 4$, confirming it is unbiased.



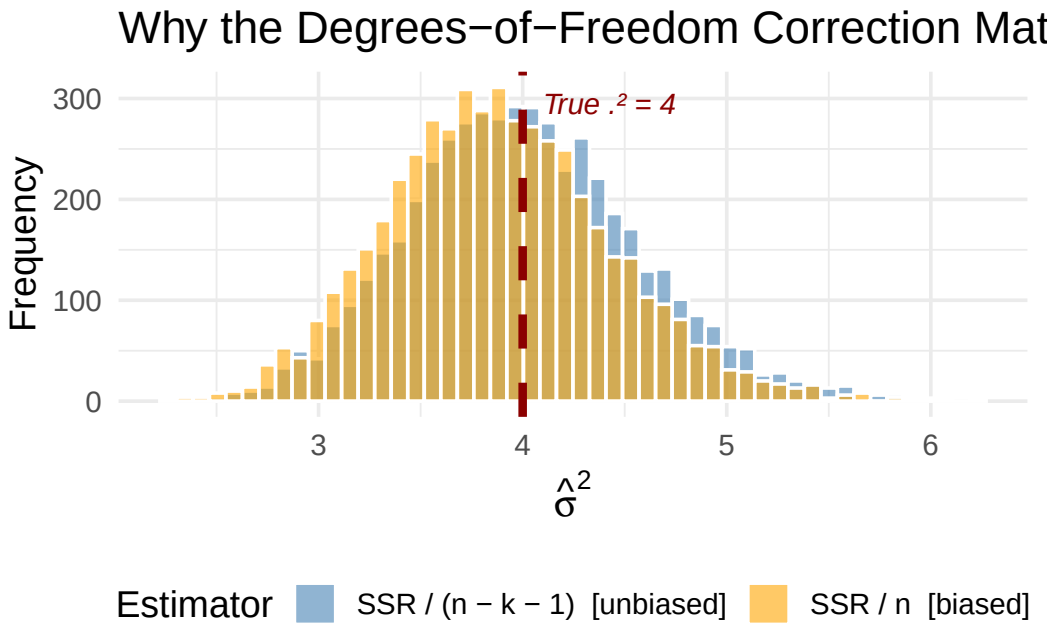
The simulation confirms two important things. First, the average of $\hat{\sigma}^2$ across all 5,000 samples is very close to the true $\sigma^2 = 4$—this is unbiasedness in action. Second, while any *single* estimate of $\hat{\sigma}^2$ can be above or below the true value, the distribution is centered in the right place.

What happens if we divide by n instead of $n - k - 1$? Let's check:

```
# Compute the naive (biased) estimator that divides by n
sigma2_naive <- sigma2_hat * (n - 2 - 1) / n  # convert back: multiply by (n-k-1) then divide

bind_rows(
  tibble(estimate = sigma2_hat, type = "SSR / (n - k - 1)  [unbiased]"),
  tibble(estimate = sigma2_naive, type = "SSR / n  [biased]")
) |>
  ggplot(aes(x = estimate, fill = type)) +
  geom_histogram(alpha = 0.6, bins = 50, color = "white", position = "identity") +
  geom_vline(xintercept = true_sigma2, color = "darkred", linewidth = 1.5, linetype = "dashed
  scale_fill_manual(values = c("steelblue", "orange")) +
  annotate("text", x = true_sigma2 + 0.1, y = Inf, vjust = 2,
           label = paste0("True  ² = ", true_sigma2),
           color = "darkred", size = 4, hjust = 0, fontface = "italic") +
```

```
  labs(x = expression(hat(sigma)^2),
       y = "Frequency",
       fill = "Estimator",
       title = "Why the Degrees-of-Freedom Correction Matters") +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom")
```

Figure 5.2: Comparing the unbiased estimator $SSR/(n-k-1)$ to the naive estimator SSR/n. Dividing by n systematically underestimates σ^2.



The orange distribution (dividing by n) is shifted to the left—it systematically *underestimates* σ^2. The degrees-of-freedom correction fixes this bias. The intuition is that the residuals are computed from the *estimated* regression line, which is already chosen to fit the data as closely as possible. This makes the residuals systematically smaller than the true errors, and dividing by $n-k-1$ instead of n corrects for this.

5.11 The Bias-Variance Tradeoff

Should you include a variable in your regression? The answer depends on a tradeoff.

5.11.1 The Costs of Omitting a Relevant Variable

If a variable truly belongs in the model ($\beta_2 \neq 0$) and is correlated with x_1, omitting it causes **bias**. Your estimate systematically misses the true value.

5.11.2 The Costs of Including an Irrelevant Variable

What if you include a variable that doesn't belong ($\beta_2 = 0$)?

From the OVB formula:

$$E[\hat{\beta}_1] = \beta_1 + (0) \cdot \tilde{\delta}_1 = \beta_1$$

No bias! Including an irrelevant variable doesn't bias your estimate.

But look at the variance formula. Including an extra variable increases R_1^2 (the correlation between x_1 and other variables), which increases the variance of $\hat{\beta}_1$.

Including irrelevant variables inflates variance without reducing bias.

5.11.3 The Tradeoff

Action	Bias	Variance
Omit relevant variable	Increases	Decreases
Include irrelevant variable	No change	Increases

This is the **bias-variance tradeoff**:

- If you're too conservative (omit too many variables), you risk bias
- If you're too liberal (include everything), you inflate variance

5.11.4 Practical Guidance

There's no perfect solution, but some principles help:

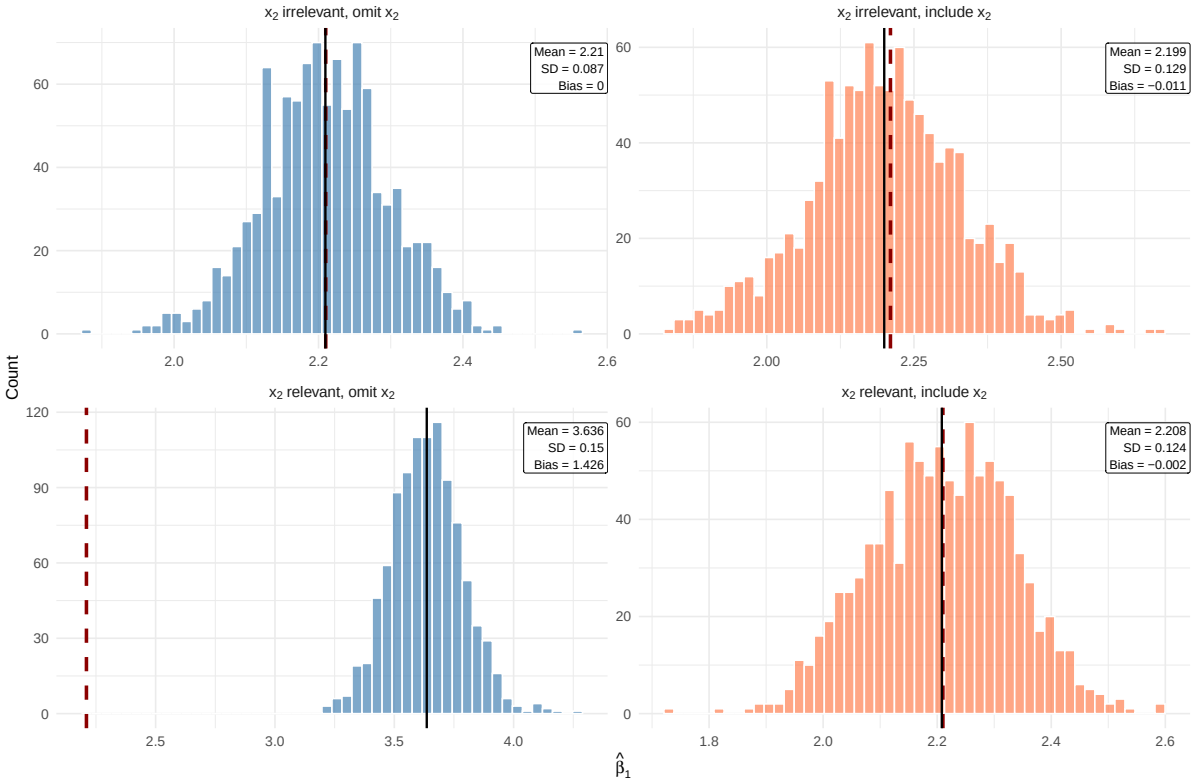
1. **Include variables you have strong theoretical reasons to believe are confounders.** Use economic theory and prior research to guide variable selection.
2. **Be cautious about highly correlated controls.** If two controls are very highly correlated with each other, including both can dramatically inflate variance.
3. **Larger samples help.** As n increases, variance decreases, making the "include it just in case" strategy less costly.

4. **Check how estimates change.** If adding a control substantially changes your coefficient of interest, that control is probably important.

5.12 Simulation: Bias vs. Variance

Let's see the bias-variance tradeoff in action with a simulation. We generate two populations—one where x_2 is irrelevant ($\beta_2 = 0$) and one where x_2 is relevant ($\beta_2 \neq 0$)—then repeatedly estimate $\hat{\beta}_1$ using either the short model (omit x_2) or the long model (include x_2).

Figure 5.3: The bias-variance tradeoff across all four cases. Each panel shows the sampling distribution of $\hat{\beta}_1$ from 1,000 simulations. The dashed red line marks the true $\beta_1 = 2.21$.



Each panel shows one of the four possible scenarios:

- **Case 1** (top-left): x_2 is irrelevant and we omit it. Both bias and variance are small. This is the *best case*—we correctly leave out a variable that doesn't belong.
- **Case 2** (top-right): x_2 is irrelevant but we include it anyway. The estimates are still centered on the true value (no bias), but the distribution is *wider* (higher variance).

Including an irrelevant variable costs us precision for no benefit.

- **Case 3** (bottom-left): x_2 is relevant but we omit it. The distribution shifts away from the true β_1—this is **omitted variable bias**. Even though the variance is low, the estimates are *systematically wrong*.
- **Case 4** (bottom-right): x_2 is relevant and we include it. The distribution is centered on the true value (no bias), though variance is slightly higher than Case 1. This is the *correct* specification—the variance cost is worth paying to eliminate bias.

5.13 Summary

This chapter extended OLS to the multivariate case, allowing us to control for confounding variables.

5.13.1 Key Takeaways

- **Multivariate OLS** includes multiple independent variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \mu$$

- **Coefficients have ceteris paribus interpretations:** β_j is the effect of x_j holding all other variables constant
- **The zero conditional mean assumption** now requires $E[\mu|x_1, \ldots, x_k] = 0$—more plausible with good controls
- **Multivariate OLS is like matching:** It compares observations within groups defined by control variables
- **The OVB formula** tells us the direction of bias from omitting a variable:

$$\text{Bias} = \beta_2 \cdot \tilde{\delta}_1$$

- **There's a bias-variance tradeoff:** Omitting relevant variables causes bias; including irrelevant variables inflates variance
- **Perfect multicollinearity** occurs when one variable is an exact linear function of others; it must be avoided

5.14 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

6 Statistical Inference

Key Questions

- How do we know if our OLS estimate reflects a "real" effect or is just due to random chance?
- What is the sampling distribution of our OLS estimator and why does it matter?
- How do we conduct hypothesis tests using t-statistics and p-values?
- What do confidence intervals tell us about our estimates?
- How do we test multiple hypotheses simultaneously using F-tests?
- How do we read and interpret regression tables in published research?

Suggested Readings

- Wooldridge (2019), Ch. 4

6.1 From Point Estimates to Uncertainty

In the previous chapters, we learned how to estimate the parameters of a regression model using OLS. We discussed how, under certain assumptions, our OLS estimator $\hat{\beta}_j$ is *unbiased*—meaning that on average, across many hypothetical samples, our estimate equals the true population parameter β_j.

But here's the catch: we only ever observe *one* sample. Our single estimate might be too high or too low relative to the truth, simply due to the random variation inherent in sampling. **Statistical inference** is the set of tools that allows us to quantify this uncertainty and make statements about population parameters based on sample data.

The central question of statistical inference is: *Is our single OLS estimate "real," or is it due to random chance?*

6.2 The Sampling Distribution: A Simulation

To understand why uncertainty matters, let's run a simulation. Imagine we have a population where the true relationship between x and y is:

$$y = 2.45 + 0 \cdot x + \mu$$

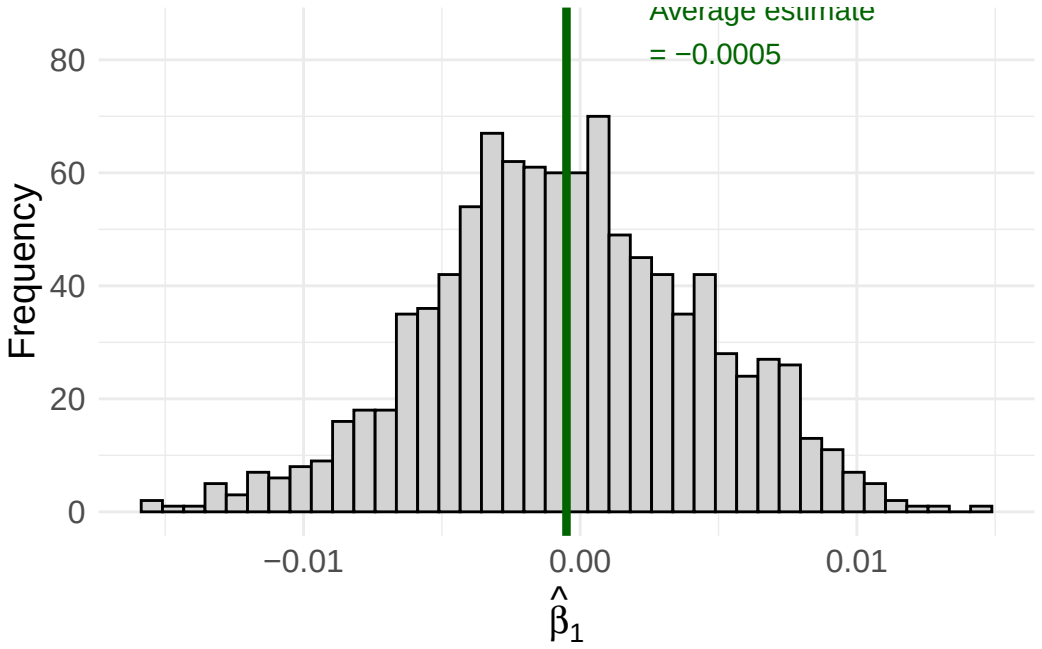
Notice that the true effect of x on y is **exactly zero**. In the population, x has no effect on y whatsoever.

But what happens when we draw samples from this population and estimate OLS regressions? Let's find out.

```
set.seed(124890)

# Create our "population" data
pop_data <- tibble(
  x = sample(1:100, size = 1000, replace = TRUE),
  err = rnorm(1000),
  y = 2.45 + 0 * x + err  # True beta_1 = 0
)

# Number of simulation repetitions
n_sims <- 1000

# Run the simulation
sim_results <- map_dfr(1:n_sims, function(i) {
  # Draw a random sample of 50 observations
  sample_data <- pop_data |>
    slice_sample(n = 50, replace = TRUE)

  # Estimate OLS regression
  reg <- lm(y ~ x, data = sample_data)

  # Store the estimate
  tibble(
    iteration = i,
    b1_hat = coef(reg)["x"]
  )
})

# Calculate the average estimate
```

```r
avg_estimate <- mean(sim_results$b1_hat)

# Create the visualization
ggplot(sim_results, aes(x = b1_hat)) +
  geom_histogram(fill = "lightgrey", color = "black", bins = 40) +
  geom_vline(xintercept = avg_estimate, color = "darkgreen", linewidth = 1.5) +
  annotate("text", x = avg_estimate + 0.003, y = 85,
           label = paste0("Average estimate\n= ", round(avg_estimate, 4)),
           color = "darkgreen", size = 4, hjust = 0) +
  labs(
    x = expression(hat(beta)[1]),
    y = "Frequency"
  ) +
  theme_minimal() +
  theme(
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14)
  )
```

Figure 6.1: Sampling distribution of ˆ when the true = 0. Each bar represents the frequency of estimates across 1,000 different samples.



Look at what happened! Even though the true $\beta_1 = 0$, we got a *distribution* of different

estimates ranging from about -0.02 to +0.02. Some samples produced positive estimates, others produced negative estimates. The average across all samples is very close to zero (as expected from unbiasedness), but any *individual* sample could give us an estimate that deviates from zero.

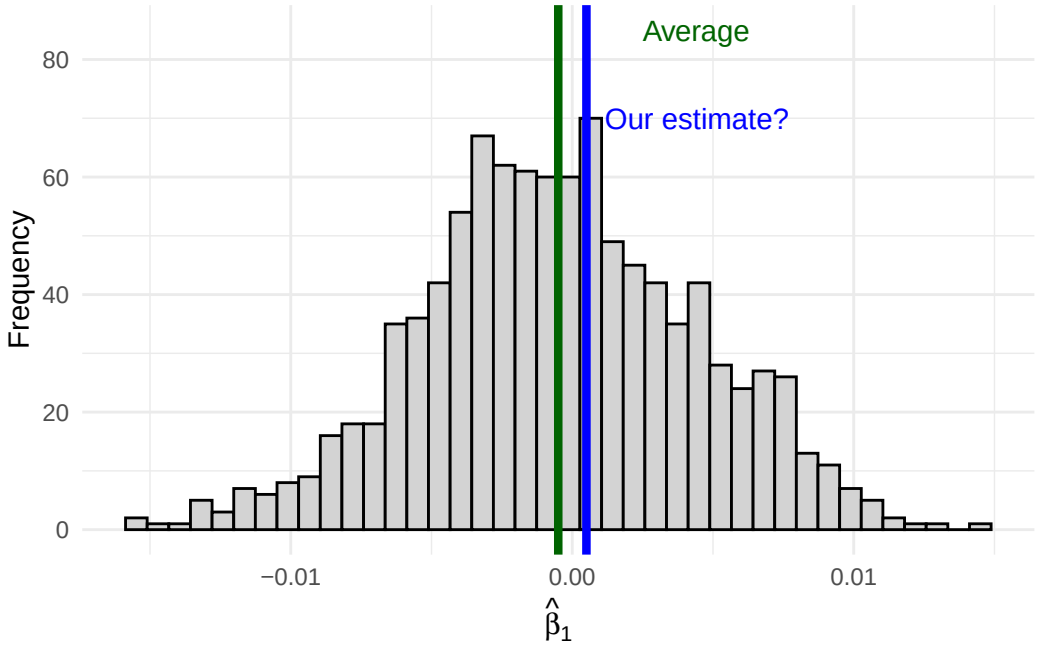
This distribution of estimates across all possible samples is called the **sampling distribution** of $\hat{\beta}_j$.

6.3 Where Does Our Estimate Fall?

Now imagine you only have access to *one* sample—as is always the case in practice. You run your regression and get an estimate of, say, $\hat{\beta}_1 = 0.001$.

How do you know if your estimate comes from here (close to the true value):

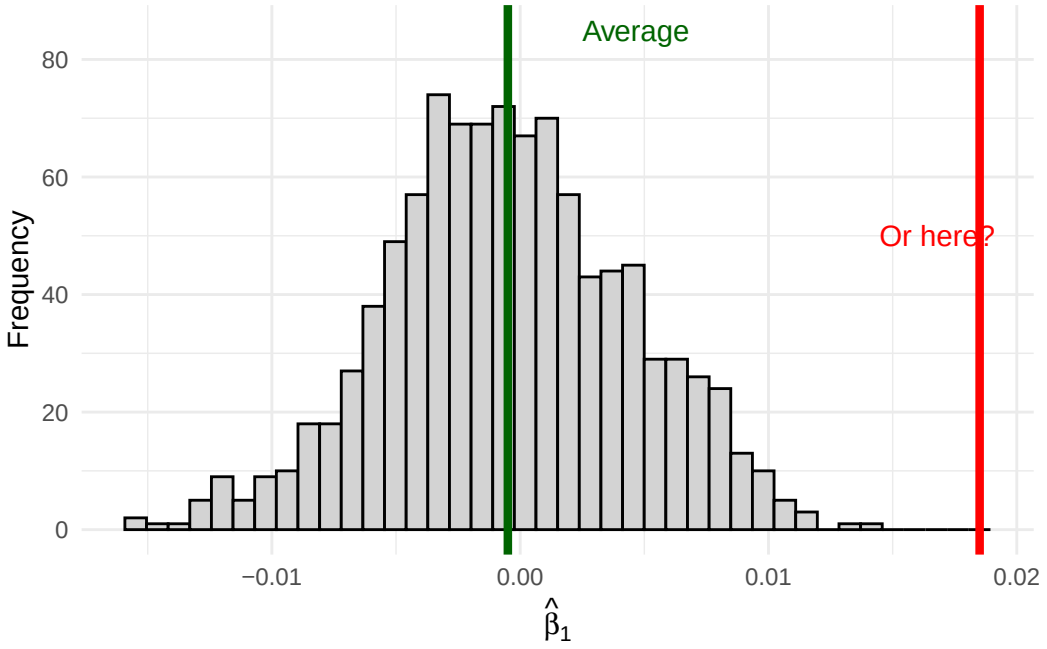
Figure 6.2: An estimate close to the average is likely consistent with the null hypothesis.



Or from here (far out in the tails):

Remember, when you are out in the wild running regressions, you won't actually be able to observe the full distribution of estimates since you have only one sample. Sure, the *unbiasedness* property will tell us that *on average* our estimate will be correct, but that only tells us about the *center of our distribution*, not where any specific estimate lays along the *entire* distribution. If your specific estimate from your one sample falls in the middle of the distribution where $\beta_1 = 0$, it's consistent with the null hypothesis of no effect. But if your estimate falls way out

Figure 6.3: An estimate in the tail is unlikely if the true effect is zero—suggesting the true effect might not be zero.



in the tails, it suggests that maybe the true β_1 isn't zero after all—because getting such an extreme estimate would be very unlikely if β_1 really were zero.

The tools of statistical inference formalize this intuition.

6.4 The Sampling Distribution of $\hat{\beta}_j$

To conduct statistical inference, we need to make a few assumptions. We've already established two key properties of our OLS estimator (under the Gauss-Markov assumptions):

Expected Value (Unbiasedness):

$$E[\hat{\beta}_j] = \beta_j$$

Variance (under homoskedasticity):

$$Var(\hat{\beta}_j) = \frac{\sigma^2}{SST_j(1 - R_j^2)}$$

where SST_j is the total sum of squares of x_j and R_j^2 is the R-squared from regressing x_j on all other independent variables.

But knowing the mean and variance isn't enough—we need to know the *shape* of the distribution. (Recall from the previous chapter that we estimate σ^2 from the residuals using $\hat{\sigma}^2 = SSR/(n-k-1)$, which gives us the computable standard error $se(\hat{\beta}_j) = \hat{\sigma}/\sqrt{SST_j(1-R_j^2)}$.)

6.4.1 The Normality Assumption

To fully characterize the sampling distribution, we make an additional assumption:

> **!** Normality Assumption
>
> The population error μ is independent of the explanatory variables $x_1, x_2, ..., x_k$ and is normally distributed with zero mean and variance σ^2:
>
> $$\mu \sim Normal(0, \sigma^2)$$
>
> Equivalently, we can write:
>
> $$y|\mathbf{x} \sim Normal(\beta_0 + \beta_1 x_1 + ... + \beta_k x_k, \sigma^2)$$

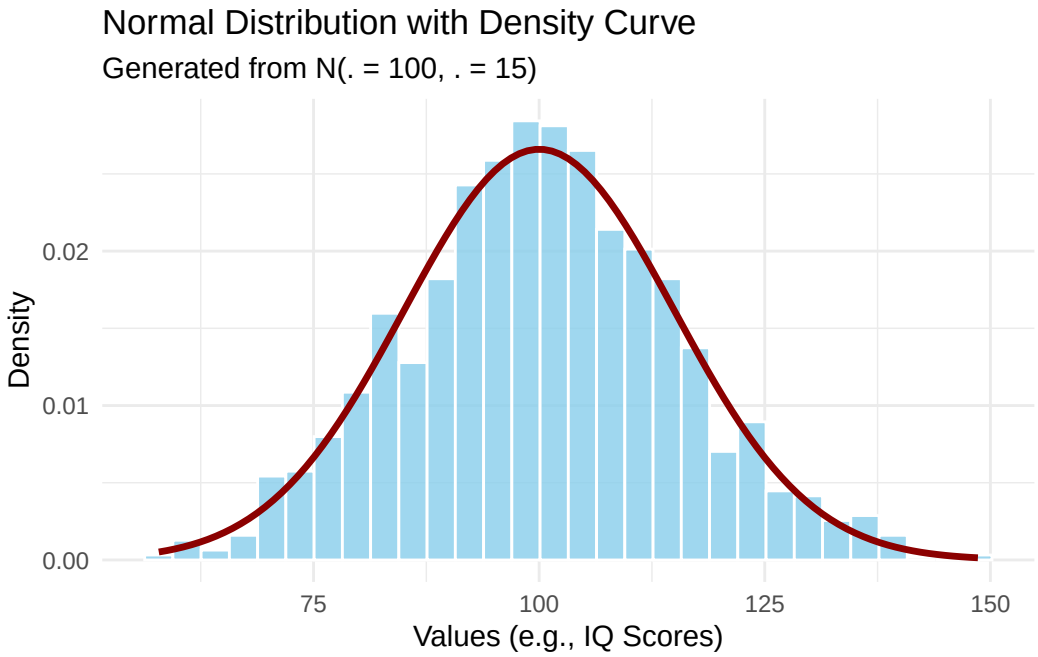
This says that conditional on the independent variables, the outcome y follows a normal distribution centered on the regression line.

Let's visualize what a normal distribution looks like:

```
set.seed(123)
mu <- 100
sigma <- 15

normal_data <- tibble(
  values = rnorm(n = 1000, mean = mu, sd = sigma)
)

ggplot(normal_data, aes(x = values)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 30,
                 fill = "skyblue",
                 color = "white",
                 alpha = 0.8) +
  stat_function(fun = dnorm,
                args = list(mean = mu, sd = sigma),
                color = "darkred",
                linewidth = 1.2) +
```

```
  labs(
    title = "Normal Distribution with Density Curve",
    subtitle = paste0("Generated from N(  = ", mu, ",   = ", sigma, ")"),
    x = "Values (e.g., IQ Scores)",
    y = "Density"
  ) +
  theme_minimal()
```

Figure 6.4: The normal (Gaussian) distribution—the classic 'bell curve.' The example shows IQ scores, which are designed to follow N(100, 15).



At first glance, the normality assumption seems quite strong. For many economic variables—like wages, which are bounded below by zero and have a right skew—the distribution is clearly not normal. However, there's good news: as long as our sample size is large enough, the *sampling distribution of* $\hat{\beta}_j$ will be approximately normal *even if the errors themselves aren't normally distributed.* This result is called **asymptotic normality** and follows from the Central Limit Theorem.

6.4.2 The Normal Sampling Distribution

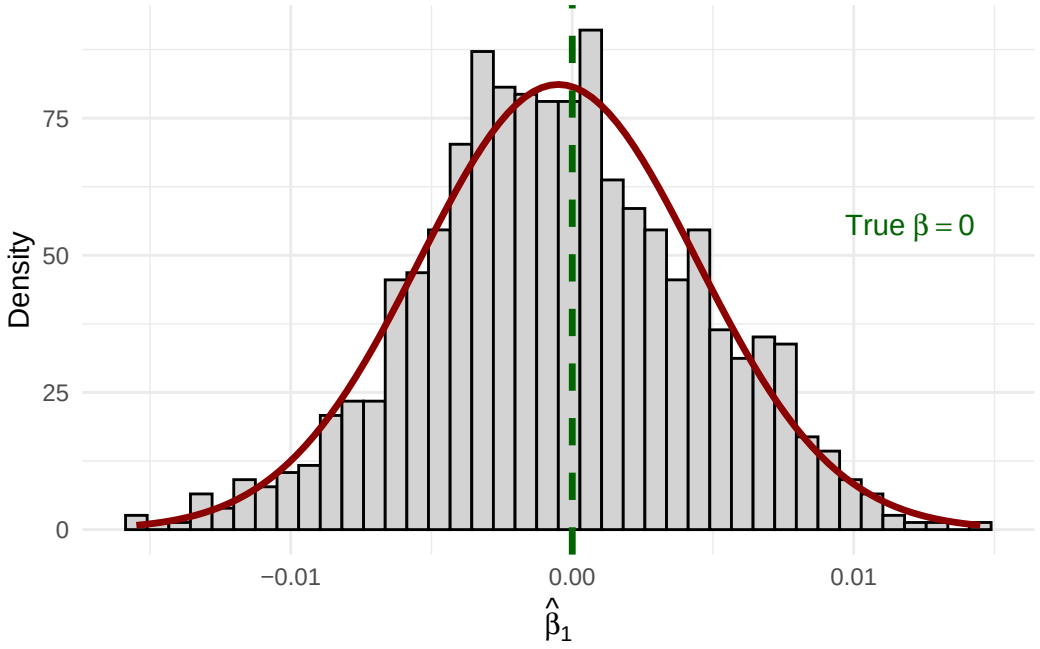
Under the classical linear model (CLM) assumptions—that is, the Gauss-Markov assumptions plus the normality assumption—the normality of the error term directly gives us:

$$\hat{\beta}_j \sim Normal\left[\beta_j, Var(\hat{\beta}_j)\right]$$

We already showed above that the sampling distribution of $\hat{\beta}_1$ from our simulation looks like a bell curve. Let's overlay the theoretical normal density to confirm:

```
# Overlay a normal density on our simulation results
ggplot(sim_results, aes(x = b1_hat)) +
  geom_histogram(aes(y = after_stat(density)),
                 fill = "lightgrey", color = "black", bins = 40) +
  stat_function(
    fun = dnorm,
    args = list(mean = mean(sim_results$b1_hat),
                sd = sd(sim_results$b1_hat)),
    color = "darkred",
    linewidth = 1.2
  ) +
  geom_vline(xintercept = 0, color = "darkgreen", linewidth = 1.2, linetype = "dashed") +
  annotate("text", x = 0.012, y = 55,
           label = "True~beta == 0", parse = TRUE, color = "darkgreen", size = 4) +
  labs(
    x = expression(hat(beta)[1]),
    y = "Density"
  ) +
  theme_minimal()
```

Figure 6.5: The sampling distribution of ˆ follows a normal distribution (red curve) centered on the true value.



The histogram of our 1,000 simulated estimates matches the theoretical normal curve almost perfectly. But in that simulation, the errors were drawn from a normal distribution. What if they aren't?

6.4.3 Asymptotic Normality: Why It Still Works with Non-Normal Errors

In practice, we rarely believe the error term is truly normal. Wages are right-skewed. Health expenditures have a long right tail. Test scores may be bimodal. But here is the good news: by the **Central Limit Theorem**, as $n \to \infty$, the sampling distribution of $\hat{\beta}_j$ converges to a normal distribution *regardless of the distribution of* u, provided the Guass-Markov assumptions hold and a few regularity conditions are satisfied. Formally:

$$\frac{\hat{\beta}_j - \beta_j}{se(\hat{\beta}_j)} \xrightarrow{d} Normal(0,1)$$

The intuition is straightforward: $\hat{\beta}_j$ is a weighted average of the y_i values, and by extension a weighted average of the u_i. The CLT tells us that averages of independent random variables become approximately normal, even when the underlying variables are far from normal themselves.

To see this in action, let's compare two data generating processes: one where the errors are well-behaved (normal), and one where they are heavily skewed—as we might expect when modeling something like wages.

6.4.3.1 Model 1: Normal Errors

Our first DGP uses the classic setup with normally distributed errors:

$$y_i = 2 + 3x_i + u_i, \qquad u_i \sim Normal(0, 4)$$

```
set.seed(42)
n_obs <- 50          # modest sample size
n_sims_asym <- 2000
beta0_true <- 2
beta1_true <- 3

# Store estimates
sim_normal <- tibble(b1_hat = numeric(n_sims_asym))

for (s in 1:n_sims_asym) {
  x <- runif(n_obs, 1, 10)
  u <- rnorm(n_obs, mean = 0, sd = 2)        # normal errors
  y <- beta0_true + beta1_true * x + u
  fit <- lm(y ~ x)
  sim_normal$b1_hat[s] <- coef(fit)["x"]
}
```

6.4.3.2 Model 2: Skewed Errors (Wage-Like)

Now consider a model that looks more like a wage equation. Wages are right-skewed: most workers earn moderate amounts, but a long tail stretches toward high earners. We can capture this by drawing errors from a shifted exponential distribution, which is decidedly non-normal:

$$wage_i = 2 + 3 \cdot educ_i + u_i, \qquad u_i \sim Exponential(\lambda = 0.5) - 2$$

The errors here have a mean of zero (we shift the exponential so it's centered), but they are heavily right-skewed with a skewness of 2—very far from the symmetric bell curve.

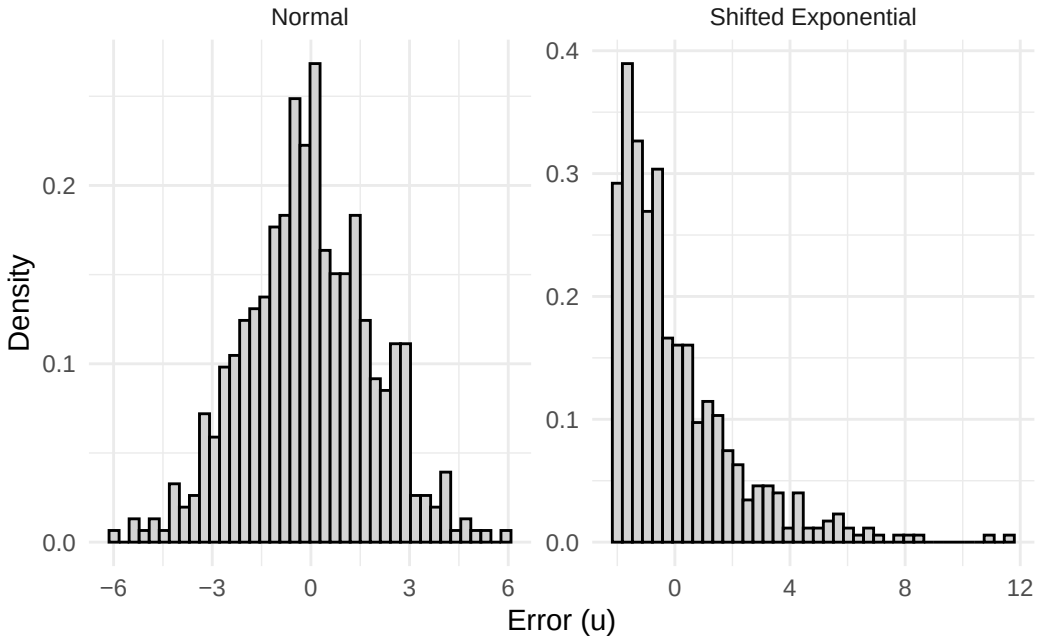
```r
sim_skewed <- tibble(b1_hat = numeric(n_sims_asym))

lambda_rate <- 0.5
error_mean <- 1 / lambda_rate   # mean of Exp(0.5) is 2

for (s in 1:n_sims_asym) {
  x <- runif(n_obs, 1, 10)
  u <- rexp(n_obs, rate = lambda_rate) - error_mean   # shifted so E[u] = 0, skewness = 2
  y <- beta0_true + beta1_true * x + u
  fit <- lm(y ~ x)
  sim_skewed$b1_hat[s] <- coef(fit)["x"]
}
```

6.4.3.3 Comparing the Error Distributions

Before looking at the sampling distributions, let's see what a single draw of errors looks like in each case. The contrast is stark:

```r
set.seed(42)
errors_df <- tibble(
  Normal = rnorm(500, 0, 2),
  `Shifted Exponential` = rexp(500, rate = lambda_rate) - error_mean
) |>
  pivot_longer(everything(), names_to = "Distribution", values_to = "u")

ggplot(errors_df, aes(x = u)) +
  geom_histogram(aes(y = after_stat(density)),
                 fill = "lightgrey", color = "black", bins = 40) +
  facet_wrap(~Distribution, scales = "free") +
  labs(x = "Error (u)", y = "Density") +
  theme_minimal()
```

Figure 6.6: A single draw of errors from each DGP. The normal errors (left) are symmetric; the exponential errors (right) are heavily right-skewed.



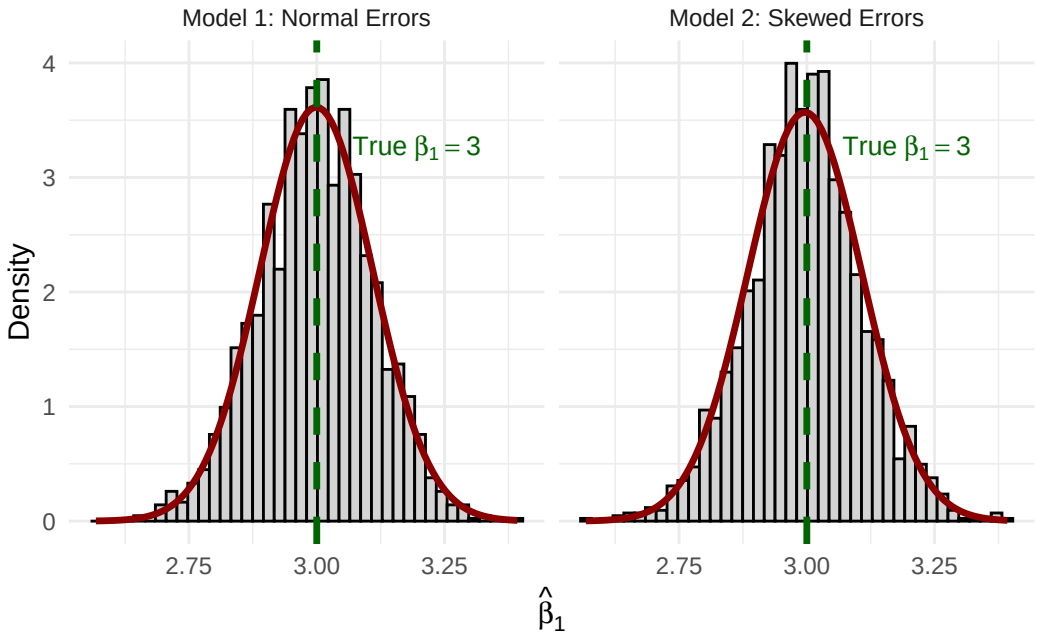
6.4.3.4 The Punchline: Both Sampling Distributions Are Normal

Despite the dramatic difference in error distributions, the sampling distribution of $\hat{\beta}_1$ is approximately normal in *both* cases:

```
combined <- bind_rows(
  sim_normal |> mutate(DGP = "Model 1: Normal Errors"),
  sim_skewed |> mutate(DGP = "Model 2: Skewed Errors")
)

ggplot(combined, aes(x = b1_hat)) +
  geom_histogram(aes(y = after_stat(density)),
                 fill = "lightgrey", color = "black", bins = 40) +
  stat_function(
    data = combined |> filter(DGP == "Model 1: Normal Errors"),
    fun = dnorm,
    args = list(mean = mean(sim_normal$b1_hat), sd = sd(sim_normal$b1_hat)),
    color = "darkred", linewidth = 1.2
  ) +
  stat_function(
```

```
  data = combined |> filter(DGP == "Model 2: Skewed Errors"),
  fun = dnorm,
  args = list(mean = mean(sim_skewed$b1_hat), sd = sd(sim_skewed$b1_hat)),
  color = "darkred", linewidth = 1.2
) +
geom_vline(xintercept = beta1_true, color = "darkgreen", linewidth = 1.2, linetype = "dashe
annotate("text", x = beta1_true + 0.07, y = max(
  dnorm(mean(sim_normal$b1_hat), mean(sim_normal$b1_hat), sd(sim_normal$b1_hat)),
  dnorm(mean(sim_skewed$b1_hat), mean(sim_skewed$b1_hat), sd(sim_skewed$b1_hat))
) * 0.9,
label = paste0("True~beta[1] == ", beta1_true), parse = TRUE, color = "darkgreen", size = 3
facet_wrap(~DGP) +
labs(x = expression(hat(beta)[1]), y = "Density") +
theme_minimal()
```

Figure 6.7: Sampling distributions of$\hat{}$ from 2,000 replications. Despite skewed errors in Model 2, both distributions are well-approximated by the normal curve (red).



The normal curve fits both histograms remarkably well. Even with $n = 50$ and a heavily skewed error distribution, the CLT does its work. Both sampling distributions are centered on the true value $\beta_1 = 3$ (unbiasedness), and both are well-approximated by a normal density.

This is why asymptotic normality is so powerful for applied econometrics: we don't need to believe that wages, test scores, or health expenditures are normally distributed. We only need

a large enough sample for the CLT to kick in—and in practice, $n = 50$ is often sufficient.

> **Practical Implication**
>
> This result justifies our use of t-statistics and confidence intervals even when the dependent variable (and hence the errors) is clearly non-normal. When you estimate a wage equation and `summary(lm(...))` reports p-values based on the t-distribution, it's asymptotic normality doing the heavy lifting.

Quick Check: This section contains interactive content only available in the HTML version.

6.5 Hypothesis Testing

Now that we understand the sampling distribution, we can use it to make inferences about population parameters. The key idea is **hypothesis testing**: we hypothesize about the value of β_j in the population, then use our sample estimate to evaluate whether the hypothesis is plausible.

Here's the roadmap: first, we'll state the hypothesis. Then we'll ask, "If the null hypothesis were true, what would the world look like?" Once we have that picture in mind, we'll introduce the tool—the t-statistic—that lets us locate our estimate in that picture. Finally, we'll formalize a decision rule.

6.5.1 The Null and Alternative Hypotheses

In econometrics, we're usually interested in whether a variable has *any* effect on the outcome. We formalize this as:

Null Hypothesis: $H_0 : \beta_j = 0$

The null hypothesis states that the variable x_j has no effect on y in the population (after controlling for other variables).

Alternative Hypothesis: We test the null against one of:

- $H_1 : \beta_j \neq 0$ (two-sided test, most common)
- $H_1 : \beta_j > 0$ (one-sided, if we expect a positive effect)
- $H_1 : \beta_j < 0$ (one-sided, if we expect a negative effect)

For example, consider the wage equation:

$$wage = \beta_0 + \beta_1(educ) + \beta_2(experience) + \beta_3(tenure) + \mu$$

The null hypothesis $H_0 : \beta_2 = 0$ states that, after controlling for education and tenure, experience has no effect on wages.

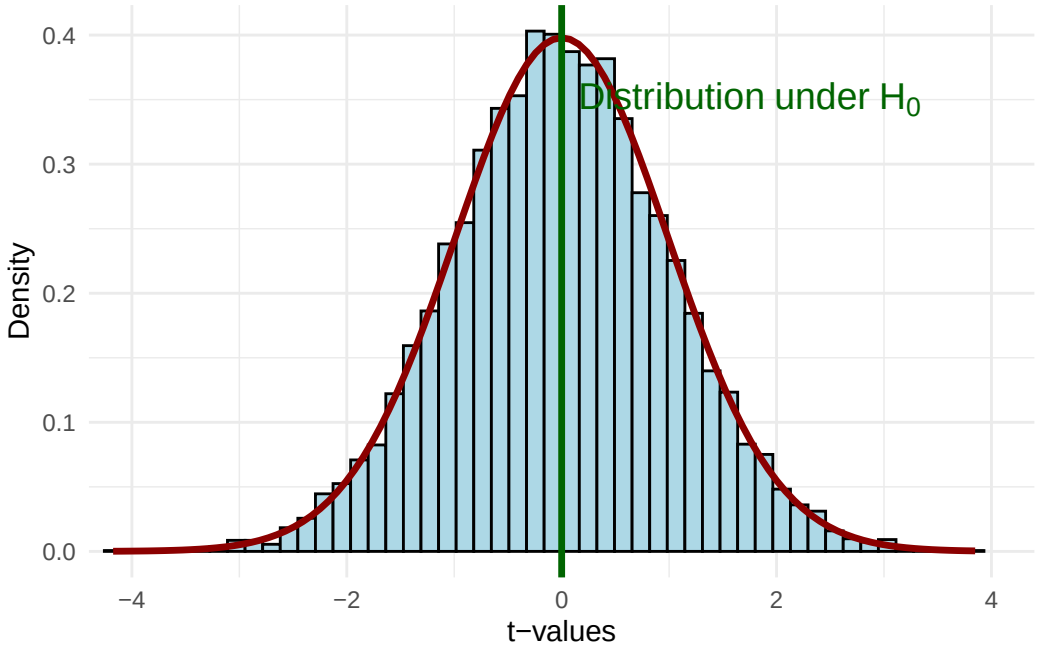
6.5.2 Step 1: What Does the World Look Like Under the Null Hypothesis?

Before we get to any formulas, let's think about what the null hypothesis *implies*. If $H_0 : \beta_j = 0$ is true—that is, if the variable truly has no effect—then our non-zero estimate $\hat{\beta}_j$ is just picking up random noise from sampling variation. Sometimes it'll be a little positive, sometimes a little negative, but it should hover around zero.

We already saw this in our simulation earlier: when the true $\beta_1 = 0$, repeated sampling produced a bell-shaped distribution of estimates centered on zero. Under the null, the *standardized* version of our estimator follows a t-distribution:

```
set.seed(42)
df_value <- 100  # degrees of freedom

t_sample <- tibble(
  t_value = rt(10000, df = df_value)
)

ggplot(t_sample, aes(x = t_value)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 50,
                 fill = "lightblue",
                 color = "black") +
  stat_function(fun = dt,
                args = list(df = df_value),
                color = "darkred",
                linewidth = 1.2) +
  geom_vline(xintercept = 0, color = "darkgreen", linewidth = 1.2) +
  annotate("text", x = 1.75, y = 0.35,
           label = "Distribution~under~H[0]", parse = TRUE,
           color = "darkgreen", size = 5) +
  labs(x = "t-values", y = "Density") +
  coord_cartesian(xlim = c(-4, 4)) +
  theme_minimal()
```

Figure 6.8: The t-distribution under the null hypothesis H : = 0. If the null is true, most standardized estimates will fall near zero.



This is the world we're assuming when we conduct a hypothesis test. Most values cluster near zero, and values far from zero are rare. The question becomes: *does our estimate look like it belongs in this distribution, or does it look like an outlier?*

6.5.3 Step 2: The t-Statistic—Placing Our Estimate on the Distribution

Now we need a tool for locating where our particular estimate falls on that distribution. We can't just use $\hat{\beta}_j$ directly, because the scale depends on the units of measurement and the amount of noise in the data. Instead, we *standardize* the estimate by dividing by its **standard error**.

Recall that the standard error $se(\hat{\beta}_j)$ measures the standard deviation of the sampling distribution of $\hat{\beta}_j$—it tells us how much our estimate would typically vary across repeated samples. A large standard error means there's a lot of noise in our estimate; a small standard error means our estimate is relatively precise. When you run `summary()` on an `lm()` object in R, the standard error is reported in the `Std. Error` column right next to each coefficient.

We standardize by dividing our estimate by the standard error:

$$t_{\hat{\beta}_j} = \frac{\hat{\beta}_j}{se(\hat{\beta}_j)}$$

This is the **t-statistic**. It tells us: *how many standard errors away from zero is our estimate?* Under the null hypothesis, this quantity follows a t-distribution with $n - k - 1$ degrees of freedom:

$$\frac{\hat{\beta}_j - \beta_j}{se(\hat{\beta}_j)} \sim t_{n-k-1}$$

where β_j is the assumed value of the true population parameter under the null hypothesis, $\hat{\beta}_j$ is the estimate, $se(\hat{\beta}_j)$ is the standard error of the estimate, and $n - k - 1$ are the degrees of freedom (sample size minus the number of parameters minus 1).

When testing $H_0 : \beta_j = 0$, the β_j in the numerator drops out, giving us the simple formula above.

The t-statistic has several useful properties:

1. **Same sign as the estimate:** Since $se(\hat{\beta}_j) > 0$, the t-stat has the same sign as $\hat{\beta}_j$
2. **Magnitude matters:** As $\hat{\beta}_j$ grows in magnitude, so does $t_{\hat{\beta}_j}$
3. **Signal-to-noise ratio:** The t-stat measures how large our estimate is *relative to the noise* (uncertainty) in our data

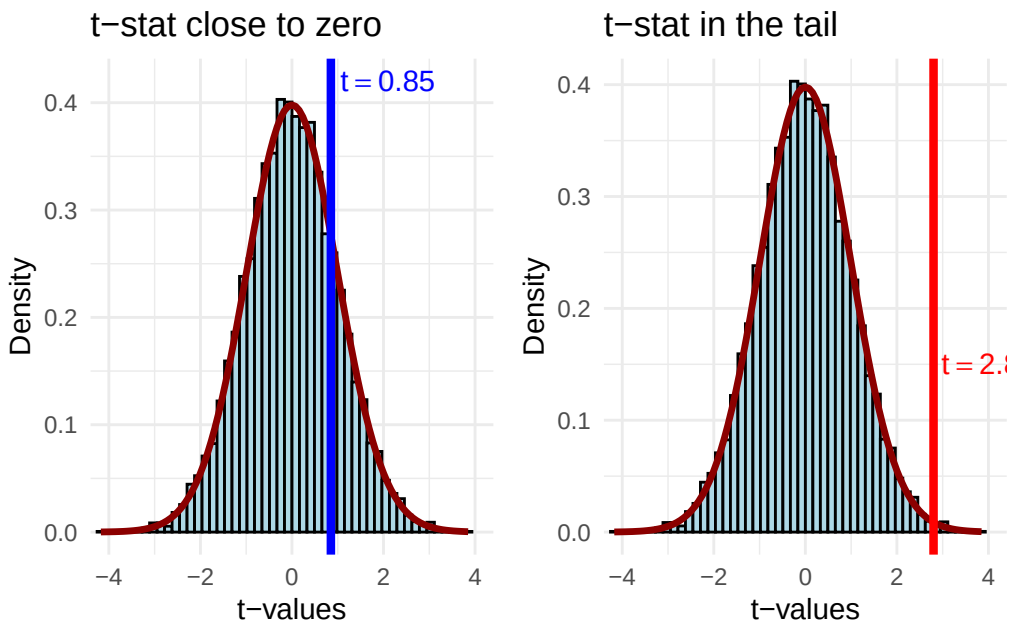
6.5.4 Step 3: How Far is "Too Far"?

Now we can put it all together. We have our imagined distribution of possible estimates under the null, and we have a t-statistic that tells us where our estimate falls on it. The question is: *is our t-statistic close enough to zero to be consistent with the null, or is it so far out in the tails that we should doubt the null?*

```
p_close <- ggplot(t_sample, aes(x = t_value)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 50, fill = "lightblue", color = "black") +
  stat_function(fun = dt, args = list(df = df_value),
                color = "darkred", linewidth = 1.2) +
  geom_vline(xintercept = 0.85, color = "blue", linewidth = 1.5) +
  annotate("text", x = 0.85, y = 0.42, label = "t == 0.85",
           parse = TRUE, color = "blue", size = 4, hjust = -0.1) +
```

```
  labs(x = "t-values", y = "Density",
       title = "t-stat close to zero") +
  coord_cartesian(xlim = c(-4, 4)) +
  theme_minimal()

p_far <- ggplot(t_sample, aes(x = t_value)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 50, fill = "lightblue", color = "black") +
  stat_function(fun = dt, args = list(df = df_value),
                color = "darkred", linewidth = 1.2) +
  geom_vline(xintercept = 2.8, color = "red", linewidth = 1.5) +
  annotate("text", x = 2.8, y = 0.15, label = "t == 2.8",
           parse = TRUE, color = "red", size = 4, hjust = -0.1) +
  labs(x = "t-values", y = "Density",
       title = "t-stat in the tail") +
  coord_cartesian(xlim = c(-4, 4)) +
  theme_minimal()

p_close + p_far
```

Figure 6.9: Two possible t-statistics: one at 0.85 (consistent with H) and one at 2.8 (in the tail, suggesting we reject H).



A t-statistic of 0.85 falls well within the "body" of the distribution—values like this occur

frequently when H_0 is true. We'd have no reason to doubt the null. But a t-statistic of 2.8 is out in the tail—such extreme values are rare when H_0 is true. This is evidence that the null may not be correct.

6.5.5 Critical Values and Rejection Regions

So we have a way to measure whether our estimate is "far" from zero (the t-statistic), and we can see from the comparison above that values of t far out in the tails seem inconsistent with the null. But how far is far enough to *reject* the null? We need a formal decision rule.

The key concern is that we might make a mistake. Even when H_0 is true, we could get an unlucky sample that produces a large t-statistic, leading us to reject a null that is actually correct. This is called a **Type I error** (a "false positive"). We want to control how often this happens.

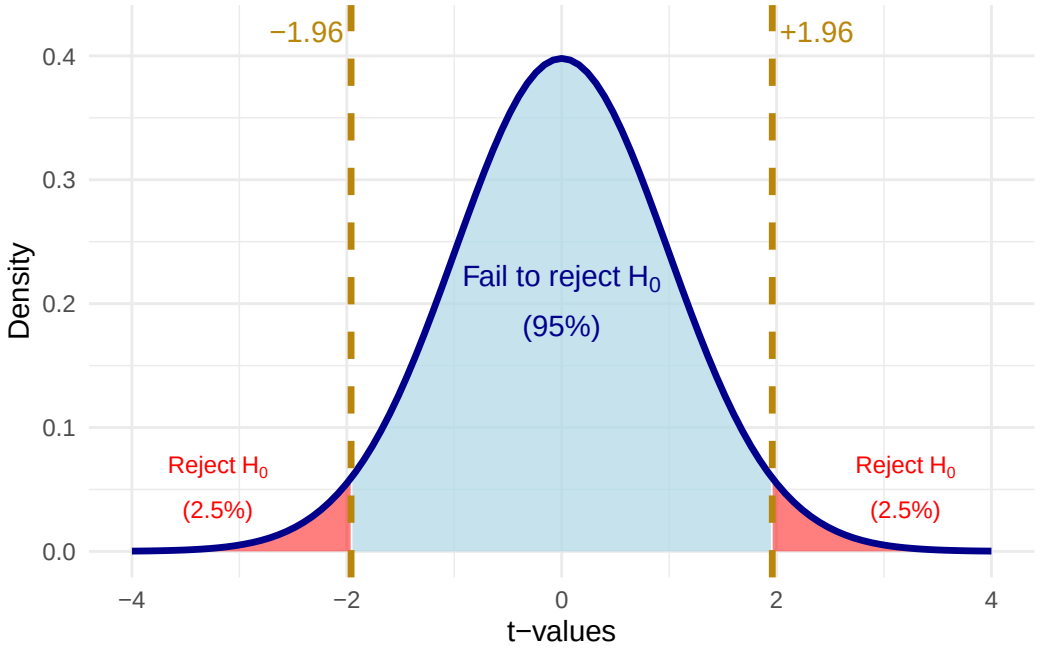
The **significance level** α is the probability of committing a Type I error—the probability of rejecting H_0 when it is in fact true. By choosing α before we look at the data, we set our tolerance for false positives. Common choices are 10%, 5%, and 1%.

With the significance level in hand, we can define a **critical value** c: the threshold on the t-distribution beyond which we reject H_0. For a two-sided test at the 5% level, we reject H_0 if $|t_{\hat{\beta}_j}| > c$, where c is the value that leaves 2.5% of the distribution in each tail (so the total probability of being in either tail is 5%).

```
# Calculate critical value for 5% significance, two-sided
alpha <- 0.05
c_val <- 1.96

# Create shading data for rejection regions
t_grid <- seq(-4, 4, length.out = 500)
t_dens <- dt(t_grid, df = df_value)

shade_data <- tibble(t_value = t_grid, density = t_dens) |>
  mutate(
    region = case_when(
      t_value < -c_val ~ "reject",
      t_value > c_val ~ "reject",
      TRUE ~ "fail_to_reject"
    )
  )

ggplot() +
  # Fail to reject region (blue)
```

```
  geom_ribbon(data = shade_data |> filter(region == "fail_to_reject"),
              aes(x = t_value, ymin = 0, ymax = density),
              fill = "lightblue", alpha = 0.7) +
  # Left rejection region (red)
  geom_ribbon(data = shade_data |> filter(t_value <= -c_val),
              aes(x = t_value, ymin = 0, ymax = density),
              fill = "red", alpha = 0.5) +
  # Right rejection region (red)
  geom_ribbon(data = shade_data |> filter(t_value >= c_val),
              aes(x = t_value, ymin = 0, ymax = density),
              fill = "red", alpha = 0.5) +
  # Distribution curve
  stat_function(fun = dt, args = list(df = df_value),
                color = "darkblue", linewidth = 1.2) +
  # Critical value lines
  geom_vline(xintercept = c(-c_val, c_val),
             color = "darkgoldenrod", linewidth = 1.2, linetype = "dashed") +
  # Annotations
  annotate("text", x = -c_val, y = 0.42,
           label = paste0("-", round(c_val, 2)),
           color = "darkgoldenrod", size = 4, hjust = 1.1) +
  annotate("text", x = c_val, y = 0.42,
           label = paste0("+", round(c_val, 2)),
           color = "darkgoldenrod", size = 4, hjust = -0.1) +
  annotate("text", x = 0, y = 0.2,
           label = "atop('Fail to reject'~H[0], '(95%)')", parse = TRUE,
           color = "darkblue", size = 4) +
  annotate("text", x = 3.2, y = 0.05,
           label = "atop('Reject'~H[0], '(2.5%)')", parse = TRUE,
           color = "red", size = 3) +
  annotate("text", x = -3.2, y = 0.05,
           label = "atop('Reject'~H[0], '(2.5%)')", parse = TRUE,
           color = "red", size = 3) +
  labs(x = "t-values", y = "Density") +
  coord_cartesian(xlim = c(-4, 4)) +
  theme_minimal()
```

Figure 6.10: Critical values for a two-sided test at the 5% significance level. We reject H if |t| > 1.96.



6.5.6 Hypothesis Testing Example

Let's work through a complete example. We'll create some simulated housing data and test whether lot area affects sale price:

```
# Create simulated housing data
set.seed(2025)
n <- 500

housing_data <- tibble(
  lot_area = runif(n, 5000, 20000),
  pool_area = rbinom(n, 1, 0.15) * runif(n, 200, 600),
  garage_area = runif(n, 200, 800),
  year_built = sample(1960:2020, n, replace = TRUE),
  year_remod = pmax(year_built, sample(1980:2023, n, replace = TRUE)),
  # True relationship: lot_area has effect of $1.50 per sq ft
  sale_price = -2500000 + 1.50 * lot_area + 80 * pool_area +
               150 * garage_area + 500 * year_built +
               900 * year_remod + rnorm(n, 0, 50000)
)
```

```r
# Estimate the regression
reg_housing <- lm(sale_price ~ lot_area + pool_area + garage_area +
                    year_built + year_remod,
                  data = housing_data)

summary(reg_housing)
```

```
Call:
lm(formula = sale_price ~ lot_area + pool_area + garage_area +
    year_built + year_remod, data = housing_data)

Residuals:
    Min      1Q  Median      3Q     Max
-134582  -32117     138   32584  167861

Coefficients:
                Estimate    Std. Error t value             Pr(>|t|)
(Intercept) -2369345.4532   407348.8976  -5.817         0.0000000108 ***
lot_area           1.5528        0.5207   2.982             0.003002 **
pool_area         81.5522       14.6980   5.549         0.0000000470 ***
garage_area      167.0837       13.4931  12.383 < 0.00000000000000002 ***
year_built       554.2853      143.8278   3.854             0.000132 ***
year_remod       775.7472      223.5861   3.470             0.000567 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 51130 on 494 degrees of freedom
Multiple R-squared:  0.3396,    Adjusted R-squared:  0.3329
F-statistic: 50.81 on 5 and 494 DF,  p-value: < 0.00000000000000022
```

To test $H_0 : \beta_1 = 0$ (lot area has no effect), we extract the coefficient and standard error, compute the t-statistic, and compare it to a critical value. To find the critical value in R, we use the `qt()` function, which is R's **quantile function** for the t-distribution. It returns the t-value that leaves a given probability in the tail. For a two-sided 5% test, we pass `0.025` (half of 0.05) to get the value that leaves 2.5% in each tail:

```r
# Extract the coefficient and standard error for lot_area
b1_hat <- coef(reg_housing)["lot_area"]
se_b1 <- summary(reg_housing)$coefficients["lot_area", "Std. Error"]
```

```r
# Calculate the t-statistic
t_stat <- b1_hat / se_b1

# Find the critical value at 5% significance
sig_level <- 0.05
n_obs <- nobs(reg_housing)
k <- 5  # number of independent variables
df <- n_obs - k - 1

t_crit <- qt(sig_level / 2, df = df, lower.tail = FALSE)

# Display results
cat("Estimate:", round(b1_hat, 4), "\n")
```

```
Estimate: 1.5528
```

```r
cat("Standard Error:", round(se_b1, 4), "\n")
```

```
Standard Error: 0.5207
```

```r
cat("t-statistic:", round(t_stat, 2), "\n")
```

```
t-statistic: 2.98
```

```r
cat("Critical value (5%):", round(t_crit, 2), "\n")
```

```
Critical value (5%): 1.96
```

```r
cat("Reject H0?", abs(t_stat) > t_crit, "\n")
```

```
Reject H0? TRUE
```

Since $|t_{\hat{\beta}_1}|$ exceeds the critical value of approximately 1.96, we **reject the null hypothesis** that lot area has no effect on sale price. The estimate is **statistically significant at the 5% level**.

6.5.7 Rule of Thumb

With modern datasets that have hundreds or thousands of observations, the critical value for a two-sided 5% test is essentially always **1.96 2**.

This gives us a handy rule of thumb:

> The "Rule of 2"
>
> An estimate is statistically significant at the 5% level if:
>
> $$|\hat{\beta}_j| > 2 \times se(\hat{\beta}_j)$$
>
> In other words, if your estimate is more than twice as large as its standard error, you can reject the null hypothesis of no effect at the 5% level.

6.6 P-Values

While the t-test with a fixed significance level is useful, it has limitations. Saying an estimate is "significant at 5%" doesn't tell us *how strongly* we reject the null. Was it a narrow rejection or a "knock out"?

The **p-value** answers this question.

Put differently, in the critical value approach, *we* pick a significance level (say 5%) and then ask whether our t-statistic exceeds the corresponding cutoff. The p-value flips this around. Instead of fixing the significance level and checking if we reject, the p-value asks: *what is the smallest significance level at which we would still reject?* If that number is very small (like 0.001), it means we'd reject even with an extremely strict threshold—strong evidence against the null. If it's large (like 0.45), it means we'd need a very loose threshold to reject—weak or no evidence against the null.

More concretely, the p-value is the probability of observing a t-statistic as extreme (or more extreme) as the one we calculated, *assuming the null hypothesis is true*. It measures how "surprising" our data are under the null. A small p-value means the data would be very unlikely if H_0 were true, which casts doubt on H_0.

> Definition: P-Value
>
> The p-value is the smallest significance level at which we would reject the null hypothesis. Equivalently: the p-value is the probability of observing a t-statistic as extreme (or more extreme) as the one we calculated, *if the null hypothesis were true*.

> ⚠ Common Misconception
>
> The p-value is **not** the probability that the null hypothesis is true. It is the probability of seeing data as extreme as ours *if* the null were true. This is a subtle but critical distinction. Saying "there's a 3% chance that $\beta_j = 0$" is incorrect. The correct interpretation is: "if β_j really were zero, there's only a 3% chance we'd observe an estimate this far from zero."

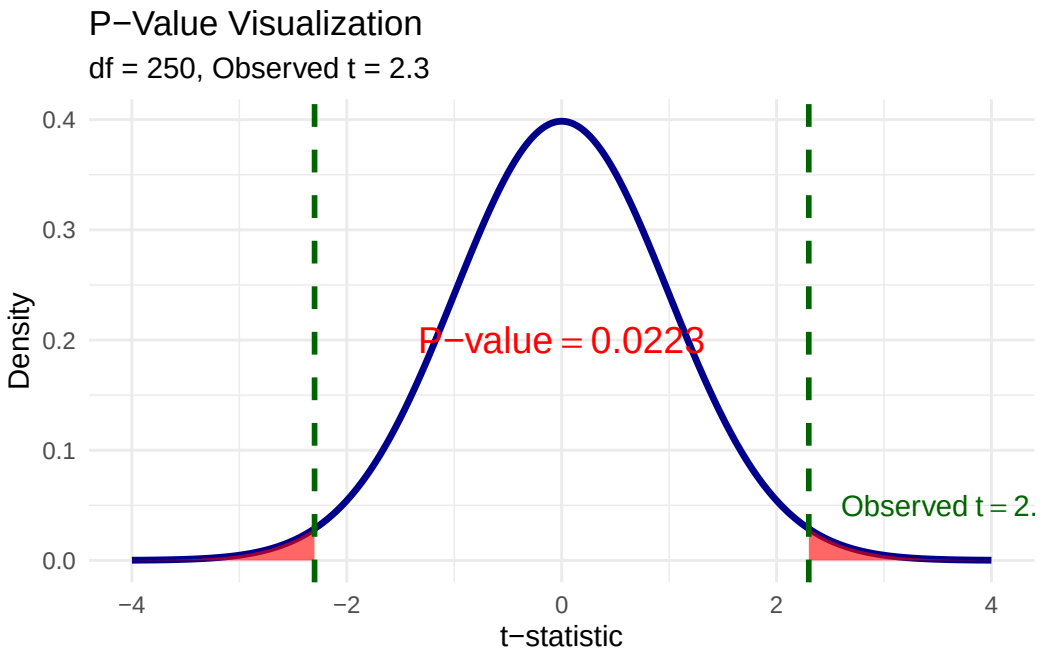
6.6.1 Visualizing P-Values

Let's make this concrete. Suppose we run a regression and get a t-statistic of 2.3. To compute the p-value, we ask: "Under the null, what fraction of the t-distribution lies beyond ± 2.3?" The red shaded area in the figure below is exactly that probability. We shade *both* tails because, for a two-sided test, an estimate of -2.3 would be just as much evidence against the null as $+2.3$. The total red area—the sum of both tails—is the p-value.

```
# Parameters
df_viz <- 250
observed_t <- 2.3

# Calculate p-value
p_val <- 2 * pt(abs(observed_t), df = df_viz, lower.tail = FALSE)

# Create the visualization
t_grid <- seq(-4, 4, length.out = 500)
t_dens <- dt(t_grid, df = df_viz)

shade_data <- tibble(t_value = t_grid, density = t_dens) |>
  mutate(
    in_tail = abs(t_value) >= abs(observed_t)
  )

ggplot() +
  # Main curve
  geom_line(data = shade_data, aes(x = t_value, y = density),
            color = "darkblue", linewidth = 1.2) +
  # Shaded tails (p-value area)
  geom_area(data = shade_data |> filter(in_tail & t_value > 0),
            aes(x = t_value, y = density),
            fill = "red", alpha = 0.6) +
  geom_area(data = shade_data |> filter(in_tail & t_value < 0),
            aes(x = t_value, y = density),
```

```
          fill = "red", alpha = 0.6) +
# Observed t-statistic lines
geom_vline(xintercept = c(-observed_t, observed_t),
           color = "darkgreen", linewidth = 1, linetype = "dashed") +
# Annotations
annotate("text", x = observed_t + 0.3, y = 0.05,
         label = paste0("'Observed'~t == ", observed_t),
         parse = TRUE, color = "darkgreen", size = 4, hjust = 0) +
annotate("text", x = 0, y = 0.2,
         label = paste0("'P-value' == ", round(p_val, 4)),
         parse = TRUE, color = "red", size = 5) +
labs(
  title = "P-Value Visualization",
  subtitle = paste0("df = ", df_viz, ", Observed t = ", observed_t),
  x = "t-statistic",
  y = "Density"
) +
theme_minimal()
```

Figure 6.11: The p-value is the total shaded area in both tails beyond the observed t-statistic. Here, the observed t = 2.3, and the combined red area gives p 0.022.



6.6.2 Interpreting P-Values

The p-value tells us the probability of getting our estimate (or something more extreme) if the null were true:

- **p = 0.035:** There's a 3.5% chance of getting our estimate if $\beta_j = 0$. That's quite unlikely—evidence against H_0.
- **p = 0.35:** There's a 35% chance of getting our estimate if $\beta_j = 0$. That's not unusual at all—no evidence against H_0.

Common decision rules:

- p < 0.10: Reject H_0 at 10% level (weak evidence against H_0)
- p < 0.05: Reject H_0 at 5% level (moderate evidence against H_0)
- p < 0.01: Reject H_0 at 1% level (strong evidence against H_0)

6.6.3 A Note on Language

When we *cannot* reject the null hypothesis, we say:

> "We **fail to reject** the null at the x% level."

We do **not** say we "accept" the null. Why? Because there are many possible values of β_j that we would also fail to reject. Failing to reject $\beta_j = 0$ doesn't mean β_j actually equals zero—it just means our data aren't precise enough to distinguish β_j from zero.

6.6.4 Statistical vs. Practical Significance

> ⚠ Important Distinction
>
> **Statistical significance** and **practical (economic) significance** are not the same thing!
> A coefficient can be statistically significant but practically unimportant. This is especially common in very large samples, where even tiny effects can be detected.

For example, suppose we find that an additional year of education increases wages by \$0.50 per year, and this is statistically significant with p < 0.001. Statistically, we're very confident the effect isn't zero. But practically, a 50-cent annual raise is economically trivial.

Always interpret the *magnitude* of coefficients, not just their statistical significance.

Quick Check: This section contains interactive content only available in the HTML version.

6.7 Confidence Intervals

Another way to quantify uncertainty is through **confidence intervals**. While hypothesis tests ask "is β_j different from zero?", confidence intervals ask "what range of values is β_j likely to fall within?"

6.7.1 Computing a Confidence Interval

A confidence interval at the $(1 - \alpha)$ level is:

$$\hat{\beta}_j \pm c_\alpha \times se(\hat{\beta}_j)$$

where c_α is the critical value at significance level α.

For a 95% confidence interval with large samples:

$$\hat{\beta}_j \pm 1.96 \times se(\hat{\beta}_j)$$

6.7.2 Example

Using our housing regression:

```
# 95% confidence interval for lot_area coefficient
lower <- b1_hat - 1.96 * se_b1
upper <- b1_hat + 1.96 * se_b1

cat("Point estimate:", round(b1_hat, 4), "\n")
```

```
Point estimate: 1.5528
```

```
cat("95% CI: [", round(lower, 4), ",", round(upper, 4), "]\n")
```

```
95% CI: [ 0.5323 , 2.5733 ]
```

In practice, you don't need to compute confidence intervals by hand. R's `confint()` function does it for you. The first argument is the fitted model, the second specifies which coefficient (or omit it to get CIs for all coefficients), and `level` sets the confidence level (0.95 for a 95% CI):

```
confint(reg_housing, "lot_area", level = 0.95)
```

```
             2.5 %  97.5 %
lot_area 0.5298262 2.57581
```

6.7.3 Interpreting Confidence Intervals

> "We are 95% confident that the true effect of lot area on sale price is between \$0.53 and \$2.57 per square foot."

What does "95% confident" actually mean? It does *not* mean there's a 95% probability that the true β_j falls in this particular interval. The true β_j is a fixed number—it's either in the interval or it isn't. Rather, the "95%" refers to the *procedure*: if we drew 100 different samples from the same population and computed a 95% confidence interval from each one, about 95 of those intervals would contain the true β_j, and about 5 would miss it. Any single interval is one draw from this process, and we don't know whether ours is one of the lucky 95 or the unlucky 5.

This is why confidence intervals are so useful in practice: they give us a range of *plausible* values for the population parameter, not just a single point estimate. A narrow confidence interval means our estimate is precise; a wide one means there's still a lot of uncertainty.

Key insight: There is a direct connection between confidence intervals and hypothesis testing. If the 95% confidence interval includes zero, then we *cannot* reject $H_0 : \beta_j = 0$ at the 5% level. Conversely, if the interval excludes zero, we *can* reject. This makes confidence intervals a handy visual shortcut: just check whether zero is inside or outside the interval.

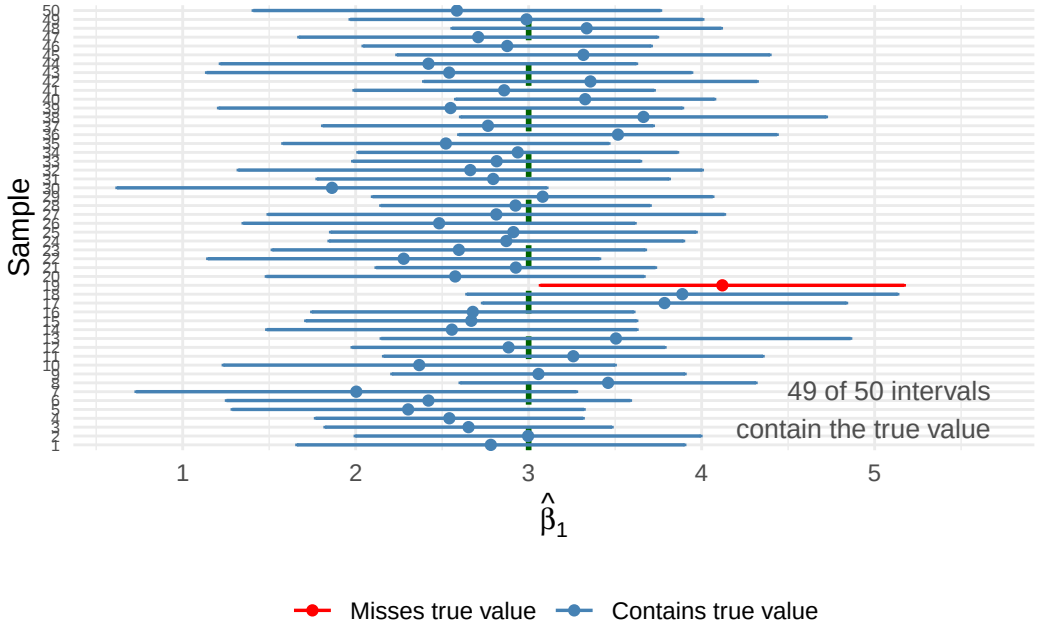
6.7.4 Simulation: Understanding Confidence Intervals

Let's run a simulation to see what "95% confidence" really means. We'll use a simple DGP with enough noise that some intervals will miss the true value—exactly as the theory predicts:

```
set.seed(321)

# Simple DGP: y = 5 + 3x + u, with enough noise to get some misses
true_b0 <- 5
true_b1_ci <- 3
n_per_sample <- 30     # small samples = wider CIs = more misses
n_samples <- 50

ci_data <- map_dfr(1:n_samples, function(i) {
```

```
  x <- runif(n_per_sample, 0, 10)
  u <- rnorm(n_per_sample, 0, 8)    # substantial noise
  y <- true_b0 + true_b1_ci * x + u

  reg <- lm(y ~ x)
  ci <- confint(reg, "x", level = 0.95)

  tibble(
    sample = i,
    estimate = coef(reg)["x"],
    lower = ci[1],
    upper = ci[2],
    covers_true = lower <= true_b1_ci & upper >= true_b1_ci
  )
})

# Count coverage
n_covered <- sum(ci_data$covers_true)

ggplot(ci_data, aes(y = reorder(factor(sample), sample))) +
  geom_vline(xintercept = true_b1_ci, linetype = "dashed",
             color = "darkgreen", linewidth = 1) +
  geom_errorbar(aes(xmin = lower, xmax = upper,
                    color = covers_true),
                width = 0.3, linewidth = 0.6, orientation = "y") +
  geom_point(aes(x = estimate, color = covers_true), size = 1.5) +
  scale_color_manual(values = c("TRUE" = "steelblue", "FALSE" = "red"),
                     labels = c("TRUE" = "Contains true value",
                                "FALSE" = "Misses true value")) +
  annotate("text", x = max(ci_data$upper) - 0.5 + 1, y = 5,
           label = paste0(n_covered, " of ", n_samples, " intervals\ncontain the true value"),
           color = "grey30", size = 3.5, hjust = 1) +
  labs(
    x = expression(hat(beta)[1]),
    y = "Sample",
    color = NULL
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    axis.text.y = element_text(size = 6)
  )
```

Figure 6.12: Fifty 95% confidence intervals from different samples. The true = 3 (dashed line). Blue intervals contain the true value; red intervals miss it. About 95% of intervals should capture the truth.



Notice the red intervals: these are the roughly 5% of samples where the confidence interval happened to miss the true value. This is *not* a failure of the method—it's exactly what "95% confidence" means. The procedure works correctly 95% of the time, but any individual interval might be one of the unlucky ones. This is why we say we are "95% confident" rather than "100% certain."

6.8 T-Tests, P-Values, and CIs: A Comparison

These three approaches are closely related but answer slightly different questions:

Method	Question Answered
T-statistic	Is my estimate large relative to the noise in the data?
P-value	What's the probability of seeing my estimate if H_0 were true?
Confidence Interval	What range of values is plausible for β_j?

All three are mathematically connected:

- If $|t| > 1.96$, then $p < 0.05$, and the 95% CI excludes zero

- If p < 0.05, then $|t| > 1.96$, and the 95% CI excludes zero
- If the 95% CI excludes zero, then p < 0.05 and $|t| > 1.96$

6.9 F-Tests: Testing Multiple Hypotheses

So far, all of our inference tools—t-tests, p-values, confidence intervals—have focused on testing *one* coefficient at a time. But in practice, we often want to ask broader questions. For instance, suppose you're estimating a wage equation with education, experience, tenure, and age. You might want to know: "Do experience and tenure *jointly* matter for wages?" That's not a question about a single β_j—it's a question about multiple coefficients at once.

You might be tempted to just look at the individual t-tests for experience and tenure separately. But this approach has problems.

6.9.1 Why Not Just Use Multiple t-Tests?

There are two issues with testing each coefficient individually:

1. **No restrictions on other parameters:** A t-test on β_2 puts no restrictions on β_3. If the variables are correlated, this can be misleading. You could find that neither is individually significant, yet they jointly explain a lot of variation in the outcome.
2. **Multiple comparisons problem:** If you run many tests at the 5% level, you'll reject *some* true null hypotheses just by chance. With 20 tests, you'd expect about 1 false rejection even if all nulls are true!

The **F-test** solves both problems by testing multiple coefficients simultaneously.

6.9.2 Setting Up the F-Test

Consider the wage model:

$$\log(wage) = \beta_0 + \beta_1(education) + \beta_2(experience) + \beta_3(tenure) + \beta_4(age) + \mu$$

Suppose we want to test whether on-the-job training variables (experience and tenure) matter at all:

$$H_0 : \beta_2 = \beta_3 = 0$$
$$H_1 : H_0 \text{ is not true}$$

The alternative is satisfied if *either* $\beta_2 \neq 0$ or $\beta_3 \neq 0$ (or both).

6.9.3 Computing the F-Statistic

The F-test compares two models:

Unrestricted Model: The full model with all variables

$$UR : \widehat{\log(wage)} = \hat{\beta}_0 + \hat{\beta}_1(educ) + \hat{\beta}_2(exper) + \hat{\beta}_3(tenure) + \hat{\beta}_4(age)$$

Restricted Model: The model assuming the null is true (dropping the restricted variables)

$$R : \widehat{\log(wage)} = \hat{\beta}_0 + \hat{\beta}_1(educ) + \hat{\beta}_4(age)$$

The F-statistic is:

$$F = \frac{(SSR_R - SSR_{UR})/q}{SSR_{UR}/df_{UR}}$$

where:

- SSR_R = Sum of squared residuals from restricted model
- SSR_{UR} = Sum of squared residuals from unrestricted model
- q = Number of restrictions (variables dropped)
- df_{UR} = Degrees of freedom in unrestricted model ($n - k - 1$)

The intuition: if the restricted variables actually matter, then dropping them should make the model fit *worse*—that is, SSR_R should be much larger than SSR_{UR}. The F-statistic captures how much worse the fit gets, scaled by the noise in the data.

6.9.4 F-Test Example

Let's test whether year built and year remodeled jointly affect home prices:

```
# Unrestricted model (full model)
reg_unrestricted <- lm(sale_price ~ lot_area + pool_area + garage_area +
                         year_built + year_remod,
                       data = housing_data)

# Restricted model (dropping year variables)
reg_restricted <- lm(sale_price ~ lot_area + pool_area + garage_area,
                     data = housing_data)

# Compute F-statistic manually
ssr_r <- sum(resid(reg_restricted)^2)
```

```r
ssr_ur <- sum(resid(reg_unrestricted)^2)
df_ur <- df.residual(reg_unrestricted)
q <- 2  # number of restrictions

f_numerator <- (ssr_r - ssr_ur) / q
f_denominator <- ssr_ur / df_ur
f_stat <- f_numerator / f_denominator

# Critical value at 5% significance
f_crit <- qf(0.05, df1 = q, df2 = df_ur, lower.tail = FALSE)

cat("F-statistic:", round(f_stat, 2), "\n")
```

```
F-statistic: 24.69
```

```r
cat("Critical value (5%):", round(f_crit, 2), "\n")
```

```
Critical value (5%): 3.01
```

```r
cat("Reject H0?", f_stat > f_crit, "\n")
```

```
Reject H0? TRUE
```

Since the F-statistic exceeds the critical value, we **reject the null hypothesis**. Year built and year remodeled are **jointly statistically significant**—they collectively improve the model's fit.

6.9.5 Using R's anova() Function

In practice, you'll use R's `anova()` function to perform F-tests. Pass it the restricted model first, then the unrestricted model:

```r
anova(reg_restricted, reg_unrestricted)
```

```
Analysis of Variance Table

Model 1: sale_price ~ lot_area + pool_area + garage_area
Model 2: sale_price ~ lot_area + pool_area + garage_area + year_built +
```

```
    year_remod
  Res.Df           RSS Df    Sum of Sq      F           Pr(>F)
1    496 1420348267818
2    494 1291292854097  2 129055413721 24.686 0.00000000006048 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The output shows the F-statistic, degrees of freedom, and p-value directly.

6.9.6 The F-Statistic in Terms of R^2

The F-statistic can also be written in terms of R-squared values:

$$F = \frac{(R^2_{UR} - R^2_R)/q}{(1 - R^2_{UR})/df_{UR}}$$

This shows that the F-test is essentially asking: *Does the unrestricted model fit the data significantly better than the restricted model?*

If R^2_{UR} is much larger than R^2_R, the F-statistic will be large, and we'll reject the null.

Properties of the F-Test

- The F-statistic is always non-negative, so F-tests are always **one-sided**
- If the null is rejected, we say the variables are **jointly statistically significant**
- We cannot determine *which* individual variable is significant—only that at least one is
- If variables are jointly insignificant, it provides justification for dropping them from the model

6.10 Reading Regression Tables

Now that we have the full toolkit of statistical inference—t-tests, p-values, confidence intervals, and F-tests—we can properly interpret the regression tables you'll encounter in academic papers.

6.10.1 Decoding R's `summary()` Output

Let's start with what you already know: the output from `summary()` in R. It packs a lot of information into one screen, and now that we've covered t-statistics, p-values, and confidence intervals, we can decode every piece of it.

```
# Standard R output
summary(reg_housing)
```

```
Call:
lm(formula = sale_price ~ lot_area + pool_area + garage_area +
    year_built + year_remod, data = housing_data)

Residuals:
    Min      1Q  Median      3Q     Max
-134582  -32117     138   32584  167861

Coefficients:
                Estimate    Std. Error t value             Pr(>|t|)
(Intercept) -2369345.4532   407348.8976  -5.817         0.0000000108 ***
lot_area           1.5528        0.5207   2.982             0.003002 **
pool_area         81.5522       14.6980   5.549         0.0000000470 ***
garage_area      167.0837       13.4931  12.383 < 0.0000000000000002 ***
year_built       554.2853      143.8278   3.854             0.000132 ***
year_remod       775.7472      223.5861   3.470             0.000567 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 51130 on 494 degrees of freedom
Multiple R-squared:  0.3396,    Adjusted R-squared:  0.3329
F-statistic: 50.81 on 5 and 494 DF,  p-value: < 0.00000000000000022
```

The output has several blocks. Let's walk through them.

The Coefficients Table is the most important part. It has four columns:

- **`Estimate`**: You know this one. It is the OLS point estimates $\hat{\beta}_j$. For lot area, this is the estimated effect of one additional square foot of lot area on sale price.
- **`Std. Error`**: The standard error $se(\hat{\beta}_j)$ (i.e., the square root of the variance) of each estimate. This measures the precision of our estimate—how much it would typically vary across repeated samples. Smaller standard errors mean more precise estimates.

- `t value`: The t-statistic based on a null hypothesis where $H_0 = 0$, computed as `Estimate / Std. Error`. This is exactly the calculation we did by hand earlier. It tells us how many standard errors our estimate is from zero.
- `Pr(>|t|)`: The **p-value** for the two-sided test $H_0 : \beta_j = 0$. Smaller numbers mean stronger evidence against the null. A value below 0.05 means the estimate is statistically significant at the 5% level.

Significance Stars appear to the right of the p-values as a quick visual guide:

- No star: not significant at the 10% level ($p \geq 0.10$)
- `.` : significant at 10% but not 5% ($0.05 \leq p < 0.10$)
- `*` : significant at 5% but not 1% ($0.01 \leq p < 0.05$)
- `**` : significant at 1% but not 0.1% ($0.001 \leq p < 0.01$)
- `***` : significant at 0.1% ($p < 0.001$)

More stars = stronger evidence against the null. But remember: statistical significance is not the same as practical significance!

Residual standard error ($\hat{\sigma}$) is an estimate of the standard deviation of the error term u. It tells you the typical size of the prediction error in the units of y. The degrees of freedom ($n-k-1$) appear next to it. This is not something that is useful on its own, so we won't refer to it very often.

Multiple R-squared (R^2) is the fraction of variation in y explained by the model. **Adjusted R-squared** penalizes for adding more variables and is generally preferred for comparing models with different numbers of regressors, as discussed in Chapter 5.

F-statistic at the bottom tests the joint null hypothesis that *all* slope coefficients are zero: $H_0 : \beta_1 = \beta_2 = \cdots = \beta_k = 0$. This asks: "Does this model explain anything at all?" A large F-statistic (or small p-value) means the model as a whole is statistically significant. We covered F-tests earlier in this chapter.

6.10.2 From R Output to Publication Format

While `summary()` gives us everything we need, academic papers present this information in a standardized, cleaner format. In a publication, the same regression would appear as:

6.10.3 Decoding the Table Structure

Rows are variables: The top indicates the dependent variable. Rows below show independent variables and the constant (intercept).

The main numbers are coefficients: Each cell contains $\hat{\beta}_j$.

Table 6.2: The Effect of Lot Area on Home Sale Price

	Sale Price
(Intercept)	−2 369 345.453***
	(407 348.898)
Lot Area (sq ft)	1.553***
	(0.521)
Pool Area	81.552***
	(14.698)
Garage Area	167.084***
	(13.493)
Year Built	554.285***
	(143.828)
Year Remodeled	775.747***
	(223.586)
Num.Obs.	500
R2	0.340
R2 Adj.	0.333

* p <0.1, ** p <0.05, *** p <0.01
Standard errors in parentheses.

Numbers in parentheses are standard errors: These are $se(\hat{\beta}_j)$.

Stars indicate significance levels:

- * = significant at 10% level ($p < 0.10$)
- ** = significant at 5% level ($p < 0.05$)
- *** = significant at 1% level ($p < 0.01$)

Bottom statistics: Sample size (N), R-squared, and sometimes other diagnostics.

6.10.4 Multiple Columns Show Robustness

Most regression tables have multiple columns, each representing a different specification. This lets readers see how estimates change as controls are added:

```
# Three specifications with increasing controls
reg1 <- lm(sale_price ~ lot_area, data = housing_data)
reg2 <- lm(sale_price ~ lot_area + pool_area + garage_area, data = housing_data)
reg3 <- lm(sale_price ~ lot_area + pool_area + garage_area +
             year_built + year_remod, data = housing_data)
```

Notice how the coefficient on Lot Area changes across specifications. In column (1), without controls, the estimate is different from column (3). This pattern often reveals omitted variable bias in simpler specifications.

6.10.5 What to Look for in Regression Tables

1. **What is the coefficient on the variable of interest?** How do we interpret it?
2. **Is it statistically significant?** At what level?
3. **How does the coefficient change across specifications?** Does it remain stable or shift dramatically?
4. **Are the sample size and R-squared reasonable?** Very small N or very low R^2 might raise concerns.

Practice: Reading a Regression Table

Consider this table examining the effect of stock performance on CEO salary:

Is the effect of Return on Stock statistically significant at the 5% level in Column (1)? Select answer… Yes No

Table 6.3: The Effect of Lot Area on Home Sale Price: Multiple Specifications

	(1)	(2)	(3)
(Intercept)	385 464.906***	288 761.744***	−2 369 345.453***
	(8462.611)	(10 454.349)	(407 348.898)
Lot Area (sq ft)	1.038	1.472***	1.553***
	(0.635)	(0.545)	(0.521)
Pool Area		86.081***	81.552***
		(15.352)	(14.698)
Garage Area		170.687***	167.084***
		(14.112)	(13.493)
Year Built			554.285***
			(143.828)
Year Remodeled			775.747***
			(223.586)
Num.Obs.	500	500	500
R2	0.005	0.274	0.340
R2 Adj.	0.003	0.269	0.333

* p <0.1, ** p <0.05, *** p <0.01
Standard errors in parentheses.

Table 6.4: Effect of Stock Performance on CEO Salary

	(1)	(2)
(Intercept)	6.806***	4.788***
	(0.041)	(0.234)
Return on Stock (%)	0.001	0.001
	(0.001)	(0.001)
Log(Sales)		0.287***
		(0.033)
Num.Obs.	209	209
R2 Adj.	−0.004	0.264

* p <0.1, ** p <0.05, *** p <0.01

What happens to the Adjusted R^2 when we add log(Sales) as a control? Select answer… It increases substantially It decreases It stays about the same

Based on this table, which variable appears to be a more important determinant of CEO salary? Select answer… Return on Stock Company Sales Both are equally important

6.11 Chapter Summary

Key Takeaways

1. **Statistical inference** allows us to move from sample estimates to statements about population parameters, accounting for sampling variability.
2. The **sampling distribution** of $\hat{\beta}_j$ describes how our estimates would vary across repeated samples. Under the normality assumption (or with large samples), it follows a normal distribution.
3. **Hypothesis testing** uses the t-statistic to determine whether our estimate is "far enough" from zero to reject the null hypothesis: $t = \hat{\beta}_j / se(\hat{\beta}_j)$
4. **P-values** tell us the probability of seeing our data if the null were true. Smaller p-values provide stronger evidence against H_0.
5. **Confidence intervals** provide a range of plausible values for the population parameter. A 95% CI excludes zero if and only if the estimate is significant at the 5% level.
6. **Statistical significance practical significance.** Always consider the magnitude of effects, not just their p-values.
7. **F-tests** allow us to test multiple hypotheses simultaneously, avoiding the multiple comparisons problem.
8. **Regression tables** in academic papers present coefficients, standard errors (in parentheses), and significance stars. Multiple columns show robustness across specifications.

6.12 Practice Exercises

Note: This section contains interactive content only available in the HTML version.

7 Dummy Variables and Non-Linear Models

Key Questions

- How do we include categorical information (like race, gender, or region) in a regression?
- What is a dummy variable and how do we interpret its coefficient?
- How do we avoid the "dummy variable trap"?
- How can we model non-linear relationships using polynomials?
- When should we use logarithmic transformations?

Suggested Readings

- Wooldridge (2019) Ch. 7
- Bailey (2020), Ch. 5

So far, we have primarily considered **quantitative** variables in our regression models: wages measured in dollars, years of education, birthweight in grams, and so on. But in empirical work, some of the most important variables are **qualitative**: race, gender, region, industry, or whether someone participated in a program. This chapter introduces techniques for incorporating categorical information into regression models and for modeling non-linear relationships.

7.1 Categorical Variables in Regression

Regression requires numeric inputs, so we cannot simply include a variable like "region" with values "South," "Midwest," "Northeast," and "West" directly in our model. We need a way to translate qualitative information into numbers.

7.1.1 Dummy Variables

The most common approach is to create **dummy variables** (also called binary or indicator variables). A dummy variable takes only two values: 1 if the observation has a particular characteristic, and 0 otherwise.

Consider this simple example from the Star Wars dataset:

```
ex_data <- starwars |>
    head(5) |>
    select(name, height, mass, species)

knitr::kable(ex_data, caption = "Star Wars Characters")
```

Table 7.1: Star Wars Characters

name	height	mass	species
Luke Skywalker	172	77	Human
C-3PO	167	75	Droid
R2-D2	96	32	Droid
Darth Vader	202	136	Human
Leia Organa	150	49	Human

The `species` variable is qualitative—we cannot use it directly in a regression. But we can create a dummy variable called `human` that equals 1 for humans and 0 for droids:

```
ex_data <- ex_data |>
    mutate(human = case_when(species == "Human" ~ 1,
                             TRUE ~ 0))

knitr::kable(ex_data, caption = "Star Wars Characters with Dummy Variable")
```

Table 7.2: Star Wars Characters with Dummy Variable

name	height	mass	species	human
Luke Skywalker	172	77	Human	1
C-3PO	167	75	Droid	0
R2-D2	96	32	Droid	0
Darth Vader	202	136	Human	1
Leia Organa	150	49	Human	1

Now we have a numeric variable that captures whether each character is human.

7.1.2 Interpreting Dummy Variable Coefficients

To see how dummy variables work in regression, let's examine a practical example. Suppose we want to know whether taking calculus in high school affects performance in college economics courses. We have data on students' economics exam scores, their high school GPA, and whether they took calculus.

We estimate the model:

$$score = \beta_0 + \beta_1(hsgpa) + \beta_2(calculus) + \mu$$

where `calculus` equals 1 if the student took calculus and 0 otherwise.

```
lm1 <- lm(score ~ hsgpa + calculus, data = wooldridge::econmath)
summary(lm1)
```

```
Call:
lm(formula = score ~ hsgpa + calculus, data = wooldridge::econmath)

Residuals:
    Min      1Q  Median      3Q     Max
-54.355  -7.457   1.148   8.699  29.825

Coefficients:
            Estimate Std. Error t value             Pr(>|t|)
(Intercept)  24.5276     4.0759   6.018       0.000000002624 ***
hsgpa        13.2038     1.2257  10.772 < 0.0000000000000002 ***
calculus      5.8243     0.8992   6.477       0.000000000158 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.18 on 853 degrees of freedom
Multiple R-squared:  0.1756,    Adjusted R-squared:  0.1737
F-statistic: 90.87 on 2 and 853 DF,  p-value: < 0.00000000000000022
```

How do we interpret these coefficients? The coefficient on `hsgpa` ($\hat{\beta}_1 \approx 13.2$) tells us that a one-point increase in high school GPA is associated with about a 13-point increase in economics score, holding calculus status constant.

The coefficient on `calculus` ($\hat{\beta}_2 \approx 5.8$) tells us that students who took calculus score about 5.8 points higher on average than students who did not, holding GPA constant. This is the **difference in intercepts** between the two groups.

7.1.3 Visualizing What Dummy Variables Do

To understand what's happening geometrically, think about the regression line for each group.

For students who did **not** take calculus (`calculus = 0`):

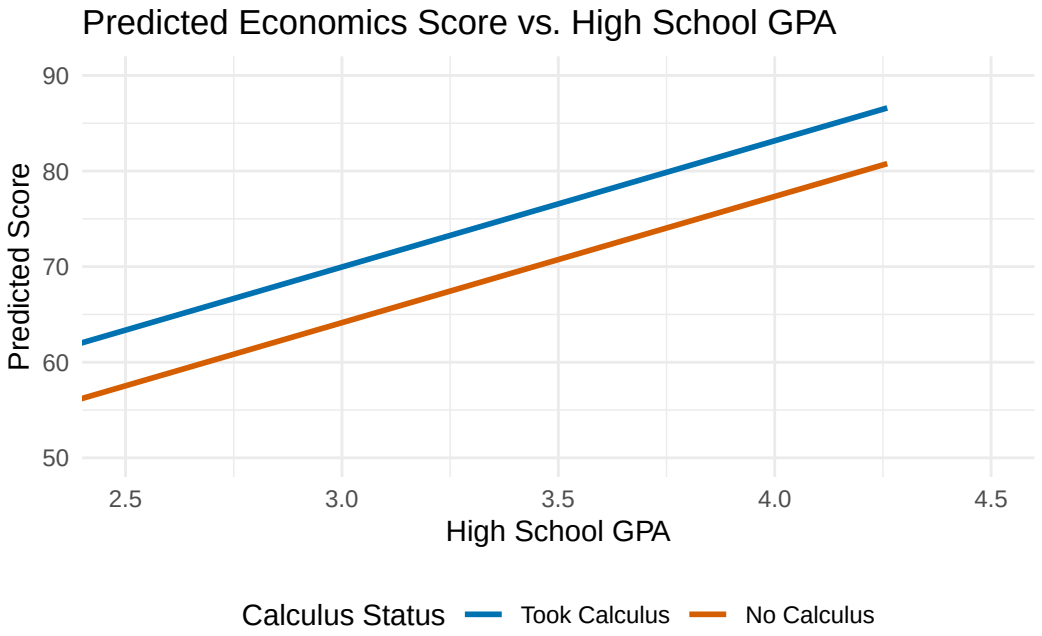
$$\widehat{score} = \hat{\beta}_0 + \hat{\beta}_1(hsgpa) + \hat{\beta}_2(0) = \hat{\beta}_0 + \hat{\beta}_1(hsgpa)$$

For students who **did** take calculus (`calculus = 1`):

$$\widehat{score} = \hat{\beta}_0 + \hat{\beta}_1(hsgpa) + \hat{\beta}_2(1) = (\hat{\beta}_0 + \hat{\beta}_2) + \hat{\beta}_1(hsgpa)$$

Both groups have the **same slope** ($\hat{\beta}_1$), but the calculus group has a higher intercept by exactly $\hat{\beta}_2$. The dummy variable shifts the regression line up or down:

Figure 7.1: Dummy variables shift the intercept while keeping the slope constant.



! Key Insight

A dummy variable coefficient represents the **difference in intercepts** between groups. Both groups share the same slope—the effect of increasing x by one unit is the same regardless of group membership.

7.2 The Dummy Variable Trap

You might wonder: why not include dummy variables for *both* calculus-takers and non-calculus-takers? In other words, why not estimate:

$$score = \beta_0 + \beta_1(hsgpa) + \beta_2(calculus) + \beta_3(no\_calculus) + \mu$$

The problem is that this creates **perfect multicollinearity**. For every student in the dataset:

$$calculus + no\_calculus = 1$$

This means $no\_calculus = 1 - calculus$, so one variable is a perfect linear function of the other. OLS cannot separately identify the effects of perfectly collinear variables.

This is called the **dummy variable trap**. To avoid it, we always omit one category. The omitted category becomes our **baseline** or **reference group**, and all coefficients are interpreted relative to this baseline.

In our example, non-calculus-takers are the baseline. The coefficient $\hat{\beta}_2 = 5.8$ means calculus-takers score 5.8 points higher *than non-calculus-takers*.

7.3 Multiple Categories

What if our categorical variable has more than two values? Consider the Palmer Penguins data, where species can be Adelie, Chinstrap, or Gentoo.

```
set.seed(12487345)

ex_data <- penguins |>
    sample_n(5) |>
    select(body_mass_g, flipper_length_mm, sex, species)

knitr::kable(ex_data, caption = "Palmer Penguins Sample")
```

Table 7.3: Palmer Penguins Sample

body_mass_g	flipper_length_mm	sex	species
3450	193	female	Adelie
3475	187	male	Adelie
4650	184	male	Adelie

body_mass_g	flipper_length_mm	sex	species
3650	187	female	Chinstrap
4600	209	female	Gentoo

To include species in a regression, we create $g - 1$ **dummy variables** where g is the number of categories. With three species, we need two dummy variables:

```
ex_data <- ex_data |>
    mutate(species_Chinstrap = case_when(species == "Chinstrap" ~ 1, TRUE ~ 0),
           species_Gentoo = case_when(species == "Gentoo" ~ 1, TRUE ~ 0))

knitr::kable(ex_data, caption = "Penguins with Dummy Variables")
```

Table 7.4: Penguins with Dummy Variables

body_mass_g	flipper_length_mm	sex	species	species_Chinstrap	species_Gentoo
3450	193	female	Adelie	0	0
3475	187	male	Adelie	0	0
4650	184	male	Adelie	0	0
3650	187	female	Chinstrap	1	0
4600	209	female	Gentoo	0	1

Notice that Adelie penguins have 0 for both dummies—they are the omitted baseline category.

Now we can estimate the effect of body mass on flipper length, controlling for species:

$$flipper\_length = \beta_0 + \beta_1(body\_mass) + \beta_2(Chinstrap) + \beta_3(Gentoo) + \mu$$

```
penguin_data <- penguins |>
    mutate(species_Chinstrap = case_when(species == "Chinstrap" ~ 1, TRUE ~ 0),
           species_Gentoo = case_when(species == "Gentoo" ~ 1, TRUE ~ 0))

lm_penguin <- lm(flipper_length_mm ~ body_mass_g + species_Chinstrap + species_Gentoo,
                 data = penguin_data)
summary(lm_penguin)
```

```
Call:
lm(formula = flipper_length_mm ~ body_mass_g + species_Chinstrap +
    species_Gentoo, data = penguin_data)

Residuals:
     Min       1Q   Median       3Q      Max
-14.5455  -3.1845   0.1307   3.3533  17.5313

Coefficients:
                     Estimate  Std. Error t value             Pr(>|t|)
(Intercept)       158.8602606   2.3865767  66.564 < 0.0000000000000002 ***
body_mass_g         0.0084021   0.0006339  13.255 < 0.0000000000000002 ***
species_Chinstrap   5.5974403   0.7882166   7.101      0.00000000000733 ***
species_Gentoo     15.6774699   1.0906591  14.374 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.395 on 338 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.8541,    Adjusted R-squared:  0.8528
F-statistic: 659.4 on 3 and 338 DF,  p-value: < 0.00000000000000022
```

Interpretation:

- $\hat{\beta}_1 \approx 0.008$: A 1-gram increase in body mass is associated with a 0.008 mm increase in flipper length, holding species constant.
- $\hat{\beta}_2 \approx 5.6$: Chinstrap penguins have flippers about 5.6 mm longer than Adelie penguins of the same body mass.
- $\hat{\beta}_3 \approx 15.7$: Gentoo penguins have flippers about 15.7 mm longer than Adelie penguins of the same body mass.

7.3.1 Choosing the Baseline Category

How should we choose which category to omit? It depends on our research question.

If we only care about controlling for the categorical variable (not interpreting its coefficients), then the choice doesn't matter—the coefficient on our main variable of interest won't change.

If we do care about interpreting the dummy coefficients, choose a baseline that makes interpretation natural. For studying the gender wage gap, making "male" the baseline means the female coefficient directly measures the wage gap relative to men.

Let's verify that changing the baseline doesn't affect other coefficients:

```r
# Switching baseline to Gentoo
penguin_data <- penguin_data |>
    mutate(species_Adelie = case_when(species == "Adelie" ~ 1, TRUE ~ 0))

lm_penguin2 <- lm(flipper_length_mm ~ body_mass_g + species_Chinstrap + species_Adelie,
                  data = penguin_data)

# Compare body mass coefficients
cat("Body mass coefficient (Adelie baseline):", round(coef(lm_penguin)["body_mass_g"], 5), "
```

```
Body mass coefficient (Adelie baseline): 0.0084
```

```r
cat("Body mass coefficient (Gentoo baseline):", round(coef(lm_penguin2)["body_mass_g"], 5))
```

```
Body mass coefficient (Gentoo baseline): 0.0084
```

The coefficient on body mass is identical regardless of which species is the baseline.

7.4 Factor Variables in R

Creating dummy variables manually is tedious, especially when categories are numerous. R's **factor** variables automate this process.

When you include a factor variable in a regression, R automatically creates the necessary dummy variables and omits one category:

```r
# Convert species to factor
penguin_data <- penguins |>
    mutate(species_f = factor(species))

# R handles dummy creation automatically
lm_factor <- lm(flipper_length_mm ~ body_mass_g + species_f, data = penguin_data)
summary(lm_factor)
```

```
Call:
lm(formula = flipper_length_mm ~ body_mass_g + species_f, data = penguin_data)

Residuals:
```

```
     Min       1Q   Median       3Q      Max
-14.5455  -3.1845   0.1307   3.3533  17.5313

Coefficients:
                     Estimate  Std. Error t value             Pr(>|t|)
(Intercept)        158.8602606   2.3865767  66.564 < 0.0000000000000002 ***
body_mass_g          0.0084021   0.0006339  13.255 < 0.0000000000000002 ***
species_fChinstrap   5.5974403   0.7882166   7.101      0.00000000000733 ***
species_fGentoo     15.6774699   1.0906591  14.374 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.395 on 338 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.8541,    Adjusted R-squared:  0.8528
F-statistic: 659.4 on 3 and 338 DF,  p-value: < 0.00000000000000022
```

Notice the coefficients are identical to our manual approach. R chose "Adelie" as the baseline because it comes first alphabetically.

To change the baseline category, use the `relevel()` function:

```r
# Make Gentoo the baseline
penguin_data <- penguin_data |>
    mutate(species_f = relevel(species_f, ref = "Gentoo"))

lm_factor2 <- lm(flipper_length_mm ~ body_mass_g + species_f, data = penguin_data)
summary(lm_factor2)
```

```
Call:
lm(formula = flipper_length_mm ~ body_mass_g + species_f, data = penguin_data)

Residuals:
     Min       1Q   Median       3Q      Max
-14.5455  -3.1845   0.1307   3.3533  17.5313

Coefficients:
                     Estimate  Std. Error t value             Pr(>|t|)
(Intercept)        174.5377305   3.2542422  53.634 < 0.0000000000000002 ***
body_mass_g          0.0084021   0.0006339  13.255 < 0.0000000000000002 ***
species_fAdelie    -15.6774699   1.0906591 -14.374 < 0.0000000000000002 ***
```

```
species_fChinstrap -10.0800297   1.1787380  -8.552 0.000000000000000428 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.395 on 338 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.8541,    Adjusted R-squared:  0.8528
F-statistic: 659.4 on 3 and 338 DF,  p-value: < 0.00000000000000022
```

Now coefficients are relative to Gentoo penguins.

7.5 Non-Linear Relationships: Polynomial Models

So far we've assumed linear relationships between variables. But many economic relationships are non-linear. Consider the effect of work experience on wages: at low experience levels, additional experience substantially increases productivity and wages; but at very high experience levels, workers may become less productive as they age.

This suggests a **quadratic** relationship—wages increase with experience at first, then eventually decrease. A quadratic model takes the general form:

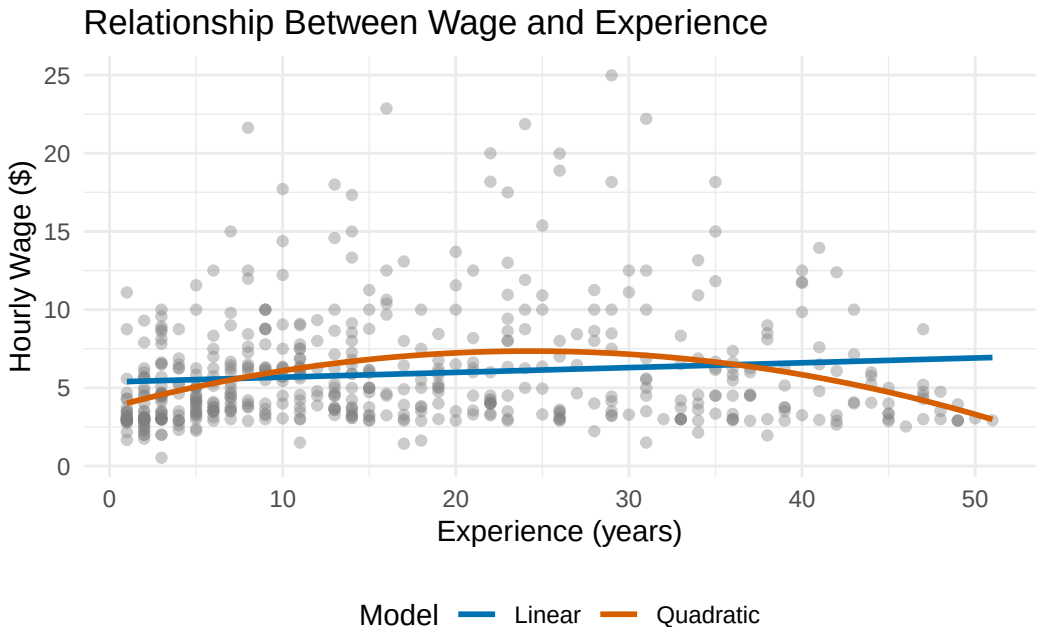
$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \mu$$

This is called a "quadratic" because it includes X raised to the second power (X^2). Recall from algebra that the graph of a quadratic function is a **parabola**—a U-shape or an inverted-U shape. The sign of β_2 determines the shape: if $\beta_2 > 0$, the parabola opens upward (U-shaped); if $\beta_2 < 0$, the parabola opens downward (inverted-U or "hump" shaped).

For the experience-wage relationship, we would expect $\beta_2 < 0$: wages rise with experience early in a career but eventually level off or decline. Let's see how a quadratic compares to a simple linear model:

Notice in Figure 7.2 that the linear model predicts a constant increase in wages for every additional year of experience. The quadratic model, by contrast, curves—it captures the fact that wages rise steeply at first and then flatten out. This is a much more realistic representation of the data.

Figure 7.2: A quadratic model (orange) captures the curvature in the experience-wage relationship that a linear model (blue) misses.



7.5.1 Estimating Quadratic Models

To estimate a quadratic model, we simply include both the variable and its square as regressors. For our wage-experience example, the model is:

$$wage = \beta_0 + \beta_1(exper) + \beta_2(exper^2) + \mu$$

In R, we create the squared term and include it as an additional variable:

```
wage_data <- wooldridge::wage1 |>
    mutate(exper2 = exper^2)

lm_quad <- lm(wage ~ exper + exper2, data = wage_data)
summary(lm_quad)
```

```
Call:
lm(formula = wage ~ exper + exper2, data = wage_data)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-5.5916 -2.1440 -0.8603  1.1801 17.7649

Coefficients:
              Estimate Std. Error t value             Pr(>|t|)
(Intercept)  3.7254058  0.3459392  10.769 < 0.0000000000000002 ***
exper        0.2981001  0.0409655   7.277      0.00000000000126 ***
exper2      -0.0061299  0.0009025  -6.792      0.00000000003015 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.524 on 523 degrees of freedom
Multiple R-squared:  0.09277,   Adjusted R-squared:  0.0893
F-statistic: 26.74 on 2 and 523 DF,  p-value: 0.000000000008774
```

Notice that both `exper` and `exper2` appear in the model. The positive coefficient on `exper` and the negative coefficient on `exper2` confirm the inverted-U shape we saw in the figure: wages increase with experience at first, but at a decreasing rate.

7.5.2 Interpreting Quadratic Models

With a quadratic term, the marginal effect of X on Y is **no longer constant**—it changes depending on the current value of X. This is the key difference from a linear model, where the marginal effect is always β_1 regardless of where you are.

To find the marginal effect in a quadratic model, we take the derivative of Y with respect to X:

$$\frac{\partial Y}{\partial X} = \beta_1 + 2\beta_2 X$$

The effect of one more year of experience **depends on how much experience you already have**.

Using our estimates ($\hat{\beta}_1 \approx 0.298$, $\hat{\beta}_2 \approx -0.006$):

For a worker with **5 years** of experience:

$$0.298 + 2(-0.006)(5) = 0.298 - 0.06 = 0.238$$

An additional year of experience is associated with a \$0.24 increase in hourly wages.

For a worker with **25 years** of experience:

$$0.298 + 2(-0.006)(25) = 0.298 - 0.30 = -0.002$$

An additional year of experience has essentially zero effect on wages—this worker is near the peak.

For a worker with **40 years** of experience:

$$0.298 + 2(-0.006)(40) = 0.298 - 0.48 = -0.182$$

An additional year actually *decreases* wages by about \$0.18—the worker is past the peak of the experience-wage relationship.

i Finding the Peak

The experience level where wages are maximized occurs where the marginal effect equals zero:

$$\beta_1 + 2\beta_2 x^* = 0 \implies x^* = -\frac{\beta_1}{2\beta_2}$$

With our estimates: $x^* = -\frac{0.298}{2(-0.006)} \approx 24.8$ years. This tells us that, according to our model, wages peak at about 25 years of experience.
We can verify this in R:

```
peak_exper <- -coef(lm_quad)["exper"] / (2 * coef(lm_quad)["exper2"])
cat("Estimated peak experience:", round(peak_exper, 1), "years\n")
```

```
Estimated peak experience: 24.3 years
```

7.6 Logarithmic Transformations

Up to this point, all of our regression models have used variables in their original units—dollars, years, grams, and so on. But economists very frequently use the **natural logarithm** (denoted ln or log) to transform variables before putting them into a regression. Why would we do this?

There are two main motivations. First, many economic variables—like wages, income, prices, and population—are **right-skewed**: most observations cluster at lower values, but a long tail stretches to the right. Taking the log "compresses" this tail and makes the distribution more symmetric, which often produces a better-fitting model. Second, and more importantly for interpretation, using logs lets us talk about **percentage changes** rather than absolute changes, which is often more economically meaningful. A \$1 raise means very different things to someone earning \$10/hour versus \$100/hour, but a 10% raise is comparable across both.

7.6.1 Quick Review: What Is a Logarithm?

Recall that the natural logarithm is the inverse of the exponential function. If $e^a = b$, then $\ln(b) = a$, where $e \approx 2.718$.

A key property that makes logs so useful in regression is:

$$\ln(Y_1) - \ln(Y_0) = \ln\left(\frac{Y_1}{Y_0}\right)$$

This means that the **difference in logs** is closely related to the **percentage change**. Specifically, for small changes:

$$\ln(Y_1) - \ln(Y_0) \approx \frac{Y_1 - Y_0}{Y_0} = \text{proportional change in } Y$$

This approximation works because of a calculus result: the derivative of $\ln(Y)$ with respect to Y is $1/Y$, so $d\ln(Y) = dY/Y$—a small change in the log of Y equals the proportional change in Y itself. Multiplying by 100 gives us the **percent change**.

> **i** Why This Approximation Works
>
> Consider a wage increase from \$20 to \$21. The proportional change is $(21-20)/20 = 0.05$, or 5%. The difference in logs is $\ln(21) - \ln(20) = 3.045 - 2.996 = 0.049$, which is very close to 0.05.
> But for larger changes the approximation gets worse. A change from \$20 to \$30 is a 50% increase, but $\ln(30) - \ln(20) = 0.405$, not 0.50. We'll discuss what to do about larger changes below.

7.6.2 Log-Linear Models

The most common log transformation in applied economics is to log only the dependent variable. This gives us the **log-linear** model:

$$\ln(Y) = \beta_0 + \beta_1 X + \mu$$

Here the left-hand side is in logs but the right-hand side variable X is in its original units (levels).

Deriving the interpretation. To see why logs change the interpretation, suppose X increases by one unit (from X_0 to $X_0 + 1$), holding everything else constant. Then:

$$\ln(Y_1) = \beta_0 + \beta_1(X_0 + 1)$$
$$\ln(Y_0) = \beta_0 + \beta_1(X_0)$$

Subtracting:

$$\ln(Y_1) - \ln(Y_0) = \beta_1$$

Now recall that $\ln(Y_1) - \ln(Y_0) \approx (Y_1 - Y_0)/Y_0$ for small changes. So:

$$\frac{Y_1 - Y_0}{Y_0} \approx \beta_1$$

Multiplying both sides by 100:

$$\text{Percent change in } Y \approx 100 \times \beta_1$$

Interpretation: A one-unit increase in X is associated with an approximately $100 \times \beta_1$ percent change in Y.

Let's see this in action with our wage data. We'll estimate a model with log wages as the dependent variable and education as the independent variable:

```
lm_loglin <- lm(log(wage) ~ educ, data = wooldridge::wage1)
summary(lm_loglin)
```

```
Call:
lm(formula = log(wage) ~ educ, data = wooldridge::wage1)

Residuals:
     Min       1Q   Median       3Q      Max
-2.21158 -0.36393 -0.07263  0.29712  1.52339

Coefficients:
            Estimate Std. Error t value             Pr(>|t|)
(Intercept) 0.583773   0.097336   5.998        0.00000000374 ***
educ        0.082744   0.007567  10.935 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.4801 on 524 degrees of freedom
Multiple R-squared:  0.1858,    Adjusted R-squared:  0.1843
F-statistic: 119.6 on 1 and 524 DF,  p-value: < 0.00000000000000022
```

The coefficient on `educ` is approximately 0.083. Multiplying by 100 gives 8.3. So we interpret this as: *an additional year of education is associated with approximately an 8.3% increase in hourly wages.*

Notice how different this is from a level-level model, where we would say "an additional year of education is associated with a $X increase in wages." The percentage interpretation is arguably more natural for wages, since a $1 raise means different things at different wage levels.

> ⚠ When the Approximation Breaks Down
>
> The "$100 \times \beta_1$ percent" interpretation is an approximation that works well when β_1 is small—roughly when $|\beta_1| < 0.20$.
> For larger coefficients, the approximation becomes inaccurate. In those cases, use the **exact** formula:
>
> $$\text{Exact percent change} = 100 \times (e^{\beta_1} - 1)$$
>
> For example, if $\hat{\beta}_1 = 0.40$, the approximate interpretation would be a 40% change, but the exact calculation gives $100 \times (e^{0.40} - 1) = 100 \times (1.492 - 1) = 49.2\%$. That's a meaningful difference!
> This exact formula is particularly important when interpreting dummy variable coefficients in log-linear models, since those coefficients are often large enough for the approximation to be poor.

7.6.3 Linear-Log Models

We can also log an independent variable while keeping the dependent variable in levels:

$$Y = \beta_0 + \beta_1 \ln(X) + \mu$$

Deriving the interpretation. Now suppose X increases by 1% (i.e., X goes from X_0 to $1.01 \times X_0$). The change in $\ln(X)$ is:

$$\ln(1.01 \times X_0) - \ln(X_0) = \ln(1.01) \approx 0.01$$

So a 1% change in X changes $\ln(X)$ by approximately 0.01. The resulting change in Y is:

$$\Delta Y = \beta_1 \times 0.01 = \frac{\beta_1}{100}$$

Interpretation: A 1% increase in X is associated with a $\beta_1/100$ unit change in Y.

This functional form is useful when X varies over a wide range and we believe that proportional changes in X matter more than absolute changes. For instance, if we regress birthweight in grams on the log of family income, a 1% increase in income might be associated with a $\beta_1/100$-gram increase in birthweight—regardless of whether the family earns $20,000 or $200,000.

```
bwght_data <- wooldridge::bwght |>
    filter(faminc > 0)  # drop zeros since log(0) is undefined

lm_linlog <- lm(bwght ~ log(faminc), data = bwght_data)
summary(lm_linlog)
```

```
Call:
lm(formula = bwght ~ log(faminc), data = bwght_data)

Residuals:
    Min      1Q  Median      3Q     Max
-95.799 -11.799   0.621  13.014 150.708

Coefficients:
            Estimate Std. Error t value             Pr(>|t|)
(Intercept) 111.4881     1.8982  58.735 < 0.0000000000000002 ***
log(faminc)   2.3481     0.5922   3.965            0.0000771 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 20.25 on 1386 degrees of freedom
Multiple R-squared:  0.01122,   Adjusted R-squared:  0.0105
F-statistic: 15.72 on 1 and 1386 DF,  p-value: 0.00007708
```

The coefficient on `log(faminc)` is approximately 2.47. This means a 1% increase in family income is associated with a $2.47/100 = 0.025$ ounce increase in birthweight. Or equivalently, a 10% increase in family income is associated with about a 0.25 ounce increase in birthweight.

7.6.4 Log-Log Models (Elasticities)

When both sides are logged:

$$\ln(Y) = \beta_0 + \beta_1 \ln(X) + \mu$$

Deriving the interpretation. A 1% increase in X changes $\ln(X)$ by approximately 0.01 (as we derived above). The change in $\ln(Y)$ is then:

$$\Delta \ln(Y) = \beta_1 \times 0.01$$

Since $\Delta \ln(Y) \approx$ proportional change in Y, multiplying by 100:

$$\text{Percent change in } Y \approx \beta_1 \times 0.01 \times 100 = \beta_1 \times 1\%$$

So: a 1% increase in X is associated with a β_1% change in Y.

Interpretation: β_1 is the **elasticity** of Y with respect to X—a unit-free measure of how responsive Y is to changes in X. Elasticities are a cornerstone of economics. If $\beta_1 = -0.5$, for example, we would say that Y is "inelastic" with respect to X—a 1% increase in X leads to only a 0.5% decrease in Y.

```
wage_educ <- wooldridge::wage1 |>
    filter(educ > 0)  # drop zeros since log(0) is undefined

lm_loglog <- lm(log(wage) ~ log(educ), data = wage_educ)
summary(lm_loglog)
```

```
Call:
lm(formula = log(wage) ~ log(educ), data = wage_educ)

Residuals:
     Min       1Q   Median       3Q      Max
-2.24076 -0.38426 -0.05421  0.32273  1.49421

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)
(Intercept) -0.44468    0.21785  -2.041              0.0417 *
log(educ)    0.82521    0.08645   9.546 <0.0000000000000002 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4913 on 522 degrees of freedom
Multiple R-squared:  0.1486,    Adjusted R-squared:  0.147
F-statistic: 91.12 on 1 and 522 DF,  p-value: < 0.00000000000000022
```

The coefficient is approximately 0.825, meaning a 1% increase in education is associated with approximately a 0.825% increase in wages. Education and wages are inelastic in this specification—a 1% increase in education corresponds to less than a 1% increase in wages.

7.6.5 When to Use Logs

There are no hard-and-fast rules, but here are some practical guidelines:

Variables that are commonly logged include wages, income, prices, population, firm size (revenue, employees), and expenditures. These tend to be strictly positive, right-skewed, and best thought of in percentage terms.

Variables that are typically **not** logged include years of education, age, percentages or rates (unemployment rate, graduation rate), dummy variables, and variables that can take zero or negative values (since the log of zero or a negative number is undefined).

> Practical Advice
>
> When in doubt, try plotting the variable. If its distribution is right-skewed and the variable is strictly positive, logging it is often a good idea. You can also compare the R^2 of models with and without the log transformation—though model fit alone shouldn't be the sole criterion. The most important consideration is whether the interpretation makes economic sense.

7.7 Application: Policy Evaluation with Dummy Variables

Dummy variables are essential for evaluating policies and programs. Suppose we want to estimate the effect of a job training program on wages:

$$\ln(wage) = \beta_0 + \beta_1(training) + \beta_2(educ) + \beta_3(exper) + \mu$$

where `training` equals 1 if the person participated in the program and 0 otherwise.

The coefficient β_1 measures the **treatment effect**—the percentage difference in wages between program participants and non-participants, controlling for education and experience. Since the

dependent variable is logged, we multiply β_1 by 100 to get the approximate percentage effect (or use the exact formula $100 \times (e^{\beta_1} - 1)$ for larger coefficients).

This is essentially a regression-based version of comparing treatment and control groups, where the control variables help us account for differences between those who did and didn't participate.

7.8 Functional Form Interpretation Reference

The table below summarizes how to interpret β_1 under each functional form we have covered. This is one of the most important references in the course—you should be able to quickly identify the correct interpretation for any model you encounter.

Table 7.5: Functional Form Interpretation Guide

Model	Equation	Interpretation of $\widehat{\beta}_1$	Example
Level-Level	$Y = \beta_0 + \beta_1 X$	A 1-unit increase in $X \to \beta_1$-unit change in Y	$wage = 2.1 + 0.54 \cdot educ$: one more year of education → \$0.54 higher wages
Log-Level	$\ln(Y) = \beta_0 + \beta_1 X$	A 1-unit increase in $X \to (100 \times \beta_1)\%$ change in Y	$\ln(wage) = 0.58 + 0.083 \cdot educ$: one more year of education → 8.3% higher wages
Level-Log	$Y = \beta_0 + \beta_1 \ln(X)$	A 1% increase in $X \to \beta_1/100$ unit change in Y	$bwght = 116.1 + 2.47 \cdot \ln(faminc)$: 1% higher income → 0.025 oz higher birthweight

Model	Equation	Interpretation of $\hat{\beta}_1$	Example
Log-Log	$\ln(Y) = \beta_0 + \beta_1 \ln(X)$	A 1% increase in $X \to \beta_1$% change in Y (elasticity)	$\ln(wage) = -0.02 + 0.83 \cdot \ln(educ)$: 1% more education $\to$ 0.83% higher wages
Quadratic	$Y = \beta_0 + \beta_1 X + \beta_2 X^2$	Marginal effect is $\beta_1 + 2\beta_2 X$ (depends on X)	$wage = 3.73 + 0.30 \cdot exper - 0.006 \cdot exper^2$: at 10 yrs $\to$ +\$0.18; at 30 yrs $\to$ −\$0.06
Dummy	$Y = \beta_0 + \beta_1 D$	Difference in Y between $D = 1$ group and baseline	$score = 24.5 + 5.8 \cdot calculus$: calculus-takers score 5.8 pts higher
Dummy in Log	$\ln(Y) = \beta_0 + \beta_1 D$	$D = 1$ group has $(100 \times \beta_1)$% different Y. Exact: $100(e^{\beta_1} - 1)$%	$\ln(wage) = 1.0 - 0.15 \cdot female$: women earn 15% less (exact: −13.9%)

How to Use This Table

When you encounter a regression in a paper or problem set, the first thing to check is: **is the dependent variable logged? Is the independent variable logged?** Your answers determine which row of the table applies. Then use the interpretation column to state the result in plain English.

7.9 Summary

This chapter introduced two powerful extensions to basic regression: categorical variables and non-linear functional forms.

Dummy variables allow us to include qualitative information in regression models. The coefficient on a dummy variable represents the difference in intercepts between groups—the effect of having a particular characteristic relative to the baseline category. When a categorical variable has g categories, we include $g - 1$ dummy variables to avoid perfect multicollinearity (the dummy variable trap).

Polynomial models capture non-linear relationships by including squared or higher-order terms. With quadratic models, the marginal effect of x on y depends on the current level of x, allowing us to model relationships that increase and then decrease (or vice versa).

Logarithmic transformations change the interpretation of coefficients. The key insight is that a change in a logged variable approximates a proportional (percentage) change. Log-linear models give the percentage change in Y for a one-unit change in X; linear-log models give the unit change in Y for a one-percent change in X; and log-log models give elasticities—the percentage change in Y for a one-percent change in X. These transformations are especially useful when dealing with variables like wages or prices that are right-skewed and where percentage changes are more meaningful than absolute changes. See Table 7.5 for a complete reference.

7.10 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

8 Heteroskedasticity

Key Questions

- What is homoskedasticity and why is it a Gauss-Markov assumption?
- What happens to OLS estimates when heteroskedasticity is present?
- How does heteroskedasticity affect our hypothesis tests?
- What are heteroskedasticity-robust standard errors and when should we use them?
- How can we detect heteroskedasticity in our data?

Suggested Readings

- Wooldridge (2019), Ch. 8

One of the key assumptions underlying OLS is that the variance of the error term is constant across all values of the explanatory variables. When this assumption fails, we have **heteroskedasticity**—and while our coefficient estimates remain unbiased, our standard errors become unreliable. This chapter explores what heteroskedasticity is, why it matters, and how to fix it.

8.1 The Homoskedasticity Assumption

8.1.1 Gauss-Markov Assumption 5

Recall the Gauss-Markov assumptions that make OLS the Best Linear Unbiased Estimator (BLUE). The fifth assumption is **homoskedasticity**:

Homoskedasticity

The error term μ has the **same variance** given any value of the explanatory variables:

$$Var(\mu|x_1, x_2, ..., x_k) = \sigma^2$$

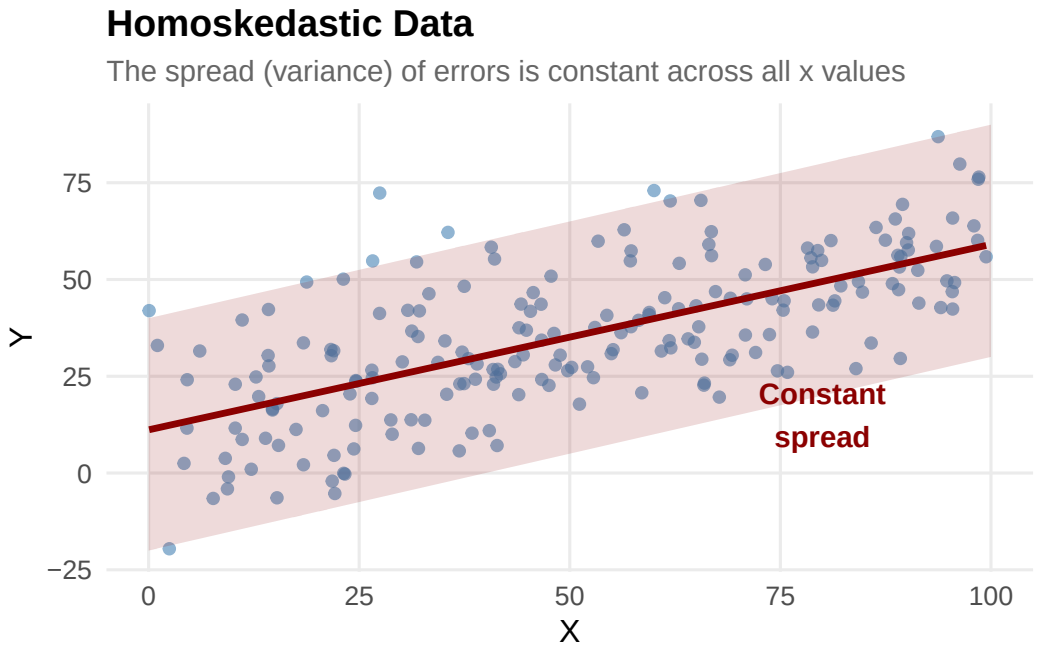
The variance is constant—it doesn't depend on x.

This assumption is crucial because it allows us to derive the correct formula for the variance of $\hat{\beta}$, which in turn gives us valid standard errors, t-statistics, and p-values.

8.1.2 What Does Homoskedasticity Look Like?

In a homoskedastic regression, the spread of observations around the regression line is **constant** across all values of x.

Figure 8.1: Homoskedastic data: the spread of points around the regression line is constant across all values of x.



Notice how the "cone" of data points has roughly the same width whether x is small or large. This is homoskedasticity in action.

8.2 Heteroskedasticity: When Variance Changes

8.2.1 What Is Heteroskedasticity?

Heteroskedasticity occurs when the variance of the error term changes with the level of the explanatory variable:

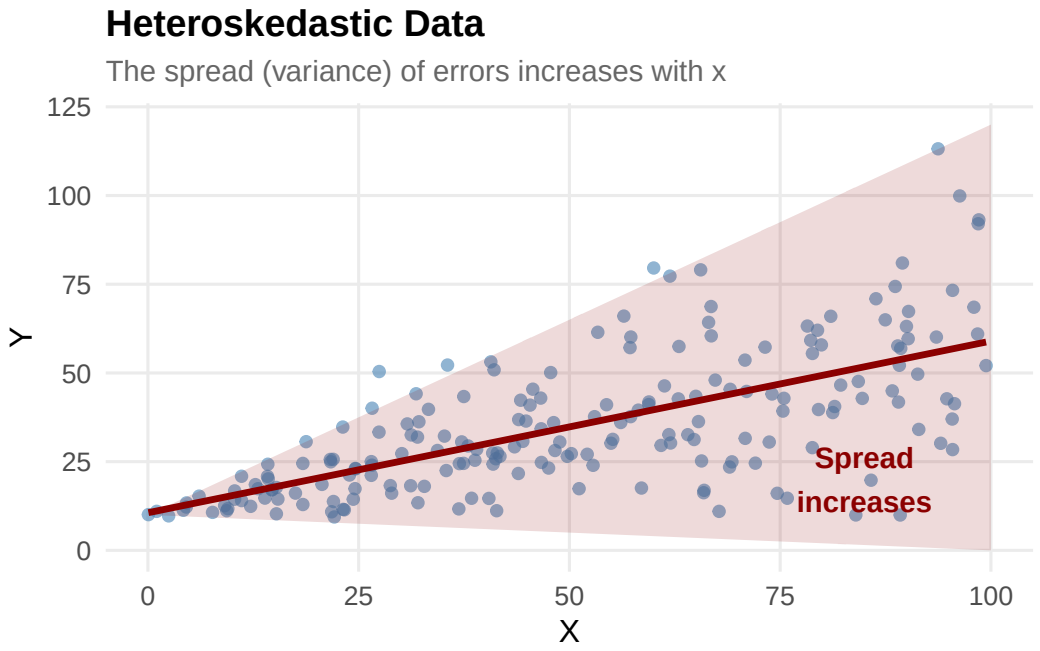
$$Var(\mu_i|x_i) = \sigma_i^2$$

The subscript i indicates that variance depends on observation i's value of x.

8.2.2 What Does Heteroskedasticity Look Like?

The classic pattern is a "fan" or "cone" shape, where the spread of data increases (or decreases) as x changes:

Figure 8.2: Heteroskedastic data: the spread of points around the regression line increases with x, creating a 'fan' or 'cone' pattern.



This pattern is extremely common in economic data. For example:

- **Income and consumption:** Higher-income households have more variable consumption patterns
- **Firm size and profits:** Larger firms have more variable profit margins
- **Experience and wages:** More experienced workers have more variable wages

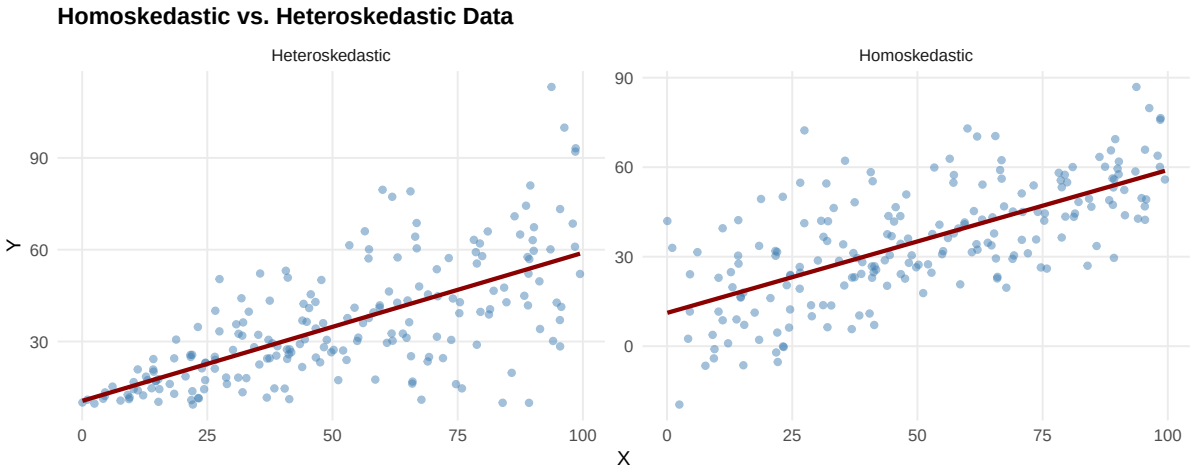
8.2.3 Side-by-Side Comparison

8.3 Why Does Heteroskedasticity Matter?

8.3.1 The Good News: Coefficients Are Still Unbiased

Here's the crucial point: **heteroskedasticity does NOT bias our coefficient estimates**. The unbiasedness of OLS depends only on Gauss-Markov assumptions 1-4, not on homoskedasticity.

Figure 8.3: Comparing homoskedastic (left) and heteroskedastic (right) data. The key difference is whether the spread of points changes with x.



So $E[\hat{\beta}] = \beta$ still holds. Our estimates are centered on the truth.

8.3.2 The Bad News: Standard Errors Are Wrong

However, the standard error formulas we've been using assume homoskedasticity. When this assumption is violated:

- The formula $SE(\hat{\beta}_1) = \frac{\hat{\sigma}}{\sqrt{SST_x}}$ is **invalid**
- Our computed standard errors will typically be **too small**
- This means t-statistics are **too large**
- And p-values are **too small**

The result? **We reject the null hypothesis too often**, even when it's true.

8.3.3 Type I Error: A Quick Review

> **!** Type I Error
>
> A **Type I error** (false positive) occurs when we reject a true null hypothesis.
> When we set $\alpha = 0.05$, we're accepting a 5% probability of making this error. If our test is working correctly, we should reject true null hypotheses exactly 5% of the time.

With heteroskedasticity and standard OLS standard errors, the actual Type I error rate can be much higher than 5%—sometimes 10%, 15%, or even higher.

8.4 Simulation: Seeing the Problem

8.4.1 Setup: Simulating Hypothesis Tests

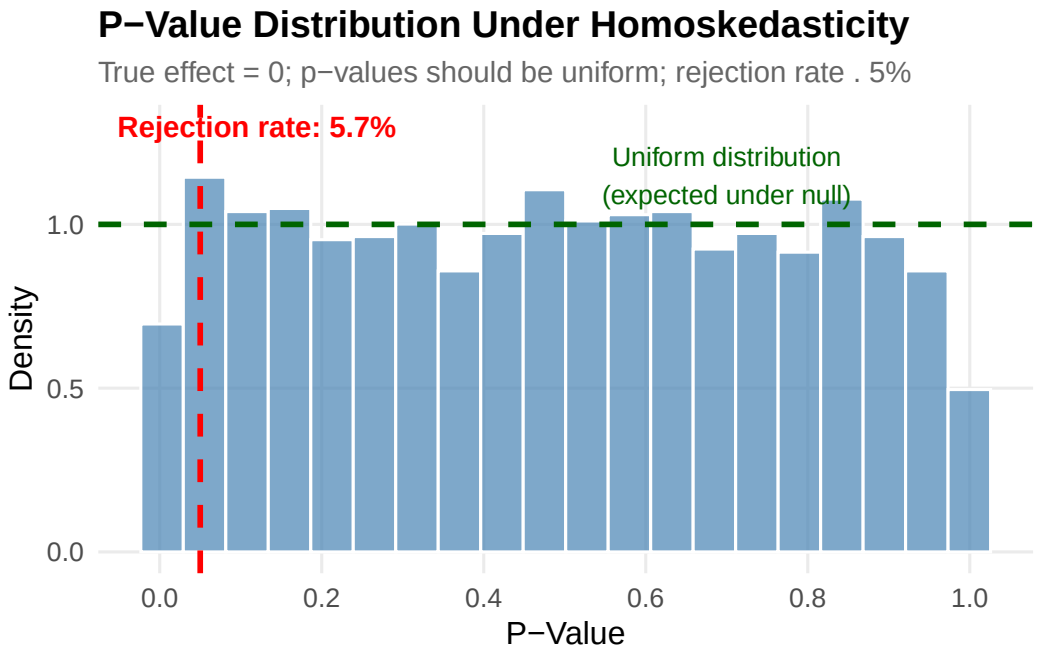
Let's run a simulation to see exactly what goes wrong. We'll generate data where the **true effect is zero** ($\beta_1 = 0$), then see how often we incorrectly reject this true null hypothesis.

First, let's see what happens under **homoskedasticity** (where everything works correctly):

```
# Simulation parameters
n_sims <- 2000       # Number of simulations
n_obs <- 250         # Observations per sample
beta0_true <- 5      # True intercept
beta1_true <- 0      # True slope (NULL HYPOTHESIS IS TRUE!)

set.seed(42)

# Function for ONE simulation trial under HOMOSKEDASTICITY
run_homo_trial <- function(i) {
    x <- runif(n_obs, 1, 50)
    error <- rnorm(n_obs, mean = 0, sd = 10)  # Constant variance
    y <- beta0_true + beta1_true * x + error

    model <- lm(y ~ x)

    # Get OLS p-value
    pval_ols <- summary(model)$coefficients["x", "Pr(>|t|)"]

    return(data.frame(pval_ols = pval_ols))
}

# Run simulations
homo_results <- map_dfr(1:n_sims, run_homo_trial)

# Calculate rejection rate (Type I error rate)
homo_reject_rate <- mean(homo_results$pval_ols < 0.05)
```

Under homoskedasticity, the Type I error rate is 5.7%—almost exactly 5%, as expected.

Figure 8.4: Under homoskedasticity, p-values are uniformly distributed when the null is true, and the rejection rate matches our significance level.



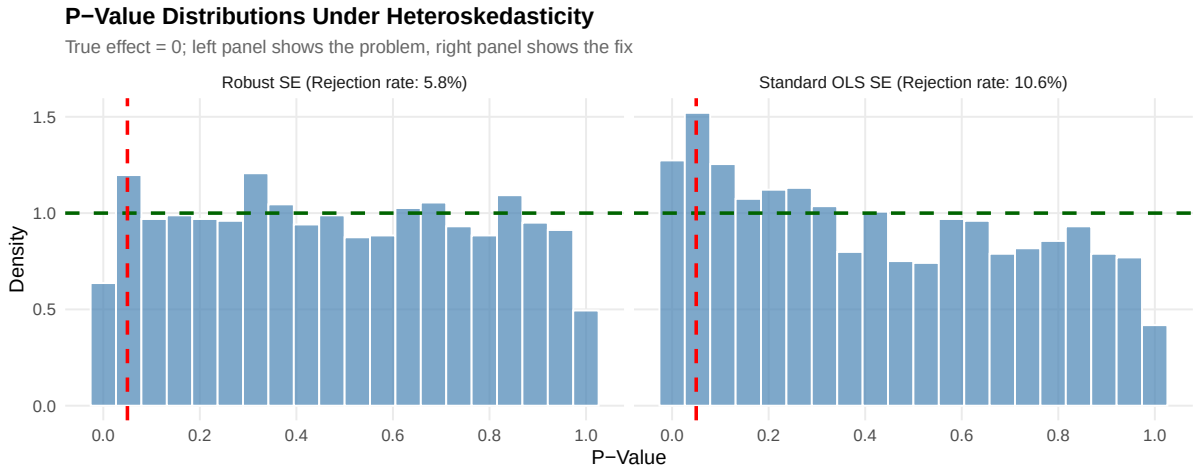
8.4.2 Now With Heteroskedasticity

Now let's see what happens when we have heteroskedasticity but use standard OLS standard errors:

```
# Function for ONE simulation trial under HETEROSKEDASTICITY
run_hetero_trial <- function(i) {
    x <- runif(n_obs, 1, 50)
    # Variance INCREASES with x (heteroskedasticity!)
    error <- rnorm(n_obs, mean = 0, sd = sqrt(0.2 * x^3))
    y <- beta0_true + beta1_true * x + error

    model <- lm(y ~ x)

    # Get both OLS and robust p-values
    res_ols <- coeftest(model)
    res_robust <- coeftest(model, vcov. = vcovHC(model, type = "HC1"))

    return(data.frame(
        pval_ols = res_ols["x", "Pr(>|t|)"],
        pval_robust = res_robust["x", "Pr(>|t|)"]
```

```
    ))
}

# Run simulations
hetero_results <- map_dfr(1:n_sims, run_hetero_trial)

# Calculate rejection rates
ols_reject_rate <- mean(hetero_results$pval_ols < 0.05)
robust_reject_rate <- mean(hetero_results$pval_robust < 0.05)
```

Figure 8.5: Under heteroskedasticity, standard OLS p-values pile up near zero, causing excessive rejections. Robust standard errors fix this problem.



The difference is stark! With standard OLS standard errors (left panel), we reject the true null hypothesis 10.6% of the time—more than **twice** our intended 5% rate. This is a serious problem: we're finding "significant" effects that don't exist.

With heteroskedasticity-robust standard errors (right panel), the rejection rate returns to approximately 5%.

8.5 Heteroskedasticity-Robust Standard Errors

8.5.1 The Solution

The fix is remarkably simple: use a different formula for the standard errors that accounts for heteroskedasticity. These are called **heteroskedasticity-robust standard errors** (also known as White standard errors or Huber-White standard errors).

The standard OLS variance formula is:

$$Var(\hat{\beta}_1) = \frac{\sum_{i=1}^{n} \hat{u}_i^2/(n-2)}{SST_x^2}$$

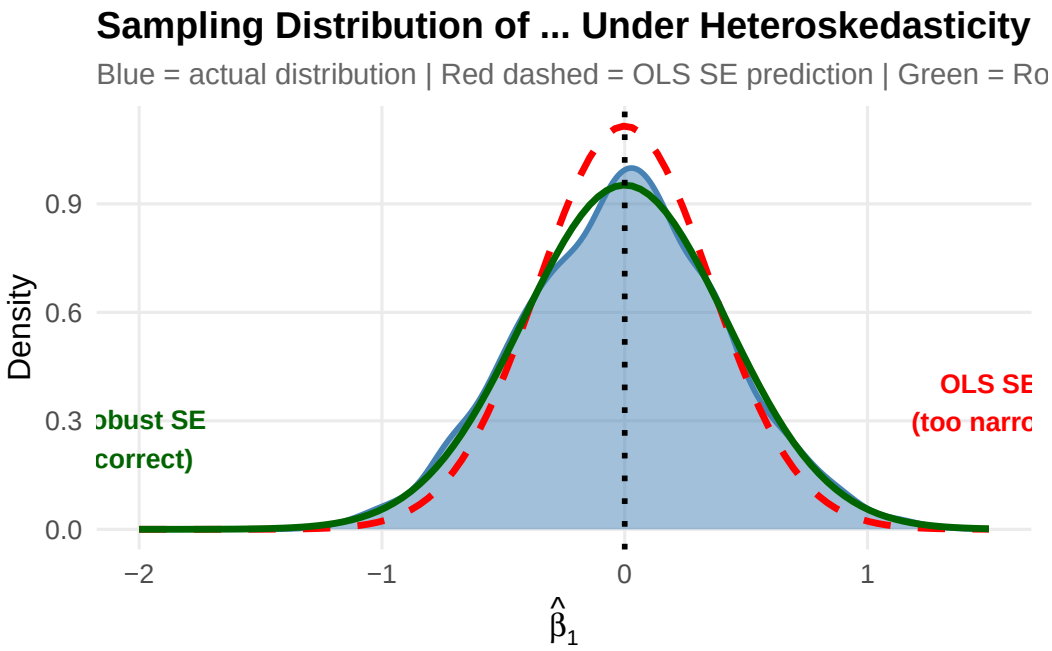
The heteroskedasticity-robust formula is:

$$Var(\hat{\beta}_1) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2 \hat{u}_i^2}{SST_x^2}$$

The key difference is that robust standard errors **weight each squared residual by how far its x_i is from the mean**. This accounts for the fact that residuals may have different variances at different values of x.

8.5.2 Visualizing the Sampling Distributions

Figure 8.6: The actual sampling distribution (blue) compared to what standard OLS SEs predict (red dashed) vs. robust SEs (green). OLS SEs are too narrow; robust SEs match reality.

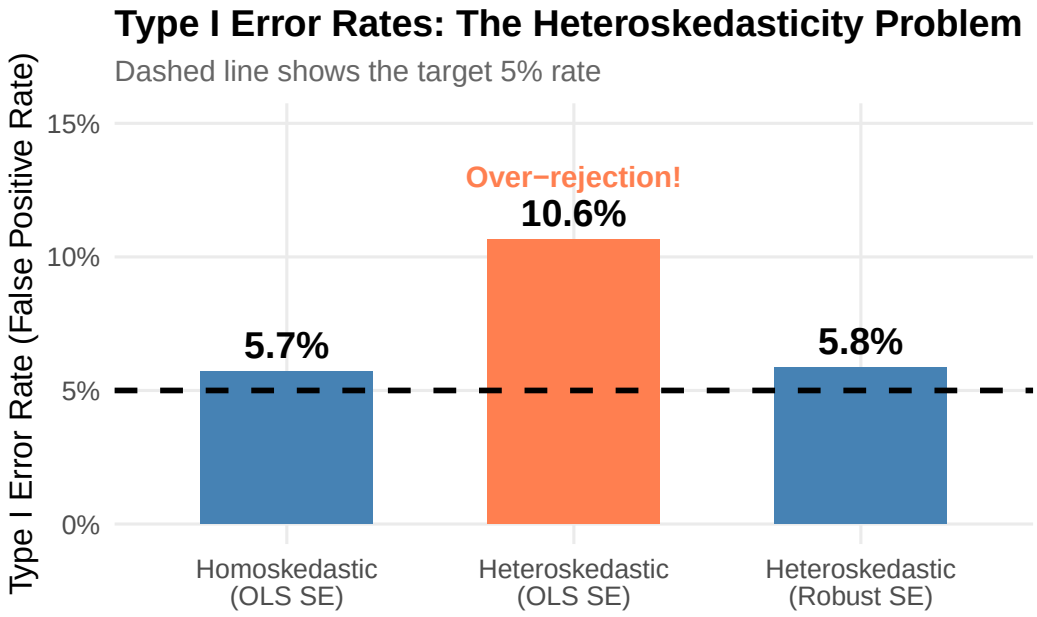


The figure shows that:

- The **actual** sampling distribution (blue) has heavier tails than OLS standard errors predict
- Standard OLS SEs (red dashed) assume a narrower distribution than reality
- Robust SEs (green) correctly capture the true variability

8.5.3 Comparing Type I Error Rates

Figure 8.7: Type I error rates across different scenarios. Robust standard errors maintain the correct 5% rate even with heteroskedasticity.



8.6 Implementing Robust Standard Errors in R

8.6.1 Using `fixest`

The easiest way to get robust standard errors is with the `fixest` package, which we'll use throughout this course:

```
# Generate some heteroskedastic data
set.seed(123)
example_data <- tibble(
    x = runif(200, 1, 50),
    y = 5 + 0.5 * x + rnorm(200, 0, sqrt(0.1 * x^2))
)

# Standard OLS (wrong SEs with heteroskedasticity)
model_ols <- feols(y ~ x, data = example_data)
```

	Standard SE	Robust SE
(Intercept)	5.353***	5.353***
	(1.216)	(0.906)
x	0.482***	0.482***
	(0.042)	(0.045)
Num.Obs.	200	200
R2	0.402	0.402

+ p <0.1, * p <0.05, ** p <0.01, *** p <0.001

```
# With heteroskedasticity-robust SEs
model_robust <- feols(y ~ x, vcov = "HC1", data = example_data)
```

```
# Compare the results
modelsummary(
    list("Standard SE" = model_ols, "Robust SE" = model_robust),
    stars = TRUE,
    gof_map = c("nobs", "r.squared")
)
```

Notice that:

- The **coefficients are identical** (unbiasedness is not affected)
- The **standard errors differ** (robust SEs are typically larger)
- This affects **t-statistics and p-values**

8.6.2 Best Practice: Always Use Robust Standard Errors

Best Practice

Always report heteroskedasticity-robust standard errors.
Why? Because:

1. If heteroskedasticity is present, robust SEs are correct and standard SEs are wrong
2. If homoskedasticity holds, robust SEs are still valid (just slightly less efficient)

You can't lose by using robust SEs, but you can lose badly by not using them.

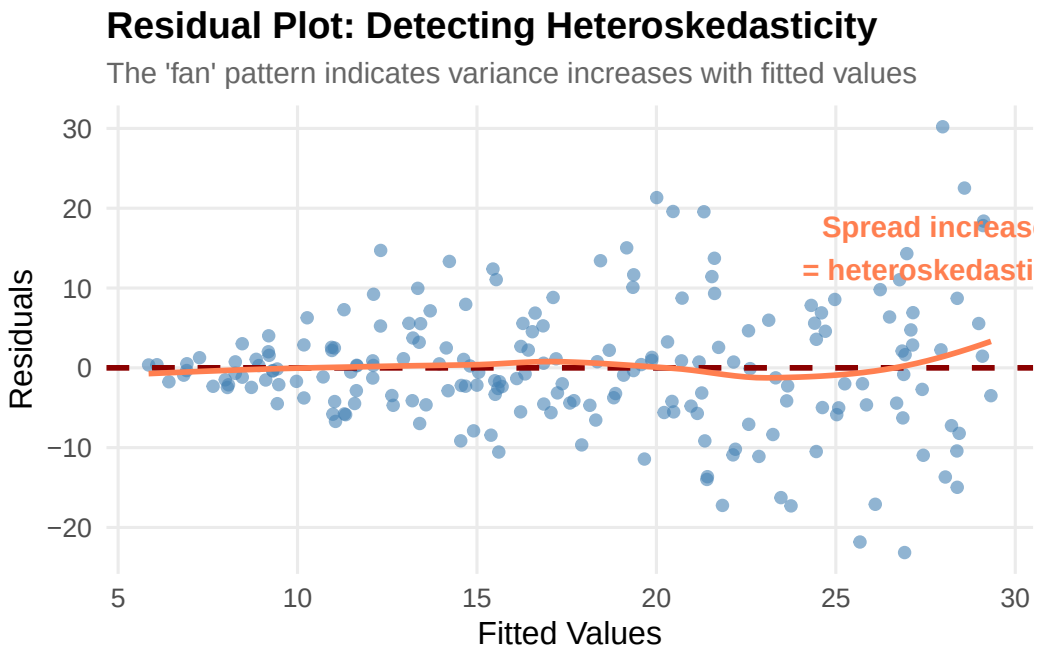
8.7 Detecting Heteroskedasticity

8.7.1 Residual Plots

The best way to detect heteroskedasticity is to **look at your residuals**. Plot the residuals against the fitted values (or against each explanatory variable) and look for patterns:

```
# Fit model to heteroskedastic data
model <- lm(y ~ x, data = example_data)

# Add residuals and fitted values to data
example_data <- example_data |>
    mutate(
        fitted = fitted(model),
        resid = residuals(model)
    )
```

Figure 8.8: Residual plot showing the 'fan' pattern characteristic of heteroskedasticity. The spread of residuals increases with fitted values.



8.7.2 What to Look For

Homoskedasticity: Residuals form a random cloud with constant spread

Heteroskedasticity: Look for:

- Fan/cone shapes (spread increases or decreases)
- Patterns in the spread related to fitted values
- Different variability for different groups

8.7.3 Formal Tests

While visual inspection is usually sufficient, you can also use formal tests like the **Breusch-Pagan test**:

```
# Breusch-Pagan test
bptest(model)
```

```
    studentized Breusch-Pagan test

data:  model
BP = 29.91, df = 1, p-value = 0.00000004526
```

A small p-value (< 0.05) suggests heteroskedasticity is present. However, **don't rely solely on formal tests**—always look at your residual plots too.

8.8 Summary

Heteroskedasticity is a common issue in economic data where the variance of errors changes with the explanatory variables.

Key Points:

1. **Heteroskedasticity does NOT bias coefficients** - $E[\hat{\beta}] = \beta$ still holds
2. **Heteroskedasticity DOES invalidate standard errors** - leading to incorrect inference
3. **The main symptom is inflated Type I error rates** - we reject true nulls too often
4. **The solution is simple: use robust standard errors** - specify `vcov = "HC1"` in `feols()`
5. **Detect heteroskedasticity with residual plots** - look for fan/cone patterns

Best Practice: Always use heteroskedasticity-robust standard errors. They're valid whether or not heteroskedasticity is present, so there's no downside to using them.

8.9 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

9 Binary Outcomes

Key Questions

- What happens when our dependent variable is binary (0 or 1) rather than continuous?
- How do we interpret OLS coefficients in the linear probability model?
- What problems arise when we use OLS with a binary outcome?
- What is logistic regression and how does it differ from OLS?
- How do we interpret logistic regression coefficients using log-odds, odds ratios, and marginal effects?
- When should we use the LPM versus logistic regression?

Suggested Readings

- Wooldridge (2019), Ch. 7.5, 17.1
- Bailey (2020), Ch. 12

Up until now, all of our dependent variables have been **quantitative** outcomes: wages, prices, birthweight, exam scores, and so on. But many important questions in economics—and especially in health economics and policy evaluation—involve **qualitative** outcomes. Did a patient survive? Did a worker find a job? Did a student graduate? Did a family enroll in a public program?

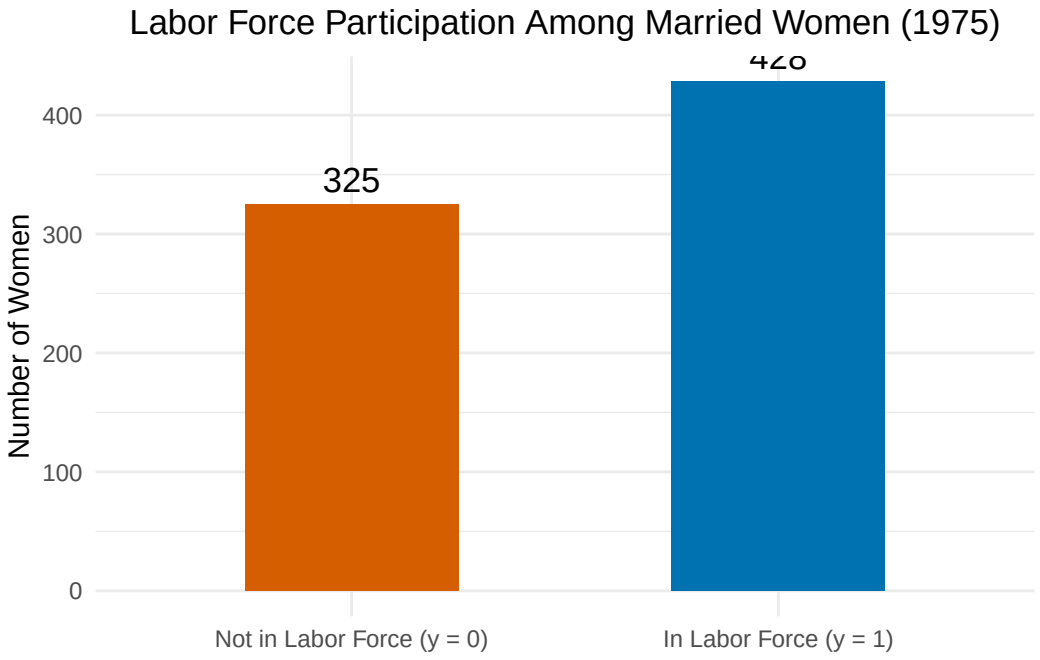
In the simplest case, we can represent these outcomes as **binary variables** that take the value 1 if the event occurred and 0 otherwise. This chapter explores how to model binary outcomes using regression, starting with the straightforward linear probability model and then introducing logistic regression as an alternative.

9.1 From Continuous to Binary Outcomes

To make the shift from continuous to binary outcomes concrete, let's look at some data. The `mroz` dataset in the `wooldridge` package contains information on 753 married women in 1975. The variable `inlf` equals 1 if a woman is in the labor force and 0 otherwise.

```r
lf_data <- wooldridge::mroz

ggplot(lf_data, aes(x = factor(inlf), fill = factor(inlf))) +
    geom_bar(width = 0.5) +
    geom_text(stat = "count", aes(label = after_stat(count)), vjust = -0.5, size = 4.5) +
    scale_fill_manual(values = c("0" = "#D55E00", "1" = "#0072B2"),
                      labels = c("Not in LF", "In LF")) +
    scale_x_discrete(labels = c("0" = "Not in Labor Force (y = 0)",
                                "1" = "In Labor Force (y = 1)")) +
    labs(x = NULL,
         y = "Number of Women",
         title = "Labor Force Participation Among Married Women (1975)") +
    theme_minimal() +
    theme(legend.position = "none",
          plot.title = element_text(hjust = 0.5))
```

Figure 9.1: The distribution of a binary outcome: women's labor force participation. Unlike a continuous variable, the outcome can only take two values.



As Figure 9.1 shows, about 57% of the women in the sample are in the labor force and 43% are not. This is fundamentally different from the continuous outcomes we have worked with before. There are no "in-between" values—a woman is either in the labor force or she isn't.

So what happens if we try to use OLS to model this outcome? The answer leads us to the **linear probability model**.

9.2 The Linear Probability Model

9.2.1 Why $E(y|\mathbf{x}) = P(y = 1|\mathbf{x})$

When our dependent variable y is binary, something interesting happens to the standard OLS regression. Consider the model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \mu$$

Recall that in OLS, the predicted value $\hat{y}$ estimates the conditional expectation $E(y|\mathbf{x})$. When y can only be 0 or 1, this conditional expectation has a special meaning. By the definition of expected value:

$$E(y|\mathbf{x}) = 1 \times P(y = 1|\mathbf{x}) + 0 \times P(y = 0|\mathbf{x}) = P(y = 1|\mathbf{x})$$

In other words, the predicted value from our regression is the **probability** that the event occurs, given the values of the explanatory variables. This is a powerful result: even though we're running plain OLS, the output has a probabilistic interpretation.

This means we cannot interpret β_j in the usual way—as the change in y when x_j increases by one unit—because y doesn't change continuously. Instead, we interpret β_j as the change in the **probability** that $y = 1$ when x_j increases by one unit, holding all other variables constant.

This is why OLS applied to a binary outcome is called the **linear probability model** (LPM).

> **The LPM Interpretation Rule**
>
> In the linear probability model, $\hat{\beta}_j$ tells us the **change in probability** (not percentage change!) of the event occurring for a one-unit increase in x_j. If $\hat{\beta}_j = -0.15$, the probability decreases by 0.15, or equivalently by 15 **percentage points**. Be careful with language: "15 percentage points" and "15 percent" are not the same thing!

9.3 LPM Example: Women's Labor Force Participation

To see how the LPM works in practice, let's examine a classic question in labor economics: what factors affect whether married women participate in the labor force?

We estimate the following model:

$$inlf = \beta_0 + \beta_1(kidslt6) + \beta_2(kidsge6) + \beta_3(nwifeinc) + \beta_4(motheduc) + \mu$$

where `kidslt6` is the number of children under age 6, `kidsge6` is the number of children aged 6–18, `nwifeinc` is the wife's non-labor income (in thousands of dollars), and `motheduc` is the mother's years of education.

Before estimating the model, let's visualize the raw relationship between our key variable of interest—young children—and labor force participation:

```
lf_data |>
    group_by(kidslt6) |>
    summarise(pct_inlf = mean(inlf),
              n = n()) |>
    ggplot(aes(x = factor(kidslt6), y = pct_inlf)) +
    geom_col(fill = "#0072B2", width = 0.5) +
    geom_text(aes(label = paste0(round(pct_inlf * 100, 1), "%\n(n=", n, ")")),
              vjust = -0.3, size = 3.5) +
    scale_y_continuous(labels = scales::percent_format(), limits = c(0, 0.8)) +
    labs(x = "Number of Children Under Age 6",
         y = "Share in Labor Force",
         title = "Labor Force Participation by Number of Young Children") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5))
```

Figure 9.2: Women with more young children are less likely to participate in the labor force. Each bar shows the share of women in each group who are working.

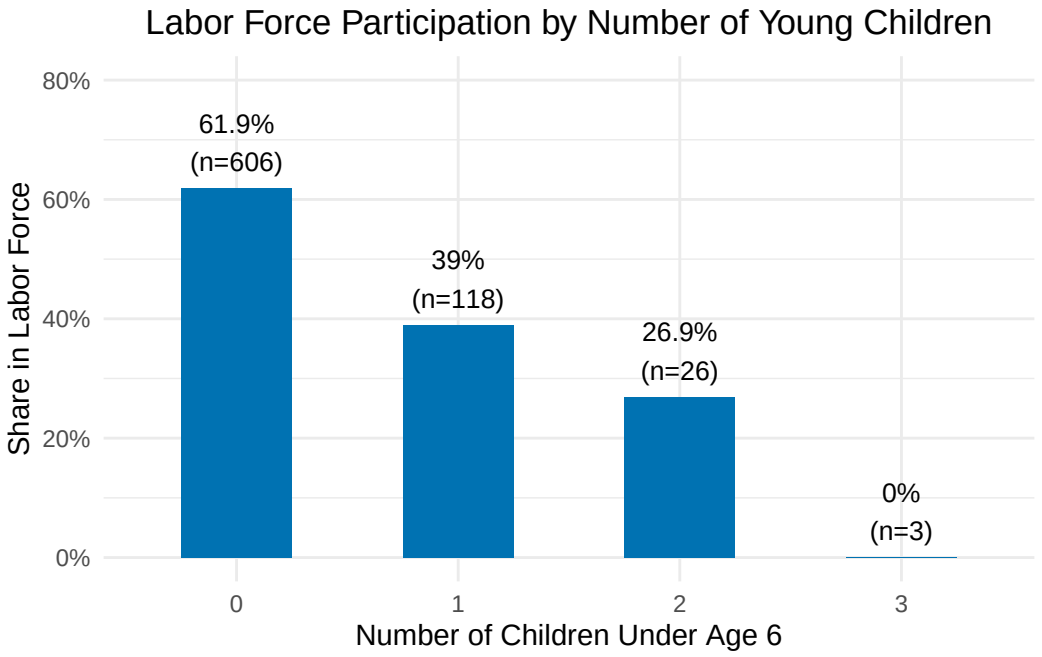


Figure 9.2 makes the pattern clear even before we run any regressions: women with no young children have the highest labor force participation rate, and the rate drops sharply with each additional young child. Now let's see what OLS tells us after controlling for other factors:

```
reg <- lm(inlf ~ kidslt6 + kidsge6 + nwifeinc + motheduc, data = lf_data)
summary(reg)
```

```
Call:
lm(formula = inlf ~ kidslt6 + kidsge6 + nwifeinc + motheduc,
    data = lf_data)

Residuals:
    Min      1Q  Median      3Q     Max
-0.8023 -0.5409  0.2867  0.3992  0.9144

Coefficients:
             Estimate Std. Error t value             Pr(>|t|)
(Intercept)  0.541676   0.058607   9.242 < 0.0000000000000002 ***
kidslt6     -0.212154   0.033588  -6.316        0.000000000459 ***
```

```
kidsge6      0.005765   0.013262   0.435             0.663874
nwifeinc    -0.005374   0.001512  -3.555             0.000402 ***
motheduc     0.019190   0.005250   3.655             0.000275 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4781 on 748 degrees of freedom
Multiple R-squared:  0.07454,   Adjusted R-squared:  0.06959
F-statistic: 15.06 on 4 and 748 DF,  p-value: 0.000000000007567
```

9.3.1 Interpreting the LPM Coefficients

Let's walk through each coefficient carefully.

The coefficient on `kidslt6` ($\hat{\beta}_1 \approx -0.21$) tells us that each additional child under age 6 **decreases the probability** of being in the labor force by about 21 percentage points, holding all else constant. This is a large and statistically significant effect. To put this in perspective, going from 0 to 1 young child reduces the predicted probability of working by roughly one-fifth.

The coefficient on `kidsge6` ($\hat{\beta}_2 \approx 0.01$) suggests that children aged 6 and older have essentially no effect on labor force participation—the coefficient is small and not statistically significant. This makes intuitive sense: school-age children require less constant supervision than young children, so they impose fewer constraints on a mother's ability to work outside the home.

The coefficient on `nwifeinc` ($\hat{\beta}_3 \approx -0.005$) indicates that higher non-labor income is associated with a slightly lower probability of working. Each additional thousand dollars of non-labor income reduces the probability by about half a percentage point. This is consistent with basic labor supply theory: when a household has more non-labor income (e.g., from a higher-earning husband), the wife has less financial pressure to enter the labor force.

The coefficient on `motheduc` ($\hat{\beta}_4 \approx 0.02$) suggests that women whose mothers had more education are slightly more likely to participate in the labor force. Each additional year of the mother's education is associated with about a 2 percentage point increase in the probability of working.

9.4 Problems with the Linear Probability Model

While the LPM is simple and intuitive, it has some important drawbacks that we need to understand.

9.4.1 Problem 1: Predictions Outside [0, 1]

The most obvious problem is that the LPM can produce predicted probabilities that are **less than 0 or greater than 1**. Since probabilities must lie between 0 and 1, this is nonsensical.

To see why this happens, recall that the LPM is just a straight line. A straight line extends infinitely in both directions, so for extreme enough values of x, the predicted value will inevitably cross the 0 or 1 boundary.

Let's see this concretely. Consider a woman with 2 children under 6, 0 older children, non-labor income of 30,000, and a mother with 9 years of education. Using our estimates, her predicted probability of being in the labor force is:

```
# Predicted probability for a specific woman
example_pred <- predict(reg, newdata = data.frame(
    kidslt6 = 2, kidsge6 = 0, nwifeinc = 30, motheduc = 9
))
cat("Predicted probability:", round(example_pred, 4), "\n")
```

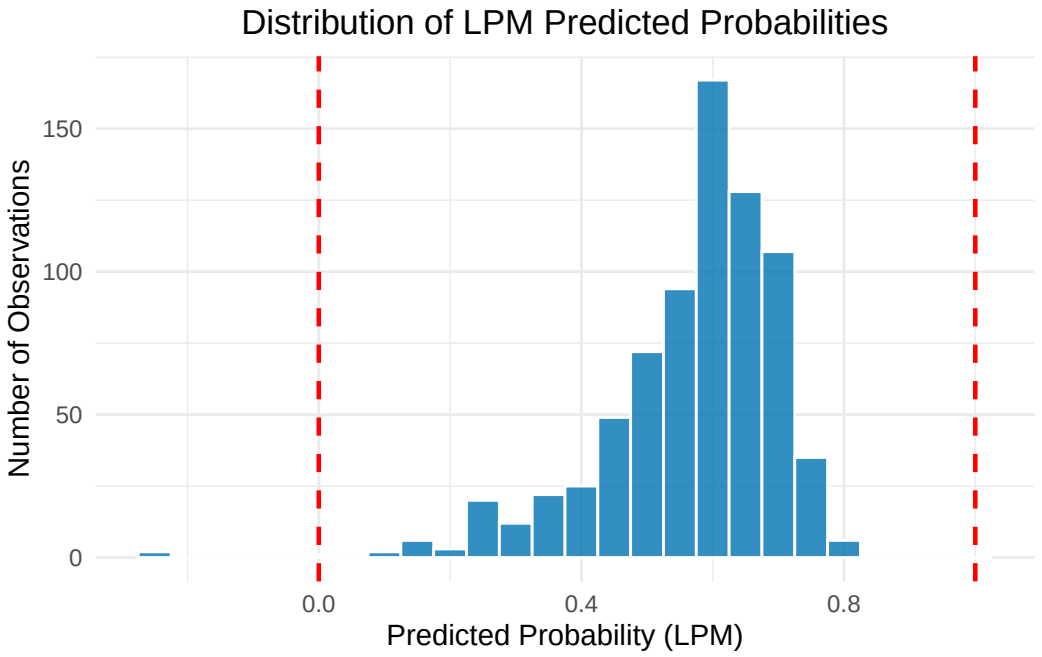
```
Predicted probability: 0.1288
```

A *negative* probability—obviously impossible!

We can see how widespread this problem is by looking at the full distribution of predicted probabilities from our model:

```
lf_data <- lf_data |>
    mutate(pred = predict(reg, type = "response"))

ggplot(data = lf_data, aes(x = pred)) +
    geom_histogram(binwidth = 0.05, fill = "#0072B2", color = "white", alpha = 0.8) +
    geom_vline(xintercept = 0, linetype = "dashed", color = "red", linewidth = 0.8) +
    geom_vline(xintercept = 1, linetype = "dashed", color = "red", linewidth = 0.8) +
    labs(x = "Predicted Probability (LPM)",
         y = "Number of Observations",
         title = "Distribution of LPM Predicted Probabilities") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5))
```

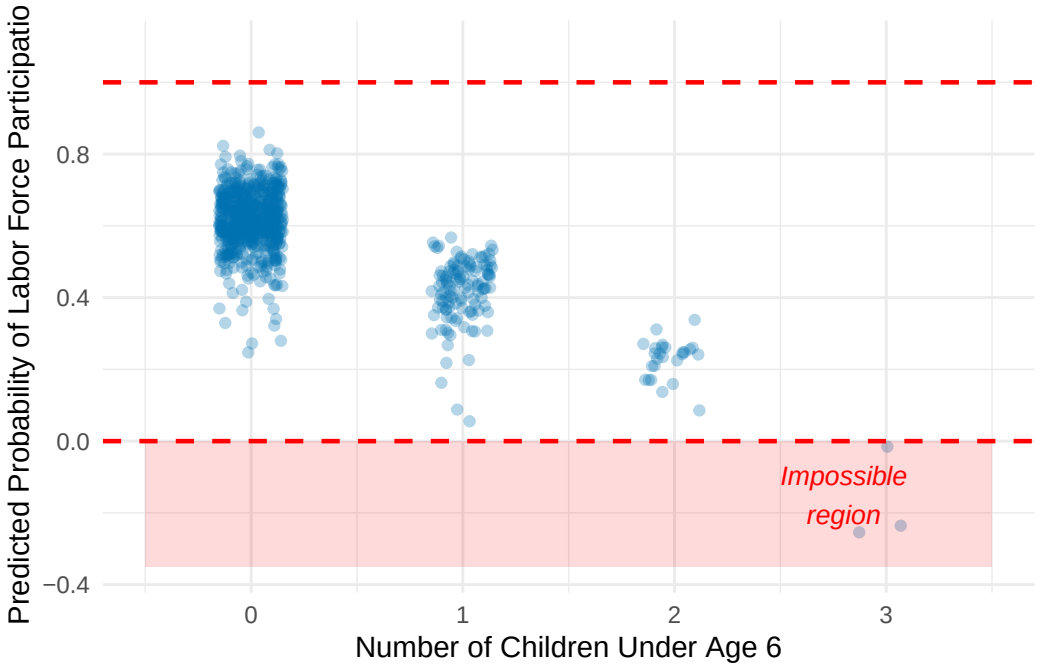
Figure 9.3: Distribution of LPM predicted probabilities. The dashed red lines mark the valid [0, 1] range. Some predictions fall below zero.



We can also see this by plotting the predicted probabilities against the number of young children:

```
ggplot(data = lf_data) +
    geom_jitter(aes(x = kidslt6, y = pred), alpha = 0.3, width = 0.15, color = "#0072B2") +
    geom_hline(yintercept = 0, linetype = "dashed", color = "red", linewidth = 0.8) +
    geom_hline(yintercept = 1, linetype = "dashed", color = "red", linewidth = 0.8) +
    annotate("rect", xmin = -0.5, xmax = 3.5, ymin = -0.35, ymax = 0,
             alpha = 0.15, fill = "red") +
    annotate("text", x = 2.8, y = -0.15, label = "Impossible\nregion",
             color = "red", fontface = "italic", size = 3.5) +
    coord_cartesian(ylim = c(-0.35, 1.1)) +
    labs(x = "Number of Children Under Age 6",
         y = "Predicted Probability of Labor Force Participation") +
    theme_minimal()
```

Figure 9.4: LPM predicted probabilities by number of young children. Some predictions fall below zero, violating the basic rules of probability.



As Figure 9.4 shows, some predicted probabilities fall below zero—particularly for women with multiple young children. The LPM simply cannot respect the fact that probabilities are bounded.

9.4.2 Problem 2: Heteroskedasticity

A second, more subtle problem is that the LPM **always** exhibits **heteroskedasticity**. Recall that one of the Gauss-Markov assumptions (GM5) requires that the variance of the error term be constant across observations. When y is binary, this assumption is necessarily violated.

To see why, think about what the error term looks like in the LPM. When y is 0 or 1, the error $\mu = y - E(y|\mathbf{x})$ can only take two possible values for any given $\mathbf{x}$:

- If $y = 1$: $\mu = 1 - p(\mathbf{x})$, which happens with probability $p(\mathbf{x})$
- If $y = 0$: $\mu = 0 - p(\mathbf{x}) = -p(\mathbf{x})$, which happens with probability $1 - p(\mathbf{x})$

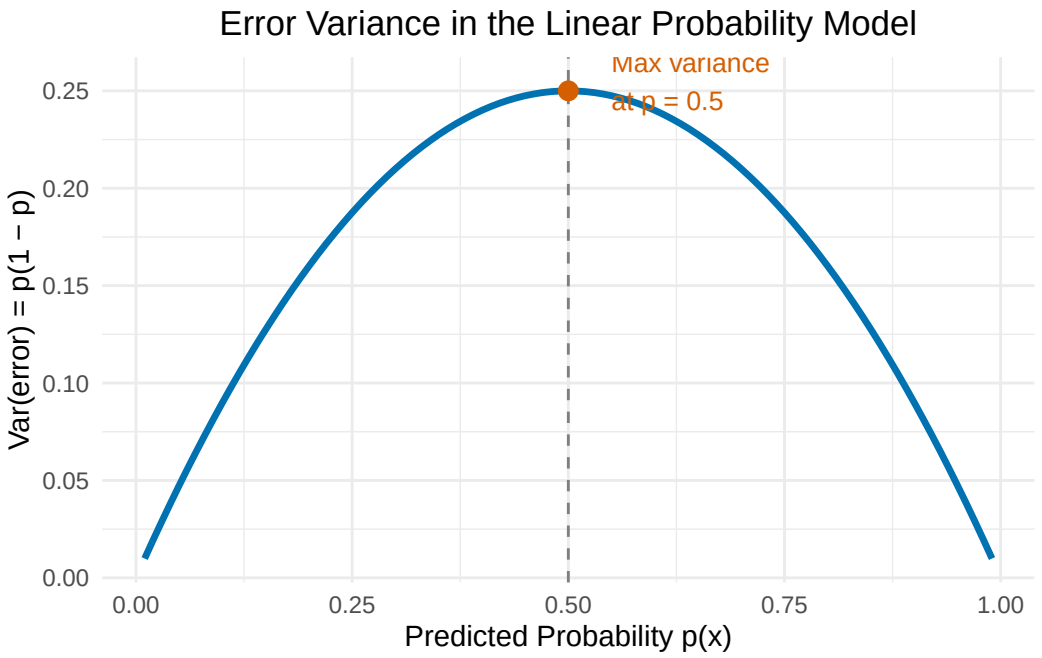
where $p(\mathbf{x}) = E(y|\mathbf{x})$ is the predicted probability. The variance of this error is:

$$\text{Var}(\mu|\mathbf{x}) = p(\mathbf{x}) \times (1 - p(\mathbf{x}))$$

This variance depends on $p(\mathbf{x})$, which depends on $\mathbf{x}$. The variance is largest when $p = 0.5$ (maximum uncertainty about the outcome) and smallest when p is near 0 or 1 (the outcome is nearly certain).

Let's visualize this:

Figure 9.5: The variance of the error term in a binary outcome model depends on the predicted probability. Variance is maximized at p = 0.5 and falls toward zero as p approaches 0 or 1.



Because of this built-in heteroskedasticity, the OLS estimator in the LPM is **not BLUE**—it is unbiased, but not the most efficient linear estimator. The practical consequence is that our standard errors will be incorrect unless we use **robust standard errors**, as we discussed in the chapter on heteroskedasticity.

! LPM: Use with Caution

Despite these problems, the LPM remains widely used in economics because of its simplicity and because the coefficient estimates are often very similar to those from more sophisticated models. The key is to be aware of its limitations: always use robust standard errors to address heteroskedasticity, and be cautious about interpreting predicted probabilities near the boundaries.

9.5 An Alternative: Logistic Regression

Given the issues with the LPM, researchers sometimes turn to alternative estimators that are specifically designed for binary outcomes. The most common of these is the **logit** (or **logistic regression**) model.

The key idea is to replace the linear function with a non-linear function that is guaranteed to produce predictions between 0 and 1. Instead of modeling the probability directly as a linear function of $\mathbf{x}$, we write:

$$P(y = 1|\mathbf{x}) = G(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)$$

where $G(\cdot)$ is a function that maps any real number into the interval $(0, 1)$.

9.5.1 The Logistic Function

In logit regression, we specifically define $G(\cdot)$ to be the **logistic function**:

$$G(z) = \frac{\exp(z)}{1 + \exp(z)}$$

This function has an S-shaped (sigmoid) curve that is worth studying carefully:

Several features of Figure 9.6 are worth noting. When z is very negative, $G(z)$ is close to 0; when z is very positive, $G(z)$ is close to 1; and at $z = 0$, $G(z) = 0.5$. Crucially, $G(z)$ is **strictly** between 0 and 1 for all values of z—so the logit model can never produce impossible predicted probabilities.

Notice also that the logistic function is **nearly linear in the middle** of its range (roughly between $z = -2$ and $z = 2$) but curves toward the boundaries at the extremes. This will be important later when we compare the LPM and logit results.

9.5.2 Comparing the LPM and Logit Visually

To build intuition for why the logistic function matters, let's compare what the LPM and logit model look like when fit to the same data. Using our full model with all four explanatory variables, we can compute a single "index" for each woman—the linear combination $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_k x_k$—and then plot how each model translates that index into a predicted probability. This is the clearest way to see the difference between the two approaches:

In Figure 9.7, the gray points show the actual data—all at $y = 0$ or $y = 1$. The x-axis is the linear index, which combines all the explanatory variables into a single score for each

Figure 9.6: The logistic function maps any real number to the (0, 1) interval. It is nearly linear in the middle but curves at the extremes, ensuring predicted probabilities always stay between 0 and 1.

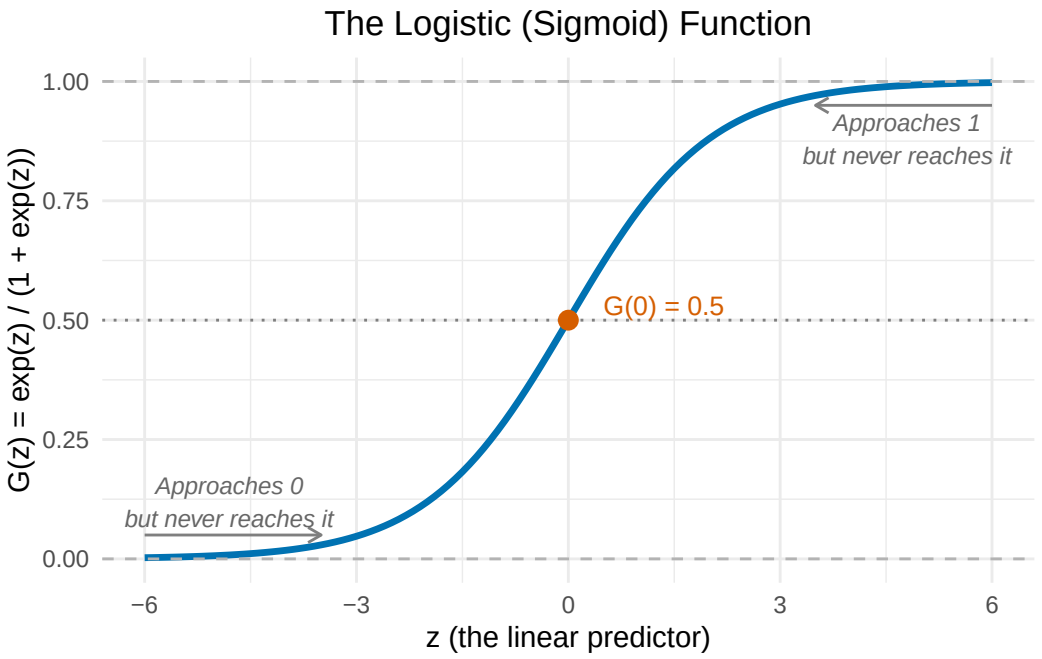
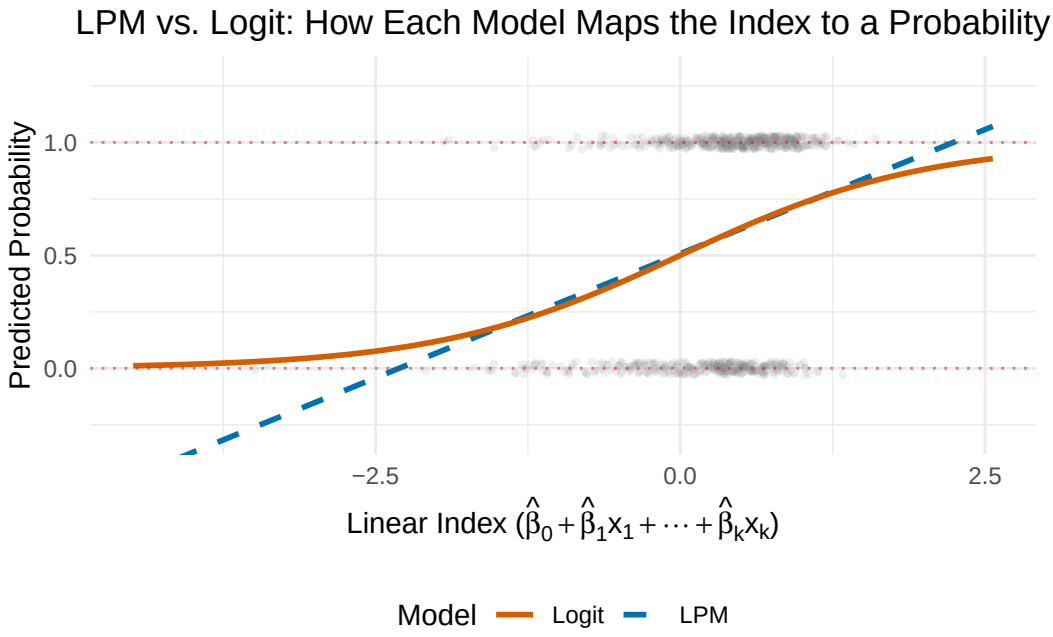


Figure 9.7: The LPM (blue, dashed) predicts probability as a straight line that can exceed [0, 1]. The logit (orange, solid) wraps the same linear index through the S-shaped logistic function, keeping predictions in bounds. Each gray point is one woman in the data, plotted at y = 0 or y = 1.



woman. Women with low index values have characteristics that predict low labor force participation (e.g., many young children, high non-labor income); women with high index values have characteristics that predict high participation.

The LPM maps this index to a predicted probability using a straight line, which inevitably extends beyond the [0, 1] bounds at the extremes. The logit wraps the same index through the S-shaped logistic function, keeping all predictions between 0 and 1. In the middle range—where most of the data sits—the two models agree closely. They diverge mainly at the tails, which is exactly where the LPM's out-of-bounds problem arises.

9.5.3 Maximum Likelihood Estimation

How do we estimate the logit model if we can't use OLS? The answer is **Maximum Likelihood Estimation (MLE)**. The intuition behind MLE is straightforward, even if the mathematical details are complex.

Think of it this way. For any set of β coefficients, the logistic function gives us a predicted probability $\hat{p}_i$ for each observation i. If $y_i = 1$ (the event occurred), we want $\hat{p}_i$ to be high—close to 1. If $y_i = 0$ (the event did not occur), we want $\hat{p}_i$ to be low—close to 0.

The **likelihood** is a measure of how well the model's predicted probabilities match the actual outcomes. MLE finds the set of β coefficients that maximizes this likelihood—that is, the coefficients that make the observed data most probable.

The process works iteratively:

1. Start with initial guesses for the β coefficients.
2. Plug these βs into the logistic function to calculate predicted probabilities.
3. Compute the total likelihood of observing the actual 0s and 1s given those predictions.
4. Adjust the βs to increase the likelihood.
5. Repeat until the likelihood can't be improved further.

The details of how MLE works are beyond the scope of this course, but it is useful to know what is happening under the hood. The key takeaway is that MLE finds the coefficients that make the observed data most likely, rather than minimizing the sum of squared residuals like OLS does.

> **i** OLS vs. MLE: Two Different Objectives
>
> OLS minimizes $\sum(y_i - \hat{y}_i)^2$—the sum of squared residuals. MLE maximizes the probability of observing the actual data given the model. For linear regression with normal errors, these two approaches give the same answer. For logistic regression, MLE is the appropriate method because the errors are not normal.

9.6 Estimating Logistic Regression in R

Estimating logistic regression in R is straightforward. We use `glm()` (generalized linear model) instead of `lm()`, and specify `family = binomial(link = "logit")`:

```
logit_reg <- glm(inlf ~ kidslt6 + kidsge6 + nwifeinc + motheduc,
    family = binomial(link = "logit"),
    data = lf_data)

summary(logit_reg)
```

```
Call:
glm(formula = inlf ~ kidslt6 + kidsge6 + nwifeinc + motheduc,
    family = binomial(link = "logit"), data = lf_data)

Coefficients:
             Estimate Std. Error z value      Pr(>|z|)
(Intercept)  0.177419   0.257368   0.689      0.490598
kidslt6     -0.964810   0.163790  -5.891 0.00000000385 ***
kidsge6      0.027471   0.058680   0.468      0.639682
nwifeinc    -0.024769   0.007063  -3.507      0.000453 ***
motheduc     0.085459   0.023509   3.635      0.000278 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1029.75  on 752  degrees of freedom
Residual deviance:  970.89  on 748  degrees of freedom
AIC: 980.89

Number of Fisher Scoring iterations: 4
```

Notice that the signs of the coefficients match the LPM: more young children reduce the probability of labor force participation, higher non-labor income reduces it, and more maternal education increases it. But the magnitudes look very different—the logit coefficients are not directly comparable to the LPM coefficients. We will discuss interpretation in detail below.

Table 9.1: Comparison of LPM and Logistic Regression estimates for women's labor force participation.

Table 9.2: LPM vs. Logit: Women's Labor Force Participation

	LPM (OLS)	Logit
(Intercept)	0.542***	0.177
	(0.059)	(0.257)
kidslt6	−0.212***	−0.965***
	(0.034)	(0.164)
kidsge6	0.006	0.027
	(0.013)	(0.059)
nwifeinc	−0.005***	−0.025***
	(0.002)	(0.007)
motheduc	0.019***	0.085***
	(0.005)	(0.024)
Num.Obs.	753	753
R2	0.075	
Log.Lik.	−510.246	−485.443
AIC	1032.5	980.9

* p <0.1, ** p <0.05, *** p <0.01

9.6.1 Comparing the Models Side by Side

Let's put the LPM and logit results next to each other using `modelsummary`:

```
models <- list(
    "LPM (OLS)" = reg,
    "Logit" = logit_reg
)

modelsummary(models,
             stars = c('*' = 0.1, '**' = 0.05, '***' = 0.01),
             gof_map = c("nobs", "r.squared", "logLik", "aic"),
             title = "LPM vs. Logit: Women's Labor Force Participation")
```

The key things to notice in Table 9.1 are that the signs are the same across both models

(both agree on the direction of each effect), but the magnitudes are different because the logit coefficients represent something different (changes in log-odds rather than changes in probability).

9.6.2 Predicted Probabilities Stay In Bounds

A major advantage of the logit model is that its predicted probabilities are always between 0 and 1. Let's compare the distributions of predicted probabilities from both models:

```
lf_data <- lf_data |>
    mutate(logit_pred = predict(logit_reg, type = "response"))

pred_long <- lf_data |>
    select(pred, logit_pred) |>
    pivot_longer(cols = everything(),
                 names_to = "model",
                 values_to = "predicted") |>
    mutate(model = case_when(
        model == "pred" ~ "LPM (OLS)",
        model == "logit_pred" ~ "Logit"
    ))

ggplot(pred_long, aes(x = predicted, fill = model)) +
    geom_histogram(binwidth = 0.05, color = "white", alpha = 0.8) +
    geom_vline(xintercept = 0, linetype = "dashed", color = "red", linewidth = 0.6) +
    geom_vline(xintercept = 1, linetype = "dashed", color = "red", linewidth = 0.6) +
    facet_wrap(~ model) +
    scale_fill_manual(values = c("LPM (OLS)" = "#0072B2", "Logit" = "#D55E00")) +
    labs(x = "Predicted Probability",
         y = "Count") +
    theme_minimal() +
    theme(legend.position = "none",
          strip.text = element_text(face = "bold", size = 12))
```

Figure 9.8: Comparing predicted probability distributions from the LPM (left) and logit (right) models. The LPM produces some predictions below 0, while the logit keeps all predictions within the valid range.

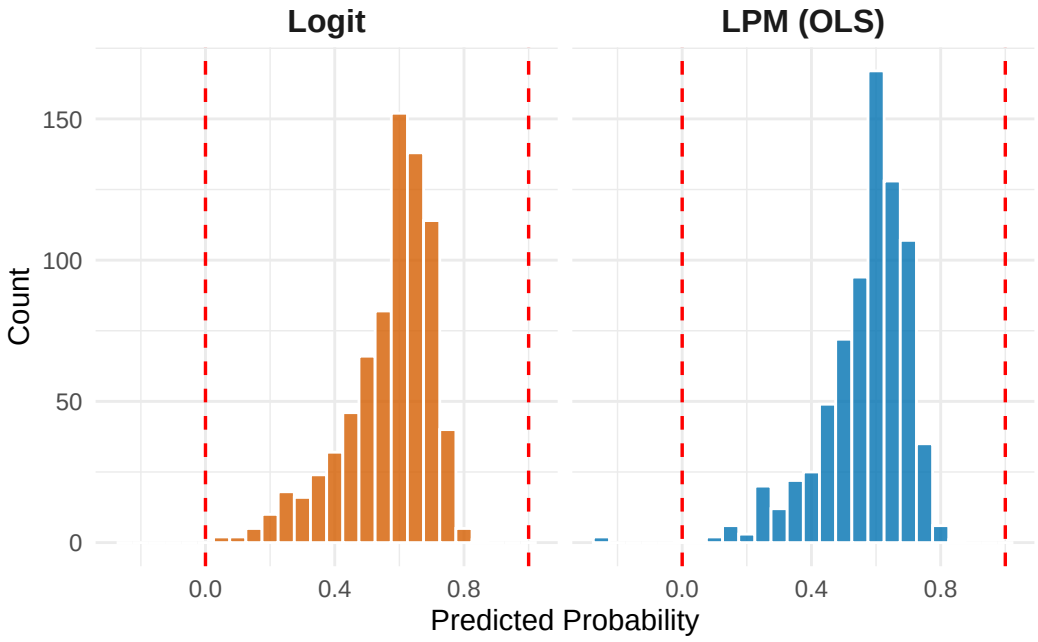


Figure 9.8 clearly shows the advantage of logistic regression: all predictions are squeezed into the valid [0, 1] range.

9.7 Interpreting Logistic Regression

The biggest challenge with logistic regression is that the coefficients are **not directly interpretable** in the same way as OLS coefficients. In the LPM, a one-unit increase in x_j changes the probability of $y = 1$ by exactly β_j percentage points—always, regardless of where you start. In logistic regression, the relationship between x and the probability is non-linear, so the effect of a one-unit change in x depends on where you start.

There are three common ways to interpret logit coefficients, each with different strengths.

9.7.1 Interpretation 1: Log-Odds

Technically, the logit coefficients represent the change in the **log-odds** of the outcome. Let's unpack what this means.

The **odds** of an event are defined as the probability of the event occurring divided by the probability of it not occurring:

$$\text{Odds} = \frac{P(y=1)}{P(y=0)} = \frac{P(y=1)}{1-P(y=1)}$$

For example, if the probability of being in the labor force is 0.75, the odds are $0.75/0.25 = 3$. We would say "the odds are 3 to 1 in favor of being in the labor force."

The **log-odds** (also called the "logit") is simply the natural log of the odds:

$$\text{Log-odds} = \ln\left(\frac{P(y=1)}{1-P(y=1)}\right)$$

In the logistic regression model, the log-odds is a linear function of the x variables:

$$\ln\left(\frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

So a one-unit increase in x_j changes the log-odds by β_j. This is mathematically clean but not very intuitive—most people don't think in terms of log-odds.

The main practical use of the log-odds interpretation is to determine **direction**: positive coefficients increase the probability of the event, and negative coefficients decrease it.

9.7.2 Interpretation 2: Odds Ratios

A more intuitive interpretation comes from exponentiating the coefficients. Since the log-odds changes by β_j for a one-unit increase in x_j, the **odds** are multiplied by $\exp(\beta_j)$. This quantity $\exp(\beta_j)$ is called the **odds ratio**.

```
or_data <- data.frame(
    variable = c("Kids < 6", "Kids 6-18", "Non-wife Income", "Mother's Educ"),
    odds_ratio = exp(coef(logit_reg)[-1]),
    lower = exp(confint.default(logit_reg)[-1, 1]),
    upper = exp(confint.default(logit_reg)[-1, 2])
)

ggplot(or_data, aes(x = odds_ratio, y = reorder(variable, odds_ratio))) +
    geom_vline(xintercept = 1, linetype = "dashed", color = "gray50") +
    geom_pointrange(aes(xmin = lower, xmax = upper), color = "#0072B2", size = 0.8) +
    labs(x = "Odds Ratio (with 95% CI)",
```

```
        y = NULL,
        title = "Odds Ratios for Labor Force Participation") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```

Figure 9.9: Odds ratios from the logistic regression. Values below 1 (to the left of the dashed line) decrease the odds of labor force participation; values above 1 increase them.

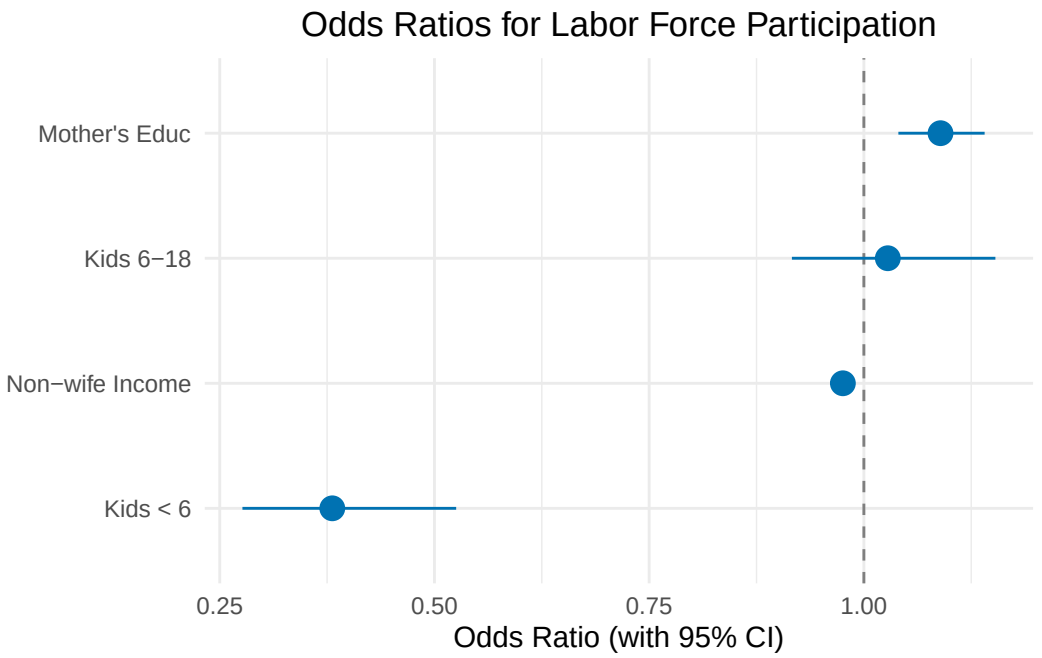


Figure 9.9 shows the odds ratios with 95% confidence intervals. Here's how to read them:

- **Kids < 6**: The odds ratio is about $\exp(-0.87) \approx 0.42$. Each additional child under 6 **multiplies** the odds of working by 0.42—reducing them by 58%. This is a large effect.
- **Mother's Educ**: The odds ratio is about $\exp(0.10) \approx 1.10$. Each additional year of mother's education multiplies the odds of working by 1.10—increasing them by 10%.
- If the confidence interval includes 1 (crosses the dashed line), the effect is not statistically significant.

Understanding Odds Ratios

An odds ratio of 1 means no effect. An odds ratio of 2 means the odds are doubled. An odds ratio of 0.5 means the odds are halved. Odds ratios are always positive—they can never be negative or zero.
Be careful not to confuse odds ratios with probability ratios. An odds ratio of 2 does

not mean the probability doubles. The relationship between odds and probability is non-linear.

9.7.3 Interpretation 3: Marginal Effects at the Mean

The most practical way to compare logit estimates to LPM estimates is to compute **marginal effects**—the change in predicted probability for a one-unit change in x, evaluated at specific values of the other variables. The most common approach is to evaluate at the **mean** of all explanatory variables.

Let's compute this for the effect of one additional young child:

```
# Create a hypothetical person with average values for all RHS variables
mean_vals <- lf_data |>
    summarise(across(all_of(c("kidslt6", "kidsge6", "nwifeinc", "motheduc")),
    \(x) mean(x, na.rm = TRUE)
    ))

mean_vals <- mean_vals |>
    slice(1) |>
    unlist()
mean_vals
```

```
   kidslt6    kidsge6   nwifeinc   motheduc
 0.2377158  1.3532537 20.1289637  9.2509960
```

```
# Predicted probability at the mean
log_pred_mean <- sum(c(1, mean_vals) * coefficients(logit_reg))
prob_mean <- plogis(log_pred_mean)
cat("Predicted probability at the mean:", round(prob_mean, 4), "\n")
```

```
Predicted probability at the mean: 0.5689
```

```
# Predicted probability with 1 more kid under 6
mean_vals_1morekid <- mean_vals + c(1, 0, 0, 0)
log_pred_1morekid <- sum(c(1, mean_vals_1morekid) * coefficients(logit_reg))
prob_1morekid <- plogis(log_pred_1morekid)
cat("Predicted probability with 1 more kid:", round(prob_1morekid, 4), "\n")
```

```
Predicted probability with 1 more kid: 0.3346
```

```r
# Marginal effect
marginal_effect <- prob_1morekid - prob_mean
cat("Marginal effect of 1 more kid (logit):", round(marginal_effect, 4), "\n")
```

```
Marginal effect of 1 more kid (logit): -0.2343
```

```r
cat("LPM coefficient on kidslt6:", round(coef(reg)["kidslt6"], 4), "\n")
```

```
LPM coefficient on kidslt6: -0.2122
```

The marginal effect of one additional young child at the mean is approximately −0.20—very close to the LPM estimate of −0.21. This illustrates a common and important finding: for observations near the middle of the probability distribution, the LPM and logit model often give very similar results.

But the marginal effect from the logit model **changes** depending on where you evaluate it. Let's see how the marginal effect of one more young child varies across the distribution:

Figure 9.10: The marginal effect of one additional young child on labor force participation probability, computed at different values of non-wife income. Unlike the constant LPM effect (dashed line), the logit marginal effect varies—it is largest where the predicted probability is near 0.5.

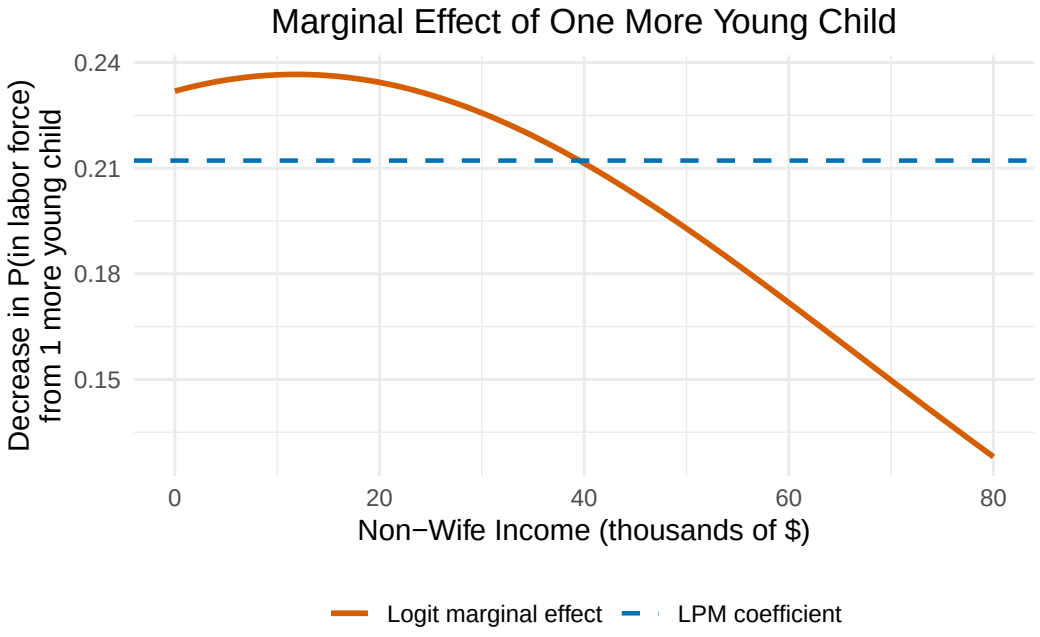


Figure 9.10 reveals a key property of the logit model: the marginal effect is not constant. The LPM assumes the same effect everywhere (the dashed line), but the logit marginal effect varies depending on the starting probability. The effect is largest when the predicted probability is near 0.5 (where the logistic curve is steepest) and smaller when the probability is near 0 or 1 (where the curve flattens out).

> **The `plogis()` Function**
>
> In R, the `plogis()` function converts log-odds to probabilities. It applies the logistic function: `plogis(z)` $= \exp(z)/(1+\exp(z))$. This is the inverse of the `qlogis()` function, which converts probabilities to log-odds.

9.8 Application: Low Birthweight and Maternal Smoking

To see binary outcome models in a health economics context, let's examine the relationship between maternal smoking and low birthweight. Using the `bwght` dataset from the `wooldridge` package, we define low birthweight as a birthweight below 88 ounces (approximately 2,500 grams—the standard clinical threshold):

```
bw_data <- wooldridge::bwght |>
    mutate(low_bwght = ifelse(bwght < 88, 1, 0))

cat("Prevalence of low birthweight:", round(mean(bw_data$low_bwght) * 100, 1), "%\n")
```

```
Prevalence of low birthweight: 6 %
```

Now let's estimate both an LPM and a logit model:

```
lpm_bw <- lm(low_bwght ~ cigs + faminc + motheduc + male, data = bw_data)

logit_bw <- glm(low_bwght ~ cigs + faminc + motheduc + male,
                family = binomial(link = "logit"),
                data = bw_data)

models_bw <- list("LPM" = lpm_bw, "Logit" = logit_bw)
modelsummary(models_bw,
             stars = c('*' = 0.1, '**' = 0.05, '***' = 0.01),
             gof_map = c("nobs", "r.squared", "logLik", "aic"),
             title = "Determinants of Low Birthweight")
```

Table 9.3: LPM and Logit estimates for the probability of low birthweight.

Table 9.4: Determinants of Low Birthweight

	LPM	Logit
(Intercept)	0.064*	−2.647***
	(0.038)	(0.681)
cigs	0.003**	0.032**
	(0.001)	(0.015)
faminc	−0.001	−0.010
	(0.000)	(0.007)
motheduc	0.001	0.010
	(0.003)	(0.056)
male	−0.005	−0.090
	(0.013)	(0.228)
Num.Obs.	1387	1387
R2	0.006	
Log.Lik.	32.138	−310.351
AIC	−52.3	630.7

* p <0.1, ** p <0.05, *** p <0.01

The LPM tells us that each additional cigarette smoked per day during pregnancy increases the probability of low birthweight by about 0.4 percentage points. The logit coefficients point in the same direction but are on the log-odds scale.

Let's visualize the predicted probability of low birthweight as a function of maternal smoking, comparing both models:

Figure 9.11: Predicted probability of low birthweight by number of cigarettes smoked per day, from both the LPM and logit models. The models agree closely in the range of the data but diverge at extremes.

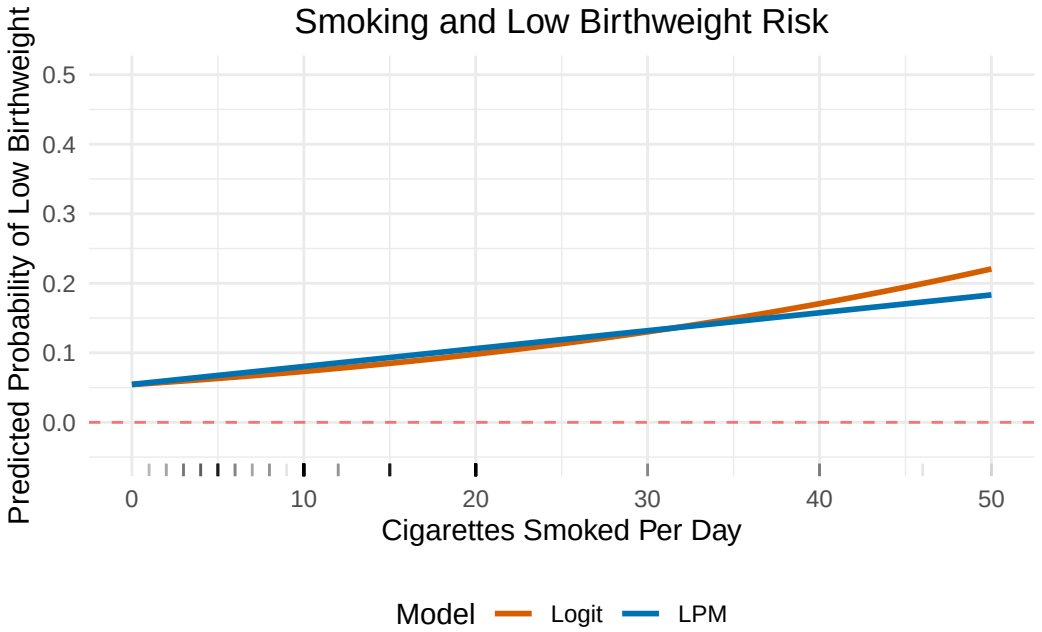


Figure 9.11 illustrates several important points. For moderate levels of smoking (roughly 0–20 cigarettes per day, where most of the data lives), the two models give very similar predictions. At high levels of smoking, the LPM continues in a straight line—eventually predicting impossibly high probabilities—while the logit curve flattens out as it approaches 1. This is exactly the kind of situation where the logit model's bounded predictions are an advantage.

9.9 LPM vs. Logistic Regression: When to Use Which?

As we have seen repeatedly, the marginal effects from the logistic regression are often quite similar to the LPM coefficients, at least for observations with characteristics near the center of the distribution. So should we bother with logistic regression at all?

The answer depends on the context and the discipline.

In **economics**, many researchers prefer the LPM for its simplicity and ease of interpretation. The out-of-bounds prediction problem can often be managed by using robust standard errors to handle heteroskedasticity and by being careful about extrapolation. Most applied microeconomics papers use the LPM as their primary specification, sometimes reporting logit results as a robustness check.

In **health sciences and epidemiology**, logistic regression is the standard approach. Researchers in these fields tend to report odds ratios, and the logit framework integrates naturally with other methods common in biostatistics. If you are reading a paper in a health or medical journal, you will almost certainly encounter logistic regression.

One area where both camps agree is **prediction**. If the goal is to predict outcomes—rather than to estimate causal effects—logistic regression generally outperforms the LPM. Its predictions are guaranteed to be valid probabilities, which matters for applications like machine learning, clinical risk scores, and forecasting.

Table 9.5: LPM vs. Logistic Regression Comparison

	LPM	Logistic Regression
Estimation	OLS	Maximum Likelihood
Coefficients	Change in probability	Change in log-odds
Interpretation	Direct and intuitive	Requires conversion (odds ratios or marginal effects)
Predictions	Can fall outside [0, 1]	Always in (0, 1)
Heteroskedasticity	Always present	Handled by MLE
Common in	Economics	Health sciences, epidemiology
Best for	Causal inference with easy interpretation	Prediction; settings where valid probabilities matter

Practical Advice

For this course, you should be comfortable with both approaches. The LPM is typically sufficient for causal inference questions, and its coefficients are immediately interpretable. Use logistic regression when you need predictions that are guaranteed to be valid probabilities, and when the prediction itself (rather than the coefficients) are most important. We will see an example use-case of this in Chapter 12.

9.10 Summary

When the dependent variable is binary (0 or 1), we need to think carefully about how to model it.

The **linear probability model** applies OLS to a binary outcome. Its coefficients are easy to interpret: β_j is the change in the probability of $y = 1$ for a one-unit increase in x_j. However, the LPM can produce predicted probabilities outside the [0, 1] range and inherently suffers from heteroskedasticity.

Logistic regression addresses the out-of-bounds problem by wrapping the linear prediction inside the logistic function, which maps any value to the (0, 1) interval. It is estimated using maximum likelihood rather than OLS. Logit coefficients represent changes in the log-odds and are harder to interpret directly, but they can be converted to odds ratios or marginal effects for more intuitive interpretation. Marginal effects computed at the mean are often similar to LPM coefficients.

In practice, the choice between LPM and logistic regression often comes down to disciplinary convention and the specific goals of the analysis. Economists tend to favor the LPM for its simplicity; health researchers tend to favor the logit for its theoretical properties. When the two approaches give similar results—as they often do for marginal effects near the center of the distribution—the choice matters less than getting the research question right.

9.11 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

10 Panel Data and Fixed Effects

Key Questions

- What distinguishes panel data from cross-sectional data?
- How can observing units over time help us address omitted variable bias?
- What is a fixed effect and what does it control for?
- How do we estimate fixed effects models in practice?
- What types of confounders can fixed effects eliminate, and what can they not?

Suggested Readings

- Wooldridge (2019), Ch. 13-14

Throughout this course, we have emphasized that omitted variable bias poses the central threat to causal inference. We can add control variables, but we can only control for what we can measure. What about unobserved factors like natural ability, motivation, local culture, or a city's history?

Panel data—observing the same units over multiple time periods—offers a powerful solution. By tracking how outcomes change within the same unit over time, we can control for all characteristics of that unit that remain constant, even if we cannot observe or measure them directly. This chapter introduces panel data methods, with a focus on **fixed effects** estimation.

10.1 Cross-Sectional vs. Panel Data

10.1.1 Cross-Sectional Data

Most datasets we have examined so far are **cross-sectional**: each unit (person, firm, city) is observed at a single point in time. Examples include housing prices in Ames, Iowa in 2010; penguin measurements from a single expedition; or wage data from one year of the Current Population Survey.

With cross-sectional data, we can describe populations, identify correlations, and use multivariate regression to control for observable confounders. But we face a fundamental limitation: we cannot control for unobserved characteristics that differ across units.

10.1.2 Panel Data

Panel data (also called longitudinal data) tracks the same units across multiple time periods. Examples include:

- NBA players observed across multiple seasons
- Employment and wages in states tracked quarterly over several years
- Crime rates in cities measured annually for a decade
- Stock prices for companies tracked daily

The key feature is that we observe the **same unit** at **different times**, allowing us to see how that unit changes when circumstances change.

10.2 Why Panel Data Helps: The Intuition

Consider estimating the effect of unemployment on crime. With cross-sectional data from a single year, we might compare cities with high unemployment to cities with low unemployment. But cities differ in countless ways: geography, demographics, policing strategies, local culture, economic history. Many of these factors affect both unemployment and crime, creating omitted variable bias.

Now imagine we observe the **same cities** over two years. Some cities experience rising unemployment; others see it fall. We can ask: when unemployment rises *within a city*, does crime also rise?

This **within-unit comparison** automatically controls for everything about that city that stays constant over time—its geography, its historical legacy, its baseline demographics. We don't need to measure these factors; we just need them to be stable across the time periods we observe.

> **!** The Key Insight
>
> Panel data allows us to control for **all time-invariant characteristics** of each unit, whether observed or unobserved. The variation we use for identification comes from changes *within* units over time, not comparisons *across* different units.

10.3 The Fixed Effects Model

10.3.1 Setting Up the Model

Let's formalize this intuition. Suppose we observe units $i = 1, 2, ..., N$ over time periods $t = 1, 2, ..., T$. Our outcome is y_{it} and our variable of interest is x_{it}.

The panel data model is:

$$y_{it} = \beta_0 + \beta_1 x_{it} + a_i + \tau_t + \mu_{it} \tag{10.1}$$

where:

- y_{it} is the outcome for unit i at time t
- x_{it} is the explanatory variable (which varies across units and time)
- a_i is the **unit fixed effect**—capturing all time-invariant characteristics of unit i
- τ_t is the **time fixed effect**—capturing shocks that affect all units equally in period t
- μ_{it} is the idiosyncratic error term

The unit fixed effect a_i is the crucial element. It absorbs everything about unit i that doesn't change over time: genetics, geography, institutional history, baseline culture. In our crime example, a_i captures each city's fixed characteristics that affect crime rates.

10.3.2 What Does This Solve?

Recall that omitted variable bias occurs when an unobserved factor is correlated with both x and y. In the standard cross-sectional regression:

$$y_i = \beta_0 + \beta_1 x_i + \mu_i$$

any time-invariant unobserved factor correlated with x gets absorbed into the error term, biasing our estimate of β_1.

With fixed effects, those time-invariant factors are captured by a_i and explicitly controlled for. As long as the confounders don't change over time, they cannot bias our estimate.

10.4 Estimating Fixed Effects: The Within Transformation

How do we actually estimate Equation 10.1 when we don't observe a_i?

One approach is the **within transformation** (also called the "demeaning" approach). For each unit i, we compute the average of each variable over time:

$$\bar{y}_i = \beta_0 + \beta_1 \bar{x}_i + a_i + \bar{\tau} + \bar{\mu}_i$$

where $\bar{y}_i = \frac{1}{T}\sum_{t=1}^{T} y_{it}$ is unit i's average outcome across all time periods.

Now subtract this time-averaged equation from the original:

$$y_{it} - \bar{y}_i = \beta_1(x_{it} - \bar{x}_i) + (\tau_t - \bar{\tau}) + (\mu_{it} - \bar{\mu}_i)$$

Notice what happened: the a_i term **completely disappears**! Since a_i is constant over time, $a_i - a_i = 0$.

We can write this more compactly as:

$$\ddot{y}_{it} = \beta_1 \ddot{x}_{it} + \ddot{\tau}_t + \ddot{\mu}_{it}$$

where the double-dots indicate "time-demeaned" variables (deviations from unit means).

> **Why "Within" Transformation?**
>
> This estimator is called the "within" transformation because the variation used to identify β_1 comes entirely from variation *within* each unit over time. Cross-sectional differences between units are eliminated by the demeaning process.

10.4.1 Example: Unemployment and Crime

Let's see this in action. We have data on crime rates and unemployment for 46 cities observed in two years (1982 and 1987).

First, let's see what happens with simple cross-sectional regression using just 1987 data:

```
crime_data <- wooldridge::crime2

# Cross-sectional regression using only 1987
reg_cross <- lm(crmrte ~ unem, data = filter(crime_data, year == 87))
summary(reg_cross)
```

```
Call:
lm(formula = crmrte ~ unem, data = filter(crime_data, year ==
    87))

Residuals:
   Min     1Q Median     3Q    Max
-57.55 -27.01 -10.56  18.01  79.75

Coefficients:
            Estimate Std. Error t value   Pr(>|t|)
(Intercept)  128.378     20.757   6.185 0.00000018 ***
unem          -4.161      3.416  -1.218       0.23
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 34.6 on 44 degrees of freedom
Multiple R-squared:  0.03262,   Adjusted R-squared:  0.01063
F-statistic: 1.483 on 1 and 44 DF,  p-value: 0.2297
```

The coefficient on unemployment is **negative** (-4.16), suggesting higher unemployment is associated with *lower* crime. This counterintuitive result likely reflects omitted variable bias: cities with low unemployment might also have other characteristics (wealth, strong institutions) that lead to lower crime.

Now let's use both years and apply the within transformation:

```r
# Create city identifier
crime_data <- crime_data |>
    mutate(city_id = rep(1:46, each = 2))

# Compute city-level means
city_means <- crime_data |>
    group_by(city_id) |>
    summarize(
        mean_crmrte = mean(crmrte),
        mean_unem = mean(unem),
        mean_d87 = mean(d87)
    )

# Merge and demean
crime_data <- crime_data |>
    left_join(city_means, by = "city_id") |>
```

```
    mutate(
        crmrte_demean = crmrte - mean_crmrte,
        unem_demean = unem - mean_unem,
        time_demean = d87 - mean_d87
    )

# Estimate on demeaned data
reg_within <- lm(crmrte_demean ~ unem_demean + time_demean, data = crime_data)
summary(reg_within)
```

```
Call:
lm(formula = crmrte_demean ~ unem_demean + time_demean, data = crime_data)

Residuals:
    Min      1Q  Median      3Q     Max
-26.458  -6.384   0.000   6.384  26.458

Coefficients:
                         Estimate            Std. Error t value  Pr(>|t|)
(Intercept)  0.000000000000001116  1.039332417007331255   0.000  1.000000
unem_demean  2.217999508245958040  0.617247710094903201   3.593  0.000535 ***
time_demean 15.402203621534521716  3.306166769286674967   4.659 0.0000111 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.969 on 89 degrees of freedom
Multiple R-squared:  0.1961,    Adjusted R-squared:  0.178
F-statistic: 10.85 on 2 and 89 DF,  p-value: 0.00006059
```

Now the coefficient on unemployment is **positive** (2.22)! After controlling for all time-invariant city characteristics, we find that when unemployment rises within a city, crime rises too. This makes much more intuitive sense.

The within transformation eliminated the bias from unobserved city characteristics that were driving the negative cross-sectional correlation.

10.5 Estimating Fixed Effects: The Dummy Variable Approach

In practice, we rarely compute the within transformation manually. Instead, we use an equivalent approach: including **dummy variables** for each unit.

We estimate:

$$y_{it} = \beta_0 + \beta_1 x_{it} + \sum_{j=2}^{N} \gamma_j D_j + \sum_{s=2}^{T} \delta_s T_s + \mu_{it}$$

where D_j is a dummy variable equal to 1 if the observation is from unit j, and T_s is a dummy equal to 1 if the observation is from time period s. We omit one unit and one time period to avoid the dummy variable trap.

This approach is mathematically equivalent to the within transformation but more flexible: it's easier to add control variables, and we can see the estimated fixed effects if desired.

```
# Create factor variable for city and dummy for 1987
crime_data <- crime_data |>
    mutate(
        city_f = factor(city_id),
        year_87 = ifelse(year == 87, 1, 0)
    )

# Estimate with dummy variables
reg_fe <- lm(crmrte ~ unem + year_87 + city_f, data = crime_data)

# Show just the key coefficients (not all 46 city dummies)
summary(reg_fe)$coefficients[1:4, ]
```

```
            Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 51.48923 12.3457756 4.170595 0.0001404348
unem         2.21800  0.8778658 2.526581 0.0151893177
year_87     15.40220  4.7021169 3.275589 0.0020604685
city_f2     17.29171 14.1954509 1.218116 0.2296714584
```

The coefficient on `unem` (2.22) is identical to our within transformation estimate!

The dummy variable coefficients themselves are usually not of primary interest, but they can provide useful information. For instance, we can see which cities have unusually high or low crime rates after accounting for unemployment.

10.6 What Fixed Effects Control For (and What They Don't)

Fixed effects are powerful, but they're not magic. It's crucial to understand exactly what they do and don't control for.

10.6.1 What Unit Fixed Effects Control For

Unit fixed effects (a_i) absorb all characteristics of unit i that are **constant over time**:

- Geography and location
- Historical legacy and founding conditions
- Stable institutional features
- Baseline demographics (to the extent they don't change much)
- Measurement differences (e.g., how crime is recorded)
- "Culture" and other hard-to-measure factors

10.6.2 What Time Fixed Effects Control For

Time fixed effects (τ_t) absorb all factors that **affect all units equally** in a given period:

- Macroeconomic conditions (recessions, booms)
- National policy changes
- Seasonal patterns
- Technological changes affecting everyone

10.6.3 What Fixed Effects Do NOT Control For

Fixed effects cannot eliminate bias from factors that vary **both across units and over time** in ways correlated with the treatment:

- Time-varying confounders specific to certain units
- Differential trends (some units changing faster than others)
- Reverse causality (if y affects x within a period)

⚠ Fixed Effects ≠ No OVB

Fixed effects dramatically reduce omitted variable bias by controlling for time-invariant confounders. But they don't eliminate all possible bias. Unit-specific, time-varying confounders can still cause problems. Always think carefully about what might be changing differently across your units over time.

For example, in our crime analysis, fixed effects control for each city's baseline characteristics. But if some cities increased their police budgets while others cut them, and these budget changes are correlated with unemployment changes, we would still have omitted variable bias. We could address this by adding police spending as an explicit control variable.

10.7 Adding Control Variables

We can include additional time-varying control variables in fixed effects models:

$$y_{it} = \beta_1 x_{it} + \gamma z_{it} + a_i + \tau_t + \mu_{it}$$

where z_{it} is a control variable that changes over time within units.

This helps address the remaining source of potential bias: unit-specific, time-varying confounders. Of course, we can only control for variables we observe and measure.

10.8 Practical Considerations

10.8.1 When to Use Fixed Effects

Fixed effects are most valuable when:

1. You have genuine panel data (same units observed multiple times)
2. There is meaningful variation in x within units over time
3. Time-invariant confounders are a major concern
4. The treatment effect is expected to be relatively immediate

10.8.2 When Fixed Effects May Not Work Well

Fixed effects may be less useful when:

1. Little within-unit variation exists (everyone's x is stable over time)
2. The treatment effect takes a long time to materialize
3. Time-varying confounders are the main concern
4. You have very few time periods (less precise estimates)

10.8.3 Standard Errors

With panel data, observations within the same unit are often correlated over time. Standard OLS standard errors assume independence and can be too small. In practice, researchers typically use **clustered standard errors** at the unit level to account for this correlation. We'll discuss this more in later chapters.

10.9 Summary

Panel data—observing the same units over time—offers a powerful tool for causal inference. By tracking how outcomes change within units when circumstances change, we can control for all time-invariant characteristics of each unit, whether observed or not.

The **fixed effects estimator** implements this idea. It can be computed via the within transformation (demeaning the data) or equivalently by including dummy variables for each unit. The coefficient of interest is identified purely from within-unit variation over time.

Fixed effects control for:

- All time-invariant unit characteristics (observed and unobserved)
- All time-specific shocks affecting all units equally (when time fixed effects are included)

Fixed effects do NOT control for:

- Unit-specific, time-varying confounders
- Differential trends across units
- Reverse causality

Panel methods are not a complete solution to omitted variable bias, but they represent a major improvement over cross-sectional analysis when time-invariant confounders are a primary concern.

10.10 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

11 Difference-in-Differences

Key Questions

- What is a natural experiment and how does it help with causal inference?
- What is the parallel trends assumption and why is it crucial for DiD?
- How do we estimate the average treatment effect on the treated (ATT) using DiD?
- What is two-way fixed effects and how does it relate to DiD?
- How do we choose good control units for a DiD design?

Suggested Readings

- TBD

We have built up an increasingly powerful toolkit for causal inference. Randomized controlled trials remain the gold standard, but are often infeasible. Multivariate OLS allows us to control for observable confounders, and fixed effects extend this to control for unobserved time-invariant factors. But what if even fixed effects aren't enough?

This chapter introduces **difference-in-differences (DiD)**, a method that combines the logic of experiments with the practicality of observational data. Instead of relying solely on control *variables*, DiD uses control *units*—a comparison group that shows us what would have happened to the treated group in the absence of treatment.

11.1 Natural Experiments

11.1.1 The Problem with Observational Data

Suppose we want to estimate the effect of immigration on native workers' wages. A naive approach might compare wages in cities with high immigration to wages in cities with low immigration. But this comparison is confounded: immigrants tend to move to cities with strong economies, creating a spurious positive correlation between immigration and wages.

Even with control variables, we may miss important unobserved factors like a city's economic trajectory or its attractiveness to both immigrants and high-wage employers. Fixed effects help

with time-invariant confounders, but cannot address factors that change differently across cities over time.

11.1.2 Natural Experiments as a Solution

A **natural experiment** occurs when some external force—nature, policy, or historical accident—assigns units to treatment and control groups in a way that resembles random assignment.

> **Natural Experiments**
>
> A **natural experiment** is a study based on observational data where "treatment" is assigned by forces outside the researcher's control, such as policy changes, geographic boundaries, or historical events. Natural experiments create conditions that *resemble* random assignment, allowing us to make causal inferences without actually randomizing.

The key insight is that if treatment assignment is "as good as random," then comparing treated and control units can identify causal effects—just like in an RCT.

11.2 The Mariel Boatlift: A Classic Natural Experiment

One of the most famous natural experiments in economics involves the Mariel Boatlift. In 1980, Cuba experienced severe economic turmoil, and Fidel Castro announced that anyone who wished to leave could do so. Between April and October, approximately 125,000 Cubans departed from Mariel Harbor and arrived in Miami.

This mass migration had nothing to do with labor market conditions in Miami. It was driven entirely by Cuban politics and Miami's geographic proximity to Cuba. In other words, the "treatment" of a sudden large increase in labor supply was essentially randomly assigned to Miami.

David Card's seminal 1990 study used this natural experiment to estimate the effect of immigration on native wages. But how exactly do we use such an event to identify causal effects?

11.3 The Difference-in-Differences Approach

11.3.1 Why Simple Before-After Comparisons Fail

Card first compared wages in Miami before the boatlift (1979) to wages after (1980-1985):

City	Pre (1979)	Post (1980-1985)	Difference
Miami	1.85	1.83	-0.02

Wages fell slightly. Can we interpret this as the causal effect of immigration?

No. Wages might have fallen everywhere during this period due to recession or other macroeconomic factors. Looking only at the treated unit's change over time confuses the treatment effect with broader trends.

11.3.2 The Need for Control Units

To isolate the treatment effect, we need to know what *would have happened* to Miami's wages if the boatlift had never occurred. This **counterfactual** is inherently unobservable—we cannot see the parallel universe where Cuba didn't open its borders.

But we can approximate it using **control units**: cities similar to Miami that were not affected by the boatlift. Card chose Atlanta, Los Angeles, Houston, and Tampa-St. Petersburg as controls, arguing they had similar demographics and economic trajectories to Miami.

City	Pre (1979)	Post (1980-1985)	Difference
Miami	1.85	1.83	-0.02
Controls	1.93	1.91	-0.02
Difference			**0.00**

The control cities experienced the *same* wage decline as Miami. This suggests the wage drop in Miami was not caused by the boatlift but by broader economic trends affecting all cities.

The **difference-in-differences** estimate is the change in Miami minus the change in controls:

$$(-0.02) - (-0.02) = 0.00$$

Card concluded that the Mariel Boatlift had essentially no effect on native wages—a surprising and influential finding.

11.4 The Parallel Trends Assumption

11.4.1 The Core Identifying Assumption

The DiD approach rests on one critical assumption:

! Parallel Trends Assumption

In the absence of treatment, the average outcome for the treated group would have followed the **same trend** as the average outcome for the control group.

Mathematically:

$$E[Y_{i,post} - Y_{i,pre}|Treated] = E[Y_{i,post} - Y_{i,pre}|Control]$$

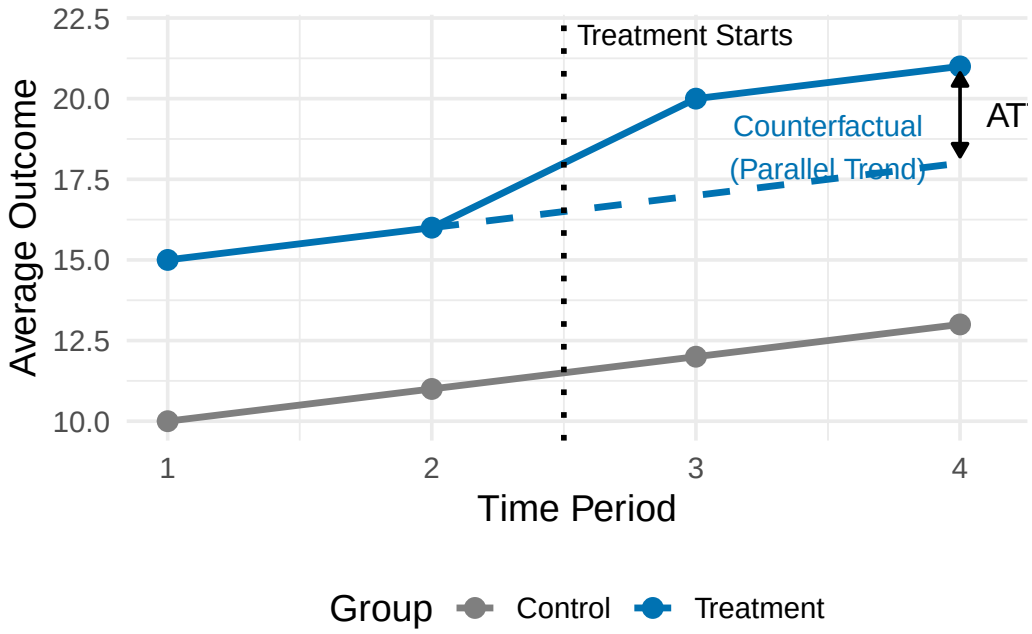
in the counterfactual world where treatment never occurred.

This assumption is **not testable** directly because we never observe the treated group's counterfactual trend. However, we can assess its plausibility by examining whether treated and control groups followed similar trends *before* treatment began. If pre-treatment trends are parallel, it's more credible that they would have remained parallel absent treatment.

11.4.2 Visualizing DiD

The logic of DiD is best understood graphically:

Figure 11.1: The DiD estimate is the gap between the observed treatment outcome and the counterfactual (dashed line) based on the control group's trend.



The **Average Treatment Effect on the Treated (ATT)** is the vertical distance between:

1. The observed outcome for the treated group after treatment
2. The counterfactual: where the treated group *would have been* if it had followed the control group's trend

The dashed line represents this counterfactual—the treated group's hypothetical trajectory without treatment.

11.5 The DiD Regression

11.5.1 Setting Up the Model

We typically estimate DiD using regression. For the simple 2×2 case (two groups, two time periods), the model is:

$$y_{it} = \beta_0 + \beta_1 D_i + \beta_2 P_t + \delta(D_i \times P_t) + \mu_{it} \tag{11.1}$$

where:

- D_i is a **treatment group indicator** (1 if treated, 0 if control)
- P_t is a **post-period indicator** (1 if post-treatment, 0 if pre-treatment)
- $D_i \times P_t$ is the interaction term (1 only for treated units in the post period)
- δ is the **DiD estimate**—our estimate of the ATT

11.5.2 How It Works

To see why δ captures the DiD effect, consider the predicted outcomes for each group-period combination:

Group	Period	Predicted Outcome
Control	Pre	0
Control	Post	$0 + 2$
Treated	Pre	$0 + 1$
Treated	Post	$0 + 1 + 2 +$

Now compute the differences:

Change for treated group: $(0 + 1 + 2 +) - (0 + 1) = 2 +$

Change for control group: $(0 + 2) - 0 = 2$

DiD estimate: $(2 +) - 2 =$

The coefficient on the interaction term exactly equals the difference-in-differences!

11.5.3 Example: Estimating the Mariel Boatlift Effect

```
# Create the Card (1990) data
miami_wages <- tibble(
    city = "miami",
    year = 1979:1985,
    wages = c(1.85, 1.83, 1.85, 1.82, 1.82, 1.82, 1.82)
)

control_wages <- tibble(
    city = "control",
    year = 1979:1985,
    wages = c(1.93, 1.90, 1.91, 1.91, 1.90, 1.91, 1.92)
)

boatlift_data <- bind_rows(miami_wages, control_wages) |>
    mutate(
        post = ifelse(year >= 1980, 1, 0),
        treated = ifelse(city == "miami", 1, 0),
        post_treat = post * treated
    )

# Estimate DiD regression
did_reg <- feols(wages ~ treated + post + post_treat,
                 vcov = "HC1",
                 data = boatlift_data)

summary(did_reg)
```

```
OLS estimation, Dep. Var.: wages
Observations: 14
Standard-errors: Heteroskedasticity-robust
             Estimate Std. Error        t value    Pr(>|t|)
(Intercept)  1.930000 0.00000100 1930000.000000   < 2.2e-16 ***
treated     -0.080000 0.00000100  -80000.000000   < 2.2e-16 ***
post        -0.021667 0.00331942      -6.527254 0.000066615 ***
post_treat  -0.001667 0.00628785      -0.265062 0.796345671
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.008522   Adj. R2: 0.947329
```

The coefficient on `post_treat` (approximately 0.01) is our DiD estimate. It suggests the Mariel Boatlift had essentially no effect on wages—consistent with Card's original findings.

11.6 Two-Way Fixed Effects

11.6.1 Extending to Multiple Units and Time Periods

The simple 2×2 DiD works well for clean natural experiments with one treated unit and one control unit observed before and after. But what if we have many treated units, many control units, and many time periods?

The **Two-Way Fixed Effects (TWFE)** model extends DiD to these more complex settings:

$$y_{it} = \delta D_{it} + \alpha_i + \tau_t + \mu_{it} \tag{11.2}$$

where:

- D_{it} equals 1 if unit i is treated at time t, and 0 otherwise
- α_i are **unit fixed effects** (one for each unit)
- τ_t are **time fixed effects** (one for each time period)
- δ is the treatment effect

This model subsumes the simple DiD: unit fixed effects capture permanent differences between groups (like $\beta_1 D_i$), and time fixed effects capture common shocks to all units (like $\beta_2 P_t$).

11.6.2 Estimating TWFE in R

The `fixest` package makes TWFE estimation straightforward. The syntax `| unit + time` specifies the fixed effects:

```
# TWFE estimation
twfe_reg <- feols(wages ~ post_treat | city + year,
                  vcov = "HC1",
                  data = boatlift_data)

summary(twfe_reg)
```

```
OLS estimation, Dep. Var.: wages
Observations: 14
Fixed-effects: city: 2,  year: 7
Standard-errors: Heteroskedasticity-robust
             Estimate Std. Error   t value Pr(>|t|)
post_treat -0.001667   0.006491 -0.256776  0.80758
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.00622     Adj. R2: 0.943875
                 Within R2: 0.002193
```

The coefficient on `post_treat` is identical to our simple DiD regression—as expected, since TWFE and the 2×2 DiD model are equivalent in this simple case.

11.7 Research Design: Choosing Good Controls

11.7.1 The Art of Causal Inference

Your DiD estimate is only as good as your research design. The goal is to find control units that make the parallel trends assumption credible.

Good Controls → Good Counterfactual → Credible DiD Estimate

Bad Controls → Bad Counterfactual → Misleading DiD Estimate

11.7.2 Example: Minimum Wages and Employment

Consider estimating the effect of minimum wage increases on employment. One approach might compare all states that raised their minimum wage to all states that didn't.

But states that raise minimum wages are systematically different: they tend to be more politically liberal, have stronger labor markets, different industries, and many other characteristics that affect employment. These differences violate parallel trends—minimum-wage states may have been on different employment trajectories regardless of the policy.

Card and Krueger (1994) proposed a more compelling research design. When New Jersey raised its minimum wage in 1992 and neighboring Pennsylvania did not, they compared fast-food restaurants in counties *right next to each other* on opposite sides of the NJ-PA border.

The identifying assumption: restaurants separated only by a state border are similar in all relevant ways except the minimum wage policy. Geographic proximity makes parallel trends much more plausible than comparing NJ to distant states with very different economies.

11.7.3 Intuition-Driven vs. Data-Driven Selection

There are two broad approaches to choosing control units:

Intuition-driven selection relies on substantive knowledge to identify appropriate comparisons. Card's choice of control cities for Miami was based on demographic and economic similarities. Card and Krueger's border design was based on geographic proximity. The advantage is transparency and interpretability.

Data-driven selection uses algorithms to find control units that match the treated unit on pre-treatment outcomes. Methods like synthetic control construct a weighted average of potential controls that best reproduces the treated unit's pre-treatment trajectory. The advantage is objectivity; the disadvantage is complexity and potential overfitting.

In practice, the best research designs often combine both approaches: use substantive knowledge to identify a reasonable set of potential controls, then verify that pre-treatment trends are indeed parallel.

11.8 What Can Go Wrong

11.8.1 Violations of Parallel Trends

The most serious threat to DiD is violation of the parallel trends assumption. This can happen when:

1. **Differential trends:** Treated and control units were already on different trajectories before treatment
2. **Anticipation effects:** Units change behavior before treatment actually begins
3. **Spillovers:** Treatment affects control units (e.g., workers move from treated to control areas)
4. **Composition changes:** The mix of units changes over time in ways correlated with treatment

11.8.2 Assessing Parallel Trends

While we cannot test parallel trends directly, we can assess plausibility by:

1. **Examining pre-treatment trends:** Do treated and control groups move together before treatment? Diverging pre-trends are a red flag.
2. **Placebo tests:** Estimate "effects" at times when no treatment occurred. Significant placebo effects suggest the design is flawed.
3. **Event studies:** Estimate treatment effects for each time period separately to visualize the dynamic pattern.

If pre-treatment trends are not parallel, the DiD estimate will be biased. The direction of bias depends on whether the treated group was trending upward or downward relative to controls.

11.9 Summary

Difference-in-differences is one of the most widely used methods for causal inference in economics and social science. It combines the intuition of experimental design with the practicality of observational data.

The key elements are:

1. **Natural experiments** provide plausibly exogenous variation in treatment assignment
2. **Control units** approximate what would have happened to treated units absent treatment
3. **The parallel trends assumption** requires that treated and control groups would have followed the same trajectory without treatment
4. **The DiD estimator** subtracts the control group's change from the treated group's change to isolate the treatment effect
5. **TWFE** extends DiD to multiple units and time periods using unit and time fixed effects

The credibility of any DiD analysis rests entirely on the parallel trends assumption. Careful research design—choosing appropriate control units and verifying parallel pre-trends—is essential for producing convincing causal estimates.

11.10 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

12 Advanced Difference-in-Differences

Key Questions

- How can we use data to select better control units for DiD?
- What are propensity scores and how do they improve DiD estimates?
- How do we estimate treatment effects that change over time?
- What is an event study and how do we interpret it?
- Why do we need cluster-robust standard errors in panel data?

Suggested Readings

- TBD

The basic difference-in-differences framework is powerful, but real-world applications often require more sophisticated tools. This chapter covers three important extensions: **data-driven control selection** using propensity scores, **dynamic DiD** for effects that evolve over time, and **cluster-robust standard errors** for proper inference with grouped data.

12.1 Data-Driven Control Selection

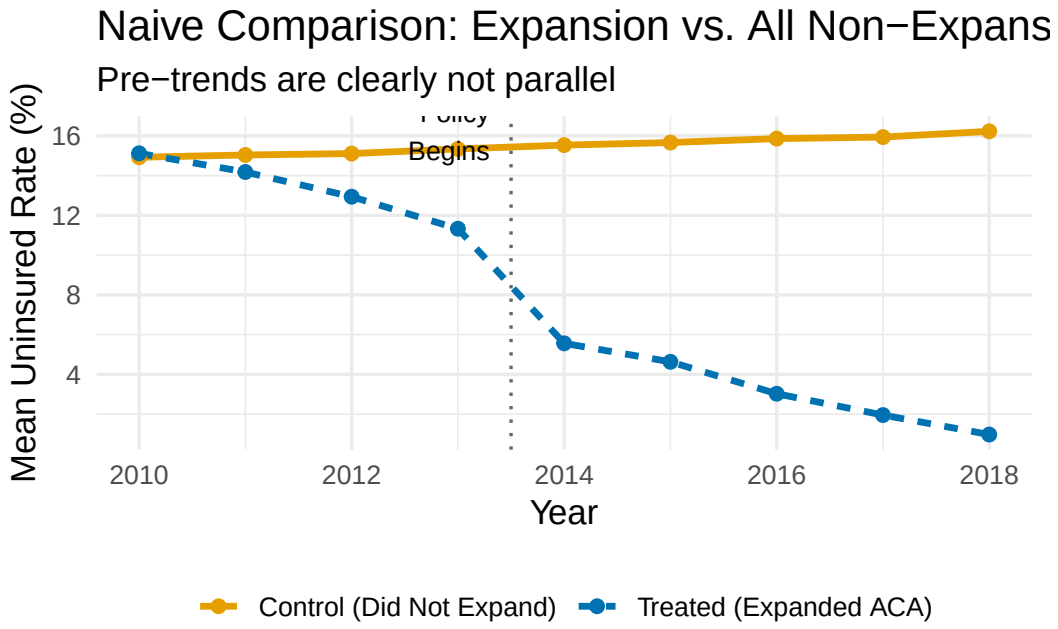
12.1.1 The Problem: Systematically Different Treatment and Control Groups

In the previous chapter, we emphasized that your DiD estimate is only as good as your counterfactual. When treated and control groups are systematically different, the parallel trends assumption becomes implausible.

Consider estimating the effect of the Affordable Care Act (ACA) Medicaid expansion on uninsured rates. States that expanded Medicaid are very different from states that didn't: they tend to be more politically liberal, have different demographics, different industries, and different baseline health care systems.

Using all non-expanding states as controls produces biased estimates because these states were on different trajectories *even before* the policy.

Figure 12.1: Without adjustment, treated and control states show diverging pre-trends, violating the parallel trends assumption.



The figure shows that even *before* the ACA expansion in 2014, treated states had steeper downward trends in uninsured rates. This pre-existing difference would be incorrectly attributed to the policy in a naive DiD analysis.

12.1.2 Propensity Score Weighting: The Intuition

The solution is to find control units that are *similar* to treated units. If we could compare only states that were "on the verge" of expanding but didn't, we would have a much better counterfactual.

Propensity score methods formalize this intuition. The idea is:

1. Estimate each unit's probability of being treated based on observable characteristics
2. Give more weight to control units that "look like" they should have been treated
3. Use these weights in the DiD regression

Control units with high propensity scores (high predicted probability of treatment) but who weren't actually treated become our primary comparison group. They're similar to treated units on observables, making parallel trends more plausible.

12.1.3 The IPW-DiD Algorithm

Inverse Propensity Score Weighted DiD (IPW-DiD) proceeds in three steps:

Step 1: Estimate the Propensity Score

We estimate a logistic regression predicting treatment status from pre-treatment characteristics:

$$P(D_i = 1|X) = G(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k)$$

where $G()$ is the logistic function and the X variables are characteristics that predict treatment.

For the ACA example, we might include political leaning (Democratic vote share), region, and baseline poverty rates.

```
# Aggregate to state level for propensity score estimation
state_data <- panel_data |>
    filter(post == 0) |>  # Use only pre-treatment data
    group_by(state_id, is_expanded) |>
    summarize(
        dem_share = mean(X1),
        region = mean(X2),
        poverty = mean(X3),
        .groups = 'drop'
    )

# Estimate propensity scores with logistic regression
ps_model <- glm(is_expanded ~ dem_share + region + poverty,
                data = state_data,
                family = binomial("logit"))

# Get predicted probabilities
state_data <- state_data |>
    mutate(ps_score = predict(ps_model, type = "response"))
```

Step 2: Calculate IPW Weights

The inverse propensity weights are:

$$IPW_i = D_i + (1 - D_i) \times \frac{\hat{P}_i}{1 - \hat{P}_i}$$

- For treated units ($D_i = 1$): weight $= 1$
- For control units ($D_i = 0$): weight $= \frac{\hat{P}_i}{1-\hat{P}_i}$

Control units with high propensity scores get large weights; those with low propensity scores get small weights.

```
# Calculate IPW weights
state_data <- state_data |>
    mutate(
        ipw = case_when(
            is_expanded == 1 ~ 1,
            is_expanded == 0 ~ ps_score / (1 - ps_score)
        )
    )

# Show example weights
state_data |>
    select(state_id, is_expanded, ps_score, ipw) |>
    arrange(desc(ipw)) |>
    head(8) |>
    knitr::kable(digits = 2, caption = "Example IPW Weights")
```

Table 12.1: Example IPW Weights

state_id	is_expanded	ps_score	ipw
32	0	0.86	5.94
181	0	0.79	3.77
51	0	0.75	3.06
101	0	0.69	2.26
31	0	0.65	1.85
150	0	0.63	1.71
29	0	0.61	1.58
106	0	0.59	1.41

Step 3: Estimate Weighted TWFE

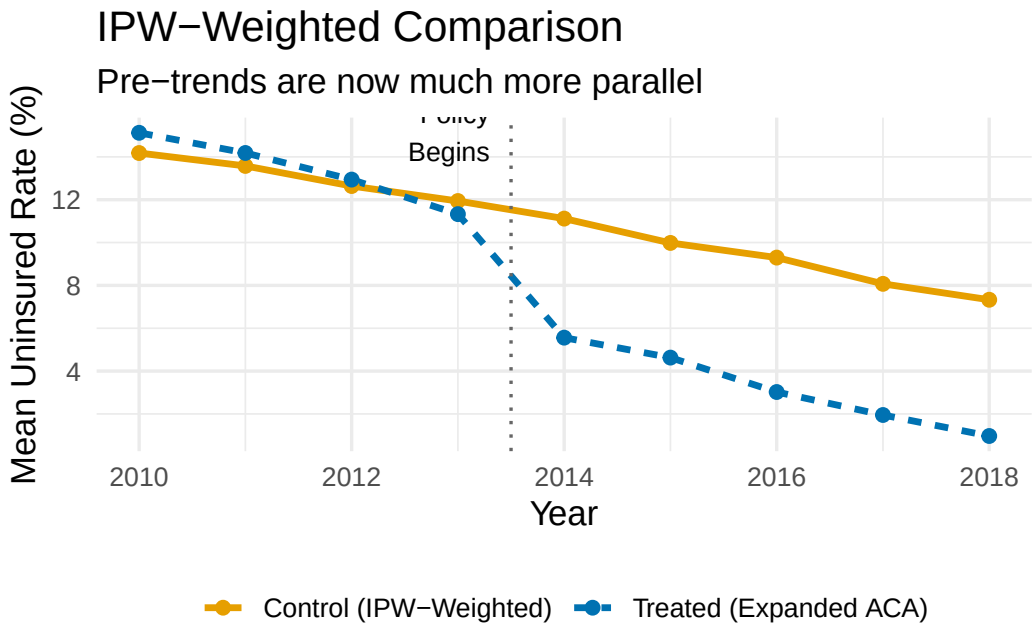
Finally, we estimate the standard TWFE DiD regression but using the IPW weights:

```
# Merge weights into panel data
panel_weighted <- panel_data |>
    left_join(select(state_data, state_id, ipw), by = "state_id")
```

```
# Naive (unweighted) DiD
did_naive <- feols(uninsured_rate ~ treat_active | state_id + year,
                   vcov = "HC1",
                   data = panel_data)

# IPW-weighted DiD
did_ipw <- feols(uninsured_rate ~ treat_active | state_id + year,
                 vcov = "HC1",
                 weights = ~ipw,
                 data = panel_weighted)
```

12.1.4 Comparing Results

Figure 12.2: IPW weighting restores parallel pre-trends by up-weighting control units similar to treated units.



The IPW-weighted trends are much more parallel in the pre-period! This gives us greater confidence in the parallel trends assumption.

Table 12.2: Comparing Naive and IPW-DiD Estimates (True effect = -5)

Method	Estimate	True Effect
Naive DiD (all controls equal weight)	-10.91	-5
IPW-Weighted DiD	-6.24	-5

The naive estimate is substantially biased (too negative) because it attributes pre-existing trends to the policy. The IPW-weighted estimate is much closer to the true effect of -5 percentage points.

> **!** Key Insight
>
> IPW-DiD works by reweighting the control group so that it looks more like the treatment group on observable characteristics. This makes parallel trends more plausible—but the method is only as good as your propensity score model. If important confounders are omitted, bias remains.

12.2 Dynamic DiD and Event Studies

12.2.1 Why Treatment Effects May Change Over Time

The basic DiD model estimates a single treatment effect that applies to all post-treatment periods. But many policies have effects that **evolve over time**:

- A job training program might have small initial effects that grow as workers gain experience
- A minimum wage increase might have immediate disemployment effects that fade as firms adjust
- An environmental regulation might have growing effects as compliance increases

Dynamic DiD allows us to estimate separate treatment effects for each time period, capturing how the policy impact unfolds.

12.2.2 Relative Time

Dynamic DiD models use **relative time**—the number of periods since treatment occurred—rather than calendar time. If treatment occurs in 2015:

Calendar Year	Relative Time
2012	-3
2013	-2
2014	-1
2015 (treatment)	0
2016	+1
2017	+2

This framing lets us estimate effects at each distance from treatment and, crucially, test whether pre-treatment "effects" are zero (as they should be if parallel trends holds).

12.2.3 The Dynamic DiD Model

The dynamic DiD model includes dummy variables for each relative time period:

$$y_{it} = \sum_{k \neq -1} \beta_k D_{it}^k + \alpha_i + \tau_t + \mu_{it}$$

where $D_{it}^k = 1$ if unit i is treated and period t is k periods from treatment.

We **omit** $k = -1$ (the period just before treatment) as the reference category. All coefficients are interpreted relative to this baseline.

12.2.4 Example: Environmental Policy and Manufacturing Employment

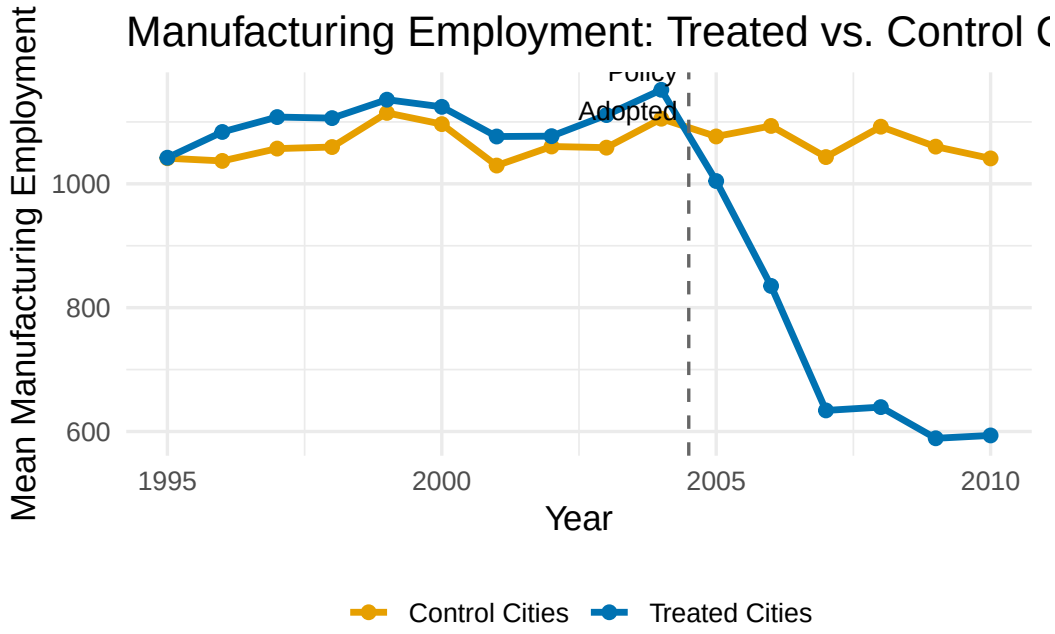
Suppose we want to estimate how an environmental regulation affects manufacturing employment in cities. We have data on 50 cities observed from 1995 to 2010, with 25 cities adopting the policy in 2005.

Let's visualize the raw trends first:

The trends are parallel before 2005, then the treated cities experience a growing decline in manufacturing employment.

12.2.5 Estimating the Dynamic Model

Figure 12.3: Treatment and control cities show parallel pre-trends, then diverge after the policy in 2005.

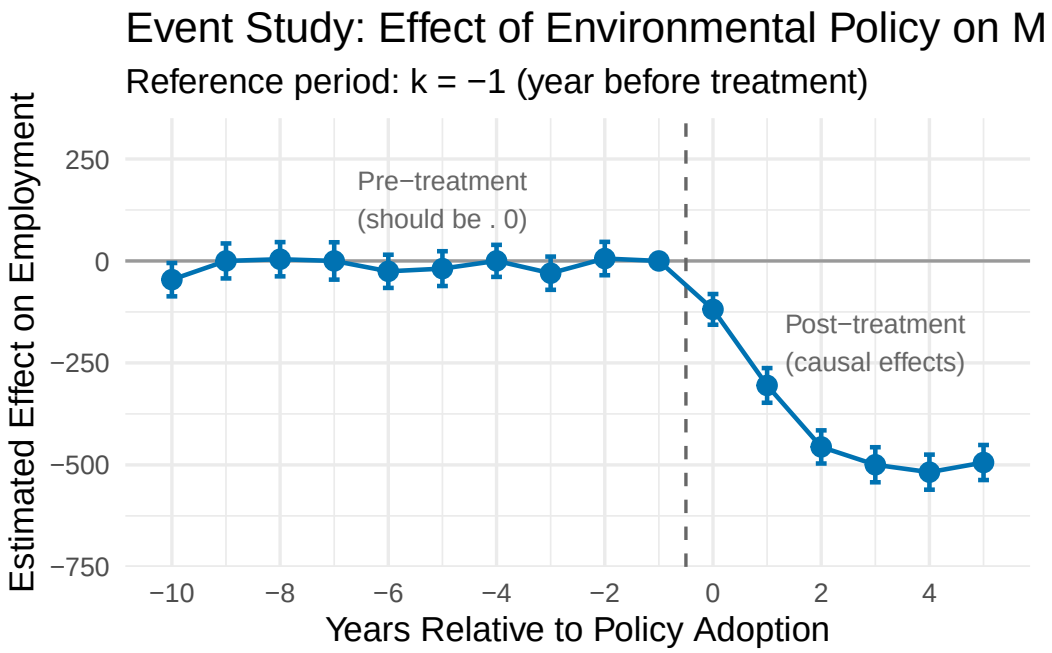



```
# Estimate dynamic DiD (note: rel_t_treat_-1 is omitted as reference)
dyn_did <- feols(
    manufacturing_emp ~ `rel_t_treat_-10` + `rel_t_treat_-9` + `rel_t_treat_-8` +
        `rel_t_treat_-7` + `rel_t_treat_-6` + `rel_t_treat_-5` + `rel_t_treat_-4` +
        `rel_t_treat_-3` + `rel_t_treat_-2` + `rel_t_treat_0` + `rel_t_treat_1` +
        `rel_t_treat_2` + `rel_t_treat_3` + `rel_t_treat_4` + `rel_t_treat_5` |
        city_id + year,
    vcov = "HC1",
    data = env_data
)
```

12.2.6 The Event Study Plot

The best way to present dynamic DiD results is an **event study plot**. We plot each coefficient with its 95% confidence interval, with the x-axis showing relative time and a vertical line at the treatment date.

Figure 12.4: Event study plot showing treatment effects over time. Pre-treatment coefficients near zero support parallel trends; post-treatment coefficients show the growing negative effect.



12.2.7 Interpreting the Event Study

The event study plot reveals several important patterns:

1. **Pre-treatment coefficients (k < -1):** These cluster around zero with confidence intervals that include zero. This is strong evidence for parallel trends—before the policy, treated and control cities were on similar trajectories.
2. **Reference period (k = -1):** This is normalized to zero by construction.
3. **Post-treatment coefficients (k 0):** These show the causal effect of the policy. The effect starts at about -100 in the adoption year and grows to about -500 by year 3, where it levels off.

> **i** Why Pre-Trends Matter
>
> The pre-treatment coefficients serve as a **placebo test**. If we found large, significant "effects" before the policy was implemented, that would suggest our control group is invalid—the groups were already diverging for reasons unrelated to the treatment.

12.3 Cluster-Robust Standard Errors

12.3.1 The Problem: Non-Independent Observations

Standard OLS assumes observations are independent. But in panel data, observations from the same unit across time are often correlated. Students in the same classroom share a teacher; workers in the same firm share management; residents of the same state share policies.

When errors are **correlated within clusters**, standard errors from regular OLS are too small. This leads to:

- T-statistics that are too large
- P-values that are too small
- Confidence intervals that are too narrow
- **Too many false positives** (Type I errors)

Figure 12.5: With clustered data, observations within groups are correlated. Regular standard errors don't account for this, leading to overconfidence.

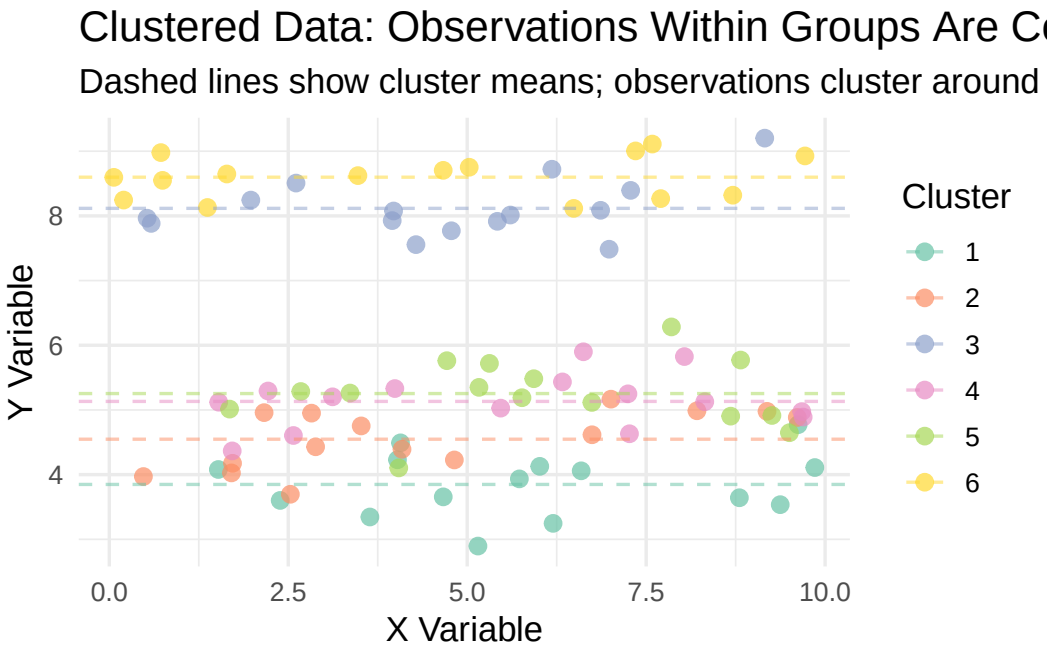
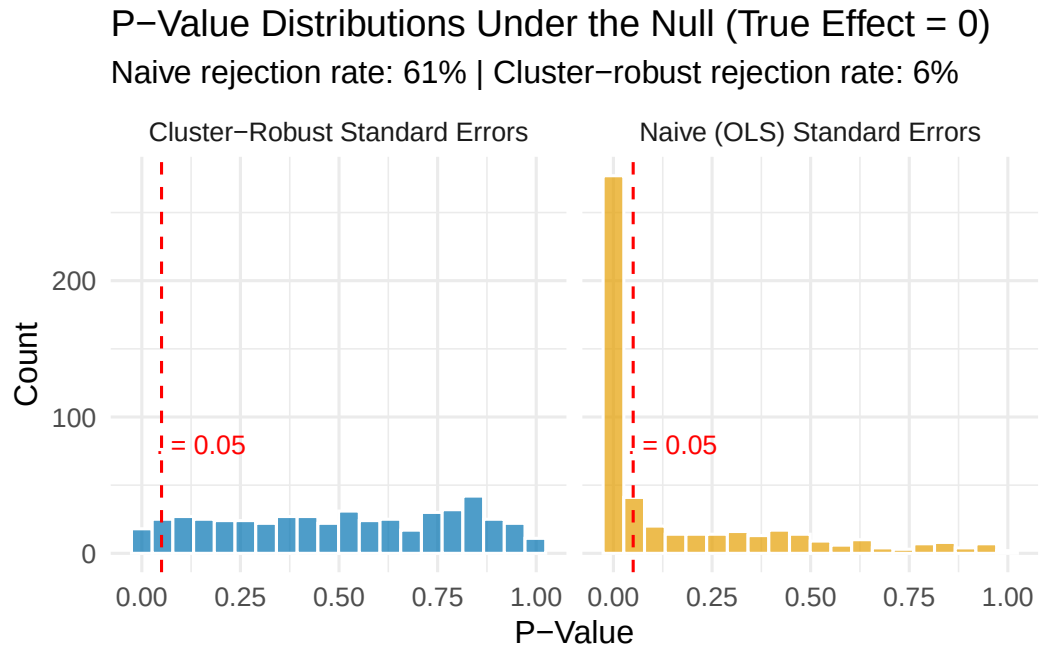


Figure 12.6: Distribution of p-values from 500 simulations where the true effect is zero. Naive standard errors (left) produce too many small p-values; cluster-robust standard errors (right) are properly calibrated.



12.3.2 Visualizing the Problem

12.3.3 Simulation: The Dangers of Ignoring Clustering

Let's simulate what happens when we ignore clustering. We'll generate data where the **true treatment effect is zero**, so any "significant" result is a false positive.

The simulation shows that:

- With **naive standard errors**: We reject the null 61% of the time (should be 5%)
- With **cluster-robust standard errors**: We reject 6% of the time (close to 5%)

Ignoring clustering leads to a **massive inflation of false positives**!

12.3.4 Implementing Cluster-Robust Standard Errors

In `fixest`, cluster-robust standard errors are easy to implement using the `cluster` argument:

```
# Standard (naive) standard errors
model_naive <- feols(y ~ treatment | state + year,
                     vcov = "HC1",
                     data = my_data)

# Cluster-robust standard errors (clustered at state level)
model_cluster <- feols(y ~ treatment | state + year,
                       cluster = "state",
                       data = my_data)
```

> Rule of Thumb: Cluster at the Treatment Level
>
> Cluster your standard errors at the level where treatment is assigned. If policies are set at the state level, cluster by state. If a program is implemented at the school level, cluster by school.

12.4 Summary

This chapter covered three important extensions to basic DiD:

Propensity Score Weighting (IPW-DiD) improves the counterfactual by up-weighting control units that are similar to treated units. When treated and control groups differ systematically, naive DiD violates parallel trends; IPW helps restore balance. The algorithm involves estimating propensity scores, computing inverse probability weights, and using those weights in the TWFE regression.

Dynamic DiD and Event Studies allow treatment effects to vary over time. By estimating separate effects for each period relative to treatment, we can see whether effects grow, shrink, or remain stable. Crucially, the pre-treatment coefficients serve as a placebo test: if they're significantly different from zero, parallel trends is violated.

Cluster-Robust Standard Errors correct for correlated errors within groups. When observations within clusters (states, schools, firms) are correlated, standard OLS inference is overconfident, leading to inflated false positive rates. Clustering at the treatment level provides valid inference.

These tools are essential for credible DiD analysis in practice.

12.5 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

13 Instrumental Variables

Key Questions

- What is endogeneity and why does it cause OLS to fail?
- What are instrumental variables and what makes an instrument valid?
- How does Two-Stage Least Squares (2SLS) work?
- What is the Local Average Treatment Effect (LATE)?
- How do we test whether an instrument is strong enough?

Suggested Readings

- TBD

We've built up a powerful toolkit for causal inference: randomized experiments, multivariate regression, fixed effects, and difference-in-differences. But what happens when none of these methods are available?

Sometimes we have only cross-sectional data, treatment is clearly endogenous, we cannot randomize, and there's no natural experiment to exploit. In these situations, **instrumental variables (IV)** offers a path forward—if we can find the right instrument.

13.1 The Fundamental Problem: Endogeneity

13.1.1 When OLS Fails

Recall that OLS provides unbiased estimates when the zero conditional mean assumption holds: $E[\mu|X] = 0$. When this assumption fails—when X is correlated with the error term—we say X is **endogenous**, and OLS is biased.

Endogeneity arises from three main sources:

1. Omitted Variable Bias

We want to estimate the effect of education on wages, but ability affects both education choices and wages. Since we can't measure ability, it's absorbed into the error term, biasing our estimate.

2. Selection Bias

We want to estimate the effect of health insurance on health outcomes, but healthier people are more likely to purchase insurance. Health status drives insurance decisions, contaminating our comparison.

3. Reverse Causality

We want to estimate the effect of police on crime, but high-crime areas get more police deployed. The causation runs both ways, making it impossible to isolate the effect of policing.

13.1.2 Formalizing the Problem

Suppose we want to estimate:

$$y = \beta_0 + \beta_1 x + \mu$$

When $E[\mu|x] \neq 0$, the variable x is **endogenous**. OLS will be biased and inconsistent—no matter how large our sample, we won't converge to the true β_1.

We cannot interpret the OLS coefficient as a causal effect.

13.2 The Instrumental Variables Solution

13.2.1 The Key Insight

The IV approach finds a variable Z that "shakes up" X in a way that's unrelated to the error term. Think of it as finding a source of **exogenous variation** in X—variation that comes from outside the system and isn't contaminated by the factors causing endogeneity.

13.2.2 Two Requirements for a Valid Instrument

For Z to be a valid instrument, it must satisfy two conditions:

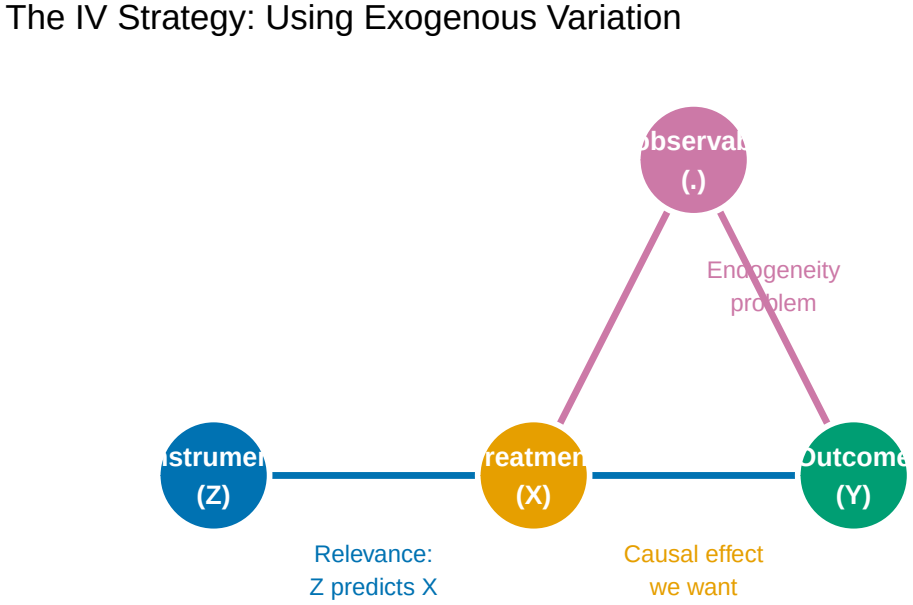
> **!** The Two IV Assumptions
>
> **1. Relevance:** $Cov(Z, X) \neq 0$
> The instrument must be correlated with the endogenous variable. This is **testable**—we can check whether Z predicts X.
> **2. Exclusion Restriction:** $Cov(Z, \mu) = 0$

The instrument affects Y **only through** X. There is no direct effect of Z on Y. This is **not directly testable**—it requires economic reasoning and assumption.

13.2.3 Graphical Intuition

The IV strategy can be visualized as a causal diagram:

Figure 13.1: The instrumental variables strategy. Z affects Y only through X (exclusion restriction), while unobservables ( ) affect both X and Y, creating endogeneity.



The key insight: X has two types of variation—"good" variation unrelated to μ, and "bad" variation correlated with μ. OLS uses both and is therefore biased. IV uses only the "good" variation induced by Z, giving us an unbiased estimate.

13.3 Application: Fertility and Labor Supply

13.3.1 The Research Question

What is the causal effect of having children on mothers' labor force participation? This question is crucial for understanding gender wage gaps, designing parental leave policies, and evaluating childcare subsidies.

But we cannot randomize fertility! And the decision to have children is deeply endogenous:

- **Reverse causality:** Women may time children based on career factors
- **Selection:** Less career-oriented women may choose larger families
- **Omitted variables:** Unobserved preferences affect both fertility and work

13.3.2 The Angrist and Evans (1998) Instrument

In a landmark study, Angrist and Evans (1998) found a clever instrument for fertility: the **sex composition of the first two children**.

The instrument works because parents prefer mixed-sex children. If your first two children are the same sex (both boys or both girls), you're more likely to try for a third child to "get" the other sex.

Why is this relevant? $(Cov(Z, X) \neq 0)$

- Parents with same-sex children are more likely to have a third child
- This is an empirical fact we can verify

Why is this excludable? $(Cov(Z, \mu) = 0)$

- Sex of children is essentially random (about 50% boys, 50% girls)
- Hard to imagine why same-sex vs. mixed-sex would *directly* affect labor supply
- The only plausible channel is through fertility decisions

13.3.3 Simulating the Angrist-Evans Setup

Let's create simulated data that mirrors the Angrist-Evans setting to illustrate how IV works:

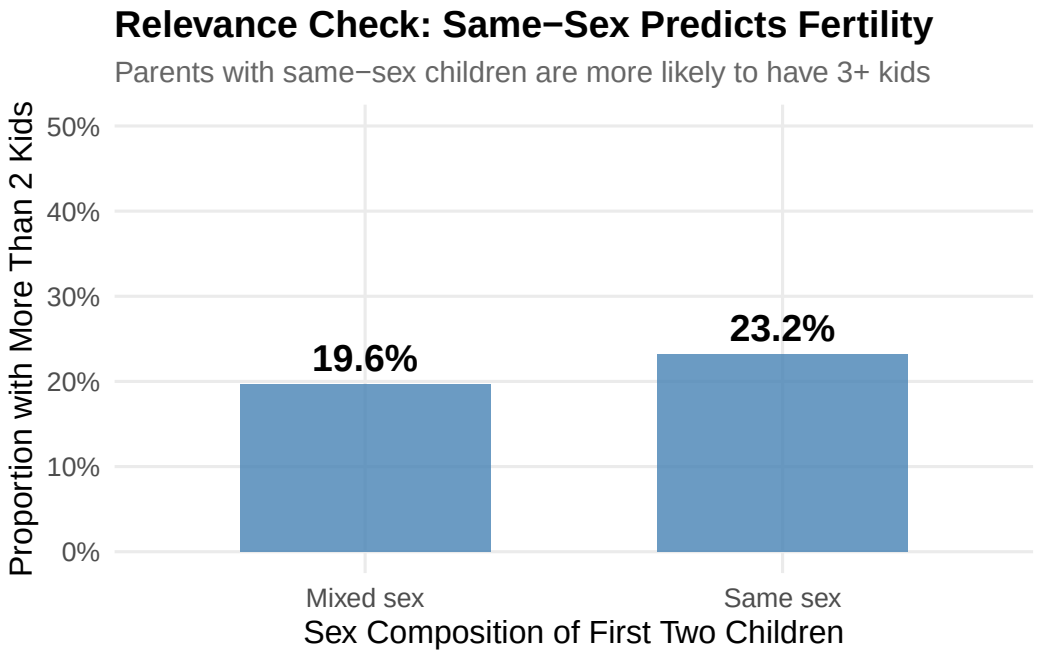
13.3.4 Testing the Relevance Condition

First, let's verify that same-sex siblings predict having more than two children:

```
# Check relevance: does samesex predict morekids?
relevance_check <- ae_sim |>
    group_by(samesex_lab) |>
    summarize(prop_morekids = mean(morekids))

relevance_check
```

```
# A tibble: 2 x 2
  samesex_lab prop_morekids
  <chr>               <dbl>
1 Mixed sex           0.196
2 Same sex            0.232
```

Figure 13.2: The relevance condition: parents with same-sex children are more likely to have a third child.



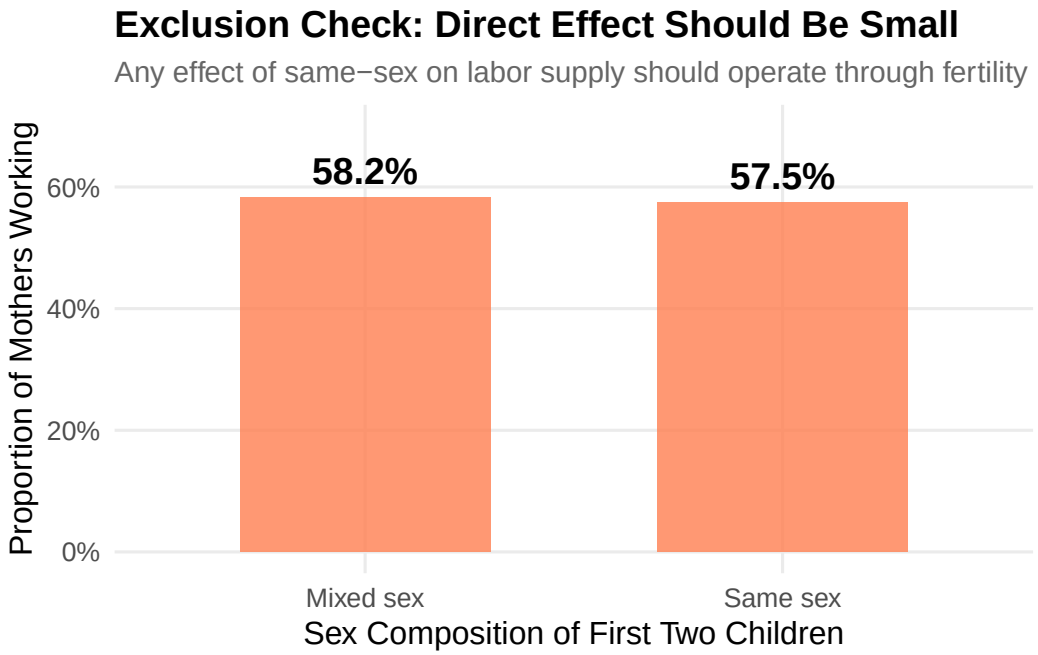
The difference is substantial—same-sex parents are about 7 percentage points more likely to have a third child. The instrument is relevant.

13.3.5 Testing Excludability (Sort of)

We cannot directly test the exclusion restriction, but we can check whether the instrument has a small *reduced-form* relationship with the outcome:

The difference is small (about 1-2 percentage points), consistent with the effect operating through fertility rather than directly. Of course, this doesn't *prove* excludability—we must rely on economic reasoning to argue there's no direct channel.

Figure 13.3: Checking excludability: the direct relationship between same-sex and labor supply is small, consistent with the effect operating through fertility.



13.4 Two-Stage Least Squares (2SLS)

13.4.1 The Estimation Strategy

IV models are typically estimated using **Two-Stage Least Squares (2SLS)**:

Stage 1 (First Stage): Regress the endogenous variable on the instrument:

$$X_i = \pi_0 + \pi_1 Z_i + \nu_i$$

Get predicted values: $\hat{X}_i = \hat{\pi}_0 + \hat{\pi}_1 Z_i$

These predicted values contain only the variation in X that comes from Z—the exogenous variation.

Stage 2 (Second Stage): Regress the outcome on the predicted values:

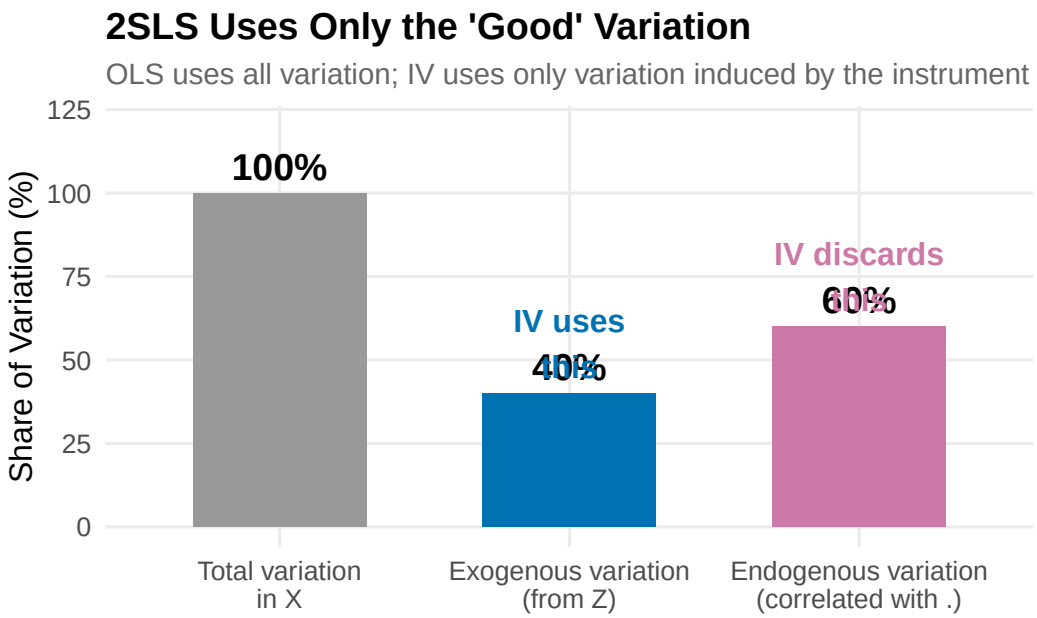
$$Y_i = \beta_0 + \beta_1 \hat{X}_i + \mu_i$$

The coefficient $\hat{\beta}_1^{2SLS}$ is our causal estimate.

13.4.2 Why Does This Work?

The first stage "purges" X of its problematic variation. Since Z is exogenous (by assumption), the predicted values $\hat{X}$ contain only exogenous variation. When we regress Y on $\hat{X}$, we're using only this "clean" variation, giving us an unbiased estimate.

Figure 13.4: 2SLS uses only the variation in X that comes from the instrument Z, discarding the endogenous variation.



13.4.3 Implementing 2SLS

Let's estimate the model step by step:

Step 1: The First Stage

```
# First stage: regress endogenous variable on instrument
first_stage <- lm(morekids ~ samesex, data = ae_sim)
summary(first_stage)
```

```
Call:
lm(formula = morekids ~ samesex, data = ae_sim)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-0.2316 -0.2316 -0.1963 -0.1963  0.8037

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)
(Intercept) 0.196288   0.002585  75.931 <0.0000000000000002 ***
samesex     0.035260   0.003664   9.623 <0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4096 on 49998 degrees of freedom
Multiple R-squared:  0.001849,  Adjusted R-squared:  0.001829
F-statistic: 92.61 on 1 and 49998 DF,  p-value: < 0.00000000000000022
```

The F-statistic on `samesex` is very large (over 100), indicating a strong instrument.

Step 2: Get Predicted Values

```
# Get predicted values from first stage
ae_sim <- ae_sim |>
    mutate(morekids_hat = predict(first_stage))

# Look at the predictions
ae_sim |>
    select(samesex, morekids, morekids_hat) |>
    head(10)
```

```
# A tibble: 10 x 3
   samesex morekids morekids_hat
     <int>    <int>        <dbl>
 1       1        0        0.232
 2       0        0        0.196
 3       0        1        0.196
 4       1        0        0.232
 5       1        0        0.232
 6       1        0        0.232
 7       1        0        0.232
 8       1        0        0.232
 9       0        0        0.196
10       0        0        0.196
```

Step 3: The Second Stage

```r
# Second stage: regress outcome on predicted values
# NOTE: Standard errors from this approach are WRONG
second_stage_manual <- lm(mom_worked ~ morekids_hat, data = ae_sim)
coef(second_stage_manual)["morekids_hat"]
```

```
morekids_hat
  -0.2002152
```

> **Standard Error Warning**
>
> The manual two-stage approach gives the correct coefficient, but the standard errors are **wrong** because they don't account for uncertainty in the first stage. Always use a proper 2SLS estimator that computes correct standard errors.

13.4.4 Proper 2SLS with `feols()`

The `fixest` package provides proper 2SLS estimation with correct standard errors:

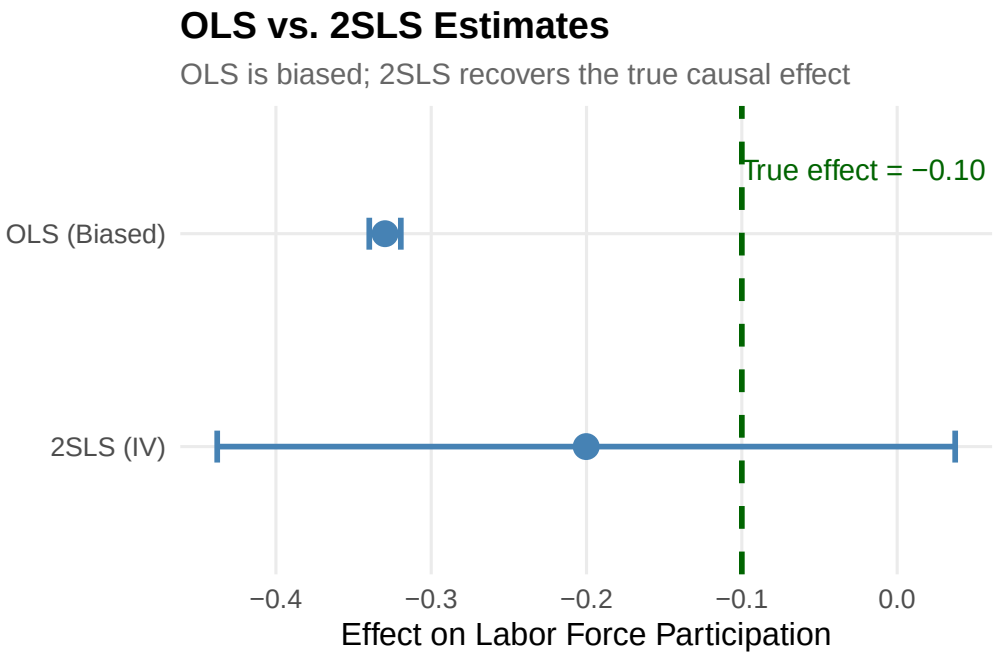
```r
# Proper 2SLS estimation
# Syntax: outcome ~ controls | fixed effects | endogenous ~ instrument
tsls <- feols(mom_worked ~ 1 |  # Just intercept (no controls)
                           0 |  # No fixed effects
                           morekids ~ samesex,  # First stage
              data = ae_sim)

summary(tsls)
```

```
TSLS estimation - Dep. Var.: mom_worked
                  Endo.    : morekids
                  Instr.   : samesex
Second stage: Dep. Var.: mom_worked
Observations: 50,000
Standard-errors: IID
              Estimate Std. Error  t value  Pr(>|t|)
(Intercept)   0.621714   0.026008 23.90457 < 2.2e-16 ***
fit_morekids -0.200215   0.121213 -1.65176   0.09859 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.477825   Adj. R2: 3.111e-5
F-test (1st stage), morekids: stat = 92.6067, p < 2.2e-16 , on 1 and 49,998 DoF.
                  Wu-Hausman: stat =  1.1581, p = 0.281862, on 1 and 49,997 DoF.
```

13.4.5 Comparing OLS and 2SLS

```
# OLS for comparison (biased)
ols <- lm(mom_worked ~ morekids, data = ae_sim)
```

Figure 13.5: Comparing OLS (biased) and 2SLS (unbiased) estimates. OLS overstates the negative effect because career orientation confounds the relationship.



The OLS estimate is more negative than the true effect (-0.10) because it's contaminated by selection: women with lower career orientation both have more children *and* are less likely to work, making the correlation more negative than the causal effect.

The 2SLS estimate is much closer to the true effect because it uses only the exogenous variation from the same-sex instrument.

13.5 Testing for Weak Instruments

13.5.1 The Weak Instrument Problem

If the instrument is only weakly correlated with X, 2SLS performs poorly:

- Estimates become biased (toward OLS!)

- Standard errors become unreliable
- Inference is invalid

13.5.2 The F-Test Rule of Thumb

Stock and Yogo (2005) established a rule of thumb:

> **Weak Instrument Test**
>
> The first-stage F-statistic should exceed **10** for the instrument to be considered strong. With $F < 10$, 2SLS estimates may be severely biased.

In our example, the first-stage F-statistic is well over 100—a very strong instrument.

13.5.3 What Happens with a Weak Instrument?

Let's see what happens when we use a weak (essentially random) instrument:

```
# Create a weak instrument (random noise)
set.seed(123)
ae_sim <- ae_sim |>
    mutate(weak_instrument = rnorm(n()))

# Try 2SLS with weak instrument
weak_iv <- feols(mom_worked ~ 1 | 0 | morekids ~ weak_instrument,
                 data = ae_sim)
```

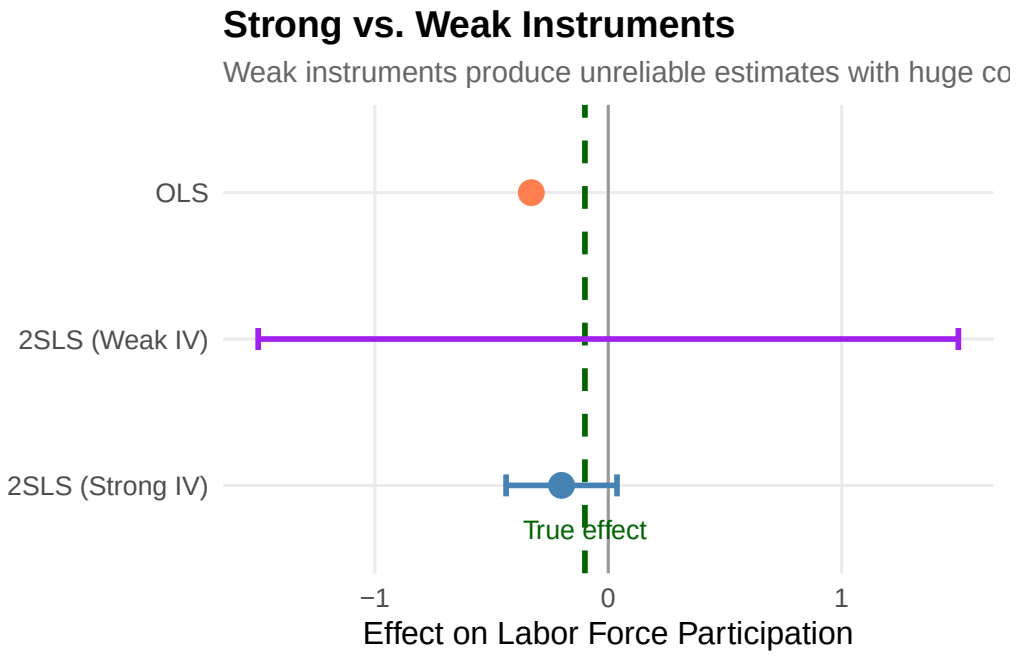
With a weak instrument, the estimate is wildly off and the confidence interval is enormous. **Always check your first-stage F-statistic!**

13.6 What Does IV Estimate? The LATE

13.6.1 The Local Average Treatment Effect

An important subtlety: IV does not estimate the same quantity as OLS or DiD. It estimates the **Local Average Treatment Effect (LATE)**—the causal effect for a specific subpopulation called **compliers**.

Figure 13.6: With a weak instrument (random noise), the 2SLS estimate becomes unreliable with huge standard errors.



13.6.2 Four Types of People

In the fertility example, we can categorize people by how their fertility responds to the instrument:

Type	Description
Always-takers	Would have 3+ kids regardless of sex composition
Never-takers	Would never have 3+ kids regardless of sex composition
Compliers	Have 3+ kids *because* first two were same-sex
Defiers	Have 3+ kids *only if* first two were different-sex

IV identifies the causal effect only for compliers—those whose treatment status was actually changed by the instrument.

13.6.3 Who Are the Compliers?

In the Angrist-Evans context, compliers are parents who:

- Wanted mixed-sex children strongly enough to try for a third

- Would have stopped at two kids if the first two were different sexes

This is **not**:

- All parents with 3+ children
- Parents who wanted large families regardless
- Parents who strongly preferred small families

13.6.4 Trade-offs of IV Estimation

Advantages	Disadvantages
Credible causal identification	Estimates LATE, not ATE
Handles endogeneity	Less generalizable
No need for RCT or natural experiment	Requires strong assumptions
Works with cross-sectional data	Often larger standard errors

13.7 Classic IV Applications

Instrumental variables have been used to study many important questions:

Study	Question	Instrument
Card (1995)	Returns to education	Distance to college
Acemoglu et al. (2001)	Do institutions cause growth?	Colonial settler mortality
Levitt (1997)	Effect of police on crime	Electoral timing
Angrist & Krueger (1991)	Returns to education	Quarter of birth

13.7.1 Example: Card (1995) - Returns to Education

Question: What is the causal effect of education on wages?

Problem: Ability is unobserved but affects both education and wages, biasing OLS upward.

Instrument: Distance to the nearest college. People who grew up closer to a college got more education.

Why relevant? Proximity reduces the cost of attending college.

Why excludable? Distance to a college shouldn't directly affect wages decades later (only through education).

Finding: Returns to education are actually *higher* than OLS suggests—the ability bias works in the opposite direction from what many expected.

13.8 Summary

Instrumental variables provide a powerful tool for causal inference when other methods fail. The key is finding a variable that affects treatment but has no direct effect on the outcome.

The two requirements:

1. **Relevance:** The instrument predicts the endogenous variable (testable via first-stage F-statistic > 10)
2. **Exclusion:** The instrument affects the outcome only through the treatment (requires economic reasoning—not directly testable)

Key insights:

- 2SLS uses only the exogenous variation in X induced by Z
- IV estimates the Local Average Treatment Effect (LATE) for compliers
- Weak instruments produce unreliable estimates—always check the first-stage F-statistic
- The exclusion restriction is crucial but untestable—think carefully about whether it holds

IV requires strong assumptions, but when those assumptions are credible, it provides a path to causal inference even in challenging observational settings.

13.9 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

References

Angrist, Joshua D., and Jörn-Steffen Pischke. 2014. *Mastering 'Metrics: The Path from Cause to Effect*. Princeton University Press.

Bailey, Michael A. 2020. *Real Econometrics: The Right Tools to Answer Important Questions*. Oxford University Press.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Incorporated.

Wooldridge, Jeffrey M. 2019. *Introductory Econometrics: A Modern Approach*. 7th ed. Cengage Learning.

A Intro to R and RStudio

Key Questions

- How do I install R and RStudio on my computer?
- What are the main components of the RStudio interface?
- How do I write and run R code?
- What are objects and how do I use them?
- How do I install and load R packages?

Suggested Readings

- Wickham, Çetinkaya-Rundel, and Grolemund (2023), Ch. 1-3

This appendix will walk you through installing R and RStudio, understanding the RStudio interface, and writing your first lines of R code. By the end of this chapter, you will be ready to start working with data and running econometric analyses.

A.1 Installing R

To get started with R, you first need to download and install the base R software. R is available for free from CRAN (the Comprehensive R Archive Network). Navigate to https://cloud.r-project.org/ and click the download link for your operating system (Windows, macOS, or Linux). Follow the installation instructions provided for your system.

Once the installation is complete, you will have R installed on your computer. However, we will not typically interact with R directly. Instead, we will use RStudio, an integrated development environment (IDE) that makes working with R much more convenient.

A.2 Installing RStudio

RStudio is a powerful IDE designed specifically for working with R. It provides a user-friendly interface with helpful features like syntax highlighting, code completion, and integrated help documentation. You can download RStudio Desktop for free from https://posit.co/download/

rstudio-desktop/. The website should automatically detect your operating system and show you the appropriate download link. Click the link and follow the installation instructions.

> **!** Only Use RStudio
>
> After installing both R and RStudio, you will only ever need to open RStudio. RStudio runs R in the background, so you don't need to open the base R application separately. Feel free to delete the shortcut for base R from your desktop if you wish. When you open an R script file (`.R`) for the first time, be sure to set RStudio as the default program to open such files.

A.3 The RStudio Interface

When you first open RStudio, you will see an interface divided into several panels (or "panes"), as seen in Figure A.1. Each panel serves a different purpose in your workflow. Understanding these panels is essential for working efficiently in R.

A.3.1 The Script Editor

The Script Editor is where you write your R code in script files. A script file is simply a text file containing R code that can be saved, edited, and run again later. Working in script files is essential for reproducibility—you can always go back and see exactly what you did, make changes, and re-run your analysis. To open a new script file, go to File → New File → R Script, or use the keyboard shortcut Cmd+Shift+N (Mac) or Ctrl+Shift+N (Windows/Linux).

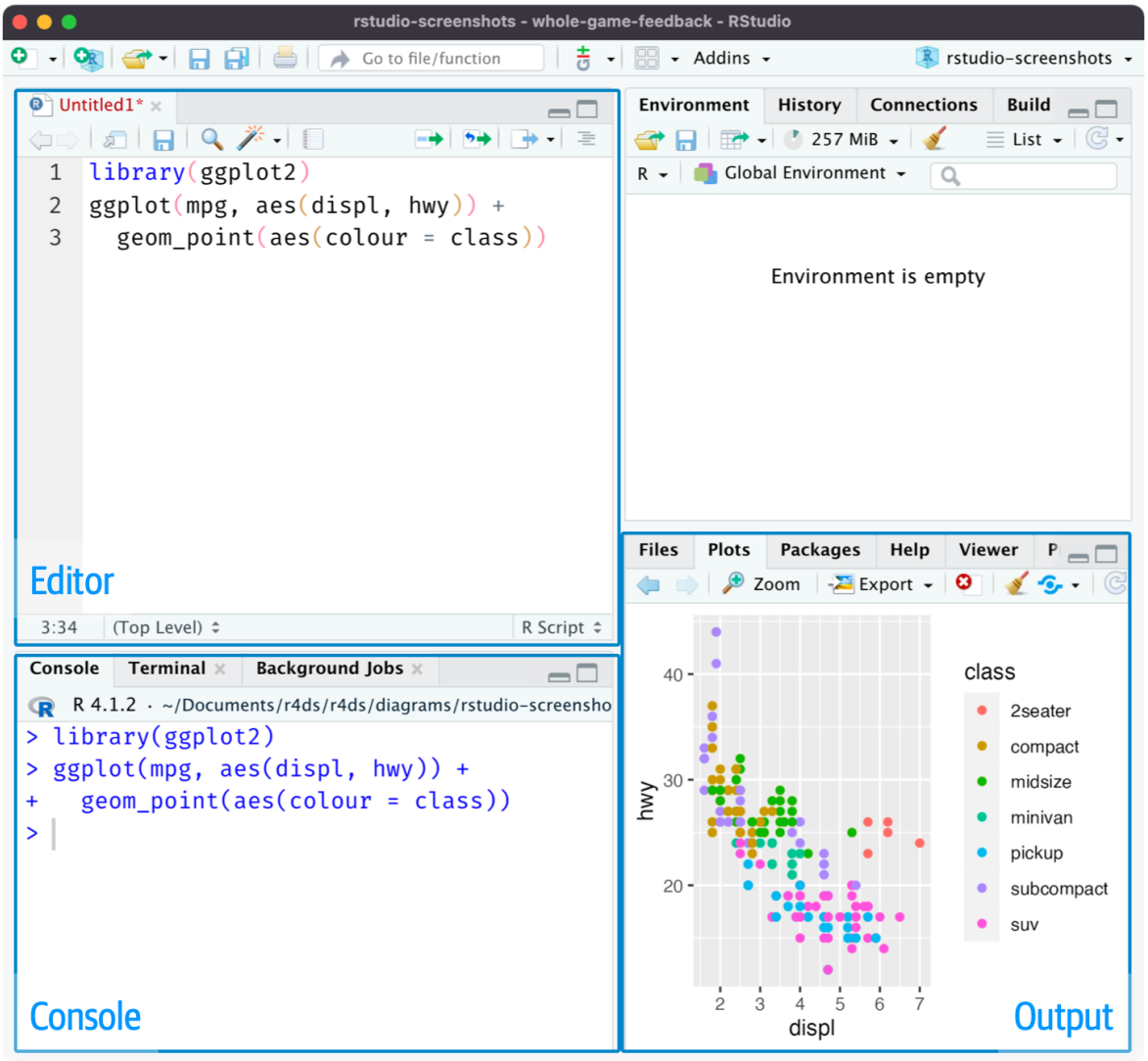
A.3.2 The Console

The Console is where R actually executes your code. When you run code from a script, it gets sent to the console line by line. You can also type code directly into the console and press Enter to run it immediately. However, code typed directly into the console is not saved, so this approach is generally only recommended for quick tests or exploratory work. Output from many functions, including regression results, will be printed to the console.

A.3.3 The Environment Panel

The Environment panel displays all the "objects" you have created in your current R session. These objects might include data frames, vectors, or stored values. You can see each object's name along with a brief summary of its contents. This panel is useful for keeping track of what data and variables you currently have available.

Figure A.1: The RStudio interface with its four main panels labeled.



A.3.4 The Plots and Output Panel

This panel serves multiple purposes. When you create visualizations (like scatter plots or histograms), they will appear here. This panel also contains tabs for browsing files on your computer, viewing installed packages, and accessing R's built-in help documentation.

> **Customizing Your Layout**
>
> You can hide or show different panels under the "View" menu. This is useful when you want more screen space for a particular task, such as focusing on your script while writing code.

A.4 Basic Coding in R

Now that RStudio is set up, let's write some code! We'll start with simple operations in the console to get familiar with R's syntax.

A.4.1 Arithmetic Operations

R can function as a sophisticated calculator. Try typing the following expressions in the console and pressing Enter:

```
2 + 2
```

```
[1] 4
```

```
20 * 5 / 87
```

```
[1] 1.149425
```

```
sin(pi / 2)
```

```
[1] 1
```

```
sqrt(6)
```

```
[1] 2.44949
```

As you can see, R can handle basic arithmetic as well as mathematical functions like `sin()` and `sqrt()`. The `pi` you see in the third example is a built-in constant in R representing the mathematical constant π.

A.4.2 Objects and Assignment

R is an "object-oriented language," meaning that data, variables, and nearly everything else is stored as an "object." You create objects using the assignment operator `<-`. The assignment operator takes whatever is on the right side and stores it in the name on the left side.

```
x <- 2 + 5
```

After running this code, the value `7` is now stored in an object called `x`. Notice that R didn't print anything—it just stored the value. To see what's stored in an object, type its name and press Enter:

```
x
```

```
[1] 7
```

You can use objects in subsequent calculations, just like variables in algebra:

```
x + 10
```

```
[1] 17
```

You can also update an object by reassigning it:

```
x <- x + 50
x
```

```
[1] 57
```

> **Naming Objects**
>
> When creating object names, keep them short but descriptive. Object names cannot contain spaces. Some good examples include: `flight_data`, `reg_controls_1`, `acs_male`, or `X_mat`.

A.4.3 Vectors

A vector is the simplest type of object in R. Vectors contain a sequence of values of the same type (all numbers, all text, etc.). You create vectors using the `c()` function, which stands for "combine":

```
primes <- c(2, 3, 5, 7, 11, 13)
primes
```

```
[1]  2  3  5  7 11 13
```

One powerful feature of R is that arithmetic operations on vectors are applied element-by-element:

```
primes * 2
```

```
[1]  4  6 10 14 22 26
```

```
primes - 1
```

```
[1]  1  2  4  6 10 12
```

You can also perform arithmetic between two vectors of the same length:

```
odds <- c(1, 3, 5, 7, 9, 11)
primes + odds
```

```
[1]  3  6 10 14 20 24
```

A.4.4 Functions

R has a large collection of built-in functions that perform specific tasks. Functions are called using the syntax: `function_name(argument1 = value1, argument2 = value2, ...)`.

For example, the `seq()` function generates a sequence of numbers:

```
bb <- seq(from = 1, to = 30, by = 3)
bb
```

```
 [1]  1  4  7 10 13 16 19 22 25 28
```

Here, `from`, `to`, and `by` are the *arguments* of the function. We set `from = 1` to start at 1, `to = 30` to end at 30, and `by = 3` to increment by 3.

> **Getting Help**
>
> To learn more about any function, type `?` followed by the function name in the console. For example, `?seq` will open the documentation for the `seq()` function.

A.5 Data Types in R

Every piece of data in R has a **type** that determines how it can be used. Understanding data types is essential because certain operations only work with certain types, and R will sometimes convert between types automatically in ways that can cause unexpected results.

A.5.1 Numeric Types

R has two main numeric types:

Integer (`int`): Whole numbers without decimal places. Created by adding `L` after a number:

```
my_integer <- 5L
typeof(my_integer)
```

```
[1] "integer"
```

Double (`dbl`): Numbers with decimal places (also called "floating point"). This is the default for most numbers:

```
my_double <- 5.7
typeof(my_double)
```

```
[1] "double"
```

In practice, the distinction between integers and doubles rarely matters for econometric analysis—R handles conversions automatically. When you see `<int>` or `<dbl>` in your data, both are simply "numbers."

A.5.2 Character Strings

Character (`chr`): Text data, always enclosed in quotes:

```
my_name <- "Economics"
typeof(my_name)
```

```
[1] "character"
```

Character strings are used for labels, names, and categorical data that hasn't been converted to a factor.

A.5.3 Logical Values

Logical (`lgl`): TRUE or FALSE values (can be abbreviated as `T` and `F`):

```
is_enrolled <- TRUE
typeof(is_enrolled)
```

```
[1] "logical"
```

Logical values are often created by comparisons:

```
5 > 3
```

```
[1] TRUE
```

```
10 == 10  # Note: == tests equality, = is for assignment
```

```
[1] TRUE
```

Logical values are essential for filtering data (e.g., "keep only observations where income > 50000").

A.5.4 Factors

Factor (`fct`): Categorical variables with a fixed set of possible values (called "levels"). Factors are crucial in regression because R treats them differently from numeric variables:

```r
# Create a factor for education level
edu_level <- factor(c("High School", "College", "Graduate", "College", "High School"))
edu_level
```

```
[1] High School College     Graduate    College     High School
Levels: College Graduate High School
```

Notice how R shows the "Levels"—the complete set of possible categories. Factors are especially important for dummy variables in regression.

> **!** Why Factors Matter
>
> When you include a factor in a regression, R automatically creates dummy variables for each level. If you include a character variable, R may treat it as text rather than a categorical variable, leading to errors or unexpected results. We will talk more about this in section Chapter 7.

A.5.5 Checking and Converting Types

You can check an object's type with `typeof()` or `class()`:

```r
class(edu_level)
```

```
[1] "factor"
```

Convert between types using `as.numeric()`, `as.character()`, `as.factor()`, etc.:

```r
# Convert character to factor
major <- c("Economics", "Math", "Economics", "Physics")
major_factor <- as.factor(major)
major_factor
```

```
[1] Economics Math      Economics Physics
Levels: Economics Math Physics
```

```r
# Convert numeric to character
numbers <- c(1, 2, 3)
as.character(numbers)
```

```
[1] "1" "2" "3"
```

A.5.6 Common Type Abbreviations

When working with data frames, you'll frequently see these abbreviations:

Table A.1: Common data type abbreviations in R

Abbreviation	Type	Description	Example
`<dbl>`	Double	Numbers with decimals	`3.14`, `2.5`
`<int>`	Integer	Whole numbers	`1`, `42`
`<chr>`	Character	Text strings	`"hello"`, `"NY"`
`<fct>`	Factor	Categorical variable	`factor("low", "med", "high")`
`<lgl>`	Logical	TRUE/FALSE	`TRUE`, `FALSE`
`<date>`	Date	Calendar dates	`as.Date("2024-01-15")`

A.5.7 Data Frames

Data frames are the primary way that structured, tabular data is stored in R. A data frame is essentially a table where each column is a vector and each row is an observation. R comes with several built-in data frames for practice, including `mtcars`, which contains data on various car models.

Use `head()` to view the first few rows of a data frame:

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

You can explore the structure of a data frame using several helpful functions:

```
nrow(mtcars)
```

```
[1] 32
```

```r
ncol(mtcars)
```

```
[1] 11
```

```r
colnames(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

To access a specific column (variable) within a data frame, use the `$` operator:

```r
head(mtcars$mpg)
```

```
[1] 21.0 21.0 22.8 21.4 18.7 18.1
```

Here, `mtcars$mpg` extracts the `mpg` column from the `mtcars` data frame as a vector. I wrapped it in `head()` to show only the first few values.

A.5.8 Comments

In R, anything preceded by the `#` symbol is treated as a comment and will not be executed. Comments are essential for documenting your code—explaining what each section does, leaving notes for yourself, or marking areas that need revision.

```r
# Find the mean miles per gallon
mean(mtcars$mpg)
```

```
[1] 20.09062
```

Good commenting habits make your code readable to others (and to your future self).

A.6 Working with R Scripts

While the console is useful for quick tests, you should do the vast majority of your work in script files. Scripts allow you to save your code, edit it easily, and reproduce your entire analysis by running the script from top to bottom.

A.6.1 Creating and Running Scripts

To create a new script, go to File → New File → R Script (or use Cmd/Ctrl + Shift + N). Write your code in the script editor, then use the following keyboard shortcuts to run it:

Cmd/Ctrl + Enter executes the current line (or highlighted selection) in the console. The cursor then automatically moves to the next line, making it easy to step through your code line by line.

Figure A.2: Example R script with cursor position shown. Pressing Cmd/Ctrl + Enter will run the complete command that creates the `not_cancelled` object, then move the cursor to the next statement.

```
library(dplyr)
library(nycflights13)

not_cancelled <- flights |>
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled |>
  group_by(year, month, day) |>
  summarize(mean = mean(dep_delay))
```

Cmd/Ctrl + Shift + S runs the entire script from top to bottom. This is useful for checking that your complete analysis works correctly and produces reproducible results.

A.6.2 Script Best Practices

Following good practices when writing scripts will save you time and frustration. Always start your script with your name and a brief description of the script's purpose in comments. Load all required packages at the beginning of the script using `library()` so anyone reading your code knows what dependencies are needed. Never include `install.packages()` in your script—you only need to install packages once on your computer, and you don't want your script to install packages on someone else's machine without their permission. Save your scripts with short, descriptive names that contain no spaces, such as `assignment_3.R` or `ec_325_data_plotting.R`.

A.7 Working Directories

R uses a concept called the "working directory"—the folder on your computer where R looks for files you want to load and where it saves files you create. You can see your current working directory at the top of the console, or by running:

```
getwd()
```

A good workflow is to save your R script in a dedicated folder for this course (or a subfolder for each week's work), put all related data files in the same folder, and then set your working directory to that location. In RStudio, go to Session → Set Working Directory → To Source File Location to automatically set the working directory to wherever your script is saved.

A.8 R Packages

R comes with a set of "base" packages that provide fundamental functionality. However, the true power of R lies in its vast ecosystem of user-contributed packages. These packages extend R's capabilities for specialized tasks—from data visualization to complex statistical modeling.

A.8.1 Installing Packages

You only need to install a package once on your computer. Use the `install.packages()` function:

```
install.packages("tidyverse")
```

A.8.2 The Tidyverse

The tidyverse is a collection of R packages designed for data science. These packages share a common philosophy and work seamlessly together, making data manipulation and visualization more intuitive. Some of the most commonly used tidyverse packages include `dplyr` (data manipulation), `ggplot2` (visualization), and `readr` (reading data files).

Installing `tidyverse` installs all of these packages at once.

A.8.3 Loading Packages

Once a package is installed, you need to load it each time you start a new R session using `library()`:

```
library(tidyverse)
```

When you load the `tidyverse`, you'll see a message listing which packages were attached. You may also see messages about "conflicts"—this just means some `tidyverse` functions have the same name as functions in base R, and the `tidyverse` versions will take precedence. This is normal and expected behavior.

A.8.4 The Pipe Operator

One of the most useful features of the `tidyverse` is the **pipe operator**: `|>`. The pipe takes the result of whatever is on its left and passes it as the first argument to the function on its right.

Think of the pipe like a conjunction in a sentence. Just as "and" or "then" connect actions in English, the pipe connects operations in R. Instead of saying "Take the data, then filter it, then summarize it," you write code that reads almost the same way:

```
data |>
  filter(year == 2020) |>
  summarize(mean_income = mean(income))
```

This code reads naturally from top to bottom: "Start with `data`, *then* filter to rows where year equals 2020, *then* summarize by calculating the mean income."

Without the pipe, you would need to either nest functions inside each other (hard to read) or create intermediate objects at each step (clutters your environment). The pipe lets you write code that mirrors how you think about the analysis.

> **i** Base R Pipe vs. Magrittr Pipe
>
> R now has a built-in pipe (`|>`) that works without loading any packages. The `tidyverse` historically used `%>%` from the `magrittr` package, which you may see other people use. Both work similarly for most purposes. We'll use the base R pipe `|>` in this course.

> **!** Remember the Difference
>
> `install.packages()` downloads and installs a package to your computer (do this once). `library()` loads an already-installed package into your current R session (do this every time you start R).

A.9 Summary and Conclusion

This appendix walked you through the essential first steps for working with R. You installed R and RStudio, learned how the RStudio interface is organized, and wrote your first R code. You now understand how to create and manipulate objects, work with vectors and data frames, write and run scripts, and install packages.

A.9.1 Key Takeaways

- **RStudio is your interface to R:** Always work in RStudio, not base R. The script editor, console, environment, and plots panels each serve distinct purposes in your workflow.
- **Scripts are essential for reproducibility:** Write your code in script files that you can save, edit, and re-run. Use comments to document what your code does.
- **Objects store everything in R:** Use the assignment operator `<-` to create objects. Vectors hold sequences of values; data frames hold structured tabular data.
- **Packages extend R's capabilities:** Install packages once with `install.packages()`, then load them each session with `library()`. The tidyverse is a particularly useful collection of packages for data science.
- **Working directories matter:** Set your working directory to the folder containing your script and data files to keep your analysis organized.

With these foundations in place, you are ready to start loading real data, creating visualizations, and running econometric analyses.

A.10 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

B Statistics and Probability Review

Key Questions

- What are the properties of the summation operator?
- What are the key rules for logarithms, and why do economists use logs so often?
- How do we interpret derivatives and partial derivatives?
- How do we find the maximum or minimum of a function?
- What is the difference between a population and a sample?
- What is a random variable and how do we describe its distribution?
- How do we measure central tendency, variability, and relationships between variables?
- What is the normal distribution and why is it important?

Suggested Readings

- Wooldridge (2019), Appendix A, B, and C

This appendix reviews the essential mathematical and statistical concepts that form the foundation for econometrics. If you've taken introductory statistics or calculus, much of this material will be familiar. However, econometrics uses these tools in specific ways, so it's worth reviewing them with an eye toward how they'll appear throughout the course.

B.1 Basic Mathematical Tools

B.1.1 The Summation Operator

The summation operator provides a compact way to write the sum of a sequence of numbers. If $\{x_i : i = 1, 2, ..., n\}$ is a sequence of n numbers, then the sum of these numbers is written as:

$$x_1 + x_2 + ... + x_n = \sum_{i=1}^{n} x_i$$

The symbol Σ (capital Greek letter sigma) tells us to add up the terms. The expression $i = 1$ below the sigma indicates that we start counting at $i = 1$, and the n above tells us to stop at $i = n$.

B.1.1.1 Properties of Summation

The summation operator has several useful properties that we will use frequently.

Property 1: Summing a constant. If we add a constant c to itself n times, we get n times the constant:

$$\sum_{i=1}^{n} c = nc$$

Property 2: Factoring out constants. We can pull constants outside the summation:

$$\sum_{i=1}^{n} cx_i = c \sum_{i=1}^{n} x_i$$

Property 3: Separating additive terms. Sums can be split across addition:

$$\sum_{i=1}^{n} (ax_i + by_i) = a \sum_{i=1}^{n} x_i + b \sum_{i=1}^{n} y_i$$

⚠ Common Mistakes with Summation

These properties do **not** extend to division or multiplication in the ways you might expect. We **cannot** break up divided expressions:

$$\sum_{i=1}^{n} \frac{x_i}{y_i} \neq \frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} y_i}$$

We **cannot** break up multiplicative expressions:

$$\sum_{i=1}^{n} x_i^2 \neq \left(\sum_{i=1}^{n} x_i \right)^2$$

B.1.1.2 The Sample Mean

Given n numbers $\{x_i : i = 1, 2, ..., n\}$, we can compute their **average** or **mean** value by adding them up and dividing by n. The average value, denoted $\bar{x}$ (read "x-bar"), is written as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The average is an example of a **descriptive statistic**: a value that describes or summarizes a characteristic of a set of data points. In this case, the mean describes the **central tendency** of x_i—where the "middle" of the data lies.

B.1.1.3 Useful Summation Results

Two results involving summations will appear repeatedly in this course.

Result 1: Deviations from the mean sum to zero. The sum of the deviations $(x_i - \bar{x})$ from the mean always equals zero:

$$\sum_{i=1}^{n} (x_i - \bar{x}) = 0$$

This can be proven using the properties above:

$$\sum_{i=1}^{n} (x_i - \bar{x}) = \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \bar{x} = \sum_{i=1}^{n} x_i - n\bar{x} = n\bar{x} - n\bar{x} = 0$$

The last step follows because $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ implies $n\bar{x} = \sum_{i=1}^{n} x_i$.

Result 2: Sum of squared deviations. The sum of squared deviations can be rewritten as:

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$$

More generally, for two variables x and y:

$$\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}$$

These identities are useful for computing variances and covariances, which we will discuss shortly.

B.1.2 Derivatives

The derivative $\frac{dy}{dx}$ of a function $y = f(x)$ measures its **instantaneous rate of change**. It tells you how sensitive the function's output is to a small change in its input. In econometrics, derivatives help us interpret how changes in independent variables affect dependent variables.

Here are derivatives of some common functional forms in econometrics:

Function	Derivative
$y = \beta_0 + \beta_1 x$	$\frac{dy}{dx} = \beta_1$
$y = \beta_0 + \beta_1 x + \beta_2 x^2$	$\frac{dy}{dx} = \beta_1 + 2\beta_2 x$
$y = \beta_0 + \frac{\beta_1}{x}$	$\frac{dy}{dx} = -\frac{\beta_1}{x^2}$
$y = \beta_0 + \beta_1 \ln(x)$	$\frac{dy}{dx} = \frac{\beta_1}{x}$

Notice that in the linear case, the derivative is simply the coefficient β_1—the effect of x on y is constant. In the other cases, the derivative depends on the value of x, meaning the effect of x on y varies depending on where you start.

B.1.3 Percentages and Percent Changes

Understanding percentages is essential for econometrics. Many economic variables are reported as percentages (unemployment rate, interest rates, inflation), and we frequently describe changes in percentage terms. Getting the language right matters—confusing "percent" with "percentage points" is a common source of errors.

B.1.3.1 Percentages

A **percentage** expresses a number as a fraction of 100. To convert a decimal to a percentage, multiply by 100:

$$0.25 = 25\%$$

To convert a percentage to a decimal, divide by 100:

$$25\% = 0.25$$

In econometrics, we typically work with decimals in our calculations and convert to percentages for interpretation.

B.1.3.2 Percent Change

The **percent change** (or **percentage change**) measures how much a variable changed relative to its initial value:

$$\text{Percent Change} = \frac{\text{New Value} - \text{Old Value}}{\text{Old Value}} \times 100\%$$

Or more compactly:

$$\%\Delta Y = \frac{Y_2 - Y_1}{Y_1} \times 100\%$$

Example: If wages increase from \$20 to \$22 per hour:

$$\%\Delta wage = \frac{22 - 20}{20} \times 100\% = \frac{2}{20} \times 100\% = 10\%$$

Wages increased by **10 percent**.

⚠ Percent Change vs. Percentage Point Change

These are **not** the same thing! This distinction trips up many students.

- **Percent change:** The change relative to the starting value
- **Percentage point change:** The arithmetic difference between two percentages

Example: If the unemployment rate rises from 4% to 5%:

- The **percentage point change** is: $5\% - 4\% = 1$ percentage point
- The **percent change** is: $\frac{5-4}{4} \times 100\% = 25\%$

Both statements are correct:

- "Unemployment rose by 1 percentage point"
- "Unemployment rose by 25 percent"

They mean very different things! In news and policy discussions, this distinction matters enormously.

B.1.3.3 When to Use Which

Use **percentage points** when:

- Describing changes in variables already measured as percentages (interest rates, unemployment rates, test scores out of 100)
- You want to describe the absolute change in the percentage

Use **percent change** when:

- Describing changes in levels (wages, prices, GDP, population)
- You want to describe the relative or proportional change

Example: An interest rate cut from 5% to 4% is:

- A 1 percentage point decrease
- A 20% decrease (both are correct, but mean different things)
- ~~A 1% decrease~~ (this is ambiguous and often wrong)

B.1.3.4 Calculating Values After Percent Changes

If Y increases by $p\%$, the new value is:

$$Y_{new} = Y_{old} \times (1 + p/100)$$

If Y decreases by $p\%$, the new value is:

$$Y_{new} = Y_{old} \times (1 - p/100)$$

Example: If a \$50 item increases in price by 20%:

$$Price_{new} = 50 \times (1 + 0.20) = 50 \times 1.20 = \$60$$

B.1.3.5 Compounding Percent Changes

Percent changes don't add—they compound. If wages increase by 10% and then by another 10%, the total increase is **not** 20%.

$$\text{Final} = \text{Initial} \times 1.10 \times 1.10 = \text{Initial} \times 1.21$$

The total increase is **21%**, not 20%.

This is why we use logarithms: log changes **do** add up, making them convenient for analyzing growth over time.

B.1.4 Logarithms

Logarithms appear constantly in econometrics. We use them to model percentage changes, interpret coefficients as elasticities, and transform skewed variables like income and wages. Understanding log rules is essential for working with regression models.

B.1.4.1 What is a Logarithm?

The natural logarithm, denoted $\ln(x)$, is the inverse of the exponential function e^x. If $y = \ln(x)$, then $e^y = x$.

Some key values to remember:

- $\ln(1) = 0$ because $e^0 = 1$
- $\ln(e) = 1$ because $e^1 = e$
- $\ln(x)$ is only defined for $x > 0$

B.1.4.2 Essential Log Rules

Three rules form the foundation for working with logarithms:

Rule 1: Product Rule. The log of a product is the sum of the logs:

$$\ln(ab) = \ln(a) + \ln(b)$$

Rule 2: Quotient Rule. The log of a ratio is the difference of the logs:

$$\ln\left(\frac{a}{b}\right) = \ln(a) - \ln(b)$$

Rule 3: Power Rule. The log of a power brings the exponent down:

$$\ln(a^k) = k\ln(a)$$

These rules are used constantly when manipulating regression equations and interpreting coefficients.

B.1.4.3 Logs and Percentage Changes

Here is the most important property of logs for econometrics:

> **!** The Key Insight
>
> For small changes, the **difference in logs approximately equals the percentage change**:
>
> $$\ln(Y_2) - \ln(Y_1) \approx \frac{Y_2 - Y_1}{Y_1}$$
>
> In other words: $\Delta ln(Y) \approx$ % **change in Y**

This approximation works well for changes up to about 20%. It follows from the quotient rule:

$$\ln(Y_2) - \ln(Y_1) = \ln\left(\frac{Y_2}{Y_1}\right)$$

When Y_2 is close to Y_1, the ratio Y_2/Y_1 is close to 1, and $\ln(1+x) \approx x$ for small x.

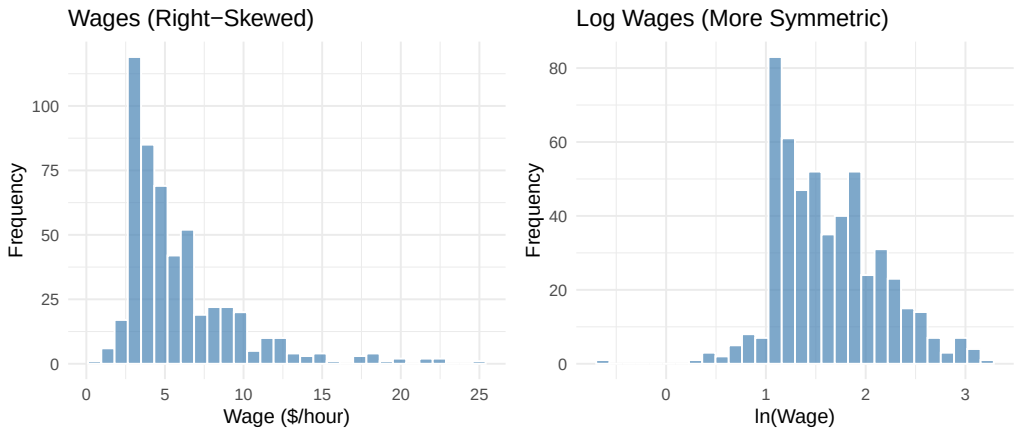
Example: If $\ln(wage)$ increases from 2.50 to 2.58, the change is 0.08. This means wages increased by approximately **8%**.

B.1.4.4 Other Uses of Logs

Logarithms are useful in economics for several reasons:

1. **Percentage interpretation:** Coefficients in log models have natural percentage interpretations (more on this in Chapter 7).
2. **Elasticities:** In a log-log model like $\ln(Y) = \beta_0 + \beta_1 \ln(X)$, the coefficient β_1 is the **elasticity**—the percentage change in Y for a 1% change in X.
3. **Reducing skewness:** Variables like income, wages, and prices are often right-skewed. Taking logs makes them more symmetric and closer to normal.
4. **Multiplicative relationships:** Logs transform multiplicative relationships into additive ones, which are easier to estimate with linear regression.

Figure B.1: Left: The distribution of wages is right-skewed. Right: The distribution of log wages is more symmetric.



B.1.5 Partial Derivatives

When y is a function of multiple variables, say $y = f(x_1, x_2)$, we use **partial derivatives** to measure how y changes with respect to one variable while holding the others constant.

The partial derivative of y with respect to x_1, holding x_2 constant, is denoted $\frac{\partial y}{\partial x_1}$. Similarly, the partial derivative with respect to x_2, holding x_1 constant, is $\frac{\partial y}{\partial x_2}$.

For example, if $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$, then:

$$\frac{\partial y}{\partial x_1} = \beta_1, \qquad \frac{\partial y}{\partial x_2} = \beta_2$$

The concept of "holding other variables constant" is central to econometrics. When we estimate a regression coefficient, we are trying to measure the partial effect of one variable on the outcome, controlling for other factors.

B.1.6 Optimization: Maxima and Minima

In econometrics, we often need to find the value of a variable that maximizes or minimizes a function. For example, Ordinary Least Squares (OLS) regression finds coefficient estimates by minimizing the sum of squared residuals. Understanding optimization is essential for understanding how these estimates are derived.

B.1.6.1 Finding Critical Points

To find the maximum or minimum of a function $f(x)$, we take the derivative, set it equal to zero, and solve for x. The solutions are called **critical points**.

For example, consider the quadratic function:

$$f(x) = ax^2 + bx + c$$

Taking the derivative and setting it equal to zero:

$$\frac{df}{dx} = 2ax + b = 0$$

Solving for x:

$$x^* = -\frac{b}{2a}$$

This critical point x^* is where the function reaches its maximum or minimum.

B.1.6.2 Determining Maximum vs. Minimum

How do we know if a critical point is a maximum or a minimum? We use the **second derivative test**. The second derivative tells us about the curvature of the function:

$$\frac{d^2f}{dx^2} = 2a$$

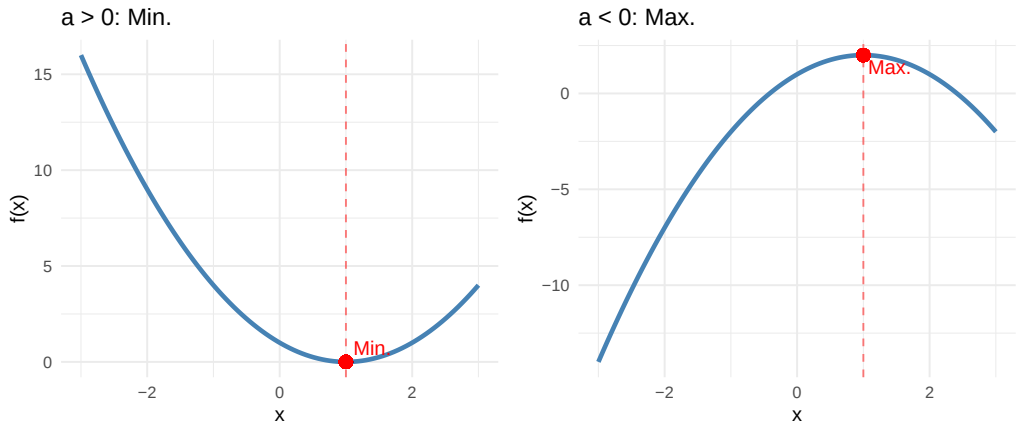
The rule is:

- If $\frac{d^2f}{dx^2} > 0$ at the critical point, the function is **concave up** (bowl-shaped), and we have a **minimum**
- If $\frac{d^2f}{dx^2} < 0$ at the critical point, the function is **concave down** (hill-shaped), and we have a **maximum**

For our quadratic function $f(x) = ax^2 + bx + c$:

- If $a > 0$: The parabola opens upward, and $x^* = -\frac{b}{2a}$ is a **minimum**
- If $a < 0$: The parabola opens downward, and $x^* = -\frac{b}{2a}$ is a **maximum**

Figure B.2: Left: When a > 0, the parabola opens upward and the critical point is a minimum. Right: When a < 0, the parabola opens downward and the critical point is a maximum.



Optimization in OLS

In OLS regression, we minimize the sum of squared residuals, which is a quadratic function of the coefficients. The second derivative is positive (the function is convex), confirming that the solution is indeed a minimum. This is why setting the first derivative equal to zero gives us the best-fitting line.

B.1.6.3 Optimization with Multiple Variables

When a function depends on multiple variables, we find critical points by setting all partial derivatives equal to zero simultaneously. For a function $f(x_1, x_2)$:

$$\frac{\partial f}{\partial x_1} = 0 \quad \text{and} \quad \frac{\partial f}{\partial x_2} = 0$$

Determining whether a critical point is a maximum, minimum, or neither (a saddle point) requires examining the second partial derivatives, which we will encounter when deriving the OLS estimator.

B.2 Population vs. Sample

Before diving into probability, it's important to understand the distinction between a **population** and a **sample**, as this distinction underlies all of statistical inference.

The **population** is the entire group of individuals, objects, or observations that we are interested in studying. For example, if we want to understand the wages of all U.S. workers, the population is every person employed in the United States. If we want to know the effect of a drug on blood pressure, the population might be all adults with hypertension.

In practice, we almost never observe the entire population. Instead, we collect data on a **sample**—a subset of the population. We might survey 5,000 workers rather than all 160 million, or run a clinical trial with 500 patients rather than everyone with hypertension.

> **!** The Goal of Statistical Inference
>
> The fundamental goal of statistics is to use information from a **sample** to draw conclusions about the **population**. We observe sample statistics (like the sample mean $\bar{x}$) and use them to make inferences about population parameters (like the population mean μ).

This distinction affects our notation:

Concept	Population Parameter	Sample Statistic
Mean	μ	$\bar{x}$
Variance	σ^2	s^2
Standard deviation	σ	s
Correlation	ρ	r

Population parameters are typically denoted with Greek letters and are usually unknown. Sample statistics are computed from our data and are used to estimate the population parameters.

For example, if we want to know the average wage of all U.S. workers (μ), we collect a sample and compute the sample mean ($\bar{x}$). Under certain conditions, $\bar{x}$ is a good estimate of μ. Much of econometrics is about understanding when and how well sample statistics estimate population parameters, and quantifying the uncertainty in those estimates.

B.3 Probability Basics

B.3.1 Random Variables

A **random variable** is a variable whose value is determined by a random phenomenon. It represents a number, but the exact value is determined by chance. Examples include the outcome of a dice roll, whether a coin lands heads or tails, or the percentage of flights that arrive on time.

Most variables we work with in econometrics can be thought of as random variables. A person's wage, years of education, or health status all have elements of randomness—we cannot perfectly predict them in advance.

B.3.2 Probability

Probability quantifies uncertainty. If X is a random variable that takes on k possible values $\{x_1, x_2, ..., x_k\}$, then the probability that X takes a particular value x_j is denoted:

$$p_j = P(X = x_j)$$

Each probability p_j must be between 0 and 1, and the probabilities must sum to 1: $p_1 + p_2 + ... + p_k = 1$.

B.3.3 Probability Distributions

The **probability density function (pdf)**, denoted $f(x)$, gives the probability that a random variable takes on a particular value x. For discrete random variables, $f(x) = P(X = x)$.

The **cumulative distribution function (cdf)**, denoted $F(x)$, gives the probability that the random variable is less than or equal to x:

$$F(x) = P(X \leq x)$$

The cdf is useful for answering questions like "what is the probability that X is below some threshold?"

B.3.4 Expected Values

The **expected value** (or **mathematical expectation**) of a random variable X, denoted $E[X]$, represents the "average" outcome we would expect if we could repeat the random process many times. It is our population "best guess" for the value of X. We often denote it with μ or μ_X.

For a discrete random variable with pdf $f(x)$, the expected value is the weighted average of all possible values, where the weights are the probabilities:

$$E[X] = \sum_{j=1}^{k} x_j f(x_j) = \mu$$

Example: Let X be the outcome of rolling a fair six-sided die. Each value (1 through 6) has probability 1/6. The expected value is:

$$E[X] = \frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) = 3.5$$

Notice that 3.5 is not a possible outcome of any single roll, but it is the average we would expect over many rolls.

B.3.4.1 Properties of Expected Values

The properties of expected values mirror those of the summation operator and are essential for working with regression models.

Property 1: Expected value of a constant. The expected value of a constant is simply that constant:

$$E[c] = c$$

This makes intuitive sense: if there is no randomness, the "expected" value is just the value itself.

Property 2: Multiplication by a constant. Constants can be factored out of the expectation operator:

$$E[cX] = cE[X]$$

If you double a random variable, its expected value doubles.

Property 3: Additivity (Linearity). The expectation of a sum equals the sum of expectations:

$$E[X + Y] = E[X] + E[Y]$$

Importantly, this property holds **regardless of whether X and Y are independent**. This is one of the most useful properties in econometrics.

Property 4: Combining the above. Combining Properties 1, 2, and 3, we get the general linearity property:

$$E[aX + bY + c] = aE[X] + bE[Y] + c$$

This is extremely useful for working with regression equations. If $Y = \beta_0 + \beta_1 X + u$, we can take expectations of both sides and apply this property.

Property 5: Product of independent variables. If X and Y are **independent** random variables, then:

$$E[XY] = E[X]E[Y]$$

> ⚠ Independence Required!
>
> This property only holds when X and Y are independent. If they are correlated, $E[XY] \neq E[X]E[Y]$ in general. In fact, $E[XY] - E[X]E[Y] = \text{Cov}(X, Y)$.

Property 6: Expectation of a function (Jensen's Inequality). In general, the expectation of a function is **not** equal to the function of the expectation:

$$E[g(X)] \neq g(E[X])$$

For example, $E[X^2] \neq (E[X])^2$. The difference between these two quantities is related to the variance: $E[X^2] - (E[X])^2 = \text{Var}(X)$.

B.4 Measures of Variability

Knowing the central tendency of a variable is useful, but it doesn't tell us everything. Two distributions can have the same mean but look very different if one is tightly clustered around the mean while the other is spread out. We need measures of **variability** or **dispersion**.

B.4.1 Variance

Let X be a random variable with mean $\mu = E[X]$. The **variance** of X, denoted σ^2 or $\text{Var}(X)$, measures how spread out the distribution is:

$$\text{Var}(X) = E[(X - \mu)^2] = \sigma^2$$

The variance is the expected squared deviation from the mean. Squaring serves two purposes: it ensures all deviations are positive (so positive and negative deviations don't cancel out), and it gives more weight to observations far from the mean.

B.4.2 Standard Deviation

The **standard deviation** of X, denoted $\text{sd}(X)$ or σ, is the positive square root of the variance:

$$\text{sd}(X) = \sqrt{\text{Var}(X)} = \sigma$$

The standard deviation is often more interpretable than the variance because it is measured in the same units as X itself. If X is measured in dollars, the standard deviation is also in dollars, whereas the variance would be in "dollars squared."

When computing the variance and standard deviation from a sample of n observations, we use the **sample variance** $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ and **sample standard deviation** $s = \sqrt{s^2}$. The $n-1$ in the denominator (rather than n) is called Bessel's correction and ensures the sample variance is an unbiased estimator of the population variance.

B.4.3 Covariance

A central concern in econometrics is understanding how variables move together. When X is large, is Y also large? Or is it small? Or is there no relationship at all?

The **covariance** quantifies the linear relationship between two random variables. Let X and Y be random variables with means $\mu_X = E[X]$ and $\mu_Y = E[Y]$. The covariance is defined as:

$$\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

The covariance measures how X and Y move together:

- **Positive covariance:** When X is above its mean, Y tends to be above its mean (and vice versa). The variables move in the same direction.
- **Negative covariance:** When X is above its mean, Y tends to be below its mean. The variables move in opposite directions.
- **Zero covariance:** There is no linear relationship between X and Y.

B.4.4 Correlation Coefficient

The covariance tells us the direction of the relationship, but its magnitude depends on the scales of X and Y, making it hard to interpret. The **correlation coefficient** standardizes the covariance to produce a unitless measure:

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\text{sd}(X) \cdot \text{sd}(Y)} = \rho_{XY}$$

The correlation coefficient is always between -1 and 1:

- $\rho_{XY} = 1$: Perfect positive linear relationship
- $\rho_{XY} = -1$: Perfect negative linear relationship
- $\rho_{XY} = 0$: No linear relationship

Perfect correlations rarely occur in practice. Values of ρ_{XY} closer to 1 or -1 indicate stronger linear relationships.

> **Correlation vs. Causation**
>
> Remember from Chapter 1: correlation does not imply causation! A strong correlation between X and Y tells us the variables move together, but it does not tell us whether X causes Y, Y causes X, or some third factor causes both.

B.4.5 Conditional Expectations

While correlation measures the overall linear relationship between two variables, we often want to describe how the expected value of one variable changes with another. This is where **conditional expectations** come in.

Let X and Y be random variables. The conditional expectation of Y given that X takes a specific value x is denoted:

$$E[Y|X = x]$$

This asks: what is our "best guess" for Y when we know that X equals x?

Example: What is the expected wage (Y) for people with a college degree ($X = 1$) versus those without ($X = 0$)? $E[Y|X = 1]$ and $E[Y|X = 0]$ answer these questions.

Conditional expectations are fundamental to regression analysis. When we estimate a regression of Y on X, we are essentially modeling how $E[Y|X]$ changes with X.

B.5 Distributions

B.5.1 The Normal Distribution

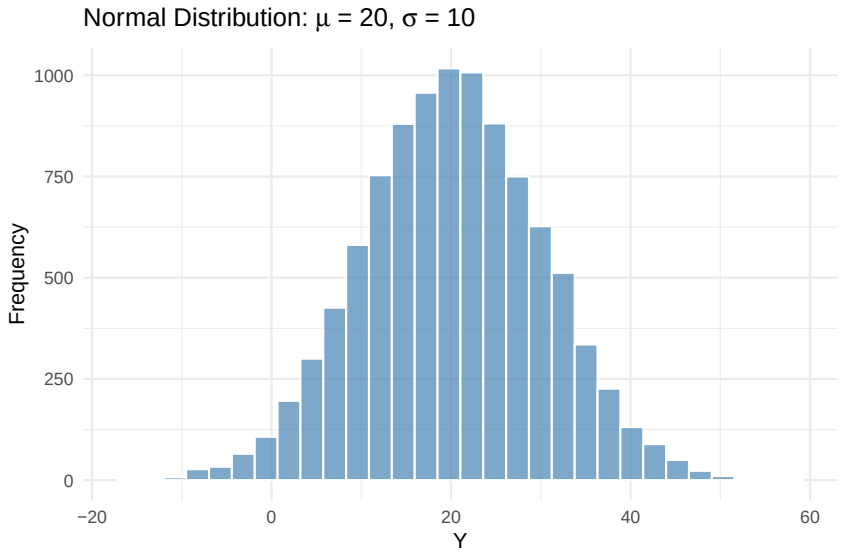
The **normal distribution** (also called the Gaussian distribution) is the most important distribution in statistics and econometrics. Many naturally occurring phenomena are approximately normally distributed, and many statistical procedures assume normality.

A random variable X with mean μ and variance σ^2 that follows a normal distribution is denoted:

$$X \sim N(\mu, \sigma^2)$$

The normal distribution has a characteristic "bell curve" shape: symmetric around the mean, with most observations clustered near the center and fewer observations in the tails.

Figure B.3: A histogram of 10,000 draws from a normal distribution with mean 20 and standard deviation 10.



Two key facts about the normal distribution:

- Approximately **68%** of observations lie within one standard deviation of the mean (between $\mu - \sigma$ and $\mu + \sigma$)
- Approximately **95%** of observations lie within two standard deviations of the mean (between $\mu - 2\sigma$ and $\mu + 2\sigma$)

B.5.2 The Standard Normal Distribution

The **standard normal distribution** is a special case of the normal distribution with mean 0 and variance 1:

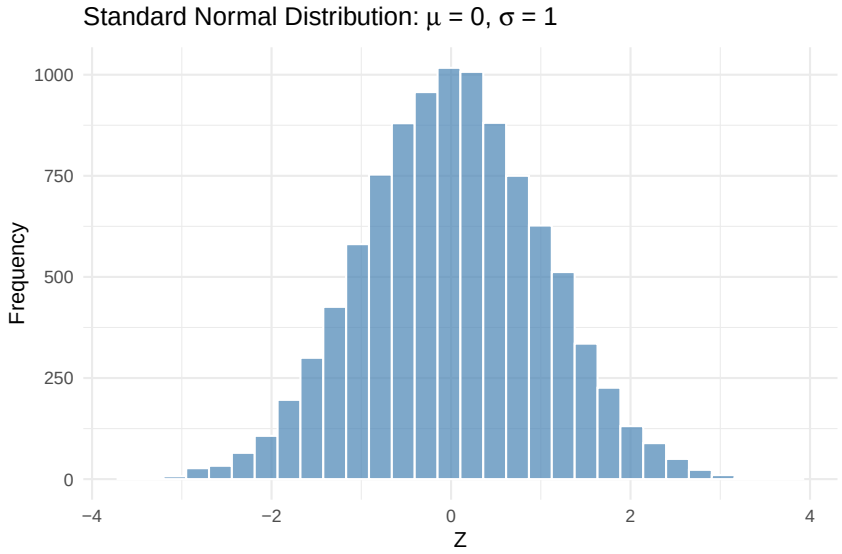
$$Z \sim N(0, 1)$$

The standard normal is important because any normal random variable can be converted to a standard normal through **standardization**. If $X \sim N(\mu, \sigma^2)$, then:

$$Z = \frac{X - \mu}{\sigma} \sim N(0, 1)$$

Standardization subtracts the mean (centering the distribution at zero) and divides by the standard deviation (scaling it to have unit variance).

Figure B.4: A histogram of 10,000 draws from a standard normal distribution (mean 0, standard deviation 1).



The standard normal is what z-tables (or statistical software) use to compute probabilities. Once you standardize a variable, you can look up the probability of observing values in any range.

B.6 Summary

This appendix reviewed the mathematical and statistical foundations needed for econometrics. We covered the summation operator and its properties, which are essential for understanding formulas throughout the course. We introduced derivatives and partial derivatives, which help us interpret how changes in one variable affect another, and learned how to find and classify maxima and minima of functions. We distinguished between populations and samples—a fundamental concept for statistical inference. We then turned to probability, defining random variables, expected values, variance, covariance, and correlation. Finally, we introduced the normal distribution, which plays a central role in statistical inference.

B.6.1 Key Formulas

Concept	Formula
Sample mean	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
Percent change	$\%\Delta Y = \frac{Y_2 - Y_1}{Y_1} \times 100\%$
Value after % increase	$Y_{new} = Y_{old} \times (1 + p/100)$
Log product rule	$\ln(ab) = \ln(a) + \ln(b)$
Log quotient rule	$\ln(a/b) = \ln(a) - \ln(b)$
Log power rule	$\ln(a^k) = k\ln(a)$
Logs and % changes	$\ln(Y_2) - \ln(Y_1) \approx \%\Delta Y$
Derivative of ln(x)	$\frac{d}{dx}\ln(x) = \frac{1}{x}$
Critical point (quadratic)	$x^* = -\frac{b}{2a}$ for $f(x) = ax^2 + bx + c$
Second derivative test	Max if $f''(x^*) < 0$; Min if $f''(x^*) > 0$
Expected value	$E[X] = \sum_{j=1}^{k} x_j f(x_j) = \mu$
E[constant]	$E[c] = c$
E[linear combo]	$E[aX + bY + c] = aE[X] + bE[Y] + c$
E[product] (indep.)	$E[XY] = E[X]E[Y]$ (if independent)
Variance	$\mathrm{Var}(X) = E[(X-\mu)^2] = \sigma^2$
Standard deviation	$\mathrm{sd}(X) = \sqrt{\mathrm{Var}(X)} = \sigma$
Covariance	$\mathrm{Cov}(X,Y) = E[(X-\mu_X)(Y-\mu_Y)]$
Correlation	$\rho_{XY} = \frac{\mathrm{Cov}(X,Y)}{\mathrm{sd}(X)\cdot\mathrm{sd}(Y)}$
Standardization	$Z = \frac{X-\mu}{\sigma}$

B.7 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

C Importing and Working with Data

Key Questions

- How do I import data files into R?
- What are README files and codebooks, and why do they matter?
- What are the first things I should do after loading a dataset?
- What is the difference between wide and long data?
- How do I reshape data between wide and long formats?
- How do I export data from R?

Suggested Readings

- Wickham, Çetinkaya-Rundel, and Grolemund (2023), Ch. 7-8, 11

Every empirical analysis starts with the same step: getting data into R. Whether you're working with a dataset from a published paper, government statistics, or data you collected yourself, you need to know how to import it, understand its structure, and get it into the right shape for your analysis. This appendix walks you through the entire process, from reading in a data file to reshaping and exporting your results.

C.1 README Files and Codebooks

Before you ever open a dataset in R, you should look for its **documentation**. Two types of documentation are especially important: the README file and the codebook.

C.1.1 README Files

A README file provides a high-level overview of a dataset or project. It typically contains information about where the data came from, when it was collected, who collected it, and any important notes about how the data should be used or cited. README files are usually plain text (`.txt`) or Markdown (`.md`) files.

A good README for a research dataset might look something like this:

```
================================================================
README: County-Level Birth Outcomes Data
================================================================

Source:       National Vital Statistics System (NVSS)
Years:        2015-2020
Unit:         County-year
Coverage:     All U.S. counties with 100+ births per year

Description:
  This dataset contains county-level measures of birth outcomes
  including low birth weight rates, preterm birth rates, and
  infant mortality rates. Data are derived from individual birth
  certificate records aggregated to the county level.

Files:
  birth_outcomes.csv     - Main analysis dataset
  county_covariates.csv  - County demographic characteristics
  codebook.txt           - Variable descriptions and coding

Notes:
  - Counties with fewer than 100 births in a given year are
    suppressed for privacy.
  - Infant mortality rates are per 1,000 live births.
  - FIPS codes follow 2015 county definitions.

Citation:
  National Center for Health Statistics. National Vital
  Statistics System, Natality Data, 2015-2020.
================================================================
```

Even if a dataset doesn't come with a README, you should create one for your own projects. When you come back to an analysis six months later, you will thank yourself for writing down where the data came from and what the key variables mean.

C.1.2 Codebooks

A **codebook** (sometimes called a data dictionary) is more detailed than a README. It describes every variable in the dataset: what it measures, what values it can take, how missing values are coded, and any important notes about interpretation.

Here's an example of what a codebook might look like for the birth outcomes dataset described above:

```
================================================================
CODEBOOK: birth_outcomes.csv
================================================================

Variable        Type      Description
----------------------------------------------------------------
fips            character 5-digit county FIPS code
state           character 2-letter state abbreviation
county_name     character County name
year            integer   Calendar year (2015-2020)
total_births    integer   Total number of live births
lbw_rate        numeric   Low birth weight rate (% of births
                          under 2,500 grams)
preterm_rate    numeric   Preterm birth rate (% of births
                          before 37 weeks gestation)
imr             numeric   Infant mortality rate
                          (deaths per 1,000 live births)
median_income   numeric   County median household income
                          (in 2020 dollars)
pct_uninsured   numeric   Percent of population uninsured
urban           integer   1 = metropolitan county,
                          0 = non-metropolitan county

Missing Values: Coded as NA
================================================================
```

> **!** Always Look for Documentation First
>
> Before you start analyzing any dataset, look for the README and codebook. Understanding what your variables actually measure—and how they are coded—is essential for avoiding errors in your analysis. For example, if you don't know that `imr` is measured per 1,000 live births, you might mistakenly interpret a coefficient as a percentage change.

C.2 Importing Data

The `tidyverse` includes a package called `readr` that provides fast, consistent functions for reading data files into R. When you load the `tidyverse` with `library(tidyverse)`, `readr` is automatically loaded for you.

C.2.1 CSV Files

The most common data format you'll encounter is the **CSV** (comma-separated values) file. A CSV file is essentially a plain text file where each row is an observation and each value is separated by commas. To read a CSV file, use `read_csv()`:

```
# Read a CSV file from your working directory
birth_data <- read_csv("birth_outcomes.csv")
```

When you run `read_csv()`, it will print a message showing you what type it assigned to each column. This is helpful for catching issues—for example, if a column you expected to be numeric shows up as character, that might indicate there are non-numeric entries (like "N/A" or "-") in the data that need to be addressed.

You can also read CSV files directly from a URL, which is convenient for datasets hosted online:

```
# Read a CSV file from a URL
storms_data <- read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/dplyr/storms.csv

head(storms_data)
```

```
# A tibble: 6 x 14
  rownames name   year month   day  hour   lat  long status       category  wind
     <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>           <dbl> <dbl>
1        1 Amy    1975     6    27     0  27.5 -79   tropical de~       NA    25
2        2 Amy    1975     6    27     6  28.5 -79   tropical de~       NA    25
3        3 Amy    1975     6    27    12  29.5 -79   tropical de~       NA    25
4        4 Amy    1975     6    27    18  30.5 -79   tropical de~       NA    25
5        5 Amy    1975     6    28     0  31.5 -78.8 tropical de~       NA    25
6        6 Amy    1975     6    28     6  32.4 -78.7 tropical de~       NA    25
# i 3 more variables: pressure <dbl>, tropicalstorm_force_diameter <dbl>,
#   hurricane_force_diameter <dbl>
```

C.2.2 Other Delimited Files

Not all text-based data files use commas as separators. Some use tabs, semicolons, or other characters. The `readr` package (included when loading the `tidyverse`) has functions for common variants:

```r
# Tab-separated files
my_data <- read_tsv("data_file.tsv")

# Semicolon-separated files (common in some European data)
my_data <- read_delim("data_file.csv", delim = ";")
```

C.2.3 Excel Files

Many datasets, especially those from government agencies or organizations, come in Excel format (`.xlsx` or `.xls`). To read Excel files, you need the `readxl` package, which is installed as part of the `tidyverse` but not loaded automatically:

```r
library(readxl)

# Read the first sheet of an Excel file
my_data <- read_excel("data_file.xlsx")

# Read a specific sheet
my_data <- read_excel("data_file.xlsx", sheet = "Sheet2")

# Read a specific sheet by number
my_data <- read_excel("data_file.xlsx", sheet = 2)
```

C.2.4 Stata and Other Statistical Software Files

If you're working with data from other statistical software, the `haven` package (also installed with `tidyverse` but not loaded automatically) can read files from Stata (`.dta`), SPSS (`.sav`), and SAS (`.sas7bdat`):

```r
library(haven)

# Read a Stata file
stata_data <- read_dta("data_file.dta")

# Read an SPSS file
spss_data <- read_sav("data_file.sav")
```

Stata files are particularly common in economics research, so `read_dta()` is a function you will use often.

C.2.5 R Data Files

R has its own file formats for saving data. The most common is `.rds`, which stores a single R object:

```
# Read an RDS file
my_data <- read_rds("data_file.rds")
```

You may also encounter `.RData` or `.rda` files, which can store multiple objects at once. These are loaded using `load()`:

```
# Load an RData file (objects are loaded into your environment
# with whatever names they were saved with)
load("data_file.RData")
```

Which Function Do I Use?

Here's a quick reference for which import function to use based on your file type:

Table C.1: Import functions by file type

File Extension	Function	Package
`.csv`	`read_csv()`	readr (tidyverse)
`.tsv`	`read_tsv()`	readr (tidyverse)
`.xlsx` / `.xls`	`read_excel()`	readxl
`.dta`	`read_dta()`	haven
`.sav`	`read_sav()`	haven
`.rds`	`read_rds()`	readr (tidyverse)
`.RData`	`load()`	base R

C.3 First Things to Do with Your Data

You've imported your data—now what? Before jumping into any analysis, you should take a few minutes to explore the dataset and make sure everything looks right. For a deeper discussion of why this matters—and how descriptive statistics and visualization can help you catch errors—see Chapter 2. Here we'll focus on the three functions you should run every time you load a new dataset.

Let's demonstrate using the `penguins` dataset from the `palmerpenguins` package:

```
library(palmerpenguins)
data(penguins)
```

C.3.1 `head()`: Look at the First Few Rows

The `head()` function shows you the first few rows of your data, giving you an immediate sense of what variables are present and what the values look like:

```
head(penguins)
```

```
# A tibble: 6 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
1 Adelie  Torgersen           39.1          18.7               181        3750
2 Adelie  Torgersen           39.5          17.4               186        3800
3 Adelie  Torgersen           40.3          18                 195        3250
4 Adelie  Torgersen           NA            NA                  NA          NA
5 Adelie  Torgersen           36.7          19.3               193        3450
6 Adelie  Torgersen           39.3          20.6               190        3650
# i 2 more variables: sex <fct>, year <int>
```

By default, `head()` shows 6 rows. You can change this with the `n` argument (e.g., `head(penguins, n = 10)`).

C.3.2 `glimpse()`: Get a Compact Overview

The `glimpse()` function shows every column's name, data type, and first several values. This is one of the most efficient ways to quickly understand a dataset's structure:

```
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
```

```
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

From this output, we can immediately see that the dataset has 344 rows and 8 columns, and we can verify that each variable has the type we'd expect (e.g., `species` is a factor, `bill_length_mm` is a double).

C.3.3 `summary()`: Check Key Statistics

The `summary()` function provides basic descriptive statistics for every variable—minimums, maximums, means, medians, and counts of missing values:

```
summary(penguins)
```

```
      species          island    bill_length_mm  bill_depth_mm
 Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10
 Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60
 Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30
                                 Mean   :43.92   Mean   :17.15
                                 3rd Qu.:48.50   3rd Qu.:18.70
                                 Max.   :59.60   Max.   :21.50
                                 NA's   :2       NA's   :2
 flipper_length_mm  body_mass_g       sex           year
 Min.   :172.0     Min.   :2700   female:165   Min.   :2007
 1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007
 Median :197.0     Median :4050   NA's  : 11   Median :2008
 Mean   :200.9     Mean   :4202                Mean   :2008
 3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009
 Max.   :231.0     Max.   :6300                Max.   :2009
 NA's   :2         NA's   :2
```

This lets you quickly verify that variable ranges make sense and spot any missing data. For example, we can see that `sex` has 11 `NA's`—something to keep in mind for our analysis.

C.4 Data Structure: Wide vs. Long

One of the most important concepts in data management is the distinction between **wide** and **long** data formats. The same information can be arranged in fundamentally different ways, and the format you need depends on what you're trying to do with the data. Understanding this distinction—and knowing how to switch between formats—is a crucial skill for applied econometrics.

C.4.1 Wide Data

In **wide format**, each row represents a single observational unit (like a country, state, or person), and each time period or category gets its own column. This is how data often appears in spreadsheets and published tables.

Here's an example of wide-format data showing GDP per capita for several countries across three years:

```
gdp_wide <- tibble(
  country = c("United States", "United Kingdom", "Germany", "Japan"),
  gdp_2018 = c(62887, 42943, 47603, 39287),
  gdp_2019 = c(65298, 42330, 46468, 40247),
  gdp_2020 = c(63544, 40285, 45724, 39539)
)

gdp_wide
```

```
# A tibble: 4 x 4
  country        gdp_2018 gdp_2019 gdp_2020
  <chr>             <dbl>    <dbl>    <dbl>
1 United States     62887    65298    63544
2 United Kingdom    42943    42330    40285
3 Germany           47603    46468    45724
4 Japan             39287    40247    39539
```

Wide data is easy to read as a table. You can quickly compare across years for a given country by scanning across a row, or compare across countries for a given year by scanning down a column.

C.4.2 Long Data

In **long format**, each row represents a single observation for a single unit at a single point in time. Instead of having separate columns for each year, there is one column that identifies the time period and another column that holds the value.

Here is the same GDP data in long format:

```
gdp_long <- tibble(
  country = rep(c("United States", "United Kingdom", "Germany", "Japan"), each = 3),
  year = rep(c(2018, 2019, 2020), times = 4),
  gdp_per_capita = c(62887, 65298, 63544,
```

```
                    42943, 42330, 40285,
                    47603, 46468, 45724,
                    39287, 40247, 39539)
)

gdp_long
```

```
# A tibble: 12 x 3
   country         year gdp_per_capita
   <chr>          <dbl>          <dbl>
 1 United States   2018          62887
 2 United States   2019          65298
 3 United States   2020          63544
 4 United Kingdom  2018          42943
 5 United Kingdom  2019          42330
 6 United Kingdom  2020          40285
 7 Germany         2018          47603
 8 Germany         2019          46468
 9 Germany         2020          45724
10 Japan           2018          39287
11 Japan           2019          40247
12 Japan           2020          39539
```

Long data has more rows but fewer columns. Each row is a single country-year observation. This format might seem less intuitive at first, but it is often the format that R functions—especially `ggplot2` for visualization and regression functions—expect.

C.4.3 When to Use Which Format

Wide format is natural for presenting data in tables, reading data quickly by eye, and working with data where you want to compare values across time periods within a row.

Long format is necessary for most data analysis in R. Specifically, `ggplot2` works best with long data because you can map variables like `year` to aesthetics (like color or facets). Regression functions like `lm()` and `feols()` require long data when you want to include time as a variable. The `dplyr` verbs like `group_by()` and `summarize()` are designed for long data.

As a general rule: if your data has column names that contain variable values (like `gdp_2018`, `gdp_2019`, `gdp_2020`), it is probably in wide format and may need to be converted to long format for analysis.

C.5 Reshaping Data with `pivot_longer()` and `pivot_wider()`

The `tidyr` package (part of the `tidyverse`) provides two functions for reshaping data: `pivot_longer()` converts wide data to long, and `pivot_wider()` converts long data to wide.

C.5.1 Wide to Long with `pivot_longer()`

Let's convert our wide GDP data to long format:

```
gdp_long <- gdp_wide |>
  pivot_longer(
    cols = starts_with("gdp_"),
    names_to = "year",
    values_to = "gdp_per_capita"
  )

gdp_long
```

```
# A tibble: 12 x 3
   country        year     gdp_per_capita
   <chr>          <chr>             <dbl>
 1 United States  gdp_2018          62887
 2 United States  gdp_2019          65298
 3 United States  gdp_2020          63544
 4 United Kingdom gdp_2018          42943
 5 United Kingdom gdp_2019          42330
 6 United Kingdom gdp_2020          40285
 7 Germany        gdp_2018          47603
 8 Germany        gdp_2019          46468
 9 Germany        gdp_2020          45724
10 Japan          gdp_2018          39287
11 Japan          gdp_2019          40247
12 Japan          gdp_2020          39539
```

Let's break down the three key arguments:

- `cols` tells R which columns to "pivot" or stack. Here we use `starts_with("gdp_")` to select all columns whose names begin with `"gdp_"`. You could also list the columns explicitly as `cols = c(gdp_2018, gdp_2019, gdp_2020)`.

- `names_to` is the name of the *new* column that will store the old column names. We call it `"year"` because the old column names (`gdp_2018`, `gdp_2019`, `gdp_2020`) represent years.
- `values_to` is the name of the *new* column that will store the values. We call it `"gdp_per_capita"`.

Notice that the `year` column still has the `gdp_` prefix attached. We can clean that up by adding `names_prefix` to strip the prefix:

```
gdp_long <- gdp_wide |>
  pivot_longer(
    cols = starts_with("gdp_"),
    names_to = "year",
    names_prefix = "gdp_",
    values_to = "gdp_per_capita"
  )

gdp_long
```

```
# A tibble: 12 x 3
   country        year  gdp_per_capita
   <chr>          <chr>          <dbl>
 1 United States  2018           62887
 2 United States  2019           65298
 3 United States  2020           63544
 4 United Kingdom 2018           42943
 5 United Kingdom 2019           42330
 6 United Kingdom 2020           40285
 7 Germany        2018           47603
 8 Germany        2019           46468
 9 Germany        2020           45724
10 Japan          2018           39287
11 Japan          2019           40247
12 Japan          2020           39539
```

Now the `year` column contains just the year numbers. However, notice that `year` is still a character type (because it was extracted from column names). We can convert it to numeric by adding `names_transform`:

```
gdp_long <- gdp_wide |>
  pivot_longer(
    cols = starts_with("gdp_"),
```

```
    names_to = "year",
    names_prefix = "gdp_",
    names_transform = list(year = as.integer),
    values_to = "gdp_per_capita"
  )

gdp_long
```

```
# A tibble: 12 x 3
   country         year gdp_per_capita
   <chr>          <int>          <dbl>
 1 United States   2018          62887
 2 United States   2019          65298
 3 United States   2020          63544
 4 United Kingdom  2018          42943
 5 United Kingdom  2019          42330
 6 United Kingdom  2020          40285
 7 Germany         2018          47603
 8 Germany         2019          46468
 9 Germany         2020          45724
10 Japan           2018          39287
11 Japan           2019          40247
12 Japan           2020          39539
```

Now `year` is a proper integer column, ready for use in regressions or plots.

C.5.2 A More Detailed Example

Let's work through a slightly more complex example. Suppose you have state-level data on unemployment rates and poverty rates across several years:

```
state_wide <- tibble(
  state = c("Maine", "Vermont", "New Hampshire"),
  unemp_2018 = c(3.4, 2.7, 2.5),
  unemp_2019 = c(3.0, 2.4, 2.6),
  unemp_2020 = c(5.4, 5.0, 6.2),
  poverty_2018 = c(12.9, 11.3, 7.6),
  poverty_2019 = c(11.6, 10.2, 7.3),
  poverty_2020 = c(11.4, 10.8, 7.1)
)
```

```
state_wide
```

```
# A tibble: 3 x 7
  state  unemp_2018 unemp_2019 unemp_2020 poverty_2018 poverty_2019 poverty_2020
  <chr>       <dbl>      <dbl>      <dbl>        <dbl>        <dbl>        <dbl>
1 Maine         3.4        3          5.4         12.9         11.6         11.4
2 Vermo~        2.7        2.4        5           11.3         10.2         10.8
3 New H~        2.5        2.6        6.2          7.6          7.3          7.1
```

This dataset has two variables (unemployment and poverty) each measured across three years. To pivot this to long format, we need to handle the fact that column names encode *two* pieces of information: the variable name and the year. We use `names_sep` to tell `pivot_longer()` how to split the column names:

```
state_long <- state_wide |>
  pivot_longer(
    cols = -state,
    names_to = c("variable", "year"),
    names_sep = "_",
    values_to = "value"
  )

state_long
```

```
# A tibble: 18 x 4
   state         variable year  value
   <chr>         <chr>    <chr> <dbl>
 1 Maine         unemp    2018    3.4
 2 Maine         unemp    2019    3
 3 Maine         unemp    2020    5.4
 4 Maine         poverty  2018   12.9
 5 Maine         poverty  2019   11.6
 6 Maine         poverty  2020   11.4
 7 Vermont       unemp    2018    2.7
 8 Vermont       unemp    2019    2.4
 9 Vermont       unemp    2020    5
10 Vermont       poverty  2018   11.3
11 Vermont       poverty  2019   10.2
12 Vermont       poverty  2020   10.8
13 New Hampshire unemp    2018    2.5
```

```
14 New Hampshire unemp    2019    2.6
15 New Hampshire unemp    2020    6.2
16 New Hampshire poverty  2018    7.6
17 New Hampshire poverty  2019    7.3
18 New Hampshire poverty  2020    7.1
```

Here we used `cols = -state` to pivot all columns *except* `state`. The `names_sep = "_"` tells R to split each column name at the underscore, putting the first part into a column called `variable` and the second part into a column called `year`.

If you'd prefer each variable to have its own column (which is usually more useful for analysis), you can add one more step using `pivot_wider()`:

```
state_tidy <- state_long |>
  pivot_wider(
    names_from = variable,
    values_from = value
  )

state_tidy
```

```
# A tibble: 9 x 4
  state         year  unemp poverty
  <chr>         <chr> <dbl>   <dbl>
1 Maine         2018    3.4    12.9
2 Maine         2019    3      11.6
3 Maine         2020    5.4    11.4
4 Vermont       2018    2.7    11.3
5 Vermont       2019    2.4    10.2
6 Vermont       2020    5      10.8
7 New Hampshire 2018    2.5     7.6
8 New Hampshire 2019    2.6     7.3
9 New Hampshire 2020    6.2     7.1
```

Now each row is a state-year observation with separate columns for unemployment and poverty—exactly the format you'd want for a regression or a plot.

C.5.3 Long to Wide with `pivot_wider()`

Sometimes you need to go in the other direction—converting long data to wide. This is common when creating summary tables or when a dataset arrived in long format but you need it wide for a specific purpose.

`pivot_wider()` is essentially the reverse of `pivot_longer()`:

```
gdp_wide_again <- gdp_long |>
  pivot_wider(
    names_from = year,
    values_from = gdp_per_capita,
    names_prefix = "gdp_"
  )

gdp_wide_again
```

```
# A tibble: 4 x 4
  country        gdp_2018 gdp_2019 gdp_2020
  <chr>             <dbl>    <dbl>    <dbl>
1 United States     62887    65298    63544
2 United Kingdom    42943    42330    40285
3 Germany           47603    46468    45724
4 Japan             39287    40247    39539
```

The key arguments for `pivot_wider()` are:

- `names_from` specifies which column contains the values that will become the new column names.
- `values_from` specifies which column contains the values that will fill those new columns.
- `names_prefix` (optional) adds a prefix to the new column names. Without it, the columns would just be named 2018, 2019, 2020—which are valid R column names but harder to work with because they start with numbers.

C.5.4 Another `pivot_wider()` Example

Suppose you have panel data on birth outcomes and want to create a table comparing states:

```
birth_long <- tibble(
  state = rep(c("Maine", "Vermont", "New Hampshire"), each = 3),
  year = rep(2018:2020, times = 3),
  lbw_rate = c(7.1, 7.3, 7.0,
               6.8, 6.5, 6.9,
               6.2, 6.4, 6.1)
)

birth_long
```

```
# A tibble: 9 x 3
  state          year lbw_rate
  <chr>         <int>    <dbl>
1 Maine          2018      7.1
2 Maine          2019      7.3
3 Maine          2020      7
4 Vermont        2018      6.8
5 Vermont        2019      6.5
6 Vermont        2020      6.9
7 New Hampshire  2018      6.2
8 New Hampshire  2019      6.4
9 New Hampshire  2020      6.1
```

To create a wide table comparing states across years:

```
birth_wide <- birth_long |>
  pivot_wider(
    names_from = year,
    values_from = lbw_rate
  )

birth_wide
```

```
# A tibble: 3 x 4
  state         `2018` `2019` `2020`
  <chr>          <dbl>  <dbl>  <dbl>
1 Maine            7.1    7.3    7
2 Vermont          6.8    6.5    6.9
3 New Hampshire    6.2    6.4    6.1
```

This wide format is convenient for a presentation table where you want to compare across years at a glance.

> **Remembering the Difference**
>
> A helpful way to remember: `pivot_longer()` makes your data *longer* (more rows, fewer columns), and `pivot_wider()` makes your data *wider* (fewer rows, more columns). If your column names contain data values (like years), you probably need `pivot_longer()`. If you want to spread values across columns, you need `pivot_wider()`.

C.6 Exporting Data

After cleaning, reshaping, or creating a new dataset, you'll often want to save it for future use or to share with others. The `tidyverse` provides straightforward functions for exporting data.

C.6.1 Writing CSV Files

The most common export format is CSV, because it can be opened by virtually any software:

```
# Write a data frame to a CSV file
write_csv(gdp_long, "gdp_data_clean.csv")
```

This creates a file called `gdp_data_clean.csv` in your working directory. The file will contain the data in plain text with commas separating the values—no special formatting, no formulas, just clean data.

C.6.2 Writing R Data Files

If you're saving data that will only be used in R, the `.rds` format is preferable because it preserves all of R's data types (factors, dates, etc.) and is more compact than CSV:

```
# Save as an RDS file
write_rds(gdp_long, "gdp_data_clean.rds")
```

To read it back in later:

```
gdp_long <- read_rds("gdp_data_clean.rds")
```

C.6.3 Writing Stata Files

If you need to share data with someone using Stata, the `haven` package can write `.dta` files:

```
library(haven)

write_dta(gdp_long, "gdp_data_clean.dta")
```

C.6.4 Writing Excel Files

To export to Excel format, you can use the `writexl` package:

```
library(writexl)

write_xlsx(gdp_long, "gdp_data_clean.xlsx")
```

Export Function Quick Reference

Table C.2: Export functions by file type

Format	Function	Package
CSV	`write_csv()`	readr (tidyverse)
TSV	`write_tsv()`	readr (tidyverse)
RDS	`write_rds()`	readr (tidyverse)
Stata (.dta)	`write_dta()`	haven
Excel (.xlsx)	`write_xlsx()`	writexl

C.7 Summary and Conclusion

This appendix covered the essential workflow for getting data into and out of R and understanding its structure along the way.

C.7.1 Key Takeaways

- **Documentation first:** Always look for README files and codebooks before analyzing a dataset. Understanding what your variables measure and how they're coded prevents errors downstream.
- **Import with the right function:** Use `read_csv()` for CSV files, `read_excel()` for Excel files, `read_dta()` for Stata files, and `read_rds()` for R data files. The `tidyverse` and its companion packages cover virtually every common file format.
- **Explore before you analyze:** Run `head()`, `glimpse()`, and `summary()` on every new dataset. Check for missing values and verify that variable types and ranges make sense. This quick health check catches problems early.

- **Know wide vs. long:** Wide data has one row per unit with separate columns for time periods; long data has one row per unit-time observation. Most R analysis functions expect long data.
- **Reshape with pivot functions:** Use `pivot_longer()` to go from wide to long and `pivot_wider()` to go from long to wide. The key arguments are `cols`, `names_to`/`names_from`, and `values_to`/`values_from`.
- **Export in the right format:** Use `write_csv()` for universal compatibility, `write_rds()` for R-only workflows, and format-specific functions like `write_dta()` when sharing with colleagues who use other software.

C.8 Check Your Understanding

Note: This section contains interactive content only available in the HTML version.

D R Functions and Packages Reference

This appendix provides a quick reference for all the R functions and packages used throughout this book. Use this as a handy guide when you need to remember the syntax or arguments for a particular function.

D.1 Packages Used in This Book

D.1.1 tidyverse

The `tidyverse` is a collection of R packages designed for data science. When you load `tidyverse`, it automatically loads several packages including `ggplot2`, `dplyr`, `tidyr`, `readr`, and others.

What it's used for:

- Data manipulation and transformation (`dplyr`)
- Data visualization (`ggplot2`)
- Reshaping data (`tidyr`)
- Reading data files (`readr`)

Loading:

```
library(tidyverse)
```

D.1.2 modelsummary

The `modelsummary` package creates publication-quality tables for regression results and descriptive statistics.

What it's used for:

- Creating formatted regression tables comparing multiple models
- Generating descriptive statistics tables

- Exporting tables to Word, LaTeX, or HTML

Loading:

```
library(modelsummary)
```

D.1.3 palmerpenguins

A dataset package containing measurements of penguins from Palmer Station, Antarctica. Great for learning data visualization and exploration.

What it's used for:

- Practice dataset for learning R
- Data visualization examples
- Regression examples

Loading:

```
library(palmerpenguins)
data(penguins)
```

D.1.4 wooldridge

Contains all datasets from Wooldridge's *Introductory Econometrics* textbook.

What it's used for:

- Real-world econometric datasets
- Regression examples (wages, crime, etc.)

Loading:

```
library(wooldridge)
data(wage1)  # Example: load the wage1 dataset
```

D.1.5 fixest

A fast and powerful package for fixed effects and panel data regression.

What it's used for:

- Panel data regression with fixed effects
- Clustered standard errors
- Very large datasets (faster than `lm()`)

Loading:

```
library(fixest)
```

D.1.6 lmtest

Provides diagnostic tests for linear regression models.

What it's used for:

- Breusch-Pagan test for heteroskedasticity
- Other specification tests

Loading:

```
library(lmtest)
```

D.1.7 sandwich

Provides robust covariance matrix estimators (heteroskedasticity-consistent standard errors).

What it's used for:

- Robust standard errors
- Heteroskedasticity-consistent (HC) standard errors

Loading:

```r
library(sandwich)
```

D.1.8 patchwork

Easily combine multiple ggplot2 plots into a single figure.

What it's used for:

- Arranging multiple plots side by side
- Creating multi-panel figures

Loading:

```r
library(patchwork)
```

D.2 Data Inspection Functions

D.2.1 `head()`

Display the first few rows of a data frame.

Arguments:

- `x`: A data frame or vector
- `n`: Number of rows to display (default: 6)

Example:

```r
head(mtcars, n = 3)
```

```
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

D.2.2 `glimpse()`

Get a compact overview of a data frame's structure and column types.

Arguments:

- `x`: A data frame

Example:

```
glimpse(mtcars)
```

```
Rows: 32
Columns: 11
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,~
$ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,~
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 16~
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180~
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,~
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.~
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18~
$ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,~
$ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,~
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3,~
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2,~
```

D.2.3 `summary()`

Generate summary statistics for each variable.

Arguments:

- `object`: A data frame, vector, or model object

Example:

```
summary(mtcars$mpg)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
```

D.2.4 `nrow()` and `ncol()`

Return the number of rows or columns in a data frame.

Arguments:

- `x`: A data frame or matrix

Example:

```
nrow(mtcars)
```

```
[1] 32
```

```
ncol(mtcars)
```

```
[1] 11
```

D.2.5 `colnames()`

Return or set the column names of a data frame.

Arguments:

- `x`: A data frame or matrix

Example:

```
colnames(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

D.2.6 `class()` and `typeof()`

Return the class or underlying type of an object.

Arguments:

- `x`: Any R object

Example:

```
class(mtcars$mpg)
```

```
[1] "numeric"
```

```
typeof(mtcars$mpg)
```

```
[1] "double"
```

D.3 Descriptive Statistics Functions

D.3.1 `mean()`

Calculate the arithmetic mean.

Arguments:

- `x`: A numeric vector
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
x <- c(10, 20, 30, NA, 50)
mean(x, na.rm = TRUE)
```

```
[1] 27.5
```

D.3.2 `sd()`

Calculate the standard deviation.

Arguments:

- `x`: A numeric vector
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
sd(mtcars$mpg)
```

```
[1] 6.026948
```

D.3.3 `median()`

Calculate the median (middle value).

Arguments:

- `x`: A numeric vector
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
median(mtcars$mpg)
```

```
[1] 19.2
```

D.3.4 `min()` and `max()`

Find the minimum or maximum value.

Arguments:

- `...`: Numeric vectors
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
min(mtcars$mpg)
```

```
[1] 10.4
```

```
max(mtcars$mpg)
```

```
[1] 33.9
```

D.3.5 `sum()`

Calculate the sum of all values.

Arguments:

- `...`: Numeric vectors
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
sum(mtcars$mpg)
```

```
[1] 642.9
```

D.3.6 `var()`

Calculate the variance.

Arguments:

- `x`: A numeric vector
- `na.rm`: Remove missing values? (default: `FALSE`)

Example:

```
var(mtcars$mpg)
```

```
[1] 36.3241
```

D.3.7 `cor()`

Calculate the correlation between two variables.

Arguments:

- `x`, `y`: Numeric vectors
- `use`: How to handle missing values (e.g., `"complete.obs"`)

Example:

```
cor(mtcars$mpg, mtcars$hp)
```

```
[1] -0.7761684
```

D.4 Data Manipulation Functions (dplyr)

D.4.1 `select()`

Choose which columns to keep.

Arguments:

- `.data`: A data frame
- `...`: Column names or selection helpers

Example:

```
mtcars |>
  select(mpg, cyl, hp) |>
  head(3)
```

```
                mpg cyl  hp
Mazda RX4      21.0   6 110
Mazda RX4 Wag  21.0   6 110
Datsun 710     22.8   4  93
```

D.4.2 `filter()`

Keep rows that meet a condition.

Arguments:

- `.data`: A data frame
- `...`: Logical conditions

Example:

```
mtcars |>
  filter(mpg > 25) |>
  head(3)
```

```
                mpg cyl disp hp drat    wt  qsec vs am gear carb
Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
```

D.4.3 `mutate()`

Create new variables or modify existing ones.

Arguments:

- `.data`: A data frame
- `...`: Name-value pairs of expressions

Example:

```
mtcars |>
  mutate(kpl = mpg * 0.425) |>  # Convert to km per liter
  select(mpg, kpl) |>
  head(3)
```

```
                mpg   kpl
Mazda RX4      21.0 8.925
Mazda RX4 Wag  21.0 8.925
Datsun 710     22.8 9.690
```

D.4.4 `group_by()`

Group data by one or more variables (usually followed by `summarize()`).

Arguments:

- `.data`: A data frame
- `...`: Variables to group by

Example:

```
mtcars |>
  group_by(cyl) |>
  summarize(avg_mpg = mean(mpg))
```

```
# A tibble: 3 x 2
    cyl avg_mpg
  <dbl>   <dbl>
1     4    26.7
2     6    19.7
3     8    15.1
```

D.4.5 `summarize()` / `summarise()`

Calculate summary statistics for each group.

Arguments:

- `.data`: A data frame (usually grouped)
- `...`: Name-value pairs of summary functions

Example:

```
mtcars |>
  group_by(cyl) |>
  summarize(
    avg_mpg = mean(mpg),
    sd_mpg = sd(mpg),
    n = n()
  )
```

```
# A tibble: 3 x 4
    cyl avg_mpg sd_mpg     n
  <dbl>   <dbl>  <dbl> <int>
1     4    26.7   4.51    11
2     6    19.7   1.45     7
3     8    15.1   2.56    14
```

D.4.6 `count()`

Count observations by group.

Arguments:

- `x`: A data frame
- `...`: Variables to count by
- `sort`: Sort by count? (default: `FALSE`)

Example:

```
mtcars |>
  count(cyl)
```

```
  cyl  n
1   4 11
2   6  7
3   8 14
```

D.4.7 `arrange()`

Sort rows by one or more variables.

Arguments:

- `.data`: A data frame
- `...`: Variables to sort by (use `desc()` for descending)

Example:

```
mtcars |>
  arrange(desc(mpg)) |>
  head(3)
```

```
                mpg cyl disp hp drat    wt  qsec vs am gear carb
Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7 52 4.93 1.615 18.52  1  1    4    2
```

D.4.8 `case_when()`

Vectorized if-else for creating categorical variables.

Arguments:

- `...`: A sequence of two-sided formulas: `condition ~ value`

Example:

```
mtcars |>
  mutate(
    mpg_category = case_when(
      mpg < 15 ~ "Low",
      mpg < 25 ~ "Medium",
      TRUE ~ "High"
    )
  ) |>
  count(mpg_category)
```

```
  mpg_category  n
1         High  6
2          Low  5
3       Medium 21
```

D.4.9 `ifelse()`

Simple conditional: if condition is TRUE, return one value; otherwise, another.

Arguments:

- `test`: A logical condition
- `yes`: Value if TRUE
- `no`: Value if FALSE

Example:

```
mtcars |>
  mutate(efficient = ifelse(mpg > 20, 1, 0)) |>
  count(efficient)
```

```
  efficient  n
1         0 18
2         1 14
```

D.4.10 `n()`

Count the number of observations in the current group (used inside `summarize()`).

Arguments: None

Example:

```
mtcars |>
  group_by(cyl) |>
  summarize(count = n())
```

```
# A tibble: 3 x 2
    cyl count
  <dbl> <int>
1     4    11
2     6     7
3     8    14
```

D.5 Regression Functions

D.5.1 `lm()`

Fit a linear model using Ordinary Least Squares (OLS).

Arguments:

- `formula`: A formula like `y ~ x1 + x2`
- `data`: A data frame

Example:

```r
reg <- lm(mpg ~ hp + wt, data = mtcars)
summary(reg)
```

```
Call:
lm(formula = mpg ~ hp + wt, data = mtcars)

Residuals:
   Min     1Q Median     3Q    Max
-3.941 -1.600 -0.182  1.050  5.854

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 **
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.593 on 29 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8148
F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

D.5.2 `summary()` (for regression)

Display detailed regression results including coefficients, standard errors, t-statistics, and p-values.

Arguments:

- `object`: A fitted model object

Example:

```r
reg <- lm(mpg ~ hp, data = mtcars)
summary(reg)
```

D.5.3 `coef()`

Extract the estimated coefficients from a model.

Arguments:

- `object`: A fitted model object

Example:

```
reg <- lm(mpg ~ hp, data = mtcars)
coef(reg)
```

```
(Intercept)          hp
30.09886054 -0.06822828
```

D.5.4 `confint()`

Calculate confidence intervals for model coefficients.

Arguments:

- `object`: A fitted model object
- `level`: Confidence level (default: 0.95)

Example:

```
reg <- lm(mpg ~ hp, data = mtcars)
confint(reg, level = 0.95)
```

```
                2.5 %     97.5 %
(Intercept) 26.76194879 33.4357723
hp          -0.08889465 -0.0475619
```

D.5.5 `predict()`

Generate predicted values from a fitted model.

Arguments:

- `object`: A fitted model object
- `newdata`: Optional data frame with new predictor values

Example:

```r
reg <- lm(mpg ~ hp, data = mtcars)
# Predict for existing data
head(predict(reg))
```

```
        Mazda RX4     Mazda RX4 Wag        Datsun 710    Hornet 4 Drive
         22.59375          22.59375          23.75363          22.59375
Hornet Sportabout           Valiant
         18.15891          22.93489
```

```r
# Predict for new data
predict(reg, newdata = data.frame(hp = c(100, 150, 200)))
```

```
       1        2        3
23.27603 19.86462 16.45320
```

D.5.6 `residuals()`

Extract residuals (prediction errors) from a fitted model.

Arguments:

- `object`: A fitted model object

Example:

```r
reg <- lm(mpg ~ hp, data = mtcars)
head(residuals(reg))
```

```
        Mazda RX4     Mazda RX4 Wag        Datsun 710    Hornet 4 Drive
       -1.5937500        -1.5937500        -0.9536307        -1.1937500
Hornet Sportabout           Valiant
        0.5410881        -4.8348913
```

D.5.7 `feols()` (fixest package)

Fit linear models with fixed effects and robust standard errors.

Arguments:

- `fml`: A formula (use `|` for fixed effects)
- `data`: A data frame
- `vcov`: Type of standard errors (e.g., `"HC1"` for robust)

Example:

```
library(fixest)
reg <- feols(mpg ~ hp + wt, data = mtcars, vcov = "HC1")
summary(reg)
```

D.5.8 `modelsummary()` (modelsummary package)

Create publication-quality regression tables.

Arguments:

- `models`: A model or list of models
- `stars`: Show significance stars?
- `gof_map`: Which goodness-of-fit statistics to show

Example:

```
library(modelsummary)
reg1 <- lm(mpg ~ hp, data = mtcars)
reg2 <- lm(mpg ~ hp + wt, data = mtcars)
modelsummary(list(reg1, reg2), stars = TRUE)
```

D.6 Visualization Functions (ggplot2)

D.6.1 `ggplot()`

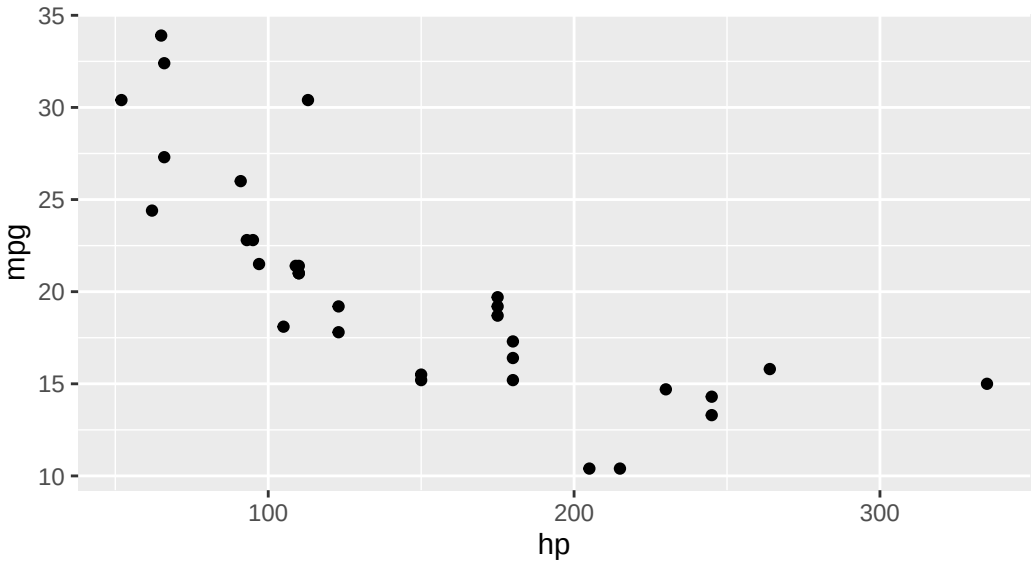
Initialize a ggplot object.

Arguments:

- `data`: A data frame
- `mapping`: Aesthetic mappings created by `aes()`

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point()
```



D.6.2 `aes()`

Define aesthetic mappings (which variables map to x, y, color, etc.).

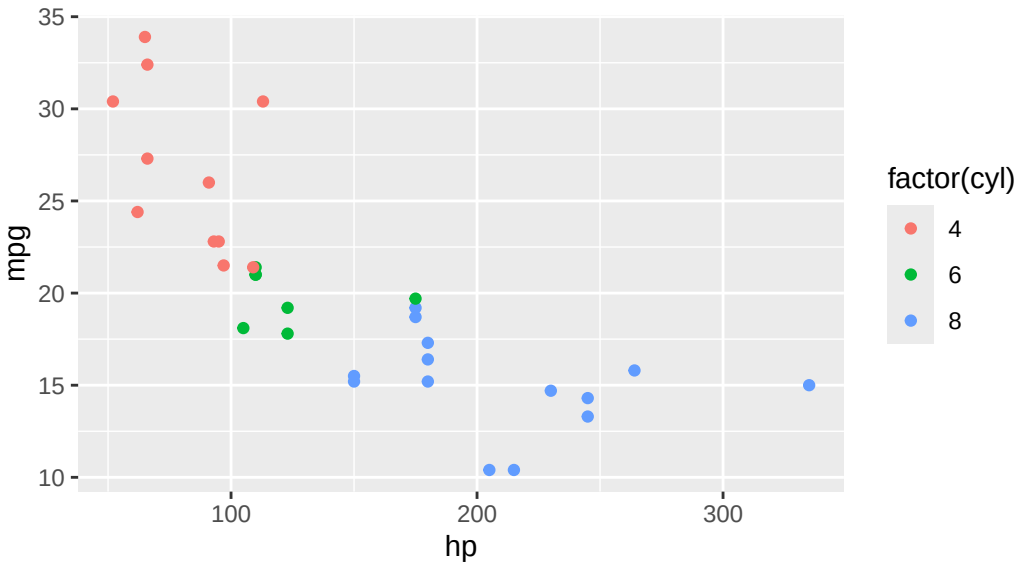
Arguments:

- `x`, `y`: Variables for axes
- `color`, `fill`: Variables for color

- `size`, `shape`: Variables for size and shape

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(cyl))) +
  geom_point()
```



D.6.3 `geom_point()`

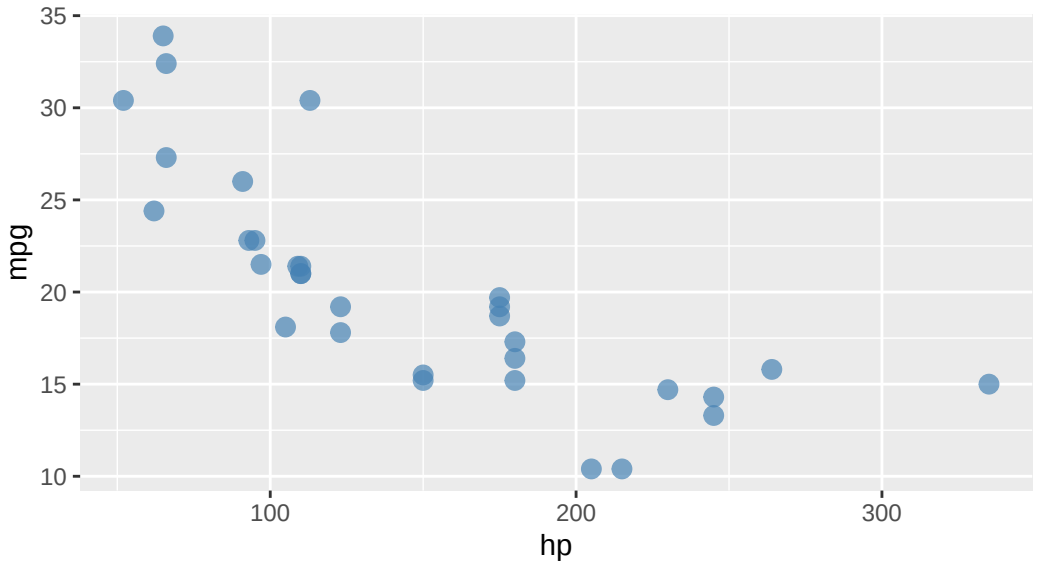
Add points to a plot (scatterplot).

Arguments:

- `size`: Point size
- `color`: Point color
- `alpha`: Transparency (0 to 1)

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point(size = 3, color = "steelblue", alpha = 0.7)
```



D.6.4 `geom_line()`

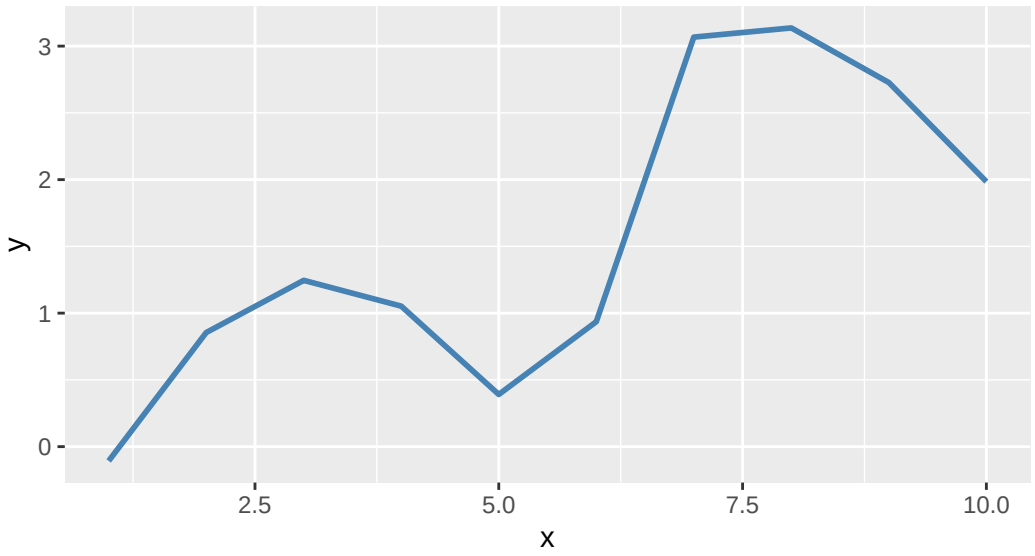
Add lines connecting points.

Arguments:

- `linewidth`: Line thickness
- `color`: Line color
- `linetype`: Line type (e.g., "dashed")

Example:

```
# Line plot (useful for time series)
df <- data.frame(x = 1:10, y = cumsum(rnorm(10)))
ggplot(df, aes(x = x, y = y)) +
  geom_line(color = "steelblue", linewidth = 1)
```



D.6.5 `geom_histogram()`

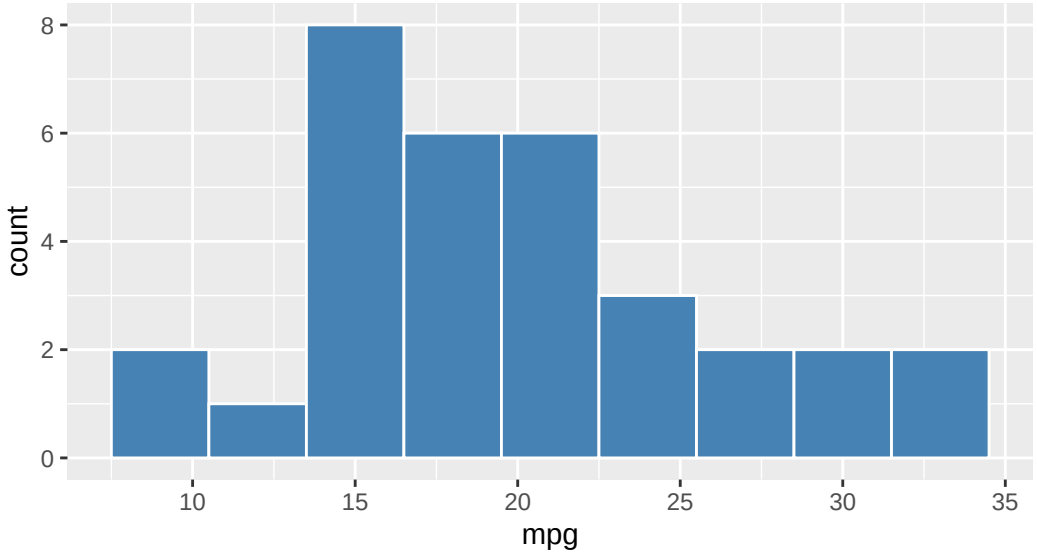
Create a histogram.

Arguments:

- `binwidth`: Width of each bin
- `bins`: Number of bins
- `fill`: Bar fill color
- `color`: Bar outline color

Example:

```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram(binwidth = 3, fill = "steelblue", color = "white")
```



D.6.6 `geom_boxplot()`

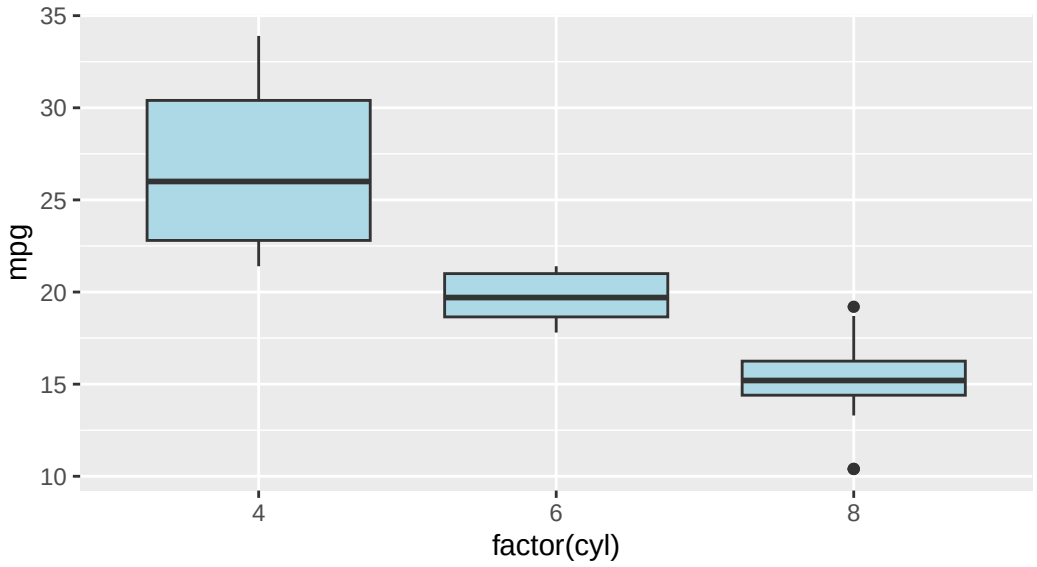
Create a boxplot.

Arguments:

- `fill`: Box fill color
- `outlier.color`: Color of outlier points

Example:

```
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot(fill = "lightblue")
```



D.6.7 `geom_bar()` and `geom_col()`

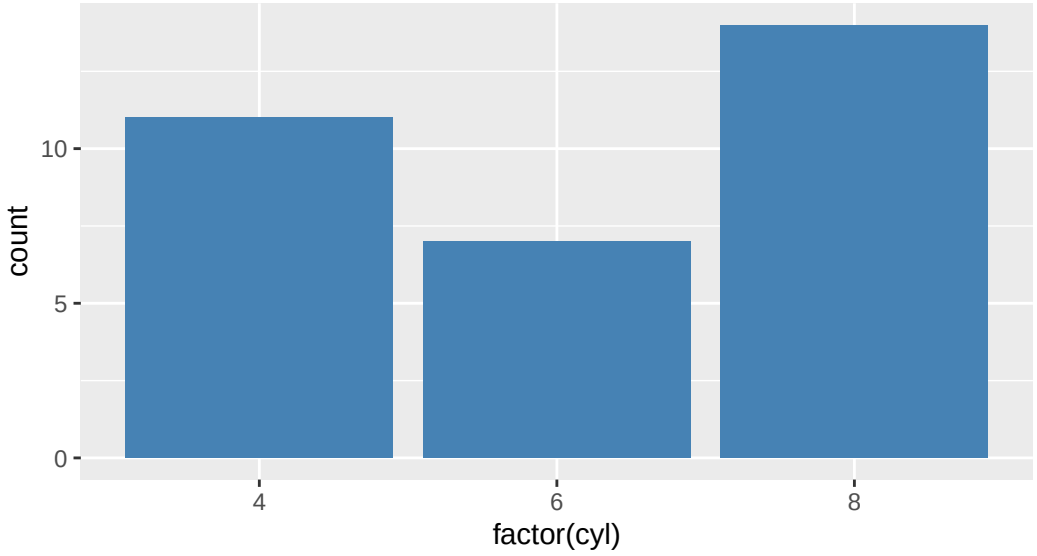
Create bar charts. `geom_bar()` counts observations; `geom_col()` uses values directly.

Arguments:

- `fill`: Bar fill color
- `stat`: For `geom_bar()`, use `"identity"` to plot values directly

Example:

```
# geom_bar counts automatically
ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar(fill = "steelblue")
```



D.6.8 `geom_smooth()`

Add a smoothed conditional mean (often a regression line).

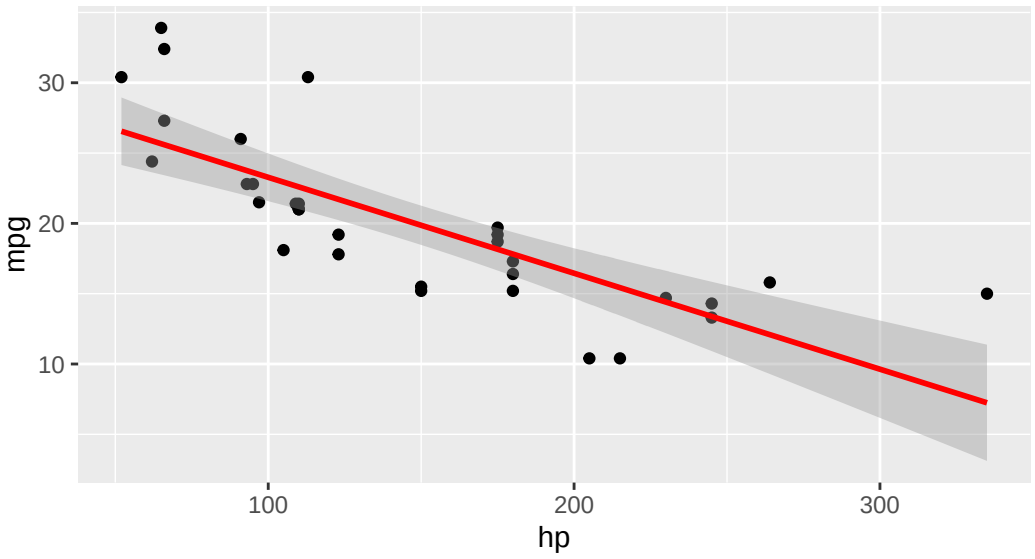
Arguments:

- `method`: Smoothing method (e.g., `"lm"` for linear)
- `se`: Show confidence interval? (default: `TRUE`)
- `color`: Line color

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE, color = "red")
```

```
`geom_smooth()` using formula = 'y ~ x'
```



D.6.9 `geom_hline()` and `geom_vline()`

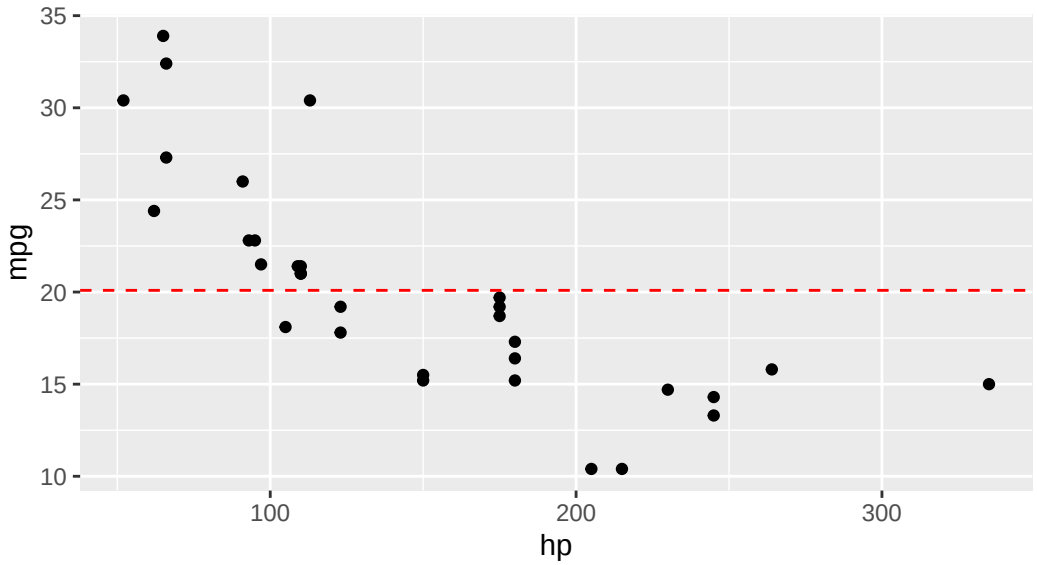
Add horizontal or vertical reference lines.

Arguments:

- `yintercept` / `xintercept`: Where to draw the line
- `linetype`: Line type (e.g., `"dashed"`)
- `color`: Line color

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_hline(yintercept = mean(mtcars$mpg), linetype = "dashed", color = "red")
```



D.6.10 `labs()`

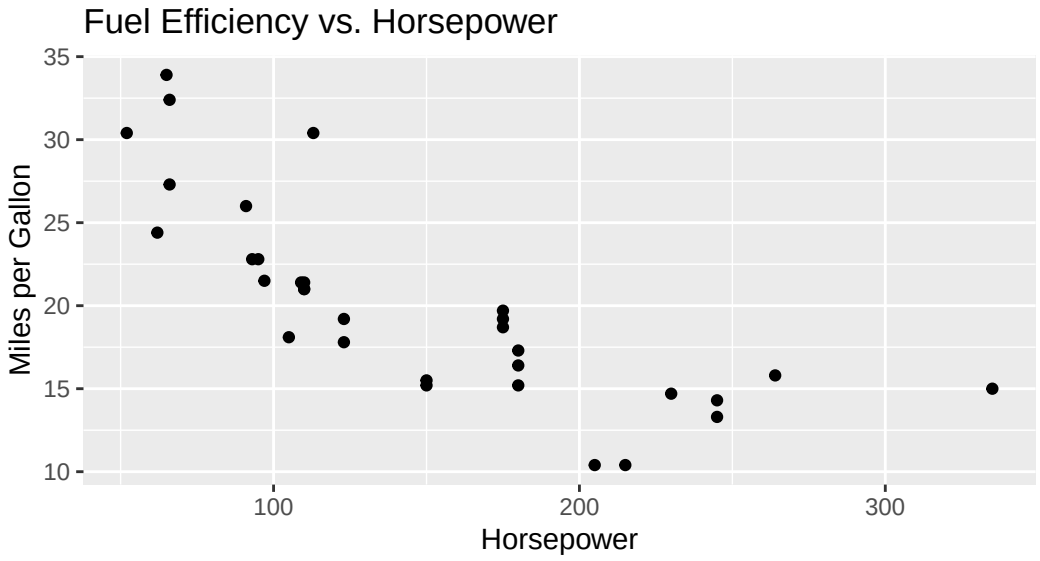
Add labels to the plot (title, axis labels, etc.).

Arguments:

- `title`: Plot title
- `x`, `y`: Axis labels
- `color`, `fill`: Legend titles

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  labs(
    title = "Fuel Efficiency vs. Horsepower",
    x = "Horsepower",
    y = "Miles per Gallon"
  )
```



D.6.11 `facet_wrap()`

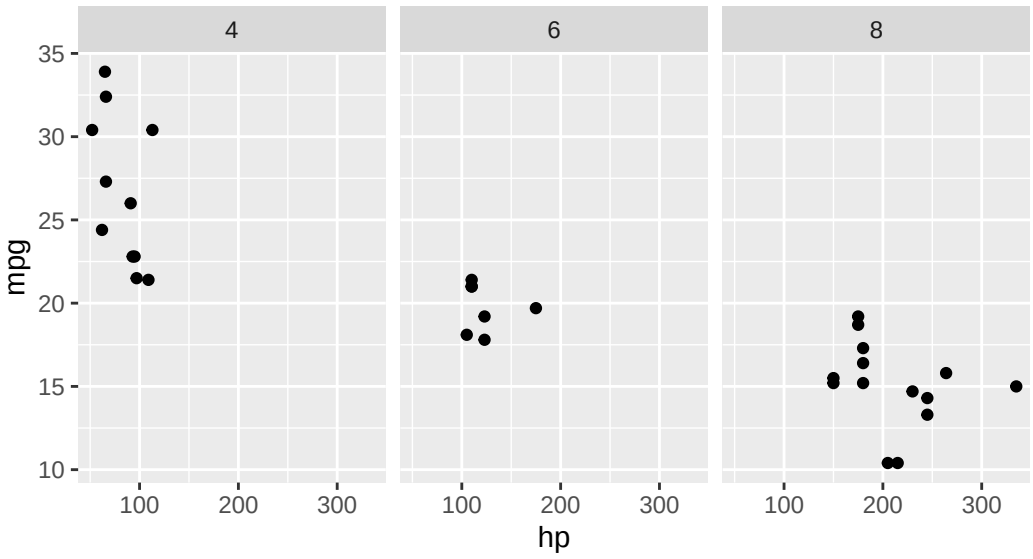
Create small multiples (separate panels for each level of a variable).

Arguments:

- `facets`: A formula like `~ variable`
- `nrow`, `ncol`: Number of rows or columns
- `scales`: Should scales be fixed or free?

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  facet_wrap(~ cyl, nrow = 1)
```



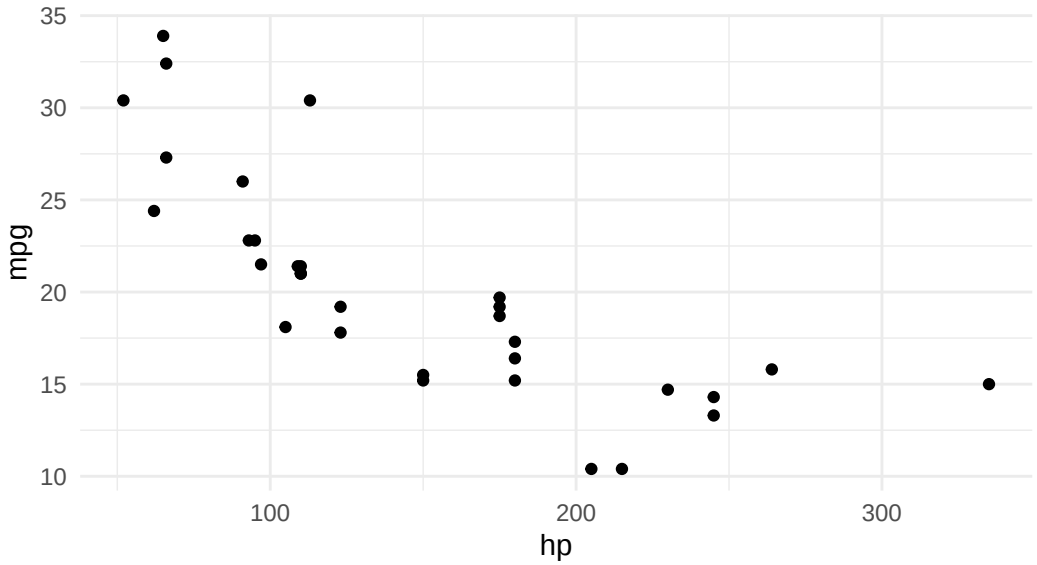
D.6.12 `theme_minimal()`

Apply a clean, minimal theme to the plot.

Arguments: None (or see `theme()` for customization)

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme_minimal()
```



D.7 Utility Functions

D.7.1 `c()`

Combine values into a vector.

Arguments:

- `...`: Values to combine

Example:

```
x <- c(1, 2, 3, 4, 5)
x
```

```
[1] 1 2 3 4 5
```

D.7.2 `seq()`

Generate a sequence of numbers.

Arguments:

- `from`: Starting value
- `to`: Ending value
- `by`: Increment
- `length.out`: Desired length of sequence

Example:

```
seq(from = 0, to = 10, by = 2)
```

```
[1]  0  2  4  6  8 10
```

```
seq(from = 0, to = 1, length.out = 5)
```

```
[1] 0.00 0.25 0.50 0.75 1.00
```

D.7.3 `rep()`

Repeat values.

Arguments:

- `x`: Value(s) to repeat
- `times`: Number of times to repeat

Example:

```
rep(c("A", "B"), times = 3)
```

```
[1] "A" "B" "A" "B" "A" "B"
```

D.7.4 `sample()`

Take a random sample.

Arguments:

- `x`: Vector to sample from
- `size`: Number of items to sample
- `replace`: Sample with replacement?

Example:

```
set.seed(123)
sample(1:10, size = 5, replace = FALSE)
```

```
[1]  3 10  2  8  6
```

D.7.5 `set.seed()`

Set the random number seed for reproducibility.

Arguments:

- `seed`: An integer

Example:

```
set.seed(42)
rnorm(3)  # Will always produce the same values
```

```
[1]  1.3709584 -0.5646982  0.3631284
```

D.7.6 `rnorm()`

Generate random numbers from a normal distribution.

Arguments:

- `n`: Number of values to generate
- `mean`: Mean of the distribution (default: 0)
- `sd`: Standard deviation (default: 1)

Example:

```
set.seed(123)
rnorm(5, mean = 100, sd = 15)
```

```
[1]  91.59287  96.54734 123.38062 101.05763 101.93932
```

D.7.7 `factor()`

Create a factor (categorical variable).

Arguments:

- `x`: A vector
- `levels`: The allowed levels (in order)
- `labels`: Labels for the levels

Example:

```
education <- c("HS", "College", "HS", "Graduate")
factor(education, levels = c("HS", "College", "Graduate"))
```

```
[1] HS       College  HS       Graduate
Levels: HS College Graduate
```

D.7.8 `as.factor()`, `as.numeric()`, `as.character()`

Convert objects to a different type.

Arguments:

- `x`: Object to convert

Example:

```
x <- c("1", "2", "3")
as.numeric(x)
```

```
[1] 1 2 3
```

D.7.9 `is.na()`

Check for missing values.

Arguments:

- `x`: A vector or data frame

Example:

```
x <- c(1, 2, NA, 4)
is.na(x)
```

```
[1] FALSE FALSE  TRUE FALSE
```

```
sum(is.na(x))  # Count missing values
```

```
[1] 1
```

D.7.10 `library()`

Load an installed package.

Arguments:

- `package`: Name of the package (unquoted)

Example:

```
library(tidyverse)
```

D.7.11 `install.packages()`

Install a package from CRAN (run once, not in scripts).

Arguments:

- `pkgs`: Package name (quoted)

Example:

```
install.packages("tidyverse")
```

D.8 The Pipe Operator: |>

The pipe operator takes the output from the left side and passes it as the first argument to the function on the right. This allows you to chain operations together in a readable way.

Example without pipe:

```
# Nested functions are hard to read
summary(select(filter(mtcars, mpg > 20), mpg, hp))
```

Example with pipe:

```r
# Piped version is much clearer
mtcars |>
  filter(mpg > 20) |>
  select(mpg, hp) |>
  summary()
```

```
      mpg              hp
 Min.   :21.00   Min.   : 52.0
 1st Qu.:21.43   1st Qu.: 66.0
 Median :23.60   Median : 94.0
 Mean   :25.48   Mean   : 88.5
 3rd Qu.:29.62   3rd Qu.:109.8
 Max.   :33.90   Max.   :113.0
```

> Keyboard Shortcut
>
> In RStudio, type **Cmd/Ctrl + Shift + M** to insert the pipe operator.

D.9 Statistical Distribution Functions

R has a consistent naming convention for distribution functions. Each distribution has four functions, prefixed by a letter:

- `d` = **density** (the height of the PDF at a given value)
- `p` = **probability** (the CDF—cumulative probability up to a given value)
- `q` = **quantile** (the inverse of the CDF—find the value for a given probability)
- `r` = **random** (generate random draws from the distribution)

For example, for the normal distribution: `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()`.

D.9.1 `dnorm()` and `dt()`

Compute the probability density function (PDF) for the normal or t-distribution. Useful for plotting distribution curves.

Arguments:

- `x`: Value(s) at which to evaluate the density

- `mean`, `sd`: Parameters of the normal distribution (for `dnorm()`)
- `df`: Degrees of freedom (for `dt()`)

Example:

```
# Height of the standard normal PDF at x = 0
dnorm(0, mean = 0, sd = 1)
```

```
[1] 0.3989423
```

```
# Height of the t-distribution PDF at x = 2 with 30 df
dt(2, df = 30)
```

```
[1] 0.05685228
```

D.9.2 `pt()`

Compute the cumulative distribution function (CDF) for the t-distribution. This gives you the probability that a t-distributed random variable is less than or equal to a given value. Essential for computing p-values.

Arguments:

- `q`: The t-value(s) to evaluate
- `df`: Degrees of freedom
- `lower.tail`: If `TRUE` (default), returns $P(T \leq q)$; if `FALSE`, returns $P(T > q)$

Example:

```
# P-value for a two-sided test with t = 2.3 and 100 df
2 * pt(abs(2.3), df = 100, lower.tail = FALSE)
```

```
[1] 0.0235262
```

D.9.3 `qt()`

Compute the quantile function (inverse CDF) for the t-distribution. Given a probability, it returns the corresponding t-value. Used to find critical values for hypothesis tests.

Arguments:

- `p`: Probability (between 0 and 1)
- `df`: Degrees of freedom
- `lower.tail`: If `TRUE` (default), finds the value where $P(T \leq q) = p$; if `FALSE`, finds the value where $P(T > q) = p$

Example:

```
# Critical value for a two-sided 5% test with 100 df
# We want the value that leaves 2.5% in the upper tail
qt(0.025, df = 100, lower.tail = FALSE)
```

```
[1] 1.983972
```

D.9.4 `qf()`

Compute the quantile function for the F-distribution. Used to find critical values for F-tests.

Arguments:

- `p`: Probability
- `df1`: Numerator degrees of freedom (number of restrictions)
- `df2`: Denominator degrees of freedom (from the unrestricted model)
- `lower.tail`: If `FALSE`, returns the value where $P(F > q) = p$

Example:

```
# Critical value for an F-test with 2 and 494 degrees of freedom at 5%
qf(0.05, df1 = 2, df2 = 494, lower.tail = FALSE)
```

```
[1] 3.013973
```

D.9.5 `runif()`

Generate random draws from a uniform distribution.

Arguments:

- `n`: Number of values to generate
- `min`: Minimum value (default: 0)
- `max`: Maximum value (default: 1)

Example:

```
set.seed(123)
runif(5, min = 1, max = 10)
```

```
[1] 3.588198 8.094746 4.680792 8.947157 9.464206
```

D.9.6 `rexp()`

Generate random draws from an exponential distribution.

Arguments:

- `n`: Number of values to generate
- `rate`: Rate parameter λ (default: 1). The mean of the distribution is $1/\lambda$.

Example:

```
set.seed(123)
rexp(5, rate = 0.5)  # Mean = 1/0.5 = 2
```

```
[1] 1.68691452 1.15322054 2.65810974 0.06315472 0.11242195
```

D.9.7 `rbinom()`

Generate random draws from a binomial distribution. Useful for simulating binary outcomes (e.g., treatment assignment).

Arguments:

- `n`: Number of values to generate
- `size`: Number of trials
- `prob`: Probability of success on each trial

Example:

```
set.seed(123)
# Simulate 10 coin flips (1 = heads, 0 = tails)
rbinom(10, size = 1, prob = 0.5)
```

```
 [1] 0 1 0 1 1 0 1 1 1 0
```

D.9.8 `rt()`

Generate random draws from a t-distribution.

Arguments:

- `n`: Number of values to generate
- `df`: Degrees of freedom

Example:

```
set.seed(123)
rt(5, df = 30)
```

```
[1] -0.5878234 -1.4779045 -0.1125616 -1.4142351  1.6124113
```

D.10 Model Diagnostic Functions

D.10.1 `anova()`

Compare nested models using an F-test. Pass the restricted (smaller) model first, then the unrestricted (larger) model. Tests whether the additional variables in the unrestricted model are jointly significant.

Arguments:

- `object`: One or more fitted model objects

Example:

```
reg_small <- lm(mpg ~ hp, data = mtcars)
reg_large <- lm(mpg ~ hp + wt + cyl, data = mtcars)
anova(reg_small, reg_large)
```

```
Analysis of Variance Table

Model 1: mpg ~ hp
Model 2: mpg ~ hp + wt + cyl
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     30 447.67
2     28 176.62  2    271.05 21.485 2.214e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

D.10.2 `nobs()`

Return the number of observations used to fit a model.

Arguments:

- `object`: A fitted model object

Example:

```
reg <- lm(mpg ~ hp, data = mtcars)
nobs(reg)
```

```
[1] 32
```

D.10.3 `df.residual()`

Return the residual degrees of freedom $(n - k - 1)$ from a fitted model.

Arguments:

- `object`: A fitted model object

Example:

```
reg <- lm(mpg ~ hp + wt, data = mtcars)
df.residual(reg)  # 32 - 2 - 1 = 29
```

```
[1] 29
```

D.10.4 `resid()`

Extract residuals from a fitted model. Equivalent to `residuals()`.

Arguments:

- `object`: A fitted model object

Example:

```
reg <- lm(mpg ~ hp, data = mtcars)
head(resid(reg))
```

```
        Mazda RX4     Mazda RX4 Wag        Datsun 710    Hornet 4 Drive
       -1.5937500        -1.5937500        -0.9536307        -1.1937500
Hornet Sportabout           Valiant
        0.5410881        -4.8348913
```

D.11 Additional Data Manipulation Functions

D.11.1 `tibble()`

Create a data frame (modern version). Works like `data.frame()` but with better defaults: it doesn't convert strings to factors and prints more cleanly.

Arguments:

- `...`: Name-value pairs of columns

Example:

```
tibble(
  name = c("Alice", "Bob", "Carol"),
  age = c(25, 30, 35),
  income = c(50000, 60000, 70000)
)
```

```
# A tibble: 3 x 3
  name    age income
  <chr> <dbl>  <dbl>
1 Alice    25  50000
2 Bob      30  60000
3 Carol    35  70000
```

D.11.2 `bind_rows()`

Stack data frames on top of each other (by rows).

Arguments:

- `...`: Data frames to bind together

Example:

```
df1 <- tibble(x = 1:3, y = c("a", "b", "c"))
df2 <- tibble(x = 4:6, y = c("d", "e", "f"))
bind_rows(df1, df2)
```

```
# A tibble: 6 x 2
      x y
  <int> <chr>
1     1 a
2     2 b
3     3 c
4     4 d
5     5 e
6     6 f
```

D.11.3 `pivot_longer()`

Reshape data from wide format to long format. Useful for making data "tidy" for ggplot2.

Arguments:

- `data`: A data frame
- `cols`: Columns to pivot into longer format
- `names_to`: Name of the new column that will contain the old column names
- `values_to`: Name of the new column that will contain the values

Example:

```r
wide_data <- tibble(
  id = 1:3,
  score_2020 = c(80, 90, 85),
  score_2021 = c(85, 92, 88)
)

wide_data |>
  pivot_longer(
    cols = starts_with("score"),
    names_to = "year",
    values_to = "score"
  )
```

```
# A tibble: 6 x 3
     id year       score
  <int> <chr>      <dbl>
1     1 score_2020    80
```

```
2     1 score_2021    85
3     2 score_2020    90
4     2 score_2021    92
5     3 score_2020    85
6     3 score_2021    88
```

D.11.4 `slice_sample()`

Randomly sample rows from a data frame.

Arguments:

- `.data`: A data frame
- `n`: Number of rows to sample
- `replace`: Sample with replacement? (default: `FALSE`)

Example:

```r
set.seed(123)
mtcars |>
  slice_sample(n = 5)
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Maserati Bora    15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Honda Civic      30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Merc 450SLC      15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
```

D.11.5 `map_dfr()` (purrr)

Apply a function to each element of a list or vector and combine the results into a single data frame by binding rows. Useful for running simulations.

Arguments:

- `.x`: A list or vector to iterate over
- `.f`: A function to apply to each element

Example:

```
# Run 5 simulations and collect results
map_dfr(1:5, function(i) {
  x <- rnorm(50)
  tibble(sim = i, mean_x = mean(x))
})
```

```
# A tibble: 5 x 2
    sim    mean_x
  <int>     <dbl>
1     1 -0.0600
2     2  0.0103
3     3  0.0978
4     4 -0.000785
5     5 -0.0664
```

D.12 Additional ggplot2 Functions

D.12.1 `annotate()`

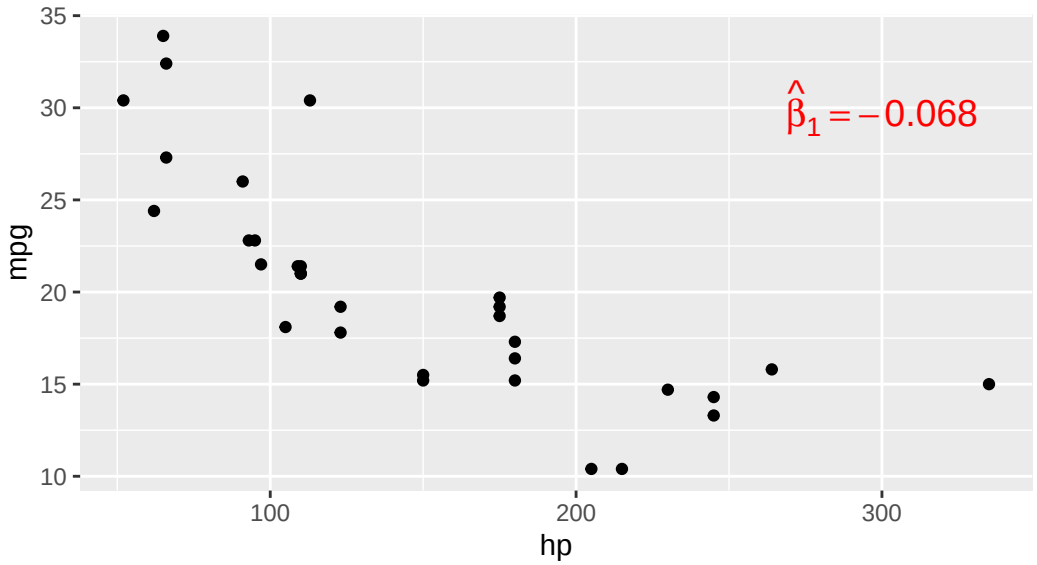
Add text, labels, or shapes to a plot at specific coordinates. Unlike `geom_text()`, `annotate()` is for adding single annotations rather than mapping data to text.

Arguments:

- `geom`: Type of annotation (e.g., `"text"`, `"rect"`, `"segment"`)
- `x`, `y`: Position of the annotation
- `label`: Text to display (for `"text"` geom)
- `parse`: If `TRUE`, interpret the label as a plotmath expression (default: `FALSE`)
- `color`, `size`: Styling options

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  annotate("text", x = 300, y = 30,
           label = "hat(beta)[1] == -0.068",
           parse = TRUE, color = "red", size = 5)
```



D.12.2 `stat_function()`

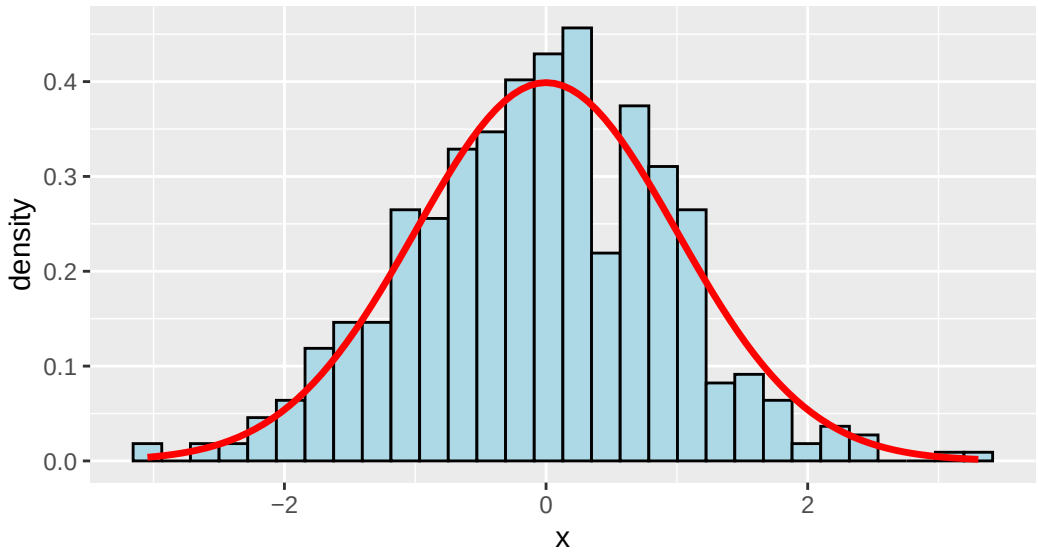
Overlay a mathematical function on a ggplot. Useful for plotting theoretical distributions on top of histograms.

Arguments:

- `fun`: The function to plot (e.g., `dnorm`, `dt`)
- `args`: A list of additional arguments to pass to the function
- `color`, `linewidth`: Styling options

Example:

```
ggplot(data.frame(x = rnorm(500)), aes(x = x)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 30, fill = "lightblue", color = "black") +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 1),
                color = "red", linewidth = 1.2)
```



D.12.3 `geom_segment()`

Draw line segments between specified start and end points. Useful for adding arrows, error bars, or connecting points.

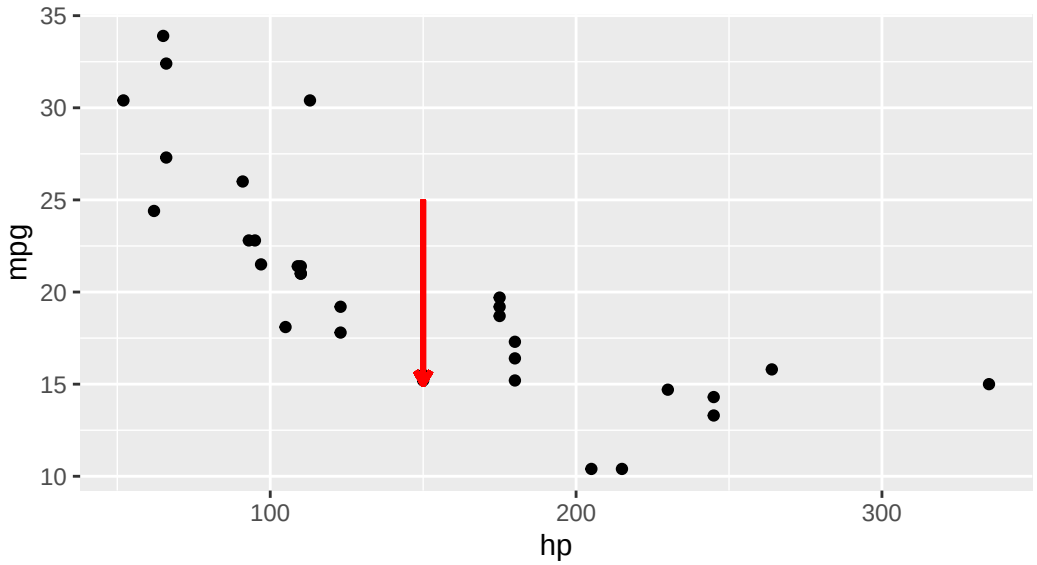
Arguments:

- `aes(x, y, xend, yend)`: Start and end coordinates
- `arrow`: Add arrowheads with `arrow()`
- `color`, `linewidth`, `linetype`: Styling options

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_segment(aes(x = 150, y = 25, xend = 150, yend = 15),
               arrow = arrow(length = unit(0.2, "cm")),
               color = "red", linewidth = 1)
```

```
Warning in geom_segment(aes(x = 150, y = 25, xend = 150, yend = 15), arrow = arrow(length = u
i Please consider using `annotate()` or provide this layer with data containing
  a single row.
```



D.12.4 `geom_ribbon()` and `geom_area()`

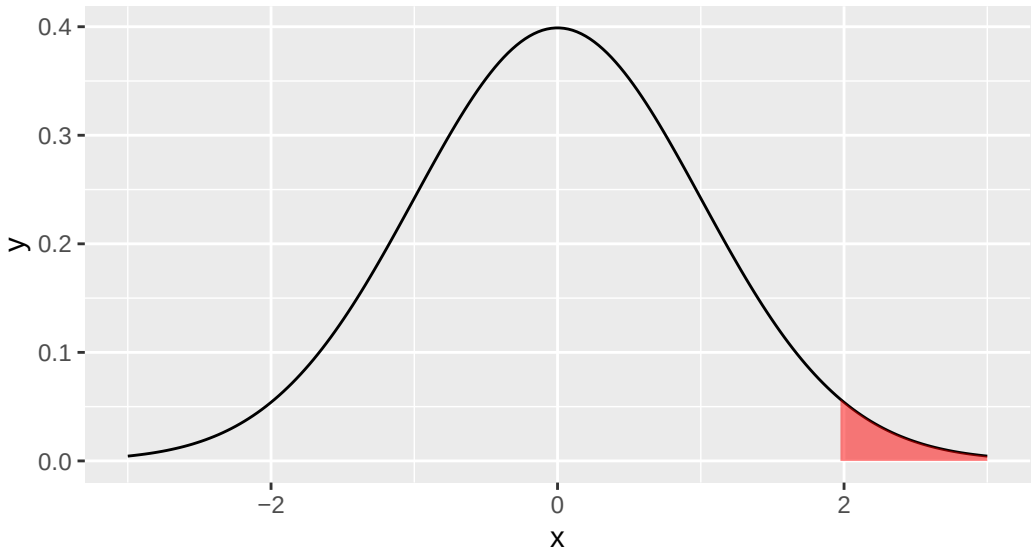
Shade a region between a `ymin` and `ymax`. `geom_area()` is a special case where `ymin = 0`. Useful for shading regions under distribution curves (e.g., p-values, rejection regions).

Arguments:

- `aes(x, ymin, ymax)`: Boundaries of the shaded region
- `fill`: Fill color
- `alpha`: Transparency

Example:

```
shade_data <- tibble(
  x = seq(-3, 3, length.out = 200),
  y = dnorm(x)
)

ggplot(shade_data, aes(x = x)) +
  geom_line(aes(y = y)) +
  geom_ribbon(data = shade_data |> filter(x >= 1.96),
              aes(ymin = 0, ymax = y),
              fill = "red", alpha = 0.5)
```



D.12.5 `geom_errorbar()`

Add error bars to a plot. Can be vertical (default) or horizontal (with `orientation = "y"`). Commonly used for confidence interval plots.

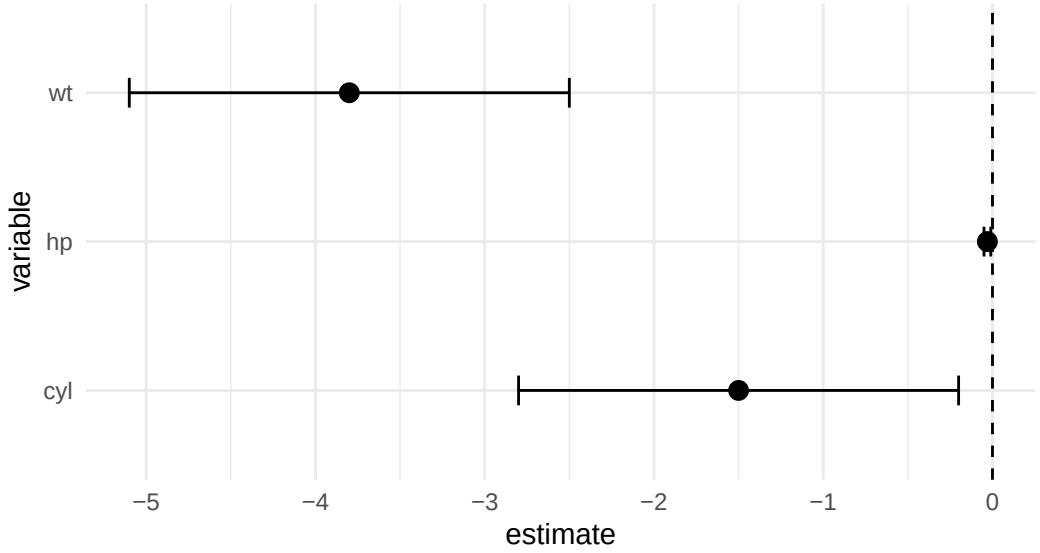
Arguments:

- `aes(ymin, ymax)`: Lower and upper bounds (vertical), or `aes(xmin, xmax)` with `orientation = "y"` (horizontal)
- `width`: Width of the error bar caps
- `orientation`: Set to `"y"` for horizontal error bars

Example:

```
ci_data <- tibble(
  variable = c("hp", "wt", "cyl"),
  estimate = c(-0.03, -3.8, -1.5),
  lower = c(-0.05, -5.1, -2.8),
  upper = c(-0.01, -2.5, -0.2)
)

ggplot(ci_data, aes(y = variable, x = estimate)) +
  geom_point(size = 3) +
  geom_errorbar(aes(xmin = lower, xmax = upper),
```

```
                width = 0.2, orientation = "y") +
  geom_vline(xintercept = 0, linetype = "dashed") +
  theme_minimal()
```



D.12.6 `geom_density()`

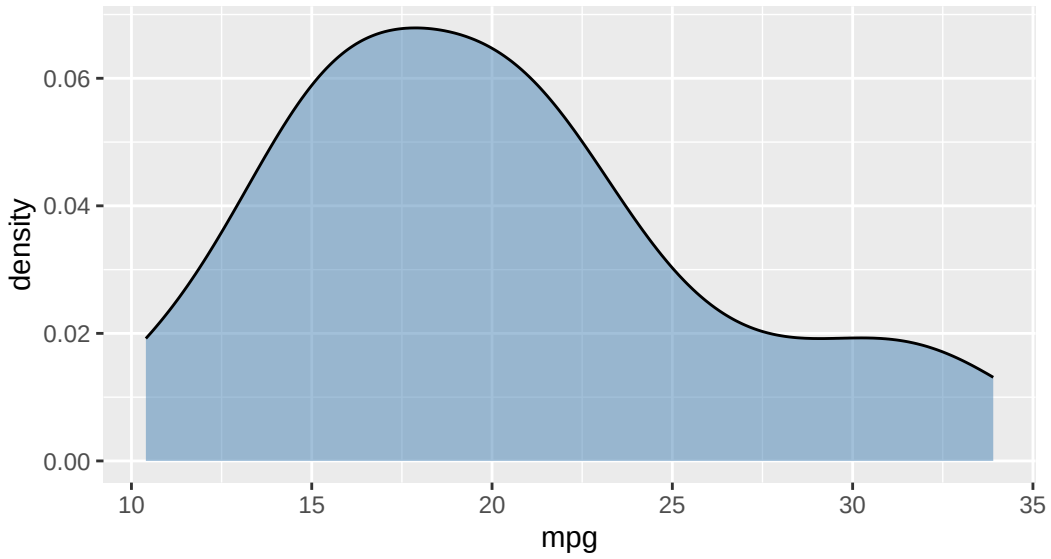
Plot a smoothed density estimate. An alternative to histograms for visualizing distributions.

Arguments:

- `fill`: Fill color
- `alpha`: Transparency
- `color`: Line color

Example:

```
ggplot(mtcars, aes(x = mpg)) +
  geom_density(fill = "steelblue", alpha = 0.5)
```



D.12.7 `scale_color_manual()` and `scale_fill_manual()`

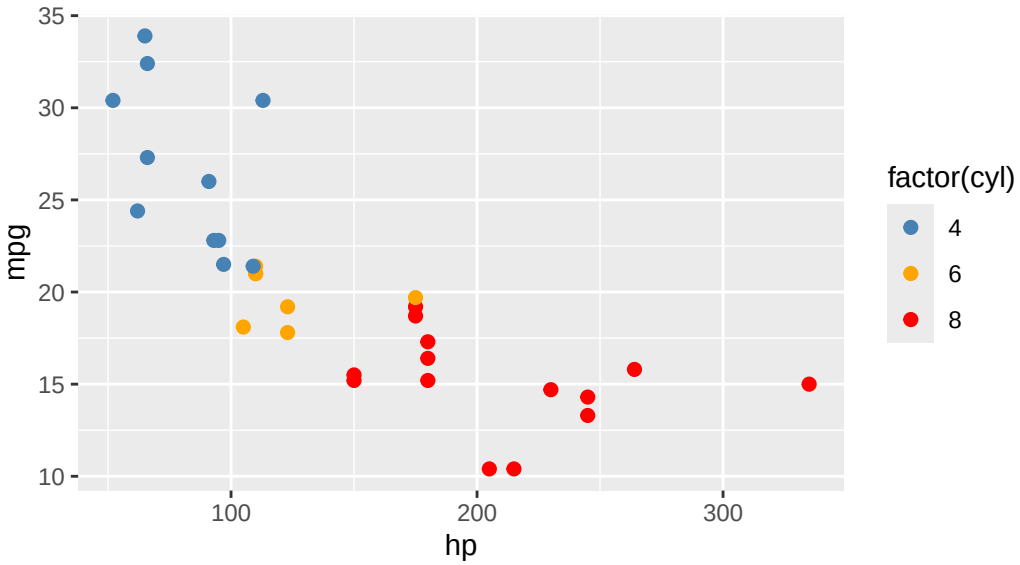
Manually set colors for categorical variables mapped to `color` or `fill`.

Arguments:

- `values`: A named vector of colors
- `labels`: Optional labels for the legend

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(cyl))) +
  geom_point(size = 2) +
  scale_color_manual(values = c("4" = "steelblue", "6" = "orange", "8" = "red"))
```



D.12.8 `coord_cartesian()`

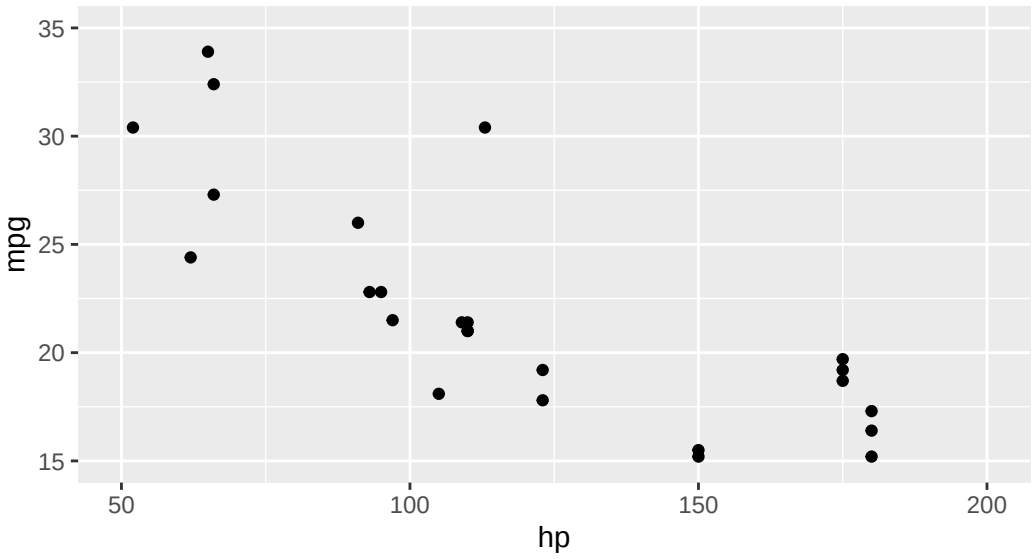
Zoom into a region of the plot without dropping data points (unlike `xlim()`/`ylim()`, which remove data outside the range).

Arguments:

- `xlim`: Range for x-axis as `c(min, max)`
- `ylim`: Range for y-axis as `c(min, max)`

Example:

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  coord_cartesian(xlim = c(50, 200), ylim = c(15, 35))
```



D.13 Quick Reference Tables

D.13.1 Descriptive Statistics Functions

Table D.1: Descriptive statistics functions

Function	Purpose	Key Argument
`mean()`	Average	`na.rm = TRUE`
`sd()`	Standard deviation	`na.rm = TRUE`
`median()`	Middle value	`na.rm = TRUE`
`min()`, `max()`	Extremes	`na.rm = TRUE`
`sum()`	Total	`na.rm = TRUE`
`var()`	Variance	`na.rm = TRUE`
`cor()`	Correlation	`use = "complete.obs"`
`summary()`	Multiple stats	—

D.13.2 Data Manipulation Functions (dplyr)

Table D.2: Data manipulation functions

Function	Purpose	Example
`select()`	Choose columns	`select(data, col1, col2)`
`filter()`	Choose rows	`filter(data, x > 5)`
`mutate()`	Create/modify variables	`mutate(data, new = x * 2)`
`group_by()`	Group data	`group_by(data, category)`
`summarize()`	Aggregate	`summarize(data, avg = mean(x))`
`count()`	Count rows	`count(data, category)`
`arrange()`	Sort rows	`arrange(data, desc(x))`

D.13.3 Regression Functions

Table D.3: Regression functions

Function	Purpose	Package
`lm()`	OLS regression	base R
`summary()`	Model results	base R
`coef()`	Coefficients	base R
`confint()`	Confidence intervals	base R
`predict()`	Fitted values	base R
`residuals()` / `resid()`	Residuals	base R
`anova()`	F-test (compare models)	base R
`nobs()`	Number of observations	base R
`df.residual()`	Residual degrees of freedom	base R
`feols()`	Fixed effects + robust SE	fixest
`modelsummary()`	Formatted tables	modelsummary

D.13.4 Statistical Distribution Functions

Table D.4: Statistical distribution functions

Function	Purpose	Example
`dnorm()`, `dt()`	Density (PDF height)	`dnorm(0)`, `dt(2, df=30)`
`pt()`	CDF for t-distribution	`pt(2.3, df=100)`
`qt()`	Critical values (t-dist)	`qt(0.025, df=100)`
`qf()`	Critical values (F-dist)	`qf(0.05, df1=2, df2=494)`

Function	Purpose	Example
`rnorm()`	Random normal draws	`rnorm(100, mean=0, sd=1)`
`runif()`	Random uniform draws	`runif(100, min=0, max=1)`
`rbinom()`	Random binomial draws	`rbinom(100, size=1, prob=0.5)`
`rexp()`	Random exponential draws	`rexp(100, rate=0.5)`
`rt()`	Random t-dist draws	`rt(100, df=30)`

D.13.5 ggplot2 Geometries

Table D.5: ggplot2 geometries

Function	Plot Type
`geom_point()`	Scatterplot
`geom_line()`	Line plot
`geom_histogram()`	Histogram
`geom_density()`	Smoothed density
`geom_boxplot()`	Boxplot
`geom_bar()`	Bar chart (counts)
`geom_col()`	Bar chart (values)
`geom_smooth()`	Smoothed line/regression
`geom_segment()`	Line segments/arrows
`geom_ribbon()` / `geom_area()`	Shaded regions
`geom_errorbar()`	Error bars / CIs
`geom_hline()`	Horizontal line
`geom_vline()`	Vertical line
`annotate()`	Text/shape annotations
`stat_function()`	Overlay math functions