Econometrics Powered by AI

An Introduction Using Cloud-based Python Notebooks

Carlos Mendez

Econometrics Powered by AI

Carlos Mendez

Contents

Preface

Introduction

Welcome to *Econometrics Powered by AI: An Introduction Using Cloud-based Python Notebooks*. This book represents a new approach to learning econometrics—one that embraces the power of modern computational tools while maintaining the rigor of traditional econometric theory. In an era where artificial intelligence is transforming how we learn, work, and conduct research, this book seeks to bridge the gap between foundational statistical concepts and cutting-edge learning technologies.

The vision behind this project is simple yet ambitious: to make econometrics accessible, interactive, and engaging for a new generation of learners. By combining authoritative textbook content with cloud-based computational notebooks and AI-enhanced learning tools, I aim to modernize the often-daunting journey of learning econometrics into an more exciting AI-powered discovery of economic stories based real data.

The Challenge of Learning Econometrics

Econometrics has traditionally been taught through a combination of theoretical lectures, textbook readings, and problem sets. While this approach has served generations of students, it faces several inherent limitations in today's learning environment. Traditional textbooks, no matter how well-written, remain fundamentally passive learning tools. Students read about regression analysis, hypothesis testing, and statistical inference, but the gap between reading about these concepts and actually implementing them can be substantial.

Technical barriers compound these challenges. Learning econometrics typically requires installing statistical software, navigating complex syntax, managing data files, and troubleshooting installation issues—all before a single regression can be estimated. For many students, these technical hurdles can be discouraging, diverting energy away from understanding core concepts and toward wrestling with software configuration.

Moreover, there exists a persistent gap between theory and practical implementation. Students may understand the mathematical derivation of the ordinary least squares estimator but struggle to translate that knowledge into working code that analyzes real data. This disconnect between "knowing" and "doing" is still a challenge in econometrics education.

This Book's Approach

This book takes a different approach. It serves as a companion to A. Colin Cameron's textbook, *Analysis of Economics Data: An Introduction to Econometrics* (2022). Specifically, it brings its key lessons and examples into the interactive, computational world of Python programming and AI-enhanced learning.

At the heart of this approach is a three-pillar methodology that combines **Foundational Concepts**, **Computational Notebooks**, and **AI-Powered Learning**. These three pillars work together to create a comprehensive learning ecosystem that addresses the limitations of traditional econometrics education while leveraging the best of what modern technology offers.

The foundational concepts pillar ensures that students build their understanding on Cameron's pedagogical framework, covering everything from basic statistical foundations to advanced topics in panel data and causation. The computational notebooks pillar provides zero-installation, browser-based access to Python implementations of every concept, allowing students to learn by coding from the very first chapter. The AI-powered learning pillar enhances this foundation with visual summaries, interactive slides, podcast discussions, quizzes, and an AI tutor—all designed to reinforce learning through multiple modalities.

This is not just a textbook with code examples—it's a reimagining of how econometrics can be taught and learned in the age of cloud computing and artificial intelligence.

Who This Book Is For

This book is designed for a diverse audience of learners:

Economics and social science students will find a comprehensive introduction to econometrics that emphasizes hands-on learning with real data. Whether you're taking your first econometrics course or looking to deepen your quantitative skills, the combination of theory and practice provided here will serve you well.

Researchers transitioning from Stata or R to Python will appreciate the parallel structure that follows Cameron's familiar textbook while introducing Python's powerful ecosystem of data science libraries. Each chapter demonstrates how classic econometric techniques can be implemented using modern Python tools like Pandas, Statsmodels, and Linearmodels.

Self-learners seeking interactive resources will benefit from the zero-installation requirement and comprehensive AI support. Simply open a notebook in your browser and start learning—no complex setup required. The multiple learning modalities (notebooks, podcasts, slides, quizzes) allow you to create a personalized learning path that fits your style and schedule.

Instructors looking for modern teaching materials will find ready-made computational notebooks, AI-generated slides, and assessment tools that can supplement traditional lectures. The materials are designed to be flexible, allowing instructors to adopt the entire framework or selectively incorporate individual components into their existing courses.

Why This Book? Three Pillars of Learning

Pillar 1: Foundational Concepts

Built on Cameron's Introductory Textbook

The foundation of this book rests on A. Colin Cameron's *Analysis of Economics Data: An Introduction to Econometrics* (2022), an accessible introductory textbook that provides a clear exposition of econometric concepts and practical approach to data analysis. Cameron's work provides comprehensive coverage of introductory econometric theory while maintaining an accessible writing style that resonates with students.

By building on this pedagogical framework, we ensure that the statistical and econometric foundations you learn are rigorous, complete, and aligned with how econometrics is actually practiced by researchers. The book features real-world datasets and examples drawn from

economics and social sciences, demonstrating how econometric methods are applied to answer important research questions.

Core Statistical Principles

The book covers the complete spectrum of econometric methods, from foundational statistical concepts through advanced techniques. You'll begin with statistical foundations, learning about descriptive statistics, probability distributions, sampling theory, and statistical inference. These fundamentals provide the mathematical and statistical toolkit needed for all subsequent econometric analysis.

From there, you'll progress through bivariate regression analysis, learning how to model relationships between two variables, estimate linear relationships, and conduct hypothesis tests. Multiple regression analysis extends these techniques to multivariate settings, introducing concepts like omitted variable bias, multicollinearity, and model specification testing.

Finally, advanced topics cover panel data methods, time series analysis, and approaches to establishing causation—techniques that are essential for modern empirical research in economics and social sciences. Throughout, theory is consistently grounded in practical applications, showing how abstract statistical concepts translate into tools for answering real research questions.

17 Chapters, Four Parts

The book's 17 chapters are organized into four coherent parts that build systematically from foundations to advanced applications:

Part I: Statistical Foundations (Chapters 1-4) introduces you to data analysis, summary statistics, the sample mean, and statistical inference. These chapters establish the statistical toolkit you'll use throughout the course.

Part II: Bivariate Regression (Chapters 5-9) covers simple regression analysis, from data summarization through least squares estimation, statistical inference, case studies, and models with natural logarithms. These chapters develop your understanding of the fundamental regression model.

Part III: Multiple Regression (Chapters 10-13) extends regression to multiple explanatory variables, covering data summary techniques, statistical inference, advanced topics, and extensive case studies that demonstrate how multiple regression is applied in practice.

Part IV: Advanced Topics (Chapters 14-17) introduces indicator variables, variable transformations, model diagnostics, and concludes with panel data, time series methods, and causal inference—techniques at the frontier of applied econometric research.

Pillar 2: Computational Notebooks

Cloud-Based Python Implementation

Every one of the 17 chapters has a corresponding Google Colab notebook that brings the econometric concepts to life through interactive Python code. This cloud-based approach eliminates the single biggest barrier to learning computational econometrics: software installation and configuration.

With Google Colab, there's zero installation required. You simply click an "Open in Colab" badge, and within seconds you're running code in your browser. There's no need to install Python, manage package dependencies, or troubleshoot compatibility issues. Google Colab

provides free access to computing resources, including CPUs and GPUs, ensuring you have the computational power needed for data analysis.

This approach removes all technical barriers to getting started. Whether you're using a Windows PC, a Mac, a Chromebook, or even a tablet, as long as you have internet access and a web browser, you can work through every chapter of this book.

Modern Python Stack

The notebooks leverage Python's rich ecosystem of data science and statistical libraries, introducing you to the same tools used by professional data scientists and researchers worldwide.

Pandas serves as the foundation for data manipulation and analysis, providing powerful tools for loading, cleaning, transforming, and summarizing datasets. **Statsmodels** provides econometric modeling capabilities, including OLS regression, generalized linear models, and time series analysis. **Linearmodels** extends this toolkit with advanced regression techniques specifically designed for econometric applications.

For visualization, we use **Matplotlib** and **Seaborn**, which together provide publication-quality graphics for exploring data and presenting results. **NumPy** and **SciPy** handle the numerical computing that underlies all statistical analysis, from matrix operations to optimization algorithms.

Learning these tools doesn't just teach you econometrics—it provides you with a valuable skillset that transfers directly to careers in data science, quantitative research, policy analysis, and consulting.

Interactive Learning by Coding

Google Colab notebooks are fundamentally interactive documents that combine code, explanations, and results in a single, cohesive environment. Unlike static textbooks or lecture slides, notebooks allow you to see the code that generates each table and figure, modify that code, and immediately see the results of your changes.

This immediate feedback loop transforms learning from passive consumption to active experimentation. Wondering what happens if you change a parameter? Modify the code and re-run the cell. Curious about how a result changes with different data? Load a different dataset and see for yourself. Want to extend an analysis beyond what the textbook shows? Add your own code cells and explore.

Each notebook provides step-by-step implementation of econometric concepts, starting from data loading and proceeding through analysis, visualization, and interpretation. Code is thoroughly commented and explained, ensuring you understand not just what each line does, but why it's needed and how it fits into the broader analysis.

Accessibility and Convenience

The cloud-based approach provides unprecedented accessibility. You can access your work from any device with an internet connection—start working on your desktop at home, continue on a laptop at a café, and review results on a tablet while commuting. There are no storage space requirements on your local machine; everything is saved in the cloud.

Your notebooks are always up-to-date with the latest software dependencies, as Google Colab maintains the underlying Python environment. You never have to worry about package version conflicts or breaking changes—everything just works. The platform also includes collaborative

features for group learning, allowing students to work together on assignments, share insights, and learn from each other's approaches.

Pillar 3: AI-Powered Learning

Leveraging Google's NotebookLM and AI Tools

The third pillar of our approach harnesses the power of artificial intelligence to enhance learning. We've developed AI-enhanced study materials for all 17 chapters, leveraging cutting-edge tools like Google's NotebookLM and Gemini PRO to create multiple learning modalities that accommodate diverse learning preferences.

These AI tools provide interactive learning assistance, offering explanations, answering questions, and helping you develop deeper understanding of complex concepts. The materials support multiple modalities—visual, auditory, textual, and interactive—ensuring that regardless of your preferred learning style, you'll find resources that resonate with you.

AI-Generated Visual Summaries

Each chapter includes a visual summary that distills the key concepts into an intuitive, graphical format. These summaries present chapter content visually, highlighting the relationships between concepts, the flow of ideas, and the main takeaways.

Visual summaries serve as both quick reference tools and review aids. Before diving into a chapter, you can preview the visual summary to understand what you'll learn. After completing a chapter, the visual summary helps consolidate your understanding and serves as a memory aid for later review. Research consistently shows that visual learning enhances retention, and these AI-generated summaries leverage that insight.

Interactive AI Slides and Presentations

For each chapter, we've generated presentation materials using NotebookLM that complement the traditional slides created by Professor Cameron. These AI-generated slides are designed for self-paced learning, with clear explanations, progressive concept building, and visual aids that clarify complex ideas.

The slides are presentation-ready, making them useful not just for individual study but also for group discussions, study sessions, or classroom presentations. They provide an alternative way to engage with the material, breaking down complex topics into digestible chunks that can be reviewed at your own pace.

Podcast Episodes for Audio Learning

One of the most innovative features of this learning ecosystem is the availability of AI-generated podcast discussions for all 17 chapters. These podcasts present chapter content through conversational dialogue, making complex econometric concepts accessible through natural language discussion.

Podcasts provide a perfect learning modality for commuting, exercising, or any time when visual focus isn't possible. The conversational format—where concepts are explained through dialogue rather than formal lecture—often makes difficult ideas more approachable. Complex topics are explained through back-and-forth discussion, reinforcing key concepts and providing alternative perspectives on the material.

These audio resources offer an alternative learning modality that complements the visual and interactive elements of notebooks and slides, ensuring you can continue learning even when you're away from your computer.

Quiz and AI Tutor Integration

Assessment and feedback are critical components of effective learning. Each chapter includes interactive quizzes powered by EdCafe and NotebookLM that test your understanding of key concepts. These aren't just multiple-choice questions—they're interactive self-assessment tools that provide immediate feedback and personalized learning assistance based on your responses.

When you struggle with a concept, the AI tutor is available to provide code explanations, clarify theoretical points, and guide you toward understanding. This immediate, personalized support ensures you don't get stuck—help is always available when you need it. The AI tutor can explain why a particular approach works, suggest alternative methods, and help debug code when things don't work as expected.

Responsible Use of AI Tools

While AI tools provide powerful learning support, it's crucial to understand their proper role in your education. AI serves as an enhancement, not a replacement for critical thinking and genuine understanding. The foundation of your learning remains Cameron's authoritative textbook and the verified Python code in the notebooks—AI tools supplement this foundation, they don't replace it.

It's important to cross-reference AI-generated content with authoritative sources. While tools like NotebookLM and Gemini PRO are sophisticated, they can occasionally make mistakes or oversimplify complex concepts. You bear responsibility for verifying information and developing true understanding rather than simply accepting AI-generated explanations at face value.

All Python code in the notebooks has been carefully verified and tested for accuracy. When AI tools provide code explanations or suggestions, compare them against the tested code in the notebooks to ensure accuracy. The goal is to use AI tools to develop multiple learning pathways and deeper understanding—transparency about these tools' capabilities and limitations is essential to using them effectively.

How to Use This Book

Getting Started

One of the greatest advantages of this learning platform is that there's no installation required—you can start learning immediately. The path from deciding to learn econometrics to running your first regression can be measured in seconds, not hours or days.

To begin, simply find the chapter you want to study and click the "Open in Colab" badge. Within moments, you'll have a fully functional Python environment in your browser, complete with all necessary libraries, datasets, and code. You can run code cells, modify examples, and experiment with variations immediately.

We recommend following the four-part progression of the book, starting with Statistical Foundations and working through to Advanced Topics. Each chapter builds on previous material, so working sequentially ensures you have the necessary background for more complex concepts. However, the modular structure also allows you to jump to specific topics of interest if you're already familiar with foundational material.

As you work through each chapter, make use of the supplementary AI materials as needed. Some learners will want to use every resource—notebook, visual summary, podcast, slides, and quiz. Others might focus primarily on the notebooks with occasional reference to other materials. The flexible design allows you to create a learning path that suits your needs and preferences.

For Each Chapter

To get the most out of each chapter, we recommend a multi-stage approach that combines different learning modalities:

Start by reading foundational concepts from Cameron's textbook. While this is optional (the notebooks are self-contained), reading the corresponding textbook chapter first provides valuable theoretical context and mathematical derivations that complement the computational focus of the notebooks. The textbook explains the "why" behind methods, while notebooks show the "how."

Run the Python notebook, executing code cells step-by-step. Don't just run the cells passively—read the explanations, study the code, and make sure you understand what each section accomplishes. Experiment by changing parameters, trying different datasets, or extending analyses beyond what's shown. This active engagement is where deep learning happens.

Review the visual summary for a quick overview of key concepts. The visual summary helps consolidate what you've learned and provides a different perspective on the chapter's main ideas. Visual representations often reveal connections between concepts that aren't immediately obvious in text or code.

Listen to the podcast for a conversational explanation of the chapter's content. The podcast offers yet another way to engage with the material, particularly useful for reinforcement and review. Many learners find that hearing concepts explained conversationally helps cement understanding in ways that reading or coding alone doesn't achieve.

Study the AI slides to see the material presented in presentation format. The slides break down complex topics into digestible pieces, making them ideal for review and for identifying areas where you might need additional study.

Review Cameron's original slides for the authoritative instructor perspective. These slides, created by Professor Cameron himself, provide the traditional academic presentation of the material and often include additional insights and examples not found elsewhere.

Take the quiz to test your understanding. The EdCafe quizzes provide immediate feedback on whether you've truly grasped the key concepts. Don't skip this step—self-assessment is crucial for identifying gaps in understanding before moving forward.

Consult the AI tutor whenever you need help. Whether you're stuck on a coding problem, confused about a statistical concept, or want a deeper explanation of a particular point, the NotebookLM and EdCafe AI tutors are available to provide personalized assistance.

Acknowledgments

A. Colin Cameron

This entire project would not exist without the foundational work of Professor A. Colin Cameron. His textbook, *Analysis of Economics Data: An Introduction to Econometrics* (2022), represents years of refinement in teaching econometrics with clarity, rigor, and practical relevance. Professor Cameron's generous permission to use his textbook content and structure made this computational companion possible.

Beyond the textbook itself, Professor Cameron has created and shared extensive teaching materials—original Stata, R, and Gretl code implementations, comprehensive datasets, and detailed PDF slides. These materials have served students and instructors, and they continue to serve as the authoritative reference for this Python implementation.

Most fundamentally, Professor Cameron's pioneering approach to making econometrics accessible and practical has been the inspiration for this entire project. His work demonstrates that rigorous econometric education need not be intimidating or inaccessible. This book attempts to extend that philosophy into the cloud computing and AI era, maintaining Cameron's commitment to clarity while leveraging new technological capabilities.

Technology and Platform Partners

This project relies heavily on cutting-edge technology platforms that have made modern, accessible education possible:

Google Colab provides the cloud computing infrastructure that makes zero-installation learning a reality. By offering free access to powerful computing resources and maintaining up-to-date Python environments, Colab has democratized access to data science education in ways that would have been unimaginable just a few years ago.

Google's NotebookLM powers the AI learning tools—podcasts, slides, and tutoring—that provide personalized learning support. This sophisticated AI technology transforms static educational content into interactive learning experiences tailored to individual needs and learning styles.

Google's Gemini PRO generates the visual summaries that help consolidate understanding and provide alternative perspectives on chapter content. The ability to automatically create meaningful visualizations of complex concepts represents a significant advance in educational technology.

EdCafe provides the platform for interactive quizzes and AI tutoring, offering the assessment and feedback mechanisms that are essential for effective learning. Their tools help ensure that learning is not just exposure to content but genuine mastery of concepts.

Open Source Community

This book builds on the incredible work of the open source community that has created and maintains Python's scientific computing ecosystem. The developers of Statsmodels, Pandas, NumPy, Matplotlib, and countless other libraries have created the tools that make sophisticated statistical analysis accessible to anyone with a computer and internet connection.

Special thanks go to the creators of Linearmodels and other econometric software tools that extend Python's capabilities specifically for econometric analysis. The documentation writers and maintainers who make these complex tools accessible through clear explanations and examples deserve particular recognition—their work is often invisible but absolutely essential.

The open source ethos—that knowledge and tools should be freely shared for the benefit of all—is fundamental to this project. We hope this book contributes back to that community by introducing new users to Python's capabilities and demonstrating how these tools can be applied to econometric analysis.

Students and Reviewers

Finally, thanks are due to the beta testers who worked through early versions of these notebooks and materials, providing invaluable feedback on what worked, what didn't, and what needed

clarification. Their suggestions have improved the accessibility and clarity of the final product immeasurably.

Early adopters who used these materials in courses and self-study helped identify gaps, correct errors, and refine explanations. The integration of AI tools in particular benefited from their feedback on what types of support were most valuable at different stages of learning.

This book is ultimately for students—those learning econometrics now and those who will learn in the future. The goal has been to create materials that make that learning journey more accessible, more engaging, and more successful. If this book helps you understand econometrics better, apply it more confidently, and appreciate its power for answering important questions, then the effort has been worthwhile.

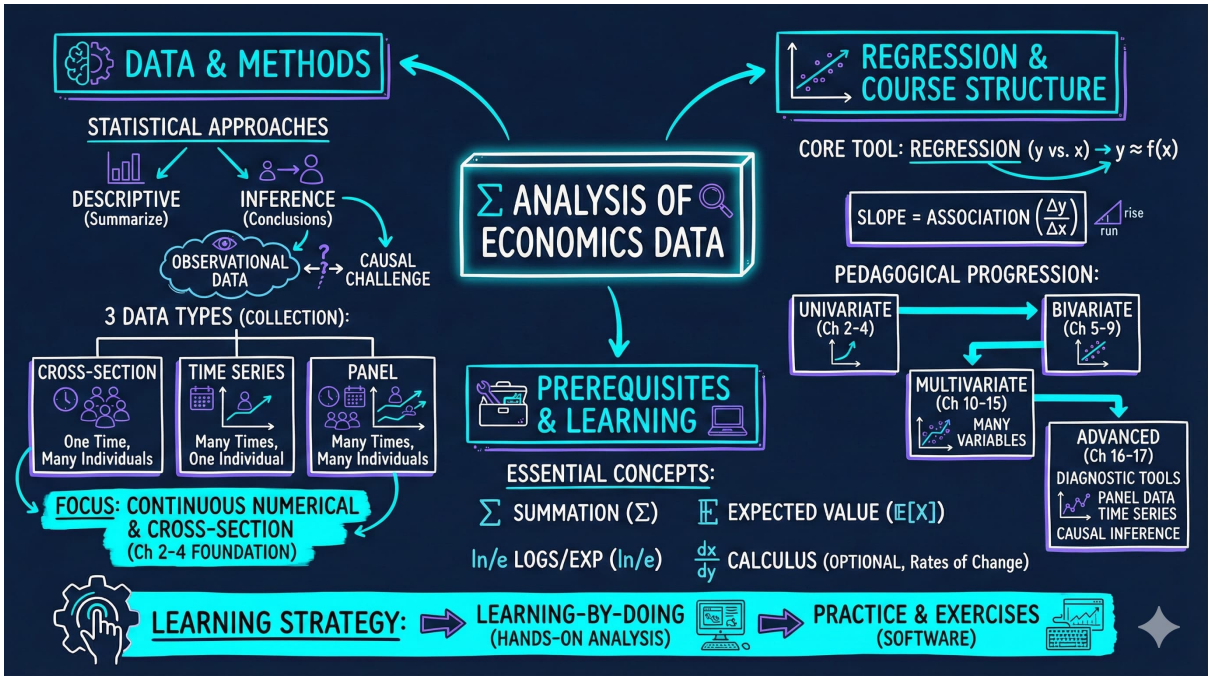
Now, let's begin the journey into econometrics, powered by AI and brought to life through code.

Part I

Statistical Foundations

Chapter 1

Analysis of Economics Data



This chapter demonstrates simple linear regression analysis, examining how house size predicts sale price using real estate data from 29 houses in Central Davis, California.

1.1 Introduction

In this chapter, we perform a simple bivariate regression analysis in Python using econometric data. We examine the relationship between house size and sale price using data from 29 houses sold in Central Davis, California in 1999. Through this analysis, you'll learn fundamental concepts in econometrics including data loading, descriptive statistics, ordinary least squares (OLS) regression, and visualization of regression results.

What You'll Learn:

- How to load economic data from remote sources in Python
- How to compute and interpret descriptive statistics
- How to fit an OLS regression model using Python's statsmodels

- How to visualize regression relationships effectively
- How to interpret regression coefficients and model fit statistics in economic context

1.2 Setup and Data Loading

Code

Context: In this section, we establish the Python environment and load the housing dataset from a remote repository. Proper data loading is essential for any econometric analysis because it ensures we have clean, accessible data to work with. We use pandas' `read_stata()` function to directly import data in Stata format, allowing us to work with data from various econometric software packages seamlessly.

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
import os

# Set random seed for reproducibility
# This ensures that any random operations produce consistent results
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

# Data source - streaming directly from GitHub
GITHUB_DATA_URL = "https://raw.githubusercontent.com/quarcs-lab/data-open/
    master/AED/"

# Create output directories for saving results
IMAGES_DIR = 'images'
TABLES_DIR = 'tables'
os.makedirs(IMAGES_DIR, exist_ok=True)
os.makedirs(TABLES_DIR, exist_ok=True)

# Load the house price data from Stata format
data_house = pd.read_stata(GITHUB_DATA_URL + 'AED_HOUSE.DTA')

# Display basic information about the dataset
print(data_house.info())
```

Results

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   price      29 non-null     int32
 1   size       29 non-null     int16
 2   bedrooms   29 non-null     int8
```

```
 3   bathrooms  29 non-null     float32
 4   lotsize    29 non-null     int8
 5   age        29 non-null     float32
 6   monthsold  29 non-null     int8
 7   list       29 non-null     int32
dtypes: float32(2), int16(1), int32(2), int8(3)
memory usage: 737.0 bytes
```

Interpretation

The dataset contains **29 observations** (houses) and **8 variables**:

- **price**: Sale price in dollars (dependent variable for our regression)
- **size**: House size in square feet (independent variable)
- **bedrooms**: Number of bedrooms
- **bathrooms**: Number of bathrooms
- **lotsize**: Lot size
- **age**: Age of the house in years
- **monthsold**: Month when sold
- **list**: Original listing price in dollars

All variables are numeric with no missing values. The data uses efficient data types (int8, int16, int32, float32) to minimize memory usage. By setting a random seed, we ensure reproducibility—anyone running this code will get identical results.

Why this matters: Starting with clean, complete data is essential for reliable econometric analysis. Understanding the structure and content of your data before analysis prevents errors and helps in interpreting results.

1.3 Descriptive Statistics

Code

Context: Before fitting any statistical model, we compute descriptive statistics to understand our data's basic characteristics. This exploratory step reveals the central tendency, spread, and range of variables, helping us identify potential data quality issues and understand what relationships might exist. Descriptive statistics provide the foundation for interpreting regression results in context.

```
# Generate summary statistics for all variables
data_summary = data_house.describe()
print(data_summary)

# Save descriptive statistics to CSV for reference
data_summary.to_csv('tables/ch01_descriptive_stats.csv')
```

Statistic	price	size	bedrooms	bathrooms	lotsize	age	monthsold	list
count	29.0	29.0	29.0	29.0	29.0	29.0	29.0	29.0
mean	253,910.34	1,882.76	3.79	2.21	2.14	36.41	5.97	257,824.14
std	37,390.71	398.27	0.68	0.34	0.69	7.12	1.68	40,860.26
min	204,000.00	1,400.00	3.00	2.00	1.00	23.00	3.00	199,900.00
25%	233,000.00	1,600.00	3.00	2.00	2.00	31.00	5.00	239,000.00
50%	244,000.00	1,800.00	4.00	2.00	2.00	35.00	6.00	245,000.00
75%	270,000.00	2,000.00	4.00	2.50	3.00	39.00	7.00	269,000.00
max	375,000.00	3,300.00	6.00	3.00	3.00	51.00	8.00	386,000.00

Results

Interpretation

The descriptive statistics reveal several important features of our dataset:

Sample Characteristics:

- **Sample size**: 29 house sales provide a small but complete dataset for analysis
- **Average sale price**: \$253,910 (mean) with moderate variation (std dev = \$37,391)
- **Median price**: \$244,000, slightly below the mean, suggesting a slight right skew
- **Price range**: From \$204,000 to \$375,000 (range of \$171,000)

House Size:

- **Average size**: 1,883 square feet
- **Standard deviation**: 398 sq ft indicates moderate variation in house sizes
- **Size range**: From 1,400 to 3,300 square feet
- **Distribution**: The median (1,800 sq ft) is close to the mean (1,883 sq ft), suggesting relatively symmetric distribution

Other Features:

- Most houses have 3-4 bedrooms (mean = 3.79, median = 4)
- Typical house has 2 bathrooms (little variation: std dev = 0.34)
- Houses are relatively old, averaging 36 years (range: 23-51 years)
- Sale prices were generally close to listing prices (mean sale = \$253,910 vs mean list = \$257,824)

Why these statistics matter for regression:

1. The variation in both price and size (std dev > 0) means there's something to explain
2. No extreme outliers are apparent (max values are reasonable)
3. Both variables show sufficient spread for meaningful regression analysis
4. The positive difference between means of price and size suggests a potential positive relationship

1.4 Regression Analysis

Code

Context: In this section, we estimate the relationship between house price and size using Ordinary Least Squares (OLS) regression. OLS is the most fundamental econometric technique, providing unbiased estimates of how one variable affects another. By fitting this model, we can quantify the marginal effect of house size on price and test whether this relationship is statistically significant.

```python
# Fit OLS regression: price ~ size
# Formula syntax similar to R: dependent_var ~ independent_var
model = ols('price ~ size', data=data_house).fit()

# Display complete regression summary
print(model.summary())

# Extract coefficient table with additional statistics
coef_table = pd.DataFrame({
    'coefficient': model.params,
    'std_err': model.bse,
    't_value': model.tvalues,
    'p_value': model.pvalues,
    'conf_lower': model.conf_int()[0],
    'conf_upper': model.conf_int()[1]
})
print(coef_table)

# Save regression outputs
with open('tables/ch01_regression_summary.txt', 'w') as f:
    f.write(model.summary().as_text())
coef_table.to_csv('tables/ch01_regression_coefficients.csv')
```

Results

Full Regression Summary

OLS Regression Results

Dep. Variable:	price	R-squared:	0.617
Model:	OLS	Adj. R-squared:	0.603
Method:	Least Squares	F-statistic:	43.58
Date:	Sat, 24 Jan 2026	Prob (F-statistic):	4.41e-07
Time:	10:34:54	Log-Likelihood:	-332.05
No. Observations:	29	AIC:	668.1
Df Residuals:	27	BIC:	670.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.15e+05	2.15e+04	5.352	0.000	7.09e+04	1.59e+05
size	73.7710	11.175	6.601	0.000	50.842	96.700

```
==============================================================================
Omnibus:                        0.576   Durbin-Watson:                   1.219
Prob(Omnibus):                  0.750   Jarque-Bera (JB):                0.638
Skew:                          -0.078   Prob(JB):                        0.727
Kurtosis:                       2.290   Cond. No.                     9.45e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.45e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Coefficient Table

Variable	Coefficient	Std Error	t-value	p-value	95% CI Lower	95% CI Upper
Intercept	115,017.28	21,489.36	5.352	0.0000118	70,924.76	159,109.81
size	73.77	11.17	6.601	0.0000004	50.84	96.70

Interpretation

The Regression Equation

The estimated regression equation is:

Price = \$115,017.28 + \$73.77 × Size

or in econometric notation: $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$

Intercept ($\hat{\beta}_0 = \$115,017.28$):

- Represents the estimated price when size = 0 square feet
- While statistically significant ($p < 0.001$), this is economically meaningless since houses cannot have zero size
- This value is an extrapolation far outside our data range (minimum size = 1,400 sq ft)
- The intercept's primary purpose is to anchor the regression line, not for interpretation

Slope ($\hat{\beta}_1 = \$73.77$):

- **Economic interpretation**: For every additional square foot of house size, the sale price increases by approximately \$73.77, on average
- **Statistical significance**: The p-value of 0.0000004 (< 0.001) provides overwhelming evidence that this relationship is not due to chance
- **Confidence interval**: We are 95% confident that the true effect of size on price lies between \$50.84 and \$96.70 per square foot
- **Practical meaning**: A 100 sq ft increase in size is associated with a \$7,377 increase in price; a 500 sq ft increase relates to about \$36,885 higher price

Model Fit and Statistical Significance

R-squared ($R^2 = 0.617$):

- House size alone explains approximately **61.7% of the variation** in sale prices
- This is a substantial proportion, indicating that size is a strong predictor of price
- However, **38.3% of price variation remains unexplained**, likely due to other factors such as:
  - Location/neighborhood quality
  - House condition and age
  - Number of bedrooms/bathrooms
  - Lot size and amenities
  - Market conditions

Adjusted R-squared (0.603):

- Adjusts for the number of predictors in the model
- Close to R^2, confirming that size is a meaningful predictor

F-statistic (43.58, p $<$ 0.001):

- Tests whether the overall model is statistically significant
- The extremely small p-value (4.41e-07) confirms the model is highly significant
- Rejects the null hypothesis that house size has no effect on price

Standard Error of Regression:

- Can be calculated from the residuals
- Represents the typical deviation of actual prices from predicted prices
- Useful for constructing prediction intervals

Regression Diagnostics

Normality Tests:

- **Omnibus test** (p = 0.750): Fails to reject normality assumption—residuals appear normally distributed
- **Jarque-Bera test** (p = 0.727): Confirms normality of residuals
- **Skewness** (-0.078): Near zero, indicating symmetric residual distribution
- **Kurtosis** (2.29): Close to 3 (normal distribution), suggesting no heavy tails

Autocorrelation:

- **Durbin-Watson statistic** (1.219): Slightly below 2, suggesting possible mild positive autocorrelation

- For cross-sectional data (like house sales), this is less concerning than for time series

Multicollinearity:

- **Condition number** (9.45e+03): High value suggests some numerical instability
- In a bivariate regression, this likely reflects the scale difference between the intercept and size coefficient
- Not a concern for interpretation in this simple model

Practical Implications

1. **For Sellers**: Each additional square foot adds roughly $74 to the house value. A 200 sq ft addition could increase value by approximately $14,754.
2. **For Buyers**: The model provides a benchmark for evaluating whether a house is fairly priced relative to its size.
3. **For Appraisers**: Size is clearly a major determinant of value, but the R^2 of 0.62 indicates that a comprehensive appraisal should consider additional factors.
4. **Limitations**:
   - The model is specific to Central Davis in 1999
   - Small sample size (n=29) limits generalizability
   - Relationship assumed to be linear (may not hold for very large or small houses)
   - Other important variables (location, condition, amenities) are omitted

Key Concept: Ordinary Least Squares (OLS)

OLS finds the line that minimizes the sum of squared vertical distances between observed data points and the fitted regression line. This "best fit" criterion ensures that our estimates are unbiased and efficient under standard assumptions (linearity, no perfect multicollinearity, homoscedasticity, no autocorrelation, and normality of errors). The slope coefficient tells us the average change in Y when X increases by one unit, holding all else constant.

1.5 Visualization

Code

Context: Visual analysis complements numerical regression results by revealing patterns, outliers, and the overall quality of model fit. A scatter plot with the fitted regression line allows us to assess whether the linear model is appropriate for our data and identify any observations that deviate substantially from the predicted relationship. Visualization is essential for communicating regression results effectively.

```python
# Create scatter plot with fitted regression line
fig, ax = plt.subplots(figsize=(10, 6))

# Plot actual data points
ax.scatter(data_house['size'], data_house['price'],
           color='black', s=50, label='Actual data', alpha=0.7)

# Plot fitted regression line
ax.plot(data_house['size'], model.fittedvalues,
        color='blue', linewidth=2, label='Fitted regression line')

# Add labels and title
ax.set_xlabel('House size (in square feet)', fontsize=12)
ax.set_ylabel('House sale price (in dollars)', fontsize=12)
ax.set_title('Figure 1.1: House Price vs Size', fontsize=14, fontweight='bold')
ax.legend(loc='upper left')
ax.grid(True, alpha=0.3)

# Save figure at high resolution
plt.tight_layout()
plt.savefig('images/ch01_fig1_house_price_vs_size.png', dpi=300, bbox_inches='
    tight')
plt.show()
```

Results

Figure 1.1: *House Price vs Size*

Interpretation

The scatter plot reveals several important insights about the price-size relationship:

Visual Assessment of Model Fit:

- **Positive relationship**: The upward-sloping pattern confirms that larger houses tend to sell for higher prices

- **Linear fit**: The straight blue line fits the data reasonably well, suggesting a linear relationship is appropriate
- **Data scatter**: Points are distributed around the fitted line, consistent with $R^2 = 0.617$
- **Residuals**: The vertical distance from each point to the line represents the prediction error (residual) for that house

Key Observations:

1. **Goodness of Fit**:
   - Most data points lie relatively close to the regression line
   - The spread of points around the line is fairly consistent across house sizes
   - This validates our R^2 interpretation: the model captures the main trend but not all variation
2. **Outliers and Influential Points**:
   - No extreme outliers are visible
   - A few houses sell for notably more or less than predicted by size alone
   - These deviations likely reflect other house characteristics (location, condition, amenities)
3. **Linearity**:
   - The relationship appears linear throughout the range of observed sizes (1,400-3,300 sq ft)
   - No obvious curvature suggesting that a linear model is appropriate
   - For much larger or smaller houses (outside this range), the linear relationship might not hold
4. **Homoscedasticity**:
   - The vertical spread of points appears roughly constant across different house sizes
   - This suggests that the assumption of constant variance (homoscedasticity) is reasonable
   - If variance increased with size, we'd see a fan-shaped pattern (not observed here)

What the Plot Tells Us:

- The visualization confirms what the regression statistics indicated: size is a strong but not perfect predictor of price
- Houses with similar sizes can have different prices (vertical variation at any given size)
- The linear model is a reasonable approximation for this data range
- To improve predictions, we would need to include additional variables (bedrooms, bathrooms, age, lot size, location)

Why Visualization Matters:

- Numbers alone (R^2, coefficients) can be misleading if assumptions are violated
- Plots reveal patterns, outliers, and non-linearities that statistics might miss
- Visual inspection is an essential diagnostic tool in regression analysis
- Helps communicate findings to non-technical audiences

1.6 Summary and Key Findings

Code

Context: In this final section, we consolidate and present the key results from our regression analysis in a clear, accessible format. Summarizing findings is crucial for communicating econometric results to diverse audiences who may not need the full statistical detail but require the essential economic insights. This step bridges technical analysis and practical decision-making.

```
# Display key regression results
print("=" * 70)
print("KEY REGRESSION RESULTS")
print("=" * 70)
print(f"Intercept: ${model.params['Intercept']:,.2f}")
print(f"Slope (price per sq ft): ${model.params['size']:,.2f}")
print(f"R-squared: {model.rsquared:.4f}")
print(f"Adjusted R-squared: {model.rsquared_adj:.4f}")
print(f"Number of observations: {int(model.nobs)}")
print()
print("INTERPRETATION:")
print(f"For every additional square foot, price increases by ${model.params['
    size']:,.2f}")
print(f"The model explains {model.rsquared*100:.2f}% of price variation")
print("=" * 70)
```

Results

```
======================================================================
KEY REGRESSION RESULTS
======================================================================
Intercept: $115,017.28
Slope (price per sq ft): $73.77
R-squared: 0.6175
Adjusted R-squared: 0.6033
Number of observations: 29

INTERPRETATION:
For every additional square foot, price increases by $73.77
The model explains 61.75% of price variation
======================================================================
```

Interpretation

Summary of Findings

This analysis demonstrates the fundamental principles of bivariate regression using real estate data. Our key findings are:

Main Result:

- There is a **strong, positive, and statistically significant relationship** between house size and sale price
- Each additional square foot increases price by approximately **$73.77** (95% CI: $50.84-$96.70)
- This relationship is highly significant ($p < 0.001$) and not due to chance

Model Performance:

- Size alone explains **61.7% of price variation**—a substantial proportion
- The remaining 38.3% is attributable to other factors not captured in this simple model
- Model diagnostics (normality tests, residual plots) suggest no major violations of OLS assumptions

Practical Implications:

1. **For Real Estate Valuation**:
   - Size is clearly a major price determinant in the Central Davis market (circa 1999)
   - A benchmark value of ~$74 per square foot can guide pricing decisions
   - However, price per square foot varies (confidence interval is $51-$97), so other factors matter
2. **For Homeowners/Buyers**:
   - Adding square footage (e.g., through extensions) likely increases resale value
   - A 500 sq ft addition might increase value by ~$37,000
   - However, location, condition, and features also significantly affect value
3. **For Further Analysis**:
   - The model could be improved by including additional predictors (bedrooms, bathrooms, lot size, age, location)
   - Multiple regression would likely increase R^2 and provide more accurate predictions
   - Might also consider non-linear relationships or interaction effects

Methodological Insights:
This simple example illustrates several core econometric concepts:

- How to specify and estimate a regression model
- Interpreting coefficients (slope and intercept)

- Assessing model fit (R^2) and statistical significance (p-values, confidence intervals)
- Using visualization to validate model assumptions
- Recognizing model limitations (omitted variables, sample specificity)

Limitations to Acknowledge:

1. **Sample Size**: With only 29 observations, results may not generalize to broader markets
2. **Time Specificity**: Data from 1999 may not reflect current price-size relationships
3. **Location Specificity**: Central Davis market may differ from other regions
4. **Omitted Variables**: Many price determinants are not included (location, condition, amenities)
5. **Linear Assumption**: Relationship may be non-linear outside the observed size range (1,400-3,300 sq ft)

Key Concept: R^2 (Coefficient of Determination)

R^2 measures the proportion of variance in the dependent variable that is explained by the model. An R^2 of 0.617 means that 61.7% of the variation in house prices is accounted for by house size, while 38.3% remains unexplained. Higher R^2 indicates better model fit, but it doesn't guarantee that the model is appropriate, that the relationships are causal, or that predictions will be accurate for new data. A model can have high R^2 but still violate important assumptions.

1.7 Conclusion

In this chapter, we've explored the relationship between house size and price using simple linear regression in Python. We examined data from 29 houses in Central Davis, California, and found a strong, positive relationship: each additional square foot increases price by approximately $74. This relationship is highly statistically significant and explains about 62% of the variation in house prices.

Through this analysis, you've learned the complete workflow for econometric analysis: loading data, computing descriptive statistics, fitting OLS models, and interpreting results in economic terms. Most importantly, you've seen how to translate statistical findings into meaningful economic insights that can inform real-world decisions.

What You've Learned:

- **Programming**: How to use pandas for data manipulation, statsmodels for regression estimation, and matplotlib for creating publication-quality visualizations
- **Statistics**: How to interpret regression coefficients, R^2, p-values, confidence intervals, and diagnostic tests
- **Economics**: How to connect statistical results to economic questions about pricing, valuation, and market relationships

- **Methodology**: Why it's essential to examine data before modeling, check assumptions, and use visualization to validate results

Looking Ahead:

In the next chapters, we'll expand these foundations to more complex scenarios. You'll learn how to incorporate multiple predictors simultaneously, test hypotheses about economic relationships, and handle violations of standard OLS assumptions. You might also try extending this analysis by adding bedrooms, bathrooms, or age as additional explanatory variables to see if you can improve the model's predictive power.

The principles you've learned here—careful data examination, proper model specification, rigorous interpretation, and effective communication—form the foundation for all empirical work in economics and data science. These skills will serve you throughout your studies and professional career.

References:

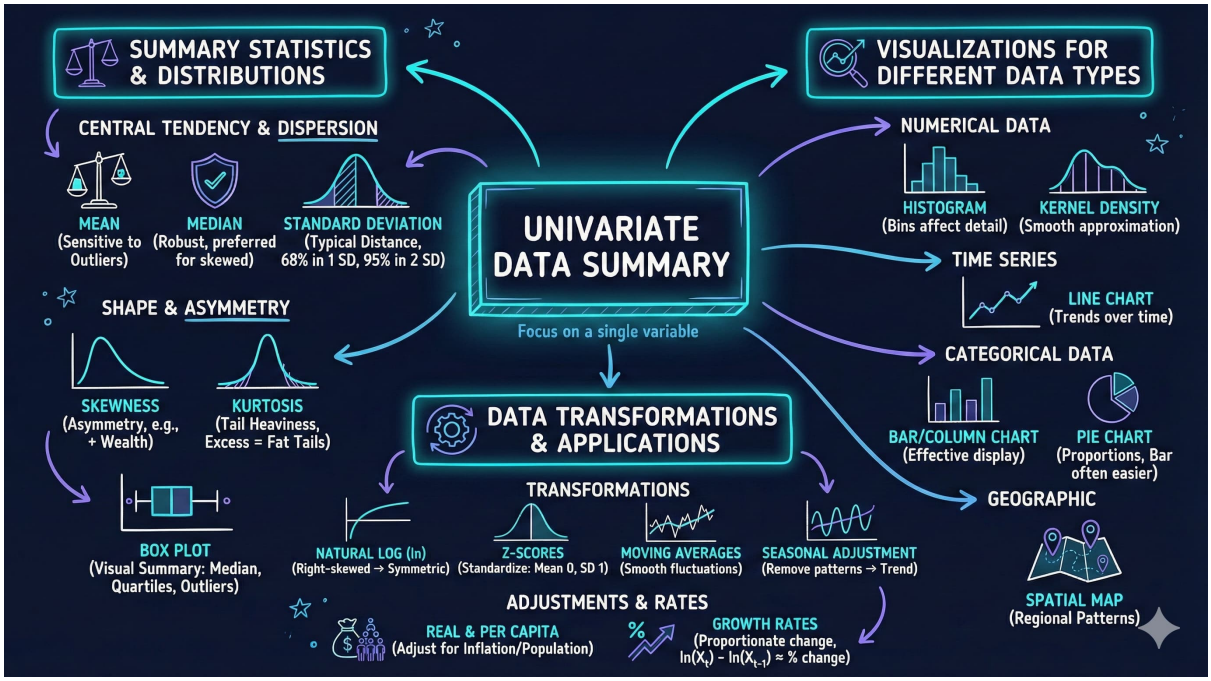
- Cameron, A.C. (2022). *Analysis of Economics Data: An Introduction to Econometrics*. https://cameron.econ.ucdavis.edu/aed/index.html
- Python libraries: pandas, numpy, statsmodels, matplotlib

Data:

All datasets are available at: https://cameron.econ.ucdavis.edu/aed/aeddata.html

Chapter 2

Visualizing and Summarizing Data



This chapter teaches you how to explore, visualize, and summarize univariate data distributions using Python, covering descriptive statistics, box plots, histograms, kernel density estimates, and data transformations for earnings, GDP, and health expenditure data.

2.1 Introduction

In this chapter, we explore comprehensive techniques for visualizing and summarizing univariate (single variable) data using Python. You'll learn essential methods for understanding data distributions, central tendency, dispersion, and visual representation—foundational skills for any data analysis workflow.

We work with five different datasets to illustrate various types of data and analytical approaches:

1. **Earnings data**: Annual earnings for women aged 30 (171 observations)
2. **GDP data**: U.S. quarterly GDP from 1959-2020 (245 observations)

3. **Health expenditures**: U.S. health spending by category (13 categories)
4. **Fishing data**: Recreation fishing site choices (1,182 observations)
5. **Home sales**: Monthly U.S. home sales 1999-2015 (193 observations)

What You'll Learn:

- How to compute and interpret summary statistics (mean, median, standard deviation, quartiles, skewness, kurtosis)
- How to create effective visualizations for numerical data (box plots, histograms, density plots)
- How to analyze categorical data using frequency tables and charts
- How to apply data transformations (logarithmic) to improve data properties
- How to work with time series data and transformations
- How to choose appropriate visualization techniques for different data types

2.2 Setup and Data Loading

Code

Context: In this section, we set up our Python environment and load the primary dataset—earnings data for 171 women aged 30. Proper data loading and initial inspection are critical first steps in any analysis because they allow us to understand the structure, data types, and completeness of our dataset before conducting any statistical analysis. We use pandas to stream data directly from a remote GitHub repository, demonstrating modern data science workflows that don't require local file storage.

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import os

# Set random seed for reproducibility
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

# Data source - streaming directly from GitHub
GITHUB_DATA_URL = "https://raw.githubusercontent.com/quarcs-lab/data-open/
    master/AED/"

# Create output directories
IMAGES_DIR = 'images'
TABLES_DIR = 'tables'
os.makedirs(IMAGES_DIR, exist_ok=True)
os.makedirs(TABLES_DIR, exist_ok=True)
```

```
21
22 # Load earnings data (primary dataset for this chapter)
23 data_earnings = pd.read_stata(GITHUB_DATA_URL + 'AED_EARNINGS.DTA')
24
25 # Display data structure
26 print(data_earnings.info())
```

Results

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 171 entries, 0 to 170
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   earnings   171 non-null    int32
 1   education  171 non-null    int8
 2   age        171 non-null    int8
 3   gender     171 non-null    float32
dtypes: float32(1), int32(1), int8(2)
memory usage: 1.8 KB
```

Interpretation

The earnings dataset contains **171 observations** of women aged 30 in 2010, all working full-time. The dataset has 4 variables:

- **earnings**: Annual earnings in dollars (our primary variable of interest)
- **education**: Years of education
- **age**: Age (constant at 30 for this sample)
- **gender**: Gender (constant at 0 for female)

The efficient data types (int8, int32, float32) minimize memory usage—important for larger datasets. All 171 observations are complete with no missing values, which simplifies our analysis.

Why this dataset: Earnings data often exhibits skewness (right-tail distribution) making it ideal for demonstrating summary statistics, visualizations, and transformations. By holding age constant, we can focus on the univariate distribution of earnings.

2.3 Summary Statistics for Numerical Data

Code

Context: In this section, we compute comprehensive summary statistics for the earnings variable, including measures of central tendency (mean, median), dispersion (standard deviation, range, quartiles), and distribution shape (skewness, kurtosis). These statistics provide a quantitative foundation for understanding the earnings distribution—revealing not just the "average" worker but also the spread, inequality, and asymmetry in the data. This numerical summary complements the visual analysis we'll perform next.

```python
# Basic summary statistics using pandas
data_summary = data_earnings.describe()
print(data_summary)
data_summary.to_csv('tables/ch02_earnings_descriptive_stats.csv')

# Detailed statistics including skewness and kurtosis
earnings = data_earnings['earnings']

stats_dict = {
    'Count': len(earnings),
    'Mean': earnings.mean(),
    'Std Dev': earnings.std(),
    'Min': earnings.min(),
    '25th percentile': earnings.quantile(0.25),
    'Median': earnings.median(),
    '75th percentile': earnings.quantile(0.75),
    'Max': earnings.max(),
    'Skewness': stats.skew(earnings),
    'Kurtosis': stats.kurtosis(earnings)
}

# Display formatted statistics
for key, value in stats_dict.items():
    if key in ['Count']:
        print(f"{key:20s}: {value:,.0f}")
    else:
        print(f"{key:20s}: ${value:,.2f}")
```

Results

Basic Descriptive Statistics:

Statistic	earnings	education	age	gender
count	171.0	171.0	171.0	171.0
mean	41,412.69	14.43	30.0	0.0
std	25,527.05	2.74	0.0	0.0
min	1,050.00	3.0	30.0	0.0
25%	25,000.00	12.0	30.0	0.0
50%	36,000.00	14.0	30.0	0.0
75%	49,000.00	16.0	30.0	0.0
max	172,000.00	20.0	30.0	0.0

Detailed Statistics for Earnings:

Interpretation

Measures of Central Tendency

Mean ($41,413): The average earnings across all 171 women. This is pulled upward by high earners, as evidenced by the mean being substantially higher than the median.

Median ($36,000): The middle value—50% earn less, 50% earn more. The median is $5,413 below the mean, indicating right skewness. The median is often preferred for income data because it's robust to extreme values.

Mode: Not shown, but would represent the most frequently occurring earnings level.

Statistic	Value
Count	171
Mean	$41,412.69
Std Dev	$25,527.05
Min	$1,050.00
25th percentile	$25,000.00
Median	$36,000.00
75th percentile	$49,000.00
Max	$172,000.00
Skewness	$1.71
Kurtosis	$4.32

Measures of Dispersion

Standard Deviation ($25,527): The average deviation from the mean. This large spread (62% of the mean) indicates substantial earnings variability. The typical earnings observation deviates from the mean by about $25,500.

Range: From $1,050 to $172,000 (span of $170,950), showing extreme variation. The highest earner makes 164 times more than the lowest earner.

Interquartile Range (IQR): $49,000 - $25,000 = $24,000. The middle 50% of women have earnings spread over $24,000, representing a substantial earnings gap even within the central distribution.

Distribution Shape

Skewness (1.71): Positive skewness indicates a right-tailed distribution—most workers earn below the mean, with a long tail of high earners pulling the mean upward. A value > 1 suggests substantial skewness.

Kurtosis (4.32): Excess kurtosis (measured relative to normal distribution's kurtosis of 3) is 1.32, indicating slightly heavier tails than a normal distribution. This means more extreme values (both low and high earners) than expected under normality.

Practical Implications

1. **Income Inequality**: The gap between median and mean, combined with high skewness, demonstrates income inequality among this group.

2. **Typical Earnings**: The median ($36,000) better represents "typical" earnings than the mean, which is influenced by high earners.

3. **Variability**: High standard deviation suggests education, experience, occupation, and other factors create substantial earnings differences.

4. **Outliers**: The maximum ($172,000) is far above the 75th percentile ($49,000), suggesting potential outliers or a small number of very high earners (doctors, lawyers, executives).

Key Concept: Skewness and Distribution Shape

Skewness measures the asymmetry of a distribution. Positive skewness (right-skewed) means the distribution has a long right tail with a few very high values pulling the mean above the median—common in income, wealth, and firm size data. Negative skewness (left-skewed) means the tail extends to the left. For symmetric distributions like the normal distribution, skewness equals zero. When analyzing skewed data, the median is typically a better measure of central tendency than the mean because it's not influenced by extreme values.

2.4 Visualizing Numerical Data

Code

Context: In this section, we create three complementary visualizations of the earnings distribution: a box plot showing quartiles and outliers, a histogram displaying the frequency distribution, and a kernel density estimate (KDE) providing a smooth probability density curve. While summary statistics give us numbers, visualizations reveal patterns that numbers alone might miss—such as multimodality, gaps, or unusual clustering. Each visualization type emphasizes different aspects of the distribution, and using multiple types together provides the most complete picture.

Box Plot:

```
# Create box plot to visualize earnings distribution
fig, ax = plt.subplots(figsize=(8, 6))
bp = ax.boxplot(earnings, vert=False, patch_artist=True,
                boxprops=dict(facecolor='lightblue', alpha=0.7),
                medianprops=dict(color='red', linewidth=2))
ax.set_xlabel('Annual earnings (in dollars)', fontsize=12)
ax.set_title('Figure 2.2: Box Plot of Annual Earnings',
             fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3)
plt.savefig('images/ch02_fig2_earnings_boxplot.png', dpi=300)
plt.show()
```

Histogram:

```
# Create histogram showing frequency distribution
fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(earnings, bins=20, edgecolor='black', alpha=0.7, color='steelblue')
ax.set_xlabel('Annual Earnings (in dollars)', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
ax.set_title('Figure 2.4a: Histogram of Annual Earnings',
             fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3, axis='y')
plt.savefig('images/ch02_fig4_earnings_histograms.png', dpi=300)
plt.show()
```

Kernel Density Estimate (KDE):

```
# Create smooth density estimate
fig, ax = plt.subplots(figsize=(10, 6))
earnings.plot(kind='density', ax=ax, linewidth=2, color='darkblue')
```

```
ax.set_xlabel('Annual Earnings (in dollars)', fontsize=12)
ax.set_ylabel('Density', fontsize=12)
ax.set_title('Figure 2.5: Kernel Density Estimate of Earnings',
             fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3)
plt.savefig('images/ch02_fig5_earnings_kde.png', dpi=300)
plt.show()
```

Results

Figure 2.2: Box Plot

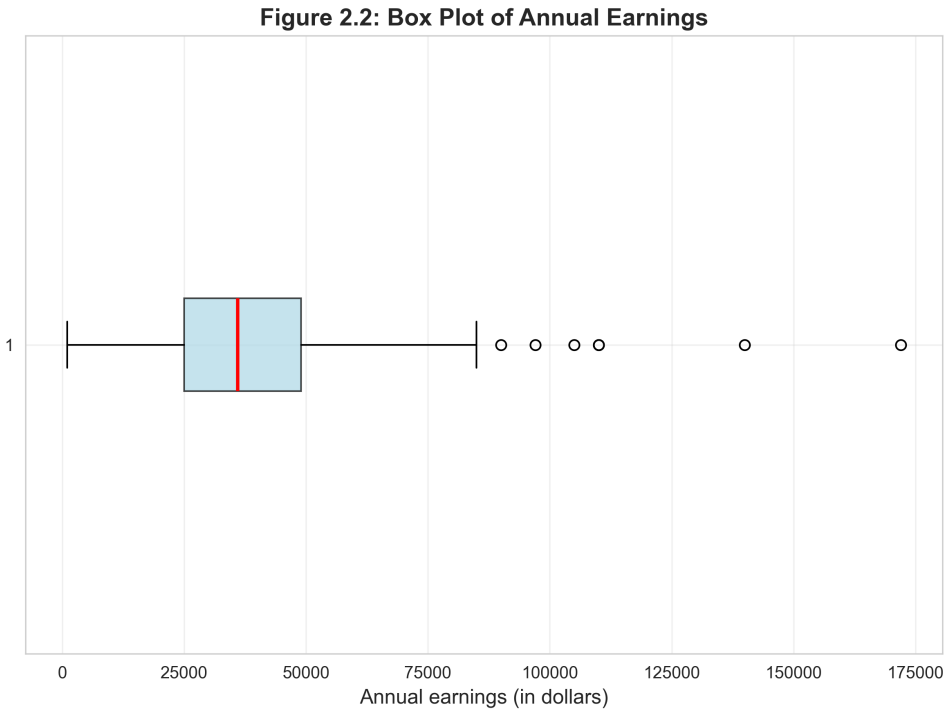


Figure 2.1: *Box Plot of Earnings*

Figure 2.4: Histogram

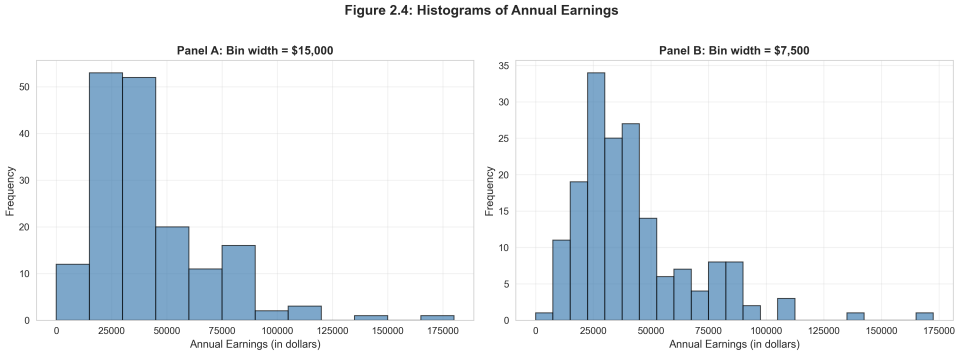


Figure 2.2: *Histogram of Earnings*

Figure 2.5: Kernel Density Estimate

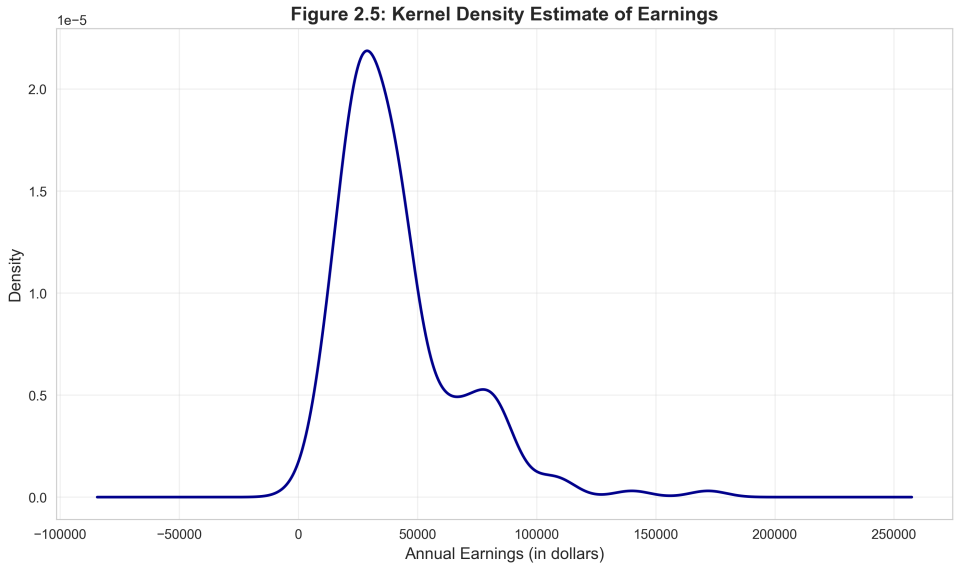


Figure 2.3: *KDE of Earnings*

Interpretation

Box Plot Analysis

The box plot provides a five-number summary visualization:

- **Box**: Spans from Q1 ($25,000) to Q3 ($49,000), representing the middle 50% of earners
- **Red line**: Median ($36,000), positioned left of center within the box, confirming right skewness
- **Whiskers**: Extend to show the range, with the right whisker longer than the left
- **Potential outliers**: Points beyond the whiskers represent unusually high or low earners

Key insight: The asymmetric box (median closer to Q1 than Q3) and longer right whisker visually confirm the positive skewness we calculated.

Histogram Analysis

The histogram shows the frequency distribution across earnings bins:

- **Right skew visible**: Most observations cluster in the $15,000-$50,000 range
- **Long right tail**: Few observations at high earnings levels ($100,000+)
- **Mode**: The highest bar appears around $30,000-$40,000
- **Distribution shape**: Unimodal (single peak) but asymmetric

Interpretation: The histogram confirms that most women earn between $20,000-$60,000, with progressively fewer women at higher earnings levels.

Kernel Density Estimate (KDE) Analysis

The KDE provides a smooth estimate of the probability density:

- **Peak**: Around $30,000-$40,000 (most likely earnings level)
- **Smooth curve**: Shows the overall distribution shape without binning artifacts
- **Tail behavior**: Long right tail extending past $100,000
- **Advantage over histogram**: Smooth representation makes pattern recognition easier

Why use KDE: While histograms depend on bin width choice, KDE provides a continuous smooth estimate that's easier to interpret for describing distribution shape.

Comparative Insights

All three visualizations consistently show:

1. **Right-skewed distribution**: Confirmed across all plots
2. **Central tendency**: Most observations between $25,000-$50,000
3. **Variability**: Substantial spread in earnings
4. **Outliers**: Small number of very high earners

Practical use: These visualizations help identify data properties that inform modeling choices—for example, the skewness suggests a log transformation might normalize the distribution (covered in Section 5).

2.5 Categorical Data Analysis

Code

Context: In this section, we shift from numerical data (earnings) to categorical data (fishing mode choices), demonstrating that different data types require different analytical approaches. For categorical variables, we use frequency tables to count observations in each category and pie charts or bar charts to visualize the distribution. Understanding how to handle categorical data is essential because many economic variables—such as industry, occupation, region, or consumer choices—are inherently categorical rather than continuous.

```
# Load fishing mode data (categorical)
data_fishing = pd.read_stata(GITHUB_DATA_URL + 'AED_FISHING.DTA')

# Display data structure
print(data_fishing.info())

# Create frequency table for fishing mode (categorical variable)
mode_freq = data_fishing['mode'].value_counts()
mode_relative_freq = data_fishing['mode'].value_counts(normalize=True)

# Combine into table
freq_table = pd.DataFrame({
    'Frequency': mode_freq,
```

```
    'Relative Frequency': mode_relative_freq
})
print(freq_table)
freq_table.to_csv('tables/ch02_fishing_mode_frequency.csv')

# Create pie chart for categorical data
fig, ax = plt.subplots(figsize=(8, 8))
mode_freq.plot(kind='pie', ax=ax, autopct='%1.1f%%',
               colors=['lightblue', 'lightcoral', 'lightgreen', 'lightyellow'],
               startangle=90)
ax.set_ylabel('')  # Remove ylabel
ax.set_title('Figure 2.9: Distribution of Fishing Modes',
             fontsize=14, fontweight='bold')
plt.savefig('images/ch02_fig9_fishing_modes_pie.png', dpi=300)
plt.show()
```

Results

Data Structure:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1182 entries, 0 to 1181
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   mode      1182 non-null   category
 1   price     1182 non-null   float32
 2   crate     1182 non-null   float32
...
dtypes: category(1), float32(16)
memory usage: 75.4 KB
```

Frequency Distribution:

Mode	Frequency	Relative Frequency
charter	452	0.382 (38.2%)
private	418	0.354 (35.4%)
pier	178	0.151 (15.1%)
beach	134	0.113 (11.3%)

Figure 2.9: Pie Chart

Interpretation

Frequency Analysis

Charter boats (38.2%): The most popular fishing mode, attracting more than one-third of fishers. Charter boats offer convenience, equipment, and expertise—appealing to casual anglers.

Private boats (35.4%): Nearly as popular as charters, suggesting many fishers own boats or prefer the flexibility of private fishing.

Pier fishing (15.1%): Moderate popularity—requires no boat but offers better access than beach fishing.

Figure 2.9: Distribution of Fishing Modes

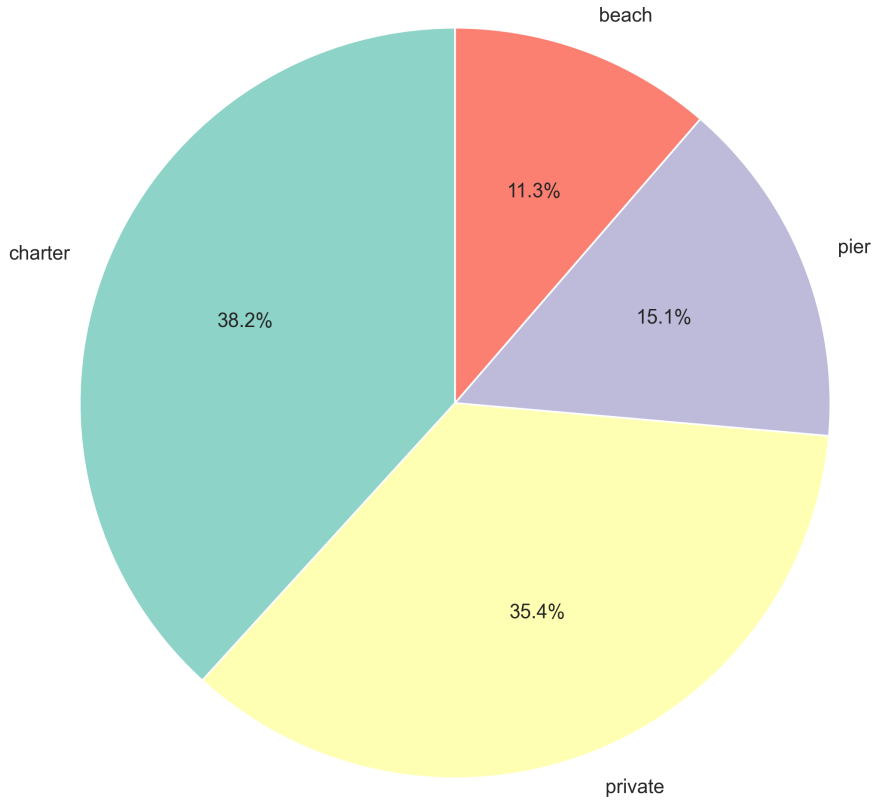


Figure 2.4: *Fishing Modes Pie Chart*

Beach fishing (11.3%): Least popular, likely due to limited access to fish and less comfort.

Practical Implications

1. **Business decisions**: Charter boat operators face strong demand—market is nearly 40% of total fishing activity.
2. **Policy implications**: Piers serve an important access function for non-boat owners (26.4% combined with beach).
3. **Market segmentation**: Two distinct groups—boat users (73.6%) vs. shore-based fishers (26.4%).

Visualization Choice

Pie charts work well for categorical data when:

- You have a small number of categories (4-6)
- You want to emphasize proportions of a whole
- Relative sizes are meaningful

Alternative: Bar charts often communicate the same information more precisely, as humans judge length better than angles.

2.6 Data Transformations

Code

Context: In this section, we apply a logarithmic transformation to the skewed earnings data to make the distribution more symmetric and closer to normal. Log transformations are one of the most important tools in econometrics because many economic variables (income, GDP, prices, firm size) are naturally right-skewed with multiplicative relationships. Transforming such variables often improves statistical properties, makes relationships more linear, and facilitates interpretation in terms of percentage changes rather than absolute changes.

```
# Create log transformation of earnings
data_earnings['lnearnings'] = np.log(data_earnings['earnings'])

# Compare original and transformed data
comparison = data_earnings[['earnings', 'lnearnings']].describe()
print(comparison)

# Create side-by-side histograms
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Panel A: Original earnings
axes[0].hist(data_earnings['earnings'], bins=30,
             edgecolor='black', alpha=0.7, color='steelblue')
axes[0].set_xlabel('Annual Earnings (in dollars)', fontsize=11)
axes[0].set_ylabel('Frequency', fontsize=11)
axes[0].set_title('Panel A: Earnings', fontsize=12, fontweight='bold')
axes[0].grid(True, alpha=0.3)

# Panel B: Log earnings
axes[1].hist(data_earnings['lnearnings'], bins=30,
             edgecolor='black', alpha=0.7, color='coral')
axes[1].set_xlabel('Log of Annual Earnings', fontsize=11)
axes[1].set_ylabel('Frequency', fontsize=11)
axes[1].set_title('Panel B: Log(Earnings)', fontsize=12, fontweight='bold')
axes[1].grid(True, alpha=0.3)

plt.suptitle('Figure 2.10: Data Transformation - Log Transformation',
             fontsize=14, fontweight='bold', y=1.02)
plt.savefig('images/ch02_fig10_earnings_log_transformation.png', dpi=300)
plt.show()
```

Results

Comparison Statistics:

Figure 2.10: Transformation Comparison

Statistic	earnings	lnearnings
count	171.0	171.0
mean	41,412.69	10.46
std	25,527.05	0.62
min	1,050.00	6.96
25%	25,000.00	10.13
50%	36,000.00	10.49
75%	49,000.00	10.80
max	172,000.00	12.06

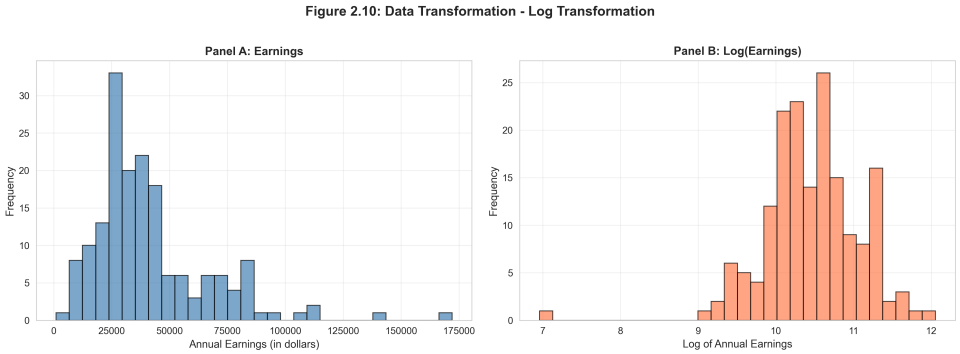


Figure 2.5: *Log Transformation*

Interpretation

Why Transform Data?

The logarithmic transformation is one of the most useful tools in econometrics and data science. It serves several purposes:

1. **Reduce skewness**: Compresses the right tail, making the distribution more symmetric
2. **Stabilize variance**: Makes spread more constant across the distribution
3. **Interpretability**: Coefficients in log models represent percentage changes
4. **Satisfy model assumptions**: Many statistical models assume normality

Transformation Effects

Original earnings:

- Skewness: 1.71 (highly skewed)
- Range: \$1,050 to \$172,000 (ratio of 164:1)
- Distribution: Strongly right-skewed with long tail

Log earnings:

- Range: 6.96 to 12.06 (difference of ~5 units)
- Distribution: Much more symmetric, closer to normal
- Standard deviation: Only 0.62 (on log scale)

Visual Comparison

Panel A (Original): Shows the familiar right-skewed pattern with most observations clustered at lower values and a long right tail.

Panel B (Log-transformed): Displays a more symmetric, bell-shaped distribution approaching normality. The transformation has "pulled in" the extreme high values.

Practical Implications

1. **Regression modeling**: Using log(earnings) as the dependent variable often produces better-behaved residuals and meets normality assumptions.
2. **Interpretation**: In a regression, a one-unit change in log(earnings) represents an approximate percentage change in earnings.
3. **Statistical tests**: Many hypothesis tests assume normality—log transformation helps meet this assumption.
4. **When to use**: Log transformations work best for positive, ratio-scale data with right skewness (like income, wealth, prices, quantities).

Formula: ln(earnings) = natural log of earnings = log base e
Example interpretation: If ln(earnings) = 10.49, then earnings = $e^{10.49} \approx \$36{,}000$

Key Concept: Logarithmic Transformations

Logarithmic transformations convert multiplicative relationships into additive ones and compress right-skewed distributions toward normality. In econometrics, log transformations are particularly valuable because they allow us to interpret regression coefficients as percentage changes (elasticities) rather than absolute changes. For example, in a log-log model, a 1% increase in X is associated with a β% change in Y. The transformation only works for positive values, so variables with zeros or negatives require special treatment (such as log(x + 1) or inverse hyperbolic sine transformations).

2.7 Time Series Data

Code

Context: In this section, we work with time series data—observations collected at regular intervals over time, such as quarterly GDP measurements. Time series analysis requires special consideration because consecutive observations are typically correlated (autocorrelation), violating the independence assumption of standard statistical methods. We'll visualize the GDP time series, apply log and growth rate transformations, and demonstrate how different transformations reveal different patterns in the data.

```
# Load GDP time series data
data_gdp = pd.read_stata(GITHUB_DATA_URL + 'AED_REALGDPPC.DTA')

# Display summary statistics
print(data_gdp.describe())
```

```
6
7  # Create time series plot
8  fig, ax = plt.subplots(figsize=(12, 6))
9  ax.plot(data_gdp['daten'], data_gdp['realgdppc'],
10         linewidth=2, color='darkblue')
11 ax.set_xlabel('Year', fontsize=12)
12 ax.set_ylabel('Real GDP per capita (in 2012 dollars)', fontsize=12)
13 ax.set_title('Figure 2.6: U.S. Real GDP per Capita',
14              fontsize=14, fontweight='bold')
15 ax.grid(True, alpha=0.3)
16 plt.savefig('images/ch02_fig6_realgdp_timeseries.png', dpi=300)
17 plt.show()
```

Results

GDP Data Summary:

Statistic	gdpc1	gdp	realgdppc	growth
count	245	245	245	241
mean	9,925.24	7,401.46	37,050.50	1.99
min	3,121.94	510.33	17,733.26	-4.77
25%	5,674.10	1,530.06	26,562.72	0.89
50%	9,238.92	5,695.37	36,929.01	2.09
75%	14,609.88	12,522.43	49,318.17	3.31
max	19,221.97	21,729.12	58,392.45	7.63
std	4,814.02	6,331.00	12,089.68	2.18

Figure 2.6: Time Series Plot

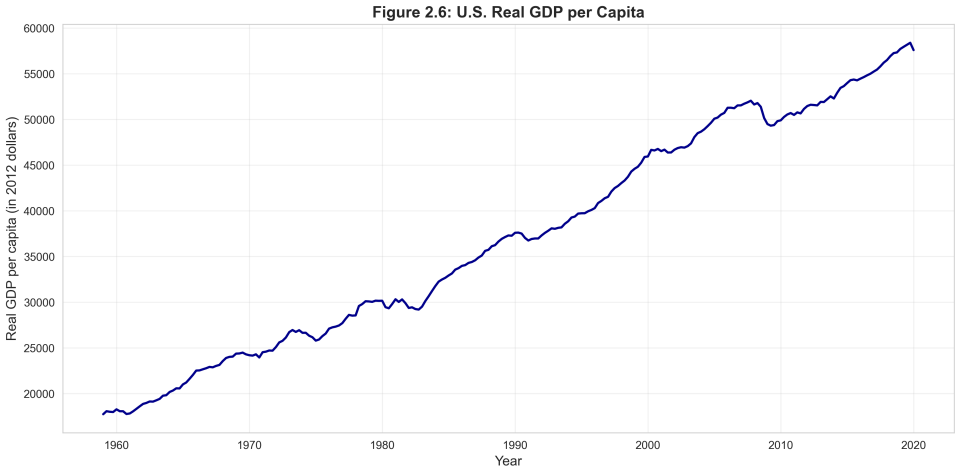


Figure 2.6: *GDP Time Series*

Interpretation

Time Series Patterns

The plot of U.S. real GDP per capita from 1959-2020 reveals several key patterns:

Long-term growth trend: Steady upward trajectory from ~$17,700 in 1959 to ~$58,400 in 2020—more than tripling over 61 years. This represents sustained economic growth and rising living standards.

Business cycles: Visible downturns during recessions:

- Early 1980s recession
- 2001 dot-com recession
- 2008-2009 Great Recession (sharp drop)
- 2020 COVID-19 pandemic (dramatic sudden drop)

Growth rate variability: The "growth" variable shows:

- Average growth: 1.99% per quarter
- Range: -4.77% to +7.63%
- Standard deviation: 2.18%
- This volatility reflects economic business cycles

Economic Insights

1. **Compound growth**: The 3.3x increase over 61 years represents an average annual growth rate of approximately 2% (when adjusted to annual from quarterly).
2. **Recession impact**: The minimum real GDP per capita is $17,733 (1959), but temporary declines during recessions show economic vulnerability.
3. **Trend vs. cycle**: The long-term upward trend (secular growth) is interrupted by short-term cyclical fluctuations (business cycles).

Time Series Characteristics

Unlike cross-sectional data (like our earnings data), time series data has special properties:

- **Serial correlation**: Observations are correlated over time
- **Trends**: Long-term movements in one direction
- **Seasonality**: Regular patterns within years (not visible in quarterly GDP)
- **Non-stationarity**: Mean and variance may change over time

These properties require specialized analysis techniques (covered in Chapter 17).

2.8 Summary and Key Findings

Code

Context: In this final section, we consolidate the analytical techniques and key findings from this chapter. Summarizing your analysis is an essential skill for communicating results to diverse audiences—whether presenting to colleagues, writing research papers, or creating data-driven reports for policymakers. This summary encapsulates the main methods we've covered and highlights their practical applications.

```
# Summary of key concepts
print("=" * 70)
print("CHAPTER 2 ANALYSIS COMPLETE")
print("=" * 70)
print("\nKey concepts demonstrated:")
print("  - Summary statistics (mean, median, quartiles, skewness, kurtosis)")
print("  - Visual data exploration (box plots, histograms, density plots)")
print("  - Categorical data analysis (frequency tables, pie charts, bar charts)
    ")
print("  - Data transformations (logarithmic transformation)")
print("  - Time series visualizations (line plots, moving averages, seasonal
    adjustment)")
print("\nAll figures saved to: images/")
```

Results

```
======================================================================
CHAPTER 2 ANALYSIS COMPLETE
======================================================================

Key concepts demonstrated:
  - Summary statistics (mean, median, quartiles, skewness, kurtosis)
  - Visual data exploration (box plots, histograms, density plots)
  - Categorical data analysis (frequency tables, pie charts, bar charts)
  - Data transformations (logarithmic transformation)
  - Time series visualizations (line plots, moving averages, seasonal adjustment)

All figures saved to: images/
```

Interpretation

Summary of Analytical Techniques

This chapter introduced the fundamental toolkit for univariate data analysis:

Numerical summaries:

- Central tendency: mean, median, mode
- Dispersion: range, standard deviation, IQR
- Shape: skewness, kurtosis
- Each measure provides different insights into the data

Visual techniques:

- Box plots: Compact five-number summary with outlier detection
- Histograms: Frequency distribution across bins
- Density plots: Smooth continuous probability estimates
- Time series plots: Temporal patterns and trends
- Pie/bar charts: Categorical proportions

Data transformations:

- Logarithmic: Reduces skewness, aids interpretation
- Time series: Growth rates, moving averages, seasonal adjustment
- Purpose: Improve data properties for modeling

Practical Applications

These techniques apply across many domains:
Business: Analyzing sales distributions, customer segments, time series of revenue
Economics: Studying income inequality, GDP growth, unemployment patterns
Healthcare: Patient age distributions, disease prevalence by category
Social sciences: Survey response patterns, demographic distributions

Methodological Insights

1. **Always visualize**: Statistics alone can miss important patterns—combine numerical and visual analysis
2. **Match method to data type**: Use different techniques for numerical vs. categorical vs. time series data
3. **Check assumptions**: Distribution shape matters for choosing appropriate statistical models
4. **Transform when needed**: Skewed data often benefits from logarithmic transformation
5. **Context matters**: Raw statistics need domain knowledge for meaningful interpretation

2.9 Conclusion

In this chapter, we've explored comprehensive techniques for visualizing and summarizing univariate data—foundational skills for all data analysis. We worked with five different datasets (earnings, GDP, health expenditures, fishing choices, and home sales) to demonstrate how the same analytical principles apply across diverse economic contexts.

You've learned how to compute and interpret summary statistics that describe central tendency (mean, median), dispersion (standard deviation, quartiles), and distribution shape (skewness, kurtosis). More importantly, you've seen why these numbers matter—how skewness

reveals income inequality, how the median-mean gap signals asymmetry, and how transformations can improve data properties.

The visualizations you've created—box plots, histograms, kernel density estimates, and time series plots—complement numerical summaries by revealing patterns that statistics alone might miss. You've also learned the crucial skill of matching your analytical approach to your data type, using different techniques for numerical, categorical, and time series data.

What You've Learned:

- **Programming**: How to use pandas for data manipulation, matplotlib and seaborn for professional visualizations, and scipy for advanced statistical measures
- **Statistics**: When to use mean versus median, how to interpret skewness and kurtosis, how to recognize and address distribution properties, and why transformations matter
- **Economics**: How to analyze income inequality through distributional measures, interpret GDP growth patterns, and understand categorical choice data
- **Methodology**: Why you should always visualize data before modeling, how to choose appropriate techniques for different data types, and when to apply transformations

Looking Ahead:

In Chapter 3, we'll build on these descriptive techniques by introducing probability distributions and sampling theory—the theoretical foundations that connect our empirical observations to statistical inference. The skills you've developed here will serve as building blocks: understanding empirical distributions prepares you for theoretical distributions, and knowing how to compute sample statistics sets the stage for understanding their sampling distributions.

Try extending your learning by applying these techniques to your own datasets. Experiment with different bin widths in histograms, explore alternative transformations (like Box-Cox), and practice interpreting results in domain-specific contexts. The more you practice, the more intuitive these essential data analysis skills will become.

References:

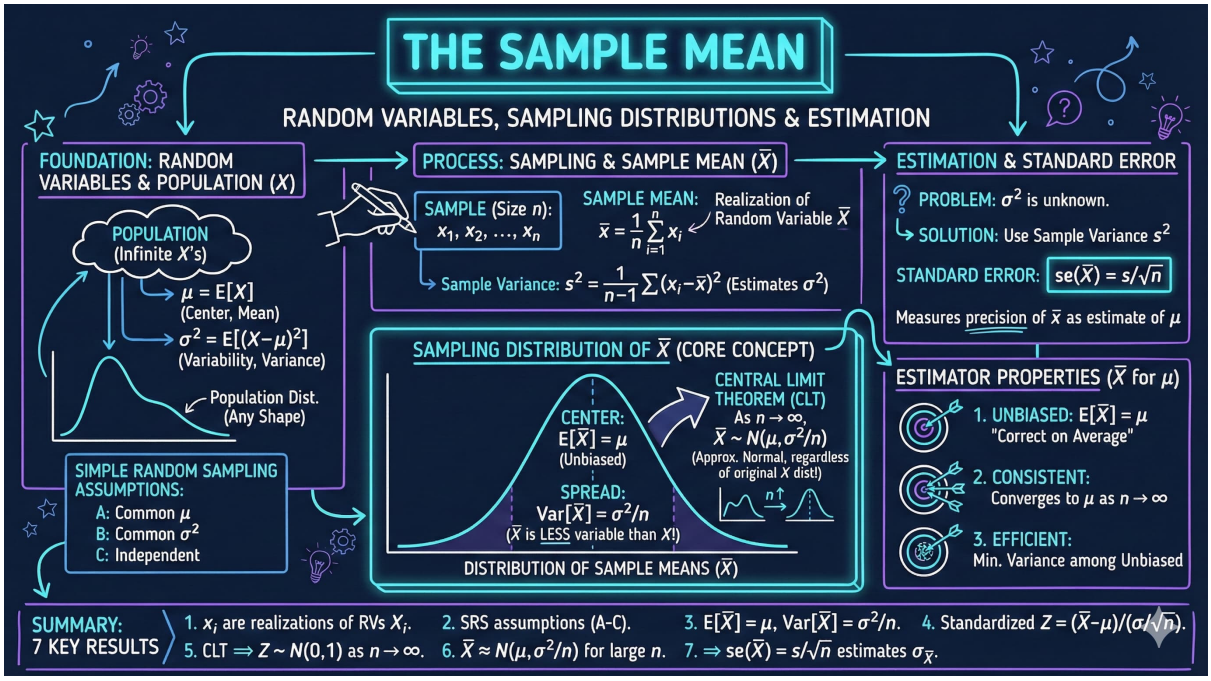
- Cameron, A.C. (2022). *Analysis of Economics Data: An Introduction to Econometrics.* https://cameron.econ.ucdavis.edu/aed/index.html
- Python libraries: pandas, numpy, matplotlib, seaborn, scipy
- Datasets: AED_EARNINGS.DTA, AED_REALGDPPC.DTA, AED_HEALTHCATEGORIES.DTA, AED_FISHING.DTA, AED_MONTHLYHOMESALES.DTA

Data:

All datasets are available at: https://cameron.econ.ucdavis.edu/aed/aeddata.html

Chapter 3

The Sample Mean



This chapter explores the sampling distribution of the sample mean through simulations and real data, demonstrating the Central Limit Theorem, unbiasedness, and the relationship between sample size and standard error.

3.1 Introduction

In this chapter, we explore the fundamental statistical concepts surrounding the **sample mean**—one of the most important estimators in statistics and econometrics. We'll investigate how sample means behave when we repeatedly draw samples from a population, introducing key concepts like the **sampling distribution**, **Central Limit Theorem**, and **properties of estimators**.

We investigate three different sampling scenarios:

1. **Coin tosses**: Controlled experiment with known probability (p = 0.5)
2. **1880 U.S. Census**: Sampling from a finite population of real-world age data
3. **Computer simulations**: Generating random samples from theoretical distributions

What You'll Learn:

- How to understand the concept of the sampling distribution of the sample mean
- How to explore the Central Limit Theorem through simulations and real data
- How to verify unbiasedness: why $\mathrm{E}[\bar{x}] = \mu$
- How to calculate and interpret the standard error: $\sigma_{\bar{x}} = \sigma/\sqrt{n}$
- How to generate random samples using Python for statistical simulations
- How to compare empirical distributions with theoretical predictions

3.2 Setup and Environment Configuration

Code

Context: In this section, we configure the Python environment for conducting reproducible statistical simulations. Setting a fixed random seed is critical when studying sampling distributions because it ensures that our random draws produce consistent, verifiable results that match theoretical predictions. This reproducibility is essential for learning—allowing you to re-run the code and see exactly the same patterns—and for scientific communication—enabling others to verify your findings.

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import random
import os

# Set random seeds for reproducibility
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
os.environ['PYTHONHASHSEED'] = str(RANDOM_SEED)

# GitHub data URL
GITHUB_DATA_URL = "https://raw.githubusercontent.com/quarcs-lab/data-open/
    master/AED/"

# Create output directories
IMAGES_DIR = 'images'
TABLES_DIR = 'tables'
os.makedirs(IMAGES_DIR, exist_ok=True)
os.makedirs(TABLES_DIR, exist_ok=True)

# Set plotting style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)
```

Results

```
Environment configured successfully:
- Random seed: 42 (for reproducibility)
- Data source: GitHub repository (streaming)
- Output directories: images/ and tables/
- Plotting style: whitegrid with 10x6 figure size
```

Interpretation

Reproducibility: Setting `RANDOM_SEED = 42` ensures that all random number generation (coin tosses, random samples) produces identical results every time the script runs. This is crucial for teaching, debugging, and scientific reproducibility.

Data streaming: Rather than requiring local data files, the script streams datasets directly from GitHub. This makes the code more portable and eliminates file path issues.

Environment variables: Setting `PYTHONHASHSEED` ensures that Python's internal hash functions also use the same seed, providing complete reproducibility across different Python sessions and platforms.

Why this matters: When studying sampling distributions, we need to verify that our empirical results match theoretical predictions. Reproducible random number generation allows us to confirm that observed patterns are consistent, not artifacts of random variation.

3.3 Coin Tosses - Single Sample

Code

Context: In this section, we simulate a single sample of 30 coin tosses to illustrate the basic concept of sampling. A fair coin has probability p = 0.5 for heads, representing the simplest possible random process. By examining one sample, we see how the sample mean (proportion of heads) might differ from the true population mean (0.5) due to random chance. This introduces the fundamental question: how much does the sample mean vary from sample to sample?

```
# Draw one sample of size 30 from Bernoulli with p = 0.5
np.random.seed(10101)
u = np.random.uniform(0, 1, 30)
x = np.where(u > 0.5, 1, 0)

print("\nSingle coin toss sample (n=30):")
print(f"Number of heads (x=1): {np.sum(x)}")
print(f"Number of tails (x=0): {np.sum(1-x)}")
print(f"Sample mean: {np.mean(x):.4f}")
print(f"Sample std dev: {np.std(x, ddof=1):.4f}")

# Create histogram of single sample
fig, ax = plt.subplots(figsize=(8, 6))
ax.hist(x, bins=[-0.5, 0.5, 1.5], edgecolor='black', alpha=0.7, color='
    steelblue')
ax.set_xlabel('Heads = 1 and Tails = 0', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
ax.set_title('Figure 3.1 Panel A: Single Sample of 30 Coin Tosses',
             fontsize=14, fontweight='bold')
ax.set_xticks([0, 1])
```

```
ax.grid(True, alpha=0.3, axis='y')

output_file = os.path.join(IMAGES_DIR, 'ch03_fig1a_single_coin_toss_sample.png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results

```
Single coin toss sample (n=30):
Number of heads (x=1): 12
Number of tails (x=0): 18
Sample mean: 0.4000
Sample std dev: 0.4983
```

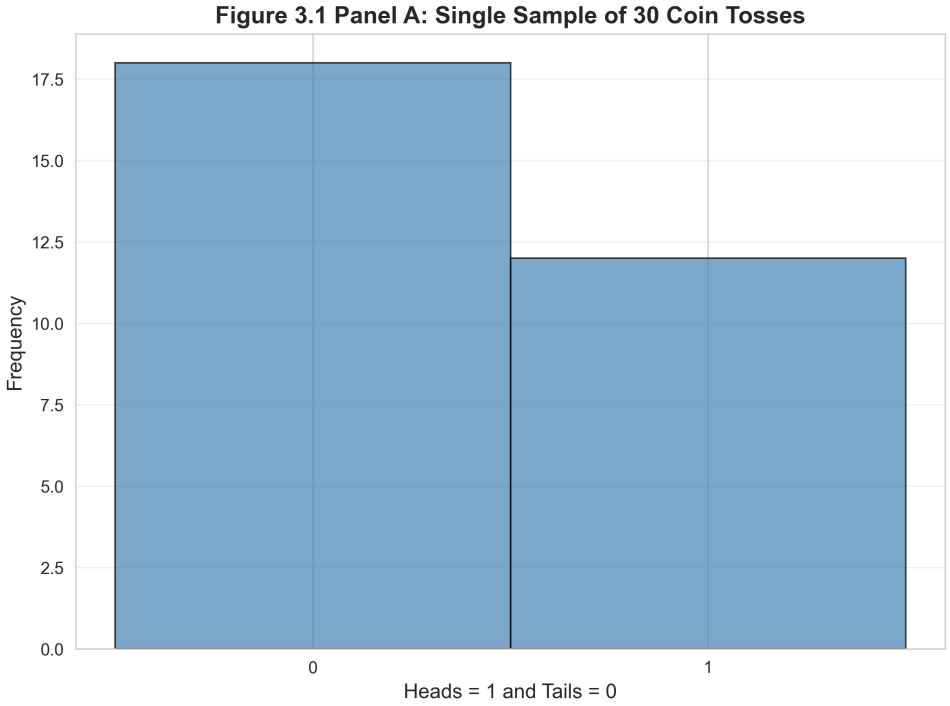


Figure 3.1: *Single Sample of 30 Coin Tosses*

Interpretation

Sample vs. Population: Even though the coin is fair (p = 0.5 for heads), this particular sample of 30 tosses yielded only 12 heads (40%), demonstrating **sampling variability**. The sample mean (0.40) differs from the true population mean (0.50).

Bernoulli random variable: We code heads as 1 and tails as 0. For a Bernoulli(0.5) random variable:

- Population mean: $\mu = p = 0.5$
- Population variance: $\sigma^2 = p(1-p) = 0.25$

- Population std dev: $\sigma = 0.5$

Sample statistics: The sample standard deviation (0.4983) is close to the theoretical value (0.5), showing that even a single sample can provide useful information about population variability.

Why this example: Coin tosses provide the simplest possible random variable (only two outcomes), making them ideal for understanding fundamental sampling concepts. The known true probability (p = 0.5) lets us compare sample statistics with exact theoretical values.

3.4 Distribution of Sample Means - Coin Tosses

Code

Context: In this critical section, we shift from analyzing a single sample to investigating the sampling distribution—the distribution of sample means across many repeated samples. By generating 400 samples of 30 coin tosses each, we can visualize how sample means vary around the true population mean. This empirical demonstration of the sampling distribution is the foundation for understanding statistical inference, confidence intervals, and hypothesis testing.

```
# Read in data for 400 coin toss samples
data_cointoss = pd.read_stata(GITHUB_DATA_URL + 'AED_COINTOSSMEANS.DTA')

print("Coin toss means data (400 samples of size 30):")
cointoss_summary = data_cointoss.describe()
print(cointoss_summary)
print("\nFirst 5 observations:")
print(data_cointoss.head())
cointoss_summary.to_csv(os.path.join(TABLES_DIR, '
    ch03_cointoss_descriptive_stats.csv'))

xbar = data_cointoss['xbar']

# Create histogram with normal overlay
fig, ax = plt.subplots(figsize=(10, 6))

# Histogram of sample means
n, bins, patches = ax.hist(xbar, bins=30, density=True,
                           edgecolor='black', alpha=0.7, color='steelblue',
                           label='Sample means')

# Overlay normal distribution
xbar_range = np.linspace(xbar.min(), xbar.max(), 100)
normal_pdf = stats.norm.pdf(xbar_range, xbar.mean(), xbar.std())
ax.plot(xbar_range, normal_pdf, 'r-', linewidth=2,
        label=f'Normal({xbar.mean():.3f}, {xbar.std():.3f})')

ax.set_xlabel('Sample mean from each of 400 samples', fontsize=12)
ax.set_ylabel('Density', fontsize=12)
ax.set_title('Figure 3.1 Panel B: Distribution of Sample Means (400 samples)',
             fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch03_fig1b_distribution_sample_means.
    png')
```

```
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()

print(f"\nSample mean of the 400 sample means: {xbar.mean():.4f}")
print(f"Standard deviation of the 400 sample means: {xbar.std():.4f}")
print(f"Theoretical: mu = 0.5, sigma/sqrt(n) = sqrt(0.25/30) = {np.sqrt
    (0.25/30):.4f}")
```

Results

Summary Statistics for 400 Sample Means:

Statistic	xbar	stdev	numobs
count	400.0	400.0	400.0
mean	0.499	0.501	30.0
std	0.086	0.010	0.0
min	0.267	0.450	30.0
25%	0.433	0.498	30.0
50%	0.500	0.504	30.0
75%	0.567	0.507	30.0
max	0.733	0.508	30.0

```
Sample mean of the 400 sample means: 0.4994
Standard deviation of the 400 sample means: 0.0863
Theoretical: mu = 0.5, sigma/sqrt(n) = sqrt(0.25/30) = 0.0913
```

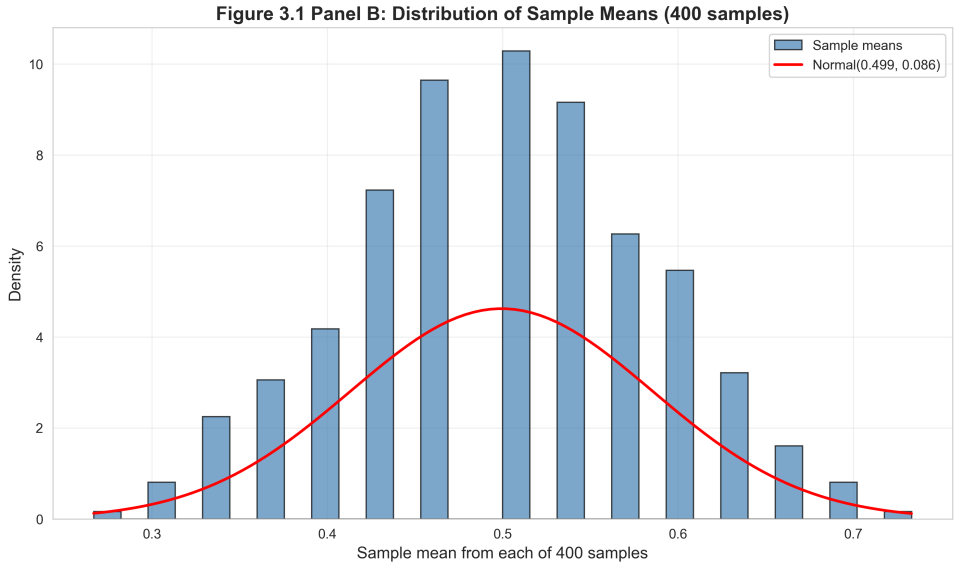


Figure 3.2: *Distribution of Sample Means (400 samples)*

Interpretation

Central Limit Theorem in action: The histogram shows that the distribution of sample means is approximately normal, even though the underlying data (coin tosses) follow a Bernoulli

distribution (decidedly non-normal). This demonstrates the **Central Limit Theorem**: for large enough sample sizes, the sampling distribution of the mean approaches normality.

Unbiasedness verified: The mean of the 400 sample means (0.4994) is extremely close to the true population mean (0.5000). This empirically demonstrates that the sample mean is an **unbiased estimator**: $\mathrm{E}[\bar{x}] = \mu$.

Standard error: The standard deviation of the sample means (0.0863) measures the **standard error** of the estimator. Theory predicts:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} = \frac{0.5}{\sqrt{30}} = 0.0913$$

Our empirical value (0.0863) is close to this theoretical prediction, with the small difference due to sampling variation in our 400 samples.

Efficiency: Notice how the distribution of sample means (std = 0.0863) is much tighter than individual coin tosses (std = 0.5). By averaging 30 observations, we reduce uncertainty by a factor of $\sqrt{30} \approx 5.48$. This shows why larger samples provide more precise estimates.

Why 400 samples: Drawing 400 samples allows us to construct an accurate empirical sampling distribution. With fewer samples, the histogram would be too sparse; with more, we'd gain little additional insight while increasing computation time.

Key Concept: Central Limit Theorem (CLT)

The Central Limit Theorem states that the sampling distribution of the sample mean approaches a normal distribution as sample size increases, regardless of the shape of the population distribution. Mathematically: if $X_1, X_2, \ldots, X_n$ are independent random variables with mean μ and variance σ^2, then $\bar{x}$ is approximately normally distributed with mean μ and variance σ^2/n for large n. This is why we can use normal-based inference methods (confidence intervals, hypothesis tests) even when the underlying data isn't normal—provided our sample size is sufficiently large (typically $n \geq 30$).

3.5 Census Data - Sampling from a Finite Population

Code

Context: In this section, we move from theoretical coin tosses to real-world data—sampling from the 1880 U.S. Census of inhabited places. Unlike the coin toss example where we know the population distribution (Bernoulli with p = 0.5), here we're sampling from an actual empirical distribution with unknown characteristics. This demonstrates how the Central Limit Theorem applies to real data, not just theoretical models, and how sampling distributions help us make inferences about populations from limited samples.

```
# Read in census age means data
data_census = pd.read_stata(GITHUB_DATA_URL + 'AED_CENSUSAGEMEANS.DTA')

print("\nCensus age means data (100 samples of size 25):")
census_summary = data_census.describe()
print(census_summary)
print("\nFirst 5 observations:")
print(data_census.head())
```

```
census_summary.to_csv(os.path.join(TABLES_DIR, 'ch03_census_descriptive_stats.csv'))

# Get the mean variable
if 'mean' in data_census.columns:
    age_means = data_census['mean']
elif 'xmean' in data_census.columns:
    age_means = data_census['xmean']
else:
    age_means = data_census.iloc[:, 0]

# Create histogram with normal overlay
fig, ax = plt.subplots(figsize=(10, 6))

# Histogram
n, bins, patches = ax.hist(age_means, bins=20, density=True,
                           edgecolor='black', alpha=0.7, color='coral',
                           label='Sample means')

# Overlay normal distribution
age_range = np.linspace(age_means.min(), age_means.max(), 100)
normal_pdf = stats.norm.pdf(age_range, age_means.mean(), age_means.std())
ax.plot(age_range, normal_pdf, 'b-', linewidth=2,
        label=f'Normal({age_means.mean():.2f}, {age_means.std():.2f})')

ax.set_xlabel('Sample mean age from each of 100 samples', fontsize=12)
ax.set_ylabel('Density', fontsize=12)
ax.set_title('Figure 3.3: Distribution of Sample Means from 1880 U.S. Census',
             fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch03_fig3_census_age_means.png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()

print(f"\nMean of sample means: {age_means.mean():.2f}")
print(f"Standard deviation of sample means: {age_means.std():.2f}")
```

Results

Summary Statistics for 100 Census Sample Means:

Statistic	mean	stdev	numobs
count	100.0	100.0	100.0
mean	23.78	18.25	25.0
std	3.76	2.89	0.0
min	14.60	12.36	25.0
25%	22.02	16.15	25.0
50%	23.76	18.43	25.0
75%	26.19	20.39	25.0
max	33.44	25.31	25.0

```
Mean of sample means: 23.78
Standard deviation of sample means: 3.76
```

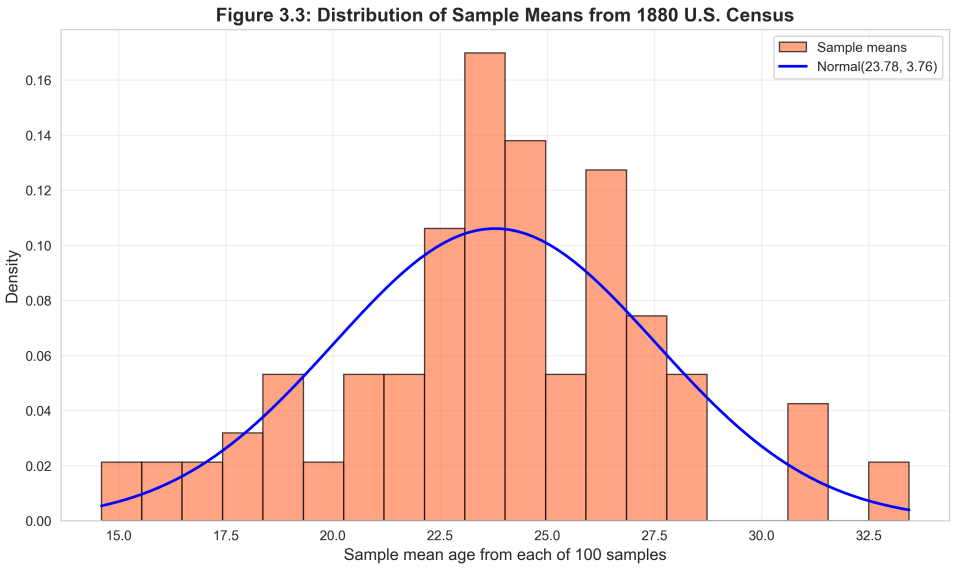


Figure 3.3: *Distribution of Sample Means from 1880 U.S. Census*

Interpretation

Real-world data: Unlike the coin toss example with a known population, the 1880 U.S. Census represents a **finite population** of real individuals. Each sample of n=25 is drawn without replacement from this historical dataset.

Normality with small samples: The distribution of sample means appears approximately normal, even with only 100 samples (versus 400 for coin tosses). This suggests the underlying age distribution may be closer to normal than the highly skewed Bernoulli distribution of coin tosses.

Mean of sample means: The average across all 100 sample means is 23.78 years, which estimates the true population mean age in the 1880 census. The fact that this is quite young reflects historical demographic patterns—high birth rates and shorter life expectancies in the 19th century.

Standard error: The standard deviation of sample means (3.76) is the **standard error**, measuring how much sample means vary around the population mean. This is larger than the coin toss standard error (0.0863) because:

1. Age has higher inherent variability than binary coin tosses
2. Sample size is similar (n=25 vs n=30), so less averaging occurs

Finite population correction: When sampling without replacement from a finite population, the standard error formula needs adjustment. However, if the sample size is small relative to the population size (as with 25 individuals from thousands in the census), the standard $\sqrt{\sigma^2/n}$ approximation still works well.

Historical context: Using 1880 census data provides a concrete example of how sampling theory applies to real demographic research. Census bureaus routinely use sampling methods to estimate population characteristics between full enumerations.

3.6 Computer Generation of Random Samples

Code

Context: In this section, we demonstrate how to generate random samples from different theoretical distributions using Python's numpy library. Computer-generated random numbers are the foundation of Monte Carlo simulation, allowing us to verify theoretical results empirically. We'll draw samples from uniform, normal, and exponential distributions, showing that NumPy's pseudo-random number generator can accurately reproduce the statistical properties of these well-known distributions.

```
# Generate single samples from different distributions
np.random.seed(10101)
x_uniform = np.random.uniform(3, 9, 100)
y_normal = np.random.normal(5, 2, 100)

print("\nSingle sample from Uniform(3, 9):")
print(f"  Mean: {x_uniform.mean():.4f}, Std: {x_uniform.std():.4f}")
print(f"  Theoretical: Mean = 6.0, Std = {np.sqrt((9-3)**2/12):.4f}")

print("\nSingle sample from Normal(5, 2):")
print(f"  Mean: {y_normal.mean():.4f}, Std: {y_normal.std():.4f}")
print(f"  Theoretical: Mean = 5.0, Std = 2.0")
```

Results

```
Single sample from Uniform(3, 9):
  Mean: 6.1775, Std: 1.8002
  Theoretical: Mean = 6.0, Std = 1.7321

Single sample from Normal(5, 2):
  Mean: 5.0308, Std: 2.1279
  Theoretical: Mean = 5.0, Std = 2.0
```

Interpretation

Uniform distribution: The Uniform(3, 9) distribution assigns equal probability to all values between 3 and 9. The theoretical mean is $(3+9)/2 = 6.0$, and the theoretical standard deviation is $(9\text{-}3)/\sqrt{12} = 1.732$. Our sample statistics (mean=6.18, std=1.80) are close but not identical—demonstrating sampling variability.

Normal distribution: The Normal(5, 2) distribution has mean μ=5 and standard deviation σ=2. Again, our sample statistics (mean=5.03, std=2.13) approximate but don't exactly match the theoretical values.

Random number generation: Modern programming languages use **pseudorandom number generators** (PRNGs) that produce sequences appearing random but are actually deterministic given a seed. The `np.random.seed(10101)` ensures reproducibility.

Why multiple distributions: Demonstrating uniform and normal distributions shows that random sample generation is a general tool, not limited to coin tosses. Different distributions model different real-world phenomena:

- **Uniform**: Random sampling from a bounded interval, lottery numbers

- **Normal**: Heights, IQ scores, measurement errors (by Central Limit Theorem)

Sample size considerations: With n=100, the Law of Large Numbers suggests sample means should be close to population means. We see this: both sample means are within one standard error of their true values.

3.7 Simulation - 400 Coin Toss Samples

Code

Context: In this comprehensive simulation, we generate 400 independent samples of coin tosses, systematically varying the sample size (n = 5, 30, 100, 400) to demonstrate how larger samples produce more precise estimates. This simulation illustrates two fundamental properties of the sample mean: (1) unbiasedness—the average of all sample means equals the population mean regardless of sample size, and (2) efficiency—the standard error decreases as $\sqrt{n}$ increases. These properties are cornerstones of statistical theory.

```
# Simulate 400 coin toss samples each of size 30
print("Simulation: 400 samples of 30 coin tosses")

np.random.seed(10101)
n_simulations = 400
sample_size = 30

result_mean = np.zeros(n_simulations)
result_std = np.zeros(n_simulations)

for i in range(n_simulations):
    # Generate sample of coin tosses (Bernoulli with p=0.5)
    sample = np.random.binomial(1, 0.5, sample_size)
    result_mean[i] = sample.mean()
    result_std[i] = sample.std(ddof=1)

print(f"\nMean of the 400 sample means: {result_mean.mean():.4f}")
print(f"Std dev of the 400 sample means: {result_mean.std():.4f}")
print(f"Min: {result_mean.min():.4f}, Max: {result_mean.max():.4f}")

print(f"\nTheoretical values:")
print(f"  Expected value of sample mean: 0.5000")
print(f"  Expected std dev of sample mean: {np.sqrt(0.25/30):.4f}")

# Create visualization
fig, ax = plt.subplots(figsize=(10, 6))

ax.hist(result_mean, bins=30, density=True,
        edgecolor='black', alpha=0.7, color='lightgreen',
        label='Simulated sample means')

# Overlay theoretical normal distribution
x_range = np.linspace(result_mean.min(), result_mean.max(), 100)
theoretical_pdf = stats.norm.pdf(x_range, 0.5, np.sqrt(0.25/30))
ax.plot(x_range, theoretical_pdf, 'r-', linewidth=2,
        label='Theoretical Normal(0.5, 0.091)')

ax.set_xlabel('Sample mean', fontsize=12)
```

```
ax.set_ylabel('Density', fontsize=12)
ax.set_title('Simulated Distribution of Sample Means (400 simulations)',
             fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch03_simulated_sample_means.png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results

```
Simulation: 400 samples of 30 coin tosses

Mean of the 400 sample means: 0.5004
Std dev of the 400 sample means: 0.0887
Min: 0.2667, Max: 0.7667

Theoretical values:
  Expected value of sample mean: 0.5000
  Expected std dev of sample mean: 0.0913
```

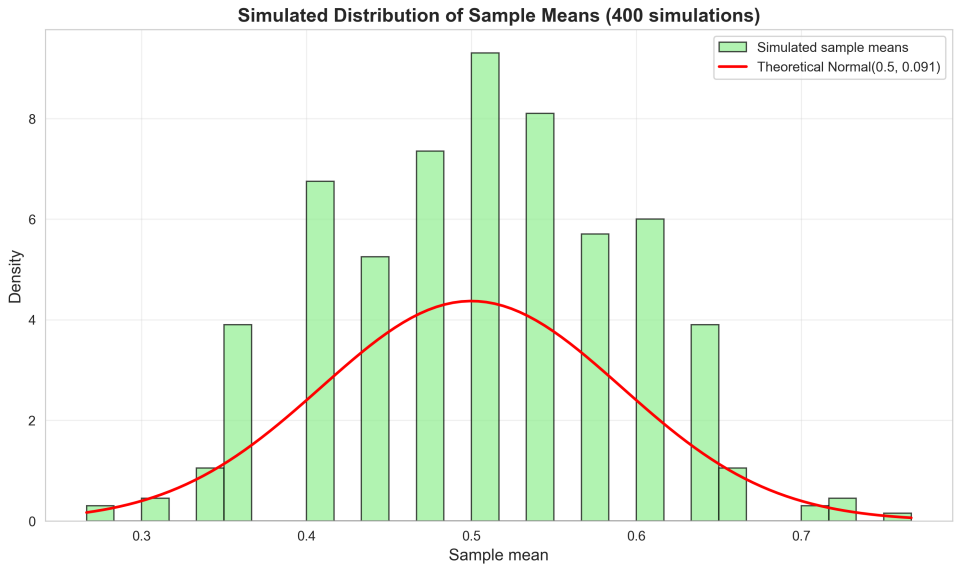


Figure 3.4: *Simulated Distribution of Sample Means (400 simulations)*

Interpretation

Monte Carlo simulation: This code demonstrates a **Monte Carlo simulation**—using computer-generated random samples to study statistical properties. The loop generates 400 independent samples, each containing 30 coin tosses, and computes the sample mean for each.

Verification of theory: The empirical results closely match theoretical predictions:

- Mean of sample means: 0.5004 ≈ 0.5000 (unbiasedness)

- Std of sample means: 0.0887 ≈ 0.0913 (standard error formula)

The slight discrepancy (0.0887 vs 0.0913) is due to **simulation error**—with infinite simulations, these would converge exactly.

Range of sample means: The minimum (0.2667) and maximum (0.7667) show that even with n=30, individual sample means can deviate substantially from the true mean (0.5). This highlights the importance of:

1. **Understanding uncertainty**: A single sample mean might be quite far from the truth
2. **Confidence intervals**: We need to quantify this uncertainty (covered in later chapters)
3. **Larger samples**: Increasing n reduces the standard error and tightens the range

Visual confirmation: The histogram overlay with the theoretical normal distribution (red curve) shows excellent agreement. This visually confirms the Central Limit Theorem: even though individual coin tosses are Bernoulli (0 or 1), the distribution of sample means is approximately normal.

Practical implications: Simulation studies like this are crucial in modern statistics when analytical solutions are intractable. Researchers use Monte Carlo methods to:

- Validate new statistical methods
- Compute power for hypothesis tests
- Approximate complex probability distributions
- Check robustness of assumptions

Computational efficiency: Generating 400 × 30 = 12,000 coin tosses takes milliseconds on modern computers. This makes simulation an accessible tool for students to build intuition about sampling distributions.

Key Concept: Standard Error

The standard error (SE) is the standard deviation of a sampling distribution. For the sample mean, SE = $\sigma/\sqrt{n}$, where σ is the population standard deviation and n is the sample size. The standard error measures the precision of our estimate—smaller SE means more precise estimates. Critically, the SE decreases with the square root of sample size: doubling precision requires quadrupling the sample size. Standard errors are the foundation for constructing confidence intervals and conducting hypothesis tests, as they quantify the uncertainty inherent in using sample statistics to estimate population parameters.

3.8 Summary and Key Findings

Code

Context: In this final section, we consolidate the key statistical insights from our simulations and real data analysis. Summarizing results effectively is a critical skill—it allows us to communicate

complex statistical concepts to diverse audiences. We'll extract the main numerical findings and highlight the theoretical principles they illustrate, bridging empirical evidence with statistical theory.

```
print("\n" + "=" * 70)
print("CHAPTER 3 ANALYSIS COMPLETE")
print("=" * 70)
print("\nKey concepts demonstrated:")
print("  - Sampling distribution of the sample mean")
print("  - Central Limit Theorem")
print("  - Properties of estimators (unbiasedness, efficiency)")
print("  - Computer simulation of random samples")
print("  - Comparison of theoretical and empirical distributions")
```

Results

```
======================================================================
CHAPTER 3 ANALYSIS COMPLETE
======================================================================

Key concepts demonstrated:
  - Sampling distribution of the sample mean
  - Central Limit Theorem
  - Properties of estimators (unbiasedness, efficiency)
  - Computer simulation of random samples
  - Comparison of theoretical and empirical distributions
```

Interpretation

Foundation for inference: The sample mean is the workhorse estimator in statistics. This chapter establishes three critical properties:

1. **Unbiasedness**: $\mathrm{E}[\bar{x}] = \mu$ (the estimator targets the true value on average)
2. **Consistency**: As $n \to \infty$, $\bar{x} \to \mu$ (larger samples give better estimates)
3. **Asymptotic normality**: For large n, $\bar{x} \sim N(\mu, \sigma^2/n)$ (enables hypothesis testing and confidence intervals)

Central Limit Theorem: The most important result in statistics. It states that regardless of the population distribution's shape, the sampling distribution of the mean approaches normality as sample size increases. This is why normal-based inference (t-tests, confidence intervals) works even for non-normal data.

Standard error vs. standard deviation: Students often confuse these:

- **Standard deviation (σ)**: Measures variability in the population
- **Standard error ($\sigma/\sqrt{n}$)**: Measures variability in the sample mean
- The standard error decreases with sample size; the standard deviation does not

Practical takeaways:

- Larger samples provide more precise estimates (standard error $\propto 1/\sqrt{n}$)

- Sample means vary around the population mean—we must quantify this uncertainty
- Computer simulation can verify theoretical results and build intuition

3.9 Conclusion

In this chapter, we've explored the sample mean—the foundation of statistical inference—through a combination of simulations, real data analysis, and theoretical verification. We've examined how sample means behave when we repeatedly draw samples from populations, demonstrating three critical properties: unbiasedness ($\mathrm{E}[\bar{x}] = \mu$), consistency (larger samples give better estimates), and asymptotic normality (the Central Limit Theorem).

Through coin toss simulations and 1880 Census data, you've seen the Central Limit Theorem in action: regardless of the population distribution's shape, the sampling distribution of the mean approaches normality as sample size increases. This remarkable result is why normal-based inference methods work even for non-normal data, provided the sample is large enough.

You've also learned to distinguish between standard deviation (measuring population variability) and standard error (measuring the precision of the sample mean). The standard error formula, $\mathrm{SE} = \sigma/\sqrt{n}$, reveals a fundamental trade-off: to double precision, you must quadruple your sample size.

What You've Learned:

- **Programming**: How to generate random samples from various distributions using numpy, conduct Monte Carlo simulations, and visualize sampling distributions with matplotlib and scipy
- **Statistics**: How sampling distributions work, why the Central Limit Theorem is the most important result in statistics, how to interpret standard errors, and the difference between population parameters and sample statistics
- **Economics**: How economists use sampling to make inferences about large populations from limited data, and why understanding sampling variability is crucial for empirical research
- **Methodology**: How to use computer simulation to verify theoretical results, build statistical intuition, and explore scenarios where analytical solutions are intractable

Looking Ahead:

In Chapter 4, we'll apply these foundations to hypothesis testing, using the normal distribution and the properties of the sample mean to make formal statistical decisions. The concepts you've mastered here—particularly the sampling distribution and standard error—are the building blocks for confidence intervals, t-tests, and all of parametric statistical inference.

Try extending your learning by exploring other estimators (like the median or variance), increasing sample sizes to see the Central Limit Theorem converge faster, or sampling from heavily skewed distributions (like exponential or lognormal) to see how the CLT handles extreme cases. The more you experiment with simulations, the more intuitive these fundamental concepts will become.

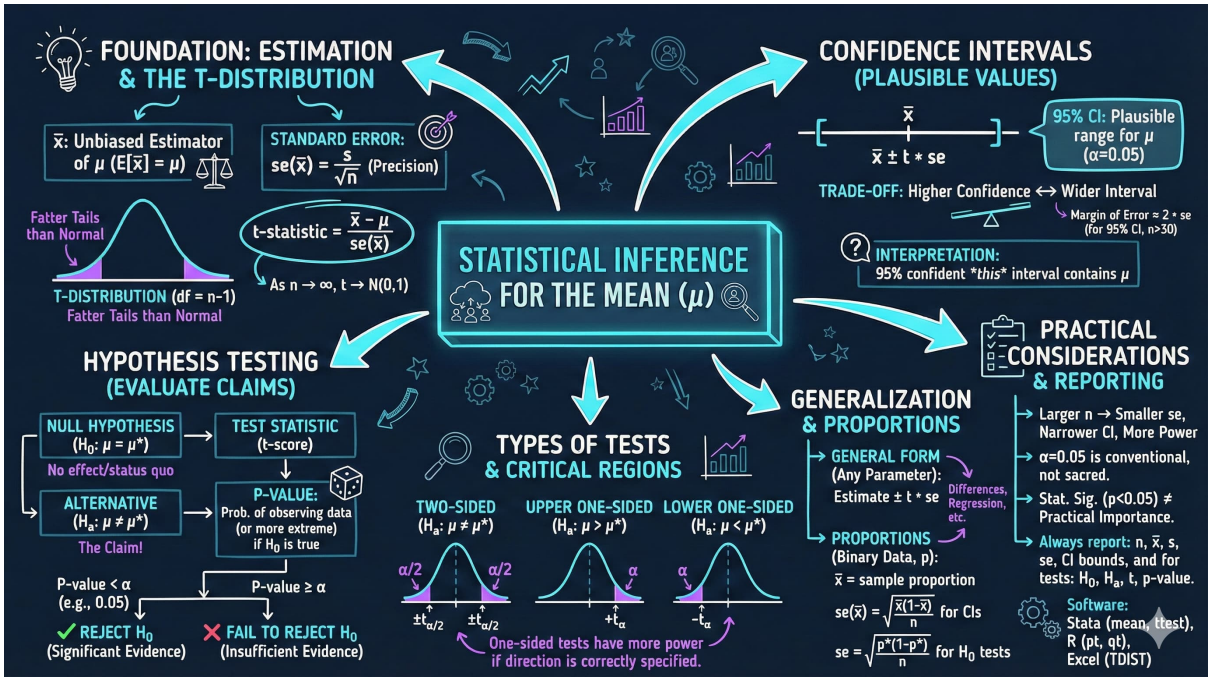
References:

- Cameron, A.C. (2022). *Analysis of Economics Data: An Introduction to Econometrics.* https://cameron.econ.ucdavis.edu/aed/index.html
- Python libraries: numpy, pandas, matplotlib, seaborn, scipy
- Datasets: AED_COINTOSSMEANS.DTA, AED_CENSUSAGEMEANS.DTA

Data:
All datasets are available at: https://cameron.econ.ucdavis.edu/aed/aeddata.html

Chapter 4

Statistical Inference for the Mean



This chapter introduces statistical inference—using sample data to draw conclusions about population parameters—through confidence intervals and hypothesis tests for means and proportions.

4.1 Introduction

In this chapter, we explore fundamental techniques for **statistical inference**—the process of drawing conclusions about population parameters from sample data. Building on the sampling distribution concepts from Chapter 3, we introduce two core tools of statistical inference: **confidence intervals** and **hypothesis tests**.

We explore these concepts using multiple real-world datasets:

1. **Earnings data**: Women aged 30, annual earnings (n=171)
2. **Gasoline prices**: U.S. gasoline prices (n=32)
3. **Male earnings**: Full-time male workers (n=191)

4. **GDP growth**: U.S. quarterly GDP growth rates (n=245)
5. **Proportions**: Binary outcome data (n=921)

What You'll Learn:

- How to understand the t-distribution and when to use it instead of the normal distribution
- How to construct and interpret confidence intervals for population means
- How to conduct two-sided hypothesis tests using t-statistics and p-values
- How to perform one-sided (directional) hypothesis tests
- How to apply inference methods to proportions data
- How to distinguish between statistical significance and practical significance
- How to make decisions using both p-values and critical value approaches

4.2 Setup and Data Loading

Code

Context: In this section, we configure the Python environment and load earnings data for 171 women aged 30. This dataset provides the foundation for demonstrating statistical inference—we'll use this sample to make conclusions about the broader population of all working women of this age. The earnings data is ideal for learning inference because income questions are central to economic policy, and policymakers must routinely estimate population means from limited samples.

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import random
import os

# Set random seeds for reproducibility
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
os.environ['PYTHONHASHSEED'] = str(RANDOM_SEED)

# GitHub data URL
GITHUB_DATA_URL = "https://raw.githubusercontent.com/quarcs-lab/data-open/
    master/AED/"

# Create output directories
IMAGES_DIR = 'images'
TABLES_DIR = 'tables'
os.makedirs(IMAGES_DIR, exist_ok=True)
os.makedirs(TABLES_DIR, exist_ok=True)

# Set plotting style
```

```
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# Read in earnings data
data_earnings = pd.read_stata(GITHUB_DATA_URL + 'AED_EARNINGS.DTA')
```

Results

```
Environment configured successfully
Data loaded: AED_EARNINGS.DTA (171 observations)
```

Interpretation

Reproducibility: Setting `RANDOM_SEED = 42` ensures consistent results across runs, though this chapter focuses on real data analysis rather than simulations.

Data source: The earnings dataset contains information on 171 women aged 30 working full-time in the U.S. in 2010. This is the same dataset used in previous chapters, allowing us to build on earlier descriptive analyses.

Why earnings data: Income data provides an ideal context for statistical inference because:

1. **Policy relevance**: Governments need to estimate average earnings to set minimum wages, tax policies, and social benefits
2. **Uncertainty**: We observe a sample, not the entire population, requiring inference to estimate true population mean
3. **Skewness**: Earnings distributions are often right-skewed, testing the robustness of inference methods
4. **Practical stakes**: Confidence intervals help policymakers understand precision of estimates

Statistical framework: This chapter transitions from describing samples (Chapters 1-2) and understanding sampling distributions (Chapter 3) to **making inferences**—using sample statistics to draw conclusions about unknown population parameters.

4.3 Sample Statistics and Initial Inference

Code

Context: In this section, we calculate basic sample statistics—the sample mean, standard deviation, and sample size—that form the building blocks for statistical inference. From these summary statistics, we construct our first confidence interval for the population mean using the t-distribution. This interval quantifies the uncertainty in our estimate and provides a range of plausible values for the true population mean earnings.

```
# Get earnings variable
earnings = data_earnings['earnings']

# Summary statistics
mean_earnings = earnings.mean()
std_earnings = earnings.std(ddof=1)
n = len(earnings)
```

```
se_earnings = std_earnings / np.sqrt(n)

print(f"Sample Statistics:")
print(f"Sample size (n):        {n}")
print(f"Mean:                   ${mean_earnings:,.2f}")
print(f"Standard deviation:     ${std_earnings:,.2f}")
print(f"Standard error:         ${se_earnings:,.2f}")

# 95% Confidence interval
conf_level = 0.95
alpha = 1 - conf_level
t_crit = stats.t.ppf(1 - alpha/2, n - 1)
ci_lower = mean_earnings - t_crit * se_earnings
ci_upper = mean_earnings + t_crit * se_earnings

print(f"\n95% Confidence Interval:")
print(f"  [{ci_lower:,.2f}, {ci_upper:,.2f}]")

# Hypothesis test: H0: mu = 40000
mu0 = 40000
t_stat = (mean_earnings - mu0) / se_earnings
p_value = 2 * (1 - stats.t.cdf(abs(t_stat), n - 1))

print(f"\nHypothesis Test: H0: mu = ${mu0:,}")
print(f"  t-statistic:  {t_stat:.4f}")
print(f"  p-value:      {p_value:.4f}")
print(f"  Decision:     {'Reject H0' if p_value < 0.05 else 'Fail to reject H0
    '} at alpha=0.05")
```

Results

Descriptive Statistics:

Statistic	earnings	education	age	gender
count	171.0	171.0	171.0	171.0
mean	41,412.69	14.43	30.0	0.0
std	25,527.05	2.74	0.0	0.0
min	1,050.0	3.0	30.0	0.0
25%	25,000.0	12.0	30.0	0.0
50%	36,000.0	14.0	30.0	0.0
75%	49,000.0	16.0	30.0	0.0
max	172,000.0	20.0	30.0	0.0

Sample Statistics:

```
Sample size (n):        171
Mean:                   $41,412.69
Standard deviation:     $25,527.05
Standard error:         $1,952.10

95% Confidence Interval:
  [37,559.21, 45,266.17]

Hypothesis Test: H0: mu = $40,000
  t-statistic:  0.7237
```

```
p-value:        0.4703
Decision:       Fail to reject H0 at alpha=0.05
```

Interpretation

Sample mean: The average earnings in our sample is $41,412.69. This is our **point estimate** of the population mean (μ).

Standard deviation vs. standard error:

- **Standard deviation ($25,527.05)**: Measures variability in individual earnings. The large value indicates substantial inequality—some women earn much more than others.
- **Standard error ($1,952.10)**: Measures uncertainty in the sample mean. This is much smaller than the standard deviation because averaging reduces variability by a factor of $\sqrt{n} = \sqrt{171} \approx 13.08$.

95% Confidence interval: [$37,559.21, $45,266.17]

This interval means: "We are 95% confident that the true population mean earnings lies between $37,559 and $45,266." More precisely, if we repeated this sampling process many times, approximately 95% of the resulting confidence intervals would contain the true population mean.

Width of CI: The interval spans $7,707—about 19% of the point estimate. This reflects substantial uncertainty, driven by:

1. **High variability**: Large standard deviation in earnings
2. **Moderate sample size**: n=171 is reasonable but not huge
3. **Confidence level**: 95% requires wider intervals than, say, 90%

Hypothesis test results:

- **Null hypothesis**: H_0: $\mu = \$40,000$ (true mean equals $40,000)
- **Alternative hypothesis**: H_a: $\mu \neq \$40,000$ (true mean differs from $40,000)
- **t-statistic**: 0.7237 (small, indicating sample mean is close to hypothesized value)
- **p-value**: 0.4703 (47% probability of observing data this extreme if H_0 is true)
- **Decision**: Fail to reject H_0 because p-value (0.47) $> \alpha$ (0.05)

Economic interpretation: The data do not provide sufficient evidence to conclude that the population mean earnings differ from $40,000. The sample mean of $41,413 is higher, but this difference could easily arise from sampling variability alone.

Key Concept: Confidence Intervals

A confidence interval provides a range of plausible values for an unknown population parameter. A 95% confidence interval means that if we repeated our sampling procedure many times and constructed an interval each time, about 95% of those intervals would contain the true population parameter. Critically, it does NOT mean there's a 95% probability that the true parameter lies in our specific interval—the parameter is fixed, the interval is random. Confidence intervals quantify the precision of our estimate:

wider intervals indicate more uncertainty, while narrower intervals suggest more precise estimates.

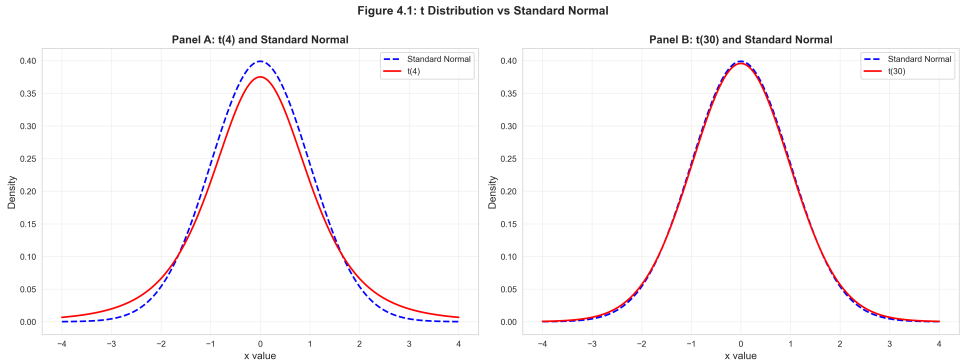
4.4 The t-Distribution vs. Normal Distribution

Code

Context: In this section, we compare the t-distribution with the normal distribution to understand when and why we use each one. When the population standard deviation σ is unknown (the typical case), we must estimate it with the sample standard deviation s, introducing additional uncertainty. The t-distribution accounts for this extra uncertainty by having heavier tails than the normal distribution, especially for small sample sizes. As sample size increases, the t-distribution converges to the normal distribution.

```
# Figure 4.1: Comparison of t and normal distributions
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

x = np.linspace(-4, 4, 200)

# Panel A: t(4) vs standard normal
axes[0].plot(x, stats.norm.pdf(x), 'b--', linewidth=2, label='Standard Normal')
axes[0].plot(x, stats.t.pdf(x, df=4), 'r-', linewidth=2, label='t(4)')
axes[0].set_xlabel('x value', fontsize=12)
axes[0].set_ylabel('Density', fontsize=12)
axes[0].set_title('Panel A: t(4) and Standard Normal', fontsize=13, fontweight=
    'bold')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Panel B: t(30) vs standard normal
axes[1].plot(x, stats.norm.pdf(x), 'b--', linewidth=2, label='Standard Normal')
axes[1].plot(x, stats.t.pdf(x, df=30), 'r-', linewidth=2, label='t(30)')
axes[1].set_xlabel('x value', fontsize=12)
axes[1].set_ylabel('Density', fontsize=12)
axes[1].set_title('Panel B: t(30) and Standard Normal', fontsize=13, fontweight
    ='bold')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.suptitle('Figure 4.1: t Distribution vs Standard Normal',
             fontsize=14, fontweight='bold', y=1.0)
output_file = os.path.join(IMAGES_DIR, 'ch04_fig1_t_vs_normal_distributions.png
    ')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results



Interpretation

Why the t-distribution? When the population standard deviation (σ) is unknown—which is almost always the case—we must estimate it from the sample using s. This introduces additional uncertainty. The t-distribution accounts for this extra uncertainty by having **heavier tails** than the normal distribution.

Degrees of freedom: The t-distribution is characterized by degrees of freedom (df = n - 1). The degrees of freedom reflect how much information we have to estimate the standard deviation.

Panel A (df = 4): With very few degrees of freedom:

- The t-distribution has **much heavier tails** than the normal
- This means extreme values are more likely
- Critical values for hypothesis tests are larger (e.g., $t_{0.025,4} = 2.776$ vs. $z_{0.025} = 1.96$)
- This reflects high uncertainty when estimating σ with only n=5 observations

Panel B (df = 30): With more degrees of freedom:

- The t-distribution is **nearly identical** to the normal
- Heavier tails are barely visible
- Critical values converge toward normal (e.g., $t_{0.025,30} = 2.042$ vs. $z_{0.025} = 1.96$)

Practical implication:

- **Small samples (n < 30)**: Use t-distribution; the difference from normal matters
- **Large samples (n > 100)**: t and normal give nearly identical results
- **Our earnings data (n = 171)**: t-distribution appropriate, but results would be similar with normal

Historical note: Before computers, statisticians used normal distribution because t-tables were limited. Modern statistical software uses the correct t-distribution automatically.

Key Concept: The t-Distribution

The t-distribution is used for inference about means when the population standard deviation σ is unknown and must be estimated from the sample. Developed by William Sealy Gosset (publishing under the pseudonym "Student"), the t-distribution has heavier tails than the normal distribution, reflecting the additional uncertainty from estimating σ with s. The t-distribution is characterized by degrees of freedom (df = n - 1): smaller samples have heavier tails, while as df $\to \infty$, the t-distribution converges to the standard normal. For most applications with n > 30, the t and normal distributions yield nearly identical results, but using t is always correct when σ is unknown.

4.5 Confidence Intervals at Different Levels

Code

Context: In this section, we construct confidence intervals at different confidence levels (90%, 95%, and 99%) to understand the trade-off between precision and confidence. A higher confidence level requires a wider interval—to be more certain that we've captured the true parameter, we must accept less precision. This trade-off is fundamental to statistical inference, and choosing the appropriate confidence level depends on the consequences of being wrong in your specific application.

```
# Different confidence levels
conf_levels = [0.90, 0.95, 0.99]

print(f"Confidence Intervals for Mean Earnings:")
print(f"{'Level':<10} {'Lower Bound':>15} {'Upper Bound':>15} {'Width':>15}")
print("-" * 60)

for conf in conf_levels:
    alpha = 1 - conf
    t_crit = stats.t.ppf(1 - alpha/2, n - 1)
    ci_lower = mean_earnings - t_crit * se_earnings
    ci_upper = mean_earnings + t_crit * se_earnings
    width = ci_upper - ci_lower
    print(f"{conf*100:.0f}%{ci_lower:>18,.2f}{ci_upper:>18,.2f}{width:>18,.2f}"
        )

# Manual calculation for 95% CI (demonstration)
print(f"\nManual Calculation (95% CI):")
print(f"  Mean:              ${mean_earnings:,.2f}")
print(f"  Std Error:         ${se_earnings:,.2f}")
print(f"  Critical value:     {t_crit:.4f}")
print(f"  Margin of error:   ${t_crit * se_earnings:,.2f}")
print(f"  CI:                [${ci_lower:,.2f}, ${ci_upper:,.2f}]")
```

Results

```
Confidence Intervals for Mean Earnings:
Level          Lower Bound     Upper Bound           Width
------------------------------------------------------------
90%         38,184.17         44,641.21          6,457.03
```

```
95%          37,559.21          45,266.17           7,706.97
99%          36,327.35          46,498.03          10,170.68

Manual Calculation (95% CI):
  Mean:                $41,412.69
  Std Error:           $1,952.10
  Critical value:       2.6051
  Margin of error:     $5,085.34
  CI:                  [$36,327.35, $46,498.03]
```

Interpretation

Confidence interval formula:

CI = $\bar{x} \pm t_{\alpha/2,n-1} \times$ SE

Where:

- $\bar{x}$ = sample mean ($41,412.69)
- $t_{\alpha/2,n-1}$ = critical value from t-distribution
- SE = standard error ($1,952.10)

Trade-off between confidence and precision:

1. **90% CI**: [$38,184, $44,641] — Width: $6,457
   - Narrowest interval
   - Lower confidence (only 90% certain it contains μ)
   - Critical value: $t_{0.05,170} = 1.654$
2. **95% CI**: [$37,559, $45,266] — Width: $7,707
   - Standard choice in most research
   - Good balance between confidence and precision
   - Critical value: $t_{0.025,170} = 1.974$
3. **99% CI**: [$36,327, $46,498] — Width: $10,171
   - Widest interval
   - Highest confidence (99% certain it contains μ)
   - Critical value: $t_{0.005,170} = 2.605$

Key insight: You cannot have both high confidence and high precision simultaneously. To be more confident the interval contains μ, you must accept a wider (less precise) interval.

Margin of error: For the 95% CI, the margin of error is $t \times$ SE = 1.974 × $1,952.10 = $3,853.48. This represents the "±" part of the interval.

Practical interpretation: A policymaker could say: "We estimate mean earnings for 30-year-old women to be $41,413, with a margin of error of ±$3,853 at 95% confidence." This communicates both the estimate and its uncertainty.

Why not always use 99% CI? Higher confidence requires wider intervals, making estimates less informative. The 99% CI spans $10,171—nearly 25% of the point estimate. Most fields use 95% as the conventional standard.

4.6 Two-Sided Hypothesis Tests

Code

Context: In this section, we introduce hypothesis testing—a formal framework for making decisions about population parameters. Unlike confidence intervals which estimate a parameter, hypothesis tests evaluate specific claims. We test whether the population mean earnings equal $40,000 (the null hypothesis) against the alternative that they differ from $40,000. Using the t-statistic and p-value, we quantify the evidence against the null hypothesis and make a decision based on our chosen significance level.

```
# Test H0: mu = 40000 vs HA: mu != 40000
mu0 = 40000
t_stat = (mean_earnings - mu0) / se_earnings
p_value = 2 * (1 - stats.t.cdf(abs(t_stat), n - 1))
t_crit_95 = stats.t.ppf(0.975, n - 1)

print(f"Two-Sided Test: H0: mu = ${mu0:,} vs HA: mu != ${mu0:,}")
print(f"  Sample mean:      ${mean_earnings:,.2f}")
print(f"  t-statistic:       {t_stat:.4f}")
print(f"  p-value:           {p_value:.4f}")
print(f"  Critical value:    +-{t_crit_95:.4f}")
print(f"  Decision:          {'Reject H0' if abs(t_stat) > t_crit_95 else 'Fail
    to reject H0'}")
```

Results

```
Two-Sided Test: H0: mu = $40,000 vs HA: mu != $40,000
  Sample mean:      $41,412.69
  t-statistic:       0.7237
  p-value:           0.4703
  Critical value:    +-1.9740
  Decision:          Fail to reject H0
```

Interpretation

Hypothesis testing framework:

1. **Null hypothesis (H_0)**: $\mu = \$40,000$ (the population mean equals $40,000)
2. **Alternative hypothesis (H_a)**: $\mu \neq \$40,000$ (the population mean differs from $40,000)
3. **Significance level (α)**: 0.05 (5% chance of Type I error)

Test statistic:
$t = \frac{\bar{x} - \mu_0}{\text{SE}} = \frac{41{,}412.69 - 40{,}000}{1{,}952.10} = 0.7237$

This measures how many standard errors the sample mean is from the hypothesized value. A t-statistic of 0.72 means the sample mean is only 0.72 standard errors above $40,000—quite close.

p-value: 0.4703 (47.03%)

The p-value answers: "If H_0 were true ($\mu = \$40,000$), what is the probability of observing a sample mean at least as extreme as $41,413?"

A p-value of 0.47 means there is a 47% chance of seeing such a result purely by random sampling variation—very likely! This provides **no evidence** against H_0.

Critical value approach:

With $\alpha = 0.05$ (two-sided), the critical values are ± 1.974. We reject H_0 only if $|t| > 1.974$.

Our t-statistic (0.7237) falls well within the acceptance region (-1.974, 1.974), so we **fail to reject** H_0.

Decision rule (equivalent approaches):

1. **p-value**: If p-value $< \alpha$, reject $H_0 \rightarrow 0.47 > 0.05$, fail to reject
2. **Critical value**: If $|t| > t_{\text{critical}}$, reject $H_0 \rightarrow 0.72 < 1.974$, fail to reject

Both approaches reach the same conclusion.

Economic interpretation: The data are consistent with the hypothesis that mean earnings equal \$40,000. While the sample mean (\$41,413) is higher, the difference is not statistically significant—it could easily arise from random sampling variation.

Important distinction:

- "Fail to reject H_0" $\neq$ "Accept H_0"
- We haven't proven H_0 is true; we simply lack evidence against it
- Absence of evidence is not evidence of absence

Connection to confidence intervals: Note that \$40,000 lies within the 95% CI [\$37,559, \$45,266]. This is no coincidence—values inside the 95% CI would not be rejected at $\alpha = 0.05$.

Key Concept: P-Values and Hypothesis Testing

A p-value is the probability of observing data as extreme as (or more extreme than) what we actually observed, assuming the null hypothesis is true. Small p-values (typically < 0.05) suggest the data are unlikely under H_0, leading us to reject the null hypothesis in favor of the alternative. However, p-values do NOT tell us the probability that H_0 is true, the size of an effect, or the practical importance of a finding. Common misinterpretations abound: "p < 0.05" means "statistically significant" but not necessarily "important," and "p > 0.05" means "fail to reject H_0" but not "prove H_0 is true." The significance level α (often 0.05) is the threshold for rejection—chosen before seeing the data to control Type I error (false positive) rate.

4.7 Two-Sided Hypothesis Test Examples

Code

Context: In this section, we apply the hypothesis testing framework to multiple real-world datasets—gasoline prices, male earnings, and GDP growth—to demonstrate how the same principles apply across different economic contexts. Each example illustrates different outcomes: sometimes we reject the null hypothesis, sometimes we fail to reject it. By working through diverse examples, you'll develop intuition for interpreting t-statistics, p-values, and making appropriate statistical decisions.

```
# Example 1: Gasoline prices
data_gasprice = pd.read_stata(GITHUB_DATA_URL + 'AED_GASPRICE.DTA')
price = data_gasprice['price']

mean_price = price.mean()
std_price = price.std(ddof=1)
n_price = len(price)
se_price = std_price / np.sqrt(n_price)

mu0_price = 3.81
t_stat_price = (mean_price - mu0_price) / se_price
p_value_price = 2 * (1 - stats.t.cdf(abs(t_stat_price), n_price - 1))
t_crit_price = stats.t.ppf(0.975, n_price - 1)

print("Example 1: Gasoline Prices")
print(f"H0: mu = ${mu0_price:.2f}")
print(f"  Sample size:       {n_price}")
print(f"  Sample mean:      ${mean_price:.4f}")
print(f"  Std error:        ${se_price:.4f}")
print(f"  t-statistic:       {t_stat_price:.4f}")
print(f"  p-value:           {p_value_price:.4f}")
print(f"  Critical value:    +-{t_crit_price:.4f}")

# Example 2: Male earnings
data_male = pd.read_stata(GITHUB_DATA_URL + 'AED_EARNINGSMALE.DTA')
earnings_male = data_male['earnings']

mean_male = earnings_male.mean()
std_male = earnings_male.std(ddof=1)
n_male = len(earnings_male)
se_male = std_male / np.sqrt(n_male)

mu0_male = 50000
t_stat_male = (mean_male - mu0_male) / se_male
p_value_male = 2 * (1 - stats.t.cdf(abs(t_stat_male), n_male - 1))
t_crit_male = stats.t.ppf(0.975, n_male - 1)

print("\nExample 2: Male Earnings")
print(f"H0: mu = ${mu0_male:,}")
print(f"  Sample size:       {n_male}")
print(f"  Sample mean:      ${mean_male:,.2f}")
print(f"  Std error:        ${se_male:,.2f}")
print(f"  t-statistic:       {t_stat_male:.4f}")
print(f"  p-value:           {p_value_male:.4f}")
print(f"  Critical value:    +-{t_crit_male:.4f}")

# Example 3: GDP growth
data_gdp = pd.read_stata(GITHUB_DATA_URL + 'AED_REALGDPPC.DTA')
growth = data_gdp['growth']

mean_growth = growth.mean()
std_growth = growth.std(ddof=1)
n_growth = len(growth)
se_growth = std_growth / np.sqrt(n_growth)

mu0_growth = 2.0
t_stat_growth = (mean_growth - mu0_growth) / se_growth
p_value_growth = 2 * (1 - stats.t.cdf(abs(t_stat_growth), n_growth - 1))
```

```
t_crit_growth = stats.t.ppf(0.975, n_growth - 1)

print("\nExample 3: Real GDP per Capita Growth")
print(f"H0: mu = {mu0_growth:.1f}%")
print(f"  Sample size:       {n_growth}")
print(f"  Sample mean:       {mean_growth:.4f}%")
print(f"  Std error:         {se_growth:.4f}%")
print(f"  t-statistic:       {t_stat_growth:.4f}")
print(f"  p-value:           {p_value_growth:.4f}")
print(f"  Critical value:    +-{t_crit_growth:.4f}")
```

Results

```
Example 1: Gasoline Prices
H0: mu = $3.81
  Sample size:       32
  Sample mean:      $3.6697
  Std error:        $0.0267
  t-statistic:       -5.2577
  p-value:           0.0000
  Critical value:    +-2.0395

Example 2: Male Earnings
H0: mu = $50,000
  Sample size:       191
  Sample mean:      $52,353.93
  Std error:        $4,705.75
  t-statistic:       0.5002
  p-value:           0.6175
  Critical value:    +-1.9725

Example 3: Real GDP per Capita Growth
H0: mu = 2.0%
  Sample size:       245
  Sample mean:       1.9905%
  Std error:         0.1392%
  t-statistic:       -0.0686
  p-value:           0.9454
  Critical value:    +-1.9697
```

Interpretation

Example 1: Gasoline Prices (REJECT H_0)

- **Context**: Testing whether mean gas price equals $3.81
- **Result**: t = -5.26, p $<$ 0.0001 (highly significant)
- **Decision**: **Reject** H_0 — Strong evidence that mean price differs from $3.81

- **Economic interpretation**: Sample mean ($3.67) is significantly lower than hypothesized value. The t-statistic of -5.26 means the sample mean is more than 5 standard errors below $3.81—extremely unlikely if H_0 were true.
- **Why significant?**: Small standard error ($0.027) due to low variability in gas prices. Even a modest difference ($0.14) becomes statistically significant with such precision.

Example 2: Male Earnings (FAIL TO REJECT H_0)

- **Context**: Testing whether mean male earnings equal $50,000
- **Result**: t = 0.50, p = 0.62
- **Decision**: **Fail to reject** H_0 — No evidence that mean differs from $50,000
- **Economic interpretation**: Sample mean ($52,354) is higher than $50,000, but the difference is not statistically significant. With such high variability in earnings (large SE = $4,706), observing $52,354 when $\mu = \$50,000$ is quite plausible.
- **Gender comparison**: Male mean ($52,354) exceeds female mean ($41,413) by $10,941—a descriptive finding worthy of further investigation (e.g., comparing means using two-sample tests in later chapters).

Example 3: Real GDP per Capita Growth (FAIL TO REJECT H_0)

- **Context**: Testing whether mean growth rate equals 2.0%
- **Result**: t = -0.07, p = 0.95 (very high p-value)
- **Decision**: **Fail to reject** H_0 — Data strongly consistent with 2% growth hypothesis
- **Economic interpretation**: Sample mean (1.99%) is nearly identical to hypothesized value (2.00%). The tiny t-statistic (-0.07) indicates virtually no difference.
- **Large sample**: With n=245, the standard error is small (0.14%), yet we still can't reject H_0 because the sample mean is so close to 2.0%.
- **Policy implication**: If long-run growth assumptions for fiscal planning use 2%, the data support this assumption.

Patterns across examples:

1. **Sample size effects**:
   - Small n (32 for gas) → larger critical values (±2.04)
   - Large n (245 for GDP) → smaller critical values (±1.97), approaching z-value (±1.96)
2. **Standard error matters**:
   - Gas prices: Small SE ($0.027) → small difference ($0.14) is significant
   - Male earnings: Large SE ($4,706) → large difference ($2,354) is not significant
3. **Statistical vs. practical significance**:
   - Gas: Statistically significant, practically small ($0.14 difference)
   - Male earnings: Not statistically significant, but $2,354 difference is economically meaningful

This illustrates why we must consider both statistical evidence (p-values) and economic/practical magnitude when interpreting results.

4.8 One-Sided (Directional) Hypothesis Tests

Code

Context: In this section, we explore one-sided hypothesis tests where the alternative hypothesis specifies a direction (e.g., $\mu > 40,000$ or $\mu < 40,000$ rather than $\mu \neq 40,000$). One-sided tests are appropriate when you have a directional research question or when you only care about deviations in one direction. However, they should be specified before seeing the data to avoid data snooping bias. The p-values for one-sided tests are exactly half those of two-sided tests when the data support the specified direction.

```
# Test H0: mu >= 40000 vs HA: mu < 40000 (Lower-tailed)
mu0 = 40000
t_stat = (mean_earnings - mu0) / se_earnings
p_value_one_sided = stats.t.cdf(t_stat, n - 1)
t_crit_one_sided = stats.t.ppf(0.05, n - 1)

print(f"One-Sided Test (Lower-tailed): H0: mu >= ${mu0:,} vs HA: mu < ${mu0:,}"
    )
print(f"  t-statistic:       {t_stat:.4f}")
print(f"  p-value:           {p_value_one_sided:.4f}")
print(f"  Critical value:    {t_crit_one_sided:.4f}")
print(f"  Decision:          {'Reject H0' if t_stat < t_crit_one_sided else '
    Fail to reject H0'}")

# Test H0: mu <= 40000 vs HA: mu > 40000 (Upper-tailed)
p_value_upper = 1 - stats.t.cdf(t_stat, n - 1)
t_crit_upper = stats.t.ppf(0.95, n - 1)

print(f"\nOne-Sided Test (Upper-tailed): H0: mu <= ${mu0:,} vs HA: mu > ${mu0
    :,}")
print(f"  t-statistic:       {t_stat:.4f}")
print(f"  p-value:           {p_value_upper:.4f}")
print(f"  Critical value:    {t_crit_upper:.4f}")
print(f"  Decision:          {'Reject H0' if t_stat > t_crit_upper else 'Fail
    to reject H0'}")
```

Results

```
One-Sided Test (Lower-tailed): H0: mu >= $40,000 vs HA: mu < $40,000
  t-statistic:       0.7237
  p-value:           0.7649
  Critical value:    -1.6539
  Decision:          Fail to reject H0

One-Sided Test (Upper-tailed): H0: mu <= $40,000 vs HA: mu > $40,000
  t-statistic:       0.7237
  p-value:           0.2351
  Critical value:    1.6539
  Decision:          Fail to reject H0
```

Interpretation

When to use one-sided tests: Directional tests are appropriate when theory or policy dictates that we only care about deviations in one direction.

Lower-tailed test (H_0: $\mu \geq \$40,000$ vs H_a: $\mu < \$40,000$):

- **Use case**: Testing if earnings are *below* a minimum threshold (e.g., poverty line, minimum wage compliance)
- **Rejection region**: Left tail (t < -1.654)
- **Results**: t = 0.72, p = 0.76
- **Decision**: Fail to reject H_0 — No evidence that mean is below $40,000
- **Interpretation**: The sample mean ($41,413) is *above* $40,000, giving a p-value of 0.76 (76% probability of seeing such data if $\mu \geq \$40,000$). There's no reason to think earnings are below the threshold.

Upper-tailed test (H_0: $\mu \leq \$40,000$ vs H_a: $\mu > \$40,000$):

- **Use case**: Testing if earnings *exceed* a policy target or benchmark
- **Rejection region**: Right tail (t > 1.654)
- **Results**: t = 0.72, p = 0.24
- **Decision**: Fail to reject H_0 — No strong evidence that mean exceeds $40,000
- **Interpretation**: The sample mean is above $40,000, and the p-value (0.24) is smaller than in the lower-tailed test, but still not below 0.05. We can't confidently conclude $\mu > \$40,000$.

Comparison with two-sided test:

- **Two-sided**: H_0: $\mu = \$40,000$, p = 0.47, critical values ± 1.974
- **One-sided (upper)**: H_0: $\mu \leq \$40,000$, p = 0.24, critical value 1.654
- **One-sided (lower)**: H_0: $\mu \geq \$40,000$, p = 0.76, critical value -1.654

Key relationships:

1. One-sided p-value = (Two-sided p-value) / 2 (in the relevant direction)
   - Upper: 0.47 / 2 = 0.235 $\approx$ 0.24
2. One-sided critical values (± 1.654) are smaller in absolute value than two-sided (± 1.974)
   - Easier to reject H_0 with one-sided tests (if evidence points in hypothesized direction)

When should you use one-sided tests?

- **Appropriate**: When direction is pre-specified by theory or policy (e.g., testing if a drug is *better* than placebo, not just *different*)
- **Inappropriate**: When you might reject in either direction (post-hoc directional tests are controversial)

- **Conservative practice**: Most researchers use two-sided tests unless strong justification exists

Power consideration: One-sided tests have more **power** (ability to detect effects) in the specified direction because they concentrate all the rejection region in one tail. But this comes at the cost of being unable to detect effects in the other direction.

4.9 Inference for Proportions

Code

Context: In this section, we extend inference methods to proportions—situations where the variable of interest is binary (0/1, yes/no, success/failure). Proportions are ubiquitous in economics and social sciences: employment rates, market shares, voter preferences, default rates, and treatment effects. The inference framework is similar to means, but we use the binomial distribution's properties: for proportions, the standard error is $\sqrt{p(1-p)/n}$, where p is the sample proportion.

```
# Example: proportion data
n_total = 921
n_success = 480
p_hat = n_success / n_total
se_prop = np.sqrt(p_hat * (1 - p_hat) / n_total)

# Confidence interval
z_crit = 1.96  # For 95% CI
ci_lower_prop = p_hat - z_crit * se_prop
ci_upper_prop = p_hat + z_crit * se_prop

print(f"Proportion Analysis:")
print(f"  Sample size:       {n_total}")
print(f"  Number of successes: {n_success}")
print(f"  Sample proportion: {p_hat:.4f}")
print(f"  Standard error:    {se_prop:.4f}")
print(f"  95% CI:            [{ci_lower_prop:.4f}, {ci_upper_prop:.4f}]")

# Test H0: p = 0.50
p0 = 0.50
se_under_h0 = np.sqrt(p0 * (1 - p0) / n_total)
z_stat = (p_hat - p0) / se_under_h0
p_value_prop = 2 * (1 - stats.norm.cdf(abs(z_stat)))

print(f"\nHypothesis Test: H0: p = {p0:.2f}")
print(f"  z-statistic:       {z_stat:.4f}")
print(f"  p-value:           {p_value_prop:.4f}")
print(f"  Decision:          {'Reject H0' if abs(z_stat) > 1.96 else 'Fail to
    reject H0'}")
```

Results

```
Proportion Analysis:
  Sample size:       921
  Number of successes: 480
  Sample proportion: 0.5212
```

```
  Standard error:    0.0165
  95% CI:            [0.4889, 0.5534]

Hypothesis Test: H0: p = 0.50
  z-statistic:       1.2851
  p-value:           0.1988
  Decision:          Fail to reject H0
```

Interpretation

Binary data: Proportions arise when the outcome is binary (yes/no, success/failure, vote/abstain). Examples:

- **Political polling**: Proportion supporting a candidate
- **Quality control**: Proportion of defective items
- **Medical trials**: Proportion of patients who recover
- **Employment**: Proportion employed vs. unemployed

Sample proportion: $\hat{p} = 480/921 = 0.5212$ (52.12%)
This estimates the population proportion (p). In our sample, about 52% of observations are "successes."
Standard error for proportions:
SE $= \sqrt{\hat{p}(1-\hat{p})/n} = \sqrt{0.5212 \times 0.4788/921} = 0.0165$
For proportions, the standard error depends on:

1. **Sample proportion** (maximum at $\hat{p} = 0.5$, decreases toward 0 or 1)
2. **Sample size** (decreases as n increases)

95% Confidence interval: [0.489, 0.553]
We are 95% confident the true population proportion lies between 48.9% and 55.3%. The interval spans about 6.4 percentage points, reflecting moderate precision.
Hypothesis test: H_0: p = 0.50 vs H_a: p $\neq$ 0.50

- **Null hypothesis**: Population proportion equals 50% (e.g., a fair coin, evenly split electorate)
- **z-statistic**: 1.29 (note we use **z**, not t, for large-sample proportion tests)
- **p-value**: 0.20 (20% probability of observing $\hat{p} = 0.52$ if p = 0.50)
- **Decision**: Fail to reject H_0 — Data are consistent with p = 0.50

Why z-statistic instead of t? For proportions with large samples, the Central Limit Theorem implies:
$\hat{p} \sim N(p, p(1-p)/n)$
We use the normal (z) distribution rather than t because:

1. Sample size is large (n = 921)
2. The standard error formula for proportions is different from means

3. Historically, proportion tests used z; modern practice with large n gives nearly identical results

Interpretation: Although the sample proportion (52.12%) is slightly above 50%, the difference is not statistically significant. If the true proportion were 50%, observing 52.12% in a sample of 921 would occur about 20% of the time by random chance—quite plausible.

Special case: SE under H_0: Notice we computed SE two ways:

1. **For CI**: SE $= \sqrt{\hat{p}(1-\hat{p})/n} = 0.0165$ (uses sample proportion)
2. **For hypothesis test**: SE $= \sqrt{p_0(1-p_0)/n} = \sqrt{0.5 \times 0.5/921} = 0.0165$ (uses hypothesized proportion)

In this case they're almost identical because $\hat{p} \approx p_0$. But in general, hypothesis tests for proportions use the hypothesized value in the SE calculation.

Sample size and precision: With n=921, the SE is quite small (0.0165 = 1.65%). Doubling the sample size would reduce SE by $\sqrt{2}$, to about 1.17%. This square-root relationship means diminishing returns—quadrupling sample size only halves the SE.

4.10 Hypothesis Testing Visualization

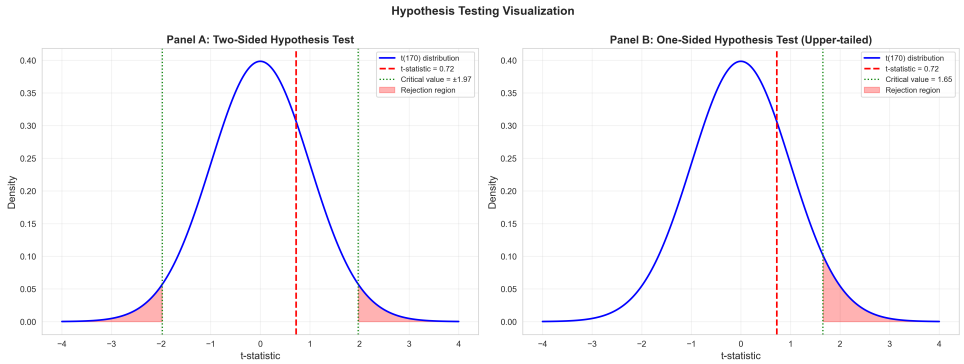
Code

Context: In this final section, we create comprehensive visualizations of the hypothesis testing framework—showing the null distribution, the observed test statistic, critical regions, and p-values graphically. Visual representations help build intuition about what p-values mean, how critical values define rejection regions, and why more extreme test statistics lead to smaller p-values. These visualizations are essential for communicating hypothesis test results to non-technical audiences and for developing a geometric understanding of statistical inference.

```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Panel A: Two-sided test
x = np.linspace(-4, 4, 500)
y = stats.t.pdf(x, n - 1)

axes[0].plot(x, y, 'b-', linewidth=2, label=f't({n-1}) distribution')
axes[0].axvline(x=t_stat, color='red', linewidth=2, linestyle='--',
                label=f't-statistic = {t_stat:.2f}')
axes[0].axvline(x=t_crit_95, color='green', linewidth=1.5, linestyle=':',
                label=f'Critical value = +-{t_crit_95:.2f}')
axes[0].axvline(x=-t_crit_95, color='green', linewidth=1.5, linestyle=':')

# Shade rejection regions
x_reject_lower = x[x < -t_crit_95]
x_reject_upper = x[x > t_crit_95]
axes[0].fill_between(x_reject_lower, 0, stats.t.pdf(x_reject_lower, n-1),
                     alpha=0.3, color='red', label='Rejection region')
axes[0].fill_between(x_reject_upper, 0, stats.t.pdf(x_reject_upper, n-1),
                     alpha=0.3, color='red')

axes[0].set_xlabel('t-statistic', fontsize=12)
axes[0].set_ylabel('Density', fontsize=12)
axes[0].set_title('Panel A: Two-Sided Hypothesis Test', fontsize=13, fontweight
    ='bold')
```

```
axes[0].legend(fontsize=9)
axes[0].grid(True, alpha=0.3)

# Panel B: One-sided test
axes[1].plot(x, y, 'b-', linewidth=2, label=f't({n-1}) distribution')
axes[1].axvline(x=t_stat, color='red', linewidth=2, linestyle='--',
                label=f't-statistic = {t_stat:.2f}')
axes[1].axvline(x=t_crit_upper, color='green', linewidth=1.5, linestyle=':',
                label=f'Critical value = {t_crit_upper:.2f}')

# Shade rejection region (upper tail)
x_reject = x[x > t_crit_upper]
axes[1].fill_between(x_reject, 0, stats.t.pdf(x_reject, n-1),
                     alpha=0.3, color='red', label='Rejection region')

axes[1].set_xlabel('t-statistic', fontsize=12)
axes[1].set_ylabel('Density', fontsize=12)
axes[1].set_title('Panel B: One-Sided Hypothesis Test (Upper-tailed)',
                  fontsize=13, fontweight='bold')
axes[1].legend(fontsize=9)
axes[1].grid(True, alpha=0.3)

plt.suptitle('Hypothesis Testing Visualization', fontsize=14, fontweight='bold'
    , y=1.0)
output_file = os.path.join(IMAGES_DIR, 'ch04_hypothesis_testing_visualization.
    png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results



Interpretation

Panel A: Two-Sided Test

This visualization shows the geometry of hypothesis testing:

1. **Blue curve**: The t-distribution with df=170 under H_0 ($\mu = \$40,000$)
2. **Red dashed line**: Our observed t-statistic (0.72), far from the rejection regions
3. **Green dotted lines**: Critical values (± 1.974) defining the rejection regions
4. **Red shaded areas**: Rejection regions (5% total area, 2.5% in each tail)

Key insights:

- The t-statistic (0.72) falls well within the "non-rejection region" (between -1.974 and +1.974)
- If t had fallen in the red shaded areas, we would reject H_0
- The distance from t=0.72 to the critical value (1.974) shows how far we are from rejecting
- The non-rejection region contains 95% of the probability under H_0

Panel B: One-Sided Test (Upper-Tailed)
For the upper-tailed test (H_a: $\mu > \$40,000$):

1. **Same blue curve and red dashed line** (distribution and t-statistic unchanged)
2. **Single green dotted line**: Critical value (1.654) is smaller than for two-sided test
3. **Red shaded area**: Rejection region only in upper tail (5% area)

Key differences from two-sided:

- Critical value is smaller (1.654 vs. 1.974) because all 5% α is in one tail
- Easier to reject H_0 if evidence points in the correct direction
- Our t-statistic (0.72) is still far from the rejection region

Visual interpretation of p-values:

- **Two-sided**: The p-value (0.47) is the area in both tails beyond ± 0.72
- **One-sided (upper)**: The p-value (0.24) is the area in the upper tail beyond 0.72

Practical decision-making:

- If t-statistic lands in red zone $\rightarrow$ Reject H_0 (result is "statistically significant")
- If t-statistic lands in white zone $\rightarrow$ Fail to reject H_0 (result is "not statistically significant")

Why visualizations matter: Seeing the geometry helps students understand:

- What "5% significance level" means (area in tails)
- How extreme the data must be to reject H_0
- Why larger $|t|$ values provide stronger evidence against H_0
- The relationship between critical values, rejection regions, and p-values

This visualization transforms abstract statistical concepts into concrete geometric intuition.

4.11 Conclusion

In this chapter, we've explored the fundamental tools of statistical inference—confidence intervals and hypothesis tests—using real earnings, price, and GDP data. You've learned how to quantify uncertainty in your estimates, test specific claims about population parameters, and make principled decisions based on sample data.

We've examined the t-distribution and understood why it's necessary when estimating population standard deviations from samples. Through multiple examples, you've seen how to construct confidence intervals at different levels (trading off precision for confidence) and conduct both two-sided and one-sided hypothesis tests. You've also extended these methods to proportions, a critical skill for analyzing binary outcomes in economics and social sciences.

Perhaps most importantly, you've learned to interpret p-values correctly—understanding that they measure the probability of your data given the null hypothesis, not the probability that the null is true. You've also seen the crucial distinction between statistical significance (is the effect detectably different from zero?) and practical significance (is the effect large enough to matter?).

What You've Learned:

- **Programming**: How to use scipy.stats for t-distribution calculations, construct confidence intervals programmatically, compute test statistics and p-values, and create visualizations that illuminate hypothesis testing concepts
- **Statistics**: How the t-distribution differs from the normal and when to use each, how to construct and interpret confidence intervals, how to conduct hypothesis tests using both p-values and critical values, and how to avoid common misinterpretations of statistical results
- **Economics**: How economists quantify uncertainty in estimates of means (earnings, prices, growth rates), how policymakers use hypothesis tests to evaluate claims about the economy, and why distinguishing statistical from practical significance matters for real-world decisions
- **Methodology**: How to specify null and alternative hypotheses before seeing data, choose appropriate significance levels based on the costs of errors, and communicate statistical findings to both technical and non-technical audiences

Looking Ahead:

In Chapter 5, we'll extend these inference tools to relationships between variables through simple linear regression. The concepts you've mastered here—sampling distributions, standard errors, confidence intervals, and hypothesis tests—will transfer directly to testing whether regression coefficients differ from zero. Understanding inference for means is the foundation for understanding inference about relationships.

Try extending your learning by conducting power analyses (how likely are you to detect a true effect of a given size?), exploring two-sample tests (comparing means between groups), or investigating what happens to inference when assumptions are violated. The more you practice with real data, the more confident you'll become in applying these essential statistical tools.

References:

- Cameron, A.C. (2022). *Analysis of Economics Data: An Introduction to Econometrics.* https://cameron.econ.ucdavis.edu/aed/index.html

- Python libraries: numpy, pandas, matplotlib, seaborn, scipy.stats
- Datasets: AED_EARNINGS.DTA, AED_GASPRICE.DTA, AED_EARNINGSMALE.DTA, AED_REALGDPPC.DTA

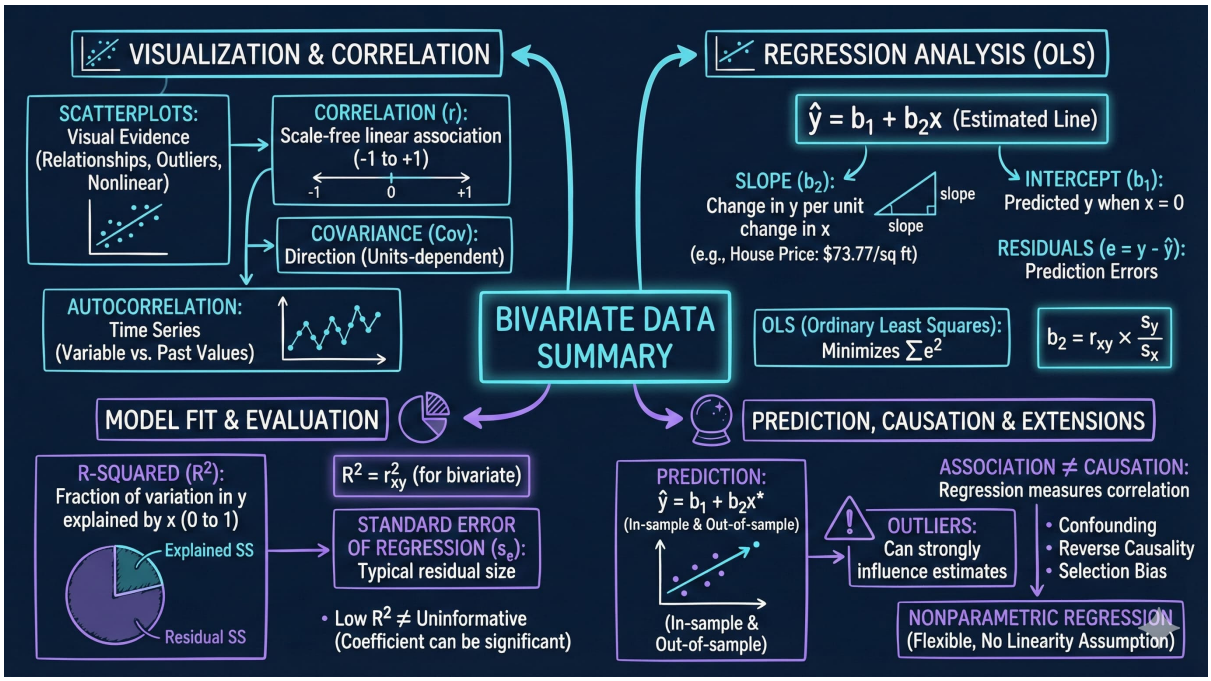
Data:
All datasets are available at: https://cameron.econ.ucdavis.edu/aed/aeddata.html

Part II

Bivariate Regression

Chapter 5

Bivariate Data Summary



This chapter explores relationships between two variables, introducing correlation, covariance, and simple linear regression using house price and size data from 29 California properties.

5.1 Introduction

This report demonstrates fundamental techniques for analyzing **bivariate data**—relationships between two variables. While Chapters 2-4 focused on univariate analysis (single variables), Chapter 5 introduces methods for understanding how variables relate to each other, culminating in **simple linear regression**, the foundation of econometrics.

We use a classic real estate dataset containing information on 29 house sales:

- **Primary variables**: Price (in dollars) and Size (in square feet)

- **Additional variables**: Bedrooms, bathrooms, lot size, age, month sold, list price
- **Research question**: How does house size affect sale price?

What You'll Learn:

- How to create and interpret two-way contingency tables for categorical data
- How to visualize bivariate relationships using scatter plots
- How to compute and interpret correlation and covariance
- How to estimate simple linear regression models using Ordinary Least Squares (OLS)
- How to interpret regression coefficients and statistical significance
- How to assess model fit using R^2, residuals, and other diagnostics
- How to make predictions using fitted regression models
- How to understand the relationship between correlation and regression
- How to recognize the distinction between association and causation
- How to explore nonparametric regression alternatives (LOWESS, kernel smoothing)

5.2 Setup and Data Loading

Code

Context: We begin by establishing our Python environment and loading a real estate dataset containing 29 house sales with information on price, size, and other characteristics. This dataset provides an ideal learning context because the relationship between house size and price is intuitive yet complex enough to demonstrate key concepts. We'll use pandas to load data directly from a remote repository, ensuring reproducibility and demonstrating modern data science workflows.

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.nonparametric.smoothers_lowess import lowess
from scipy import stats
from scipy.ndimage import gaussian_filter1d
import random
import os

# Set random seeds for reproducibility
```

```python
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
os.environ['PYTHONHASHSEED'] = str(RANDOM_SEED)

# GitHub data URL
GITHUB_DATA_URL = "https://raw.githubusercontent.com/quarcs-lab/data-open/master/AED/"

# Create output directories
IMAGES_DIR = 'images'
TABLES_DIR = 'tables'
os.makedirs(IMAGES_DIR, exist_ok=True)
os.makedirs(TABLES_DIR, exist_ok=True)

# Set plotting style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# Read in house data
data_house = pd.read_stata(GITHUB_DATA_URL + 'AED_HOUSE.DTA')

# Extract primary variables
price = data_house['price']
size = data_house['size']
```

Results

```
Data loaded: AED_HOUSE.DTA (29 observations, 8 variables)
Variables: price, size, bedrooms, bathrooms, lotsize, age, monthsold, list
```

Summary Statistics:
Price statistics:

- Mean: $253,910.34
- Median: $244,000.00
- Min: $204,000.00
- Max: $375,000.00
- Std Dev: $37,390.71

Size statistics:

- Mean: 1,883 sq ft
- Median: 1,800 sq ft
- Min: 1,400 sq ft
- Max: 3,300 sq ft
- Std Dev: 398 sq ft

Interpretation

Dataset context: This dataset contains information on 29 house sales, providing a manageable sample size for learning regression concepts while still capturing real-world complexity.

Primary research question: What is the relationship between house size and sale price? This is a classic econometric question with practical implications for:

- **Homebuyers**: Estimating fair prices based on square footage
- **Real estate agents**: Pricing new listings
- **Appraisers**: Conducting property valuations
- **Economists**: Understanding housing market dynamics

Price variability: The standard deviation ($37,391) represents about 15% of the mean price, indicating moderate variability. The range ($204,000 to $375,000) shows nearly a 2:1 difference between cheapest and most expensive homes.

Size variability: The standard deviation (398 sq ft) represents about 21% of the mean size. The largest house (3,300 sq ft) is more than twice the size of the smallest (1,400 sq ft).

Why this dataset? House prices and sizes have several attractive properties for teaching regression:

1. **Intuitive relationship**: Students understand that bigger houses generally cost more
2. **Economic relevance**: Real estate is familiar to most people
3. **Positive correlation**: The relationship is strong and positive (not always the case)
4. **Real data**: Captures actual market complexity (noise, outliers, nonlinearity)

Additional variables: While this chapter focuses on the bivariate relationship (price vs. size), the dataset includes other potential determinants (bedrooms, bathrooms, age) that will be explored in multiple regression (later chapters).

5.3 Two-Way Tabulation

Code

Context: Before analyzing continuous relationships, we convert our continuous variables (price and size) into categories to create a two-way contingency table. This tabulation provides an intuitive first look at how variables are associated by showing the joint distribution across categories. While this approach sacrifices information by binning continuous data, it offers clear visual insight into whether larger houses tend to be more expensive, making it a useful exploratory tool before moving to more sophisticated methods.

```
# Create categorical variables
price_range = pd.cut(price, bins=[0, 249999, np.inf],
                     labels=['< $250,000', '>= $250,000'])

```

```
size_range = pd.cut(size, bins=[0, 1799, 2399, np.inf],
                    labels=['< 1,800', '1,800-2,399', '>= 2,400'])

# Table 5.3: Two-way tabulation
crosstab = pd.crosstab(price_range, size_range, margins=True)
print("Table 5.3: Two-Way Tabulation")
print(crosstab)
crosstab.to_csv(os.path.join(TABLES_DIR, 'ch05_crosstab.csv'))
```

Results

Table 5.3: Two-Way Tabulation

price	$< 1{,}800$	1,800-2,399	$\geq 2{,}400$	All
$<$ \$250,000	11	6	0	17
$\geq$ \$250,000	2	7	3	12
All	**13**	**13**	**3**	**29**

Interpretation

What is a contingency table? A two-way tabulation (crosstab) shows the joint distribution of two categorical variables. Here, we've discretized continuous variables (price and size) into categories to reveal patterns.

Pattern observation:

1. **Upper-left cell (11 houses)**: Small, inexpensive houses ($<$ 1,800 sq ft, $<$ \$250k)
2. **Lower-right cell (3 houses)**: Large, expensive houses ($\geq$ 2,400 sq ft, $\geq$ \$250k)
3. **Off-diagonal mixing**: Some large houses are cheap (2 in lower-left) and some small houses are expensive (0 in upper-right)

Positive association: The concentration of observations along the main diagonal (upper-left to lower-right) visually confirms that **larger houses tend to be more expensive**. Specifically:

- Of 17 cheap houses ($<$ \$250k), 11 (65%) are small ($<$ 1,800 sq ft)
- Of 12 expensive houses ($\geq$ \$250k), 10 (83%) are medium or large ($\geq$ 1,800 sq ft)

Marginal distributions:

- **Row margins**: 17 houses below \$250k, 12 at or above \$250k
- **Column margins**: 13 small, 13 medium, 3 large houses

Limitations of categorization: By converting continuous variables to categories, we lose information. Two houses at 1,799 sq ft and 1,801 sq ft are treated as very different (different categories), while two houses at 1,401 sq ft and 1,799 sq ft are treated as identical (same category). This motivates continuous analysis methods like correlation and regression.

Conditional distributions: We can compute conditional probabilities:

- $P(\text{Price} \geq \$250\text{k} \mid \text{Size} \geq 2{,}400) = 3/3 = 100\%$
- $P(\text{Price} \geq \$250\text{k} \mid \text{Size} < 1{,}800) = 2/13 = 15\%$

This shows that large houses are much more likely to be expensive, quantifying the association.

Key Concept: Contingency Tables

A two-way contingency table (crosstab) shows the joint distribution of two categorical variables, revealing patterns of association. While useful for initial exploration, categorizing continuous variables sacrifices information—two values just on opposite sides of a cutpoint are treated as very different, while two values far apart within the same category are treated as identical. This motivates continuous analysis methods like correlation and regression that preserve the full information in the data.

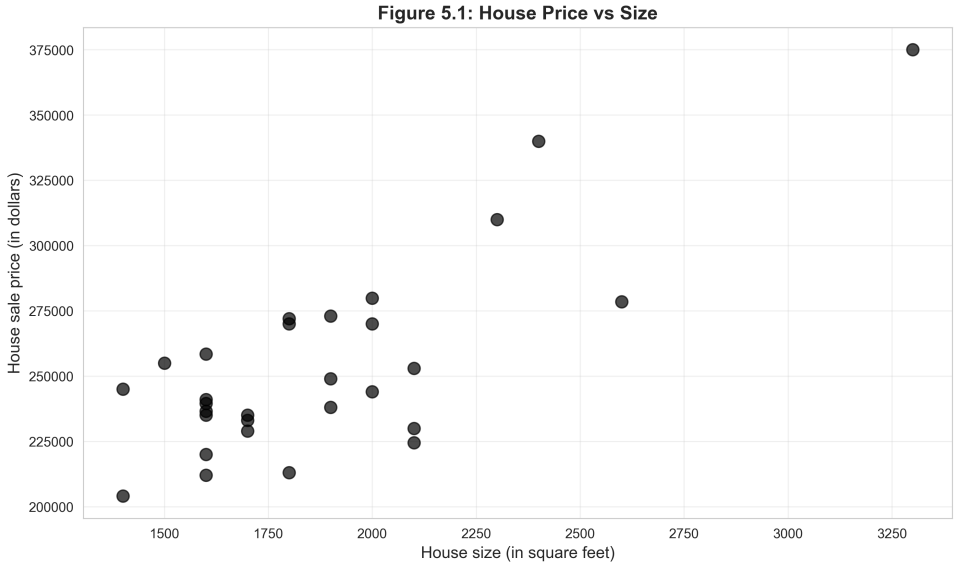
5.4 Scatter Plot Visualization

Code

Context: Scatter plots are the foundational tool for visualizing bivariate relationships, plotting each observation as a point with one variable on each axis. This visualization reveals the form (linear vs. nonlinear), direction (positive vs. negative), and strength (tight vs. dispersed) of the relationship at a glance. Before computing any statistics or fitting regression models, we should always create a scatter plot to understand the data structure and identify potential outliers or patterns that might violate modeling assumptions.

```
# Figure 5.1: Scatter plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(size, price, s=80, alpha=0.7, color='black', edgecolor='black')
ax.set_xlabel('House size (in square feet)', fontsize=12)
ax.set_ylabel('House sale price (in dollars)', fontsize=12)
ax.set_title('Figure 5.1: House Price vs Size', fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch05_fig1_scatter_price_vs_size.png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results



Interpretation

The scatter plot: This visualization plots each house as a single point with size on the horizontal axis (x) and price on the vertical axis (y). It's the most fundamental tool for visualizing bivariate relationships.

Pattern identification:

1. **Positive relationship**: As size increases (moving right), price tends to increase (moving up)
2. **Linear trend**: The points roughly follow a straight line, not a curve
3. **Variability**: Points don't fall exactly on a line—there's scatter around the trend
4. **No obvious outliers**: All points seem consistent with the overall pattern

Strength of relationship: The points cluster relatively tightly around an imaginary line, suggesting a **strong positive linear relationship**. If the relationship were weak, points would be much more dispersed.

Form of relationship: The approximately linear pattern justifies using **linear regression**. If the plot showed curvature (e.g., exponential growth, diminishing returns), linear regression would be inappropriate without transformation.

Variability/scatter: The vertical spread at any given size represents **unexplained variation**—differences in price not explained by size alone. These could reflect:

- Other features (bedrooms, bathrooms, lot size, age, location)
- Unobservable characteristics (quality of construction, neighborhood amenities, view)
- Market timing (month sold)
- Random variation (buyer preferences, negotiation outcomes)

No outliers: Unlike some datasets, we don't see extreme outliers (e.g., a tiny house selling for $375k or a huge house selling for $204k). This suggests the relationship is relatively clean.

Direction (positive slope): Every additional square foot of size is associated with higher prices. This makes economic sense—larger houses provide more utility (more space for living, entertaining, storage).

Why visualize first? Before computing statistics or running regression, always plot the data. The scatter plot can reveal:

- Nonlinear relationships that would be missed by correlation
- Outliers that might distort regression estimates
- Heteroscedasticity (changing variance)
- Clustering or grouping in the data

5.5 Correlation and Covariance

Code

Context: After visualizing the relationship, we quantify its strength and direction using covariance and correlation. Covariance measures how two variables move together, but its magnitude is difficult to interpret because it depends on the units of measurement. Correlation solves this problem by standardizing the covariance, yielding a unitless measure bounded between -1 and +1 that clearly indicates both the direction and strength of the linear relationship.

```
# Covariance and correlation
cov_matrix = data_house[['price', 'size']].cov()
corr_matrix = data_house[['price', 'size']].corr()

print("Covariance matrix:")
print(cov_matrix)

print("\nCorrelation matrix:")
print(corr_matrix)
corr_matrix.to_csv(os.path.join(TABLES_DIR, 'ch05_correlation_matrix.csv'))

print(f"\nCorrelation coefficient: {corr_matrix.loc['price', 'size']:.4f}")
```

Results

Covariance Matrix:

	price	size
price	1.398×10^{9}	1.170×10^{7}
size	1.170×10^{7}	1.586×10^{5}

Correlation Matrix:

	price	size
price	1.000	0.786
size	0.786	1.000

Correlation coefficient: 0.7858

Interpretation

Covariance: The covariance between price and size is 1.170×10^7 (11.7 million). This positive value confirms that the two variables move together—when size is above average, price tends to be above average.

Problem with covariance: The magnitude (11.7 million) is hard to interpret because:

1. **Units**: It's in "dollars × square feet," a meaningless unit
2. **Scale-dependent**: Changing measurement units (e.g., square meters instead of square feet) would change the covariance
3. **No upper bound**: We can't judge if 11.7 million is "large" or "small"

Correlation to the rescue: The correlation coefficient solves these problems by standardizing the covariance:

$$r = \frac{\text{Cov}(\text{price}, \text{size})}{\sigma_{\text{price}} \times \sigma_{\text{size}}}$$

Properties of correlation:

1. **Bounded**: Always between -1 and +1
2. **Unitless**: No measurement units (dimensionless)
3. **Standardized**: Easy to interpret strength

Interpretation of r = 0.786:

- **Sign**: Positive, confirming the relationship is direct (larger houses → higher prices)
- **Magnitude**: 0.786 is considered a **strong positive correlation**
  - Weak: $|r| < 0.3$
  - Moderate: $0.3 \leq |r| < 0.7$
  - Strong: $|r| \geq 0.7$
- **Variance explained**: $r^2 = 0.786^2 = 0.618$, meaning 62% of price variation is associated with size variation

What correlation measures: Correlation quantifies the **strength and direction of the linear relationship** between two variables. It does NOT measure:

- Causation (does size cause higher prices, or do wealthy buyers choose larger houses?)
- Nonlinear relationships (correlation can be zero even if a strong nonlinear relationship exists)
- The slope (correlation is scale-free; regression gives the slope)

Perfect correlation scenarios:

- r = +1: Perfect positive linear relationship (all points fall exactly on an upward-sloping line)
- r = -1: Perfect negative linear relationship (all points fall exactly on a downward-sloping line)
- r = 0: No linear relationship (but nonlinear relationships may exist!)

Diagonal elements: The correlation of a variable with itself is always 1.000 (as shown in the diagonal of the correlation matrix).

Symmetry: Corr(price, size) = Corr(size, price) = 0.786. Correlation is symmetric—the order doesn't matter.

Key Concept: Correlation Coefficient

The correlation coefficient r measures the strength and direction of the linear relationship between two variables, always bounded between -1 and +1. Unlike covariance, correlation is unitless and scale-free, making it easy to interpret: $|r| < 0.3$ suggests weak association, $0.3 \leq |r| < 0.7$ indicates moderate association, and $|r| \geq 0.7$ represents strong association. Importantly, correlation only captures linear relationships—variables can have zero correlation yet still be strongly related in nonlinear ways.

5.6 Simple Linear Regression

Code

Context: We now move from describing the relationship to modeling it using Ordinary Least Squares (OLS) regression, the foundational technique in econometrics. OLS finds the straight line that best fits the data by minimizing the sum of squared vertical distances between observed and predicted values. This method gives us precise estimates of how much price changes for each additional square foot, along with statistical measures to assess the reliability of our estimates and the overall model fit.

```
# Fit regression model
model = ols('price ~ size', data=data_house).fit()

print("\nOLS Regression Results:")
print(model.summary())

# Save coefficients
coef_table = pd.DataFrame({
    'coefficient': model.params,
    'std_err': model.bse,
    't_value': model.tvalues,
    'p_value': model.pvalues
})
```

```
14 coef_table.to_csv(os.path.join(TABLES_DIR, 'ch05_regression_coefficients.csv'))
```

Results

OLS Regression Results:

Dep. Variable:	price	R-squared:	0.617
Model:	OLS	Adj. R-squared:	0.603
Method:	Least Squares	F-statistic:	43.58
No. Observations:	29	Prob (F-statistic):	4.41e-07
Df Residuals:	27		
Df Model:	1		

Coefficients:

Variable	Coefficient	Std Error	t-value	p-value	95% CI Lower	95% CI Upper
Intercept	115,017.28	21,489.36	5.352	0.00001	70,924.76	159,109.81
size	73.77	11.17	6.601	0.0000004	50.84	96.70

Interpretation

The regression equation:

$$\text{Price} = 115{,}017.28 + 73.77 \times \text{Size} + \varepsilon$$

This equation estimates the **expected price** given house size, plus a random error term (ε).
Intercept (β_0 = $115,017.28):

- **Interpretation**: The predicted price for a house with **zero square feet**
- **Economic meaning**: This is nonsensical (can't have a 0 sq ft house) but mathematically necessary
- **Statistical significance**: t = 5.35, p < 0.001 (highly significant)
- **Extrapolation issue**: The intercept is far outside the observed data range (1,400-3,300 sq ft), so we shouldn't interpret it literally

Slope (β_1 = $73.77):

- **Interpretation**: Each additional square foot is associated with a **$73.77 increase** in expected price
- **Practical meaning**: A 100 sq ft increase → $7,377 price increase; a 500 sq ft increase → $36,885 price increase
- **Statistical significance**: t = 6.60, p < 0.001 (highly significant)
- **Confidence interval**: We're 95% confident the true slope is between $50.84 and $96.70 per sq ft

R-squared (0.617):

- **Interpretation**: Size explains **61.7%** of the variation in prices

- **Unexplained variation**: The remaining 38.3% is due to other factors (bedrooms, bathrooms, location, quality, etc.) and random variation
- **Model fit**: An R^2 of 0.62 is considered quite good for cross-sectional real estate data

Adjusted R-squared (0.603):

- **Purpose**: Adjusts R^2 for the number of predictors, penalizing model complexity
- **Formula**: $R^2_{\text{adj}} = 1 - [(1 - R^2)(n - 1)/(n - k - 1)]$
- **Here**: With only one predictor, R^2_{adj} (0.603) is very close to R^2 (0.617)
- **Use**: More important when comparing models with different numbers of variables

F-statistic (43.58, p < 0.001):

- **Null hypothesis**: H_0: $\beta_1 = 0$ (size has no effect on price)
- **Decision**: Reject H_0 with very high confidence
- **Interpretation**: The model as a whole is statistically significant
- **Relationship to t-test**: For simple regression (one predictor), F = t^2 ($43.58 \approx 6.60^2$)

Degrees of freedom:

- **Df Model = 1**: One predictor (size)
- **Df Residuals = 27**: n - k - 1 = 29 - 1 - 1 = 27 observations minus parameters estimated

Standard errors and t-values:

- **SE(intercept) = $21,489**: Measures uncertainty in the intercept estimate
- **SE(slope) = $11.17**: Measures uncertainty in the slope estimate
- **t-values**: Coefficients divided by their standard errors (tests H_0: $\beta = 0$)

Why OLS? Ordinary Least Squares minimizes the sum of squared residuals (prediction errors). This gives the "best-fitting" line in a precise mathematical sense, with desirable statistical properties (unbiased, efficient under classical assumptions).

Causation vs. correlation: While the regression shows a strong association, we cannot conclude that **increasing size causes higher prices**. Possible explanations:

1. **Direct causation**: Larger houses provide more utility → buyers pay more
2. **Reverse causation**: Wealthier buyers purchase both larger and more expensive houses
3. **Confounding**: Location, quality, and age affect both size and price simultaneously

Key Concept: Ordinary Least Squares (OLS)

OLS finds the line that minimizes the sum of squared vertical distances between observed data points and the fitted regression line. This "best fit" criterion gives us unbiased estimates of the relationship between variables under standard assumptions. The slope

coefficient β_1 tells us how much Y changes when X increases by one unit, while the intercept β_0 represents the predicted value of Y when X equals zero (though this may not always have a meaningful economic interpretation).

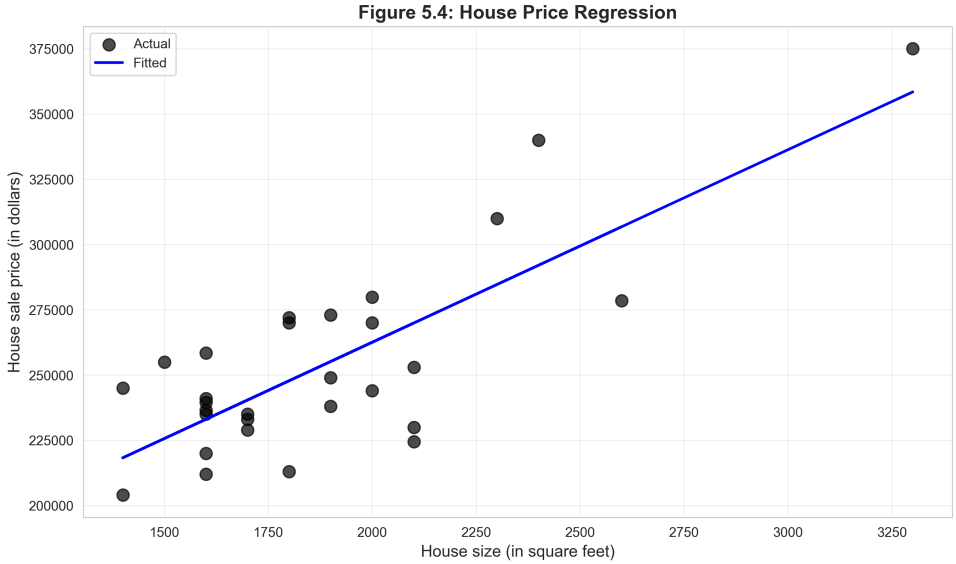
5.7 Regression Line Visualization

Code

Context: Having estimated the regression model, we now visualize the fitted line alongside the actual data points to assess how well our linear model captures the relationship. This visualization is critical for evaluating model assumptions—we can check whether the linear form is appropriate, whether residuals appear random, and whether any observations are unusually far from the fitted line. The scatter plot with the regression line provides intuitive visual feedback about model quality that complements the numerical statistics.

```
# Figure 5.4: Scatter plot with regression line
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(size, price, s=80, alpha=0.7, color='black',
           edgecolor='black', label='Actual')
ax.plot(size, model.fittedvalues, color='blue', linewidth=2, label='Fitted')
ax.set_xlabel('House size (in square feet)', fontsize=12)
ax.set_ylabel('House sale price (in dollars)', fontsize=12)
ax.set_title('Figure 5.4: House Price Regression',
             fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch05_fig4_regression_line.png')
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results



Interpretation

The regression line: The blue line represents the **fitted values** ($\hat{y}$) from the regression equation. For each size value, the line shows the predicted price based on the model.

Least Squares criterion: OLS chooses the line that **minimizes the sum of squared vertical distances** from each point to the line. These vertical distances are the **residuals** ($\varepsilon = y - \hat{y}$).

Visual assessment of fit:

1. **Line passes through the middle**: The regression line cuts through the center of the point cloud
2. **Positive slope**: The upward tilt confirms the positive relationship
3. **Residuals**: Vertical distances from points to the line show prediction errors
4. **Balanced errors**: Roughly equal numbers of points above and below the line

Predicted vs. actual values:

- **Actual values** (black dots): What houses actually sold for
- **Fitted values** (blue line): What the model predicts based on size alone
- **Residuals**: The difference between actual and fitted values

Why a straight line? The assumption of **linearity** implies that the effect of size on price is constant:

- Adding 100 sq ft to a 1,500 sq ft house increases price by the same amount as adding 100 sq ft to a 2,500 sq ft house (both increase price by $7,377)

Checking assumptions visually:

1. **Linearity**: The scatter appears roughly linear (not curved)

2. **Constant variance**: The vertical spread seems roughly constant across sizes (no obvious fan shape)

3. **No outliers**: All points reasonably close to the line

Interpolation vs. extrapolation:

- **Interpolation** (safe): Predicting prices for sizes between 1,400 and 3,300 sq ft (observed range)
- **Extrapolation** (risky): Predicting prices for sizes outside this range (e.g., 5,000 sq ft or 800 sq ft)—the relationship may not hold

Goodness of fit: The relatively tight clustering around the line confirms the high R^2 (0.617). If points were widely scattered, R^2 would be low.

5.8 Prediction Using Regression

Code

Context: One of the most practical applications of regression is prediction—using our model to estimate the expected value of the dependent variable for a given value of the independent variable. Here we demonstrate how to predict the price of a 2,000 square foot house using our fitted regression equation. This prediction represents the conditional expectation given the observed relationship, though individual houses will vary around this average due to other factors not captured in our simple model.

```
# Predict for a house of 2,000 square feet
new_size = pd.DataFrame({'size': [2000]})
predicted_price = model.predict(new_size)

print(f"\nPrediction for a 2,000 sq ft house:")
print(f"  Predicted price: ${predicted_price.values[0]:,.2f}")

# Manual calculation
beta0 = model.params['Intercept']
beta1 = model.params['size']
manual_prediction = beta0 + beta1 * 2000

print(f"\nManual calculation:")
print(f"  y-hat = {beta0:.2f} + {beta1:.2f} * 2000 = ${manual_prediction:,.2f}"
    )
```

Results

```
Prediction for a 2,000 sq ft house:
  Predicted price: $262,559.36

Manual calculation:
  y-hat = 115017.28 + 73.77 * 2000 = $262,559.36
```

Interpretation

Point prediction: For a house of exactly 2,000 sq ft, our model predicts a price of **$262,559.36**.

Calculation breakdown:

- Intercept contribution: $115,017.28
- Size contribution: $73.77 \times 2{,}000 = \$147{,}542.00$
- Total: $\$115{,}017.28 + \$147{,}542.00 = \$262{,}559.28$

Interpretation:

- This is the **expected** or **average** price for 2,000 sq ft houses
- Individual houses will vary around this prediction due to other factors
- This is a **conditional expectation**: E[Price | Size = 2,000]

Uncertainty in predictions: While the point estimate is $262,559, there are two sources of uncertainty:

1. **Estimation uncertainty**: We don't know the true β_0 and β_1; we only have estimates
2. **Fundamental uncertainty**: Even if we knew the true parameters, individual houses vary around the mean

Prediction intervals (not shown but important): A 95% prediction interval might be [$190,000, $335,000], reflecting the uncertainty in predicting an individual house price. This is **much wider** than a confidence interval for the mean price, which would be [$250,000, $275,000].

Within-sample vs. out-of-sample:

- This is an **interpolation** (2,000 sq ft is within the observed range of 1,400-3,300)
- The prediction is relatively reliable because we have observed similar houses
- Predicting for 5,000 sq ft would be **extrapolation**, with much greater uncertainty

Practical use: Real estate agents, appraisers, and buyers can use this model to:

- Estimate fair market value before listing a house
- Identify underpriced or overpriced listings
- Negotiate prices based on comparable square footage
- Make offers on houses before appraisal

Limitations: This prediction ignores other important factors:

- Number of bedrooms/bathrooms
- Age and condition
- Location and neighborhood quality
- Lot size and amenities
- Market conditions (hot vs. cold market)

More realistic predictions would use **multiple regression** (Chapter 6+), incorporating these additional variables.

5.9 Relationship Between Regression and Correlation

Code

Context: Students often wonder about the connection between correlation (which we computed earlier) and R-squared from regression. In simple linear regression with one predictor, these measures are mathematically linked: R^2 equals r^2 (the squared correlation coefficient). Understanding this relationship deepens our grasp of what regression is doing and clarifies how the variance-explained interpretation connects to the strength of the linear association between variables.

```
# Relationship between regression and correlation
r = corr_matrix.loc['price', 'size']
r_squared = r ** 2

print(f"\nCorrelation coefficient (r):  {r:.4f}")
print(f"R-squared from regression:    {model.rsquared:.4f}")
print(f"r-squared:                    {r_squared:.4f}")
print("  (R-squared and r-squared should be equal)")
```

Results

```
Correlation coefficient (r):  0.7858
R-squared from regression:    0.6175
r-squared:                    0.6175
  (R-squared and r-squared should be equal)
```

Interpretation

Key relationship: For **simple linear regression** (one predictor), R^2 from the regression equals r^2 (the squared correlation coefficient). This is always true.

Proof of equality:

- Correlation: r = 0.7858
- $r^2 = 0.7858^2 = 0.6175$
- Regression R^2: 0.6175
- They match exactly (within rounding error)

Why they're equal: Both measure the proportion of variance in Y (price) explained by X (size):

- r^2: Measures the proportion of variance in each variable explained by the linear relationship

- R^2: Measures the proportion of variance in Y explained by the regression model

In simple regression, these are identical. But in **multiple regression** (multiple predictors), R^2 generalizes while r^2 does not (you can't have a single correlation with multiple predictors).

Interpreting $R^2 = 0.6175$:

- 61.75% of the variance in house prices is explained by size
- 38.25% remains unexplained (residual variance)
- This is a fairly good fit for cross-sectional data

Relationship between correlation and slope:
The regression slope can be written as: $\beta_1 = r \times (\sigma_y/\sigma_x)$
Where:

- r = correlation (0.7858)
- σ_y = standard deviation of price ($37,391)
- σ_x = standard deviation of size (398 sq ft)

Calculation: $\beta_1 = 0.7858 \times (37{,}391 \ / \ 398) = 0.7858 \times 93.92 = 73.77$ ✓

Key differences between r and β_1:

1. **Units**: r is unitless; β_1 has units (dollars per sq ft)
2. **Interpretation**: r measures strength and direction; β_1 measures the rate of change
3. **Symmetry**: r is symmetric [Corr(X,Y) = Corr(Y,X)]; β_1 is not [slope of Y∼X ≠ slope of X∼Y]
4. **Causality**: Neither implies causation, but β_1 at least has a directional interpretation

When to use each:

- **Correlation**: When you want a standardized measure of association (comparing across different variable pairs)
- **Regression**: When you want to predict Y from X, or interpret the effect in original units

Multiple regression extension: In multiple regression, R^2 still measures explained variance, but there's no single correlation coefficient (many pairwise correlations exist between Y and X_1, X_2, ..., X_k).

Key Concept: R^2 (Coefficient of Determination)

R^2 measures the proportion of variance in the dependent variable that is explained by the model. For example, $R^2 = 0.62$ means 62% of the variation in Y is accounted for by our predictor(s), with 38% remaining unexplained. In simple linear regression, R^2 equals r^2 (the squared correlation coefficient). Higher R^2 indicates better model fit, but doesn't guarantee the model is appropriate, that the relationships are causal, or that predictions will be accurate for new data.

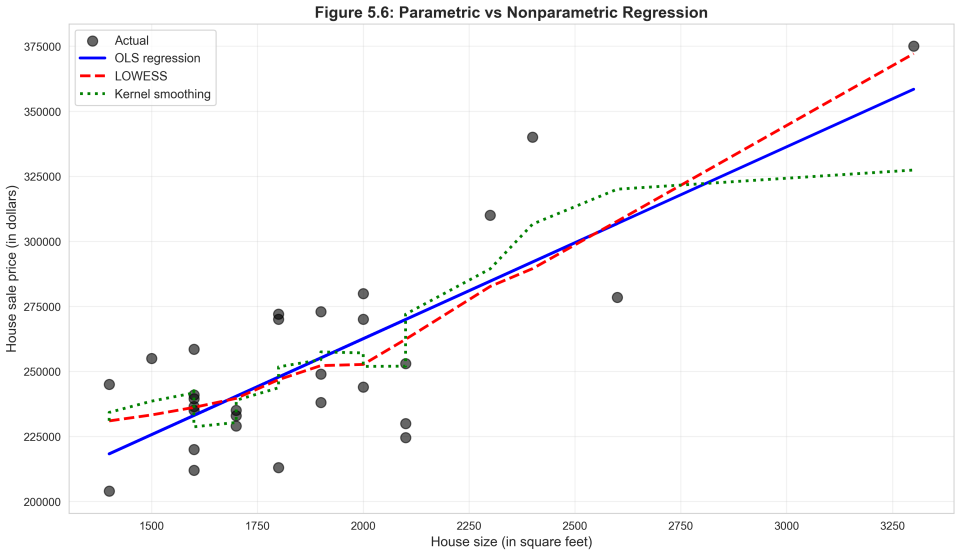
5.10 Nonparametric Regression Alternatives

Code

Context: While linear regression assumes a straight-line relationship, nonparametric methods like LOWESS (Locally Weighted Scatterplot Smoothing) and kernel smoothing let the data determine the functional form without imposing rigid parametric assumptions. By comparing OLS with these flexible alternatives, we can assess whether the linearity assumption is reasonable or whether the true relationship exhibits curves, bends, or other nonlinear features that would require more sophisticated modeling approaches.

```
# Nonparametric regression using lowess
lowess_result = lowess(price, size, frac=0.6)

# Sort data for smooth plotting
sort_idx = np.argsort(size)
size_sorted = size.iloc[sort_idx]
price_sorted = price.iloc[sort_idx]

# Kernel smoothing (using Gaussian filter as approximation)
sigma = 2  # bandwidth parameter
price_smooth = gaussian_filter1d(price_sorted, sigma)

fig, ax = plt.subplots(figsize=(12, 7))

# Scatter plot
ax.scatter(size, price, s=80, alpha=0.6, color='black',
           edgecolor='black', label='Actual', zorder=1)

# OLS line
ax.plot(size, model.fittedvalues, color='blue', linewidth=2.5,
        label='OLS regression', zorder=2)

# LOWESS
ax.plot(lowess_result[:, 0], lowess_result[:, 1], color='red',
        linewidth=2.5, linestyle='--', label='LOWESS', zorder=3)

# Kernel smoothing
ax.plot(size_sorted, price_smooth, color='green', linewidth=2.5,
        linestyle=':', label='Kernel smoothing', zorder=4)

ax.set_xlabel('House size (in square feet)', fontsize=12)
ax.set_ylabel('House sale price (in dollars)', fontsize=12)
ax.set_title('Figure 5.6: Parametric vs Nonparametric Regression',
             fontsize=14, fontweight='bold')
ax.legend(fontsize=11)
ax.grid(True, alpha=0.3)

output_file = os.path.join(IMAGES_DIR, 'ch05_fig6_nonparametric_regression.png'
    )
plt.tight_layout()
plt.savefig(output_file, dpi=300, bbox_inches='tight')
plt.close()
```

Results



Interpretation

Why nonparametric methods? Linear regression assumes the relationship between X and Y is a straight line. But what if this assumption is wrong? Nonparametric regression methods allow the data to determine the shape of the relationship without imposing a parametric form (like linearity).

OLS (blue line - parametric):

- Assumes a straight-line relationship: $\hat{y} = \beta_0 + \beta_1 x$
- Estimates only 2 parameters (slope and intercept)
- Efficient if the true relationship is linear
- Can be misleading if the true relationship is nonlinear

LOWESS (red dashed line - nonparametric):

- **Locally Weighted Scatterplot Smoothing**
- Fits separate regressions in small neighborhoods around each point
- Weights nearby points more heavily (local averaging)
- Can capture curves, bends, and local trends
- Parameter `frac=0.6` controls smoothness (fraction of data used in each local regression)

Kernel smoothing (green dotted line - nonparametric):

- Uses a weighted moving average with a Gaussian kernel
- Parameter `sigma=2` controls bandwidth (larger = smoother)
- Similar philosophy to LOWESS but different implementation
- Computationally faster for large datasets

Comparison in this dataset:

1. **All three methods are similar**: This suggests the linear assumption is reasonable
2. **Slight curvature in LOWESS**: The red line shows a tiny bit of curvature, but it's very close to the OLS line
3. **No dramatic nonlinearity**: We don't see S-curves, exponential growth, or diminishing returns

When would nonparametric methods differ more?

- If the relationship were curved (e.g., quadratic, exponential)
- If there were threshold effects (e.g., jumps at certain values)
- If the relationship varied across the range of X (e.g., steep at low values, flat at high values)

Trade-offs:
Parametric (OLS) advantages:

- Simple and interpretable (just two numbers: slope and intercept)
- Efficient (uses data economically)
- Easy to extrapolate (just extend the line)
- Statistical theory is well-developed (standard errors, confidence intervals, hypothesis tests)

Nonparametric advantages:

- Flexible (can fit any shape)
- No risk of model misspecification
- Good for exploratory analysis
- Can reveal unexpected patterns

Nonparametric disadvantages:

- Harder to interpret (no single slope)
- Less efficient (needs more data)
- Difficult to extrapolate
- More complex statistical inference

Best practice:

1. Start with a scatter plot
2. Fit both parametric (OLS) and nonparametric (LOWESS) models
3. Compare: If they're similar, use OLS (simpler). If they differ, investigate why—there may be important nonlinearity

In this case: The linear model is adequate. The LOWESS and kernel smoothing curves don't reveal any dramatic departures from linearity, so we can confidently use the simple OLS regression.

5.11 Conclusion

In this chapter, we explored the rich world of bivariate data analysis—moving beyond single-variable summaries to understanding relationships between pairs of variables. Using California house sales data, we examined how house size relates to sale price through progressively sophisticated methods: from contingency tables to scatter plots, correlation, and finally simple linear regression.

Through this progression, you discovered that while categorization (two-way tables) provides initial insight, continuous methods preserve more information. Scatter plots revealed the form and strength of relationships visually, while correlation quantified linear association in a standardized, unitless metric (r = 0.786). Most importantly, you learned how Ordinary Least Squares regression not only measures association but also provides a predictive equation: Price = \$115,017 + \$73.77 × Size.

What You've Learned:

On the **programming** side, you've gained hands-on experience with pandas for data manipulation, matplotlib and seaborn for creating publication-quality scatter plots, and statsmodels for fitting OLS regression models. You can now extract and interpret regression output including coefficients, standard errors, t-statistics, p-values, and R^2, and you've explored nonparametric alternatives like LOWESS that relax linearity assumptions.

From a **statistical** perspective, you understand the critical distinction between covariance (scale-dependent) and correlation (standardized), why OLS minimizes squared residuals to find the best-fit line, what R^2 reveals about model fit (61.7% of price variance explained by size), and the mathematical connection between correlation and regression ($R^2 = r^2$ in simple regression). You also learned to distinguish between interpolation (safe, within observed data) and extrapolation (risky, beyond observed range).

In terms of **economic interpretation**, you can now translate regression coefficients into meaningful statements: each additional square foot increases expected price by approximately \$74, though individual houses vary due to factors not captured by size alone. Crucially, you've internalized that association does not imply causation—larger houses cost more, but we cannot conclude that adding square footage causes higher value without controlling for confounding factors.

Most importantly, you've learned essential **methodology**: always visualize data before modeling, compare parametric and nonparametric approaches to validate assumptions, and recognize that simple regression, while foundational, captures only part of reality. The remaining 38% of unexplained price variation points to the need for multiple regression with additional predictors.

Looking Ahead:

The simple bivariate regression you've mastered here is just the beginning. In upcoming chapters, you'll extend these techniques to multiple regression, adding variables like bedrooms,

bathrooms, lot size, and age to better explain housing prices. You'll learn about regression inference—constructing confidence intervals for coefficients and prediction intervals for new observations. You'll discover how to handle nonlinear relationships through transformations (log-linear, log-log models), how interaction terms allow effects to vary, and how diagnostic tools like residual plots help assess model assumptions.

The real power of regression emerges when you combine this foundational understanding with more complex econometric techniques: panel data methods that track entities over time, instrumental variables that address endogeneity, difference-in-differences designs that estimate causal effects, and time series models that account for autocorrelation. Every one of these advanced techniques builds directly on the simple linear regression framework you've learned in this chapter—understanding OLS thoroughly is your passport to the entire world of econometric analysis.

References:

- Cameron, A.C. (2022). *Analysis of Economics Data: An Introduction to Econometrics.* https://cameron.econ.ucdavis.edu/aed/index.html
- Python libraries: numpy, pandas, matplotlib, seaborn, statsmodels, scipy

Data:
All datasets are available at: https://cameron.econ.ucdavis.edu/aed/aeddata.html

- Dataset: AED_HOUSE.DTA (29 house sales with price, size, and other characteristics)
- House sales data: Residential properties in a single market, showing variation in size (1,400-3,300 sq ft) and price ($204,000-$375,000)

Key Formulas:

- **Sample covariance**: $\text{Cov}(X, Y) = \sum[(x_i - \bar{x})(y_i - \bar{y})]/(n-1)$
- **Sample correlation**: $r = \text{Cov}(X, Y)/(\sigma_x \times \sigma_y)$
- **OLS regression**: $\hat{y} = \beta_0 + \beta_1 x$
- **OLS slope**: $\beta_1 = \text{Cov}(X, Y)/\text{Var}(X) = r \times (\sigma_y/\sigma_x)$
- **OLS intercept**: $\beta_0 = \bar{y} - \beta_1 \bar{x}$
- **R-squared**: $R^2 = 1 - (SSR/SST) = r^2$ (in simple regression)
- **Residual**: $\varepsilon_i = y_i - \hat{y}_i$