

Statistics for Data Science: Lecture Notes

DATA11007

Luigi Acerbi¹, Antti Honkela²

¹*Associate Professor, Department of Computer Science, University of Helsinki*

²*Professor, Department of Computer Science, University of Helsinki*

2025-09-07

Table of contents

Preface	1
About These Notes	1
How to Use These Notes	2
Prerequisites	4
Course Structure	4
Programming	5
Acknowledgments	5
1 Probability Foundations	7
1.1 Learning Objectives	7
1.2 Why Do We Need Statistics?	7
1.3 Foundations of Probability	12
1.4 Independence and Conditional Probability	17
1.5 Random Variables	23
1.6 Multivariate Distributions	44
1.7 Chapter Summary and Connections	50
2 Expectation	53
2.1 Learning Objectives	53
2.2 Introduction and Motivation	53
2.3 Foundations of Expectation	58
2.4 Properties of Expectation	64
2.5 Variance and Its Properties	68
2.6 Sample Mean and Variance	75
2.7 Covariance and Correlation	76
2.8 Expectation with Matrices	79
2.9 Conditional Expectation	84
2.10 More About the Normal Distribution	86
2.11 Chapter Summary and Connections	89
3 Convergence and The Basics of Inference	93
3.1 Learning Objectives	93
3.2 Introduction and Motivation	93
3.3 Inequalities: Bounding the Unknown	95
3.4 Convergence of Random Variables	101
3.5 The Two Fundamental Theorems of Statistics	107
3.6 The Language of Statistical Inference	111
3.7 Chapter Summary and Connections	118
4 Nonparametric Estimation and The Bootstrap	123
4.1 Learning Objectives	123
4.2 Introduction and Motivation	123

4.3	Confidence Sets: The Foundation	124
4.4	The Plug-In Principle: A General Method for Estimation	126
4.5	The Bootstrap: Simulating Uncertainty	132
4.6	Bootstrap Confidence Intervals	138
4.7	Bootstrap Application: Higher Moments	144
4.8	When The Bootstrap Fails	149
4.9	Chapter Summary and Connections	151
5	Parametric Inference I: Finding Estimators	157
5.1	Learning Objectives	157
5.2	Introduction: Machine Learning As Statistical Estimation	157
5.3	Parametric Models	158
5.4	The Method of Moments (MoM)	160
5.5	Maximum Likelihood Estimation (MLE)	164
5.6	MLE Via Numerical Optimization	170
5.7	Chapter Summary and Connections	186
6	Parametric Inference II: Properties of Estimators	189
6.1	Learning Objectives	189
6.2	Introduction: How Good Are Our Estimators?	189
6.3	Warm-up: A Complete MLE Example with Numerical Optimization	190
6.4	Core Properties of the Maximum Likelihood Estimator	192
6.5	Fisher Information and Confidence Intervals	197
6.6	Additional Topics	207
6.7	Connection to Machine Learning: Cross-Entropy as MLE	212
6.8	MLE for Latent Variable Models: The EM Algorithm	213
6.9	Chapter Summary and Connections	219
7	Hypothesis Testing and p-values	223
7.1	Learning Objectives	223
7.2	Introduction: Is an Observed Effect Real or Just Random Chance?	223
7.3	The Framework of Hypothesis Testing	225
7.4	The p-value: A Continuous Measure of Evidence	232
7.5	Constructing Statistical Tests	241
7.6	The Multiple Testing Problem: The Peril of Many Tests	250
7.7	NHST in Practice: A Critical View	257
7.8	Chapter Summary	258
8	Bayesian Inference and Statistical Decision Theory	263
8.1	Learning Objectives	263
8.2	Introduction: A Different Way of Thinking	263
8.3	The Bayesian Method: Updating Beliefs with Data	271
8.4	Bayesian Inference in Action	277
8.5	Statistical Decision Theory: A Framework for “Best”	283
8.6	Optimal Estimators: Bayes and Minimax Rules	288
8.7	Admissibility: Ruling Out Bad Estimators	291
8.8	Chapter Summary and Connections	292
9	Linear and Logistic Regression	297
9.1	Learning Objectives	297
9.2	Introduction: Why Linear Models Still Matter	297
9.3	Simple Linear Regression	300
9.4	Multiple Linear Regression	321
9.5	Logistic Regression	331
9.6	Chapter Summary and Connections	341

References	347
Download Complete PDF	349

Preface

Welcome to the lecture notes for the [Statistics for Data Science](#) (DATA11007) course offered by the [Master's Programme in Data Science](#) at the University of Helsinki.

These notes are designed to be an accessible companion to the course, providing multiple perspectives on statistical concepts for students with diverse backgrounds.

Note

Please refer to the official [Course Description](#) and Moodle page for specific information about current implementations of the course.

About These Notes

These lecture notes complement the main course textbook, *All of Statistics* by Larry Wasserman; Wasserman (2013). While Wasserman's book provides a comprehensive and rigorous treatment, these notes aim to:

- Provide gentler introductions to complex topics
- Offer multiple perspectives (intuitive, practical, mathematical, computational)
- Include extensive code examples in Python (and occasionally R)
- Bridge prerequisite gaps for students from different backgrounds

Formats and Feedback

These notes are available in two formats:

- [HTML](#) (recommended): Interactive version with tabs, code folding, and better navigation.
- [PDF](#): Traditional document format for printing or offline reading.

The content is almost identical in both formats. The PDF version is automatically generated from the source using Quarto's LaTeX conversion and we are aware it contains some formatting artifacts (e.g., additional page breaks), but this should not affect readability. Interactive code elements and end-of-chapter code blocks are not included in the PDF version. The HTML version of the notes is the officially recommended version.

Your Feedback is Welcome!

These lecture notes are new and under active development. We welcome feedback, corrections, and suggestions! If you spot any errors, unclear explanations, or have ideas for improvements, please let your instructor know. Your feedback will contribute directly to the development of these materials.

⚠ Work in Progress

These lecture notes are being developed throughout the course. While the initial chapters are currently available, additional chapters will be released progressively as they are prepared. Each chapter will be made available before its corresponding lecture, ensuring you have the materials when you need them. We appreciate your patience as we continue to build this comprehensive resource!

How to Use These Notes

Throughout these notes, you'll encounter sections with multiple perspectives on the same concept. These are presented in tabbed panels to help you explore different viewpoints:

Multiple Perspectives

Intuitive

This perspective provides conceptual understanding through analogies, visualizations, and real-world examples. It focuses on building intuition without heavy mathematical formalism.

Mathematical

This perspective explains concepts using mathematical language and notation. It may provide alternative formulations or connect to other mathematical ideas that some readers find clearer or more familiar.

Computational

This perspective shows how to implement and explore concepts through code. It includes simulations, numerical experiments, and practical algorithms.

Important: These perspectives often contain more advanced material than required for the course, or *teasers* for material which will be covered later. You're not expected to understand everything in every panel! Think of them as:

- **Additional resources** for deeper exploration
- **Different entry points** to the same concept
- **Teasers** for the content of future lectures
- **Optional enrichment** beyond the core material

Choose the perspective that best matches your learning style, or explore all three to build a richer understanding. The core course content is always presented in the main text outside these panels.

Example: Understanding Variance

Here's a brief example of how these perspectives work. Consider the concept of variance (Var), which measures how spread out data values are:

Multiple Perspectives

Intuitive

Variance measures how far data points typically are from their average.

Imagine test scores in two classes:

- Class A: Everyone scores between 75-85 (low variance)

- Class B: Scores range from 40-100 (high variance)
- Both might have the same average (say, 80), but Class B has much more spread. Variance captures this spread numerically.

Mathematical

The variance of a random variable X is defined as:

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

For a discrete distribution:

$$\text{Var}(X) = \sum_i (x_i - \mu)^2 \cdot \mathbb{P}(X = x_i)$$

Key properties:

- $\text{Var}(aX + b) = a^2 \text{Var}(X)$
- If X, Y are independent: $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

Computational

```

import numpy as np
import matplotlib.pyplot as plt

# Two datasets with same mean but different variance
class_a = [75, 78, 80, 82, 85, 77, 83, 79, 81, 80]
class_b = [40, 95, 100, 65, 85, 90, 45, 98, 70, 82]

print(f"Class A: mean = {np.mean(class_a):.1f}, variance = {np.var(class_a):.1f}")
print(f"Class B: mean = {np.mean(class_b):.1f}, variance = {np.var(class_b):.1f}")

# Visualize to see the spread
plt.figure(figsize=(7, 3))

# Use same bins for both histograms to ensure consistent bar width
bins = np.linspace(30, 110, 17) # 16 bins from 30 to 110

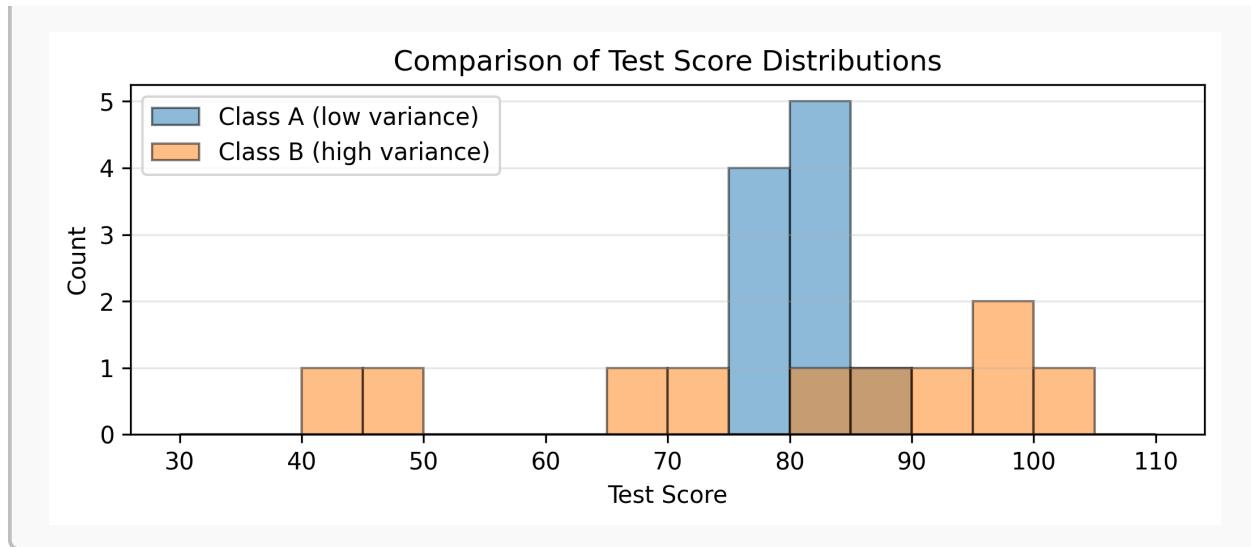
plt.hist(class_a, alpha=0.5, label='Class A (low variance)', bins=bins, edgecolor='black')
plt.hist(class_b, alpha=0.5, label='Class B (high variance)', bins=bins, edgecolor='black')
plt.xlabel('Test Score')
plt.ylabel('Count')
plt.title('Comparison of Test Score Distributions')
plt.legend(loc='upper left')

# Set y-axis to show only integers
plt.gca().yaxis.set_major_locator(plt.MaxNLocator(integer=True))

plt.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()

Class A: mean = 80.0, variance = 7.8
Class B: mean = 77.0, variance = 413.8

```



Notice how each perspective offers something different: intuition through examples, mathematical rigor through formulas, and hands-on exploration through code.

Prerequisites

The course assumes:

- Basic programming ability (Python or R)
- Calculus (derivatives and integrals)
- Linear algebra (matrices and vectors)
- Basic probability (sample spaces, events)

Don't worry if you're rusty on some topics – we'll review key concepts as needed.

Course Structure

1. **Foundations:** Probability, random variables, distributions
2. **Inference:** Estimation, confidence intervals, hypothesis testing
3. **Methods:** Regression, resampling, Bayesian approaches
4. **Advanced Topics:** Missing data, causal inference, experimental design

Available Chapters

The following chapters are currently available:

1. Probability Foundations
2. Expectation
3. Convergence and Statistical Inference
4. Nonparametric Methods and Bootstrap
5. Parametric Inference I: Finding Estimators
6. Parametric Inference II: Properties of Estimators
7. Hypothesis Testing and p-values
8. Bayesian Inference and Decision Theory
9. Linear and Logistic Regression

Programming

In this course, we'll mainly use Python with occasional usages of R. Python is the main language for modern machine learning, and both Python and R are widely used in data science. Feel free to choose the environment you're most comfortable with.

Acknowledgments

These notes build upon materials from previous course iterations and incorporate feedback from many students. Special thanks to the teaching assistants and students who have helped improve these materials.

The course content is largely adapted from Wasserman (2013), which remains the main course textbook.

The course slides and materials in early course iterations (2021-2024) were fully developed by Antti Honkela. Starting from 2025, we introduced the lecture notes that you are currently reading, which have been developed mainly by Luigi Acerbi expanding on the existing course slides and materials.

These notes were compiled with the assistance of [Gemini 2.5 Pro](#) and the [Claude 4](#) family of AI models, particularly when it comes to the [Quarto](#) layout, coding content, and brainstorming examples.

Last updated: 2025-09-07

Chapter 1

Probability Foundations

1.1 Learning Objectives

After completing this chapter, you will be able to:

- Explain the role of probability in data science and statistical modeling.
- Apply fundamental probability concepts, including sample spaces, events, and the axioms of probability.
- Calculate probabilities using independence, conditional probability, and Bayes' theorem.
- Distinguish between discrete and continuous random variables and use their distribution functions (PMF, PDF, CDF).
- Describe and apply key univariate and multivariate distributions (e.g., Binomial, Normal, Multinomial).

i Note

This chapter covers fundamental probability concepts essential for statistical inference. The material is adapted and expanded from Chapters 1 and 2 of Wasserman (2013), which interested readers are encouraged to consult directly for a more rigorous and comprehensive treatment.

1.2 Why Do We Need Statistics?

1.2.1 Statistics and Machine Learning in Data Science

Machine learning (ML) has revolutionized our ability to make predictions. Given enough training data, modern ML models can achieve remarkable accuracy on unseen data that resembles what they've been trained on. But there's a crucial limitation: these models excel when the new data comes from the same distribution as the training data.

How do we move beyond this constraint to make reliable predictions in the real world, where conditions change and data can be messy, incomplete, or collected differently than our training set?

This is where statistics becomes essential. Statistics provides the tools to:

- **Understand principles of data collection:** How was the data gathered? What biases might exist?
- **Plan data collection strategically:** Design experiments and surveys that yield meaningful insights
- **Deal with missing data:** Real-world data is rarely complete - we need principled ways to handle gaps
- **Understand causality in modeling:** Correlation isn't causation, and confounding variables can mislead us

Without these statistical foundations, even the most sophisticated ML models can fail spectacularly when deployed in practice.

1.2.2 Case Study: IBM Watson Health

The story of IBM Watson Health illustrates why statistical thinking is crucial for real-world AI applications.

In 2011, IBM Watson made headlines by defeating human champions on the quiz show *Jeopardy!* This victory showcased the power of natural language processing and knowledge retrieval. IBM saw an opportunity: if Watson could master general knowledge, why not train it to be a doctor?

Watson Health launched in 2015 with an ambitious goal: use data from top US hospitals to train an AI system that could diagnose and treat patients anywhere in the world. The vision was compelling - bring world-class medical expertise to underserved areas through AI.

Over the years, IBM:

- Spent over \$4 billion on acquisitions
- Employed 7,000 people developing the system
- Partnered with prestigious medical institutions

Yet by 2022, IBM sold Watson Health's data and assets for just \$1 billion - a massive loss. What went wrong?

The fundamental issue was **data representativeness**. Watson Health was trained on data from elite US hospitals treating specific patient populations. But this data didn't represent:

- Different healthcare systems and practices globally
- Diverse patient populations with varying genetics, lifestyles, and environmental factors
- Resource constraints in different settings
- Variations in how medical data is recorded and coded

This failure wasn't due to inadequate machine learning algorithms - it was a failure to apply statistical thinking about data collection, representation, and generalization. No amount of computational power can overcome fundamentally biased or unrepresentative data.

Read more [in this Slate article](#).

1.2.3 Two Cultures of Statistical Modeling

In his influential essay Breiman (2001), statistician Leo Breiman identified two distinct approaches to statistical modeling, each with different goals and philosophies. These are often cast as the two approaches of **prediction vs. explanation**.

Feature	The Algorithmic Modeling Culture	The Data Modeling Culture
Goal	Accurate prediction	Understanding mechanisms
Approach	Treat the mapping from inputs to outputs as a black box	Specify interpretable models that represent how nature works
Validation	Predictive accuracy on held-out data	Statistical tests, confidence intervals, model diagnostics
Philosophy	“It doesn’t matter how it works, as long as it works”	“We need to understand which factors matter and why”
Examples	Deep learning, random forests, boosting	Linear regression, causal inference, experimental design

Multiple Perspectives

Intuitive

Think of these two cultures like different approaches to cooking:

The **algorithmic approach** is like following a top-rated recipe - you don't know why you fold (not stir) the batter or why ingredients must be room temperature, but following the steps precisely often produces better results than many trained cooks achieve. You can pick 5-star recipes and succeed without any cooking knowledge.

The **data modeling approach** is like understanding food science - you know about Maillard reactions, gluten development, and emulsification. But translating this into a great dish is slow and complex. You might spend hours calculating optimal ratios only to produce something inferior to what a simple recipe would have given you.

This creates a fundamental tension: The recipe follower often produces better food faster. The food scientist understands why things work and with time might produce a breakthrough - but may struggle to match the empirical success of well-tested recipes. In machine learning, this same tension exists - a neural network might predict customer behavior better than any theory-based model, even if we don't understand why. Sometimes, letting algorithms find patterns empirically works better than imposing our theoretical understanding. However, understanding often gives us an edge to build better algorithms and generalize to novel scenarios.

Mathematical

Formally, both cultures can be seen as addressing the problem of characterizing a mapping:

$$X \rightarrow Y$$

where X represents **input** features and Y represents the **response**.

Algorithmic approach (example): Find a function \hat{f} that minimizes prediction error. A common approach is to find the function that minimizes the **mean squared error (MSE)**,

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^N (Y_n - \hat{f}(X_n))^2$$

that is make the squared difference between the actual outcome Y_n and the prediction $\hat{f}(X_n)$ as small as possible over the available training data. In practice, we often report the **root mean squared error (RMSE)** = $\sqrt{\text{MSE}}$, which has the same units as Y . We don't care about what the function \hat{f} looks like, as long as it minimizes this error.

Data modeling approach (example): Build a mechanistic model $Y = f(X; \theta) + \epsilon$ where:

- f has a specific, interpretable form
- θ are parameters with scientific, interpretable meaning
- ϵ represents random error

While fitting the model to data often still involves optimizing some objective, the goal here is to study the best-fitting parameters θ , or find the best model f among a set of competing hypotheses.

Computational

We compare here the two approaches represented by a **random forest** (RF) model and a **linear regression** model. The former represents a traditional machine learning approach, while the latter is a staple of statistical modelling.

A trained random forest model is harder to interpret, hence falls in the “algorithmic” camp for the purpose of this example. Conversely, a fitted linear regression model yields interpretable *weights* which directly tell us how the features linearly affect the response, so it represents the data modeling camp.

```

# Comparing algorithmic vs data modeling approaches
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import statsmodels.api as sm

# Load synthetic housing price data with complex patterns
# File available here:
# https://raw.githubusercontent.com/lacerbi/stats-for-ds-website/refs/heads/main/data/housing_prices.csv
df = pd.read_csv('../data/housing_prices.csv')

# Prepare features and target
features = ['size_sqft', 'bedrooms', 'age_years', 'location_score',
            'garage_spaces', 'has_pool', 'crime_rate', 'school_rating']
X = df[features]
y = df['price']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Dataset: {X.shape[0]}:, houses with {X.shape[1]} features")
print(f"Training on {len(X_train)}:, houses, testing on {len(X_test)}:,")
print(f"\nAverage house price: €{y.mean():,.0f}")
print(f"Price standard deviation: €{y.std():,.0f}")

# ALGORITHMIC APPROACH: Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# Calculate metrics
from sklearn.metrics import mean_squared_error
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_predictions))

print("\n== ALGORITHMIC APPROACH (Random Forest) ==")
print(f"Root Mean Squared Error (RMSE): €{rf_rmse:.0f}")
print("\nFeature Importances:")
for feature, importance in zip(features, rf_model.feature_importances_):
    print(f" {feature}: {importance:.3f}")

Dataset: 5,000 houses with 8 features
Training on 4,000 houses, testing on 1,000

Average house price: €439,750
Price standard deviation: €108,720

== ALGORITHMIC APPROACH (Random Forest) ==
Root Mean Squared Error (RMSE): €32,608

Feature Importances:
size_sqft: 0.125
bedrooms: 0.022

```

```

age_years: 0.022
location_score: 0.047
garage_spaces: 0.013
has_pool: 0.007
crime_rate: 0.018
school_rating: 0.746

# DATA MODELING APPROACH: Linear Regression
# Add constant term for intercept
X_train_lm = sm.add_constant(X_train)
X_test_lm = sm.add_constant(X_test)

# Fit linear model
lm_model = sm.OLS(y_train, X_train_lm)
lm_results = lm_model.fit()

# Make predictions
lm_predictions = lm_results.predict(X_test_lm)
lm_rmse = np.sqrt(mean_squared_error(y_test, lm_predictions))

print("\n==== DATA MODELING APPROACH (Linear Regression) ===")
print(f"Root Mean Squared Error (RMSE): {lm_rmse:.0f}")

# Show interpretable coefficients
print("\nLinear Model Coefficients:")
coef_df = pd.DataFrame({
    'Feature': ['Intercept'] + features,
    'Coefficient': lm_results.params,
    'Std Error': lm_results.bse,
    'P-value': lm_results.pvalues
})
coef_df['Significant'] = coef_df['P-value'] < 0.05
print(coef_df.to_string(index=False))

print("\n==== INTERPRETATION ===")
print("Linear model suggests:")
for i, feature in enumerate(features):
    coef = lm_results.params[i+1] # +1 to skip intercept
    if abs(coef) > 100:
        print(f"- Each unit increase in {feature}: {coef:.0f} change in price")
print("\nBUT: The model performs (slightly) worse than Random Forest.")
print(f"RF RMSE: {rf_rmse:.0f} vs Linear RMSE: {lm_rmse:.0f}")
print(f"That's {lm_rmse - rf_rmse:.0f} worse prediction error.")
print(f"Should we care more about prediction or understanding?")

==== DATA MODELING APPROACH (Linear Regression) ===
Root Mean Squared Error (RMSE): €34,061

Linear Model Coefficients:
      Feature   Coefficient   Std Error       P-value  Significant
Intercept -241058.748914  4060.095405  0.000000e+00      True
size_sqft      72.843101   1.113174  0.000000e+00      True
bedrooms     17505.976934  561.915079 6.369481e-191      True

```

age_years	66.884992	37.769905	7.666128e-02	False
location_score	6326.172511	191.797597	3.372091e-211	True
garage_spaces	15599.005835	602.273749	7.547388e-137	True
has_pool	29992.244032	1362.847287	2.110344e-101	True
crime_rate	-1268.321711	110.805186	7.133961e-30	True
school_rating	66814.033458	394.636343	0.000000e+00	True

==== INTERPRETATION ====

Linear model suggests:

- Each unit increase in bedrooms: €17,506 change in price
- Each unit increase in location_score: €6,326 change in price
- Each unit increase in garage_spaces: €15,599 change in price
- Each unit increase in has_pool: €29,992 change in price
- Each unit increase in crime_rate: €-1,268 change in price
- Each unit increase in school_rating: €66,814 change in price

BUT: The model performs (slightly) worse than Random Forest.

RF RMSE: €32,608 vs Linear RMSE: €34,061

That's €1,453 worse prediction error.

Should we care more about prediction or understanding?

Both approaches have their place in modern data science. The **algorithmic culture** has driven breakthroughs in areas like computer vision and natural language processing, where prediction accuracy is paramount. For example, large language models (LLMs) are massively large deep neural networks (pre)trained with the extremely simple objective of just “predicting the next word”¹ – without any attempt at *understanding* the underlying process.

The **data modeling** culture remains essential for scientific understanding, policy decisions, and any application where we need to know not just *what* will happen, but *why*. For LLMs, and in ML more broadly, this aspect is studied by the field of *interpretability* or “explainable ML” – trying to understand how modern ML models “think” and reach their conclusions.

1.3 Foundations of Probability

Probability provides the mathematical language for quantifying uncertainty. Before we can make statistical inferences or build predictive models, we need a solid foundation in probability theory.

i Finnish Terminology Reference

This course is taught internationally, but for Finnish-speaking students, here’s a reference table of key probability terms you may have encountered in your earlier studies:

English	Finnish	Context
Sample space	Perusjoukko, otosavaruus	The set of all possible outcomes
Event	Tapahtuma	A subset of the sample space
Probability distribution	Todennäköisyyssjakauma	Assignment of probabilities to events
Probability measure	Todennäköisyytsmitta	Mathematical function P satisfying axioms

¹More correctly, LLMs predict *tokens*, which are parts of words and other characters.

Independent	Riippumattomat	Events that don't influence each other
Conditional probability	Ehdollinen todennäköisyys	Probability given some information
Bayes' Theorem	Bayesin kaava	Formula for updating probabilities
Random variable	Satunnaismuuttuja	Function mapping outcomes to numbers
Cumulative distribution function (CDF)	Kertymäfunktio	$P(X \leq x)$
Discrete	Diskreetti	Taking countable values
Probability mass function (PMF)	Todennäköisyysmassafunktio	$P(X = x)$ for discrete X
Probability density function (PDF)	Tiheysfunktio	Density for continuous variables
Quantile function	Kvantilifunktio	Inverse of CDF
First quartile	Ensimmäinen kvartiili	25th percentile
Median	Mediaani	50th percentile
Joint density function	Yhteistihesfunktio	PDF for multiple variables
Marginal density	Reunatiheysfunktio	PDF of one variable from joint
Conditional density	Ehdollinen tiheysfunktio	PDF given another variable's value
Random vector	Satunnaisvektori	Vector of random variables
Independent and identically distributed (IID)	Riippumattomat ja samoin jakautuneet	Common assumption for data
Random sample	Satunnaisotos	IID observations from population
Frequentist probability	Frekventistinen todennäköisyys	Long-run frequency interpretation
Subjective probability	Subjektiivinen todennäköisyys	Degree of belief interpretation

Note: Some terms have multiple Finnish translations. We report here the most common ones.

1.3.1 Sample Spaces and Events

The **sample space** Ω is the set of all possible outcomes of an experiment. Individual elements $\omega \in \Omega$ are called **sample outcomes**, **realizations**, or just **elements**. Subsets of Ω are called **events**.

Notation: ω and Ω are the lowercase (respectively, uppercase) version of the Greek letter **omega**.

Example: Coin flips

If we flip a coin twice, where each outcome can be head (H) or tails (T), the sample space is:

$$\Omega = \{HH, HT, TH, TT\}$$

The event “first flip is heads” is $A = \{HH, HT\}$.

Example: Temperature measurement

When measuring temperature, the sample space might be the full set of real numbers:

$$\Omega = \mathbb{R} = (-\infty, \infty)$$

The event “temperature between 20°C and 25°C” is the interval $A = [20, 25]$.

Note that we often take Ω to be larger than strictly necessary - in this case for example we are including physically impossible values like -1000°C. This is still *mathematically* valid. As we will see later, we can assign zero probability to impossible events.

Notation: $[a, b]$ denotes the *interval* between a and b (included), whereas (a, b) is the interval between a and b (excluded).

Sample spaces can be:

- **Finite:** $\Omega = \{1, 2, 3, 4, 5, 6\}$ for a die roll
- **Countably infinite:** $\Omega = \{1, 2, 3, \dots\}$ for “number of flips until first heads”
- **Uncountably infinite:** $\Omega = [0, 1]$ for “random number between 0 and 1”

1.3.2 Set Operations for Events

Since events are sets, we can combine them using standard set operations:

Operation	Notation	Meaning
Complement	A^c	“not A” - all outcomes not in A
Union	$A \cup B$	“A or B” - outcomes in either A or B (or both)
Intersection	$A \cap B$	“A and B” - outcomes in both A and B
Difference	$A \setminus B$	Outcomes in A but not in B

i Note

Disjoint events: Events A and B are disjoint (or mutually exclusive) if $A \cap B = \emptyset$. This means they cannot occur simultaneously. For example, in the case of a standard six-sided die roll, let A = “rolling an even number” = $\{2, 4, 6\}$ and B = “rolling a 1” = $\{1\}$. These events are disjoint because you can’t roll both an even number AND a 1 simultaneously.

1.3.3 Probability Axioms

Now that we have defined the space of possible events, we can define the probability of an event. A probability measure must satisfy three fundamental axioms:

A function \mathbb{P} that assigns a real number $\mathbb{P}(A)$ to each event A is a **probability measure** if:

1. **Non-negativity:** $\mathbb{P}(A) \geq 0$ for every event A
2. **Normalization:** $\mathbb{P}(\Omega) = 1$
3. **Countable additivity:** If A_1, A_2, \dots are disjoint events, then:

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i)$$

i Why These Axioms

These axioms ensure probability respects intuitive laws:

1. **Non-negativity:** The probability of rain tomorrow cannot be negative.
2. **Normalization:** When rolling a six-sided die, the probability of getting one of the faces $\{1, 2, 3, 4, 5, 6\}$ is 1: 1 represent the total probability.
3. **Countable additivity:** The probability of rolling a 1 *or* a 2 on a die is the sum of their individual probabilities, as these events cannot happen together.

It turns out that under some assumptions, it can be shown that these axioms are exactly what you would pick if one wants to quantify the concept of “possibility of an event” with a single number – a result known as [Cox’s theorem](#).

From these axioms, we can derive many useful properties:

- $\mathbb{P}(\emptyset) = 0$ (the impossible event has probability 0)
- $\mathbb{P}(A^c) = 1 - \mathbb{P}(A)$ (complement rule)
- If $A \subset B$, then $\mathbb{P}(A) \leq \mathbb{P}(B)$ (monotonicity)
- $0 \leq \mathbb{P}(A) \leq 1$ for any event A

For any events A and B :

$$\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$$

Why? This formula accounts for the “double counting” when we add $\mathbb{P}(A)$ and $\mathbb{P}(B)$ – the intersection $A \cap B$ gets counted twice, so we subtract it once.

i Note

Sometimes you will see the notation $\mathbb{P}(AB)$ to denote $\mathbb{P}(A \cap B)$.

1.3.4 Interpretations of Probability

Probability can be understood from different philosophical perspectives, each leading to the same mathematical framework.

Multiple Perspectives

Intuitive

There are two main ways to think about what probability means:

Frequency interpretation: Probability is the long-run proportion of times an event occurs in repeated experiments. If we flip a fair coin millions of times, we expect heads about 50% of the time.

Subjective interpretation: Probability represents a degree of belief. When a weather forecaster says “30% chance of rain,” they’re expressing their confidence based on available information.

Both interpretations are useful, and both lead to the same mathematical rules.

Mathematical

Frequentist probability:

$$\mathbb{P}(A) = \lim_{n \rightarrow \infty} \frac{\text{number of times A occurs in n trials}}{n}$$

This requires the experiment to be repeatable under identical conditions.

Subjective/Bayesian probability: $\mathbb{P}(A)$ quantifies an agent’s degree of belief that A is true, constrained by:

- Coherence: beliefs must satisfy the probability axioms
- Updating: beliefs change rationally when new information arrives (via Bayes’ theorem)

Computational

Let’s simulate the **frequentist interpretation** by flipping a fair coin many times and tracking how the proportion of heads converges to the true probability of 0.5. This directly demonstrates the mathematical definition: probability as the long-run proportion.

```

import numpy as np
import matplotlib.pyplot as plt

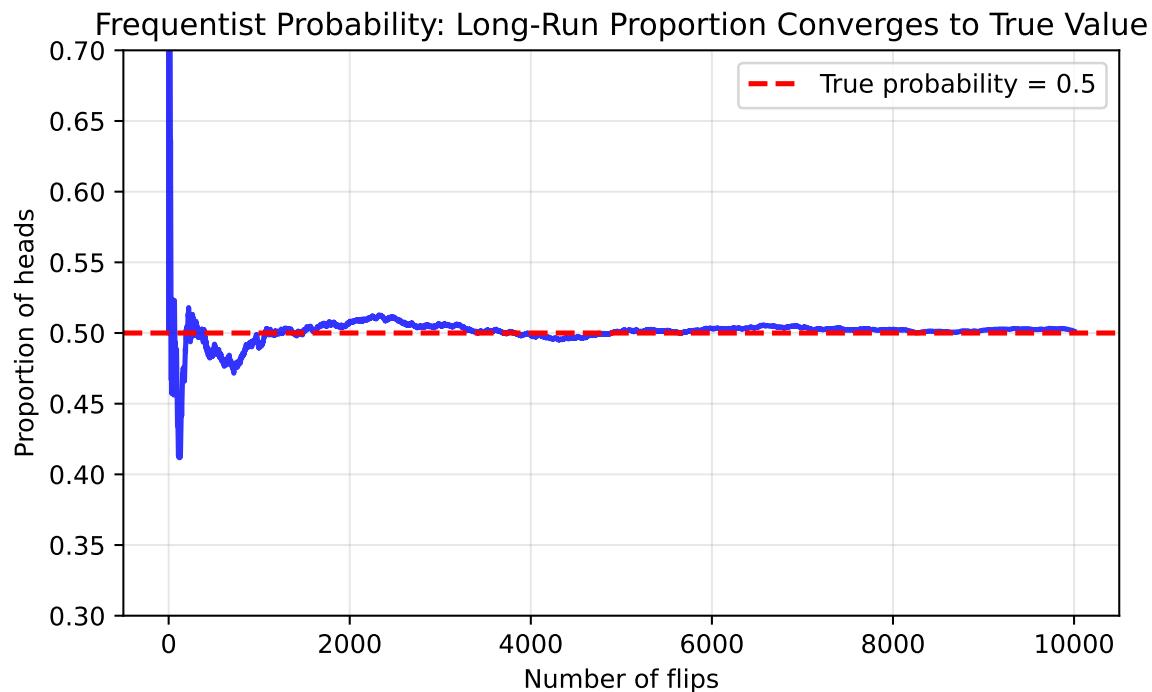
# Simulate many coin flips to see frequentist convergence
np.random.seed(42)
n_flips = 10000
flips = np.random.choice(['H', 'T'], size=n_flips)

# Calculate running proportion of heads
heads_count = np.cumsum(flips == 'H')
proportions = heads_count / np.arange(1, n_flips + 1)

# Plot convergence to true probability
plt.figure(figsize=(7, 4))
plt.plot(proportions, linewidth=2, alpha=0.8, color='blue')
plt.axhline(y=0.5, color='red', linestyle='--', label='True probability = 0.5', linewidth=2)
plt.xlabel('Number of flips')
plt.ylabel('Proportion of heads')
plt.title('Frequentist Probability: Long-Run Proportion Converges to True Value')
plt.legend()
plt.grid(True, alpha=0.3)
plt.ylim(0.3, 0.7)
plt.show()

# Print summary
print(f"After {n_flips:,} flips:")
print(f"Proportion of heads: {proportions[-1]:.4f}")
print(f"Deviation from 0.5: {abs(proportions[-1] - 0.5):.4f}")

```



Proportion of heads: 0.5013

Deviation from 0.5: 0.0013

The **subjective/Bayesian interpretation** involves updating beliefs based on evidence. We'll explore this computational approach in detail when we cover Bayes' theorem.

1.3.5 Finite Sample Spaces and Counting

When Ω is finite and all outcomes are equally likely, probability calculations reduce to counting:

$$\mathbb{P}(A) = \frac{|A|}{|\Omega|} = \frac{\text{number of outcomes in A}}{\text{total number of outcomes}}$$

Example: Rolling two dice

What's the probability the sum of rolling two six-sided dice equals 7?

i Solution

$\Omega = \{(i, j) : i, j \in \{1, 2, 3, 4, 5, 6\}\}$, so $|\Omega| = 36$.

The event "sum equals 7" is $A = \{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}$.

Therefore $\mathbb{P}(A) = \frac{6}{36} = \frac{1}{6}$.

Key counting principle - Binomial Coefficient:

The **binomial coefficient** (read as "n choose k") is:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Where $n!$ denotes the **factorial**, e.g. $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.

The binomial coefficient² counts the number of ways to choose k objects from n objects when order doesn't matter. For example:

- $\binom{5}{2} = \frac{5!}{2!3!} = \frac{5 \times 4}{2 \times 1} = 10$ ways to choose 2 items from 5
- Choosing 2 students from a class of 30: $\binom{30}{2} = 435$ possible pairs

1.4 Independence and Conditional Probability

1.4.1 Independent Events

Two events A and B are **independent** if:

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$$

We denote this as $A \perp\!\!\!\perp B$. When events are not independent, we write $A \not\perp\!\!\!\perp B$.

Independence means that knowing whether one event occurred tells us *nothing* about the other event.

²The name "binomial" comes from its appearance in the binomial theorem: $(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$.

i Note

The textbook uses non-standard notation for independence and non-independence. We use the standard notation $A \perp\!\!\!\perp B$ (and $A \not\perp\!\!\!\perp B$ for dependence), which is widely adopted in probability and statistics literature.

Example: Fair coin tosses

A fair coin is tossed twice. Let H_1 = “first toss is heads” and H_2 = “second toss is heads”. Are these two events independent?

i Solution

- $\mathbb{P}(H_1) = \mathbb{P}(H_2) = \frac{1}{2}$
- $\mathbb{P}(H_1 \cap H_2) = \mathbb{P}(\text{both heads}) = \frac{1}{4}$
- Since $\frac{1}{4} = \frac{1}{2} \times \frac{1}{2}$, the events are independent

This matches the intuition – whether we obtain head on the first flip does not tell us anything about the second flip, and vice versa.

Example: Dependent events

Draw two cards from a deck without replacement.

- A = “first card is an ace”
- B = “second card is an ace”

Are these events independent?

i Solution

- $\mathbb{P}(A) = \mathbb{P}(B) = \frac{4}{52}$
- However: $\mathbb{P}(A \cap B) = \frac{4}{52} \times \frac{3}{51} \neq \mathbb{P}(A)\mathbb{P}(B)$

The events are *dependent* because drawing an ace first changes the probability of drawing an ace second.

⚠ Warning

Common misconception: Disjoint events are NOT independent!

If A and B are disjoint with positive probability, then:

- $\mathbb{P}(A \cap B) = 0$ (since they are disjoint, they can't happen together)
- $\mathbb{P}(A)\mathbb{P}(B) > 0$ (if both have positive probability)

So disjoint events are actually maximally dependent - if one occurs, the other definitely doesn't!

1.4.2 Conditional Probability

The **conditional probability** of A given B is:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

provided $\mathbb{P}(B) > 0$.

Think of $\mathbb{P}(A|B)$ as the probability of A in the “new universe” where we know B has occurred.

 Caution

Prosecutor's Fallacy: Confusing $\mathbb{P}(A|B)$ with $\mathbb{P}(B|A)$.

These can be vastly different! For example:

- $\mathbb{P}(\text{match}|\text{guilty})$ might be 0.98.
- $\mathbb{P}(\text{guilty}|\text{match})$ might be 0.04.

The second depends on the prior probability of guilt and how many innocent people might also match.
We will see next how to compute one from the other.

1.4.3 Bayes' Theorem

Sometimes we know $\mathbb{P}(B|A)$ but we are really interested in the other way round, $\mathbb{P}(A|B)$.

For example, in the example above, we may know the probability that a test gives a match if the suspect is guilty, $\mathbb{P}(\text{match} | \text{guilty})$, but what we really want to know is the probability that the suspect is guilty given that we find a match, $\mathbb{P}(\text{guilty} | \text{match})$.

Such “inverse” conditional probabilities can be calculated via **Bayes’ theorem**.

For events A and B with $\mathbb{P}(B) > 0$:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

Where B is some information (evidence) and A an hypothesis.

If A_1, \dots, A_k partition Ω (they’re disjoint and cover all possibilities), then:

$$\mathbb{P}(B) = \sum_{i=1}^k \mathbb{P}(B|A_i)\mathbb{P}(A_i)$$

Combining these gives the full form of Bayes’ theorem:

$$\mathbb{P}(A_i|B) = \frac{\mathbb{P}(B|A_i)\mathbb{P}(A_i)}{\sum_j \mathbb{P}(B|A_j)\mathbb{P}(A_j)}$$

Terminology:

- $\mathbb{P}(A_i)$: Prior probability for hypothesis A_i (before seeing evidence B), also known as “base rate”.
- $\mathbb{P}(A_i|B)$: Posterior probability (after seeing evidence B).
- $\mathbb{P}(B|A_i)$: Likelihood of hypothesis A_i for fixed evidence B .

Example: Email spam detection

- Prior: 70% of emails are spam
- “Free” appears in 90% of spam emails
- “Free” appears in 1% of legitimate emails

If an email contains “free”, what’s the probability it’s spam?

i Solution

Let S = “email is spam” and F = “email contains ‘free’”.

Given:

- $\mathbb{P}(S) = 0.7$
- $\mathbb{P}(F|S) = 0.9$

- $\mathbb{P}(F|S^c) = 0.01$

By Bayes’ theorem:

$$\mathbb{P}(S|F) = \frac{0.9 \times 0.7}{0.9 \times 0.7 + 0.01 \times 0.3} = \frac{0.63}{0.633} \approx 0.995$$

So an email containing “free” has a 99.5% chance of being spam!

Multiple Perspectives

Intuitive

Conditional probability answers questions like:

- What’s the probability of rain given that it’s cloudy?
- What’s the probability a patient has a disease given a positive test result?
- What’s the probability a user clicks an ad given they’re on a mobile device?

The key insight: additional information changes probabilities. Knowing that B occurred restricts our attention to outcomes where B is true, potentially changing how likely A becomes.

Some conditional probabilities are easier to compute than others. For example, we may know that if the patient has a disease, then the test will return positive with a certain probability. However, to compute the “inverse” probability (if the test is positive, what’s the probability of the patient having the disease?) we need Bayes’ theorem.

Mathematical

For fixed B with $\mathbb{P}(B) > 0$, the conditional probability $\mathbb{P}(\cdot|B)$ is itself a probability measure:

1. $\mathbb{P}(A|B) \geq 0$ for all A
2. $\mathbb{P}(\Omega|B) = 1$
3. If A_1, A_2, \dots are disjoint, then $\mathbb{P}(\bigcup_i A_i|B) = \sum_i \mathbb{P}(A_i|B)$

Key relationships:

- If $A \perp\!\!\!\perp B$, then $\mathbb{P}(A|B) = \mathbb{P}(A)$ (independence means conditioning doesn’t matter)
- $\mathbb{P}(A \cap B) = \mathbb{P}(A|B)\mathbb{P}(B) = \mathbb{P}(B|A)\mathbb{P}(A)$ (multiplication rule)

Conversely, in general $\mathbb{P}(A|\cdot)$ (the likelihood) is *not* a probability measure.

Computational

We will visualize how conditional probability can be counterintuitive. We’ll simulate a medical test scenario to show how base rates affect our interpretation of test results.

```

import numpy as np
import matplotlib.pyplot as plt

# Medical test scenario
p_disease = 0.001                      # 0.1% have the disease (base rate)
p_pos_given_disease = 0.99                # 99% sensitivity
p_neg_given_healthy = 0.99                # 99% specificity

# Calculate probability of positive test
p_healthy = 1 - p_disease
p_pos_given_healthy = 1 - p_neg_given_healthy
p_positive = p_pos_given_disease * p_disease + p_pos_given_healthy * p_healthy

# Apply Bayes' theorem: P(disease|positive)
p_disease_given_pos = (p_pos_given_disease * p_disease) / p_positive

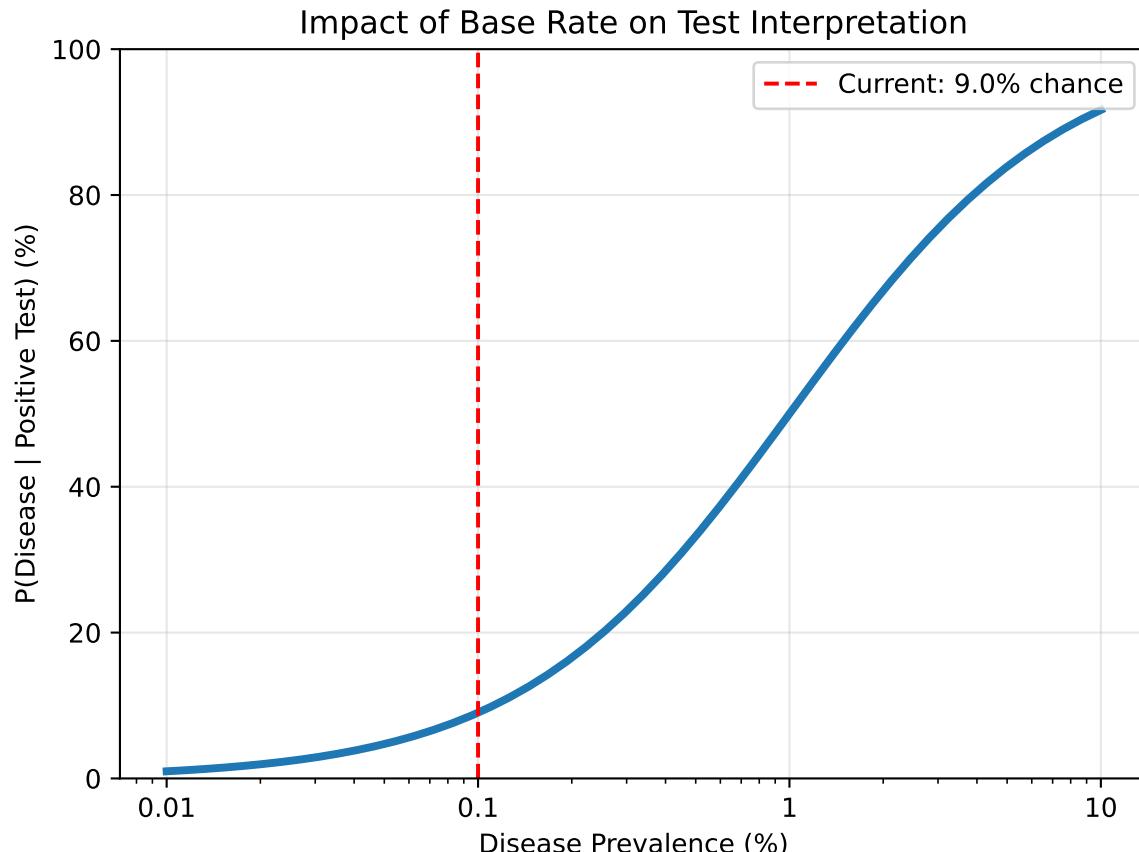
# Visualize with different base rates
base_rates = np.logspace(-4, -1, 50)  # 0.01% to 10%
posterior_probs = []

for base_rate in base_rates:
    p_pos = p_pos_given_disease * base_rate + p_pos_given_healthy * (1 - base_rate)
    posterior = (p_pos_given_disease * base_rate) / p_pos
    posterior_probs.append(posterior)

plt.figure(figsize=(7, 5))
plt.semilogx(base_rates * 100, np.array(posterior_probs) * 100, linewidth=3)
plt.axvline(x=0.1, color='red', linestyle='--', label=f'Current: {p_disease_given_pos:.1%} chance')
plt.xlabel('Disease Prevalence (%)')
plt.ylabel('P(Disease | Positive Test) (%)')
plt.title('Impact of Base Rate on Test Interpretation')
plt.ylim(0, 100)  # Set y-axis range from 0 to 100
plt.xticks([0.01, 0.1, 1, 10], ['0.01', '0.1', '1', '10'])  # Set custom x-axis ticks
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()

print(f"With 99% accurate test and 0.1% base rate:")
print(f"P(disease | positive test) = {p_disease_given_pos:.1%}")
print(f"Surprising: A positive test means only ~9% chance of disease!")

```



With 99% accurate test and 0.1% base rate:

$$P(\text{disease} \mid \text{positive test}) = 9.0\%$$

Surprising: A positive test means only ~9% chance of disease!

1.4.4 Classic Probability Examples

Let's work through some classic examples that illustrate key concepts:

Example: At least one head in 10 flips

What's the probability of getting at least one head in 10 coin flips?

Hint: Instead of counting all the ways to get 1, 2, ..., or 10 heads, use the complement.

i Solution

$$\mathbb{P}(\text{at least one head}) = 1 - \mathbb{P}(\text{no heads}) = 1 - \mathbb{P}(\text{all tails})$$

$$\text{Since flips are independent: } \mathbb{P}(\text{all tails}) = \left(\frac{1}{2}\right)^{10} = \frac{1}{1024}$$

$$\text{Therefore: } \mathbb{P}(\text{at least one head}) = 1 - \frac{1}{1024} \approx 0.999$$

Example (advanced): Basketball competition

Two players take turns shooting. Player A shoots first with probability $1/3$ of scoring. Player B shoots second with probability $1/4$. First to score wins. What's the probability A wins?

Solution

A wins if:

- A scores on first shot: probability $1/3$
- Both miss, then A scores: $(2/3)(3/4)(1/3)$
- Both miss twice, then A scores: $(2/3)(3/4)(2/3)(3/4)(1/3)$
- ...

This is a [geometric series](#):

$$\mathbb{P}(A \text{ wins}) = \frac{1}{3} \sum_{k=0}^{\infty} \left(\frac{2}{3} \cdot \frac{3}{4} \right)^k = \frac{1}{3} \cdot \frac{1}{1 - \frac{1}{2}} = \frac{2}{3}$$

1.5 Random Variables

So far, we've worked with events - subsets of the sample space. But in practice, we usually care about numerical quantities associated with random outcomes. This is where random variables come in.

1.5.1 Definition and Intuition

A **random variable** is a function $X : \Omega \rightarrow \mathbb{R}$ that assigns a real number to each outcome in the sample space.

A random variable is defined by its possible *values* (real numbers) and their *probabilities*.

In the case of a *discrete* random variable, the set of values is discrete (finite or infinite), x_1, \dots , and each value can be assigned a corresponding point probability p_1, \dots with $0 \leq p_i \leq 1$, $\sum_{i=1}^{\infty} p_i = 1$.

In the case of a continuous random variable, probabilities are defined by a non-negative probability density function that integrates to 1.

Multiple Perspectives

Intuitive

A random variable is just a way to assign numbers to outcomes. Think of it as a measurement or quantity that depends on chance.

Examples:

- Number of heads in 10 coin flips
- Time until next customer arrives
- Temperature at noon tomorrow
- Stock price at market close

The key insight: once we have numbers, we can do arithmetic, calculate averages, measure spread, and use all the tools of mathematics.

Mathematical

Warning: The following will likely make sense only if you have taken an advanced course in probability theory or measure theory. Feel free to skip it otherwise.

Formally, X is a measurable function from (Ω, \mathcal{F}) to $(\mathbb{R}, \mathcal{B})$ where:

- \mathcal{F} is the σ -algebra of events in Ω
- \mathcal{B} is the Borel σ -algebra on \mathbb{R}

Measurability means: for any Borel set $B \subset \mathbb{R}$, the pre-image $X^{-1}(B) = \{\omega : X(\omega) \in B\}$ is an event in \mathcal{F} .

This technical condition ensures we can compute probabilities like $\mathbb{P}(X \in B)$.

Computational

Here we demonstrate how random variables map outcomes to numbers, allowing us to analyze randomness mathematically. For example, we can plot a histogram for the realizations over multiple experiments.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Demonstrate a random variable: X = number of heads in 10 coin flips
np.random.seed(42)

# Single experiment
flips = np.random.choice(['H', 'T'], size=10)
X = np.sum(flips == 'H')
print(f"Outcomes (single experiment): {flips}")
print(f"X (number of heads) = {X}")

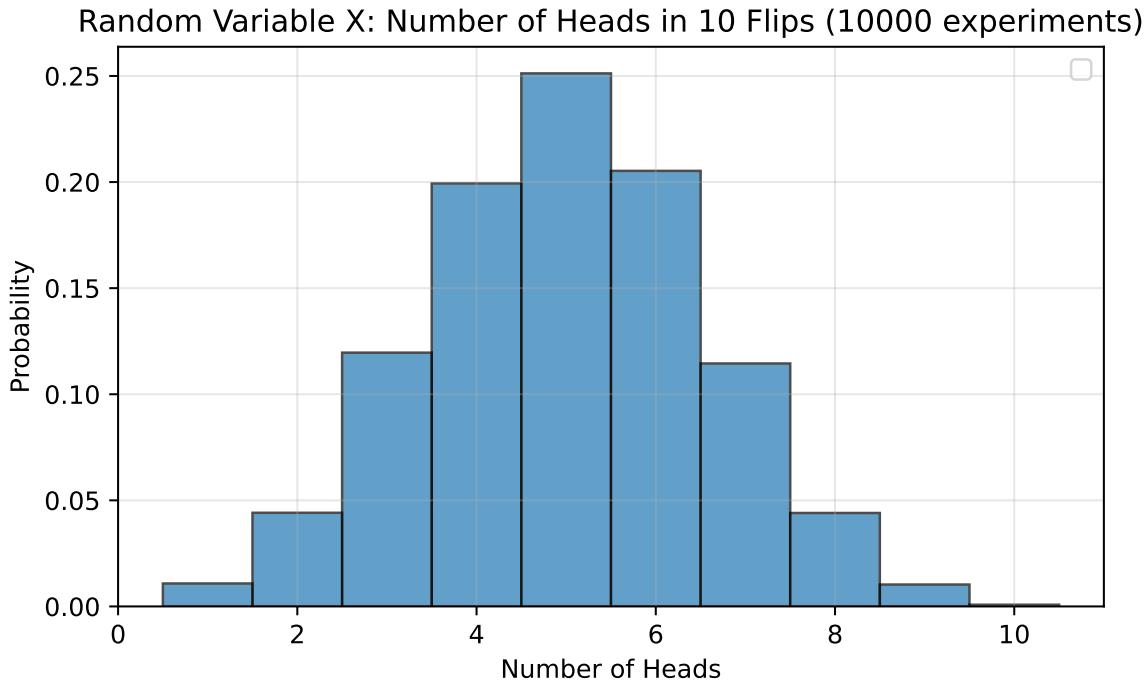
# Simulate many experiments to see the distribution
n_sims = 10000
X_values = [np.sum(np.random.choice(['H', 'T'], size=10) == 'H')
            for _ in range(n_sims)]

# Visualize distribution
plt.figure(figsize=(7, 4))
counts, bins, _ = plt.hist(X_values, bins=np.arange(0.5, 11.5, 1), density=True,
                           alpha=0.7, edgecolor='black')

x = np.arange(0, 11)
plt.xlabel('Number of Heads')
plt.ylabel('Probability')
plt.title(f'Random Variable X: Number of Heads in 10 Flips ({n_sims} experiments)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

print(f"\nAverage value: {np.mean(X_values):.3f} (theoretical: 5.0)")
Outcomes (single experiment): ['H' 'T' 'H' 'H' 'H' 'H' 'T' 'H' 'H' 'H' 'H' 'T']
X (number of heads) = 7

Average value: 4.993 (theoretical: 5.0)
```



Example: Coin flips

Within the same sample space we can define multiple distinct random variables.

For example, let $\Omega = \{HH, HT, TH, TT\}$ (two flips). Define:

- X = number of heads
- Y = 1 if first flip is heads, 0 otherwise
- Z = 1 if flips match, 0 otherwise

Then:

- $X(HH) = 2, X(HT) = 1, X(TH) = 1, X(TT) = 0$
- $Y(HH) = 1, Y(HT) = 1, Y(TH) = 0, Y(TT) = 0$
- $Z(HH) = 1, Z(HT) = 0, Z(TH) = 0, Z(TT) = 1$

Tip

Notation convention:

- Capital letters (X, Y, Z) denote random variables
- Lowercase letters (x, y, z) denote specific values
- $\{X = x\}$ is the event that X takes value x

However, do not expect people to strictly follow this convention beyond mathematical and statistical textbooks. In the real world, you will often see “ x ” used to refer both to a *value* and to a random variable “ X ” that happens to take value x .

1.5.2 Cumulative Distribution Functions

The Cumulative Distribution Function (CDF) completely characterizes a random variable’s probability distribution.

The **cumulative distribution function (CDF)** of a random variable X is the function $F_X(x) : \mathbb{R} \rightarrow [0, 1]$ defined by

$$F_X(x) = \mathbb{P}(X \leq x)$$

for all $x \in \mathbb{R}$.

Example: Two coin flips

Let X = number of heads for two flips of fair coins.

- $\mathbb{P}(X = 0) = 1/4$
- $\mathbb{P}(X = 1) = 1/2$
- $\mathbb{P}(X = 2) = 1/4$

The CDF is:

$$F_X(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1/4 & \text{if } 0 \leq x < 1 \\ 3/4 & \text{if } 1 \leq x < 2 \\ 1 & \text{if } x \geq 2 \end{cases}$$

Note: The CDF is defined for ALL real x , even though X only takes values 0, 1, 2!

```

import matplotlib.pyplot as plt
import numpy as np

# Define the CDF values
x_jumps = [0, 1, 2] # Points where jumps occur
cdf_values = [0.25, 0.75, 1.0] # CDF values after jumps
cdf_values_before = [0, 0.25, 0.75] # CDF values before jumps

fig, ax = plt.subplots(figsize=(7, 4))

# Plot the step function
# Left segment (x < 0)
ax.hlines(0, -1, 0, colors='black', linewidth=2)

# Plot each segment
for i in range(len(x_jumps)):
    # Horizontal line segment
    if i < len(x_jumps) - 1:
        ax.hlines(cdf_values[i], x_jumps[i], x_jumps[i+1], colors='black', linewidth=2)
    else:
        # Last segment extends to the right
        ax.hlines(cdf_values[i], x_jumps[i], 3, colors='black', linewidth=2)

# Open circles (at discontinuities, left endpoints)
if i > 0:
    ax.plot(x_jumps[i], cdf_values_before[i], 'o', color='black',
            markerfacecolor='white', markersize=8, markeredgewidth=2)

# Filled circles (at jump points, right endpoints)
ax.plot(x_jumps[i], cdf_values[i], 'o', color='black',
        markerfacecolor='black', markersize=8)

# Open circle at x=0, y=0
ax.plot(0, 0, 'o', color='black', markerfacecolor='white',
        markersize=8, markeredgewidth=2)

# Set axis properties
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('$F_X(x)$', fontsize=12)
ax.set_xlim(-1, 3)
ax.set_ylim(-0.1, 1.1)

# Set tick marks
ax.set_xticks([0, 1, 2])
ax.set_yticks([0.25, 0.50, 0.75, 1.0])

# Add grid
ax.grid(True, alpha=0.3)

# Add arrows to axes
ax.annotate('', xy=(3.2, 0), xytext=(3, 0),
            arrowprops=dict(arrowstyle='->', color='black', lw=1))
ax.annotate('', xy=(0, 1.15), xytext=(0, 1.1),
            arrowprops=dict(arrowstyle='->', color='black', lw=1))

plt.tight_layout()
plt.show()

```

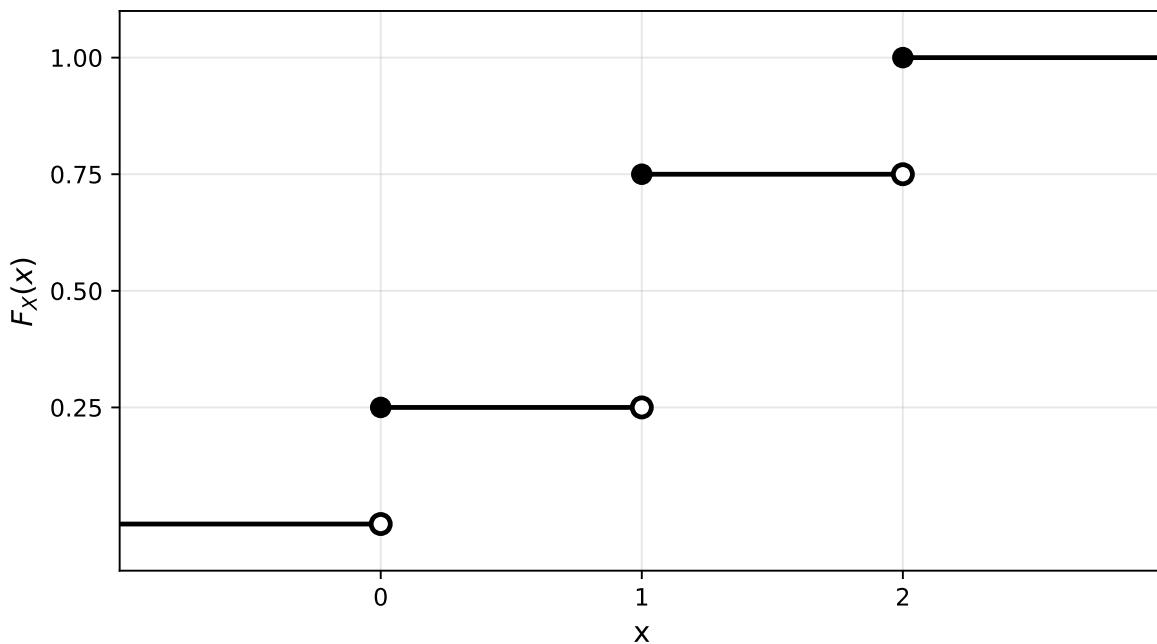


Figure 1.1: Cumulative distribution function (CDF) for the number of heads when flipping a coin twice.

1.5.3 Discrete Random Variables

A random variable X is **discrete** if it takes countably many values $\{x_1, x_2, \dots\}$. Its **probability mass function (PMF)** (sometimes just **probability function**) is defined as:

$$f_X(x) = \mathbb{P}(X = x)$$

Properties of PMFs:

- $f_X(x) \geq 0$ for all x
- $\sum_i f_X(x_i) = 1$ (probabilities sum to 1)
- $F_X(x) = \sum_{x_i \leq x} f_X(x_i)$ (CDF is sum of PMF)

```
# PMF for coin flipping example
x_values = [0, 1, 2]
pmf_values = [0.25, 0.5, 0.25]

fig, ax = plt.subplots(figsize=(7, 4))

# Plot vertical lines from x-axis to probability values
for x, p in zip(x_values, pmf_values):
    ax.plot([x, x], [0, p], 'k-', linewidth=2)
    # Add filled circles at the top
    ax.plot(x, p, 'ko', markersize=8, markerfacecolor='black')

# Set axis properties
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('$f_X(x)$', fontsize=12)
ax.set_xlim(-0.5, 2.5)
```

```

ax.set_ylim(0, 1)

# Set tick marks
ax.set_xticks([0, 1, 2])
ax.set_yticks([0.25, 0.5, 0.75, 1])

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

# Add a horizontal line at y=0 for clarity
ax.axhline(y=0, color='black', linewidth=0.5)

plt.tight_layout()
plt.show()

```

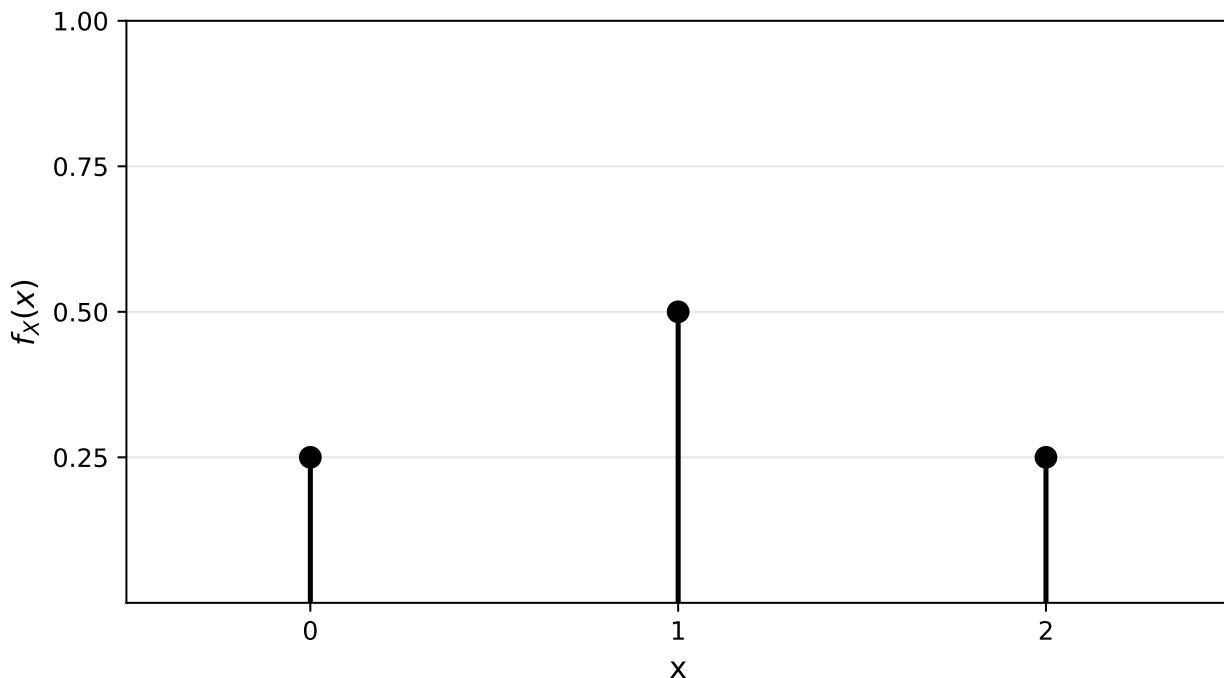


Figure 1.2: Probability mass function (PMF) for the number of heads when flipping a coin twice.

1.5.4 Core Discrete Distributions

i Note

Notation preview: We'll use $\mathbb{E}[X]$ to denote the *expected value* (mean) of a random variable X , and $\text{Var}(X)$ or σ^2 for its *variance* (a measure of spread). These concepts will be covered in detail in Chapter 2 of the lecture notes.

1.5.4.1 Bernoulli Distribution

The **Bernoulli distribution** is the simplest non-trivial random variable – a single binary outcome with probability $p \in [0, 1]$ of happening.

$X \sim \text{Bernoulli}(p)$ if:

$$f_X(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

An outcome of $X = 1$ is often referred to as a “hit” or a “success”, while $X = 0$ is a “miss” or a “failure”.

Use cases:

- Coin flip (heads/tails)
 - If $p \neq 0.5$, this is known as a *biased* coin (as opposed to a *fair* coin with $p = 0.5$)
 - Here what constitutes a “hit” and a “miss” is arbitrary!
- Success/failure of a single trial
- Binary classification (spam/not spam)
- User clicks/doesn’t click an ad

```
# Bernoulli distribution PMF
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import bernoulli

# Parameter
p = 0.3 # probability of success

# PMF visualization
fig, ax = plt.subplots(figsize=(7, 4))

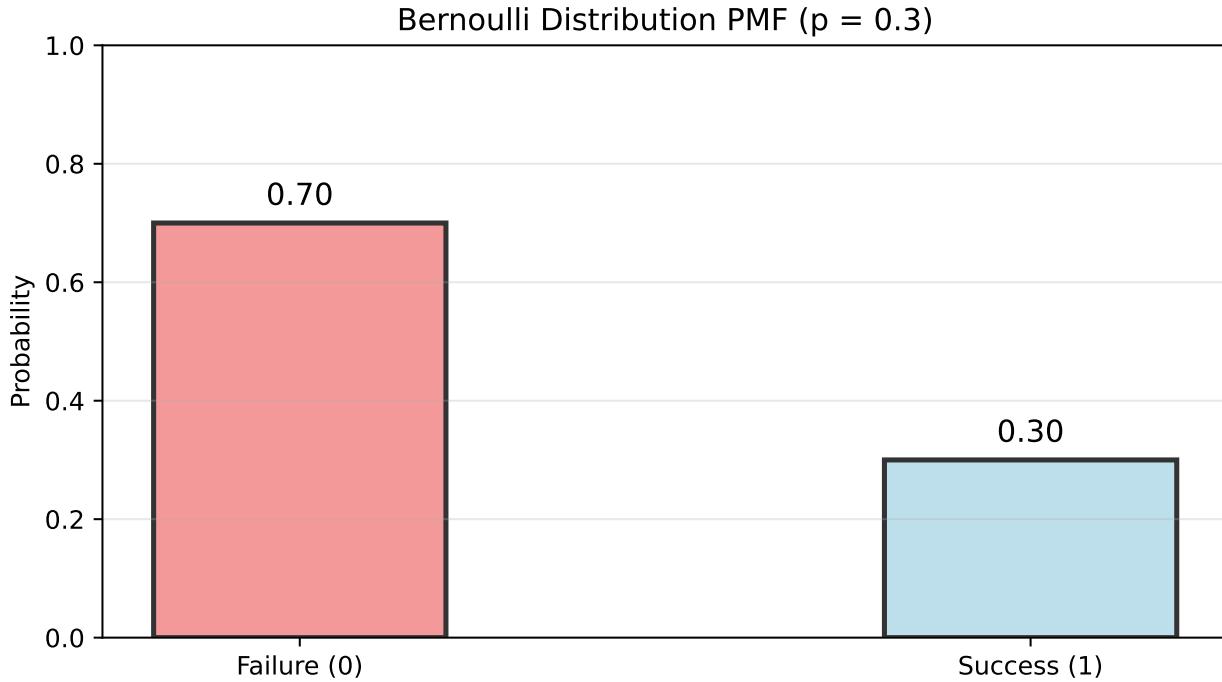
# Plot PMF
x = [0, 1]
pmf = [1-p, p]
bars = ax.bar(x, pmf, width=0.4, alpha=0.8, color=['lightcoral', 'lightblue'],
               edgecolor='black', linewidth=2)

# Add value labels
for i, (xi, pi) in enumerate(zip(x, pmf)):
    ax.text(xi, pi + 0.02, f'{pi:.2f}', ha='center', va='bottom', fontsize=12)

ax.set_xticks([0, 1])
ax.set_xticklabels(['Failure (0)', 'Success (1)'])
ax.set_ylabel('Probability')
ax.set_title(f'Bernoulli Distribution PMF (p = {p})')
ax.set_ylim(0, 1)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print(f"E[X] = p = {p}")
print(f"Var(X) = p(1-p) = {p*(1-p):.3f}")
```



$$E[X] = p = 0.3$$

$$\text{Var}(X) = p(1-p) = 0.210$$

1.5.4.2 Binomial Distribution

The [binomial distribution](#) counts the number of successes in a fixed number n of independent Bernoulli trials each with probability p .

$X \sim \text{Binomial}(n, p)$ if:

$$f_X(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, \dots, n$$

Key properties:

- Sum of independent Bernoullis: If $X_i \sim \text{Bernoulli}(p)$ are independent, then $\sum_{i=1}^n X_i \sim \text{Binomial}(n, p)$
- Additivity: If $X \sim \text{Binomial}(n_1, p)$ and $Y \sim \text{Binomial}(n_2, p)$ are independent, then $X + Y \sim \text{Binomial}(n_1 + n_2, p)$

Use cases:

- Number of heads in n coin flips
- Number of defective items in a batch
- Number of customers who make a purchase
- Number of successful treatments in a clinical trial



Warning

Independence assumption: The binomial distribution assumes all trials are independent - each outcome does not affect the probability of subsequent outcomes. This assumption may not hold in practice!

For example, if items are defective because a machine has broken (rather than random variation), then finding one defective item suggests all subsequent items might also be defective. In such cases, the binomial distribution would be inappropriate.

```
# Binomial distribution visualization
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

# Parameters
n, p = 20, 0.3
x = np.arange(0, n+1)

# Create figure
fig, ax = plt.subplots(figsize=(7, 5))

# Plot PMF
pmf = binom.pmf(x, n, p)
bars = ax.bar(x, pmf, alpha=0.8, color='steelblue', edgecolor='black')

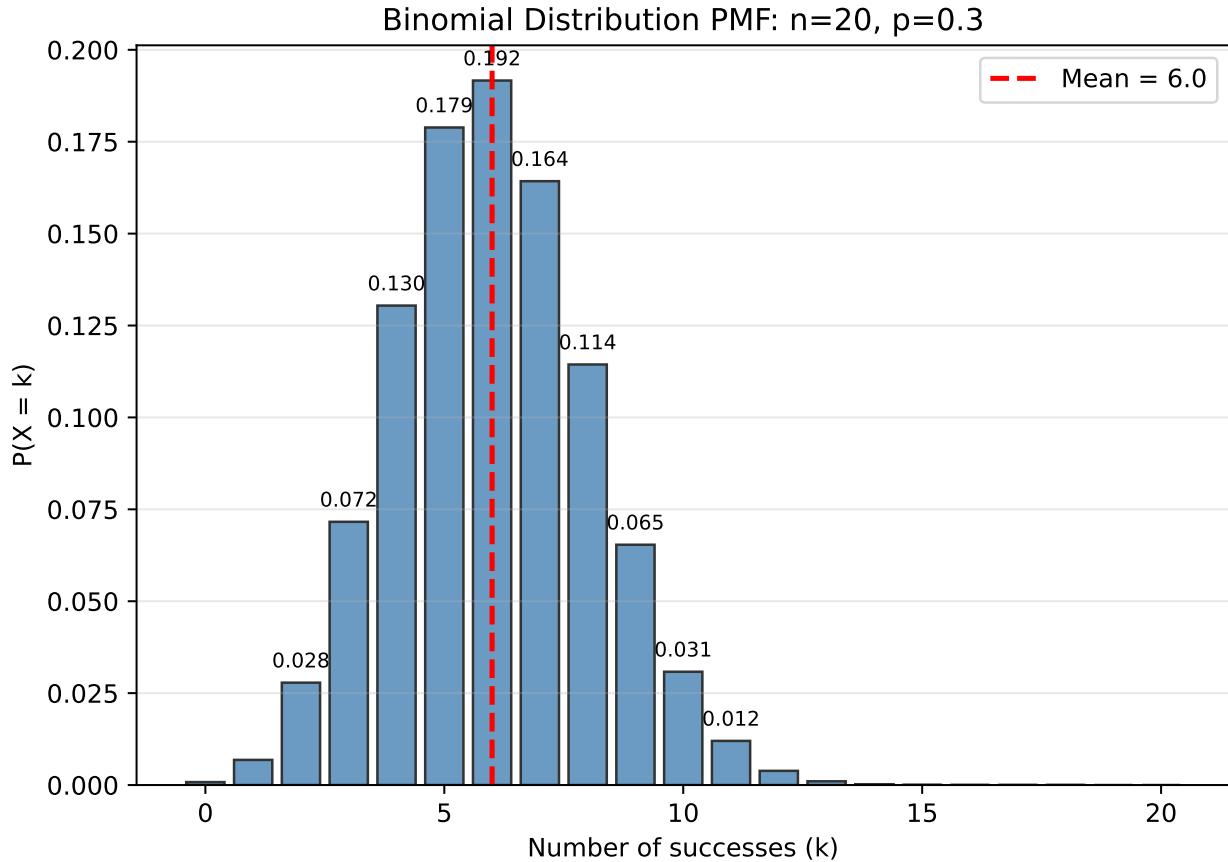
# Highlight mean
mean = n * p
ax.axvline(mean, color='red', linestyle='--', linewidth=2, label=f'Mean = {mean:.1f}')

# Add value labels on significant bars
for i, (xi, pi) in enumerate(zip(x, pmf)):
    if pi > 0.01: # Only label visible bars
        ax.text(xi, pi + 0.003, f'{pi:.3f}', ha='center', va='bottom', fontsize=8)

ax.set_xlabel('Number of successes (k)')
ax.set_ylabel('P(X = k)')
ax.set_title(f'Binomial Distribution PMF: n={n}, p={p}')
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print(f"E[X] = np = {n*p}")
print(f"Var(X) = np(1-p) = {n*p*(1-p)}")
print(f" = {np.sqrt(n*p*(1-p)):.3f}")
```



$$E[X] = np = 6.0$$

$$\begin{aligned} \text{Var}(X) &= np(1-p) = 4.19999999999999 \\ &= 2.049 \end{aligned}$$

1.5.4.3 Discrete Uniform Distribution

The [discrete uniform distribution](#) is another simple discrete distribution - every outcome is equally likely.

$X \sim \text{DiscreteUniform}(a, b)$ if:

$$f_X(x) = \frac{1}{b-a+1}, \quad x \in \{a, a+1, \dots, b\}$$

where a and b are integers with $a \leq b$.

Key properties:

- $E[X] = \frac{a+b}{2}$
- $\text{Var}(X) = \frac{(b-a+1)^2 - 1}{12}$

Use cases:

- Fair die roll: $\text{DiscreteUniform}(1, 6)$
- [Attack roll](#): $\text{DiscreteUniform}(1, 20)$
- Random selection from a finite set
- Lottery number selection

```

# Discrete Uniform distribution
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import randint

# Example: fair die
a, b = 1, 6
x = np.arange(a, b+1)
pmf = [1/(b-a+1)] * len(x)

fig, ax = plt.subplots(figsize=(7, 4))
bars = ax.bar(x, pmf, width=0.6, alpha=0.8, color='lightgreen', edgecolor='black', linewidth=2)

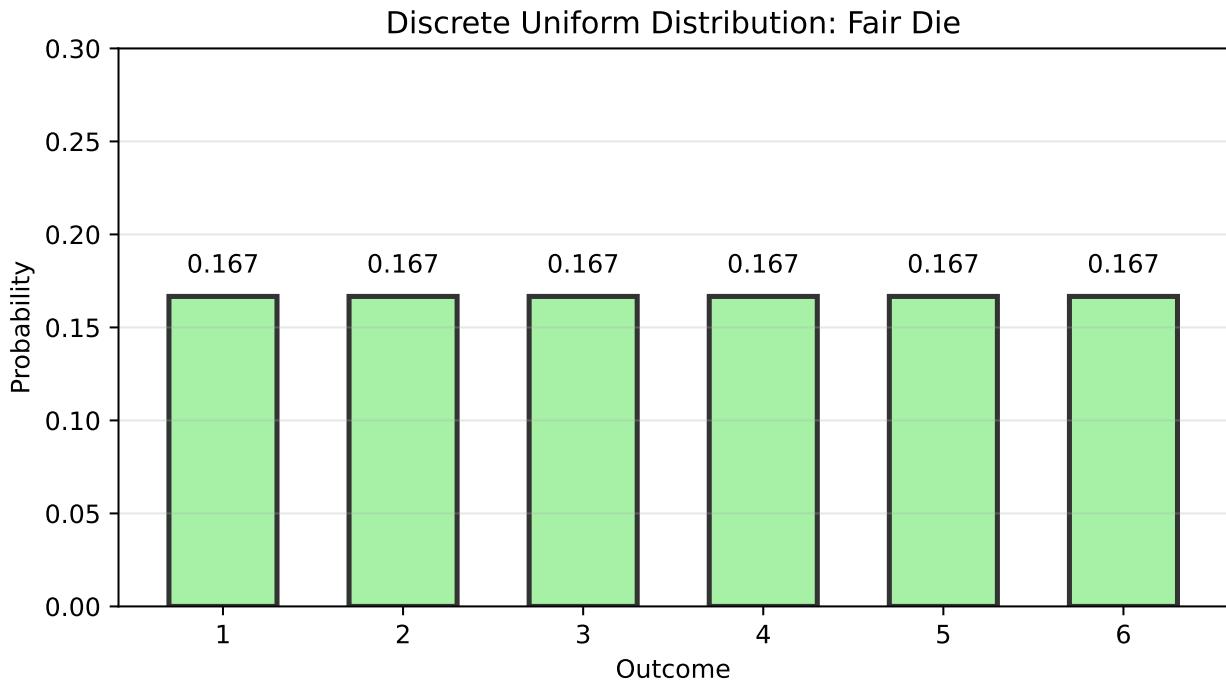
# Add value labels
for i, (xi, pi) in enumerate(zip(x, pmf)):
    ax.text(xi, pi + 0.01, f'{pi:.3f}', ha='center', va='bottom', fontsize=10)

ax.set_xlabel('Outcome')
ax.set_ylabel('Probability')
ax.set_title(f'Discrete Uniform Distribution: Fair Die')
ax.set_ylim(0, 0.3)
ax.set_xticks(x)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print(f"E[X] = {(a+b)/2}")
print(f"Var(X) = {((b-a+1)**2 - 1)/12:.3f}")

```



$$E[X] = 3.5$$

```
Var(X) = 2.917
```

1.5.4.4 Categorical Distribution

The [categorical distribution](#) is a generalization of Bernoulli to multiple categories (also called “Generalized Bernoulli” or “Multinoulli”). You can also see it as a generalization of the discrete uniform distribution to a discrete *non-uniform* distribution.

$X \sim \text{Categorical}(p_1, \dots, p_k)$ if:

$$f_X(x) = p_x, \quad x \in \{1, 2, \dots, k\}$$

where $p_i \geq 0$ and $\sum_{i=1}^k p_i = 1$.

Key properties:

- One-hot encoding: Often represented as a vector with one 1 and rest 0s
- Special case: Categorical with $k = 2$ is equivalent to Bernoulli
- Special case: If all probabilities are equal, it becomes a discrete uniform
- Foundation for multinomial distribution (multiple categorical trials)

Use cases:

- Outcome of rolling a (possibly unfair) die
- Classification into multiple categories
- Language modeling (next-token prediction)³
- Customer choice among products

```
# Categorical distribution
import numpy as np
import matplotlib.pyplot as plt

# Example: Customer choice among 5 products
categories = ['Product A', 'Product B', 'Product C', 'Product D', 'Product E']
probabilities = [0.30, 0.25, 0.20, 0.15, 0.10]
x = np.arange(len(categories))

fig, ax = plt.subplots(figsize=(7, 4))
bars = ax.bar(x, probabilities, alpha=0.8, color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'],
               edgecolor='black', linewidth=2)

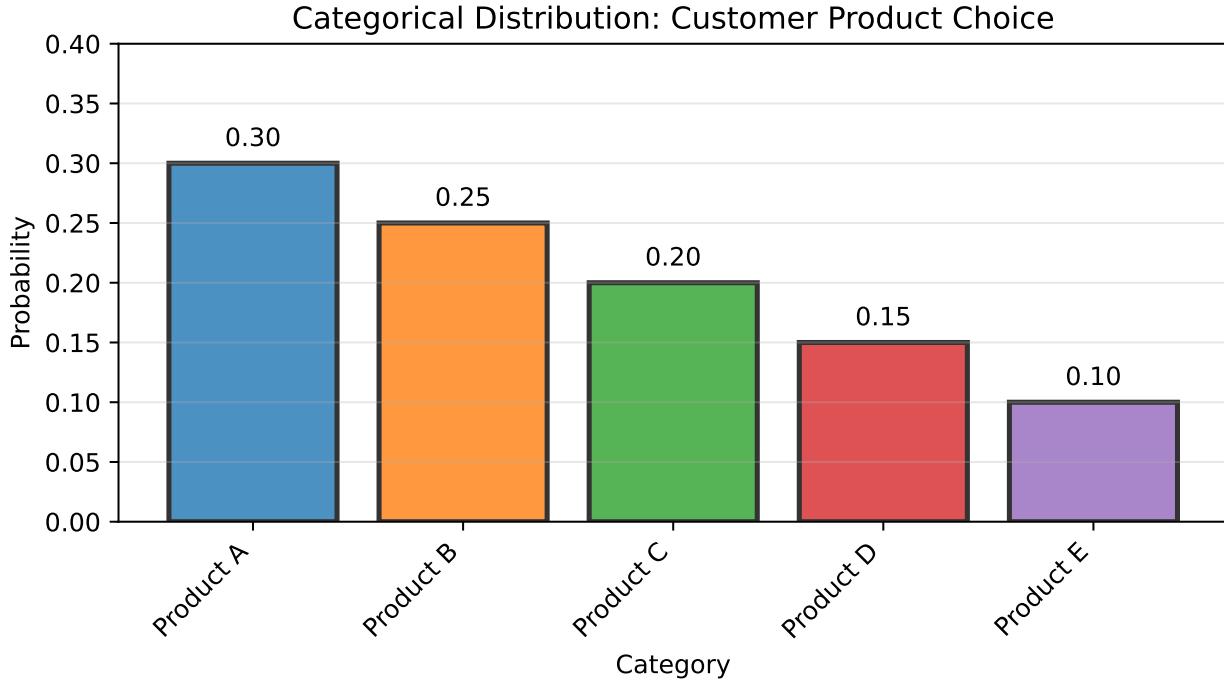
# Add value labels
for i, p in enumerate(probabilities):
    ax.text(i, p + 0.01, f'{p:.2f}', ha='center', va='bottom', fontsize=10)

ax.set_xlabel('Category')
ax.set_ylabel('Probability')
ax.set_title('Categorical Distribution: Customer Product Choice')
ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=45, ha='right')
ax.set_ylim(0, 0.4)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

³In modern LLMs, the categorical distribution is over tokens (parts of words), not full words. The token vocabulary can be huge - tens of thousands of different tokens like “a”, “aba”, “add”, etc. GPT models typically use vocabularies of 50,000-100,000 tokens.

```
# Expected value for indicator representation (does it make sense here?)
print("If we encode categories as 1, 2, 3, 4, 5:")
expected = sum((i+1) * p for i, p in enumerate(probabilities))
print(f"E[X] = {expected:.2f} (does it really make sense here?)")
```



If we encode categories as 1, 2, 3, 4, 5:
 $E[X] = 2.50$ (does it really make sense here?)

1.5.4.5 Brief Catalog: Other Discrete Distributions

Poisson(λ): The [Poisson distribution](#) models count of rare events in fixed intervals:

- PMF: $f_X(x) = e^{-\lambda} \frac{\lambda^x}{x!}$ for $x = 0, 1, 2, \dots$
- Mean = Variance = λ (*lambda*)
- Use: Email arrivals, typos per page, customer arrivals
- Approximates Binomial(n, p) when n large, p small: use $\lambda = np$

Geometric(p): The [geometric distribution](#) represents the number of trials until first success:

- PMF: $f_X(x) = p(1-p)^{x-1}$ for $x = 1, 2, \dots$
- Use: Waiting times, number of attempts until success

Negative Binomial(r, p): The [negative binomial](#) represents the number of failures before r th success

- Generalization of geometric distribution
- Use: Overdispersed count data, robust alternative to Poisson

1.5.5 Continuous Random Variables

A random variable X is **continuous** if there exists a function f_X such that:

- $f_X(x) \geq 0$ for all x
- $\int_{-\infty}^{\infty} f_X(x)dx = 1$
- For any $a < b$: $\mathbb{P}(a < X < b) = \int_a^b f_X(x)dx$

The function f_X is called the **probability density function (PDF)**.

⚠ Warning

Important distinctions from discrete case:

- $\mathbb{P}(X = x) = 0$ for any single point x : in a continuum, there is zero probability of picking one specific point
- PDF can exceed 1 (it's a *density*, not a probability!)
- We get probabilities by integrating densities over an interval, not summing

Relationship between PDF and CDF:

- $F_X(x) = \int_{-\infty}^x f_X(t)dt$
- $f_X(x) = F'_X(x)$ (where the derivative exists)

1.5.6 Core Continuous Distributions

1.5.6.1 Uniform Distribution

The **uniform distribution** is the continuous analog of “equally likely outcomes.”

$X \sim \text{Uniform}(a, b)$ if:

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Properties:

- CDF: $F_X(x) = \frac{x-a}{b-a}$ for $a \leq x \leq b$
- Every interval of equal length has equal probability

Use cases:

- Random number generation ($\text{Uniform}(0, 1)$ is fundamental in computational statistics)
- Modeling complete uncertainty within bounds
- Arrival times when “any time is equally likely”

```
# Uniform distribution visualization
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import uniform

# Example: Uniform(a=2, b=5)
a, b = 2, 5
x = np.linspace(0, 7, 1000)

# PDF
pdf = uniform.pdf(x, loc=a, scale=b-a)

fig, ax = plt.subplots(figsize=(7, 4))

# Plot the PDF
ax.plot(x, pdf, linewidth=3, color='darkblue', label=f'Uniform({a}, {b})')
ax.fill_between(x, 0, pdf, where=(x >= a) & (x <= b), alpha=0.3, color='lightblue')

# Mark the boundaries
ax.axvline(x=a, color='red', linestyle='--', linewidth=2, alpha=0.7)
```

```

ax.axvline(x=b, color='red', linestyle='--', linewidth=2, alpha=0.7)

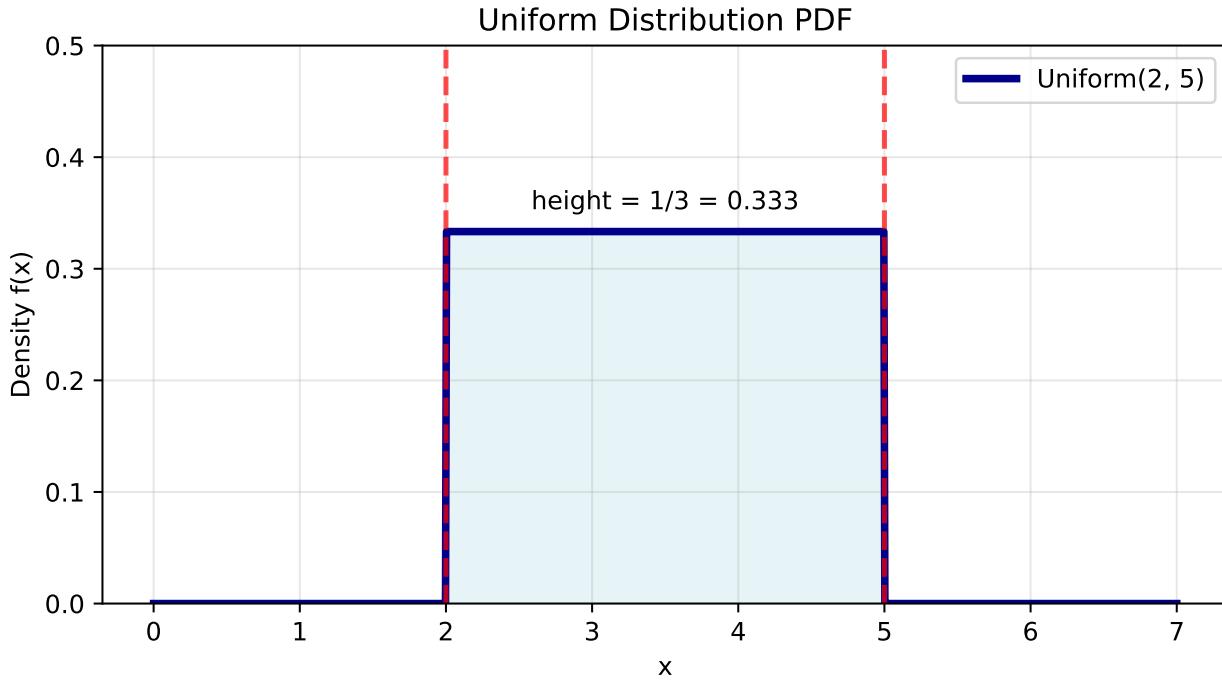
# Add height label
ax.text((a+b)/2, 1/(b-a) + 0.02, f'height = 1/{b-a} = {1/(b-a):.3f}' ,
        ha='center', fontsize=10)

ax.set_xlabel('x')
ax.set_ylabel('Density f(x)')
ax.set_title('Uniform Distribution PDF')
ax.set_ylim(0, 0.5)
ax.grid(True, alpha=0.3)
ax.legend()

plt.tight_layout()
plt.show()

print(f"E[X] = (a+b)/2 = {(a+b)/2}")
print(f"Var(X) = (b-a)^2/12 = {(b-a)**2/12:.3f}")
print(f"Total area under curve = {1/(b-a)} x {b-a} = 1 ")

```



$E[X] = (a+b)/2 = 3.5$
 $\text{Var}(X) = (b-a)^2/12 = 0.750$
 Total area under curve = $0.333333333333333 \times 3 = 1$

1.5.6.2 Normal (Gaussian) Distribution

The [normal distribution](#) (also called Gaussian distribution) is the most important distribution in statistics, arising from the Central Limit Theorem.

$X \sim \text{Normal}(\mu, \sigma^2)$ or $\mathcal{N}(\mu, \sigma^2)$ if:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Parameters:

- μ (*mu*): mean (center of distribution)
- σ^2 (*sigma squared*): variance (σ is standard deviation - controls spread)

Key properties:

- Symmetric bell curve centered at μ
- About 68% of probability within $\mu \pm \sigma$
- About 95% within $\mu \pm 2\sigma$
- About 99.7% within $\mu \pm 3\sigma$

Standard Normal: $Z \sim \mathcal{N}(0, 1)$ has $\mu = 0, \sigma = 1$

- Any normal can be standardized: If $X \sim \mathcal{N}(\mu, \sigma^2)$, then $Z = \frac{X-\mu}{\sigma} \sim \mathcal{N}(0, 1)$
- This allows us to use standard normal tables for any normal distribution

Additivity: If $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ are independent, then:

$$\sum_{i=1}^n X_i \sim \mathcal{N}\left(\sum_{i=1}^n \mu_i, \sum_{i=1}^n \sigma_i^2\right)$$

Use cases:

- Measurement errors
- Heights, weights, test scores in large populations
- Sum of many small independent effects (CLT)
- Approximation for many other distributions

```
# Normal distribution visualization
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 8))

# 1. PDF with different parameters
x = np.linspace(-6, 6, 1000)

# Different means
for mu, color in zip([-2, 0, 2], ['red', 'blue', 'green']):
    ax1.plot(x, norm.pdf(x, loc=mu), linewidth=2,
              label=f'={mu}', color=color)

# Different standard deviations
for sigma, style in zip([0.5, 1, 2], ['-', '--', ':']):
    ax1.plot(x, norm.pdf(x, scale=sigma), linewidth=2,
              label=f'= {sigma}', linestyle=style, color='black')

ax1.set_xlabel('x')
ax1.set_ylabel('Density')
ax1.set_title('Normal Distribution PDF')
ax1.legend(loc='upper right')
ax1.grid(True, alpha=0.3)

# 2. 68-95-99.7 Rule
mu, sigma = 0, 1
```

```
x_range = np.linspace(-4, 4, 1000)
y = norm.pdf(x_range, mu, sigma)
ax2.plot(x_range, y, 'k-', linewidth=2)

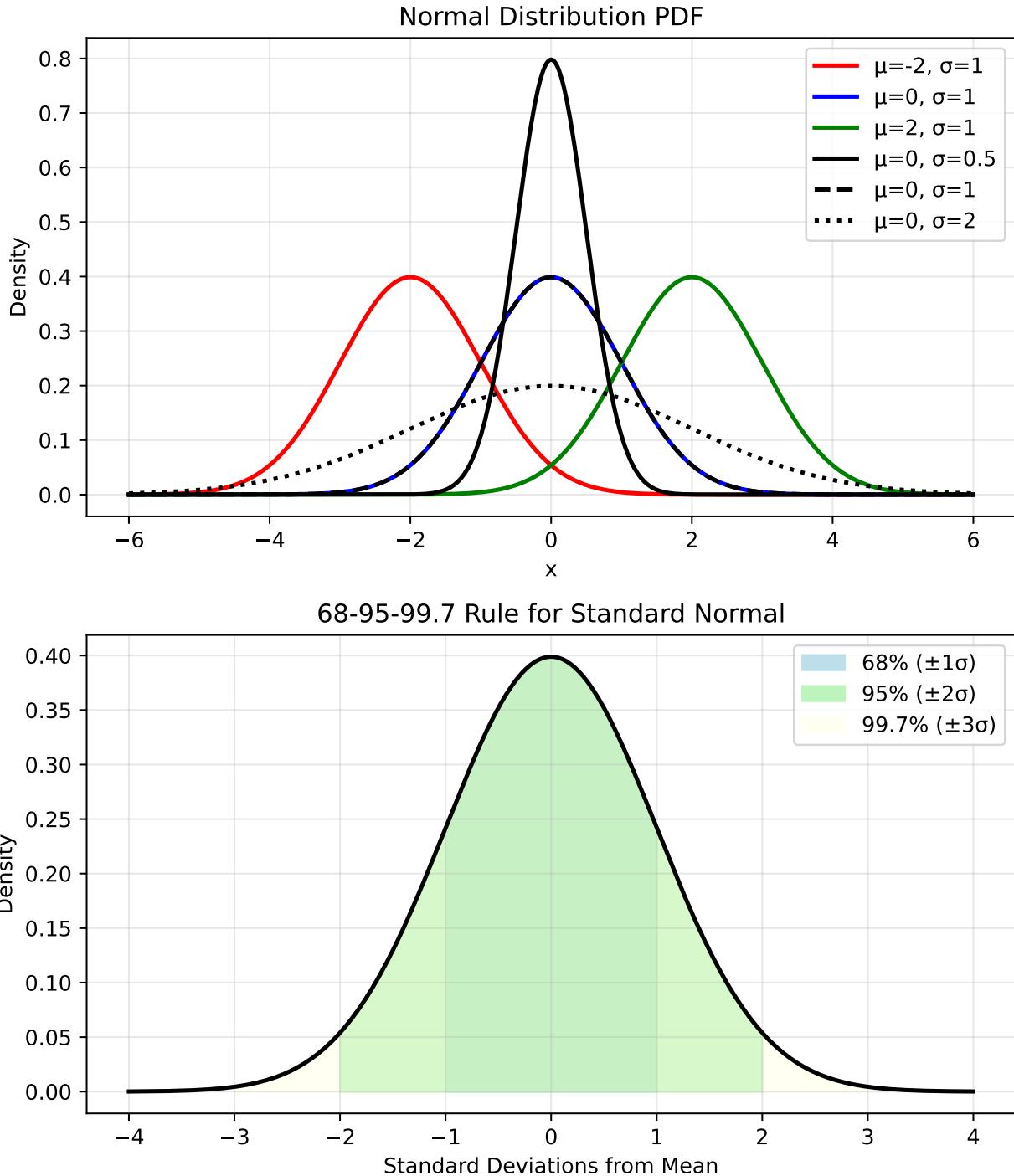
# Fill areas for different sigma ranges
colors = ['lightblue', 'lightgreen', 'lightyellow']
alphas = [0.8, 0.6, 0.4]
labels = ['68% ( $\pm 1$ )', '95% ( $\pm 2$ )', '99.7% ( $\pm 3$ )']

for n_sigma, color, alpha, label in zip([1, 2, 3], colors, alphas, labels):
    x_fill = x_range[np.abs(x_range - mu) <= n_sigma * sigma]
    y_fill = norm.pdf(x_fill, mu, sigma)
    ax2.fill_between(x_fill, 0, y_fill, color=color, alpha=alpha, label=label)

ax2.set_xlabel('Standard Deviations from Mean')
ax2.set_ylabel('Density')
ax2.set_title('68-95-99.7 Rule for Standard Normal')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Key probabilities
print("Standard Normal Probabilities:")
print(f"P(-1 ≤ Z ≤ 1) = {norm.cdf(1) - norm.cdf(-1):.3f} 0.68")
print(f"P(-2 ≤ Z ≤ 2) = {norm.cdf(2) - norm.cdf(-2):.3f} 0.95")
print(f"P(-3 ≤ Z ≤ 3) = {norm.cdf(3) - norm.cdf(-3):.3f} 0.997")
```



Standard Normal Probabilities:

$$P(-1 \leq Z \leq 1) = 0.683 \quad 0.68$$

$$P(-2 \leq Z \leq 2) = 0.954 \quad 0.95$$

$$P(-3 \leq Z \leq 3) = 0.997 \quad 0.997$$

1.5.6.3 Exponential Distribution

The [exponential distribution](#) models waiting times between events in a Poisson process.

$X \sim \text{Exponential}(\beta)$ if:

$$f_X(x) = \frac{1}{\beta} e^{-x/\beta}, \quad x > 0$$

Here β (*beta*) is the scale parameter, the average time between events.

You can also find the exponential distribution parameterized in terms of a *rate* parameter $\lambda = \frac{1}{\beta}$.

Key properties:

- **Memoryless:** $\mathbb{P}(X > s + t | X > s) = \mathbb{P}(X > t)$
– “The future doesn’t depend on how long you’ve already waited”
- Connection to Poisson: If events occur at rate $\lambda = 1/\beta$, time between events is $\text{Exponential}(\beta)$
- CDF: $F_X(x) = 1 - e^{-x/\beta}$ for $x > 0$

Use cases:

- Time between customer arrivals
- Lifetime of electronic components
- Time until next earthquake
- Duration of phone calls

```
# Exponential distribution visualization
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon

# Different values (mean waiting time)
betas = [0.5, 1, 2]
colors = ['red', 'blue', 'green']

fig, ax = plt.subplots(figsize=(7, 5))

x = np.linspace(0, 6, 1000)

for beta, color in zip(betas, colors):
    pdf = expon.pdf(x, scale=beta)
    ax.plot(x, pdf, linewidth=2, color=color, label=f' = {beta}')
    ax.axvline(beta, color=color, linestyle='--', alpha=0.5, linewidth=1)

ax.set_xlabel('Time (x)')
ax.set_ylabel('Density')
ax.set_title('Exponential Distribution PDF')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_xlim(0, 6)
ax.set_ylim(0, 2.1)

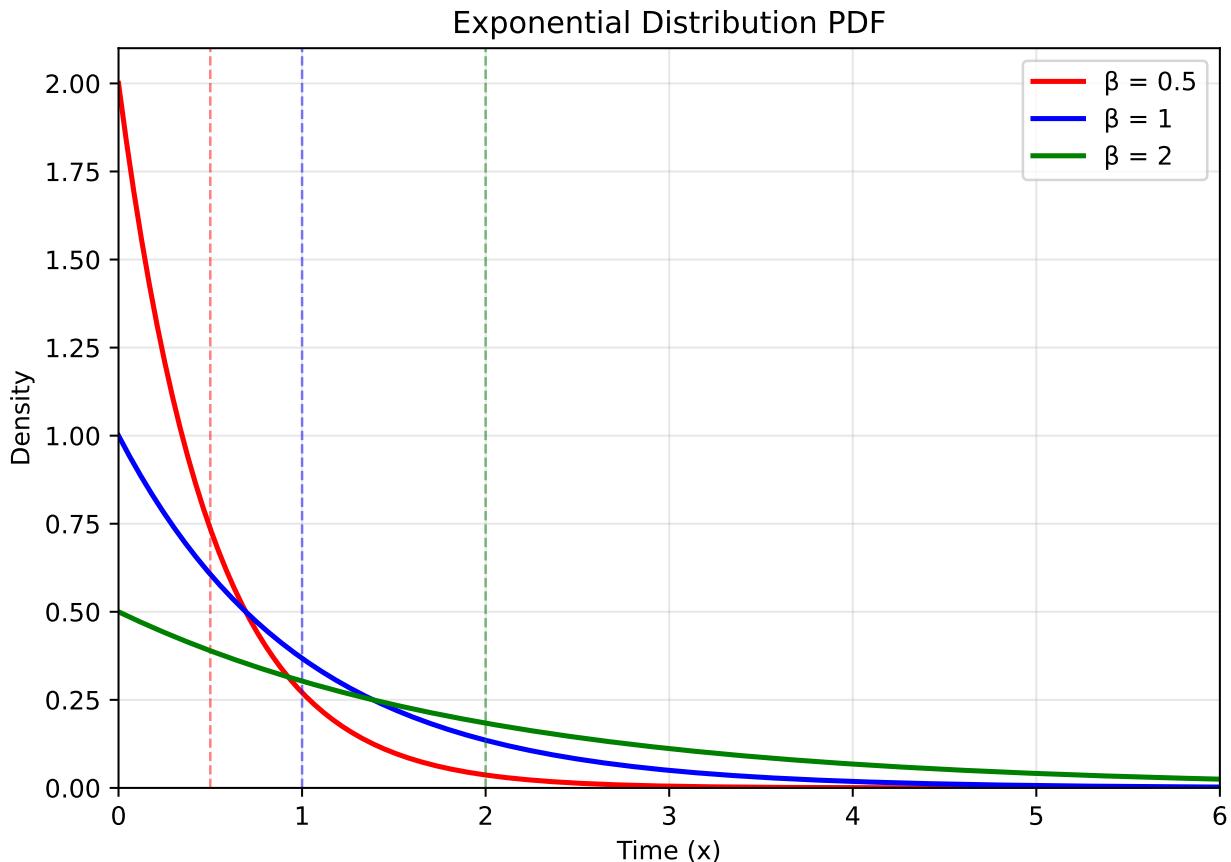
plt.tight_layout()
plt.show()

# Example calculations
beta = 1
print(f"For   = {beta} (rate   = {1/beta}):")
```

```

print(f"E[X] = {beta}")
print(f"Var(X) = ^2 = {beta**2}")
print(f"P(X > 1) = {1 - expon.cdf(1, scale=beta):.3f}")
print(f"Median waiting time = {expon.ppf(0.5, scale=beta):.3f}")

```



```

For = 1 (rate = 1.0):
E[X] = 1
Var(X) = ^2 = 1
P(X > 1) = 0.368
Median waiting time = 0.693

```

1.5.6.4 Brief Catalog: Other Continuous Distributions

Gamma(α, β): The [gamma distribution](#) is a generalization of exponential

- Sum of independent exponentials
- Models waiting time for multiple events

Beta(α, β): The [beta distribution](#) models values between 0 and 1

- Models proportions and probabilities
- Conjugate prior for binomial parameter

t(ν): The [t-distribution](#) is a heavy-tailed alternative to normal

- More probability in tails than normal
- Used when variance is unknown/unstable
- $\nu = 1$ gives Cauchy (no finite mean!)

$\chi^2(p)$: The [chi-squared distribution](#) is the sum of squared standard normals

- If $Z_1, \dots, Z_p \sim \mathcal{N}(0, 1)$ independent, then $\sum Z_i^2 \sim \chi^2(p)$
- Used in hypothesis testing and confidence intervals

1.6 Multivariate Distributions

So far we've focused on single random variables. But in practice, we often deal with multiple related variables: height and weight, temperature and humidity, stock prices of different companies. This leads us to multivariate distributions.

1.6.1 Joint Distributions

For random variables X and Y , the **joint distribution** describes their behavior together:

- **Discrete case:** Joint PMF $f_{X,Y}(x, y) = \mathbb{P}(X = x, Y = y)$
- **Continuous case:** Joint PDF $f_{X,Y}(x, y)$ where

$$\mathbb{P}((X, Y) \in A) = \iint_A f_{X,Y}(x, y) dx dy$$

Example: Discrete joint distribution

Roll two fair six-sided dice. Let X = first die, Y = second die.

The joint PMF is:

$$f_{X,Y}(x, y) = \frac{1}{36} \text{ for } x, y \in \{1, 2, 3, 4, 5, 6\}$$

We can display this as a 6×6 table with each entry equal to $1/36$.

Example: Continuous joint distribution

Let (X, Y) be uniformly distributed on the unit disk.

$$f_{X,Y}(x, y) = \begin{cases} \frac{1}{\pi} & \text{if } x^2 + y^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The normalizing constant $1/\pi$ makes the total probability equal to 1 (area of unit disk is π).

1.6.2 Marginal Distributions

Given a joint distribution, we can find the distribution of each variable separately.

The **marginal distribution** of X is obtained by “summing out” or “integrating out” the other variable:

- **Discrete:** $f_X(x) = \sum_y f_{X,Y}(x, y)$
- **Continuous:** $f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y) dy$

💡 Tip

Think of marginal distributions as projections: if you have points scattered in 2D, the marginal distribution of X is like looking at their shadows on the X-axis.

Example: Sum of two dice

Let X = first die, Y = second die, $S = X + Y$.
What is $\mathbb{P}(S = 7)$?

i Solution

To find $\mathbb{P}(S = 7)$, we sum over all ways to get 7:

- (1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)

So $\mathbb{P}(S = 7) = 6 \times \frac{1}{36} = \frac{1}{6}$

1.6.3 Independent Random Variables

Random variables X and Y are **independent** if:

$$f_{X,Y}(x,y) = f_X(x) \cdot f_Y(y)$$

for all x, y .

This means the joint distribution factors into the product of marginals - knowing the value of one variable tells us nothing about the other.

Example: Independent coin flips

Flip two fair coins. Let $X = 1$ if first is heads, 0 otherwise. Same for Y with second coin.

Joint distribution:

		Y = 0	Y = 1
X = 0	1/4	1/4	
		1/4	1/4

Since each entry equals the product of marginal probabilities (e.g., $\frac{1}{4} = \frac{1}{2} \times \frac{1}{2}$), X and Y are independent.

⚠ Warning

Common mistake: Assuming uncorrelated means independent.

Independence implies zero correlation, but zero correlation does NOT imply independence! We'll see counterexamples when we study correlation in Chapter 3.

1.6.4 Conditional Distributions

The **conditional distribution** of X given $Y = y$ is:

- **Discrete:** $f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$ if $f_Y(y) > 0$
- **Continuous:** Same formula, interpreted as densities

This tells us how X behaves when we know $Y = y$.

Example: Quality control

A factory produces items on two machines. Let:

- X = quality score (0-100)
- Y = machine (1 or 2)

Suppose Machine 1 produces 60% of items with quality $\sim \mathcal{N}(80, 25)$, and Machine 2 produces 40% with quality $\sim \mathcal{N}(70, 100)$.

If we observe a quality score of 75, which machine likely produced it? This requires the conditional distribution $\mathbb{P}(Y|X = 75)$.

1.6.5 Interactive Exploration: Marginal and Conditional Distributions

Let's explore how marginal and conditional distributions relate to a joint distribution using an interactive visualization.

Instructions:

- Use the sliders to change the x and y values
- Check the boxes to switch between marginal distributions (e.g., $f_X(x)$) and conditional distributions (e.g., $f_{X|Y}(x|y)$)
- When showing conditional distributions, red dashed lines appear on the joint distribution showing where we're conditioning
- The visualization uses the simpler shorthand notation $p(x)$ for $f_X(x)$ and $p(x|y)$ for $f_{X|Y}(x|y)$ (and analogous formulas for other pdfs)

?

Interactive Visualization Available Online

An interactive visualization of marginal and conditional distributions is available in the web version of these notes. This demo allows you to:

- Adjust x and y values with sliders
- Toggle between marginal and conditional distributions
- See how conditional distributions change as you vary the conditioning value

Visit the online version to explore this interactive demonstration.

1.6.6 Random Vectors and IID Random Variables

A **random vector** is a vector $\mathbf{X} = (X_1, X_2, \dots, X_n)^T$ where each component X_i is a random variable. The joint behavior of all components is characterized by their joint distribution.

Random vectors allow us to study multiple random quantities together, which leads us to an important special case.

IID Random Variables:

Random variables X_1, \dots, X_n are **independent and identically distributed (IID)** if:

1. They are mutually independent
2. They all have the same distribution

We write: $X_1, \dots, X_n \stackrel{iid}{\sim} F$.

If F has density f we also write $X_1, \dots, X_n \stackrel{iid}{\sim} f$.

X_1, \dots, X_n is a **random sample of size n** from F (or f , respectively).

IID assumptions are fundamental in statistics:

- **Random sampling:** Each observation comes from the same population

- **No interference:** One observation doesn't affect others
- **Stable conditions:** The underlying distribution doesn't change

Example: Customer arrivals

Times between customer arrivals at a stable business might be IID Exponential(β).

Not IID:

- Stock prices (today's price depends on yesterday's)
- Temperature readings (temporal correlation)
- Survey responses from same household (likely correlated)

1.6.7 Important Multivariate Distributions

1.6.7.1 Multinomial Distribution

The **multinomial distribution** is a generalization of binomial to multiple categories.

If we have k categories with probabilities p_1, \dots, p_k (summing to 1), and we observe n independent trials, then the counts (X_1, \dots, X_k) follow a **Multinomial** distribution:

$$f(x_1, \dots, x_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

where $\sum x_i = n$.

Use cases:

- Dice rolls (6 categories for a standard die – or k for k -sided dice)
- Survey responses (multiple choice)
- Document word counts
- Genetic allele frequencies

1.6.7.2 Multivariate Normal Distribution

The **multivariate normal distribution** is the multivariate generalization of the normal distribution.

A random vector $\mathbf{X} = (X_1, \dots, X_k)^T$ has a **multivariate normal** distribution, written $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, if:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{k/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right)$$

where:

- μ is the mean vector
- Σ is the covariance matrix (symmetric, positive definite)

Key properties:

- Marginals are normal: If $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, then $X_i \sim \mathcal{N}(\mu_i, \Sigma_{ii})$
- Linear combinations are normal: $\mathbf{a}^T \mathbf{X} \sim \mathcal{N}(\mathbf{a}^T \mu, \mathbf{a}^T \Sigma \mathbf{a})$
- Conditional distributions are normal (with formulas for conditional mean and variance)

Special case - Bivariate normal: For two variables with correlation ρ :

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

The correlation ρ controls the relationship:

- $\rho = 0$: independent (for normal variables, uncorrelated = independent!)
- $\rho > 0$: positive relationship
- $\rho < 0$: negative relationship

```
# Bivariate normal distribution visualization
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal

# Create figure
fig, ax = plt.subplots(figsize=(7, 6))

# Bivariate normal with correlation
mean = [0, 0]
cov = [[1, 0.7], [0.7, 1]] # correlation = 0.7

# Create grid
x = np.linspace(-3, 3, 100)
y = np.linspace(-3, 3, 100)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

# Calculate PDF
rv = multivariate_normal(mean, cov)
Z = rv.pdf(pos)

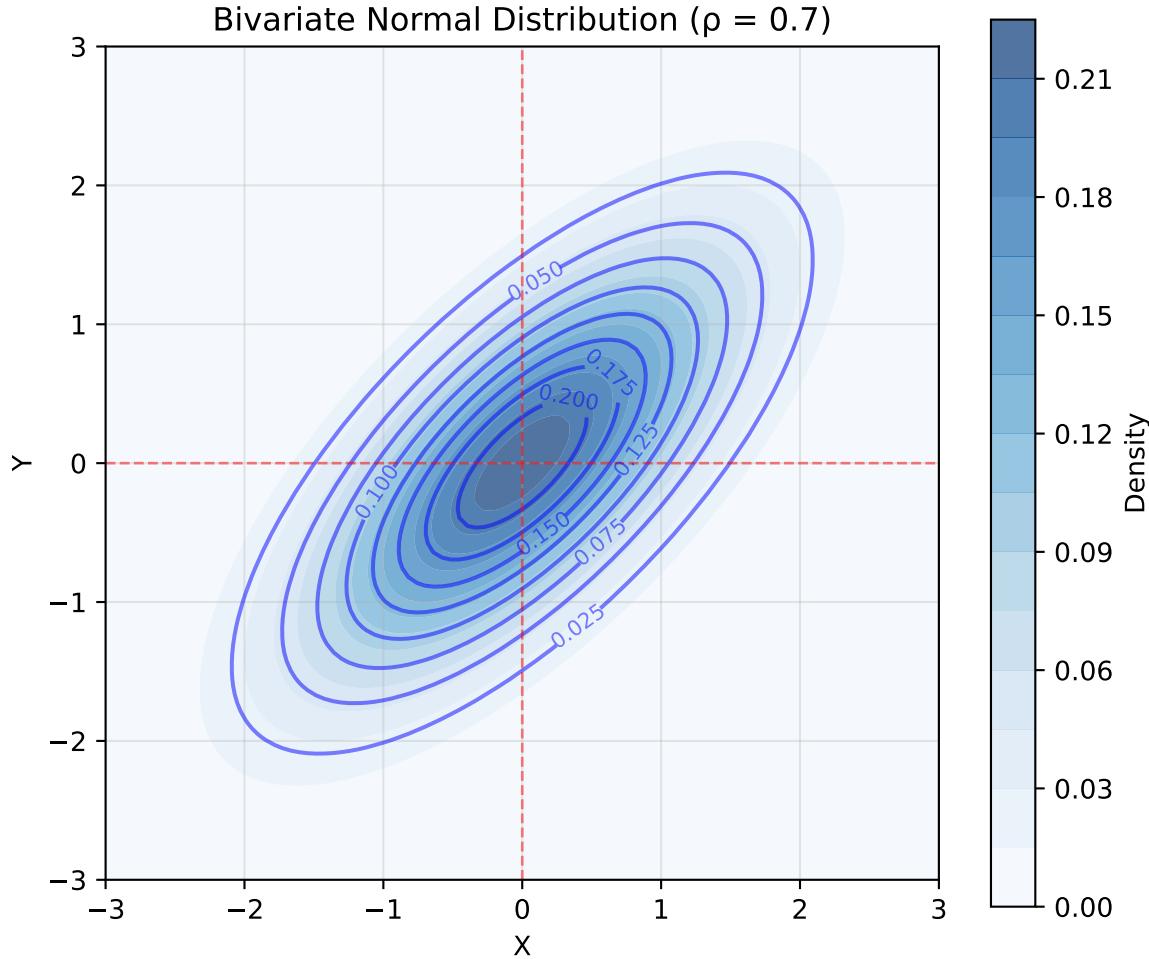
# Contour plot
contour = ax.contour(X, Y, Z, levels=10, colors='blue', alpha=0.5)
ax.clabel(contour, inline=True, fontsize=8)
contourf = ax.contourf(X, Y, Z, levels=20, cmap='Blues', alpha=0.7)
fig.colorbar(contourf, ax=ax, label='Density')

# Add marginal indicators
ax.axhline(y=0, color='red', linestyle='--', alpha=0.5, linewidth=1)
ax.axvline(x=0, color='red', linestyle='--', alpha=0.5, linewidth=1)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Bivariate Normal Distribution ( $\rho = 0.7$ )')
ax.set_aspect('equal')
ax.grid(True, alpha=0.3)

plt.show()

# Example calculations
print("Bivariate Normal with  $\rho = 0.7$ :")
print(f"Var(X) = Var(Y) = 1")
print(f"Cov(X, Y) =  $\rho \cdot \text{Var}(X) = 0.7$ ")
print(f"If we observe Y=1, then:")
print(f"   $E[X|Y=1] = \mu_X + \rho \cdot (Y - \mu_Y) = 0.7$ ")
print(f"   $\text{Var}(X|Y=1) = (1 - \rho^2) = {1 - 0.7**2:.2f}$ ")
```



Bivariate Normal with $\rho = 0.7$:

$$\text{Var}(X) = \text{Var}(Y) = 1$$

$$\text{Cov}(X, Y) = \rho \cdot \text{Var}(X) = 0.7$$

If we observe $Y=1$, then:

$$E[X|Y=1] = \rho \cdot (Y - \mu_Y) = 0.7$$

$$\text{Var}(X|Y=1) = (1 - \rho^2) = 0.51$$

i Note

The multivariate normal distribution is central to many statistical methods. We will return to it in more detail in Chapter 2 when we discuss expectations, covariances, and the properties of linear combinations of random variables.

i Advanced: Transformations of Random Variables

We often define variables that are [transformations](#) $g(\cdot)$ of other random variables. Assuming we know the distribution of X or (X, Y) , how do we find the distribution of $Y = g(X)$ or $(U, V) = g(X, Y)$?

Method 1: CDF technique

1. Find the CDF: $F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(g(X) \leq y)$
2. Differentiate to get PDF: $f_Y(y) = F'_Y(y)$

Method 2: Jacobian method (for bijective transformations)

If $(U, V) = g(X, Y)$ is one-to-one with inverse $(X, Y) = h(U, V)$, then:

$$f_{U,V}(u, v) = f_{X,Y}(h(u, v)) \cdot |J|$$

where J is the Jacobian determinant:

$$J = \det \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{pmatrix}$$

Example: Box-Muller transform

We can generate Gaussian (normal) distributed random numbers starting from uniform.

If $U_1, U_2 \sim \text{Uniform}(0, 1)$ independently, then:

- $X = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$
- $Y = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$

are independent $\mathcal{N}(0, 1)$ random variables!

1.7 Chapter Summary and Connections

1.7.1 Key Concepts Review

We've built up probability theory from its foundations:

1. **Sample spaces and events** provide the basic framework for describing uncertainty
2. **Probability axioms** give us consistent rules for quantifying uncertainty
3. **Independence and conditioning** let us model relationships between events
4. **Random variables** connect probability to numerical quantities we can analyze
5. **Distributions** characterize the behavior of random variables
6. **Multivariate distributions** handle multiple related random quantities

1.7.2 Why These Concepts Matter

For Statistical Inference:

- Random variables let us model data mathematically
- Distributions provide templates for common patterns
- Independence assumptions simplify analysis
- Conditioning lets us update beliefs with data

For Machine Learning:

- Probability distributions model uncertainty in predictions
- Bayes' theorem enables Bayesian ML methods
- Multivariate distributions handle high-dimensional data
- Independence assumptions make computation tractable

For Data Science Practice:

- Understanding distributions helps choose appropriate methods
- Recognizing dependence prevents incorrect analyses
- Conditional probability quantifies relationships in data
- Simulation using these distributions validates methods

1.7.3 Common Pitfalls to Avoid

1. **Confusing $\mathbb{P}(A|B)$ with $\mathbb{P}(B|A)$** - These can be vastly different!

2. **Assuming independence without justification** - Real-world variables are often dependent
3. **Misinterpreting PDFs as probabilities** - PDFs are densities, not probabilities
4. **Forgetting $\mathbb{P}(X = x) = 0$ for continuous variables** - Use intervals for continuous RVs
5. **Thinking disjoint means independent** - Disjoint events are maximally dependent!

1.7.4 Chapter Connections

The probability foundations from this chapter provide the mathematical language for all of statistics:

- **Next - Chapter 2 (Expectation)**: Building on our introduction to random variables, we'll explore expectation as a fundamental tool for summarizing distributions, including variance and the powerful linearity of expectation property
- **Chapter 3 (Convergence & Inference)**: Using the probability framework and IID concept from this chapter, we'll prove the Law of Large Numbers and Central Limit Theorem—the theoretical foundations that justify using samples to learn about populations
- **Chapter 4 (Bootstrap)**: Apply our understanding of empirical distributions to develop computational methods for quantifying uncertainty, providing a modern alternative to traditional parametric approaches

1.7.5 Self-Test Problems

Try to answer these questions after reading these lecture notes.

1. **Bayes in action**: A test for a disease has 95% sensitivity (true positive rate) and 98% specificity (true negative rate). If 0.1% of the population has the disease, what's the probability someone with a positive test actually has the disease?
2. **Distribution identification**: Times between earthquakes in a region average 50 days. What distribution would you use to model the time until the next earthquake? Why?
3. **Independence check**: You roll two dice. Let A = “sum is even” and B = “first die shows 3”. Are A and B independent?
4. **Conditional expectation preview**: In a factory, Machine 1 makes 70% of products with defect rate 2%. Machine 2 makes 30% with defect rate 5%. If a product is defective, what's the probability it came from Machine 1?

1.7.6 Connections to Source Material

Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Why Do We Need Statistics?	Expanded material from the slides, contextualizing statistics for data science.
Foundations of Probability	
Sample Spaces and Events	AoS §1.2
Probability Axioms	AoS §1.3 (Definition 1.5)
Interpretations of Probability	AoS §1.3
Finite Sample Spaces & Counting	AoS §1.4
Independence and Conditional Probability	
Independent Events	AoS §1.5 (Definition 1.9)
Conditional Probability	AoS §1.6 (Definition 1.12)

Bayes' Theorem & Law of Total Probability	AoS §1.7 (Theorems 1.16, 1.17)
Random Variables	
Definition and Intuition	AoS §2.1 (Definition 2.1)
CDF, PMF, and PDF	AoS §2.2 (Definitions 2.5, 2.9, 2.11)
Core Discrete Distributions	AoS §2.3
Core Continuous Distributions	AoS §2.4
Multivariate Distributions	
Joint Distributions	AoS §2.5
Marginal Distributions	AoS §2.6
Independent Random Variables	AoS §2.7 (Definition 2.29)
Conditional Distributions	AoS §2.8 (Definitions 2.35, 2.36)
Random Vectors and IID Samples	AoS §2.9 (Definition 2.41)
Important Multivariate Distributions	AoS §2.10
Transformations of Random Variables	AoS §2.11, §2.12
Chapter Summary and Connections	New summary material.

1.7.7 Further Reading

- **Probability Theory:** Ross, “A First Course in Probability” - accessible introduction
- **Mathematical Statistics:** Casella & Berger, “Statistical Inference” - rigorous treatment
- **Bayesian Perspective:** Gelman et al., “Bayesian Data Analysis” - modern Bayesian view
- **Computational Approach:** Blitzstein & Hwang, “Introduction to Probability” - simulation-based

1.7.8 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

Remember: Probability is the language of uncertainty. Master this language, and you'll be able to express and analyze uncertainty in any domain.

Chapter 2

Expectation

2.1 Learning Objectives

After completing this chapter, you will be able to:

- Explain the concept of expectation and its role in summarizing distributions and machine learning.
- Apply key properties of expectation, especially its linearity, to simplify complex calculations.
- Calculate and interpret variance, covariance, and correlation as measures of spread and linear dependence.
- Extend expectation concepts to random vectors, including mean vectors and covariance matrices.
- Define and apply conditional expectation and understand its key properties.

Note

This chapter covers expectation, variance, and related concepts essential for statistical inference. The material is adapted and expanded from Chapter 3 of Wasserman (2013), which interested readers are encouraged to consult directly.

2.2 Introduction and Motivation

2.2.1 The Essence of Supervised Machine Learning

The fundamental goal of supervised machine learning is seemingly simple: train a model that makes successful predictions on new, unseen data. However, this goal hides a deeper challenge that lies at the heart of statistics.

When we train a model, we work with a finite training set:

$$X_1, \dots, X_n \sim F_X$$

where F_X is the data generating distribution. Our true objective is to find a model that minimizes the **expected loss**:

$$\mathbb{E}[L(X)]$$

over the entire distribution F_X , where $L(\cdot)$ is some suitable loss function. But we can only compute the **empirical loss**:

$$\frac{1}{n} \sum_{i=1}^n L(X_i),$$

which is the loss function summed over the training data.

This gap between what we want (*expected* loss) and what we can calculate (*empirical* loss) is the central challenge of machine learning. The concept of expectation provides the mathematical framework to understand and bridge this gap.

Multiple Perspectives

Intuitive

Goal: Imagine training a neural network to classify cat and dog images.

You have 10,000 training images, but your model needs to work on millions of future images it's never seen. When your model achieves 98% accuracy on training data, that's just the average over your specific 10,000 images. What you really care about is the accuracy over *all possible* cat and dog images that exist or could exist.

This gap—between what we can measure (training performance) and what we want (real-world performance)—is why expectation is central to machine learning. Every loss function is secretly an expectation!

The **cross-entropy loss** used for classification tasks measures how “surprised” your model is by the true labels. Lower surprise = better predictions. The key insight: we minimize the *average surprise* over our training data, hoping it approximates the *expected surprise* over all possible data.

Mathematical

Setup: We want to classify images as cats ($y = 1$) or dogs ($y = 0$).

Our model outputs:

$\hat{p}(x)$ = predicted probability that image x is a cat.

Step 1: Define the loss for one example

For a single image-label pair (x, y) , the cross-entropy loss is:

$$L(x, y) = -[y \log(\hat{p}(x)) + (1 - y) \log(1 - \hat{p}(x))]$$

This penalizes wrong predictions:

- If $y = 1$ (cat) but $\hat{p}(x) \approx 0$: large loss
- If $y = 0$ (dog) but $\hat{p}(x) \approx 1$: large loss
- Correct predictions → small loss

Step 2: The fundamental problem

What we want to minimize (expected loss over all possible images):

$$R_{\text{true}} = \mathbb{E}_{(X, Y)}[L(X, Y)]$$

What we can compute (average loss over training data):

$$R_{\text{train}} = \frac{1}{n} \sum_{i=1}^n L(x_i, y_i)$$

Step 3: The connection to expectation

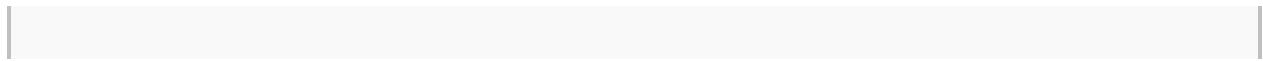
Notice that R_{train} is just the *sample mean* of the losses, while R_{true} is the *expectation* of the loss. By the Law of Large Numbers:

$$R_{\text{train}} \xrightarrow{n \rightarrow \infty} R_{\text{true}}$$

This is why machine learning is fundamentally about expectation!

Computational

Let's see cross-entropy loss in action with a simple cat/dog classifier. We'll simulate predictions and compute both the loss on a small training set and the true expected loss over the entire population. Note that in this example we are not *training* a model. We are given a model, and we want to compute its loss. What we see is how close the empirical loss is to the expected loss as we change the dataset size over which we compute the loss.



```

import numpy as np
import matplotlib.pyplot as plt

# Simulate a simple "classifier" that predicts cat probability based on
# a single feature (e.g., "ear pointiness" from 0 to 1)
np.random.seed(42)

# True probabilities: cats have pointier ears
def true_cat_probability(ear_pointiness):
    # Logistic function: more pointy → more likely cat
    return 1 / (1 + np.exp(-5 * (ear_pointiness - 0.5)))

# Generate population data
n_population = 10000
ear_pointiness = np.random.uniform(0, 1, n_population)
true_probs = true_cat_probability(ear_pointiness)
# Sample actual labels based on true probabilities
labels = (np.random.random(n_population) < true_probs).astype(int)

# Our (imperfect) model's predictions
def model_prediction(x):
    # Slightly wrong sigmoid (shifted and less steep)
    return 1 / (1 + np.exp(-3 * (x - 0.45)))

# Compute cross-entropy loss
def cross_entropy_loss(y_true, y_pred):
    # Avoid log(0) with small epsilon
    eps = 1e-7
    y_pred = np.clip(y_pred, eps, 1 - eps)
    return -(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Show what training sees vs reality
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

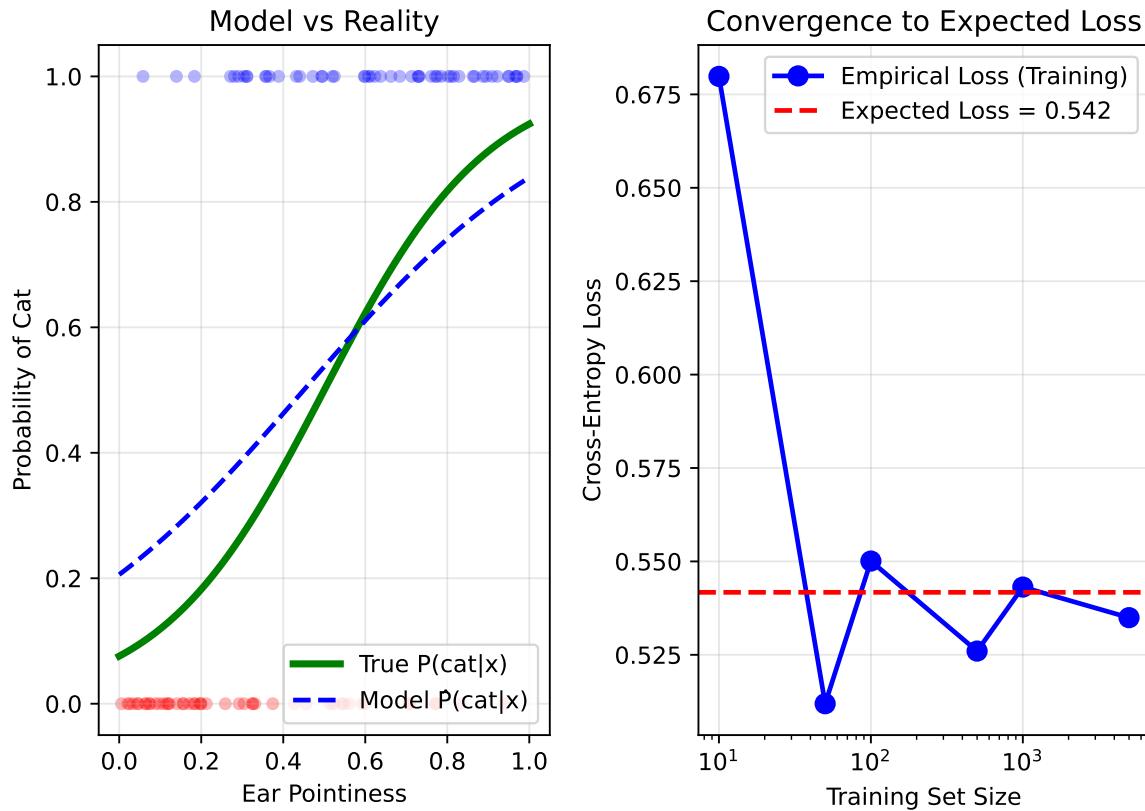
# Left: Model predictions vs truth
x_plot = np.linspace(0, 1, 100)
ax1.plot(x_plot, true_cat_probability(x_plot), 'g-', linewidth=3,
          label='True P(cat|x)')
ax1.plot(x_plot, model_prediction(x_plot), 'b--', linewidth=2,
          label='Model P(cat|x)')
ax1.scatter(ear_pointiness[:100], labels[:100], alpha=0.3, s=20,
            c=['red' if l == 0 else 'blue' for l in labels[:100]])
ax1.set_xlabel('Ear Pointiness')
ax1.set_ylabel('Probability of Cat')
ax1.set_title('Model vs Reality')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Right: Empirical vs Expected Loss
training_sizes = [10, 50, 100, 500, 1000, 5000]
empirical_losses = []

for n in training_sizes:
    # Sample n training examples
    idx = np.random.choice(n_population, n, replace=False)
    train_x = ear_pointiness[idx]
    train_y = labels[idx]
    train_pred = model_prediction(train_x)

    # Compute empirical loss
    empirical_loss = cross_entropy_loss(train_y, train_pred)
    empirical_losses.append(empirical_loss)

```



With just 10 samples: empirical loss = 0.680

With 5000 samples: empirical loss = 0.535

True expected loss: 0.542

As we get more training data to calculate the loss, our empirical loss estimate gets closer to the true expected loss we care about!

2.2.2 Why Expectation Matters in ML and Beyond

The concept of expectation appears throughout data science and statistics:

1. **Statistical Inference:** Estimating population parameters from samples
2. **Decision Theory:** Maximizing expected utility or minimizing expected risk
3. **A/B Testing:** Measuring expected treatment effects
4. **Financial Modeling:** Expected returns and risk assessment
5. **Loss Functions in Deep Learning:** Cross-entropy loss and ELBO in VAEs

In each case, we're trying to understand average behavior over some distribution, which is precisely what expectation captures.

2.3 Foundations of Expectation

i Finnish Terminology Reference

For Finnish-speaking students, here's a reference table of key terms in this chapter:

English	Finnish	Context
Expected value/Expectation	Odotusarvo	The mean of a distribution
Mean	Keskiarvo	Same as expectation
Variance	Varianssi	Measure of spread
Standard deviation	Keskihajonta	Square root of variance
Covariance	Kovarianssi	Linear relationship between variables
Correlation	Korrelaatio	Standardized covariance
Sample mean	Otoskeskiarvo	Average of data points
Sample variance	Otosvarianssi	Empirical measure of spread
Conditional expectation	Ehdollinen odotusarvo	Mean given information
Moment	Momentti	Powers of random variable
Random vector	Satunnaisvektori	Vector of random variables
Covariance matrix	Kovarianssimatriisi	Matrix of covariances
Precision matrix	Tarkkuusmatriisi	Inverse of covariance matrix
Moment generating function	Momenttigeneroiva funktio	Transform for finding moments
Central moment	Keskusmomentti	Moment about the mean

2.3.1 Definition and Basic Properties

The **expected value**, or **mean**, or **first moment** of a random variable X is defined to be:

$$\mathbb{E}(X) = \begin{cases} \sum_x x\mathbb{P}(X=x) & \text{if } X \text{ is discrete} \\ \int_{\mathbb{R}} xf_X(x) dx & \text{if } X \text{ is continuous} \end{cases}$$

assuming the sum (or integral) is well defined. We use the following notation interchangeably:

$$\mathbb{E}(X) = \mathbb{E}X = \mu = \mu_X$$

The expectation represents the average value of the distribution – the balance point where the distribution would balance if it were a physical object.

Notation: The lowercase Greek letter μ (**mu**; pronounced *mju*) is universally used to denote the mean.

i Simplified Notation

In this course, we will write the expectation using the simplified notation:

$$\mathbb{E}(X) = \int xf_X(x)dx$$

when the type of random variable is unspecified and could be either continuous or discrete.

For a discrete random variable, you would substitute the integral with a sum, and the PDF $f_X(x)$ (probability density function) with the PMF $\mathbb{P}_X(x)$ (probability mass function), as seen in Chapter 1 of the lecture notes.

Note that this is an abuse of notation and is not mathematically correct, but we found it to be more intuitive in previous iterations of the course.

Example: Simple Expectations

Let's calculate expectations for some basic distributions:

1. **Bernoulli(0.3):**

$$\mathbb{E}(X) = 0 \times 0.7 + 1 \times 0.3 = 0.3$$

2. **Fair six-sided die:**

$$\mathbb{E}(X) = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = \frac{21}{6} = 3.5$$

3. **Two coin flips** (X = number of heads):

$$\mathbb{E}(X) = 0 \times \frac{1}{4} + 1 \times \frac{1}{2} + 2 \times \frac{1}{4} = 1$$

Multiple Perspectives

Intuitive

Expectation is the “center of mass” of a distribution. Imagine:

- **Physical analogy:** If you made a [histogram out of metal](#), the expected value is where you'd place a fulcrum to balance it perfectly.
- **Long-run average:** If you repeat an experiment millions of times and average the results, you'll get very close to the expectation. This isn't just intuition—it's a theorem (the Law of Large Numbers) we'll prove in Chapter 3.
- **Fair price:** In gambling, the expectation tells you the fair price to pay for a game. If a lottery ticket has expected winnings of €2, then €2 is the break-even price.

Think of expectation as answering: “If I had to summarize this entire distribution with a single number pointing at its *center*, what would it be?” The expectation or mean is not the *only* number we could use to represent the center of a distribution, but it is a very common choice suitable for most situations.

Mathematical

The expectation is a linear functional on the space of random variables. For a random variable X with distribution function F , the correct mathematical notation would be:

$$\mathbb{E}(X) = \int x dF(x)$$

This notation correctly unifies the discrete and continuous cases:

- For discrete X : the integral becomes a sum over the jump points of F
- For continuous X : we have $dF(x) = f_X(x)dx$

This notation is particularly useful when dealing with mixed distributions or when stating results that apply to both discrete and continuous random variables without writing separate formulas. We won't be using this notation in the course, but you may find it in mathematical or statistical textbooks, including Wasserman (2013).

Computational

Let's demonstrate expectation through simulation, showing how sample averages converge to the true expectation. We'll also show a case where expectation doesn't exist (see next section).

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import cauchy

# Set up figure with subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 8))

# 1. Convergence for Bernoulli(0.3)
np.random.seed(42)
p = 0.3
n_flips = 40000
flips = np.random.choice([0, 1], size=n_flips, p=[1-p, p])
running_mean = np.cumsum(flips) / np.arange(1, n_flips + 1)

ax1.plot(running_mean, linewidth=2, alpha=0.8, color='blue')
ax1.axhline(y=p, color='red', linestyle='--', label=f'True E[X] = {p}', linewidth=2)
ax1.set_xlabel('Number of trials')
ax1.set_ylabel('Sample mean')
ax1.set_title('Sample Mean Converges to Expectation: Bernoulli(0.3)')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_ylim(0.2, 0.4)

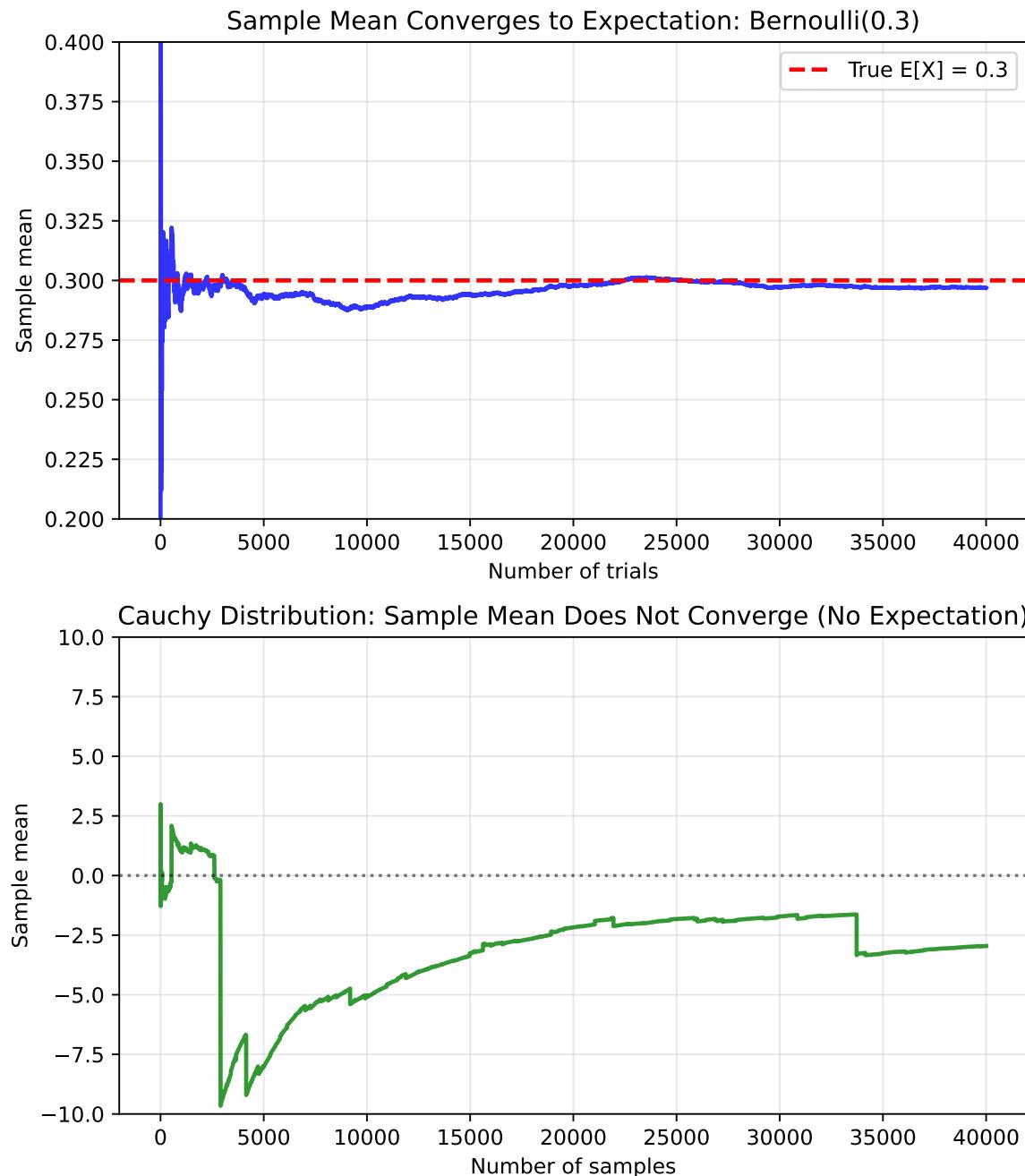
# 2. Cauchy distribution - no expectation exists
n_samples = 40000
cauchy_samples = cauchy.rvs(size=n_samples, random_state=42)
cauchy_running_mean = np.cumsum(cauchy_samples) / np.arange(1, n_samples + 1)

ax2.plot(cauchy_running_mean, linewidth=2, alpha=0.8, color='green')
ax2.axhline(y=0, color='black', linestyle=':', alpha=0.5)
ax2.set_xlabel('Number of samples')
ax2.set_ylabel('Sample mean')
ax2.set_title('Cauchy Distribution: Sample Mean Does Not Converge (No Expectation)')
ax2.grid(True, alpha=0.3)
ax2.set_ylim(-10, 10)

plt.tight_layout()
plt.show()

print(f"Bernoulli: After {n_flips} flips, sample mean = {running_mean[-1]:.4f}")
print(f"Cauchy: After {n_samples} samples, sample mean = {cauchy_running_mean[-1]:.4f}")
print("Notice how Cauchy's sample mean keeps eventually jumping around,")
print("even when you think it's converging to zero!")

```



Bernoulli: After 40000 flips, sample mean = 0.2969

Cauchy: After 40000 samples, sample mean = -2.9548

Notice how Cauchy's sample mean keeps eventually jumping around, even when you think it's converging to zero!

2.3.2 Existence of Expectation

Not all random variables have well-defined expectations.

The expectation $\mathbb{E}(X)$ exists if and only if:

$$\mathbb{E}(|X|) = \int |x| f_X(x) dx < \infty$$

⚠ The Cauchy Distribution

The [Cauchy distribution](#) is a classic example of probability density with no expectation:

$$f_X(x) = \frac{1}{\pi(1+x^2)}$$

To check if expectation exists:

$$\int_{-\infty}^{\infty} |x| \cdot \frac{1}{\pi(1+x^2)} dx = \frac{2}{\pi} \int_0^{\infty} \frac{x}{1+x^2} dx = \infty$$

The integral diverges! This means:

- Sample averages don't converge to any value
- The Law of Large Numbers doesn't apply
- Extreme observations are common due to heavy tails

2.3.3 Expectation of Functions

Often we need the expectation of a function of a random variable. The “Rule of the Lazy Statistician” saves us from finding the distribution of the transformed variable.

Let $Y = r(X)$. Then:

$$\mathbb{E}(Y) = \mathbb{E}(r(X)) = \int r(x) f_X(x) dx$$

This result is incredibly useful—we can find $\mathbb{E}(Y)$ without determining $f_Y(y)$!

Example: Breaking a Stick

A stick of unit length is broken at a random point. What's the expected length of the longer piece?
Let $X \sim \text{Uniform}(0, 1)$ be the break point. The longer piece has length:

$$Y = r(X) = \max\{X, 1-X\}$$

i Solution

We can identify that:

- If $X < 1/2$: longer piece = $1 - X$
- If $X \geq 1/2$: longer piece = X

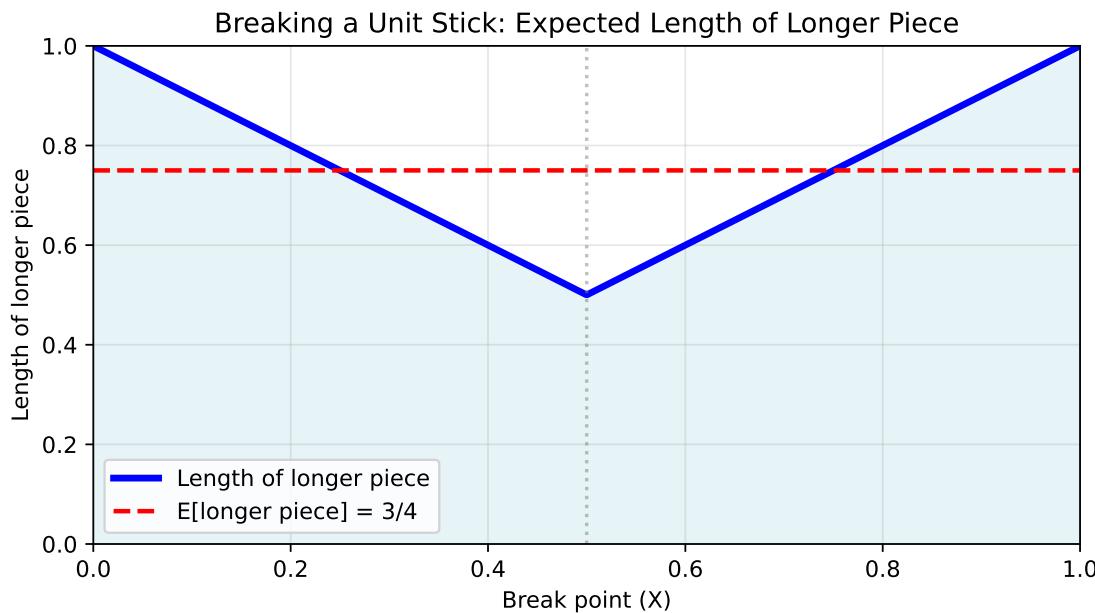
Therefore:

$$\begin{aligned}\mathbb{E}(Y) &= \int_0^{1/2} (1-x) \cdot 1 \, dx + \int_{1/2}^1 x \cdot 1 \, dx \\ &= \left[x - \frac{x^2}{2} \right]_0^{1/2} + \left[\frac{x^2}{2} \right]_{1/2}^1 \\ &= \left(\frac{1}{2} - \frac{1}{8} \right) + \left(\frac{1}{2} - \frac{1}{8} \right) = \frac{3}{4}\end{aligned}$$

```
# Visualizing the breaking stick problem
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1, 1000)
longer_piece = np.maximum(x, 1-x)

plt.figure(figsize=(7, 4))
plt.plot(x, longer_piece, 'b-', linewidth=3, label='Length of longer piece')
plt.fill_between(x, 0, longer_piece, alpha=0.3, color='lightblue')
plt.axhline(y=0.75, color='red', linestyle='--', linewidth=2,
            label='E[longer piece] = 3/4')
plt.axvline(x=0.5, color='gray', linestyle=':', alpha=0.5)
plt.xlabel('Break point (X)')
plt.ylabel('Length of longer piece')
plt.title('Breaking a Unit Stick: Expected Length of Longer Piece')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.tight_layout()
plt.show()
```



Example: Exponential Prize Game

A game show offers a prize based on rolling a die: if you roll X , you win 2^X euros. What's your expected winnings?

Solution

Using the lazy statistician's rule with $X \sim \text{DiscreteUniform}(1, 6)$:

$$\mathbb{E}(2^X) = \sum_{x=1}^6 2^x \cdot \frac{1}{6} = \frac{1}{6}(2^1 + 2^2 + \dots + 2^6)$$

Direct calculation:

$$= \frac{1}{6}(2 + 4 + 8 + 16 + 32 + 64) = \frac{126}{6} = 21$$

So you expect to win €21 on average.

Tip

Special case: Probability as expectation of indicator functions.

If A is an event and the indicator function is defined as:

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

then:

$$\mathbb{E}(I_A(X)) = \mathbb{P}(X \in A)$$

This shows that probability is just a special case of expectation!

This trick of using the indicator function with expectations is used commonly in probability, statistics and machine learning.

2.4 Properties of Expectation

2.4.1 The Linearity Property

If X_1, \dots, X_n are random variables and a_1, \dots, a_n are constants, then:

$$\mathbb{E}\left(\sum_{i=1}^n a_i X_i\right) = \sum_{i=1}^n a_i \mathbb{E}(X_i)$$

Possibly The Most Important Result in This Course

Expectation is LINEAR!

This property:

- Works WITHOUT independence (unlike the product rule)
- Simplifies hard calculations
- Is the key to understanding sampling distributions
- Will be used in almost every proof and application

If you remember only one thing from this chapter, remember that expectation is linear. You'll use it constantly throughout statistics and machine learning!

2.4.2 Applications of Linearity

The power of linearity becomes clear when we use it to solve problems that would be difficult otherwise.

Example: Binomial Mean via Indicator Decomposition

Let $X \sim \text{Binomial}(n, p)$. Finding $\mathbb{E}(X)$ directly requires evaluating:

$$\mathbb{E}(X) = \sum_{x=0}^n x \binom{n}{x} p^x (1-p)^{n-x}$$

Have fun calculating this! But with linearity, it's trivial.

Remember that $\text{Binomial}(n, p)$ is the distribution of the sum of n Bernoulli(p) random variables.

Thus, we can write $X = \sum_{i=1}^n X_i$, where X_i are independent Bernoulli(p) indicators.

With this, we have:

$$\mathbb{E}(X) = \mathbb{E}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \mathbb{E}(X_i) = \sum_{i=1}^n p = np$$

Done!

Example: Linearity Works Even with Dependent Variables

A common misconception is that linearity of expectation requires independence. It doesn't! Let's demonstrate this crucial fact by computing $\mathbb{E}[2X + 3Y]$ where X and Y are strongly correlated.

We'll generate X and Y with correlation 0.8 (highly dependent!) and verify that linearity still holds:

```
# Demonstrating linearity even with dependence
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n_sims = 10000

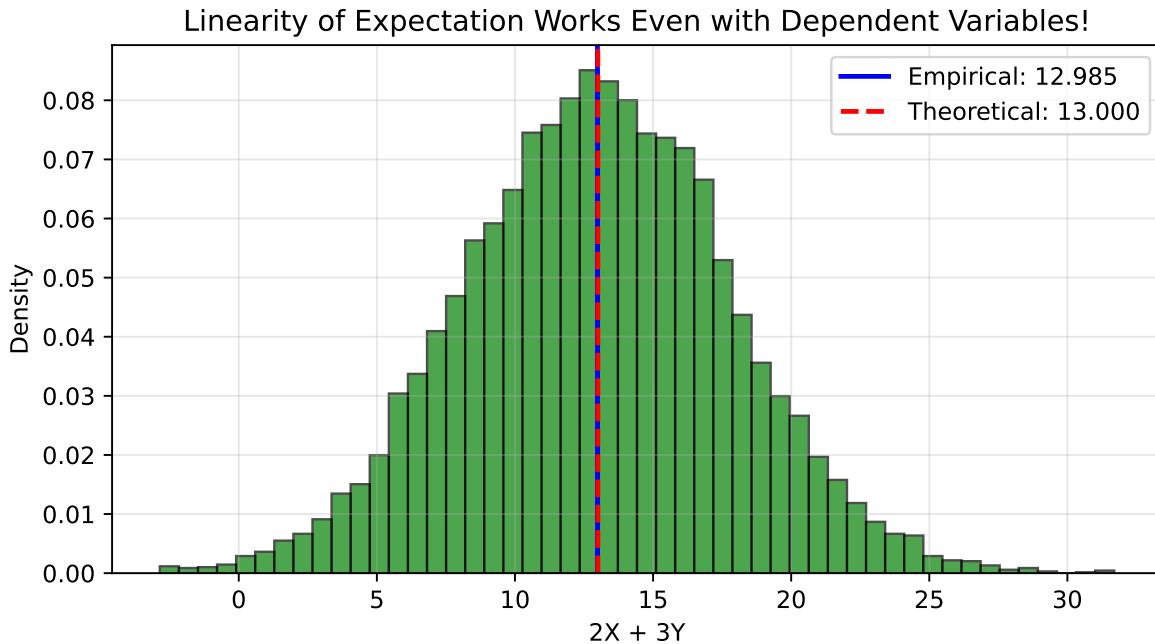
# Generate correlated X and Y
mean = [2, 3]
cov = [[1, 0.8], [0.8, 1]] # Correlation = 0.8
samples = np.random.multivariate_normal(mean, cov, n_sims)
X = samples[:, 0]
Y = samples[:, 1]

# Compute E[2X + 3Y] empirically
Z = 2*X + 3*Y
empirical_mean = np.mean(Z)

# Theoretical value using linearity
theoretical_mean = 2*mean[0] + 3*mean[1]

plt.figure(figsize=(7, 4))
plt.hist(Z, bins=50, density=True, alpha=0.7, color='green', edgecolor='black')
plt.axvline(empirical_mean, color='blue', linestyle='-', linewidth=2,
            label=f'Empirical: {empirical_mean:.3f}')
plt.axvline(theoretical_mean, color='red', linestyle='--', linewidth=2,
            label=f'Theoretical: {theoretical_mean:.3f}')
plt.xlabel('2X + 3Y')
plt.ylabel('Density')
plt.title('Linearity of Expectation Works Even with Dependent Variables!')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"X and Y are dependent (correlation = 0.8)")
print(f"But E[2X + 3Y] = 2E[X] + 3E[Y] still holds!")
print(f"Theoretical: 2×{mean[0]} + 3×{mean[1]} = {theoretical_mean}")
print(f"Empirical: {empirical_mean:.3f}")
```



X and Y are dependent (correlation = 0.8)

But $E[2X + 3Y] = 2E[X] + 3E[Y]$ still holds!

Theoretical: $2 \times 2 + 3 \times 3 = 13$

Empirical: 12.985

Key takeaway: Despite the strong correlation between X and Y, the empirical mean of $2X + 3Y$ matches the theoretical value $2E[X] + 3E[Y]$ perfectly. This is why linearity of expectation is so powerful—it works unconditionally!

i Additional Examples: More Applications of Linearity

Expected Number of Fixed Points in Random Permutation:

In a random permutation of $\{1, 2, \dots, n\}$, what's the expected number of elements that stay in their original position?

Let $X_i = 1$ if element i stays in position i , and 0 otherwise. The total number of fixed points is $X = \sum_{i=1}^n X_i$.

For any position i : $P(X_i = 1) = \frac{1}{n}$ (element i has probability $1/n$ of being in position i).

Therefore: $E(X_i) = \frac{1}{n}$

By linearity:

$$E(X) = \sum_{i=1}^n E(X_i) = \sum_{i=1}^n \frac{1}{n} = 1$$

Amazing! No matter how large n is, we expect exactly 1 fixed point on average.

Fortune Doubling Game (from Wasserman Exercise 3.1):

You start with c dollars. On each play, you either double your money or halve it, each with probability $1/2$. What's your expected fortune after n plays?

Let X_i be your fortune after i plays. Then: - $X_0 = c$ - $X_{i+1} = 2X_i$ with probability $1/2$ - $X_{i+1} = X_i/2$ with probability $1/2$

Using conditional expectation:

$$E(X_{i+1}|X_i) = \frac{1}{2}(2X_i) + \frac{1}{2}\left(\frac{X_i}{2}\right) = X_i + \frac{X_i}{4} = X_i$$

By the law of iterated expectations:

$$\mathbb{E}(X_{i+1}) = \mathbb{E}[\mathbb{E}(X_{i+1}|X_i)] = \mathbb{E}(X_i)$$

Therefore, by induction: $\mathbb{E}(X_n) = \mathbb{E}(X_0) = c$

Your expected fortune never changes! This is an example of a *martingale*—a fair game where the expected future value equals the current value.

2.4.3 Independence and Products

While expectation is linear for all random variables, products require independence.

If X_1, \dots, X_n are **independent** random variables, then:

$$\mathbb{E}\left(\prod_{i=1}^n X_i\right) = \prod_{i=1}^n \mathbb{E}(X_i)$$



Warning

This ONLY works with **independent** random variables! As a clear counterexample, $\mathbb{E}(X^2) \neq (\mathbb{E}(X))^2$ in general, since X and X are clearly not independent.

2.5 Variance and Its Properties

2.5.1 Measuring Spread

While expectation tells us the center of a distribution, variance measures how “spread out” it is.

Let X be a random variable with mean μ . The **variance** of X – denoted by σ^2 , σ_X^2 , $\mathbb{V}(X)$, $\mathbb{V}X$ or $\text{Var}(X)$ – is defined as:

$$\sigma^2 = \mathbb{V}(X) = \mathbb{E}[(X - \mu)^2]$$

assuming this expectation exists. The **standard deviation** is

$$\text{sd}(X) = \sqrt{\mathbb{V}(X)}$$

and is also denoted by σ and σ_X .

Notation: The lowercase Greek letter σ (**sigma**) is almost universally used to denote the standard deviation (and more generally the “scale” of a distribution, related to its spread).

i Why Both Variance and Standard Deviation?

Variance (σ^2) is in squared units—if X measures height in cm, then $\mathbb{V}(X)$ is in cm^2 . This makes it hard to interpret directly.

Standard deviation (σ) is in the same units as X , making it more interpretable: “typical deviation from the mean.”

So why use variance at all? Variance works better for doing math because:

- It has nicer properties (like additivity for independent variables, as we will see later)
- It appears naturally in formulas and proofs
- It’s easier to manipulate algebraically

In short: we do calculations with variance, then take the square root for interpretation.

Multiple Perspectives

Intuitive

Think of variance as measuring **how wrong your guess will typically be** if you always guess the mean.

Imagine predicting tomorrow's temperature. If you live in Nice or Lisbon (low variance), guessing the average temperature works well year-round. If you live in Helsinki or Berlin (high variance), that same strategy leads to large errors – you'll be way off in both summer and winter.

Standard deviation puts this in interpretable units:

- Low σ : Your guesses are usually close (precise manufacturing, stable processes)
- High σ : Your guesses are often far off (volatile stocks, unpredictable weather)

The famous **68-95-99.7 rule** tells us that for bell-shaped data:

- 68% of observations fall within 1σ of the mean
- 95% fall within 2σ
- 99.7% fall within 3σ

This is why “3-sigma events” are considered rare outliers in quality control.
(This rule is *exactly* true for normally-distributed data.)

Mathematical

Variance has an elegant mathematical interpretation as the **expected squared distance from the mean**:

$$\mathbb{V}(X) = \mathbb{E}[(X - \mu)^2]$$

This squared distance has deep connections:

1. **Minimization property**: The mean μ minimizes $\mathbb{E}[(X - c)^2]$ over all constants c
2. **Pythagorean theorem**: For independent X, Y :

$$\mathbb{V}(X + Y) = \mathbb{V}(X) + \mathbb{V}(Y)$$

Just like $|a + b|^2 = |a|^2 + |b|^2$ for perpendicular vectors!

3. **Information theory**: Variance of a Gaussian determines its entropy (uncertainty)

The quadratic nature (a^2 scaling) reflects that variance measures *squared deviations*: doubling the scale quadruples the variance.

Computational

Let's visualize how variance controls the spread of a distribution, using exam scores as an example.

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Set up the plot
fig, ax = plt.subplots(figsize=(7, 4))

# Common mean for all distributions
mean = 75
x = np.linspace(40, 110, 1000)

# Three different standard deviations
sigmas = [5, 10, 20]
colors = ['#2E86AB', '#A23B72', '#F18F01']
labels = [' = 5 (Low variance)', ' = 10 (Medium variance)', ' = 20 (High variance)']

# Plot each distribution
for sigma, color, label in zip(sigmas, colors, labels):
    y = stats.norm.pdf(x, mean, sigma)
    ax.plot(x, y, color=color, linewidth=2.5, label=label)

    # Shade ±1 region
    x_fill = x[(x >= mean - sigma) & (x <= mean + sigma)]
    y_fill = stats.norm.pdf(x_fill, mean, sigma)
    ax.fill_between(x_fill, y_fill, alpha=0.2, color=color)

# Add vertical line at mean
ax.axvline(mean, color='black', linestyle='--', alpha=0.7, linewidth=1.5)
ax.text(mean + 1, 0.085, 'Mean = 75', ha='left', va='bottom')

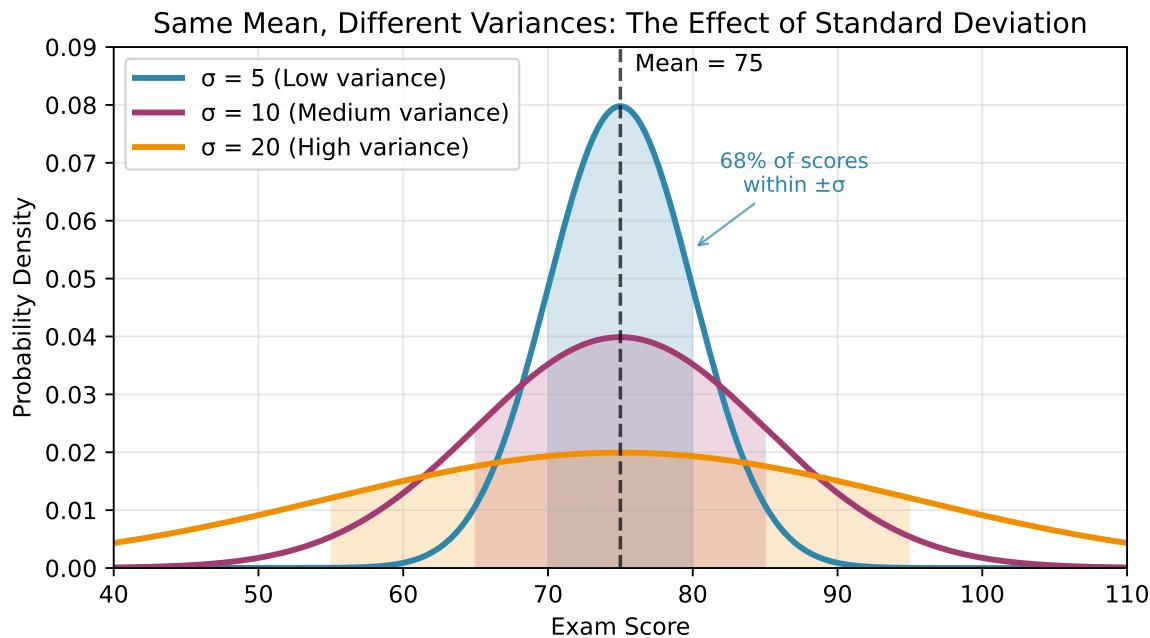
# Styling
ax.set_xlabel('Exam Score')
ax.set_ylabel('Probability Density')
ax.set_title('Same Mean, Different Variances: The Effect of Standard Deviation')
ax.legend(loc='upper left')
ax.set_xlim(40, 110)
ax.set_ylim(0, 0.09)
ax.grid(True, alpha=0.3)

# Add annotations for interpretation
ax.annotate('68% of scores\nwithin ± ', xy=(mean + 5, 0.055), xytext=(mean + 12, 0.065),
            arrowprops=dict(arrowstyle='->', color='#2E86AB', alpha=0.7),
            fontsize=9, ha='center', color='#2E86AB')

plt.tight_layout()
plt.show()

print("Interpreting the visualization:")
print("• Small (blue): Scores cluster tightly around 75. Most students perform similarly.")
print("• Medium (pink): Moderate spread. Typical variation in a well-designed exam.")
print("• Large (orange): Wide spread. Large differences in student performance.")
print(f"\nFor any normal distribution, about 68% of values fall within ±1 of the mean.")

```



Interpreting the visualization:

- Small (blue): Scores cluster tightly around 75. Most students perform similarly.
- Medium (pink): Moderate spread. Typical variation in a well-designed exam.
- Large (orange): Wide spread. Large differences in student performance.

For any normal distribution, about 68% of values fall within ± 1 of the mean.

2.5.2 Properties of Variance

The variance has a useful computational formula:

$$\mathbb{V}(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2$$

i Proof

Starting from the definition of variance:

$$\mathbb{V}(X) = \mathbb{E}[(X - \mu)^2] \quad (2.1)$$

$$= \mathbb{E}[X^2 - 2X\mu + \mu^2] \quad (2.2)$$

$$= \mathbb{E}(X^2) - 2\mu\mathbb{E}(X) + \mu^2 \quad (2.3)$$

$$= \mathbb{E}(X^2) - 2\mu^2 + \mu^2 \quad (2.4)$$

$$= \mathbb{E}(X^2) - \mu^2 \quad (2.5)$$

where we used linearity of expectation and the fact that $\mathbb{E}(X) = \mu$.

This formula simplifies many calculations and can be used to prove multiple properties of the variance.

Assuming the variance is well defined, it satisfies:

1. $\mathbb{V}(X) \geq 0$, with $\mathbb{V}(X) = 0$ if and only if X is constant (a.s.)¹

¹Almost surely (a.s.) means “with probability 1”. A random variable is almost surely constant if $\mathbb{P}(X = c) = 1$ for some

2. For constants a, b :

$$\mathbb{V}(aX + b) = a^2\mathbb{V}(X)$$

3. If X and Y are **independent**:

$$\mathbb{V}(X + Y) = \mathbb{V}(X) + \mathbb{V}(Y)$$

4. If X and Y are **independent**:

$$\mathbb{V}(X - Y) = \mathbb{V}(X) + \mathbb{V}(Y)$$

(not minus!)

5. If X_1, \dots, X_n are **independent**, for constants a_1, \dots, a_n :

$$\mathbb{V}\left(\sum_{i=1}^n a_i X_i\right) = \sum_{i=1}^n a_i^2 \mathbb{V}(X_i)$$

Property 4 often surprises students. You can find the proof below.

i Proof of Property 4

If X and Y are independent with means μ_X, μ_Y :

$$\mathbb{V}(X - Y) = \mathbb{E}[(X - Y - (\mu_X - \mu_Y))^2] \quad (2.6)$$

$$= \mathbb{E}[((X - \mu_X) - (Y - \mu_Y))^2] \quad (2.7)$$

$$= \mathbb{E}[(X - \mu_X)^2 - 2(X - \mu_X)(Y - \mu_Y) + (Y - \mu_Y)^2] \quad (2.8)$$

$$= \mathbb{E}[(X - \mu_X)^2] + \mathbb{E}[(Y - \mu_Y)^2] - 2\mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \quad (2.9)$$

$$= \mathbb{V}(X) + \mathbb{V}(Y) - 2 \cdot 0 \quad (2.10)$$

$$= \mathbb{V}(X) + \mathbb{V}(Y) \quad (2.11)$$

The key step uses independence: $\mathbb{E}[(X - \mu_X)(Y - \mu_Y)] = \mathbb{E}[X - \mu_X]\mathbb{E}[Y - \mu_Y] = 0 \cdot 0 = 0$.

Let's visualize how subtracting **independent** variables increases variance, while subtracting **dependent** variables can reduce it – to the point that subtracting perfectly correlated variables completely eliminates any variance!

constant c . The “almost” acknowledges that technically there could be probability-0 events where $X \neq c$, but these never occur in practice.

```
# Visualizing Var(X-Y) = Var(X) + Var(Y) for independent variables
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 10000

# Independent case
X_indep = np.random.normal(0, 1, n) # Var = 1
Y_indep = np.random.normal(0, 1, n) # Var = 1
diff_indep = X_indep - Y_indep # Var should be 2

# Perfectly correlated case (not independent)
X_corr = np.random.normal(0, 1, n)
Y_corr = X_corr # Perfect correlation
diff_corr = X_corr - Y_corr # Should be 0

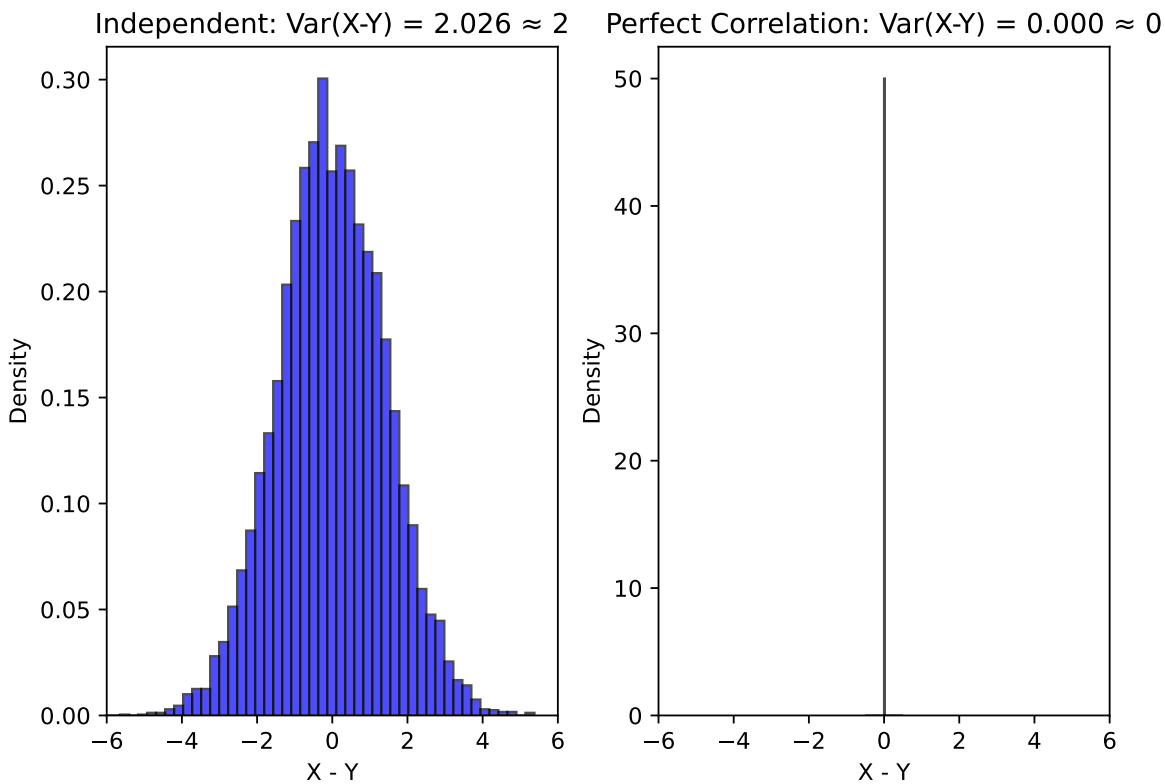
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

# Independent case
ax1.hist(diff_indep, bins=50, density=True, alpha=0.7, color='blue', edgecolor='black')
ax1.set_xlabel('X - Y')
ax1.set_ylabel('Density')
ax1.set_title(f'Independent: Var(X-Y) = {np.var(diff_indep, ddof=1):.3f} 2')
ax1.set_xlim(-6, 6)

# Correlated case
ax2.hist(diff_corr, bins=50, density=True, alpha=0.7, color='red', edgecolor='black')
ax2.set_xlabel('X - Y')
ax2.set_ylabel('Density')
ax2.set_title(f'Perfect Correlation: Var(X-Y) = {np.var(diff_corr, ddof=1):.3f} 0')
ax2.set_xlim(-6, 6)

plt.tight_layout()
plt.show()

print("When X and Y are independent N(0,1):")
print(f"  Var(X) = {np.var(X_indep, ddof=1):.3f}")
print(f"  Var(Y) = {np.var(Y_indep, ddof=1):.3f}")
print(f"  Var(X-Y) = {np.var(diff_indep, ddof=1):.3f}  Var(X) + Var(Y) = 2")
print("\nWhen Y = X (perfect dependence):")
print(f"  Var(X-Y) = Var(0) = 0")
```



When X and Y are independent $N(0,1)$:

$$\text{Var}(X) = 1.007$$

$$\text{Var}(Y) = 1.002$$

$$\text{Var}(X-Y) = 2.026 \quad \text{Var}(X) + \text{Var}(Y) = 2$$

When $Y = X$ (perfect dependence):

$$\text{Var}(X-Y) = \text{Var}(0) = 0$$

Example: Variance of Binomial via Decomposition

Let $X \sim \text{Binomial}(n, p)$. We already know $\mathbb{E}(X) = np$. What's the variance?

i Solution

Write $X = \sum_{i=1}^n X_i$ where $X_i \sim \text{Bernoulli}(p)$ independently.

For a single Bernoulli:

- $\mathbb{E}(X_i) = p$
- $\mathbb{E}(X_i^2) = 0^2 \cdot (1-p) + 1^2 \cdot p = p$
- $\mathbb{V}(X_i) = \mathbb{E}(X_i^2) - (\mathbb{E}(X_i))^2 = p - p^2 = p(1-p)$

Since the X_i are independent:

$$\mathbb{V}(X) = \mathbb{V}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \mathbb{V}(X_i) = np(1-p)$$

Note that variance is maximized when $p = 1/2$, which makes intuitive sense – there's most uncertainty when success and failure are equally likely.

i Mean and Variance of Common Distributions

Distribution	$\mathbb{E}[X]$	$\mathbb{V}(X)$	When to Use
Bernoulli(p)	p	$p(1-p)$	Single yes/no trial
Binomial(n, p)	np	$np(1-p)$	Count of successes
Poisson(λ)	λ	λ	Count of rare events
Geometric(p)	$1/p$	$(1-p)/p^2$	Trials until success
Uniform(a, b)	$(a+b)/2$	$(b-a)^2/12$	Equal likelihood
Normal(μ, σ^2)	μ	σ^2	Sums of many effects
Exponential(β)	β	β^2	Time between events
Gamma(α, β)	$\alpha\beta$	$\alpha\beta^2$	Sum of exponentials
Beta(α, β)	$\alpha/(\alpha+\beta)$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	Proportions
t_ν	0 (if $\nu > 1$)	$\nu/(\nu-2)$ (if $\nu > 2$)	Heavy-tailed data
χ_p^2	p	$2p$	Sum of squared normals

2.6 Sample Mean and Variance

When we observe data, we compute sample statistics to estimate population parameters.

Recall from our introduction: we have a **sample** X_1, \dots, X_n drawn from a **population** distribution F_X . The population has true parameters (like $\mu = \mathbb{E}(X)$ and $\sigma^2 = \mathbb{V}(X)$) that we want to know, but we can only compute statistics from our finite sample. This gap between what we can calculate and what we want to know is fundamental to statistics.

Given random variables X_1, \dots, X_n :

The **sample mean** is:

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$$

The **sample variance** is:

$$S_n^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2$$

Note the $n - 1$ in the denominator of the sample variance. This makes it an *unbiased* estimator of the population variance (see below).

Let X_1, \dots, X_n be IID with $\mu = \mathbb{E}(X_i)$ and $\sigma^2 = \mathbb{V}(X_i)$. Then:

$$\mathbb{E}(\bar{X}_n) = \mu, \quad \mathbb{V}(\bar{X}_n) = \frac{\sigma^2}{n}, \quad \mathbb{E}(S_n^2) = \sigma^2$$

This theorem tells us:

- The sample mean is unbiased (its expectation is equal to the population mean)
- Its variance decreases as n increases
- The sample variance (with $n - 1$) is unbiased

i Wait, What Does “Unbiased” Mean?

An estimator or sample statistic is **unbiased** if its expected value equals the parameter it's trying to estimate.

- **Unbiased:** $\mathbb{E}(\text{estimator}) = \text{true parameter}$
- **Biased:** $\mathbb{E}(\text{estimator}) \neq \text{true parameter}$

For example:

- As stated above, \bar{X}_n is unbiased for μ because $\mathbb{E}(\bar{X}_n) = \mu$
- S_n^2 (with $n - 1$) is unbiased for σ^2 because $\mathbb{E}(S_n^2) = \sigma^2$
- If we used n instead of $n - 1$ at the denominator, we'd get $\mathbb{E}(S_n^2) = \frac{n-1}{n}\sigma^2 < \sigma^2$ (biased!)

Being unbiased means that *on average* across many sets of samples, our sample statistic would match the true value – though any individual estimate may be too high or too low. This also doesn't tell us anything about the *rate of convergence* – how fast the estimator converges to the true value.

2.7 Covariance and Correlation

2.7.1 Linear Relationships

When we have two random variables, we often want to measure how they vary together and quantify the strength of their *linear* relation.

Let X and Y be random variables with means μ_X and μ_Y and standard deviations σ_X and σ_Y . The **covariance** between X and Y is:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)]$$

The **correlation** is:

$$\rho = \rho_{X,Y} = \rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

2.7.2 Properties of Covariance and Correlation

The covariance can be rewritten as:

$$\text{Cov}(X, Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$$

The correlation satisfies:

$$-1 \leq \rho(X, Y) \leq 1$$

The correlation is a sort of “normalized covariance”. By dividing the covariance by σ_X and σ_Y , we remove the *magnitude* (and units/scale) of the two random variables, and what remains is a pure number that measures of how much they change together on average, in a range from -1 to 1.

Covariance and correlation further satisfy the following properties:

- If $Y = aX + b$ for constants a, b :

$$\rho(X, Y) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{if } a < 0 \end{cases}$$

- If X and Y are **independent**: $\text{Cov}(X, Y) = \rho = 0$
- The converse is NOT true in general!

 Warning

Common Misconception: Uncorrelated Independent!

Independence implies zero correlation, but zero correlation does NOT imply independence.

Example: Uncorrelated but Dependent

Let $X \sim \text{Uniform}(-1, 1)$ and $Y = X^2$.

These two random variables are clearly dependent (knowing X determines Y exactly!), but:

$$\begin{aligned}\mathbb{E}(X) &= 0 \\ \mathbb{E}(Y) &= \mathbb{E}(X^2) = \int_{-1}^1 x^2 \cdot \frac{1}{2} dx = \frac{1}{3} \\ \mathbb{E}(XY) &= \mathbb{E}(X^3) = \int_{-1}^1 x^3 \cdot \frac{1}{2} dx = 0\end{aligned}$$

Therefore:

$$\text{Cov}(X, Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y) = 0 - 0 \cdot \frac{1}{3} = 0$$

So X and Y are **uncorrelated** despite being perfectly dependent!

The plot below shows X and Y . See also [this article](#) on Scientific American for more examples.

```
# Visualizing uncorrelated but dependent variables
import numpy as np
import matplotlib.pyplot as plt

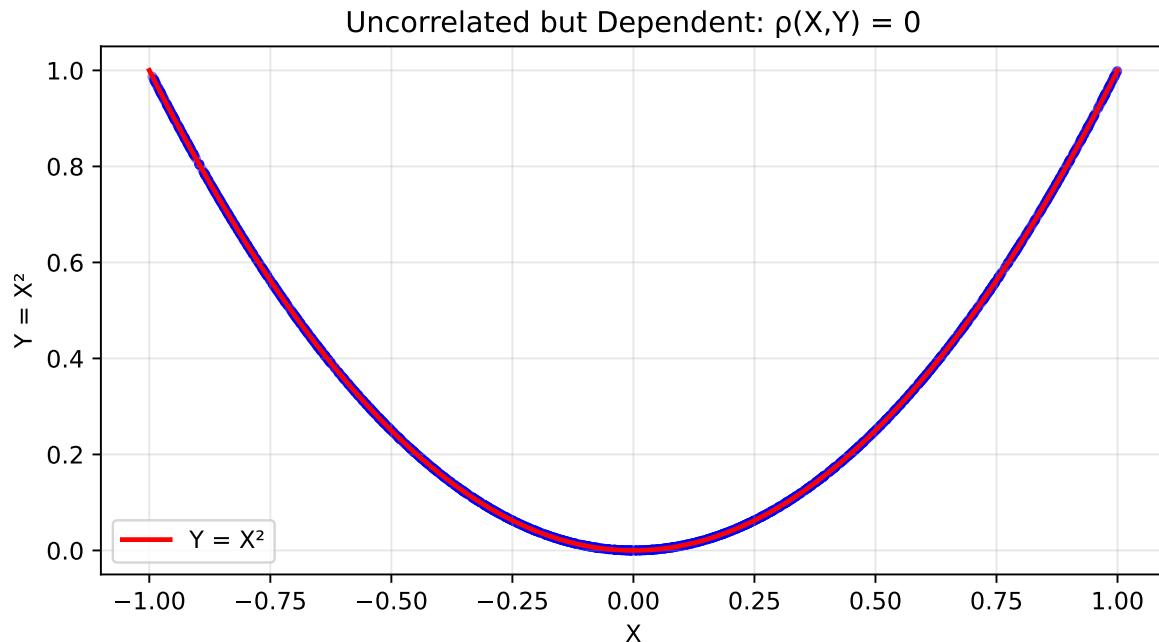
np.random.seed(42)
n = 2000

# X ~ Uniform(-1, 1), Y = X^2
X = np.random.uniform(-1, 1, n)
Y = X**2

plt.figure(figsize=(7, 4))
plt.scatter(X, Y, alpha=0.5, s=10, color='blue')
plt.xlabel('X')
plt.ylabel('Y = X^2')
plt.title('Uncorrelated but Dependent: (X,Y) = 0')
plt.grid(True, alpha=0.3)

# Add the parabola
x_line = np.linspace(-1, 1, 100)
y_line = x_line**2
plt.plot(x_line, y_line, 'r-', linewidth=2, label='Y = X^2')
plt.legend()
plt.tight_layout()
plt.show()

print("Y is completely determined by X, yet they are uncorrelated!")
print("This is because the linear association is zero due to symmetry.")
```



Y is completely determined by X, yet they are uncorrelated!
This is because the linear association is zero due to symmetry.

2.7.3 Variance of Sums (General Case)

We saw in Section 2.5.2 that for **independent** variables,

$$\mathbb{V} \left(\sum_{i=1}^n a_i X_i \right) = \sum_{i=1}^n a_i^2 \mathbb{V}(X_i) \quad (\text{independent})$$

We now generalize this result to any random variables, whether dependent or independent:

$$\mathbb{V} \left(\sum_{i=1}^n a_i X_i \right) = \sum_{i=1}^n a_i^2 \mathbb{V}(X_i) + 2 \sum_{i=1}^n \sum_{j=i+1}^n a_i a_j \text{Cov}(X_i, X_j)$$

Special cases:

- $\mathbb{V}(X + Y) = \mathbb{V}(X) + \mathbb{V}(Y) + 2\text{Cov}(X, Y)$
- $\mathbb{V}(X - Y) = \mathbb{V}(X) + \mathbb{V}(Y) - 2\text{Cov}(X, Y)$
- When the variables are independent, check that this indeed reduces to the simpler formula for independent variables

2.8 Expectation with Matrices

2.8.1 Random Vectors

In *multivariate* settings – that is, involving multiple random variables –, we work with random *vectors* and their expectations.

Notation: The mathematical convention is to work with column vectors. For example, we write

$$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

rather than (X_1, X_2, X_3) . The column vector can also be written as the transpose of a row vector: $\mathbf{X} = (X_1, X_2, X_3)^T$.

For a random vector $\mathbf{X} = (X_1, \dots, X_k)^T$:

The **mean vector** is:

$$\mu = \mathbb{E}(\mathbf{X}) = \begin{pmatrix} \mathbb{E}(X_1) \\ \vdots \\ \mathbb{E}(X_k) \end{pmatrix}$$

The **covariance matrix** Σ (also written $\mathbb{V}(\mathbf{X})$) is:

$$\Sigma = \begin{bmatrix} \mathbb{V}(X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_k) \\ \text{Cov}(X_2, X_1) & \mathbb{V}(X_2) & \dots & \text{Cov}(X_2, X_k) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_k, X_1) & \text{Cov}(X_k, X_2) & \dots & \mathbb{V}(X_k) \end{bmatrix}$$

The inverse Σ^{-1} is called the **precision matrix**.

Notation: The *uppercase* Greek letter Σ (**sigma**) is almost invariably used to denote a covariance matrix. Note that the *lowercase* σ denotes the standard deviation.

2.8.2 Covariance Matrix Properties

The covariance matrix can be written compactly as:

$$\Sigma = \mathbb{E}[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T]$$

An alternative formula for the covariance matrix is:

$$\Sigma = \mathbb{E}(\mathbf{X}\mathbf{X}^T) - \mu\mu^T$$

i Proof

Starting from the definition:

$$\Sigma = \mathbb{E}[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T] \quad (2.12)$$

$$= \mathbb{E}[\mathbf{X}\mathbf{X}^T - \mathbf{X}\mu^T - \mu\mathbf{X}^T + \mu\mu^T] \quad (2.13)$$

$$= \mathbb{E}(\mathbf{X}\mathbf{X}^T) - \mathbb{E}(\mathbf{X})\mu^T - \mu\mathbb{E}(\mathbf{X}^T) + \mu\mu^T \quad (2.14)$$

$$= \mathbb{E}(\mathbf{X}\mathbf{X}^T) - \mu\mu^T - \mu\mu^T + \mu\mu^T \quad (2.15)$$

$$= \mathbb{E}(\mathbf{X}\mathbf{X}^T) - \mu\mu^T \quad (2.16)$$

where we used the fact that $\mathbb{E}(\mathbf{X}) = \mu$ and the linearity of expectation.

Properties:

- Symmetric: $\Sigma = \Sigma^T$
- Positive semi-definite: $\mathbf{a}^T \Sigma \mathbf{a} \geq 0$ for all \mathbf{a}
- Diagonal elements are variances (non-negative)
- Off-diagonal elements are covariances

2.8.3 Linear Transformations

If \mathbf{X} has mean μ and covariance Σ , and \mathbf{A} is a matrix:

$$\mathbb{E}(\mathbf{AX}) = \mathbf{A}\mu$$

$$\mathbb{V}(\mathbf{AX}) = \mathbf{A}\Sigma\mathbf{A}^T$$

i Proof of the Variance Formula

Using the definition of variance for vectors and the fact that $\mathbb{E}(\mathbf{AX}) = \mathbf{A}\mu$:

$$\mathbb{V}(\mathbf{AX}) = \mathbb{E}[(\mathbf{AX} - \mathbf{A}\mu)(\mathbf{AX} - \mathbf{A}\mu)^T] \quad (2.17)$$

$$= \mathbb{E}[\mathbf{A}(\mathbf{X} - \mu)(\mathbf{A}(\mathbf{X} - \mu))^T] \quad (2.18)$$

$$= \mathbb{E}[\mathbf{A}(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T \mathbf{A}^T] \quad (2.19)$$

$$= \mathbf{A}\mathbb{E}[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T]\mathbf{A}^T \quad (2.20)$$

$$= \mathbf{A}\Sigma\mathbf{A}^T \quad (2.21)$$

where we used the fact that \mathbf{A} is a constant matrix that can be taken outside the expectation.

Similarly, for a vector \mathbf{a} (this is just a special case of the equations above – why?):

$$\mathbb{E}(\mathbf{a}^T \mathbf{X}) = \mathbf{a}^T \mu$$

$$\mathbb{V}(\mathbf{a}^T \mathbf{X}) = \mathbf{a}^T \Sigma \mathbf{a}$$

2.8.4 Interpreting the Covariance Matrix

The covariance matrix encodes the second-order structure of a random vector—that is, how the variables vary and co-vary together. To understand this structure, we can examine its **spectral decomposition** ([eigendecomposition](#)).

Multiple Perspectives

Intuitive

Imagine your data as a cloud of points in space. This cloud rarely forms a perfect sphere—it's usually stretched more in some directions than others, like an ellipse or ellipsoid.

The covariance matrix captures this shape:

- **Eigenvectors** are the “natural axes” of your data cloud—the directions along which it stretches
- **Eigenvalues** tell you how much the cloud stretches in each direction
- The largest eigenvalue corresponds to the direction of greatest spread

This is like finding the best way to orient a box around your data:

- The box edges align with the eigenvectors
- The box dimensions are proportional to the square roots of eigenvalues

Principal Component Analysis (PCA) uses this insight: keep the directions with large spread (high variance), discard those with little spread. This reduces dimensions while preserving most of the data’s structure.

Mathematical

Since Σ is symmetric and positive semi-definite, recall from earlier linear algebra classes that it has spectral decomposition:

$$\Sigma = \sum_{i=1}^k \lambda_i \mathbf{v}_i \mathbf{v}_i^T$$

where:

- $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k \geq 0$ are the *eigenvalues* (which we can order from larger to smaller)
- $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ are the corresponding orthonormal *eigenvectors*

This decomposition reveals the geometric structure of the data:

- **Eigenvalues** λ_i : represent the variance along each principal axis
- **Eigenvectors** \mathbf{v}_i : define the directions of these principal axes
- **Largest eigenvalue/vector**: indicates the direction of maximum variance in the data

Computational

Let’s visualize how eigendecomposition reveals the structure of data:

```

import numpy as np
import matplotlib.pyplot as plt

# Generate correlated 2D data
np.random.seed(42)
mean = [2, 3]
cov = [[2.5, 1.5],
       [1.5, 1.5]]
data = np.random.multivariate_normal(mean, cov, 300)

# Compute eigendecomposition
eigenvalues, eigenvectors = np.linalg.eigh(cov)
# Sort by eigenvalue (largest first)
idx = eigenvalues.argsort()[:-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Plot the data and principal axes
plt.figure(figsize=(7, 6))
plt.scatter(data[:, 0], data[:, 1], alpha=0.5, s=30)

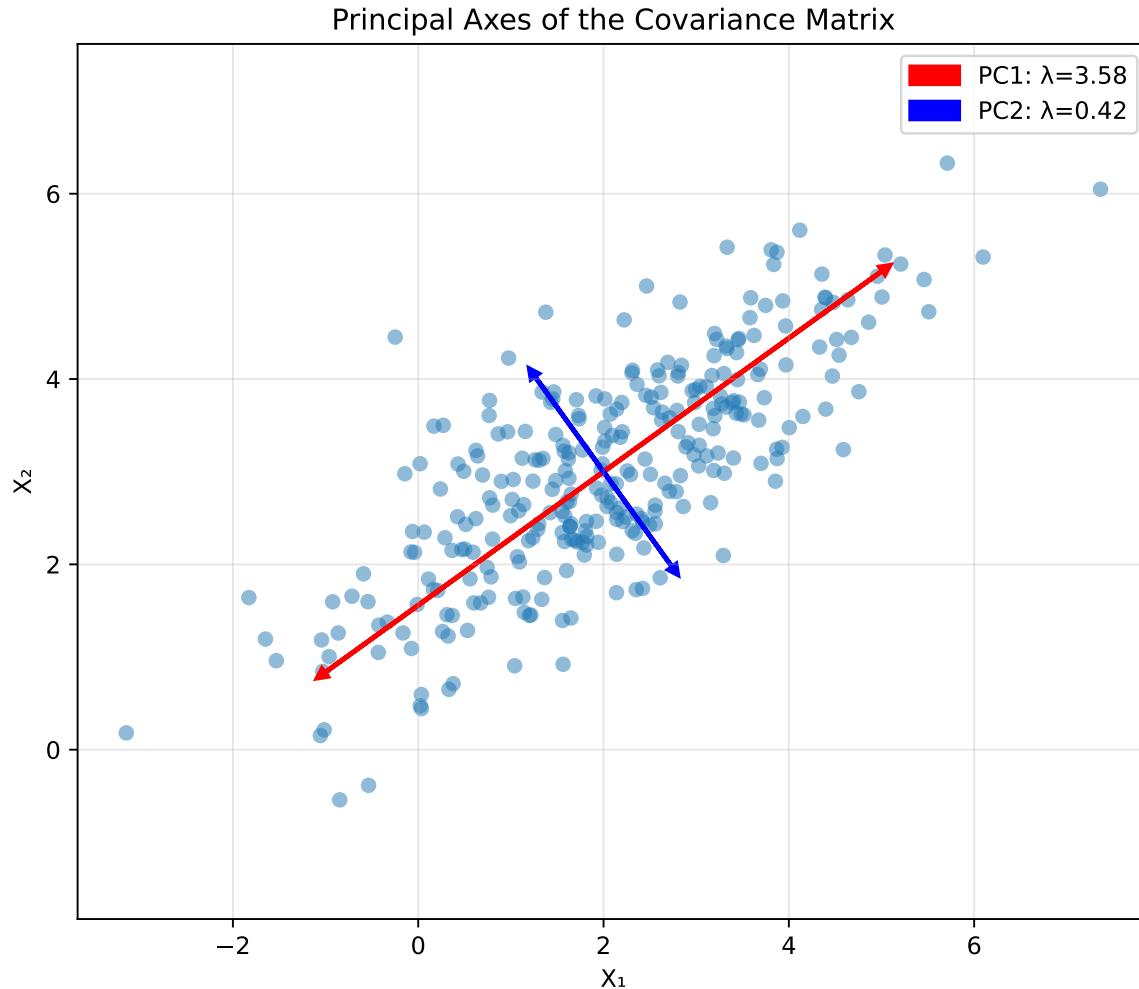
# Plot eigenvectors from the mean
colors = ['red', 'blue']
for i in range(2):
    # Scale eigenvector by sqrt(eigenvalue) for visualization
    v = eigenvectors[:, i] * np.sqrt(eigenvalues[i]) * 1.96
    plt.arrow(mean[0], mean[1], v[0], v[1],
              head_width=0.1, head_length=0.1,
              fc=colors[i], ec=colors[i], linewidth=2,
              label=f'PC{i+1}: ={eigenvalues[i]:.2f}'')

    # Also draw the negative direction
    plt.arrow(mean[0], mean[1], -v[0], -v[1],
              head_width=0.1, head_length=0.1,
              fc=colors[i], ec=colors[i], linewidth=2)

plt.xlabel('X ')
plt.ylabel('X ')
plt.title('Principal Axes of the Covariance Matrix')
plt.legend()
plt.axis('equal')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Covariance matrix:\n{np.array(cov)}")
print(f"\nEigenvalues: {eigenvalues}")
print(f"Eigenvectors (as columns):\n{eigenvectors}")
print(f"\nThe first principal component explains {100*eigenvalues[0]/sum(eigenvalues):.1f}% of the variance")

```



Covariance matrix:

```
[[2.5 1.5]
 [1.5 1.5]]
```

Eigenvalues: [3.58113883 0.41886117]

Eigenvectors (as columns):

```
[[[-0.81124219  0.58471028]
 [-0.58471028 -0.81124219]]]
```

The first principal component explains 89.5% of the variance

The red arrow shows the first principal component (direction of maximum variance), while the blue arrow shows the second. In the plot, each eigenvector \mathbf{v}_i is rescaled by $1.96\sqrt{\lambda_i}$ which covers about 95% of the normal distribution.

In Principal Component Analysis (PCA), we might keep only the first component to reduce from 2D to 1D while preserving most of the structure.

Example: Multinomial Covariance Structure

The [multinomial distribution](#) provides a concrete example of covariance matrix structure. If

$$\mathbf{X} = (X_1, \dots, X_k)^T \sim \text{Multinomial}(n, \mathbf{p})$$

where $\mathbf{p} = (p_1, \dots, p_k)^T$ with $\sum p_i = 1$, then:

Mean vector: $\mathbb{E}(\mathbf{X}) = n\mathbf{p} = (np_1, \dots, np_k)^T$

Covariance matrix:

$$\Sigma = \begin{pmatrix} np_1(1-p_1) & -np_1p_2 & \cdots & -np_1p_k \\ -np_2p_1 & np_2(1-p_2) & \cdots & -np_2p_k \\ \vdots & \vdots & \ddots & \vdots \\ -np_kp_1 & -np_kp_2 & \cdots & np_k(1-p_k) \end{pmatrix}$$

Key observations:

- Diagonal: $\mathbb{V}(X_i) = np_i(1-p_i)$ (same as binomial)
- Off-diagonal: $\text{Cov}(X_i, X_j) = -np_i p_j$ for $i \neq j$ (always negative!)
- Intuition: If more outcomes fall in category i , fewer can fall in category j

Special case: For a die roll with equal probabilities ($p_i = 1/6$ for all i):

- $\mathbb{V}(X_i) = n \cdot \frac{1}{6} \cdot \frac{5}{6} = \frac{5n}{36}$
- $\text{Cov}(X_i, X_j) = -n \cdot \frac{1}{6} \cdot \frac{1}{6} = -\frac{n}{36}$ for $i \neq j$

2.9 Conditional Expectation

2.9.1 Expectation Given Information

Conditional expectation captures how the mean changes when we have additional information. It is computed similarly to a regular expectation, just replacing the pdf (or PMF) with a *conditional* pdf (or PMF).

The **conditional expectation** of X given $Y = y$ is:

$$\mathbb{E}(X|Y = y) = \begin{cases} \sum_x x \mathbb{P}_{X|Y}(x|y) & \text{discrete case} \\ \int x f_{X|Y}(x|y) dx & \text{continuous case} \end{cases}$$

⚠ Warning

Subtle but Important:

- $\mathbb{E}(X)$ is a number
- $\mathbb{E}(X|Y = y)$ is a number (for fixed y)
- $\mathbb{E}(X|Y)$ is a random variable (because it's a function of Y !)

2.9.2 Properties of Conditional Expectation

$$\mathbb{E}[\mathbb{E}(Y|X)] = \mathbb{E}(Y)$$

More generally, for any function $r(x, y)$:

$$\mathbb{E}[\mathbb{E}(r(X, Y)|X)] = \mathbb{E}(r(X, Y))$$

i Proof of the Law of Iterated Expectations

We'll prove the first equation. Using the definition of conditional expectation and the fact that the joint pdf can be written as $f(x, y) = f_X(x)f_{Y|X}(y|x)$:

$$\mathbb{E}[\mathbb{E}(Y|X)] = \mathbb{E}\left[\int yf_{Y|X}(y|X) dy\right] \quad (2.22)$$

$$= \int \left[\int yf_{Y|X}(y|x) dy\right] f_X(x) dx \quad (2.23)$$

$$= \int \int yf_{Y|X}(y|x)f_X(x) dy dx \quad (2.24)$$

$$= \int \int yf(x, y) dy dx \quad (2.25)$$

$$= \int y \left[\int f(x, y) dx\right] dy \quad (2.26)$$

$$= \int yf_Y(y) dy \quad (2.27)$$

$$= \mathbb{E}(Y) \quad (2.28)$$

The key steps are:

1. $\mathbb{E}(Y|X)$ is a function of X , so we take its expectation with respect to X
2. We can interchange the order of integration
3. $f_{Y|X}(y|x)f_X(x) = f(x, y)$ by the definition of conditional probability
4. Integrating the joint pdf over x gives the marginal pdf $f_Y(y)$

This powerful result lets us compute expectations by conditioning on useful information.

Example: Breaking Stick Revisited

Imagine placing two random points on a unit stick. First, we place point X uniformly at random. Then, we place point Y uniformly at random between X and the end of the stick.

Formally: Draw $X \sim \text{Uniform}(0, 1)$. After observing $X = x$, draw $Y|X = x \sim \text{Uniform}(x, 1)$.

Question: What is the expected position of the second point Y ?

i Solution

We could find the marginal distribution of Y (which is complex), but it's easier to use conditional expectation:

First, find $\mathbb{E}(Y|X = x)$:

$$\mathbb{E}(Y|X = x) = \frac{x + 1}{2}$$

This makes sense: given $X = x$, point Y is uniform on $(x, 1)$, so its expected position is the midpoint.

So $\mathbb{E}(Y|X) = \frac{X+1}{2}$ (a random variable).

Now use iterated expectations:

$$\mathbb{E}(Y) = \mathbb{E}[\mathbb{E}(Y|X)] = \mathbb{E}\left[\frac{X+1}{2}\right] = \frac{\mathbb{E}(X)+1}{2} = \frac{1/2+1}{2} = \frac{3}{4}$$

The second point lands, on average, at position $3/4$ along the stick.

2.9.3 Conditional Variance

The **conditional variance** is:

$$\mathbb{V}(Y|X = x) = \mathbb{E}[(Y - \mathbb{E}(Y|X = x))^2 | X = x]$$

$$\mathbb{V}(Y) = \mathbb{E}[\mathbb{V}(Y|X)] + \mathbb{V}[\mathbb{E}(Y|X)]$$

This decomposition says: Total variance = Average within-group variance + Between-group variance.

2.10 More About the Normal Distribution

2.10.1 Quick Recap

Recall that if $X \sim \mathcal{N}(\mu, \sigma^2)$, then:

- PDF: $f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
- Mean: $\mathbb{E}(X) = \mu$
- Variance: $\mathbb{V}(X) = \sigma^2$

The normal distribution plays a central role in statistics due to the Central Limit Theorem (Chapter 3) and its many convenient mathematical properties.

2.10.2 Entropy of the Normal Distribution

We can use the expectation to compute the entropy of the normal distribution.

Example: Normal Distribution Entropy

The **differential entropy** of a continuous random variable measures the average uncertainty in the distribution:

$$H(X) = \mathbb{E}[-\ln f_X(X)] = - \int f_X(x) \ln f_X(x) dx$$

Let's calculate this for $X \sim \mathcal{N}(\mu, \sigma^2)$. First, find $-\ln f_X(x)$:

$$-\ln f_X(x) = \ln(\sqrt{2\pi\sigma^2}) + \frac{(x-\mu)^2}{2\sigma^2} = \frac{1}{2} \ln(2\pi\sigma^2) + \frac{(x-\mu)^2}{2\sigma^2}$$

Now compute the expectation:

$$H(X) = \mathbb{E}[-\ln f_X(X)] \tag{2.29}$$

$$= \mathbb{E}\left[\frac{1}{2} \ln(2\pi\sigma^2) + \frac{(X-\mu)^2}{2\sigma^2}\right] \tag{2.30}$$

$$= \frac{1}{2} \ln(2\pi\sigma^2) + \frac{1}{2\sigma^2} \mathbb{E}[(X-\mu)^2] \tag{2.31}$$

$$= \frac{1}{2} \ln(2\pi\sigma^2) + \frac{1}{2\sigma^2} \cdot \sigma^2 \tag{2.32}$$

$$= \frac{1}{2} \ln(2\pi\sigma^2) + \frac{1}{2} \tag{2.33}$$

$$= \frac{1}{2} \ln(2\pi e\sigma^2) \tag{2.34}$$

$$= \ln(\sqrt{2\pi e\sigma^2}) \tag{2.35}$$

Key insights:

- The entropy increases with σ (more spread = more uncertainty)
- Among all distributions with fixed variance σ^2 , the normal has maximum entropy

2.10.3 Multivariate Normal Properties

The d -dimensional multivariate normal is parametrized by mean vector $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ (symmetric and positive definite).

For $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$:

1. **Independence and diagonal covariance:** Components X_i and X_j are independent if and only if $\Sigma_{ij} = 0$. Thus, the components are mutually independent if and only if Σ is diagonal.
2. **Standard multivariate normal:** If $Z_1, \dots, Z_d \sim \mathcal{N}(0, 1)$ are independent, then $\mathbf{Z} = (Z_1, \dots, Z_d)^T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, where \mathbf{I}_d is the $d \times d$ identity matrix.
3. **Marginals are normal:** Each $X_i \sim \mathcal{N}(\mu_i, \Sigma_{ii})$
4. **Linear combinations are normal:** For any vector \mathbf{a} , $\mathbf{a}^T \mathbf{X} \sim \mathcal{N}(\mathbf{a}^T \mu, \mathbf{a}^T \Sigma \mathbf{a})$
5. **Conditionals are normal:** Suppose we partition:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Then the conditional distribution of \mathbf{X}_2 given $\mathbf{X}_1 = \mathbf{x}_1$ is:

$$\mathbf{X}_2 | \mathbf{X}_1 = \mathbf{x}_1 \sim \mathcal{N}(\mu_{2|1}, \Sigma_{2|1})$$

where:

- **Conditional mean:** $\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{x}_1 - \mu_1)$
- **Conditional covariance:** $\Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}$

Note that the conditional covariance doesn't depend on the observed value \mathbf{x}_1 !

These properties are used in multiple algorithms and methods in statistics, including for example:

- Gaussian processes
- Kalman filtering
- Linear regression theory
- Multivariate statistical methods

i Cholesky Decomposition

Every symmetric positive definite covariance matrix Σ can be decomposed as:

$$\Sigma = \mathbf{L}\mathbf{L}^T$$

where \mathbf{L} is a lower-triangular matrix called the **Cholesky decomposition** (or Cholesky factor).

This decomposition is crucial for:

- **Simulating multivariate normals:** If $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{X} = \mu + \mathbf{L}\mathbf{Z}$, then $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$
- **Transforming to standard form:** If $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$, then $\mathbf{L}^{-1}(\mathbf{X} - \mu) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- **Efficient computation:** Solving linear systems and computing determinants

Example: Generating Multivariate Normal Random Vectors via Cholesky

Let's see how the Cholesky decomposition can be used to transform independent standard normals into correlated multivariate normals.

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse

# Define mean and covariance
mu = np.array([1, 2])
Sigma = np.array([[2, 1.2],
                 [1.2, 1]])

# Compute Cholesky decomposition
L = np.linalg.cholesky(Sigma)
print("Covariance matrix Σ:")
print(Sigma)
print("\nCholesky factor L (lower triangular):")
print(L)
print("\nVerification: L @ L.T =")
print(L @ L.T)

# Generate samples step by step
np.random.seed(42)
n_samples = 500

# Step 1: Generate independent standard normals
Z = np.random.standard_normal((n_samples, 2)) # N(0, I)

# Step 2: Transform using Cholesky
X = mu + Z @ L.T # Transform to N(mu, Sigma)

# Visualize the transformation
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

# Plot independent standard normals
ax1.scatter(Z[:, 0], Z[:, 1], alpha=0.5, s=20, color='blue')
ax1.set_xlabel('Z ')
ax1.set_ylabel('Z ')
ax1.set_title('Independent N(0,1)')
ax1.set_xlim(-4, 4)
ax1.set_ylim(-4, 4)
ax1.grid(True, alpha=0.3)
ax1.axis('equal')

# Plot transformed correlated normals
ax2.scatter(X[:, 0], X[:, 1], alpha=0.5, s=20, color='red')

# Add confidence ellipse
eigenvalues, eigenvectors = np.linalg.eigh(Sigma)
angle = np.degrees(np.arctan2(eigenvectors[1, 1], eigenvectors[0, 1]))
ellipse = Ellipse(mu, 2*np.sqrt(eigenvalues[1]), 2*np.sqrt(eigenvalues[0]),
                  angle=angle, facecolor='none', edgecolor='red', linewidth=2)
ax2.add_patch(ellipse)

ax2.set_xlabel('X ')
ax2.set_ylabel('X ')
ax2.set_title('Correlated N( , Σ)')
ax2.grid(True, alpha=0.3)
ax2.axis('equal')

plt.tight_layout()
plt.show()

```

Covariance matrix Σ :

```
[[2.  1.2]
 [1.2 1. ]]
```

Cholesky factor L (lower triangular):

```
[[1.41421356 0.
   0.84852814 0.52915026]]
```

Verification: $L @ L.T =$

```
[[2.  1.2]
 [1.2 1. ]]
```

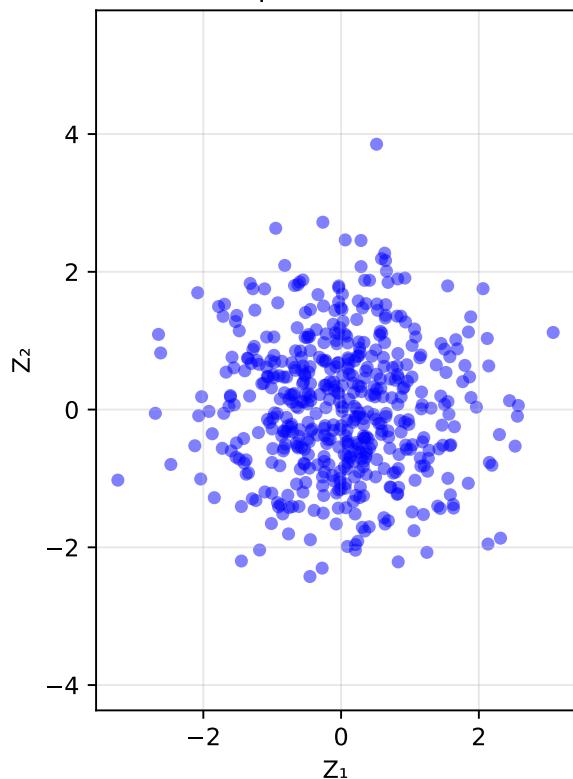
The Cholesky decomposition transforms:

- Independent standard normals $Z \sim N(0, I)$
- Into correlated normals $X = Z + LZ \sim N(\mu, \Sigma)$

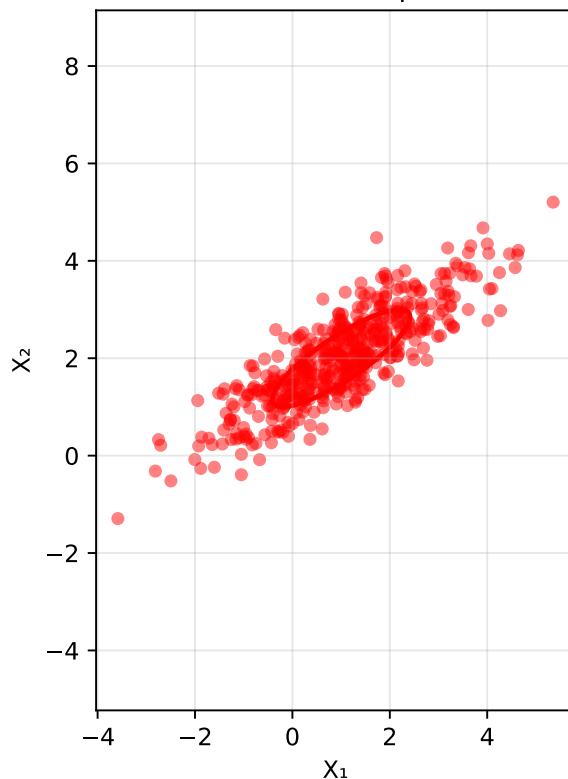
Sample covariance:

```
[[1.87031161 1.10325785
  1.10325785 0.92611656]]
```

Independent $N(0,1)$



Correlated $N(\mu, \Sigma)$



2.11 Chapter Summary and Connections

2.11.1 Key Concepts Review

We've explored the fundamental tools for summarizing and understanding random variables:

1. **Expectation as weighted average:** The fundamental summary of a distribution

2. **Linearity—the master property:** $\mathbb{E}(\sum a_i X_i) = \sum a_i \mathbb{E}(X_i)$ always works!
3. **Variance measures spread:** How far from the mean should we expect outcomes?
4. **Covariance measures linear relationships:** Do variables move together?
5. **Conditional expectation as best prediction:** What's our best guess given information?
6. **Matrix operations extend naturally:** Same concepts work for random vectors

2.11.2 Why These Concepts Matter

For Statistical Inference:

- Sample means estimate population expectations
- Variance quantifies uncertainty in estimates
- Covariance reveals relationships between variables
- Conditional expectation enables regression analysis

For Machine Learning:

- Loss functions are expectations over data distributions
- Gradient descent minimizes expected loss
- Feature correlations affect model performance
- Conditional expectations define optimal predictors

For Data Science Practice:

- Summary statistics (mean, variance) describe data concisely
- Correlation analysis reveals variable relationships
- Variance decomposition explains data structure
- Linear algebra connects to dimensionality reduction (PCA)

2.11.3 Common Pitfalls to Avoid

1. **Assuming $\mathbb{E}(XY) = \mathbb{E}(X)\mathbb{E}(Y)$ without independence**
 - This only works when X and Y are independent!
2. **Confusing the $\mathbb{V}(X - Y)$ formula**
 - Remember: $\mathbb{V}(X - Y) = \mathbb{V}(X) + \mathbb{V}(Y)$ when independent (plus, not minus!)
3. **Treating $\mathbb{E}(X|Y)$ as a number instead of a random variable**
 - $\mathbb{E}(X|Y = y)$ is a number, but $\mathbb{E}(X|Y)$ is a function of Y
4. **Assuming uncorrelated means independent**
 - Zero correlation does NOT imply independence (remember X and X^2)
5. **Forgetting existence conditions**
 - Not all distributions have finite expectation (Cauchy!)

2.11.4 Chapter Connections

This chapter builds on Chapter 1's probability foundations and provides essential tools for all statistical inference:

- **From Chapter 1:** We've formalized the expectation concept briefly introduced with random variables, showing how it connects to supervised learning through risk minimization and cross-entropy loss
- **Next - Chapter 3 (Convergence & Inference):** The sample mean and variance we studied will be shown to converge to population values (Law of Large Numbers) and have predictable distributions (Central Limit Theorem), justifying their use as estimators
- **Chapter 4 (Bootstrap):** We'll use the plug-in principle with empirical distributions to estimate variances and other functionals when theoretical calculations become intractable
- **Future Applications:** Conditional expectation forms the foundation for regression (Chapter 5+), while variance decomposition and covariance matrices are central to multivariate methods throughout the course

2.11.5 Self-Test Problems

Try these problems to test your understanding:

- Linearity puzzle:** In a class of 30 students, each has probability $1/365$ of having a birthday today. What's the expected number of birthdays today? (Ignore leap years) *Hint: Define indicator variables X_i for each student. What is $\mathbb{E}(X_i)$? Then use linearity.*
- Variance with correlation:** If $\mathbb{V}(X) = 4$, $\mathbb{V}(Y) = 9$, and $\rho(X, Y) = 0.5$, find $\mathbb{V}(2X - Y)$.
- Conditional expectation:** Toss a fair coin. If heads, draw $X \sim \mathcal{N}(0, 1)$. If tails, draw $X \sim \mathcal{N}(2, 1)$. Find $\mathbb{E}(X)$ and $\mathbb{V}(X)$.
- Matrix expectation:** If $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$ and $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, find the distribution of \mathbf{AX} .

2.11.6 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

2.11.7 Connections to Source Material

i Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Introduction and Motivation	Expanded material from the slides, contextualizing expectation for machine learning.
Foundations of Expectation	
Definition and Basic Properties	AoS §3.1
Existence of Expectation	AoS §3.1
Expectation of Functions	AoS §3.1 (Rule of the Lazy Statistician)
Properties of Expectation	
The Linearity Property	AoS §3.2 (Theorem 3.11)
Independence and Products	AoS §3.2 (Theorem 3.13)
Variance and Its Properties	
Measuring Spread	AoS §3.3 (Definition 3.14)
Properties of Variance	AoS §3.3 (Theorem 3.15)
Sample Mean and Variance	AoS §3.3 (Definitions and Theorem 3.17)
Covariance and Correlation	
Linear Relationships	AoS §3.3 (Definition 3.18)
Properties of Covariance and Correlation	AoS §3.3 (Theorem 3.19)
Variance of Sums (General Case)	AoS §3.3 (Theorem 3.20)
Expectation with Matrices	
Random Vectors & Covariance Matrix	AoS §3.4, §14.1
Linear Transformations	AoS §3.4 (Lemma 3.21)
Interpreting the Covariance Matrix	New material (connects to PCA).
Example: Multinomial Covariance	AoS §3.4
Conditional Expectation	
Expectation Given Information	AoS §3.5 (Definition 3.22)

Properties (Iterated Expectations)
Conditional Variance (Total Variance)
More About the Normal Distribution
Chapter Summary and Connections

AoS §3.5 (Theorem 3.24)
AoS §3.5 (Definition 3.26, Theorem 3.27)
New material, applying expectation concepts.
Some properties from AoS §2.10 are revisited.
New summary material.

2.11.8 Further Reading

- **Statistical perspective:** Casella & Berger, “Statistical Inference”
- **Machine learning view:** Bishop, “Pattern Recognition and Machine Learning” Chapters 1 and 2
- **Matrix cookbook:** Petersen & Pedersen, “The Matrix Cookbook” (for multivariate formulas) – [link](#)

Next time you compute a sample mean, remember: you’re estimating an expectation. When you minimize a loss function, you’re approximating an expected loss. The gap between what we can compute (sample statistics) and what we want to know (population parameters) drives all of statistical inference. Expectation is the bridge!

Chapter 3

Convergence and The Basics of Inference

3.1 Learning Objectives

After completing this chapter, you will be able to:

- Explain how probability inequalities provide bounds on uncertainty.
- Define concepts of probabilistic convergence and apply the Law of Large Numbers and Central Limit Theorem.
- Define the core vocabulary of statistical inference (models, parameters, estimators).
- Evaluate an estimator's quality using its standard error, bias, and variance.
- Explain the bias-variance tradeoff in the context of Mean Squared Error (MSE).

i Note

This chapter covers probability inequalities, convergence concepts, and the foundations of statistical inference. The material is adapted from Chapters 4, 5, and 6 of Wasserman (2013), supplemented with additional examples and perspectives relevant to data science applications.

3.2 Introduction and Motivation

3.2.1 Convergence Matters to Understand Machine Learning Algorithms

Deep learning models are trained with *stochastic* optimization algorithms. These algorithms produce a sequence of parameter estimates

$$\theta_1, \theta_2, \theta_3, \dots$$

as they iterate through the data. But here's the fundamental question: do these estimates eventually *converge* to a good solution, and how do we establish that?

The challenge is that these parameter estimates are random variables – they depend on random initialization, random mini-batch selection, and random data shuffling. We can't use the simple definition of convergence where $|x_n - x| < \epsilon$ for all large n , which you may remember from calculus. We need new mathematical tools.

This chapter develops the language of *probabilistic* convergence to understand and analyze such algorithms. We'll then use these tools to build the foundation of statistical inference – the science of drawing conclusions about populations from samples.

Consider a concrete example: training a neural network for image classification. At each iteration t :

1. Pick a random subset S of training images
2. Compute the gradient $g = \sum_{x_i \in S} \nabla_{\theta} L(\theta_t; x_i)$ of the loss¹
3. Compute next estimate of the model parameters, θ_{t+1} , using g and the current parameters²

The randomness in batch selection makes each θ_t a random variable. As mentioned before, ideally we would want $\theta_1, \theta_2, \dots$ to converge to a good solution. But what does it even mean to say the algorithm “converges”? This chapter provides the answer.

Convergence isn't just about optimization algorithms. It's central to all of statistics:

- When we compute a sample mean with increasing amount of data, does it converge to the population mean?
- More generally, when we estimate a model parameter, does our estimate improve with more data?
- When we approximate a distribution, does the approximation get better?

The remarkable answers to these questions – provided by the Law of Large Numbers and Central Limit Theorem – form the theoretical backbone of statistical inference and machine learning.

i Finnish Terminology Reference

For Finnish-speaking students, here's a reference table of key terms in this chapter:

English	Finnish	Context
Markov's inequality	Markovin epäyhtälö	Bounds probability of large values
Chebyshev's inequality	Tšebyšovin epäyhtälö	Uses variance to bound deviations
Convergence in probability	Stokastinen suppeneminen	Random variable settling to a value
Convergence in distribution	Jakaumasuppeneminen	Distribution shape converging
Law of Large Numbers	Suurten lukujen laki	Sample mean → population mean
Central Limit Theorem	Keskeinen raja-arvolause	Sums become normally distributed
Statistical model	Tilastollinen malli	Set of possible distributions
Parametric model	Parametrisen malli	Finite-dimensional parameter space
Nonparametric model	Epäparametrisen malli	Infinite-dimensional space
Nuisance parameter	Kiusaparametri	Parameter not of primary interest
Point estimation	Piste-estimointi	Single best guess of parameter
Estimator	Estimaattori	Function of data estimating parameter
Bias	Harha	Expected error of estimator
Unbiased	Harhaton	Zero expected error
Consistent	Tarkentuva	Converges to true value
Standard error	Keskivirhe	Standard deviation of estimator
Mean Squared Error (MSE)	Keskimääräinen neliövirhe	Average squared error
Sampling distribution	Otantajakauma	Distribution of the estimator

¹Remember that $\nabla_{\theta} f$ denotes the gradient of function $f(\theta; x)$ with respect to θ – its “vector derivative” with respect to θ in more than dimension.

²For example, via a simple gradient descent step:

$$\theta_{t+1} = \theta_t - \alpha_t g$$

where $\alpha_t > 0$ is the learning rate at step t .

3.3 Inequalities: Bounding the Unknown

3.3.1 Why We Need Inequalities

In probability and statistics, we often encounter quantities that are difficult or impossible to compute exactly. Inequalities provide *bounds* – upper or lower limits – that give us useful information even when exact calculations are intractable. They serve three critical purposes:

1. **Bounding quantities:** When we can't compute a probability exactly, an upper bound tells us it's "at most this large"
2. **Proving theorems:** The Law of Large Numbers and Central Limit Theorem rely on inequalities in their proofs
3. **Practical guarantees:** In machine learning, we use inequalities to create bounds on critical quantities such as generalization error³

Think of inequalities as providing universal statistical guarantees. They tell us that no matter how complicated the underlying distribution, certain bounds will always hold.

3.3.2 Markov's Inequality

Markov's Inequality: For a non-negative random variable X with finite expectation:

$$\mathbb{P}(X \geq t) \leq \frac{\mathbb{E}(X)}{t} \quad \text{for all } t > 0$$

This remarkably simple inequality says that – no matter what – the probability of a non-negative random variable exceeding a threshold t is bounded by its mean divided by t .

i Proof

Since $X \geq 0$:

$$\mathbb{E}(X) = \int_0^\infty xf(x) dx \tag{3.1}$$

$$= \int_0^t xf(x) dx + \int_t^\infty xf(x) dx \tag{3.2}$$

$$\geq \int_t^\infty xf(x) dx \tag{3.3}$$

$$\geq t \int_t^\infty f(x) dx \tag{3.4}$$

$$= t\mathbb{P}(X \geq t) \tag{3.5}$$

Rearranging gives the result.

Example: Exceeding a Multiple of the Mean

Let X be a non-negative random variable with mean $\mathbb{E}(X) = \mu$. What can we say about the probability that X exceeds k times its mean, for some $k > 1$?

³For example, in the framework known as [probably approximately correct \(PAC\) learning](#).

i Solution

Using Markov's inequality by setting $t = k\mu$:

$$\mathbb{P}(X \geq k\mu) \leq \frac{\mathbb{E}(X)}{k\mu} = \frac{\mu}{k\mu} = \frac{1}{k}$$

For example, the probability of a non-negative random variable exceeding twice its mean is at most $1/2$. The probability of it exceeding 10 times its mean is at most $1/10$. This universal bound is surprisingly useful.

Example: Exam Scores

If the average exam score is 50 points, what's the maximum probability that a randomly selected student scored 90 or more?

i Solution

Using Markov's inequality:

$$\mathbb{P}(X \geq 90) \leq \frac{50}{90} = \frac{5}{9} \approx 0.556$$

At most 55.6% of students can score 90 or more. This bound requires only knowing the average – no other information about the distribution!

Multiple Perspectives

Intuitive

Markov's inequality captures a fundamental truth: **averages constrain extremes**.

Imagine a village where the average wealth is €50,000. What fraction of villagers could be millionaires? If everyone were a millionaire, the average would be at least €1,000,000. Since the average is only €50,000, at most 5% can be millionaires:

$$\text{Fraction of millionaires} \leq \frac{\text{€}50,000}{\text{€}1,000,000} = 0.05$$

This reasoning works for any non-negative quantity: test scores, waiting times, file sizes, or loss values in machine learning. The average puts a hard limit on how often extreme values can occur.

Mathematical

Markov's inequality is the foundation for many other inequalities. Its power lies in its generality—it applies to any non-negative random variable with finite expectation.

The inequality is tight (best possible) for certain distributions. Consider:

$$X = \begin{cases} 0 & \text{with probability } 1 - \frac{\mu}{t} \\ t & \text{with probability } \frac{\mu}{t} \end{cases}$$

Then $\mathbb{E}(X) = \mu$ and $\mathbb{P}(X \geq t) = \frac{\mu}{t}$, achieving equality in Markov's inequality.

Computational

Let's visualize Markov's inequality by comparing the true tail probability with the bound for an exponential distribution.

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Set up the exponential distribution
beta = 2 # mean = 2
x = np.linspace(0, 10, 1000)
pdf = stats.expon.pdf(x, scale=beta)

# Compute true probabilities and Markov bounds
t_values = np.linspace(0.5, 10, 100)
true_probs = 1 - stats.expon.cdf(t_values, scale=beta)
markov_bounds = np.minimum(beta / t_values, 1) # E[X]/t, capped at 1

# Create the plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

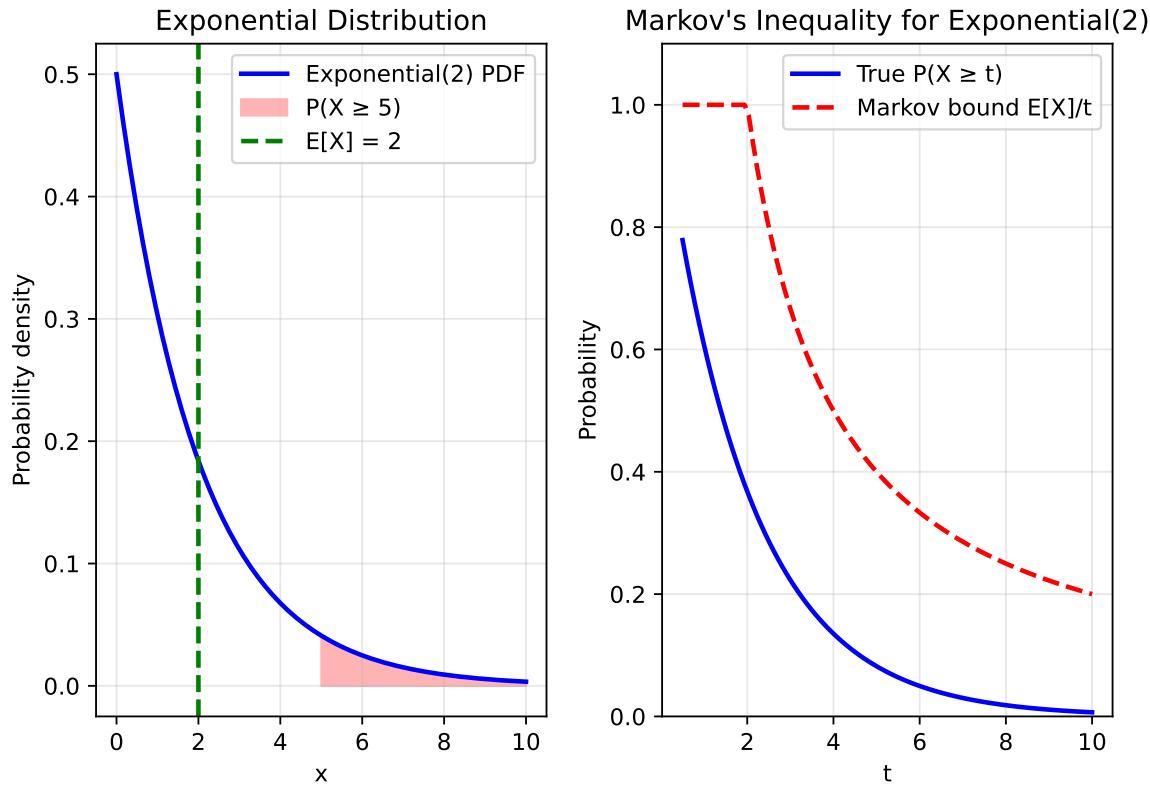
# Left plot: PDF with shaded tail
t_example = 5
ax1.plot(x, pdf, 'b-', linewidth=2, label='Exponential(2) PDF')
ax1.fill_between(x[x >= t_example], pdf[x >= t_example], alpha=0.3, color='red',
                  label=f'P(X >= {t_example})')
ax1.axvline(beta, color='green', linestyle='--', linewidth=2, label=f'E[X] = {beta}')
ax1.set_xlabel('x')
ax1.set_ylabel('Probability density')
ax1.set_title('Exponential Distribution')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Right plot: True probability vs Markov bound
ax2.plot(t_values, true_probs, 'b-', linewidth=2, label='True P(X >= t)')
ax2.plot(t_values, markov_bounds, 'r--', linewidth=2, label='Markov bound E[X]/t')
ax2.set_xlabel('t')
ax2.set_ylabel('Probability')
ax2.set_title('Markov\'s Inequality for Exponential(2)')
ax2.legend()
ax2.grid(True, alpha=0.3)
ax2.set_ylim(0, 1.1)

plt.tight_layout()
plt.show()

# Numerical comparison at specific points
print("Comparison of true probability vs Markov bound:")
for t in [1, 2, 4, 8]:
    true_p = 1 - stats.expon.cdf(t, scale=beta)
    markov_p = beta / t
    print(f't = {t}: True P(X >= {t}) = {true_p:.4f}, Markov bound = {markov_p:.4f}')

```



Comparison of true probability vs Markov bound:

$t = 1$: True $P(X \geq 1) = 0.6065$, Markov bound = 2.0000

$t = 2$: True $P(X \geq 2) = 0.3679$, Markov bound = 1.0000

$t = 4$: True $P(X \geq 4) = 0.1353$, Markov bound = 0.5000

$t = 8$: True $P(X \geq 8) = 0.0183$, Markov bound = 0.2500

Notice that the Markov bound is always valid but often loose. It becomes tighter as t increases relative to the mean.

3.3.3 Chebyshev's Inequality

While Markov's inequality uses only the mean, Chebyshev's inequality leverages the variance to provide a tighter bound on deviations from the mean.

Chebyshev's Inequality: Let X have mean μ and variance σ^2 . Then:

$$\mathbb{P}(|X - \mu| \geq t) \leq \frac{\sigma^2}{t^2} \quad \text{for all } t > 0$$

Proof

Apply Markov's inequality to the non-negative random variable $(X - \mu)^2$:

$$\mathbb{P}(|X - \mu| \geq t) = \mathbb{P}((X - \mu)^2 \geq t^2) \leq \frac{\mathbb{E}[(X - \mu)^2]}{t^2} = \frac{\sigma^2}{t^2}$$

Equivalently, in terms of standard deviations:

$$\mathbb{P}(|X - \mu| \geq k\sigma) \leq \frac{\sigma^2}{k^2\sigma^2} = \frac{1}{k^2} \quad \text{for all } k > 0$$

Example: Universal Two-Sigma Rule

For *any* distribution (not just normal!), Chebyshev's inequality tells us:

$$\mathbb{P}(|X - \mu| < k\sigma) \geq 1 - \frac{1}{k^2}.$$

Thus, for $k = 2$ and $k = 3$ we find:

- At least 75% of the data lies within 2 standard deviations of the mean: $\mathbb{P}(|X - \mu| < 2\sigma) \geq 1 - \frac{1}{4} = 0.75$
- At least 89% lies within 3 standard deviations: $\mathbb{P}(|X - \mu| < 3\sigma) \geq 1 - \frac{1}{9} \approx 0.889$

Compare this to the normal distribution where about 95% lies within 2σ and 99.7% within 3σ . Chebyshev's bounds are weaker but universal.

We show below the Chebyshev's bound compared to the actual tail probabilities of a few famous distributions (normal, uniform and exponential).

```
# Visualizing Chebyshev's inequality
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Set up comparison for different distributions
k_values = np.linspace(0.5, 4, 100)
chebyshev_bound = np.minimum(1 / k_values**2, 1)

# Compute actual probabilities for different distributions
normal_probs = 2 * (1 - stats.norm.cdf(k_values))
uniform_probs = np.maximum(0, 1 - k_values / np.sqrt(3)) # Uniform on [-sqrt(3), sqrt(3)]
exp_probs = []
for k in k_values:
    # For exponential with mean 1, mu=1, sigma=1
    exp_probs.append(stats.expon.cdf(1 - k, scale=1) + (1 - stats.expon.cdf(1 + k, scale=1)))

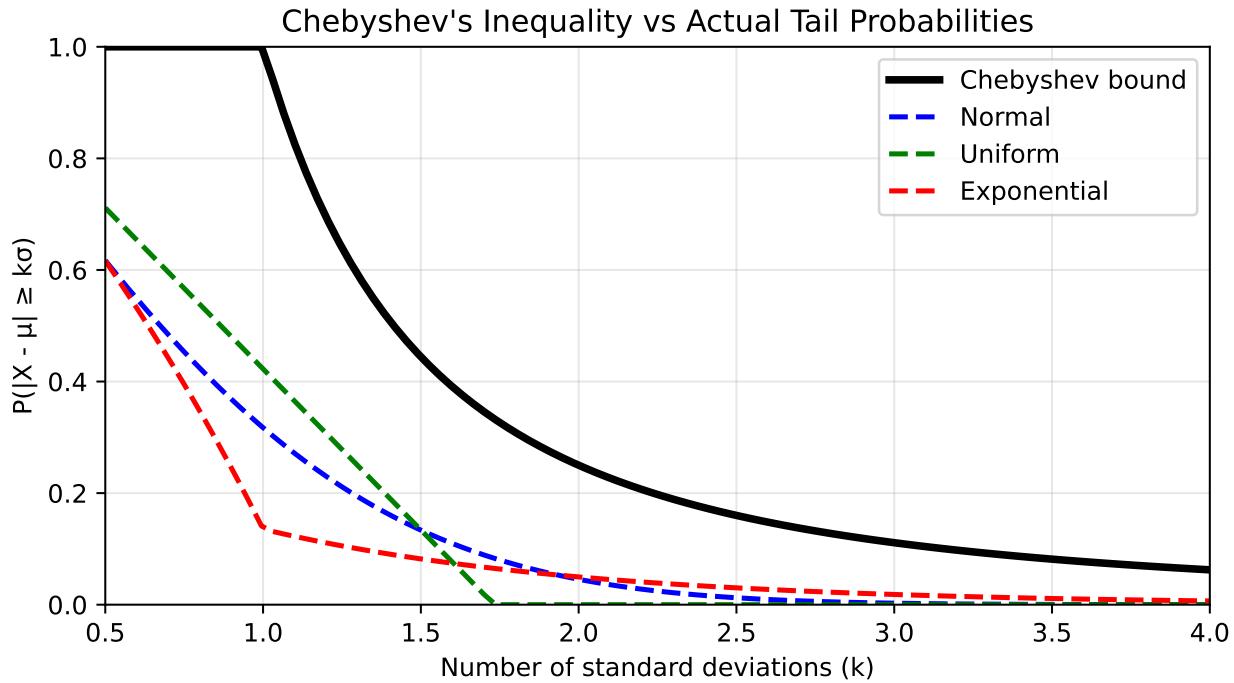
plt.figure(figsize=(7, 4))
plt.plot(k_values, chebyshev_bound, 'k-', linewidth=3, label='Chebyshev bound')
plt.plot(k_values, normal_probs, 'b--', linewidth=2, label='Normal')
plt.plot(k_values, uniform_probs, 'g--', linewidth=2, label='Uniform')
plt.plot(k_values, exp_probs, 'r--', linewidth=2, label='Exponential')
plt.xlabel('Number of standard deviations (k)')
plt.ylabel('P(|X - \mu| > k)')
plt.title('Chebyshev\\'s Inequality vs Actual Tail Probabilities')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xlim(0.5, 4)
plt.ylim(0, 1)
plt.tight_layout()
plt.show()

# Print specific values
print("Probability of being more than k standard deviations from the mean:")
print("k\tChebyshev\tNormal\tUniform\tExponential")
for k in [1, 2, 3]:
    cheby = 1/k**2
```

```

normal = 2 * (1 - stats.norm.cdf(k))
uniform = max(0, 1 - k/np.sqrt(3))
exp_val = stats.expon.cdf(1 - k, scale=1) + (1 - stats.expon.cdf(1 + k, scale=1))
print(f"\{k}\t{cheby:.4f}\t{normal:.4f}\t{uniform:.4f}\t{exp_val:.4f}")

```



Probability of being more than k standard deviations from the mean:

k	Chebyshev	Normal	Uniform	Exponential
1	1.0000	0.3173	0.4226	0.1353
2	0.2500	0.0455	0.0000	0.0498
3	0.1111	0.0027	0.0000	0.0183

i Advanced: Hoeffding's Inequality

While Chebyshev's inequality is universal, it can be quite loose. For bounded random variables, Hoeffding's inequality provides an exponentially decaying bound that's much sharper.

Let X_1, \dots, X_n be independent random variables with $X_i \in [a_i, b_i]$. Let $S_n = \sum_{i=1}^n X_i$. Then for any $t > 0$:

$$\mathbb{P}(S_n - \mathbb{E}[S_n] \geq t) \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

For the special case of n independent Bernoulli(p) random variables:

$$\mathbb{P}(|\bar{X}_n - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$$

where $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$.

The key insight is the exponential decay in n . This makes Hoeffding's inequality the foundation for many machine learning generalization bounds.

Example: Comparing Bounds

Consider estimating a probability p from $n = 100$ Bernoulli trials. How likely is our estimate to be off by more than $\epsilon = 0.2$?

i Solution

Chebyshev's bound: Since $\mathbb{V}(\bar{X}_n) = p(1-p)/n \leq 1/(4n)$:

$$\mathbb{P}(|\bar{X}_n - p| > 0.2) \leq \frac{1/(4 \times 100)}{0.2^2} = \frac{1/400}{0.04} = 0.0625$$

Hoeffding's bound:

$$\mathbb{P}(|\bar{X}_n - p| > 0.2) \leq 2e^{-2 \times 100 \times 0.2^2} = 2e^{-8} \approx 0.00067$$

Hoeffding's bound is nearly 100 times tighter! This exponential improvement is crucial for machine learning theory.

The proof of Hoeffding's inequality uses moment generating functions and is beyond the scope of this course, but the intuition is that bounded random variables have light tails, allowing for much stronger concentration.

3.4 Convergence of Random Variables

3.4.1 The Need for Probabilistic Convergence

In calculus, we say a sequence x_n converges to x if for every $\epsilon > 0$, we have $|x_n - x| < \epsilon$ for all sufficiently large n . But what about sequences of random variables?

There are multiple scenarios:

1. **Concentrating distribution:** Let $X_n \sim \mathcal{N}(0, 1/n)$. As n increases, the distribution concentrates more tightly around 0. Intuitively, X_n is “converging” to 0.
2. **Tracking outcomes:** The case above can be generalized where X_n does not converge to a constant (such as 0), but converges to the values taken by another random variable X .

The problem is that for any specific x , $\mathbb{P}(X_n = x) = 0$ for all n : continuous random variables never exactly equal any specific value.

There is then a completely different kind of convergence.

3. **Stable distribution:** Let $X_n \sim \mathcal{N}(0, 1)$ for all n . Each X_n has the same distribution, but they're different random variables. Is there a broader sense in which this sequence “converges”?

In sum, we need new definitions that capture different notions of what it means for random variables to converge.

3.4.2 Convergence in Probability

We consider the first two cases mentioned earlier: convergence of **outcomes** of a sequence of random variables to a constant or to the outcomes of another random variable, known as **convergence in probability**.

A sequence of random variables X_n **converges in probability** to a random variable X , written $X_n \xrightarrow{P} X$, if for every $\epsilon > 0$:

$$\mathbb{P}(|X_n - X| > \epsilon) \rightarrow 0 \text{ as } n \rightarrow \infty$$

When $X = c$ (a constant), we write $X_n \xrightarrow{P} c$.

This definition captures the idea that X_n becomes increasingly likely to be close to X as n grows. The probability of X_n being “far” from X (more than ϵ away) vanishes. In other words, the sequence of outcomes of the random variable X_n “track” the outcomes of X with ever-increasing accuracy as n increases.

Example: Convergence to Zero

Let $X_n \sim \mathcal{N}(0, 1/n)$. We’ll show that $X_n \xrightarrow{P} 0$.

For any $\epsilon > 0$, using Chebyshev’s inequality:

$$\mathbb{P}(|X_n - 0| > \epsilon) = \mathbb{P}(|X_n| > \epsilon) \leq \frac{\mathbb{V}(X_n)}{\epsilon^2} = \frac{1/n}{\epsilon^2} = \frac{1}{n\epsilon^2}$$

Since $\frac{1}{n\epsilon^2} \rightarrow 0$ as $n \rightarrow \infty$, we have $X_n \xrightarrow{P} 0$.

3.4.3 Convergence in Distribution

We now consider the other case, where it’s not the random variables to converge but their distribution.

A sequence of random variables X_n **converges in distribution** to a random variable X , written $X_n \rightsquigarrow X$, if:

$$\lim_{n \rightarrow \infty} F_n(t) = F(t)$$

at all points t where F is continuous. Here F_n is the CDF of X_n and F is the CDF of X .

This captures the idea that the *distribution* (or “shape”) of X_n becomes increasingly similar to that of X . We’re not saying the random variables themselves are close – just their overall probability distributions.

If X is a point mass at c , we denote $X_n \rightsquigarrow c$.

⚠ Warning

Key Distinction:

- Convergence in probability: The random variables themselves get close
- Convergence in distribution: Only the distributions get close

Let’s visualize this with the $X_n \sim \mathcal{N}(0, 1/n)$ example:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Set up the figure
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 4))

# Left plot: PDFs converging to a spike at 0
x = np.linspace(-3, 3, 1000)
n_values = [1, 2, 5, 10, 50]
colors = plt.cm.Blues(np.linspace(0.3, 0.9, len(n_values)))

for n, color in zip(n_values, colors):
    pdf = stats.norm.pdf(x, loc=0, scale=1/np.sqrt(n))
    ax1.plot(x, pdf, linewidth=2, color=color, label=f'n = {n}')
```

```

ax1.axvline(0, color='red', linestyle='--', alpha=0.7)
ax1.set_xlabel('x')
ax1.set_ylabel('Probability density')
ax1.set_title('PDFs of  $N(0, 1/n)$ ')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_ylim(0, 2.5)

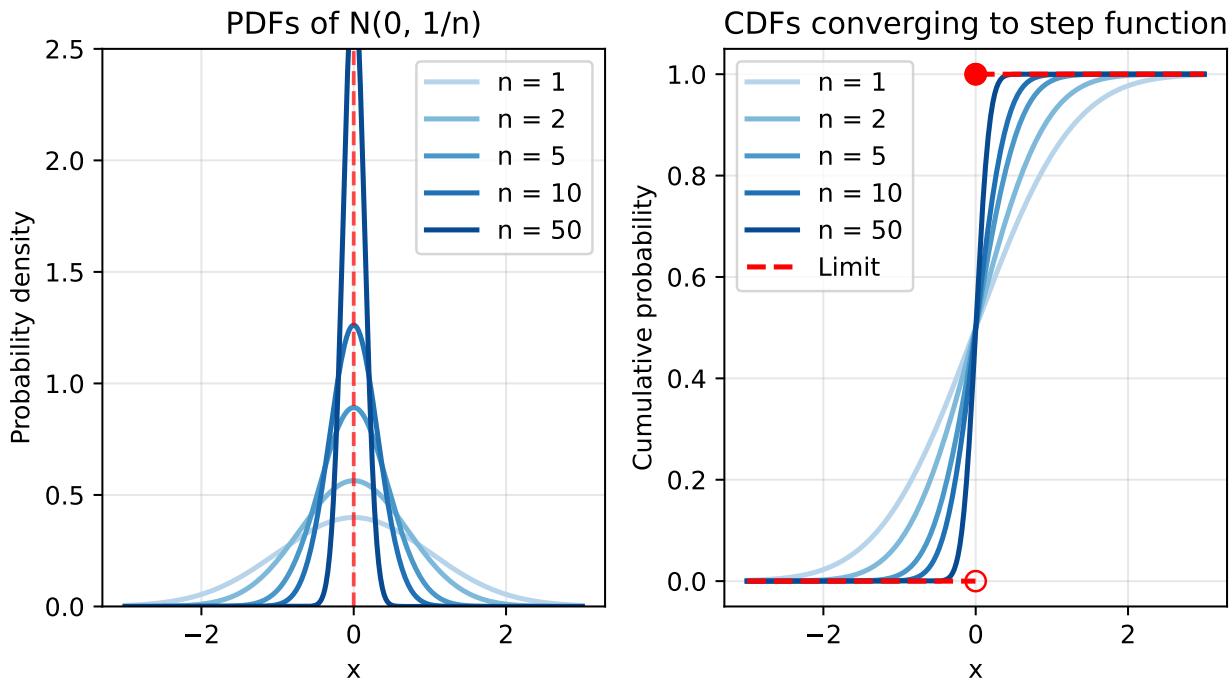
# Right plot: CDFs converging to step function
for n, color in zip(n_values, colors):
    cdf = stats.norm.cdf(x, loc=0, scale=1/np.sqrt(n))
    ax2.plot(x, cdf, linewidth=2, color=color, label=f'n = {n}')

# Plot limiting step function
ax2.plot(x[x < 0], np.zeros(sum(x < 0)), 'r--', linewidth=2, label='Limit')
ax2.plot(x[x >= 0], np.ones(sum(x >= 0)), 'r--', linewidth=2)
ax2.plot([0, 0], [0, 1], 'ro', markersize=8, fillstyle='none')
ax2.plot([0], [1], 'ro', markersize=8)

ax2.set_xlabel('x')
ax2.set_ylabel('Cumulative probability')
ax2.set_title('CDFs converging to step function')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```



As n increases:

- The PDF becomes more concentrated at 0 (spike)
- The CDF approaches a step function jumping from 0 to 1 at $x = 0$

- This is convergence in distribution to a point mass at 0

3.4.4 Comparing Modes of Convergence

Relationships Between Convergence Types

1. If $X_n \xrightarrow{P} X$ then $X_n \rightsquigarrow X$ (always)
2. If X is a point mass at c and $X_n \rightsquigarrow X$, then $X_n \xrightarrow{P} c$

Convergence in probability implies convergence in distribution, but the converse holds only for constants.

Multiple Perspectives

Intuitive

Convergence in Probability: The Perfect Weather Forecast

Let X be the actual temperature tomorrow and X_n be its forecast from an ever-improving machine learning model where n is the model version, as we make it bigger and feed it more data.

Convergence in probability ($X_n \xrightarrow{P} X$) means the forecast becomes more and more accurate as the model gets better and better. Eventually, the temperature prediction X_n gets so close to the actual temperature X that the forecast error, $|X_n - X|$, becomes negligible. The individual outcomes match.

Convergence in Distribution: The Perfect Climate Model

A climate model doesn't predict a specific day's temperature; it captures the statistical "character" of a season. Let X be the random variable for daily temperature, and X_n be a model's simulation of a typical day.

Convergence in distribution ($X_n \rightsquigarrow X$) means the model's simulated statistics (e.g., its histogram of temperatures) become identical to the real climate's statistics. The **patterns match**, but the individual **outcomes don't**.

The Takeaway:

- **Probability implies Distribution:** A perfect daily forecast naturally captures the climate's long-term statistics.
- **Distribution does NOT imply Probability:** A perfect climate model can't predict the actual temperature on next Friday.

Mathematical

We can construct a counterexample showing that convergence in distribution does NOT imply convergence in probability.

Counterexample: Let $X \sim \mathcal{N}(0, 1)$ and define $X_n = -X$ for all n . Then:

- Each $X_n \sim \mathcal{N}(0, 1)$, so trivially $X_n \rightsquigarrow X$
- But $|X_n - X| = |2X|$, so $\mathbb{P}(|X_n - X| > \epsilon) = \mathbb{P}(|X| > \epsilon/2) \not\rightarrow 0$
- Therefore X_n does NOT converge to X in probability!

The random variables have the same distribution but are perfectly anti-correlated.

Computational

Let's demonstrate both types of convergence and the counterexample computationally.

```

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
fig, axes = plt.subplots(2, 2, figsize=(7, 8))

# Case 1:  $X_n \sim N(0, 1/n) \rightarrow 0$  (both types of convergence)
ax = axes[0, 0]
n_values = [1, 10, 100, 1000]
n_samples = 5000

for i, n in enumerate(n_values):
    samples = np.random.normal(0, 1/np.sqrt(n), n_samples)
    ax.hist(samples, bins=50, alpha=0.6, density=True,
            label=f'n={n}', range=(-3, 3))

ax.axvline(0, color='red', linestyle='--', linewidth=2)
ax.set_xlabel('Value')
ax.set_ylabel('Density')
ax.set_title('Case 1:  $N(0, 1/n) \rightarrow 0$ \n(Converges in both senses)')
ax.legend()
ax.set_xlim(-3, 3)

# Case 1 continued: Show  $|X_n - 0|$  for different epsilon
ax = axes[0, 1]
epsilon = 0.5
prob_far = []
for n in range(1, 101):
    samples = np.random.normal(0, 1/np.sqrt(n), n_samples)
    prob_far.append(np.mean(np.abs(samples) > epsilon))

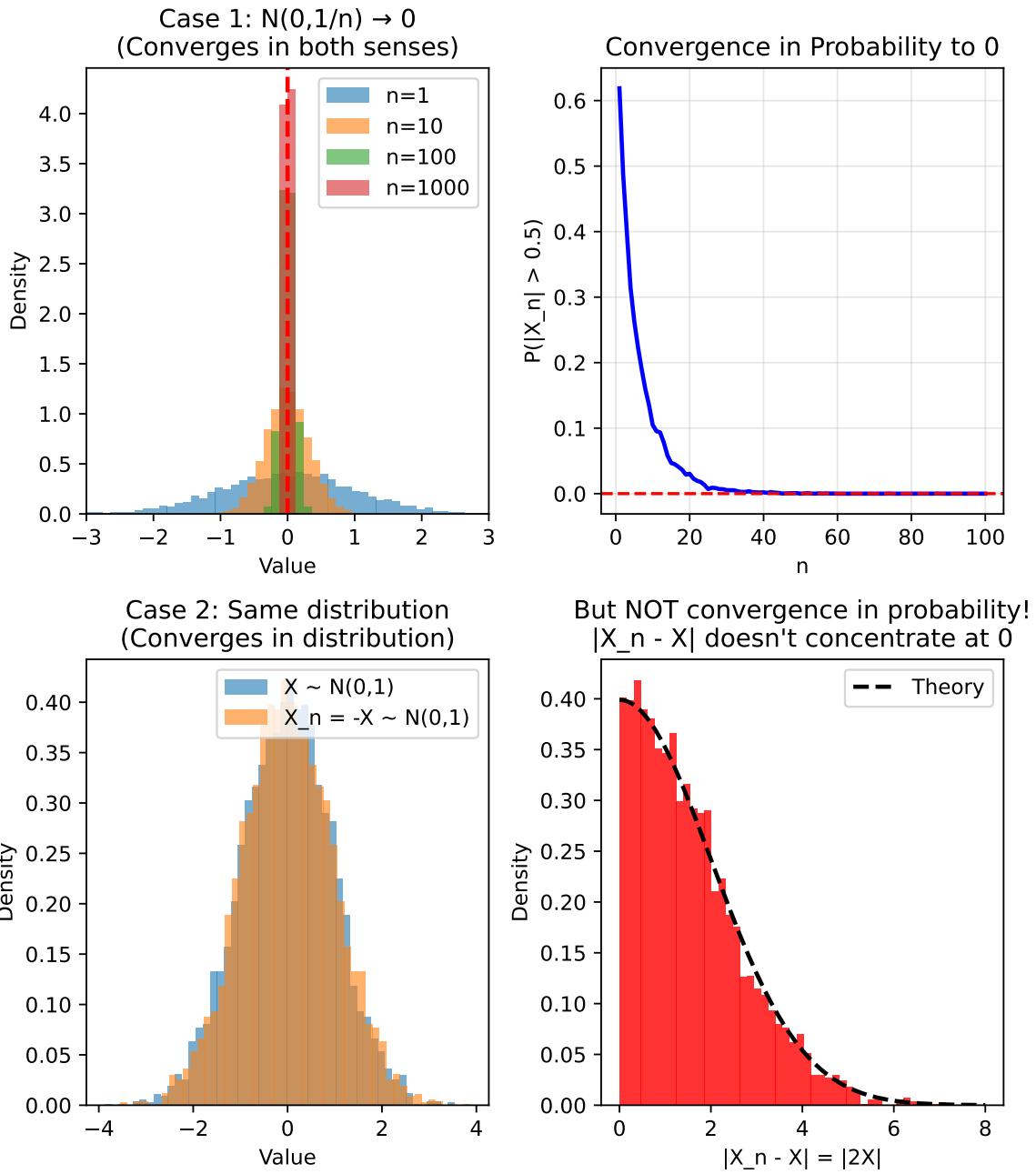
ax.plot(range(1, 101), prob_far, 'b-', linewidth=2)
ax.axhline(0, color='red', linestyle='--')
ax.set_xlabel('n')
ax.set_ylabel(f'P(| $X_n$ | > {epsilon})')
ax.set_title('Convergence in Probability to 0')
ax.grid(True, alpha=0.3)

# Case 2:  $X_n = -X$  counterexample
ax = axes[1, 0]
X = np.random.normal(0, 1, n_samples)
X_n = -X # X_n for all n

# Plot distributions (they're identical!)
ax.hist(X, bins=50, alpha=0.6, density=True, label=' $X \sim N(0, 1)$ ')
ax.hist(X_n, bins=50, alpha=0.6, density=True, label=' $X_n = -X \sim N(0, 1)$ ')
ax.set_xlabel('Value')
ax.set_ylabel('Density')
ax.set_title('Case 2: Same distribution\n(Converges in distribution)')
ax.legend()

# But  $|X_n - X| = |2X|$  doesn't converge to 0
ax = axes[1, 1]
diff = np.abs(X_n - X)
ax.hist(diff, bins=50, alpha=0.8, density=True, color='red')
ax.set_xlabel('| $X_n - X| = |2X|$ ')
ax.set_ylabel('Density')
ax.set_title('But NOT convergence in probability!\n| $X_n - X|$  doesn\'t concentrate at 0')

```

**Summary:**

- Case 1: $X_n \sim N(0, 1/n)$ converges to 0 in BOTH senses
- Case 2: $X_n = -X$ has same distribution as X but does NOT converge in probability
- Convergence in distribution is weaker than convergence in probability

3.4.5 Properties and Transformations

Understanding how convergence behaves under various operations is crucial for statistical theory. Here are the key properties:

Operations Under Convergence in Probability

If $X_n \xrightarrow{P} X$ and $Y_n \xrightarrow{P} Y$, then:

1. $X_n + Y_n \xrightarrow{P} X + Y$
2. $X_n Y_n \xrightarrow{P} XY$
3. $X_n/Y_n \xrightarrow{P} X/Y$ (if $\mathbb{P}(Y = 0) = 0$)

This shows that convergence in probability is well-behaved under standard operations of sum, product, and division not-by-zero.

Slutsky's Theorem

If $X_n \rightsquigarrow X$ and $Y_n \xrightarrow{P} c$ (constant), then:

1. $X_n + Y_n \rightsquigarrow X + c$
2. $X_n Y_n \rightsquigarrow cX$
3. $X_n/Y_n \rightsquigarrow X/c$ (if $c \neq 0$)

Slutsky's theorem tells us that convergence in distribution behaves nicely when paired with random variables that converge to a constant (this is not true in general!).

Continuous Mapping Theorem

If g is a continuous function:

1. $X_n \xrightarrow{P} X \implies g(X_n) \xrightarrow{P} g(X)$
2. $X_n \rightsquigarrow X \implies g(X_n) \rightsquigarrow g(X)$

Finally, we see that continuous mappings behave nicely for both types of convergence.

Warning

Important limitation: In general, if $X_n \rightsquigarrow X$ and $Y_n \rightsquigarrow Y$, we **cannot** conclude that $X_n + Y_n \rightsquigarrow X + Y$. Convergence in distribution does not preserve sums unless one component converges to a constant!

Counterexample: Let $X \sim \mathcal{N}(0, 1)$ and define $Y_n = -X$ for all n . Then $Y_n \sim \mathcal{N}(0, 1)$, so $Y_n \rightsquigarrow Y \sim \mathcal{N}(0, 1)$. But $X + Y_n = X - X = 0$, which does not converge in distribution to $X + Y \sim \mathcal{N}(0, 2)$.

Key takeaway

The rules for convergence are subtle. Generally speaking, convergence in probability behaves nicely under algebraic operations, but convergence in distribution requires more care. Always verify which type of convergence you have before applying these properties!

3.5 The Two Fundamental Theorems of Statistics

3.5.1 The Law of Large Numbers (LLN)

The Law of Large Numbers formalizes one of our most basic intuitions about probability: averages stabilize as we collect more data. When we flip a fair coin many times, the proportion of heads approaches $1/2$. When we measure heights of many people, the sample mean approaches the population mean. This isn't just intuition – it's a mathematical theorem.

Let X_1, X_2, \dots, X_n be independent and identically distributed (IID) random variables with $\mathbb{E}(X_i) = \mu$ and $\mathbb{V}(X_i) = \sigma^2 < \infty$. Define the **sample mean**:

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$$

Then $\bar{X}_n \xrightarrow{P} \mu$.

Interpretation: The sample mean converges in probability to the population mean. As we collect more data, our estimate gets arbitrarily close to the true value with high probability.

i Proof

We'll use Chebyshev's inequality. First, compute the mean and variance of \bar{X}_n :

$$\begin{aligned}\mathbb{E}(\bar{X}_n) &= \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n X_i\right) = \frac{1}{n} \sum_{i=1}^n \mathbb{E}(X_i) = \frac{1}{n} \cdot n\mu = \mu \\ \mathbb{V}(\bar{X}_n) &= \mathbb{V}\left(\frac{1}{n} \sum_{i=1}^n X_i\right) = \frac{1}{n^2} \sum_{i=1}^n \mathbb{V}(X_i) = \frac{1}{n^2} \cdot n\sigma^2 = \frac{\sigma^2}{n}\end{aligned}$$

(We used independence for the variance calculation.)

Now apply Chebyshev's inequality: for any $\epsilon > 0$,

$$\mathbb{P}(|\bar{X}_n - \mu| > \epsilon) \leq \frac{\mathbb{V}(\bar{X}_n)}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2} \rightarrow 0 \text{ as } n \rightarrow \infty$$

Therefore $\bar{X}_n \xrightarrow{P} \mu$.

Let's visualize the Law of Large Numbers in action by simulating repeated rolls of a standard six-sided die and computing the mean of all rolls until that point.

We show this in two plots: on a normal scale (top) and on a log-scale (bottom) for the number of rolls on the x axis. The bottom plot also zooms in on the y -axis around 3.5.

Note how the sample mean starts with high variability but converges to the true mean (3.5) as the number of rolls increases.

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Simulate die rolls
n_max = 10000
die_rolls = np.random.randint(1, 7, n_max)
cumulative_mean = np.cumsum(die_rolls) / np.arange(1, n_max + 1)
true_mean = 3.5

# Create the plot without shared x-axis
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 5))

# Top plot: Full scale with linear x-axis
ax1.plot(cumulative_mean, linewidth=2, color='blue', alpha=0.8)
ax1.axhline(y=true_mean, color='red', linestyle='--', linewidth=2,
            label=f'True mean = {true_mean}')
ax1.set_ylabel('Sample mean')
ax1.set_title('Law of Large Numbers: Sample Mean of Die Rolls')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_ylim(1, 6)
```

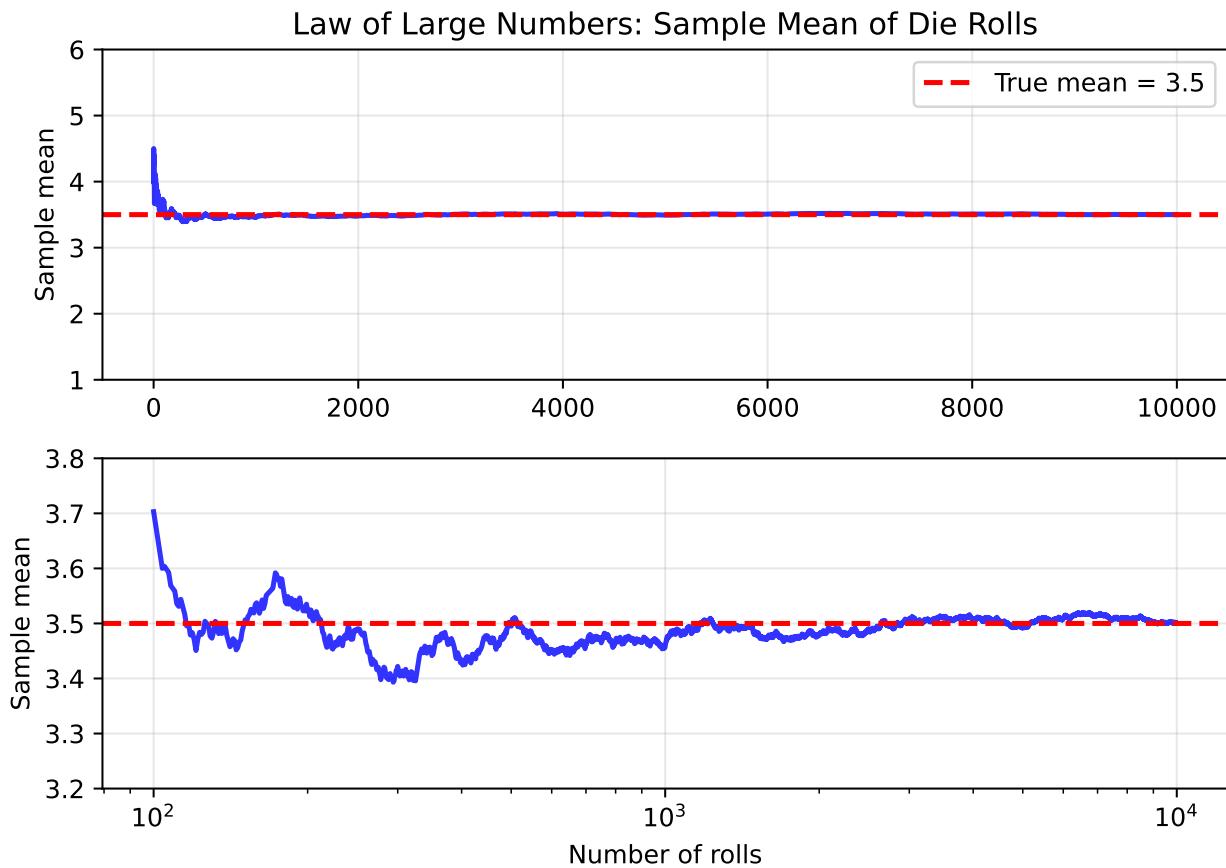
```

# Bottom plot: Zoomed in to show convergence with log scale
x_values = np.arange(100, n_max)
ax2.plot(x_values, cumulative_mean[100:], linewidth=2, color='blue', alpha=0.8)
ax2.axhline(y=true_mean, color='red', linestyle='--', linewidth=2)
ax2.set_xlabel('Number of rolls')
ax2.set_ylabel('Sample mean')
ax2.set_xscale('log')
ax2.grid(True, alpha=0.3)
ax2.set_ylim(3.2, 3.8)

plt.tight_layout()
plt.show()

# Show convergence at specific sample sizes
for n in [10, 100, 1000, 10000]:
    print(f"After {n:5d} rolls: sample mean = {cumulative_mean[n-1]:.4f}, "
          f"error = {abs(cumulative_mean[n-1] - true_mean):.4f}")

```



```

After    10 rolls: sample mean = 3.8000, error = 0.3000
After   100 rolls: sample mean = 3.6900, error = 0.1900
After  1000 rolls: sample mean = 3.4570, error = 0.0430
After 10000 rolls: sample mean = 3.4999, error = 0.0001

```

i Weak vs Strong Laws of Large Numbers

The theorem above is known as the “Weak” Law of Large Numbers because it guarantees convergence in probability. There exists a stronger version that guarantees **almost sure convergence**: $\mathbb{P}(\bar{X}_n \rightarrow \mu) = 1$. The “Strong” LLN says that with probability 1, the sample mean will eventually get arbitrarily close to μ and *stay* close, while the Weak LLN only guarantees that the probability of being far from μ goes to zero. The Weak LLN requires only finite variance, while the Strong LLN typically needs additional assumptions (like finite fourth moments) but delivers a more powerful conclusion. We present the Weak version as it has minimal assumptions and suffices for most statistical applications.

3.5.2 The Central Limit Theorem (CLT)

While the Law of Large Numbers tells us that sample means converge to the population mean, it doesn’t tell us about the *distribution* of the sample mean. The Central Limit Theorem fills this gap with a remarkable result: properly scaled sample means are approximately normal, regardless of the underlying distribution!

Let X_1, X_2, \dots, X_n be IID random variables with $\mathbb{E}(X_i) = \mu$ and $\mathbb{V}(X_i) = \sigma^2 < \infty$. Define:

$$Z_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} = \frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma}$$

Then $Z_n \rightsquigarrow Z$ where $Z \sim \mathcal{N}(0, 1)$.

Alternative notations (all mean the same thing):

- $\bar{X}_n \approx \mathcal{N}(\mu, \sigma^2/n)$ for large n
- $\sqrt{n}(\bar{X}_n - \mu) \rightsquigarrow \mathcal{N}(0, \sigma^2)$
- $(\bar{X}_n - \mu)/(\sigma/\sqrt{n}) \rightsquigarrow \mathcal{N}(0, 1)$

⚠ Warning

Critical Point: The CLT is about the distribution of the *sample mean*, not the data itself! The original data doesn’t become normal—only the sampling distribution of \bar{X}_n does.

i Online Interactive Demonstration of the CLT

An interactive visualization of the Central Limit Theorem is available in the HTML version of these notes. It allows you to select different population distributions and adjust the sample size n to see the convergence to normality in real-time.

Example: CLT in Practice

A factory produces bolts with mean length $\mu = 5$ cm and standard deviation $\sigma = 0.1$ cm. If we randomly sample 100 bolts, what’s the probability their average length exceeds 5.02 cm?

i Solution

By the CLT, $\bar{X}_{100} \approx \mathcal{N}(5, 0.1^2/100) = \mathcal{N}(5, 0.0001)$.

We want:

$$\mathbb{P}(\bar{X}_{100} > 5.02) = \mathbb{P}\left(\frac{\bar{X}_{100} - 5}{0.01} > \frac{5.02 - 5}{0.01}\right) = \mathbb{P}(Z > 2)$$

where $Z \sim \mathcal{N}(0, 1)$. From standard normal tables: $\mathbb{P}(Z > 2) \approx 0.0228$.

So there's about a 2.3% chance the sample mean exceeds 5.02 cm.

CLT with Unknown Variance: If we replace σ with the sample standard deviation

$$S_n = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2}$$

then we still have:

$$\frac{\sqrt{n}(\bar{X}_n - \mu)}{S_n} \rightsquigarrow \mathcal{N}(0, 1)$$

This version is crucial for practice since we rarely know the true variance!

Rejoinder: Understanding Algorithms

Remember the sequence of random variables $\theta_1, \theta_2, \dots$ from our stochastic optimization algorithm at the beginning of this chapter? We can now answer what kind of convergence we should expect:

Convergence in probability: We want $\theta_n \xrightarrow{P} \theta^*$ where θ^* is the true optimal solution. This means the probability of θ_n being far from the optimum vanishes as iterations increase.

The tools we've covered – probability inequalities (to bound deviations), convergence concepts (to formalize what “converges” means), and limit theorems (to understand averaging behavior) – are the foundation for analyzing when and why algorithms like stochastic gradient descent converge to good solutions. Modern machine learning theory relies heavily on these concepts to provide theoretical guarantees about algorithm performance!

3.6 The Language of Statistical Inference

3.6.1 From Probability to Inference

We've developed powerful tools: inequalities that bound uncertainty, convergence concepts that describe limiting behavior, and fundamental theorems that guarantee nice properties of averages. Now we flip the perspective.

Probability: Given a known distribution, what can we say about the data we'll observe?

Statistical Inference: Given observed data, what can we infer about the unknown distribution that generated it? More formally: given a sample $X_1, \dots, X_n \sim F$, how do we infer F ?

This process – often called “learning” in computer science – is at the core of both classical statistics and modern machine learning. Sometimes we want to infer the entire distribution F , but often we focus on specific features like its mean, variance, or other parameters.

Example: Modeling Uncertainty in Real Decisions

An online retailer tests a new ad campaign. Out of 1000 users who see the ad, 30 make a purchase (3% conversion rate). But this raises critical questions:

Immediate questions:

- What can we say about the true conversion rate? Is it exactly 3%?
- How likely is it that at least 25 out of the next 1000 users will convert?

Comparative questions:

- A competing ad had 290 conversions out of 10,000 users (2.9% rate). Which is better?
- How confident can we be that the 3% ad truly outperforms the 2.9% ad – could the difference just be due to random chance?

Long-term questions:

- What's the probability that the long-run conversion rate exceeds 2.5%?
- How many more users do we need to test to be 95% confident about the true rate?

These questions – about uncertainty, confidence, and decision-making with limited data – are at the heart of statistical inference.

3.6.2 Statistical Models

A **statistical model** \mathfrak{F} is a set of probability distributions (or densities or regression functions).

In the context of inference, we use models to represent our assumptions about which distributions could have generated our observed data. The model defines the “universe of possibilities” we’re considering – we then use data to identify which specific distribution within \mathfrak{F} is most plausible.

Models come in two main flavors, **parametric** and **nonparametric**.

3.6.2.1 Parametric Models

A **parametric model** is indexed by a finite number of parameters. We write it as:

$$\mathfrak{F} = \{f(x; \theta) : \theta \in \Theta\}$$

where:

- θ (**theta**) is the **parameter** (possibly vector-valued)⁴
- Θ (capital theta) is the **parameter space** (the set of all possible parameter values)
- $f(x; \theta)$ is the density or distribution function indexed by θ

Typically, the parameters θ are unknown quantities we want to estimate. If there are elements of the vector θ that we are not interested in, those are called **nuisance parameters**.

Example: Feature Performance as a Parametric Model

A product manager at a tech company launches a new “AI Recap” feature in their app. To determine if the feature is a success, they track the number of daily views over the first month. They hypothesize that the daily view count approximately follows a normal distribution.

The model for daily views is a parametric family \mathfrak{F} :

$$\mathfrak{F} = \{f(x; \theta) : \theta = (\mu, \sigma^2), \mu \in \mathbb{R}, \sigma^2 > 0\}$$

where the density function is:

$$f(x; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

⁴The symbol θ is almost universally reserved to represent generic “parameters” of a model in statistics and machine learning.

This is a 2-dimensional parametric model with:

- **Parameter vector:** $\theta = (\mu, \sigma^2)$
- **Parameter space:** $\Theta = \mathbb{R} \times (0, \infty)$

Nuisance parameters in action: The company has set a target: the feature will be considered successful and receive further development only if it can reliably generate more than 100,000 views per day.

- The **parameter of interest** is the average daily views, μ . The entire business decision hinges on testing the hypothesis that $\mu > 100,000$.
- The **nuisance parameter** is the variance, σ^2 . The day-to-day fluctuation in views is critical for assessing the statistical certainty of our estimate for μ , but it's not the primary metric for success. The product manager needs to account for this variability, but their core question is about the average performance, not the variability itself.

3.6.2.2 Nonparametric Models

Nonparametric Models cannot be parameterized by a finite number of parameters. These models make minimal assumptions about the distribution. For example:

$$\mathfrak{F}_{\text{ALL}} = \{\text{all continuous CDFs}\}$$

or with some constraints:

$$\mathfrak{F} = \{\text{all distributions with finite variance}\}$$

i How can we work with “all distributions”?

This seems impossibly broad! In practice, we don't explicitly enumerate all possible distributions. Instead, nonparametric methods **directly use the data** without assuming a specific functional form or parameter to be estimated. We will see multiple concrete examples of nonparametric techniques in the next chapter. So, in theory the model space is infinite-dimensional, but in practice nonparametric estimation procedures are still concrete and computable.

Example: Choosing a Model

Scenario 1: Heights of adult males in Finland.

- **Parametric choice:** $\mathfrak{F} = \{\mathcal{N}(\mu, \sigma^2) : \mu \in \mathbb{R}, \sigma > 0\}$
- **Justification:** Heights are often approximately normal due to many small genetic and environmental factors (CLT in action!)

Scenario 2: Time between website visits.

- **Parametric choice:** $\mathfrak{F} = \{\text{Exponential}(\lambda) : \lambda > 0\}$
- **Justification:** Exponential models “memoryless” waiting times

Scenario 3: Unknown distribution shape.

- **Nonparametric choice:** $\mathfrak{F} = \{\text{all distributions with finite variance}\}$
- **Justification:** Make minimal assumptions, let data speak

3.6.3 Point Estimation

Point estimation is the task of providing a single “best guess” for an unknown quantity based on data.

This quantity can be a single parameter, a full vector of parameters, even a full CDF or PDF, or prediction for a future value of some random variable.

A point estimate of θ is denoted by $\hat{\theta}$.

Given data X_1, \dots, X_n , a **point estimator** is a function:

$$\hat{\theta}_n = g(X_1, \dots, X_n)$$

The “hat” notation $\hat{\theta}$ can indicate both an estimator and the estimate.

⚠ Warning

Critical Distinction:

- **Parameter** θ : Fixed, unknown number we want to learn
 - **Estimator** $\hat{\theta}_n$: Random variable (before seeing data)
 - **Estimate** $\hat{\theta}_n$: Specific number (after seeing data) – notation can be overlapping
- For example, \bar{X}_n is an estimator; $\bar{x}_n = 3.7$ is an estimate.

The distribution of $\hat{\theta}_n$ is called the **sampling distribution**. The standard deviation of this distribution is the **standard error**:

$$\text{se}(\hat{\theta}_n) = \sqrt{\mathbb{V}(\hat{\theta}_n)}$$

When the standard error depends on unknown parameters, we use the **estimated standard error** $\widehat{\text{se}}$.

A particularly common standard error in statistics is the **standard error of the mean (SEM)**.

3.6.4 How to Evaluate Estimators

How do we judge if an estimator is “good”? Several criteria have emerged:

Bias: The systematic error of an estimator.

$$\text{bias}(\hat{\theta}_n) = \mathbb{E}(\hat{\theta}_n) - \theta$$

An estimator is **unbiased** if $\mathbb{E}(\hat{\theta}_n) = \theta$.

Consistency: An estimator is consistent if it converges to the true value.

$$\hat{\theta}_n \xrightarrow{P} \theta \text{ as } n \rightarrow \infty$$

Mean Squared Error (MSE): The average squared distance from the truth.

$$\text{MSE}(\hat{\theta}_n) = \mathbb{E}[(\hat{\theta}_n - \theta)^2]$$

Example: Evaluating Estimators for the Mean

Suppose $X_1, \dots, X_n \sim \mathcal{N}(\theta, \sigma^2)$ where θ is unknown. Consider three estimators:

1. **Constant estimator:** $\hat{\theta}_n^{(1)} = 3$
 - Bias: $\mathbb{E}(3) - \theta = 3 - \theta$ (biased unless $\theta = 3$)
 - Variance: $\mathbb{V}(3) = 0$
 - Consistent: No, always equals 3
 - MSE: $(3 - \theta)^2$
2. **First observation:** $\hat{\theta}_n^{(2)} = X_1$
 - Bias: $\mathbb{E}(X_1) - \theta = 0$ (unbiased!)
 - Variance: $\mathbb{V}(X_1) = \sigma^2$
 - Consistent: No, variance doesn't shrink
 - MSE: σ^2

3. **Sample mean:** $\hat{\theta}_n^{(3)} = \bar{X}_n$

- Bias: $\mathbb{E}(\bar{X}_n) - \theta = 0$ (unbiased!)
- Variance: $\mathbb{V}(\bar{X}_n) = \sigma^2/n$
- Consistent: Yes! (by LLN)
- MSE: $\sigma^2/n \rightarrow 0$

The sample mean is unbiased AND consistent—it improves with more data!

A classic example shows that unbiased isn't everything:

Sample variance: Two common estimators for population variance σ^2 :

1. **Unbiased version:** $S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$
 - $\mathbb{E}(S^2) = \sigma^2$ (unbiased by design)
2. **Maximum likelihood estimator:** $\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$
 - $\mathbb{E}(\hat{\sigma}^2) = \frac{n-1}{n} \sigma^2$ (biased!)

Which is better? It depends on the criterion!

3.6.5 The Bias-Variance Tradeoff

The MSE decomposes as:

$$\text{MSE} = \text{bias}^2(\hat{\theta}_n) + \mathbb{V}(\hat{\theta}_n)$$

i Proof

Let $\bar{\theta}_n = \mathbb{E}(\hat{\theta}_n)$. Then:

$$\mathbb{E}[(\hat{\theta}_n - \theta)^2] = \mathbb{E}[(\hat{\theta}_n - \bar{\theta}_n + \bar{\theta}_n - \theta)^2] \quad (3.6)$$

$$= \mathbb{E}[(\hat{\theta}_n - \bar{\theta}_n)^2] + 2(\bar{\theta}_n - \theta)\mathbb{E}[\hat{\theta}_n - \bar{\theta}_n] + (\bar{\theta}_n - \theta)^2 \quad (3.7)$$

$$= \mathbb{E}[(\hat{\theta}_n - \bar{\theta}_n)^2] + 2(\bar{\theta}_n - \theta) \cdot 0 + (\bar{\theta}_n - \theta)^2 \quad (3.8)$$

$$= \mathbb{V}(\hat{\theta}_n) + \text{bias}^2(\hat{\theta}_n) \quad (3.9)$$

where we used that $\mathbb{E}[\hat{\theta}_n - \bar{\theta}_n] = \mathbb{E}[\hat{\theta}_n] - \bar{\theta}_n = \bar{\theta}_n - \bar{\theta}_n = 0$.

This decomposition reveals a fundamental tradeoff in statistics.

Multiple Perspectives

Intuitive

Imagine you're an archer trying to hit a target. Your performance depends on two things:

Bias: How far your average shot is from the bullseye. A biased archer consistently aims too high or too far left.

Variance: How spread out your shots are. A high-variance archer is inconsistent—sometimes dead on, sometimes way off.

The best archer has low bias AND low variance. But here's the key insight: sometimes accepting a little bias can dramatically reduce variance, improving overall accuracy!

Think of it this way:

- A complex model (like memorizing training data) has low bias but high variance
- A simple model (like always predicting the average) has higher bias but low variance
- The sweet spot balances both

This tradeoff is why regularization works in machine learning – we accept a bit of bias to gain a lot in variance reduction.

Mathematical

The bias-variance decomposition gives us a precise way to understand prediction error:

$$\text{MSE}(\hat{\theta}_n) = \text{bias}^2(\hat{\theta}_n) + \mathbb{V}(\hat{\theta}_n)$$

This isn't just algebra – it reveals the two fundamental sources of error:

1. **Systematic error** (bias): Being consistently wrong
2. **Random error** (variance): Being inconsistently wrong

For prediction problems where $\hat{f}(x)$ estimates $f(x)$:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{bias}^2} + \underbrace{\mathbb{V}(\hat{f}(x))}_{\text{variance}}$$

The optimal predictor minimizes their sum. In machine learning:

- Increasing model complexity typically decreases bias but increases variance
- The art is finding the right complexity for your data

Computational

Let's visualize the bias-variance tradeoff by comparing estimators for population variance.

```

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Simulation parameters
true_variance = 1.0 # True  $\sigma^2 = 1$ 
n_values = np.arange(5, 101, 5)
n_simulations = 10000

# Storage for results
bias_unbiased = []
bias_mle = []
variance_unbiased = []
variance_mle = []
mse_unbiased = []
mse_mle = []

for n in n_values:
    # Generate many samples and compute both estimators
    unbiased_estimates = []
    mle_estimates = []

    for _ in range(n_simulations):
        # Generate sample from N(0, 1)
        sample = np.random.normal(0, 1, n)
        sample_mean = np.mean(sample)

        # Unbiased estimator (n-1 denominator)
        s_squared = np.sum((sample - sample_mean)**2) / (n - 1)
        unbiased_estimates.append(s_squared)

        # MLE (n denominator)
        sigma_hat_squared = np.sum((sample - sample_mean)**2) / n
        mle_estimates.append(sigma_hat_squared)

    # Calculate bias, variance, and MSE
    unbiased_estimates = np.array(unbiased_estimates)
    mle_estimates = np.array(mle_estimates)

    # Bias
    bias_unbiased.append(np.mean(unbiased_estimates) - true_variance)
    bias_mle.append(np.mean(mle_estimates) - true_variance)

    # Variance
    variance_unbiased.append(np.var(unbiased_estimates))
    variance_mle.append(np.var(mle_estimates))

    # MSE
    mse_unbiased.append(np.mean((unbiased_estimates - true_variance)**2))
    mse_mle.append(np.mean((mle_estimates - true_variance)**2))

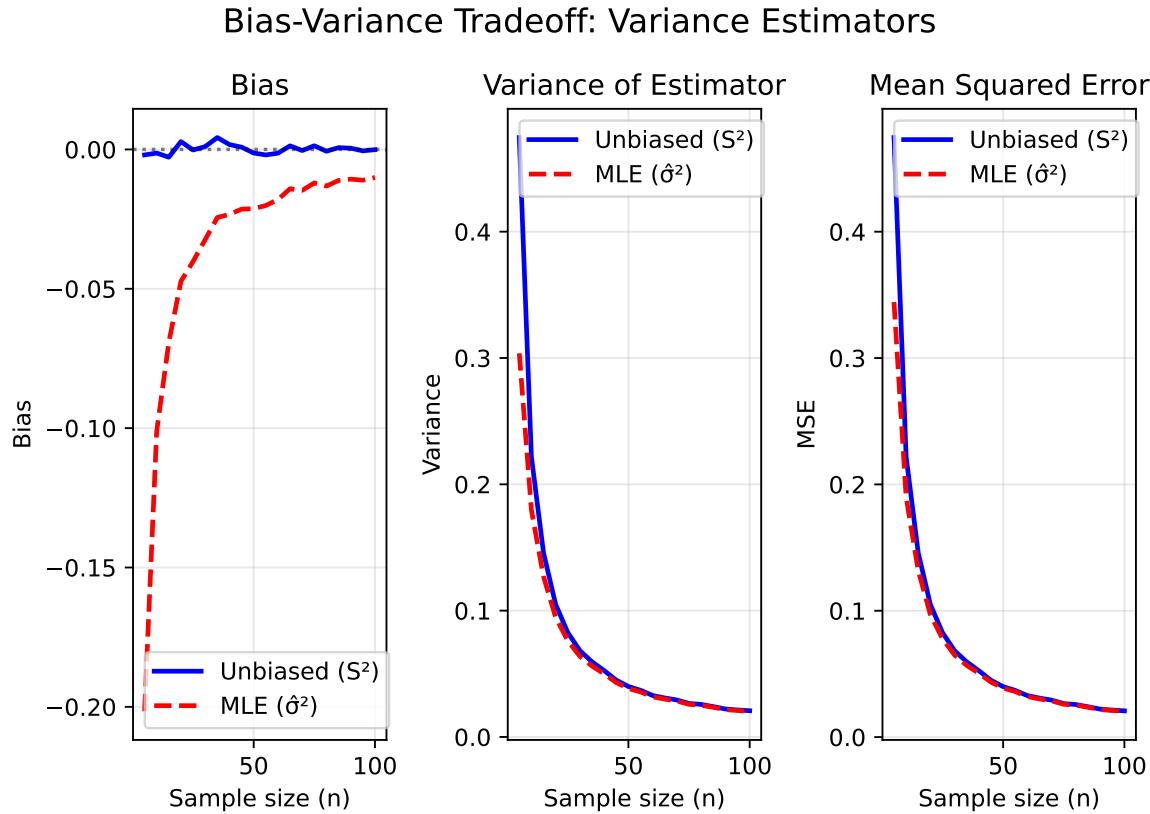
# Create plots
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(7, 5))

# Bias plot
ax1.plot(n_values, bias_unbiased, 'b-', linewidth=2, label='Unbiased ( $S^2$ )')
ax1.plot(n_values, bias_mle, 'r--', linewidth=2, label='MLE ( $\hat{\sigma}^2$ )')
ax1.axhline(y=0, color='black', linestyle=':', alpha=0.5)
ax1.set_title('Bias of Estimators')
ax1.set_xlabel('Sample Size (n)')
ax1.set_ylabel('Bias')

# Variance plot
ax2.plot(n_values, variance_unbiased, 'b-', linewidth=2, label='Unbiased ( $S^2$ )')
ax2.plot(n_values, variance_mle, 'r--', linewidth=2, label='MLE ( $\hat{\sigma}^2$ )')
ax2.set_title('Variance of Estimators')
ax2.set_xlabel('Sample Size (n)')
ax2.set_ylabel('Variance')

# MSE plot
ax3.plot(n_values, mse_unbiased, 'b-', linewidth=2, label='Unbiased ( $S^2$ )')
ax3.plot(n_values, mse_mle, 'r--', linewidth=2, label='MLE ( $\hat{\sigma}^2$ )')
ax3.set_title('MSE of Estimators')
ax3.set_xlabel('Sample Size (n)')
ax3.set_ylabel('MSE')

```



For $n=10$:

Unbiased (S^2): Bias = -0.0013, Variance = 0.2219, MSE = 0.2219
 MLE (\hat{S}^2): Bias = -0.1012, Variance = 0.1797, MSE = 0.1899

The MLE has lower MSE despite being biased!

Key insights from the visualization:

- The unbiased estimator S^2 has zero bias (blue solid line at 0)
- The MLE \hat{S}^2 has negative bias that shrinks as n grows
- The MLE has lower variance than the unbiased estimator
- **For finite samples, the biased MLE has lower MSE!**

This demonstrates a profound principle: the “best” estimator depends on your criterion. Unbiasedness is nice, but minimizing MSE often matters more in practice.

3.7 Chapter Summary and Connections

3.7.1 Key Concepts Review

We’ve built a complete framework for understanding randomness and inference:

Inequalities bound the unknown:

- **Markov:** $\mathbb{P}(X \geq t) \leq \mathbb{E}(X)/t$ — averages constrain extremes
- **Chebyshev:** $\mathbb{P}(|X - \mu| \geq k\sigma) \leq 1/k^2$ — universal bounds using variance

Convergence describes limiting behavior:

- **In probability:** $X_n \xrightarrow{P} X$ — the random variable itself settles down
- **In distribution:** $X_n \rightsquigarrow X$ — the shape of the distribution stabilizes

- **Key relationship:** Convergence in probability implies convergence in distribution

Fundamental theorems guarantee nice behavior:

- **Law of Large Numbers:** $\bar{X}_n \xrightarrow{P} \mu$ — sample means converge to population means
- **Central Limit Theorem:** $\sqrt{n}(\bar{X}_n - \mu)/\sigma \rightsquigarrow \mathcal{N}(0, 1)$ — sample means are approximately normal

Statistical inference flips the perspective:

- **Models:** Parametric (finite-dimensional) vs nonparametric (infinite-dimensional)
- **Estimators:** Functions of data that guess parameters
- **Standard error:** Standard deviation of an estimator's sampling distribution
- **Evaluation criteria:** Bias, variance, consistency, MSE
- **Bias-variance tradeoff:** $MSE = \text{bias}^2 + \text{variance}$

3.7.2 Why These Concepts Matter

For Statistical Inference:

- LLN justifies using sample statistics to estimate population parameters
- CLT enables confidence intervals and hypothesis tests (to be seen in the next chapters)
- Bias-variance tradeoff guides choice of estimators
- Consistency ensures our methods improve with more data

For Machine Learning:

- Convergence concepts analyze iterative algorithms such as stochastic optimization
- Bias-variance tradeoff explains overfitting vs underfitting
- CLT justifies bootstrap and cross-validation, as we will see in the next chapters

For Data Science Practice:

- Understanding variability in estimates prevents overconfidence
- Recognizing when CLT applies (and when it doesn't)
- Choosing between simple and complex models
- Interpreting A/B test results correctly

3.7.3 Common Pitfalls to Avoid

1. Confusing convergence types:

- “My algorithm converged” – in what sense?
- Convergence in distribution does *not* imply convergence in probability!

2. Misapplying the CLT:

- CLT is about *sample means*, not individual observations
- Need large enough n (depends on skewness)
- Doesn't work without finite variance (Cauchy!)

3. Overvaluing unbiasedness:

- Unbiased doesn't mean good (e.g., using just X_1)
- Biased can be better in statistics (regularization, priors)

4. Ignoring assumptions:

- Independence matters for variance calculations
- Finite variance required for CLT

5. Misinterpreting bounds:

- Markov/Chebyshev give worst-case bounds
- Often very loose in practice
- Tighter bounds exist for specific distributions

3.7.4 Chapter Connections

This chapter connects fundamental probability theory to practical statistical inference:

- **From Previous Chapters:** We've applied Chapter 1's probability framework and Chapter 2's expectation/variance concepts to prove convergence theorems (LLN, CLT) that explain why sample statistics work as estimators
- **Next - Chapter 4 (Bootstrap):** While this chapter gave us theoretical tools for inference (CLT-based confidence intervals), the bootstrap will provide a computational approach that works even when theoretical distributions are intractable
- **Statistical Modeling (Chapters 5+):** The bias-variance tradeoff introduced here becomes central to model selection, while MSE serves as our primary tool for comparing estimators in regression and machine learning
- **Throughout the Course:** The convergence concepts (especially CLT) and inference framework established here underpin virtually every statistical method—from hypothesis testing to Bayesian inference

3.7.5 Self-Test Problems

1. **Applying Chebyshev:** A website's daily visitors have mean 10,000 and standard deviation 2,000. Without assuming any distribution, what can you say about the probability of getting fewer than 4,000 or more than 16,000 visitors?
2. **CLT Application:** A casino's slot machine pays out €1 with probability 0.4 and €0 otherwise. If someone plays 400 times, approximate the probability their total winnings exceed €170.
3. **Comparing Estimators:** Given $X_1, \dots, X_n \sim \text{Uniform}(0, \theta)$, consider two estimators:

- $\hat{\theta}_1 = 2\bar{X}_n$
- $\hat{\theta}_2 = \frac{n+1}{n} \max\{X_1, \dots, X_n\}$

Calculate bias and variance for each. Which has lower MSE?

4. **Convergence Concepts:** Let X_n have PMF:

$$P(X_n = 0) = 1 - 1/n, \quad P(X_n = n) = 1/n$$

- Does $X_n \xrightarrow{P} 0$?
- Does $X_n \rightsquigarrow 0$?
- Does $\mathbb{E}(X_n) \rightarrow 0$?

3.7.6 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

3.7.7 Connections to Source Material

i Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Introduction and Motivation	Expanded material from the slides, contextualizing convergence for machine learning.
Inequalities: Bounding the Unknown	
Markov's Inequality	AoS §4.1 (Theorem 4.1)
Chebyshev's Inequality	AoS §4.1 (Theorem 4.2)
Hoeffding's Inequality	AoS §4.1 (Theorems 4.4 & 4.5)
Convergence of Random Variables	
The Need for Probabilistic Convergence	AoS §5.1
Convergence in Probability	AoS §5.2 (Definition 5.1)
Convergence in Distribution	AoS §5.2 (Definition 5.1)
Comparing Modes of Convergence	AoS §5.2 (Theorem 5.4)
Properties and Transformations	AoS §5.2 (Theorem 5.5, including Slutsky's Theorem)
The Two Fundamental Theorems of Statistics	
The Law of Large Numbers (LLN)	AoS §5.3 (The Weak Law of Large Numbers, Theorem 5.6)
The Central Limit Theorem (CLT)	AoS §5.4 (Theorems 5.8 & 5.10)
The Language of Statistical Inference	
From Probability to Inference	AoS §6.1
Statistical Models	AoS §6.2
Point Estimation	AoS §6.3.1
How to Evaluate Estimators	AoS §6.3.1 (covers bias, consistency, MSE)
The Bias-Variance Tradeoff	AoS §6.3.1 (MSE decomposition, Theorem 6.9)
Chapter Summary and Connections	New summary material.

3.7.8 Further Reading

- **Statistical inference:** Casella & Berger, “Statistical Inference”
- **Machine learning perspective:** Shalev-Shwartz & Ben-David, “Understanding Machine Learning: From Theory to Algorithms”

Remember: Convergence and inference concepts are the bedrock of statistics. The Law of Large Numbers tells us why sampling works. The Central Limit Theorem tells us how to quantify uncertainty. The bias-variance tradeoff tells us how to choose good estimators. Master these ideas – they’re the key to everything that follows!

Chapter 4

Nonparametric Estimation and The Bootstrap

4.1 Learning Objectives

After completing this chapter, you will be able to:

- Explain the definition and frequentist interpretation of a confidence interval
- Apply the plug-in principle with the Empirical Distribution Function (EDF) to estimate statistical functionals
- Use the bootstrap to simulate the sampling distribution of a statistic and estimate its standard error
- Construct and compare the three main types of bootstrap confidence intervals: Normal, percentile, and pivotal
- Identify the assumptions and limitations of the bootstrap, recognizing common situations where it can fail

i Note

This chapter covers nonparametric estimation methods and the bootstrap. The material is adapted from Chapters 7 and 8 of Wasserman (2013), with reference to Chapter 6 for confidence intervals. Additional examples and computational perspectives have been added for data science applications.

4.2 Introduction and Motivation

4.2.1 Beyond Point Estimates: Quantifying Uncertainty

Imagine you're a healthcare administrator planning for hospital capacity during an epidemic. Your data scientists provide a point estimate: "We expect 500 patients next week." But is this estimate reliable enough to base critical decisions on? What if the true number could reasonably be anywhere from 300 to 700? A single point estimate, without quantifying its uncertainty, is often useless for decision-making. You need a plausible *range* – this is called a **confidence interval**.

In Chapter 3, we learned how to create point estimates – single "best guesses" for unknown quantities. We saw that the sample mean \bar{X}_n estimates the population mean μ , and we even derived its standard error. But what about more complex statistics? How do we find the standard error of the median? The correlation coefficient? The 90th percentile?

Traditional statistical theory provides formulas for simple cases, but quickly becomes intractable for complex statistics. This chapter introduces two powerful ideas that revolutionized modern statistics:

1. **The Plug-In Principle:** A simple, intuitive method for estimating almost any quantity by “plugging in” the empirical distribution for the true distribution.
2. **The Bootstrap:** A computational method for estimating the standard error and confidence interval of virtually any statistic, even when no formula exists.

These methods exemplify the computational approach to statistics that has become dominant in the era of cheap computing power. Instead of deriving complex mathematical formulas, we let the computer simulate what would happen if we could repeat our experiment many times.

Finnish Terminology Reference

For Finnish-speaking students, here’s a reference table of key terms in this chapter:

English	Finnish	Context
Confidence interval	Luottamusväli	Range that contains parameter with specified probability
Coverage	Peitto	Probability that interval contains true parameter
Empirical distribution function	Empiirinen kertymäfunktio	Data-based estimate of CDF
Statistical functional	Tilastollinen funktionaali	Function of the distribution
Plug-in estimator	Pisto-estimaattori	Estimate using empirical distribution
Bootstrap	Uusio-otanta	Resampling method for uncertainty
Resampling	Uudelleenotanta	Drawing samples from data
Bootstrap sample	Bootstrap-otos	Sample drawn with replacement
Percentile interval	Prosenttipiste-luottamusväli	CI using bootstrap quantiles
Pivotal interval	Pivotaalinen luottamusväli	CI using pivot quantity
Standard error	Keskivirhe	Standard deviation of estimator
Monte Carlo error	Monte Carlo -virhe	Error from finite simulations

4.3 Confidence Sets: The Foundation

Before diving into nonparametric estimation and the bootstrap, we need to establish the formal framework for confidence intervals, a staple of classical statistics.

A $1 - \alpha$ **confidence interval** for a parameter θ is an interval $C_n = (a, b)$ where $a = a(X_1, \dots, X_n)$ and $b = b(X_1, \dots, X_n)$ are functions of the data such that

$$\mathbb{P}_\theta(\theta \in C_n) \geq 1 - \alpha, \quad \text{for all } \theta \in \Theta.$$

In other words, C_n encloses θ with probability $1 - \alpha$. This probability is called the **coverage** of the interval. A common choice is $\alpha = 0.05$ which yields 95% confidence intervals.

4.3.1 Interpretation of Confidence Sets

Confidence intervals are not probability statements about θ . The parameter θ is fixed; it’s the interval C_n that varies with different samples.

The correct interpretation is: if you repeatedly collect data and construct confidence intervals using the same procedure, $(1 - \alpha) \times 100\%$ of those intervals will contain the true parameter. This is the *frequentist* guarantee.

Bayesian credible intervals provide a different type of statement – they give a probability that θ lies in the interval given your observed data. However, Bayesian intervals are not guaranteed to achieve frequentist coverage (containing the true parameter $(1 - \alpha)$ proportion of the time across repeated sampling).

⚠ Critical Point: What is Random?

Remember that θ is **fixed** and C_n is **random**. The parameter doesn't change; the interval does. Each time we collect new data, we get a different interval. The guarantee is that $(1 - \alpha) \times 100\%$ of these intervals will contain the true parameter.

i Advanced: When Coverage Fails

The frequentist guarantee is an average over all possible datasets. For any specific confidence interval procedure, there may exist parameter values where the actual coverage is less than the nominal $(1 - \alpha)$ level.

Additionally, for some “unlucky” datasets (the α proportion), the computed interval may be far from the true parameter. This is an inherent limitation of the frequentist approach – it provides no guarantees for individual intervals, only for the long-run performance of the procedure.

This suggests different approaches may be preferred in different contexts:

- **Frequentist methods:** Well-suited for repeated analyses where long-run guarantees matter
- **Bayesian methods:** May be preferred when prior information is available and you need probabilistic statements about parameters given your specific data

Both approaches are valuable tools for quantifying uncertainty.

4.3.2 Normal-Based Confidence Intervals

When an estimator $\hat{\theta}_n$ is approximately normally distributed, we can form confidence intervals using the normal approximation. This is often the case for large samples due to the Central Limit Theorem.

Suppose $\hat{\theta}_n \approx \mathcal{N}(\theta, \widehat{\text{se}}^2)$, where $\widehat{\text{se}}$ is the (estimated) standard error of the estimator. Then we can standardize to get:

$$\frac{\hat{\theta}_n - \theta}{\widehat{\text{se}}} \approx \mathcal{N}(0, 1)$$

Let $z_{\alpha/2} = \Phi^{-1}(1 - \alpha/2)$ be the upper $\alpha/2$ quantile of the standard normal distribution, where Φ is the standard normal CDF. This means:

- $\mathbb{P}(Z > z_{\alpha/2}) = \alpha/2$ for $Z \sim \mathcal{N}(0, 1)$
- $\mathbb{P}(-z_{\alpha/2} < Z < z_{\alpha/2}) = 1 - \alpha$

Therefore:

$$\mathbb{P}\left(-z_{\alpha/2} < \frac{\hat{\theta}_n - \theta}{\widehat{\text{se}}} < z_{\alpha/2}\right) \approx 1 - \alpha$$

Rearranging to isolate θ :

$$\mathbb{P}\left(\hat{\theta}_n - z_{\alpha/2}\widehat{\text{se}} < \theta < \hat{\theta}_n + z_{\alpha/2}\widehat{\text{se}}\right) \approx 1 - \alpha$$

This gives us the approximate $(1 - \alpha)$ confidence interval:

$$C_n = (\hat{\theta}_n - z_{\alpha/2}\widehat{\text{se}}, \hat{\theta}_n + z_{\alpha/2}\widehat{\text{se}})$$

For the common case of 95% confidence intervals ($\alpha = 0.05$):

- $z_{0.025} = \Phi^{-1}(0.975) \approx 1.96 \approx 2$
- This leads to the familiar rule of thumb: $\hat{\theta}_n \pm 2\widehat{\text{se}}$

Example: Confidence Interval for the Mean

For the sample mean \bar{X}_n with known population variance σ^2 :

- Estimator: $\hat{\theta}_n = \bar{X}_n$
- Standard error: $\text{se} = \sigma/\sqrt{n}$
- 95% CI: $\bar{X}_n \pm 1.96 \cdot \sigma/\sqrt{n}$

If σ is unknown, we substitute the sample standard deviation s to get:

- 95% CI: $\bar{X}_n \pm 1.96 \cdot s/\sqrt{n}$

For small samples, we would use the [t-distribution](#) instead of the normal distribution.

4.4 The Plug-In Principle: A General Method for Estimation

This lecture focuses on **nonparametric estimation**, and the plug-in principle is one key instrument of nonparametric statistics.

4.4.1 Estimating the Entire Distribution: The EDF

The plug-in principle is a simple idea that provides a unified framework for nonparametric estimation that works for virtually any statistical problem.

The Core Idea: When we need to estimate some property of an unknown distribution F , we simply:

1. Estimate the distribution itself using our data
2. Calculate the property using our estimated distribution

The most fundamental nonparametric estimate is of the CDF itself. Let $X_1, \dots, X_n \sim F$ be an i.i.d. sample where F is a distribution function on the real line. Since we don't know the true CDF F , we estimate it with the **Empirical Distribution Function (EDF)**.

The **Empirical Distribution Function (EDF)** \hat{F}_n is the CDF that puts probability mass $1/n$ on each data point. Formally:

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^n I(X_i \leq x)$$

$$\text{where } I(X_i \leq x) = \begin{cases} 1 & \text{if } X_i \leq x \\ 0 & \text{if } X_i > x \end{cases}$$

Intuitively, $\hat{F}_n(x)$ is simply the proportion of data points less than or equal to x . It's the most natural estimate of $F(x) = \mathbb{P}(X \leq x)$.

Properties of the EDF:

For any fixed point x :

- **Unbiased:** $\mathbb{E}(\hat{F}_n(x)) = F(x)$
- **Variance:** $\mathbb{V}(\hat{F}_n(x)) = \frac{F(x)(1-F(x))}{n}$
- **MSE:** $\text{MSE}(\hat{F}_n(x)) = \mathbb{V}(\hat{F}_n(x)) = \frac{F(x)(1-F(x))}{n} \rightarrow 0$ as $n \rightarrow \infty$
- **Consistent:** $\hat{F}_n(x) \xrightarrow{P} F(x)$ as $n \rightarrow \infty$

But the EDF is even better than these pointwise properties suggest:

The EDF converges to the true CDF *uniformly*:

$$\sup_x |\hat{F}_n(x) - F(x)| \xrightarrow{P} 0$$

This is a remarkably strong guarantee – the EDF approximates the true CDF well *everywhere*, not just at individual points.

Let's visualize the EDF with an example dataset:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Simulate nerve data (waiting times between pulses)
# Using exponential distribution as a model for waiting times
np.random.seed(42)
n = 100
nerve_data = np.random.exponential(scale=0.5, size=n)

# Create the EDF plot
fig, ax = plt.subplots(figsize=(7, 5))

# Sort data for plotting
sorted_data = np.sort(nerve_data)

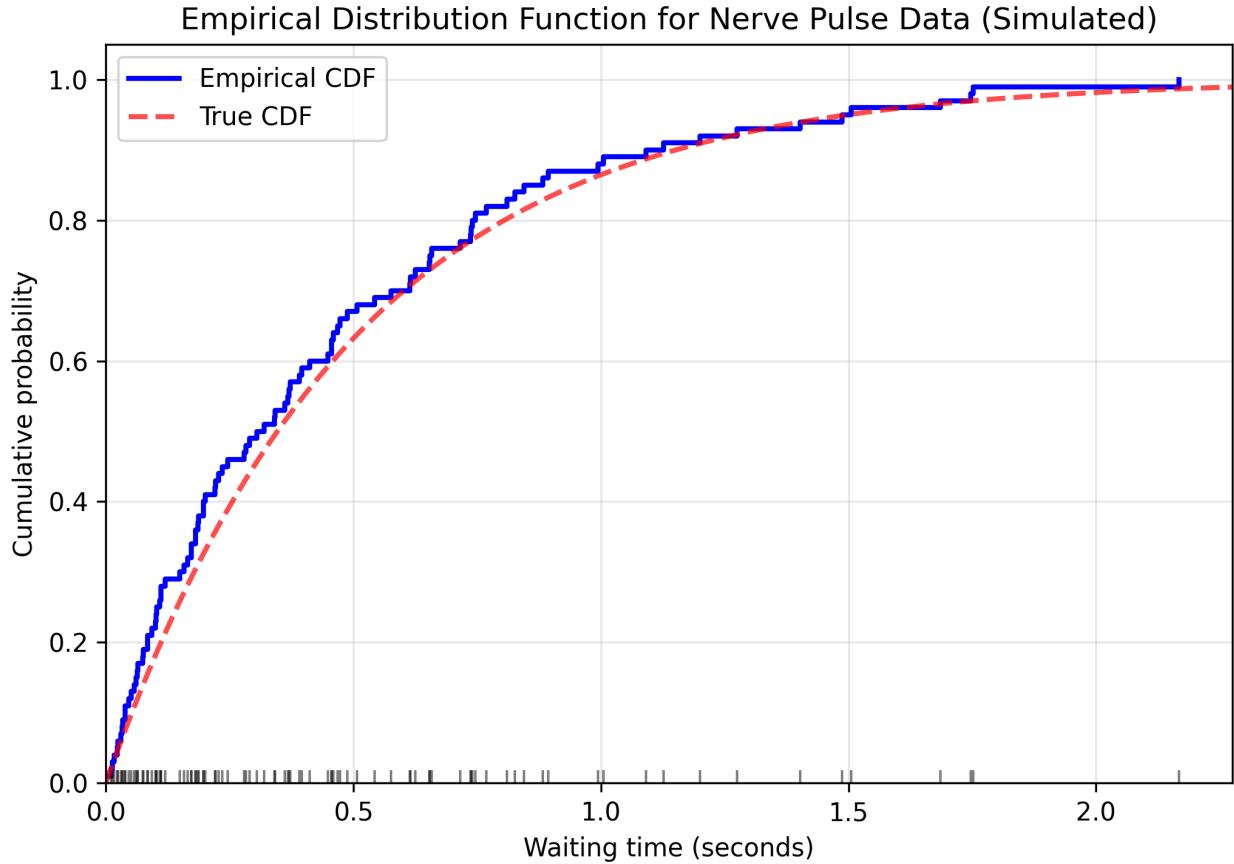
# Plot the EDF as a step function
ax.step(sorted_data, np.arange(1, n+1)/n, where='post',
        linewidth=2, color='blue', label='Empirical CDF')

# Add rug plot to show data points
ax.plot(sorted_data, np.zeros_like(sorted_data), '|',
        color='black', markersize=10, alpha=0.5)

# Add true CDF for comparison
x_range = np.linspace(0, max(sorted_data)*1.1, 1000)
true_cdf = 1 - np.exp(-x_range/0.5) # Exponential CDF
ax.plot(x_range, true_cdf, 'r--', linewidth=2, alpha=0.7, label='True CDF')

ax.set_xlabel('Waiting time (seconds)')
ax.set_ylabel('Cumulative probability')
ax.set_title('Empirical Distribution Function for Nerve Pulse Data (Simulated)')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_xlim(0, max(sorted_data)*1.05)
ax.set_ylim(0, 1.05)

plt.tight_layout()
plt.show()
```



The step function shows \hat{F}_n , jumping by $1/n$ at each data point. The vertical lines at the bottom show the actual data points. Notice how the EDF closely follows the true CDF (shown for comparison).

i Advanced: Confidence Bands for the CDF

We can construct confidence bands for the entire CDF using the [Dvoretzky-Kiefer-Wolfowitz \(DKW\) inequality](#):

$$\mathbb{P} \left(\sup_x |F(x) - \hat{F}_n(x)| > \epsilon \right) \leq 2e^{-2n\epsilon^2}$$

This leads to a $1 - \alpha$ confidence band:

$$L(x) = \max\{\hat{F}_n(x) - \epsilon_n, 0\}$$

$$U(x) = \min\{\hat{F}_n(x) + \epsilon_n, 1\}$$

where $\epsilon_n = \sqrt{\frac{1}{2n} \log \left(\frac{2}{\alpha} \right)}$.

```
# Add confidence bands to previous plot
alpha = 0.05
epsilon_n = np.sqrt(np.log(2/alpha) / (2*n))

fig, ax = plt.subplots(figsize=(7, 5))

# Plot EDF
ax.step(sorted_data, np.arange(1, n+1)/n, where='post',
         linewidth=2, color='blue', label='Empirical CDF')

# Add confidence bands
y_values = np.arange(1, n+1)/n
lower_band = np.maximum(y_values - epsilon_n, 0)
upper_band = np.minimum(y_values + epsilon_n, 1)

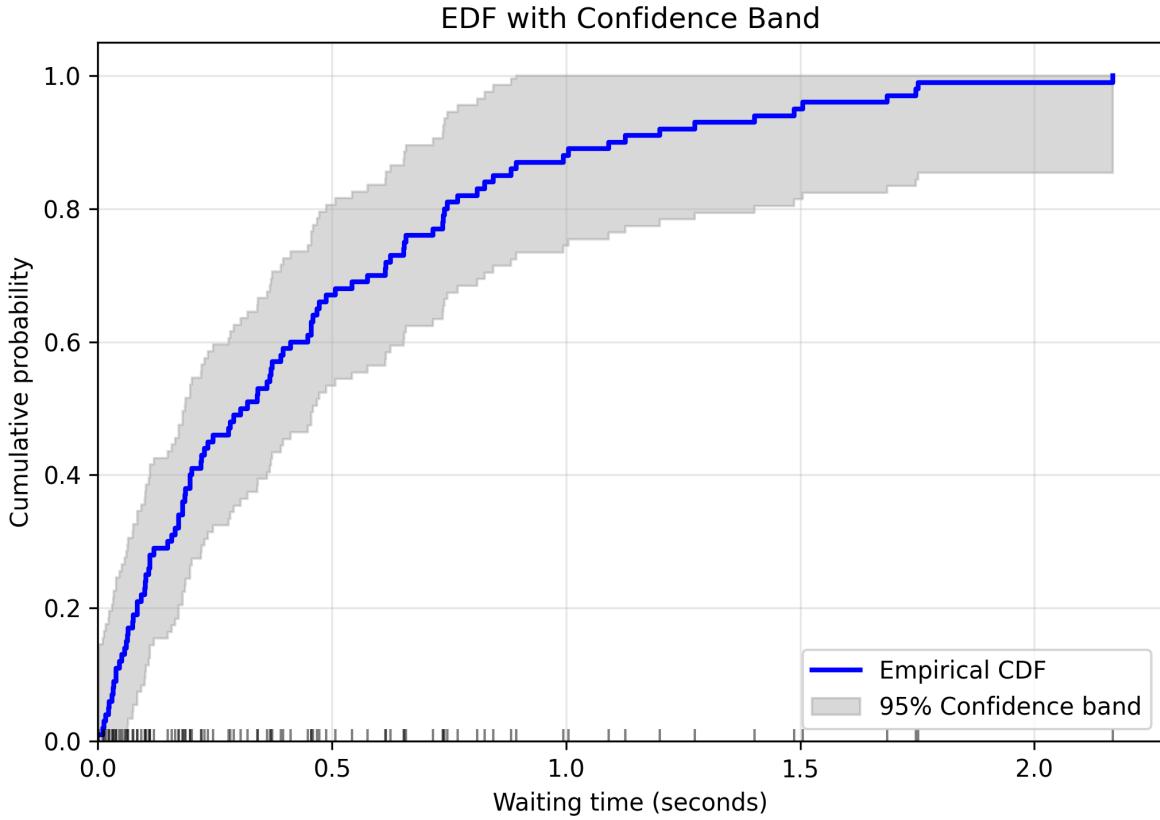
ax.fill_between(sorted_data, lower_band, upper_band,
                 step='post', alpha=0.3, color='gray',
                 label=f'{int((1-alpha)*100)}% Confidence band')

# Add rug plot
ax.plot(sorted_data, np.zeros_like(sorted_data), '|',
        color='black', markersize=10, alpha=0.5)

ax.set_xlabel('Waiting time (seconds)')
ax.set_ylabel('Cumulative probability')
ax.set_title('EDF with Confidence Band')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_xlim(0, max(sorted_data)*1.05)
ax.set_ylim(0, 1.05)

plt.tight_layout()
plt.show()

print(f"Width of confidence band: ±{epsilon_n:.3f}")
print(f"This means we're 95% confident the true CDF lies within this gray region")
```



4.4.2 Estimating Functionals: The Plug-In Estimator

Now that we can estimate the entire distribution, we can estimate any property of it. A **statistical functional** is any function of the CDF F or the PDF f .

Examples include:

- The mean: $\mu = \int xf(x)dx$
- The variance: $\sigma^2 = \int(x - \mu)^2 f(x)dx$
- The median: $m = F^{-1}(1/2)$

The **plug-in estimator** of $\theta = T(F)$ is defined by:

$$\hat{\theta}_n = T(\hat{F}_n)$$

In other words, just *plug in* the empirical CDF \hat{F}_n for the unknown CDF F .

This principle is remarkably general. To estimate any property of the population, we calculate that same property on our empirical distribution. Let's see how this works for various functionals:

The Mean:

Let $\mu = T(F) = \int xf(x)dx$.

The plug-in estimator is:

$$\hat{\mu}_n = \sum x\hat{f}_n(x) = \frac{1}{n} \sum_{i=1}^n X_i = \bar{X}_n$$

The plug-in estimator for the mean is just the sample mean!

The Variance:

Let $\sigma^2 = T(F) = \mathbb{V}(X) = \int x^2 f(x) dx - (\int x f(x) dx)^2$.

The plug-in estimator is:

$$\begin{aligned}\hat{\sigma}_n^2 &= \sum_x x^2 \hat{f}_n(x) - \left(\sum_x x \hat{f}_n(x) \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^n X_i^2 - \left(\frac{1}{n} \sum_{i=1}^n X_i \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2\end{aligned}$$

i Alternative: The Sample Variance

Another common estimator of σ^2 is the sample variance with the $n - 1$ correction:

$$S_n^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2$$

This estimator is unbiased (i.e., $\mathbb{E}[S_n^2] = \sigma^2$) and is almost identical to the plug-in estimator in practice. The factor $\frac{n}{n-1}$ makes little difference for moderate to large samples. Most statistical software uses the $n - 1$ version by default.

The Median:

The median is the value that splits the distribution in half. For the theoretical distribution:

$$m = T(F) = F^{-1}(0.5)$$

The plug-in estimator is simply the sample median - the middle value when we sort our data. For an odd number of observations, it's the middle value. For an even number, it's the average of the two middle values.

i Advanced: Other Statistical Functionals

Skewness (measures asymmetry):

$$T(F) = \frac{\mathbb{E}[(X - \mu)^3]}{\sigma^3} \implies \hat{\kappa}_n = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^3}{(\hat{\sigma}_n^2)^{3/2}}$$

Correlation for bivariate data (X, Y) :

$$T(F) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \implies \hat{\rho}_n = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)(Y_i - \bar{Y}_n)}{\sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2} \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \bar{Y}_n)^2}}$$

Quantiles in general:

$$T(F) = F^{-1}(p) \implies \hat{q}_p = \hat{F}_n^{-1}(p)$$

Since \hat{F}_n is a step function, we define:

$$\hat{F}_n^{-1}(p) = \inf\{x : \hat{F}_n(x) \geq p\}$$

This gives us the sample quantile at level p .

Confidence Intervals for Plug-in Estimators:

When the plug-in estimator $\hat{\theta}_n = T(\hat{F}_n)$ is approximately normally distributed, we can form confidence intervals using:

$$\hat{\theta}_n \pm z_{\alpha/2} \widehat{\text{se}}$$

as we saw earlier in the *Normal-Based Confidence Intervals* section.

The challenge is finding $\widehat{\text{se}}$. For the mean, we know from theory that $\text{se}(\bar{X}_n) = \sigma/\sqrt{n}$, which we can estimate by plugging in $\hat{\sigma}$ to get:

$$\bar{X}_n \pm z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}}$$

But what about the median? The 90th percentile? The interquartile range? For most functionals, **there is no simple formula for the standard error**.

This is where the bootstrap comes to the rescue, as we see next.

i Recap: Nonparametric Estimation

Let $X_1, \dots, X_n \sim F$ be an i.i.d. sample where F is a distribution function on the real line.

- The **empirical distribution function** \hat{F}_n is the CDF that puts mass $1/n$ at each data point X_i
- A **statistical functional** $T(F)$ is any function of F (e.g., mean, variance, median)
- The **plug-in estimator** of $\theta = T(F)$ is $\hat{\theta}_n = T(\hat{F}_n)$

We've seen how to create point estimates for any functional. The challenge is quantifying their uncertainty when no formula exists for the standard error.

4.5 The Bootstrap: Simulating Uncertainty

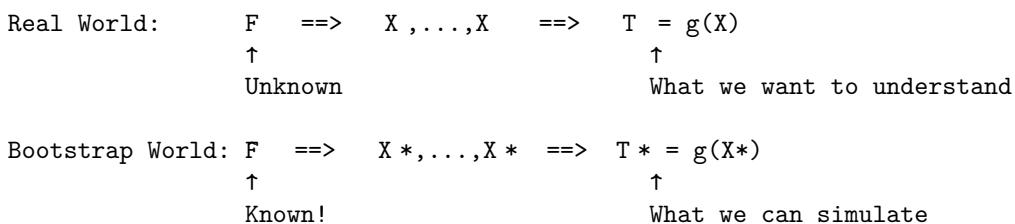
4.5.1 The Core Idea

The bootstrap, invented by Bradley Efron in 1979, is one of the most important statistical innovations of the 20th century. It provides a general, computational method for assessing the uncertainty of virtually any statistic or functional of the data.

Let $T_n = g(X_1, \dots, X_n)$ be our statistic of interest (e.g., the median, the variance, a given quantile). What's its variability?

The key insight is simple: **We can learn about the variability of our statistic by seeing how it varies across different samples. Since we only have one sample, we create new samples by resampling from our data.**

Here's the crucial diagram that illustrates the bootstrap principle:



In the real world:

- Nature draws from the unknown distribution F
- We observe one sample X_1, \dots, X_n
- We calculate our statistic $T_n = g(X_1, \dots, X_n)$
- We want to know the sampling distribution of T_n

In the bootstrap world:

- We draw from the known empirical distribution \hat{F}_n (see below)
- We get a bootstrap sample X_1^*, \dots, X_n^*
- We calculate the bootstrap statistic $T_n^* = g(X_1^*, \dots, X_n^*)$
- **We can repeat the process multiple times to simulate the sampling distribution of T_n^***

The Bootstrap Principle: The distribution of T_n^* around T_n approximates the distribution of T_n around $T(F)$.

How do we draw from \hat{F}_n ? Remember that \hat{F}_n puts mass $1/n$ at each observed data point. Therefore:

⚠ Warning

Key Point: Drawing from \hat{F}_n means sampling **with replacement** from the original data. Each bootstrap sample contains n observations, some repeated, some omitted.

4.5.2 Bootstrap Variance and Standard Error Estimation

Let's make this concrete with an algorithm:

Bootstrap Algorithm for Standard Error Estimation:

1. For $b = 1, \dots, B$:
 - Draw a bootstrap sample X_1^*, \dots, X_n^* by sampling with replacement from $\{X_1, \dots, X_n\}$
 - Compute the statistic for this bootstrap sample: $T_{n,b}^* = g(X_1^*, \dots, X_n^*)$
2. The bootstrap estimate of the standard error is:

$$\widehat{\text{se}}_{\text{boot}} = \sqrt{\frac{1}{B-1} \sum_{b=1}^B (T_{n,b}^* - \bar{T}_n^*)^2}$$

$$\text{where } \bar{T}_n^* = \frac{1}{B} \sum_{b=1}^B T_{n,b}^*$$

Let's implement this for a concrete example – estimating the standard error of the median:

Multiple Perspectives

Python

```

import numpy as np

def bootstrap_se(data, statistic, B=1000):
    """
    Compute bootstrap standard error of a statistic.

    Parameters:
    -----
    data : array-like
        Original sample
    statistic : function
        Function that computes the statistic of interest
    B : int
        Number of bootstrap replications

    Returns:
    -----
    se : float
        Bootstrap standard error
    boot_samples : array
        Bootstrap replications of the statistic
    """
    n = len(data)
    boot_samples = np.zeros(B)

    for b in range(B):
        # Draw bootstrap sample
        x_star = np.random.choice(data, size=n, replace=True)
        # Compute statistic
        boot_samples[b] = statistic(x_star)

    # Compute standard error
    se = np.std(boot_samples, ddof=1)

    return se, boot_samples

# Example: Standard error of the median
np.random.seed(42)
data = np.random.exponential(scale=2, size=50)

# Point estimate
median_est = np.median(data)

# Bootstrap standard error
se_boot, bootmedians = bootstrap_se(data, np.median, B=2000)

print(f"Sample median: {median_est:.3f}")
print(f"Bootstrap SE: {se_boot:.3f}")
print(f"Approximate 95% CI: ({median_est - 2*se_boot:.3f}, {median_est + 2*se_boot:.3f})")

Sample median: 1.146
Bootstrap SE: 0.276
Approximate 95% CI: (0.594, 1.697)

```

R

```

bootstrap_se <- function(data, statistic, B = 1000) {
  #' Compute bootstrap standard error of a statistic
  #
  #' @param data Original sample vector
  #' @param statistic Function that computes the statistic of interest
  #' @param B Number of bootstrap replications
  #' @return List with standard error and bootstrap samples

  n <- length(data)
  boot_samples <- numeric(B)

  for (b in 1:B) {
    # Draw bootstrap sample
    x_star <- sample(data, size = n, replace = TRUE)
    # Compute statistic
    boot_samples[b] <- statistic(x_star)
  }

  # Compute standard error
  se <- sd(boot_samples)

  return(list(se = se, boot_samples = boot_samples))
}

# Example: Standard error of the median
set.seed(42)
data <- rexp(50, rate = 1/2)

# Point estimate
median_est <- median(data)

# Bootstrap standard error
result <- bootstrap_se(data, median, B = 2000)
se_boot <- result$se
bootmedians <- result$boot_samples

cat(sprintf("Sample median: %.3f\n", median_est))
cat(sprintf("Bootstrap SE: %.3f\n", se_boot))
cat(sprintf("Approximate 95% CI: (%.3f, %.3f)\n",
            median_est - 2*se_boot, median_est + 2*se_boot))

```

Let's visualize the bootstrap distribution:

```

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

# Left panel: Original data
ax1.hist(data, bins=20, density=True, alpha=0.7, color='blue', edgecolor='black')
ax1.axvline(median_est, color='red', linestyle='--', linewidth=2, label=f'Median = {median_est:.3f}')
ax1.set_xlabel('Value')

```

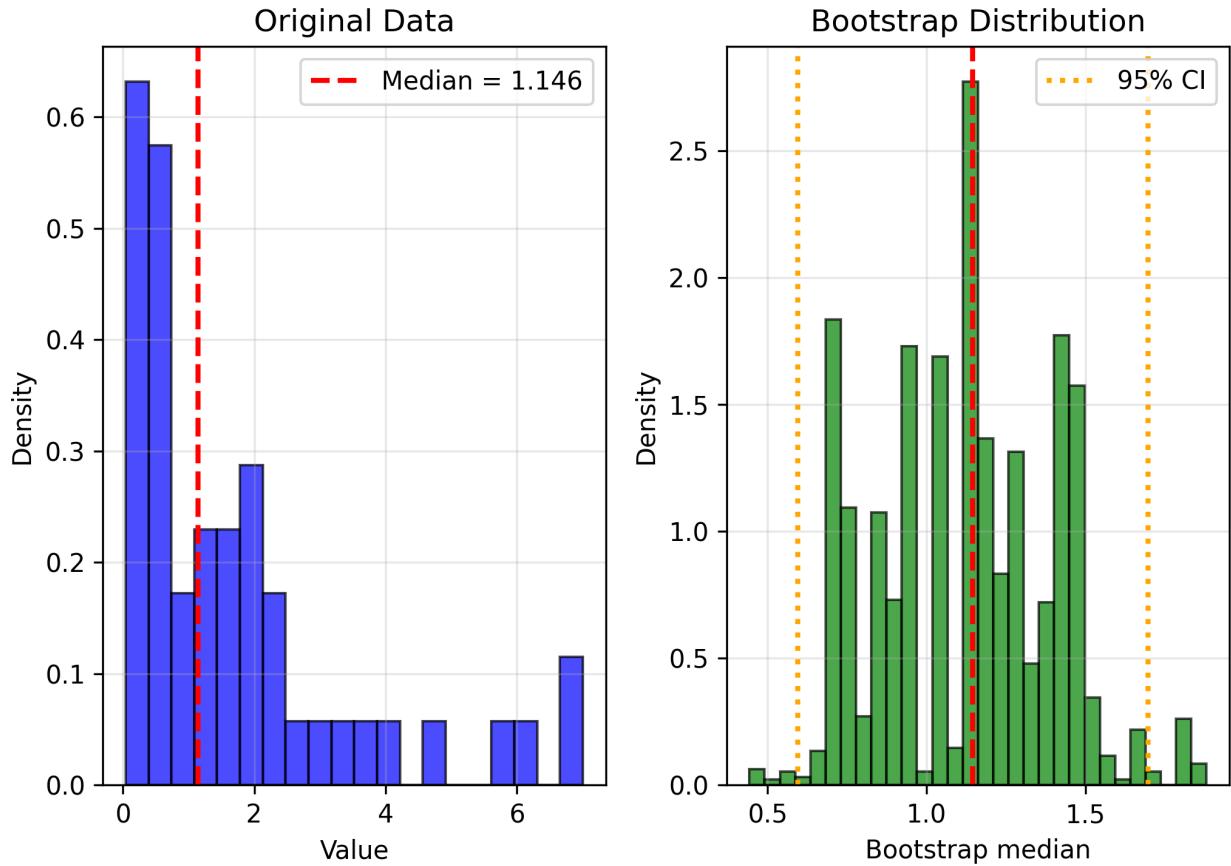
```

ax1.set_ylabel('Density')
ax1.set_title('Original Data')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Right panel: Bootstrap distribution
ax2.hist(boot_medians, bins=30, density=True, alpha=0.7, color='green', edgecolor='black')
ax2.axvline(median_est, color='red', linestyle='--', linewidth=2)
ax2.axvline(median_est - 2*se_boot, color='orange', linestyle=':', linewidth=2)
ax2.axvline(median_est + 2*se_boot, color='orange', linestyle=':', linewidth=2,
            label='95% CI')
ax2.set_xlabel('Bootstrap median')
ax2.set_ylabel('Density')
ax2.set_title('Bootstrap Distribution')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```



The left panel shows our original data. The right panel shows the distribution of the median across 2000 bootstrap samples. This distribution tells us about the uncertainty in our median estimate.

⚠ Two Sources of Error

The bootstrap involves two approximations:

1. **Statistical Approximation:** $\mathbb{V}_F(T_n) \approx \mathbb{V}_{\hat{F}_n}(T_n)$
 - This depends on how well \hat{F}_n approximates F
 - We can't control this – it depends on our sample size n
2. **Monte Carlo Approximation:** $\mathbb{V}_{\hat{F}_n}(T_n) \approx v_{\text{boot}}$
 - This depends on the number of bootstrap samples B
 - We can make this arbitrarily small by increasing B
 - Typically $B \geq 1000$ is sufficient

In formulas:

$$\mathbb{V}_F(T_n) \underset{\text{not so small}}{\approx} \mathbb{V}_{\hat{F}_n}(T_n) \underset{\text{small}}{\approx} v_{\text{boot}}$$

Remember: by increasing the bootstrap samples B we can reduce the Monte Carlo error (due to simulation), but we cannot improve the statistical approximation error without obtaining more real data.

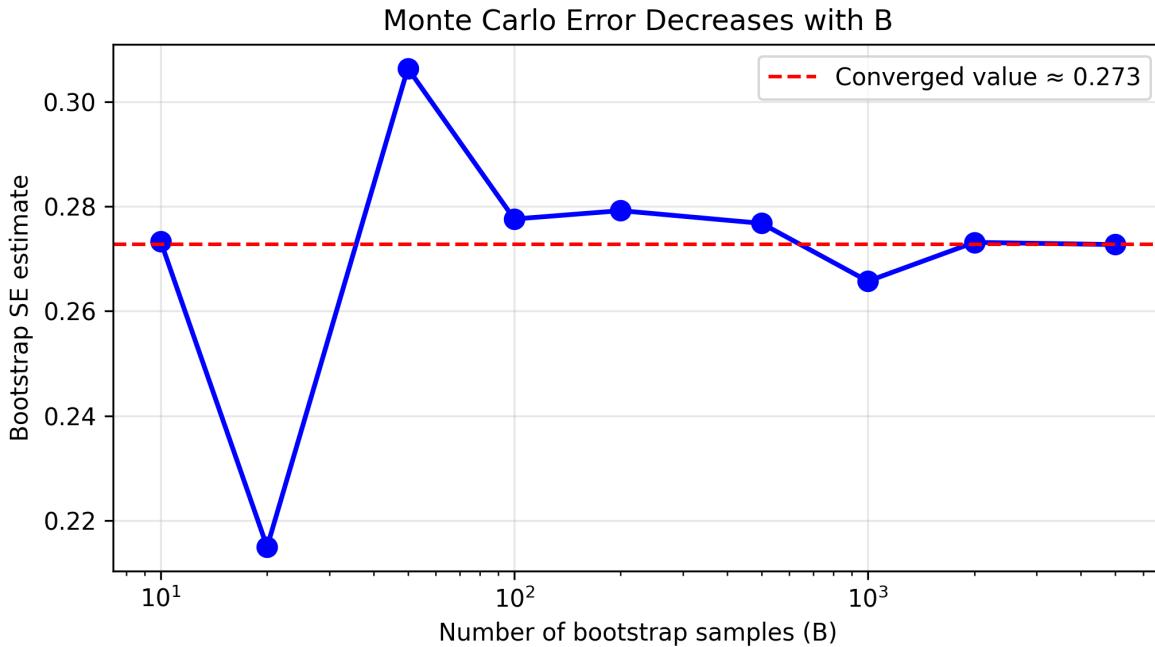
💡 Example: Monte Carlo Error Decreases With More Bootstrap Samples

Let's verify that the Monte Carlo error decreases with B :

```
# Show convergence of bootstrap SE as B increases
B_values = [10, 20, 50, 100, 200, 500, 1000, 2000, 5000]
se_estimates = []

for B in B_values:
    se, _ = bootstrap_se(data, np.median, B=B)
    se_estimates.append(se)

plt.figure(figsize=(7, 4))
plt.plot(B_values, se_estimates, 'bo-', linewidth=2, markersize=8)
plt.axhline(se_estimates[-1], color='red', linestyle='--',
            label=f'Converged value {se_estimates[-1]:.3f}')
plt.xlabel('Number of bootstrap samples (B)')
plt.ylabel('Bootstrap SE estimate')
plt.title('Monte Carlo Error Decreases with B')
plt.xscale('log')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```



The fluctuations for small B are due to Monte Carlo variability. As B increases, the bootstrap standard error estimate stabilizes. Still, this is only one component of the error – to improve further we need additional real data.

4.6 Bootstrap Confidence Intervals

4.6.1 Three Common Methods

Now that we can estimate the sampling distribution of any statistic via the bootstrap, we can construct confidence intervals. But how exactly should we use the bootstrap distribution to form an interval?

There are **three main approaches**, each with different strengths and weaknesses. We'll illustrate all three using a real example: estimating the correlation between a country's economic prosperity and the health of its population.

Example: European Health and Wealth Data

This example explores the correlation between a country's wealth (GDP per capita) and the average lifespan of its citizens (life expectancy at birth). The data is for a subset of EU countries, with [GDP per capita for 2025](#) and [life expectancy from 2023](#).

```

# European Health and Wealth Data
# GDP per capita (2025 forecast), Life Expectancy at birth (2023) for a subset of EU countries.
countries = [
    'Germany', 'France', 'Italy', 'Spain', 'Netherlands', 'Poland',
    'Belgium', 'Sweden', 'Austria', 'Ireland', 'Denmark', 'Finland',
    'Portugal', 'Greece', 'Czech Republic'
]

gdp_per_capita = np.array([
    55911, 46792, 41091, 36192, 70480, 26805, 57772, 58100, 58192,
    108919, 74969, 54163, 30002, 25756, 33039
])

life_expectancy = np.array([
    81.38, 83.33, 83.72, 83.67, 82.16, 78.63, 82.11, 83.26, 81.96,
    82.41, 81.93, 81.91, 82.36, 81.86, 79.83
])

# Combine into a single dataset for resampling
data_combined = np.column_stack((gdp_per_capita, life_expectancy))
n = len(life_expectancy)

# Define correlation function
def correlation(data):
    x, y = data[:, 0], data[:, 1]
    return np.corrcoef(x, y)[0, 1]

# Original correlation
rho_hat = correlation(data_combined)
print(f"Sample correlation: {rho_hat:.3f}")

Sample correlation: 0.238

```

Now let's generate bootstrap samples:

```

# Bootstrap the correlation
B = 5000
boot_correlations = np.zeros(B)

np.random.seed(42)
for b in range(B):
    # Resample pairs (important to maintain pairing!)
    indices = np.random.choice(n, size=n, replace=True)
    boot_sample = data_combined[indices]
    boot_correlations[b] = correlation(boot_sample)

# Bootstrap standard error
se_boot = np.std(boot_correlations, ddof=1)
print(f"Bootstrap SE: {se_boot:.3f}")

```

Bootstrap SE: 0.241

Let's visualize the bootstrap distribution:

```

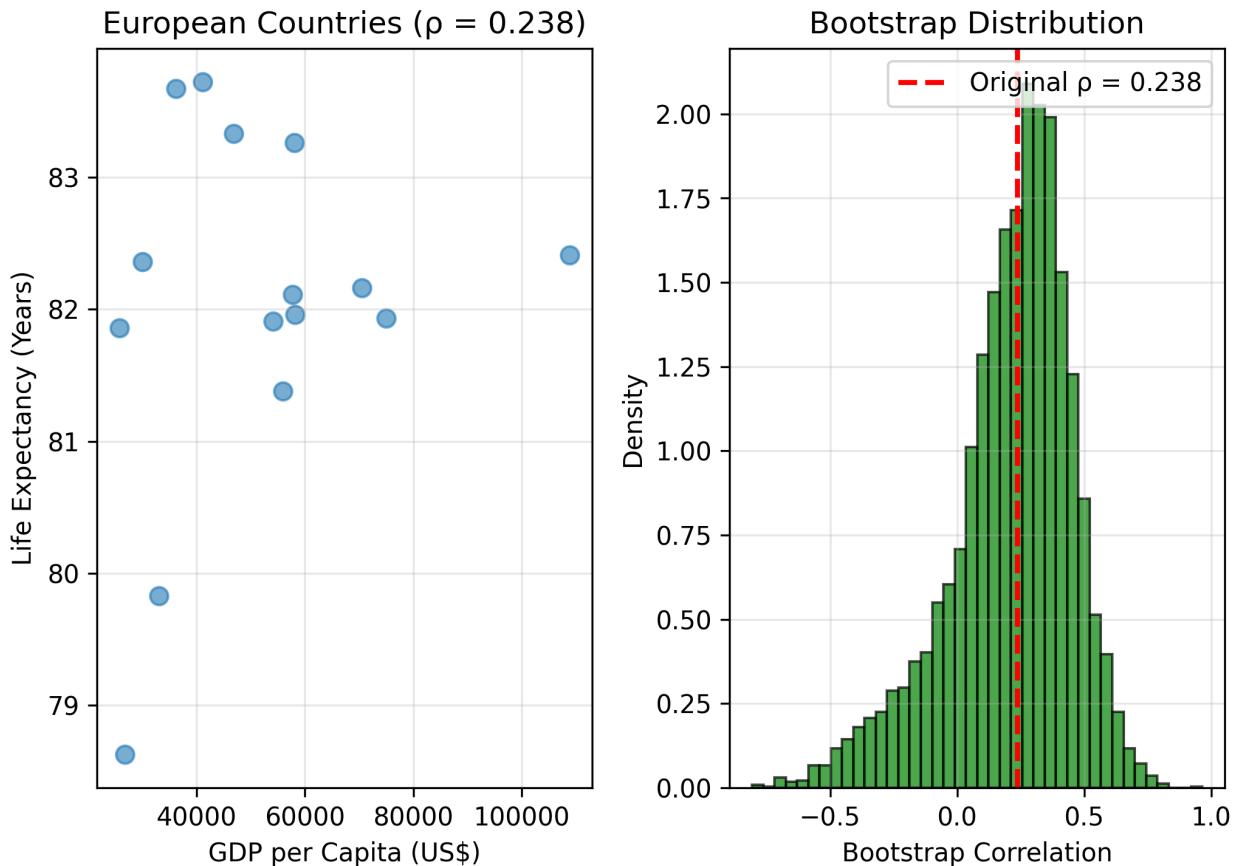
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5))

# Scatter plot of original data
ax1.scatter(gdp_per_capita, life_expectancy, alpha=0.6, s=50)
ax1.set_xlabel('GDP per Capita (US$)')
ax1.set_ylabel('Life Expectancy (Years)')
ax1.set_title(f'European Countries (ρ = {rho_hat:.3f})')
ax1.grid(True, alpha=0.3)

# Bootstrap distribution
ax2.hist(boot_correlations, bins=40, density=True, alpha=0.7,
         color='green', edgecolor='black')
ax2.axvline(rho_hat, color='red', linestyle='--', linewidth=2,
            label=f'Original ρ = {rho_hat:.3f}')
ax2.set_xlabel('Bootstrap Correlation')
ax2.set_ylabel('Density')
ax2.set_title('Bootstrap Distribution')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```



Notice that the bootstrap distribution is somewhat skewed. This skewness will affect our confidence intervals.

4.6.2 Comparing Bootstrap Confidence Intervals

Let $\hat{T}_n = T_n(\hat{F}_n)$ be the statistic evaluated on the empirical distribution, and T_n^* the bootstrap statistic.

Multiple Perspectives

Method 1: Normal Interval

Formula: $\hat{T}_n \pm z_{\alpha/2} \widehat{\text{se}}_{\text{boot}}$

This is the simplest method – we just use the bootstrap standard error in the usual normal-based formula.

```
# Normal interval
alpha = 0.05
z_alpha = stats.norm.ppf(1 - alpha/2)
normal_lower = rho_hat - z_alpha * se_boot
normal_upper = rho_hat + z_alpha * se_boot

print(f"95% Normal interval: ({normal_lower:.3f}, {normal_upper:.3f})")
95% Normal interval: (-0.235, 0.711)
```

Pros:

- Simple and intuitive
- Only requires the standard error, not the full distribution
- Familiar to those who know basic statistics

Cons:

- Assumes the sampling distribution is approximately normal
- Can give nonsensical intervals (e.g., correlation > 1)
- Ignores skewness in the bootstrap distribution

Let's check if our interval makes sense:

```
if normal_upper > 1:
    print(f"Warning: Upper limit {normal_upper:.3f} exceeds 1!")
    print("This is impossible for a correlation!")
```

Method 2: Percentile Interval

Formula: $(T_{n,\alpha/2}^*, T_{n,1-\alpha/2}^*)$

where $T_{n,\beta}^*$ denotes the β sample quantile of the bootstrapped statistic values T_n^* .

The percentile interval method uses the $\alpha/2$ and $1 - \alpha/2$ quantiles of the bootstrap distribution directly.

```
# Percentile interval
percentile_lower = np.percentile(boot_correlations, 100 * alpha/2)
percentile_upper = np.percentile(boot_correlations, 100 * (1 - alpha/2))

print(f"95% Percentile interval: ({percentile_lower:.3f}, {percentile_upper:.3f})")
95% Percentile interval: (-0.389, 0.596)
```

Pros:

- Doesn't assume normality – adapts to the actual shape
- Always respects parameter bounds (correlation stays in [-1, 1])
- Simple to understand: “middle 95% of bootstrap values”
- Works well when the bootstrap distribution is approximately unbiased

Cons:

- Can have poor coverage when there's bias
- Not transformation-invariant
- May not be accurate for small samples

Method 3: Pivotal Interval

Formula: $(2\hat{T}_n - T_{n,1-\alpha/2}^*, 2\hat{T}_n - T_{n,\alpha/2}^*)$

This method assumes that the distribution of $T_n - \theta$ is approximately the same as the distribution of $T_n^* - T_n$, where θ is the true parameter.

```
# Pivotal interval
pivotal_lower = 2 * rho_hat - np.percentile(boot_correlations, 100 * (1 - alpha/2))
pivotal_upper = 2 * rho_hat - np.percentile(boot_correlations, 100 * alpha/2)

print(f"95% Pivotal interval: ({pivotal_lower:.3f}, {pivotal_upper:.3f})")
95% Pivotal interval: (-0.120, 0.865)
```

Pros:

- Often more accurate than the other two methods
- Corrects for bias in the estimator
- Transformation-respecting (invariant under monotone transformations)

Cons:

- Less intuitive – the formula seems backwards at first
- Can occasionally give values outside parameter bounds
- Requires symmetric error distribution for best performance

Let's compare all three intervals visually:

```
fig, ax = plt.subplots(figsize=(7, 5))

# Plot bootstrap distribution
histogram_data = ax.hist(boot_correlations, bins=40, density=True, alpha=0.5,
                        color='gray', edgecolor='black', label='Bootstrap distribution')

# Get the maximum height for positioning intervals
max_density = max(histogram_data[0])

# Add vertical lines for original estimate
ax.axvline(rho_hat, color='red', linestyle='--', linewidth=3, label='Original estimate')

# Add confidence intervals overlaid on the histogram
methods = ['Normal', 'Percentile', 'Pivotal']
intervals = [(normal_lower, normal_upper),
             (percentile_lower, percentile_upper),
             (pivotal_lower, pivotal_upper)]
colors = ['blue', 'green', 'orange']

# Position intervals at different heights on the histogram
y_positions = [max_density * 0.2, max_density * 0.15, max_density * 0.1]

for method, (lower, upper), color, y_pos in zip(methods, intervals, colors, y_positions):
    # Draw the interval line
    ax.plot([lower, upper], [y_pos, y_pos], color=color, linewidth=4,
```

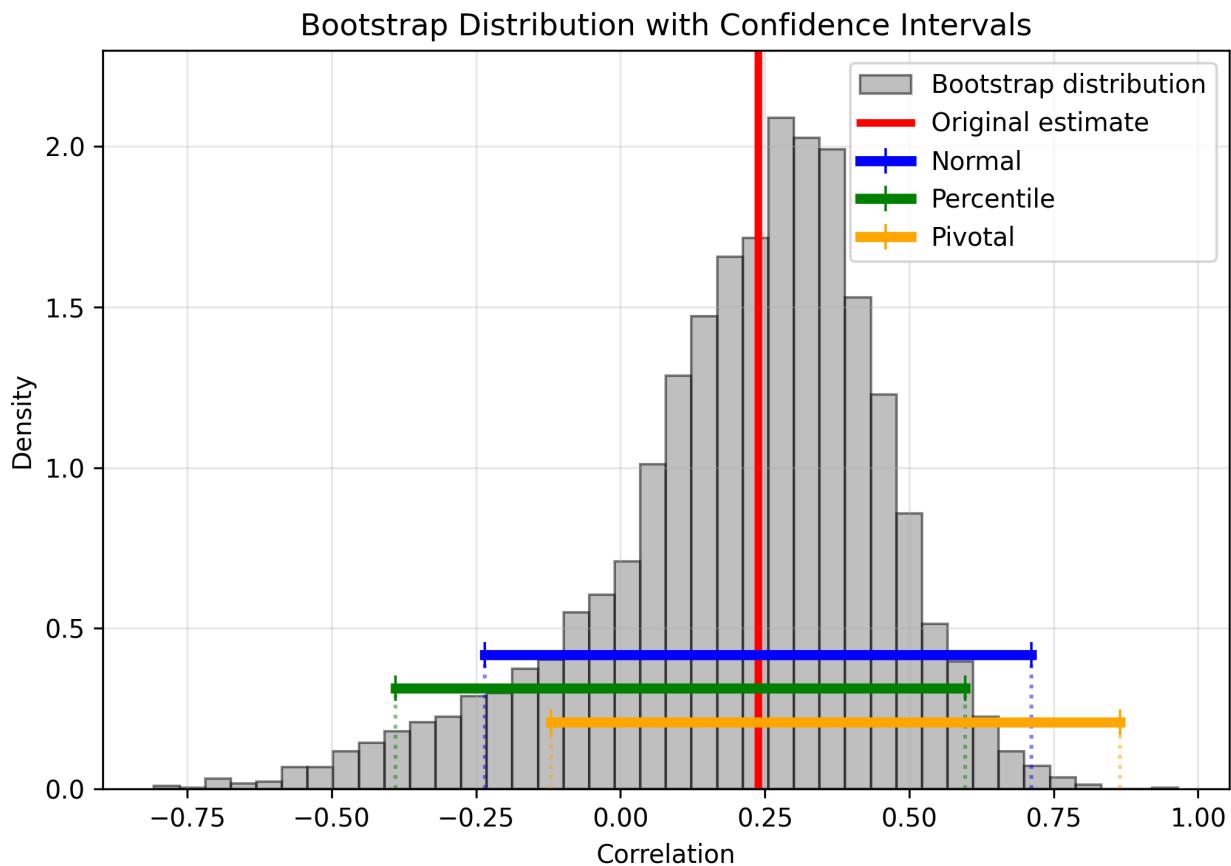
```

        marker='|', markersize=10, label=f'{method}'))
# Add vertical lines at interval endpoints
ax.vlines([lower, upper], 0, y_pos, color=color, linestyle=':', alpha=0.5)

ax.set_xlabel('Correlation')
ax.set_ylabel('Density')
ax.set_title('Bootstrap Distribution with Confidence Intervals')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_ylim(0, max_density * 1.1)

plt.tight_layout()
plt.show()

```



Let's summarize the confidence intervals in a table for easy comparison:

Method	Lower Bound	Upper Bound	Width
Normal	-0.235	0.711	0.946
Percentile	-0.389	0.596	0.986
Pivotal	-0.120	0.865	0.986

In this example:

- The **Normal interval** is symmetric around the point estimate, ignoring the skewness
- The **Percentile interval** reflects the skewness of the bootstrap distribution

- The **Pivotal interval** adjusts for bias and is slightly shifted from the percentile interval

i Advanced: Justification for the Pivotal Interval

The pivotal method seems counterintuitive at first. Why do we subtract the upper quantile from $2\hat{T}_n$ to get the lower bound?

The key insight is that the pivotal interval assumes the error $\hat{T}_n - \theta$ (where $\theta = T(F)$ is the true value) behaves similarly to the bootstrap error $T_n^* - \hat{T}_n$.

Quick derivation: Start with the bootstrap distribution:

$$\mathbb{P}(T_{n,\alpha/2}^* \leq T_n^* \leq T_{n,1-\alpha/2}^*) = 1 - \alpha$$

Subtract \hat{T}_n throughout:

$$\mathbb{P}(T_{n,\alpha/2}^* - \hat{T}_n \leq T_n^* - \hat{T}_n \leq T_{n,1-\alpha/2}^* - \hat{T}_n) = 1 - \alpha$$

The key assumption: The bootstrap error $T_n^* - \hat{T}_n$ has approximately the same distribution as the real error $\hat{T}_n - \theta$. Therefore:

$$\mathbb{P}(T_{n,\alpha/2}^* - \hat{T}_n \leq \hat{T}_n - \theta \leq T_{n,1-\alpha/2}^* - \hat{T}_n) \approx 1 - \alpha$$

Rearranging to isolate θ :

$$\mathbb{P}(2\hat{T}_n - T_{n,1-\alpha/2}^* \leq \theta \leq 2\hat{T}_n - T_{n,\alpha/2}^*) \approx 1 - \alpha$$

This gives us the pivotal interval: $(2\hat{T}_n - T_{n,1-\alpha/2}^*, 2\hat{T}_n - T_{n,\alpha/2}^*)$.

Intuition: If bootstrap values tend to be above our estimate, then our estimate is probably below the truth by a similar amount. The “2×estimate minus bootstrap quantile” formula automatically corrects for this bias.

4.7 Bootstrap Application: Higher Moments

The following example demonstrates both the power and limitations of the bootstrap.

Example: Bootstrap for Higher Moments

Consider a sample of $n = 20$ observations from a standard normal distribution $\mathcal{N}(0, 1)$. We'll use the bootstrap to estimate confidence intervals for two related statistics:

1. $T^{(1)}(F) = \mathbb{E}[X^4]$ - the fourth raw moment
2. $T^{(2)}(F) = \mathbb{E}[(X - \mu)^4]$ - the fourth central moment

For the standard normal, both have the same true value: $T^{(1)}(F) = T^{(2)}(F) = 3$.

This example is valuable because:

- We can compute the true sampling distribution via simulation
- It shows how bootstrap performs for non-standard statistics
- It reveals differences between seemingly similar estimators

Let's implement this comparison:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

```

# Set up the experiment
np.random.seed(42)
n = 20
true_value = 3 # True 4th moment for N(0,1)

# Generate one sample
sample = np.random.normal(0, 1, n)

# Compute point estimates
T1_hat = np.mean(sample**4) # Raw 4th moment
T2_hat = np.mean((sample - np.mean(sample))**4) # Central 4th moment

print(f"True value: {true_value}")
print(f"T¹ (raw 4th moment): {T1_hat:.3f}")
print(f"T² (central 4th moment): {T2_hat:.3f}")

# Bootstrap distributions
B = 2000
boot_T1 = np.zeros(B)
boot_T2 = np.zeros(B)

for b in range(B):
    boot_sample = np.random.choice(sample, size=n, replace=True)
    boot_T1[b] = np.mean(boot_sample**4)
    boot_T2[b] = np.mean((boot_sample - np.mean(boot_sample))**4)

# True sampling distributions (via simulation)
n_true = 10000
true_T1 = np.zeros(n_true)
true_T2 = np.zeros(n_true)

for i in range(n_true):
    true_sample = np.random.normal(0, 1, n)
    true_T1[i] = np.mean(true_sample**4)
    true_T2[i] = np.mean((true_sample - np.mean(true_sample))**4)

# Determine common x-axis range to show the dramatic difference
x_min = min(np.min(boot_T1), np.min(boot_T2), np.min(true_T1), np.min(true_T2))
x_max = max(np.max(boot_T1), np.max(boot_T2), np.max(true_T1), np.max(true_T2))

# Create figure with 4 subplots
fig, axes = plt.subplots(2, 2, figsize=(8, 10))

# T1: Raw 4th moment
# Bootstrap distribution
ax1 = axes[0, 0]
ax1.hist(boot_T1, bins=40, density=True, alpha=0.7, color='blue',
         edgecolor='black', label='Bootstrap')
ax1.axvline(T1_hat, color='red', linestyle='--', linewidth=2,
            label=f'Estimate = {T1_hat:.2f}')
ax1.axvline(true_value, color='black', linestyle=':', linewidth=2,
            label=f'True = {true_value}')
ax1.set_title('T¹: Bootstrap Distribution')

```

```

ax1.set_xlabel('Value')
ax1.set_ylabel('Density')
ax1.set_xlim(x_min, x_max)
ax1.legend()
ax1.grid(True, alpha=0.3)

# True sampling distribution
ax2 = axes[0, 1]
ax2.hist(true_T1, bins=40, density=True, alpha=0.7, color='green',
         edgecolor='black', label='True sampling dist')
ax2.axvline(np.mean(true_T1), color='red', linestyle='--', linewidth=2,
            label=f'Mean = {np.mean(true_T1):.2f}')
ax2.axvline(true_value, color='black', linestyle=':', linewidth=2,
            label=f'True = {true_value}')
ax2.set_title('T1: True Sampling Distribution')
ax2.set_xlabel('Value')
ax2.set_ylabel('Density')
ax2.set_xlim(x_min, x_max)
ax2.legend()
ax2.grid(True, alpha=0.3)

# T2: Central 4th moment
# Bootstrap distribution
ax3 = axes[1, 0]
ax3.hist(boot_T2, bins=40, density=True, alpha=0.7, color='blue',
         edgecolor='black', label='Bootstrap')
ax3.axvline(T2_hat, color='red', linestyle='--', linewidth=2,
            label=f'Estimate = {T2_hat:.2f}')
ax3.axvline(true_value, color='black', linestyle=':', linewidth=2,
            label=f'True = {true_value}')
ax3.set_title('T2: Bootstrap Distribution')
ax3.set_xlabel('Value')
ax3.set_ylabel('Density')
ax3.set_xlim(x_min, x_max)
ax3.legend()
ax3.grid(True, alpha=0.3)

# True sampling distribution
ax4 = axes[1, 1]
ax4.hist(true_T2, bins=40, density=True, alpha=0.7, color='green',
         edgecolor='black', label='True sampling dist')
ax4.axvline(np.mean(true_T2), color='red', linestyle='--', linewidth=2,
            label=f'Mean = {np.mean(true_T2):.2f}')
ax4.axvline(true_value, color='black', linestyle=':', linewidth=2,
            label=f'True = {true_value}')
ax4.set_title('T2: True Sampling Distribution')
ax4.set_xlabel('Value')
ax4.set_ylabel('Density')
ax4.set_xlim(x_min, x_max)
ax4.legend()
ax4.grid(True, alpha=0.3)

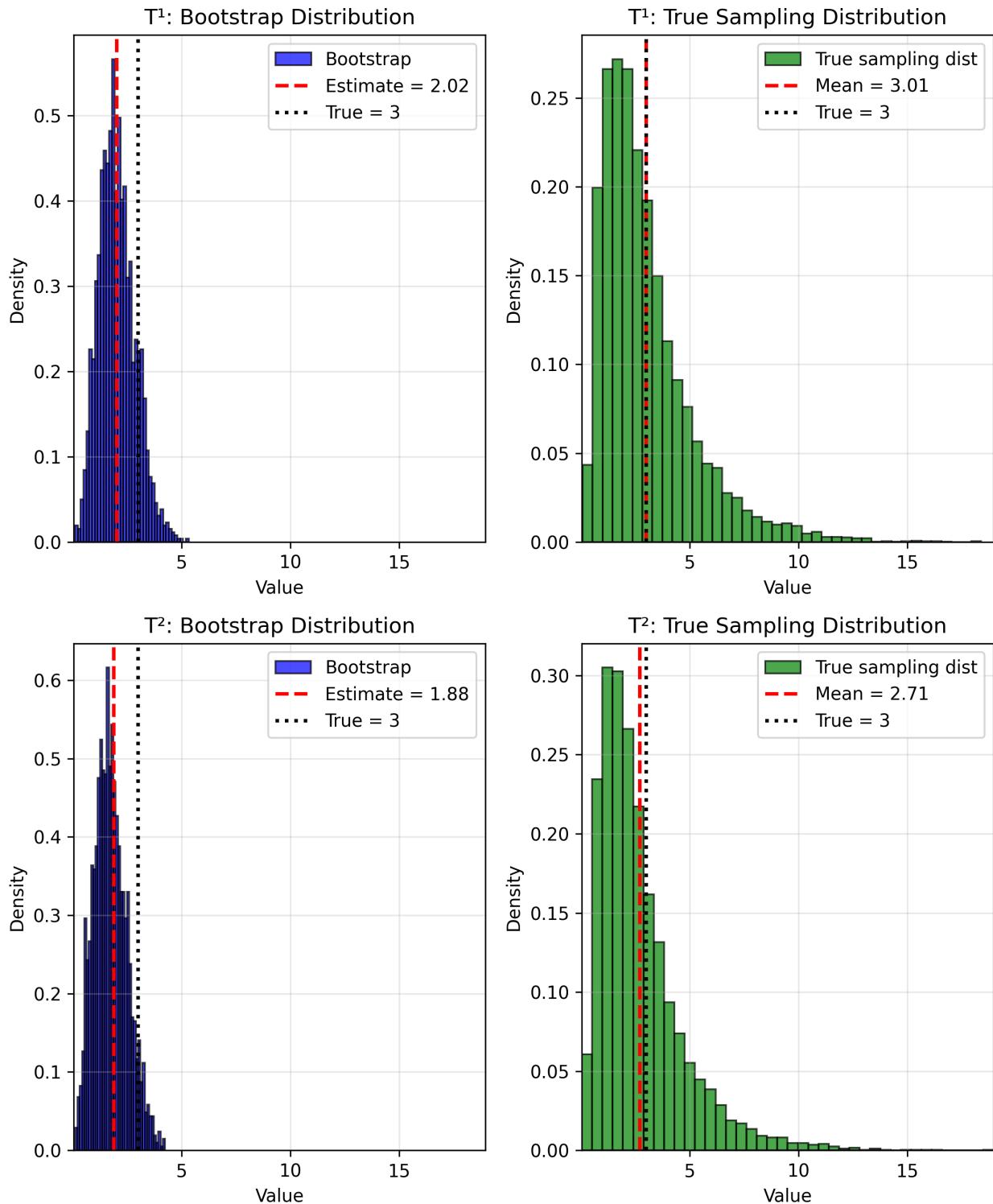
plt.suptitle('Bootstrap vs True Sampling Distributions', fontsize=14)

```

```
plt.tight_layout()  
plt.show()
```

True value: 3
 T^1 (raw 4th moment): 2.025
 T^2 (central 4th moment): 1.882

Bootstrap vs True Sampling Distributions



Now let's compare the confidence intervals:

Confidence Intervals for T¹ (Raw 4th Moment):

True 95% CI: (0.58, 8.84)

Bootstrap Normal: (0.42, 3.63)

Bootstrap Percent: (0.64, 3.79)
Bootstrap Pivotal: (0.26, 3.41)

Confidence Intervals for T^2 (Central 4th Moment):

True 95% CI: (0.50, 7.97)
Bootstrap Normal: (0.41, 3.36)
Bootstrap Percent: (0.48, 3.36)
Bootstrap Pivotal: (0.40, 3.29)

⚠ What Went Wrong?

The bootstrap catastrophically fails here – all methods estimate upper confidence bounds of ~3.4-3.8, while the true bounds are ~8-9 (more than double!).

Why? Fourth moments have heavy-tailed sampling distributions. With only $n=20$ observations, we likely missed the rare extreme values that drive the true variability. The bootstrap can only resample what it sees, so it fundamentally cannot capture the full range of uncertainty.

This isn't just a small sample problem – it's a fundamental limitation when statistics depend heavily on extreme values. Let's explore when else the bootstrap fails...

4.8 When The Bootstrap Fails

The bootstrap is remarkably general, but it's not foolproof. It relies on the sample being a good representation of the population. When this assumption breaks down, so does the bootstrap.

The bootstrap may fail or perform poorly when:

1. **Sample size is too small** (typically $n < 20-30$)
2. **Estimating extreme order statistics** (min, max, extreme quantiles)
3. **Heavy-tailed distributions** without finite moments
4. **Non-smooth statistics** (e.g., number of modes)
5. **Dependent data** (unless using specialized methods like block bootstrap)

Example: Estimators at the Boundary

The bootstrap performs poorly for statistics at the edge of the parameter space. The classic example is estimating the maximum of a uniform distribution:

```

# Uniform distribution example
np.random.seed(42)
true_max = 10
n = 50
uniform_sample = np.random.uniform(0, true_max, n)
sample_max = np.max(uniform_sample)

# True sampling distribution of the maximum
# For Uniform(0, ), the maximum has CDF F(x) = (x/ )^n
x_theory = np.linspace(sample_max * 0.8, true_max, 1000)
pdf_theory = n * (x_theory>true_max)**(n-1) / true_max

# Bootstrap distribution
B = 2000
boot_max_uniform = np.zeros(B)
for b in range(B):
    boot_sample = np.random.choice(uniform_sample, size=n, replace=True)
    boot_max_uniform[b] = np.max(boot_sample)

fig, ax = plt.subplots(figsize=(7, 5))

# Bootstrap histogram
ax.hist(boot_max_uniform, bins=40, density=True, alpha=0.7,
        color='blue', edgecolor='black', label='Bootstrap distribution')

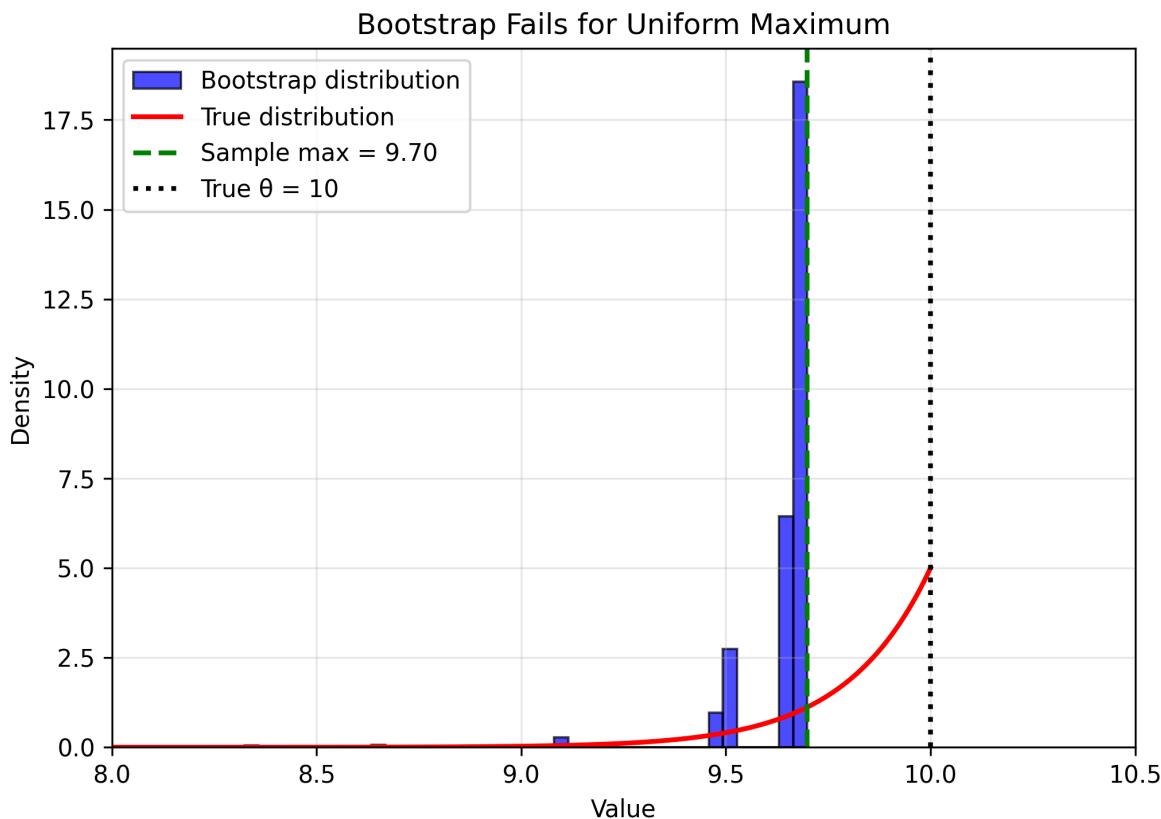
# True distribution
ax.plot(x_theory, pdf_theory, 'r-', linewidth=2, label='True distribution')

# Key values
ax.axvline(sample_max, color='green', linestyle='--', linewidth=2,
           label=f'Sample max = {sample_max:.2f}')
ax.axvline(true_max, color='black', linestyle=':', linewidth=2,
           label=f'True   = {true_max}')

ax.set_xlabel('Value')
ax.set_ylabel('Density')
ax.set_title('Bootstrap Fails for Uniform Maximum')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_xlim(8, 10.5)

plt.tight_layout()
plt.show()

```



Why it fails:

- The true maximum ($\theta = 10$) is always the sample maximum
- The true sampling distribution has support on [sample max, θ]
- But bootstrap samples can never exceed the original sample maximum!
- The bootstrap distribution is entirely to the left of where it should be

Note: Similar biases arise when estimating extreme statistics (e.g., very small or very large quantiles) of any distribution.

Despite these limitations, the bootstrap remains one of the most useful tools in modern statistics. Just remember: it's a powerful method, not a magical one.

4.9 Chapter Summary and Connections

4.9.1 Key Concepts Review

We've covered two fundamental ideas that revolutionized statistical practice:

The Plug-In Principle:

- Estimate the distribution with the empirical distribution function (EDF)
- Estimate any functional $T(F)$ by computing $T(\hat{F}_n)$
- Simple, intuitive, and widely applicable
- Gives us point estimates for any statistic

The Bootstrap:

- Assess uncertainty by resampling from the data
- Create the “bootstrap world” that mimics the real world
- Estimate standard errors and confidence intervals for any statistic

- Three types of confidence intervals: Normal, Percentile, Pivotal

4.9.2 Why These Concepts Matter

For Statistical Practice:

- No need to derive complex formulas for standard errors
- Works for statistics where theory is intractable (median, correlation, etc.)
- Provides a unified approach to uncertainty quantification
- Democratizes statistics – complex inference becomes accessible

For Data Science:

- Computational approach aligns with modern computing power
- Easy to implement and parallelize
- Works with complex models and machine learning algorithms
- Provides uncertainty estimates crucial for decision-making

For Understanding:

- Makes abstract concepts concrete through simulation
- Reveals the sampling distribution visually
- Helps build intuition about statistical variability
- Connects theoretical statistics to computational practice

4.9.3 Common Pitfalls to Avoid

1. **Forgetting to sample with replacement**
 - Bootstrap samples must be the same size as original
 - Sampling without replacement gives wrong answers
2. **Using too few bootstrap samples**
 - Use at least $B = 1,000$ for standard errors
 - Use $B = 10,000$ or more for confidence intervals
 - Monte Carlo error decreases with \sqrt{B}
 - In practice, use as many as you can given your available compute
3. **Misinterpreting confidence intervals**
 - They quantify uncertainty in the estimate
 - They are not probability statements about parameters
 - Different methods can give different intervals
4. **Applying bootstrap blindly**
 - Check if your statistic is smooth
 - Be cautious with very small samples
 - Watch out for boundary cases
5. **Ignoring the assumptions**
 - Bootstrap assumes the sample represents the population
 - It can't fix biased sampling or systematic errors
 - It's not magic – just clever resampling

4.9.4 Chapter Connections

The bootstrap builds on our theoretical foundations and provides a computational path forward:

- **From Previous Chapters:** We've applied the plug-in principle to the empirical distribution (Chapter 1's probability concepts), used Chapter 2's variance formulas for bootstrap standard errors, and provided an alternative to Chapter 3's CLT-based confidence intervals when theoretical distributions are intractable
- **Next - Hypothesis Testing (Chapter 5):** Bootstrap will create null distributions for complex test statistics, complemented by permutation tests as another resampling approach

- **Parametric Methods (Chapters 6-7):** Compare bootstrap to theoretical approaches, use it to validate assumptions, and construct confidence intervals for maximum likelihood estimates when standard theory is difficult
- **Machine Learning Applications:** Bootstrap underpins ensemble methods (bagging), provides uncertainty quantification for predictions, and helps with model selection—making it essential for modern data science

4.9.5 Rejoinder: Coming Full Circle

Recall our opening example: the healthcare administrator planning hospital capacity during an epidemic. We began by asking how to quantify the uncertainty in our estimates, not just provide point predictions.

Through this chapter, we've answered that question with two powerful tools:

1. **The plug-in principle** gave us a way to estimate any property of a distribution
2. **The bootstrap** gave us a way to quantify the uncertainty of those estimates

These tools enable data-driven decision making by providing not just estimates, but confidence intervals that capture the range of plausible values. Whether you're estimating hospital capacity, financial risk, or any other critical quantity, you now have the tools to say not just “we expect 500 patients” but “we’re 95% confident it will be between 300 and 700 patients.”

This transformation from point estimates to uncertainty quantification is what makes statistics invaluable for real-world decision making.

4.9.6 Practical Advice

1. **Start simple:** Use percentile intervals as your default
2. **Visualize:** Always plot the bootstrap distribution
3. **Compare methods:** Try different CI methods to check robustness
4. **Think about assumptions:** Is your sample representative?
5. **Use modern tools:** Most software has built-in bootstrap functions

The bootstrap exemplifies the shift from mathematical to computational statistics. Master it, and you'll have a powerful tool for almost any statistical problem you encounter.

4.9.7 Quick Self-Check

Before moving on, test your understanding with these questions:

1. **What is bootstrap sampling used for?**

i Answer

- Estimating the sampling distribution of a statistic
- Computing standard errors and confidence intervals
- Assessing uncertainty when no formula exists

The bootstrap is particularly valuable when theoretical formulas are intractable or don't exist, such as for the median, correlation coefficient, or complex machine learning predictions.

2. **What approximations does the method make?**

i Answer

- **Statistical approximation:** Assumes \hat{F}_n approximates F well
 - This depends on sample size n and cannot be improved without more data

- **Monte Carlo approximation:** Finite B approximates infinite resampling
 - This can be made arbitrarily small by increasing B (typically $B \geq 1,000$)

Remember: $\mathbb{V}_F(T_n) \underset{\text{statistical error}}{\approx} \mathbb{V}_{\hat{F}_n}(T_n) \underset{\text{Monte Carlo error}}{\approx} v_{\text{boot}}$

3. What are its limitations?

i Answer

- **Small samples** (typically $n < 20 - 30$): The empirical distribution poorly represents the population
- **Extreme order statistics:** Cannot extrapolate beyond observed data range (e.g., max of uniform distribution)
- **Heavy-tailed distributions:** May miss rare extreme values that drive variability (as we saw with 4th moments)
- **Non-smooth statistics:** Discontinuous functions like the number of modes
- **Dependent data:** Requires specialized methods like block bootstrap for time series

The key limitation: bootstrap cannot see what's not in your sample!

4. How can bootstrap be used to construct confidence intervals?

i Answer

Three main methods, each with different strengths:

- **Normal interval:** $\hat{T}_n \pm z_{\alpha/2} \cdot \widehat{\text{se}}_{\text{boot}}$
 - Simplest, but assumes normality
 - Can give impossible values (e.g., correlation > 1)
- **Percentile interval:** $(T_{n,\alpha/2}^*, T_{n,1-\alpha/2}^*)$
 - Uses bootstrap quantiles directly
 - Respects parameter bounds, good default choice
- **Pivotal interval:** $(2\hat{T}_n - T_{n,1-\alpha/2}^*, 2\hat{T}_n - T_{n,\alpha/2}^*)$
 - Corrects for bias, often most accurate
 - Can occasionally exceed parameter bounds

Choose based on your specific problem and always visualize the bootstrap distribution!

4.9.8 Self-Test Problems

1. **Implementing Bootstrap:** Write a function to bootstrap the trimmed mean (removing top and bottom 10% before averaging). Compare its standard error to the regular mean for normal and heavy-tailed data.
2. **Correlation CI:** Using the European Health and Wealth data from the chapter, compute bootstrap confidence intervals for ρ^2 (squared correlation). How do the three methods compare? What happens to the intervals when you transform from ρ to ρ^2 ?
3. **Ratio Statistics:** Given paired data (X_i, Y_i) , bootstrap the ratio \bar{Y}/\bar{X} . Why might this be challenging? Compare the three CI methods.
4. **Bootstrap Failure - Range Statistic:** Generate $n = 30$ observations from a standard normal distribution and bootstrap the range (max - min). Compare the bootstrap distribution to the true sampling distribution (via simulation). Why does the bootstrap underestimate the variability? How does this relate to our discussion of extreme order statistics?

4.9.9 Connections to Source Material

i Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Introduction and Motivation	Expanded material from the slides, providing context for nonparametric estimation and the bootstrap.
Confidence Sets: The Foundation	
Definition and Interpretation of Confidence Intervals	AoS §6.3.2
Normal-Based Confidence Intervals	AoS §6.3.2 (Theorem 6.16)
The Plug-In Principle: A General Method for Estimation	
The Empirical Distribution Function (EDF)	AoS §7.1 (Definition 7.1)
Properties of the EDF (Glivenko-Cantelli)	AoS §7.1 (Theorems 7.3, 7.4)
Confidence Bands for the CDF (DKW Inequality)	AoS §7.1 (Theorem 7.5)
The Plug-In Estimator for Statistical Functionals	AoS §7.2 (Definition 7.7)
Plug-in Examples (Mean, Variance, etc.)	AoS §7.2 (Examples 7.10, 7.11, etc.)
The Bootstrap: Simulating Uncertainty	
The Core Idea and Bootstrap World	AoS §8 (Introduction), §8.2
Bootstrap Variance and Standard Error Estimation	AoS §8.2
Bootstrap Confidence Intervals	
Three Common Methods (Normal, Percentile, Pivotal)	AoS §8.3
Comparing Bootstrap CIs (Health and Wealth Example)	New example, applies concepts from AoS §8.3.
Bootstrap Application: Higher Moments When The Bootstrap Fails	
Chapter Summary and Connections	New example from the slides. New material, summarizing common failure modes. Examples inspired by AoS exercises (e.g., §8.6 Q7 for Uniform max). New summary material.

4.9.10 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

Remember: The bootstrap transforms the abstract problem of understanding sampling distributions into the concrete task of resampling from your data. It's statistics made tangible through computation. When in doubt, bootstrap it!

Chapter 5

Parametric Inference I: Finding Estimators

5.1 Learning Objectives

After completing this chapter, you will be able to:

- Define a parametric model and explain its role in statistical inference, distinguishing between parameters of interest and nuisance parameters.
- Derive estimators using the Method of Moments (MoM) by equating theoretical and sample moments.
- Explain the principle of Maximum Likelihood Estimation (MLE) and formulate the likelihood and log-likelihood functions for a given model.
- Find the Maximum Likelihood Estimator (MLE) analytically in simple cases by maximizing the (log-)likelihood.
- Explain when and why numerical optimization is necessary for MLE, and apply standard optimization libraries to find estimators computationally.

i Note

This chapter covers parametric models and two fundamental approaches to finding estimators: the Method of Moments and Maximum Likelihood Estimation. The material is adapted from Chapter 9 of Wasserman (2013) and supplemented with computational examples and optimization techniques relevant to modern data science applications.

5.2 Introduction: Machine Learning As Statistical Estimation

When fitting a machine learning model $f(x, \theta)$ for regression with inputs x_i and targets y_i , $i = 1, \dots, n$, a common approach is to minimize the **mean squared error (MSE)**:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (f(x_i, \theta) - y_i)^2.$$

The terms in this expression are similar to the log-probability of a normal distribution:

$$\log \mathcal{N}(y_i; f(x_i, \theta), \sigma^2) = -\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (f(x_i, \theta) - y_i)^2.$$

This similarity is not a coincidence. When σ is constant, maximizing the log-likelihood means maximizing $-\frac{1}{2\sigma^2} \sum_i (f(x_i, \theta) - y_i)^2$ plus a constant – which is equivalent to minimizing the MSE.

i The ML-Statistics Connection

When you minimize MSE in machine learning, you’re performing maximum likelihood estimation under a Gaussian noise assumption! This connection reveals a fundamental truth: many machine learning algorithms are secretly (or openly) solving statistical estimation problems.

This chapter introduces the foundational principles of **parametric inference** – the engine that powers both classical statistics and modern machine learning. We’ll learn two primary methods for finding estimators for model parameters: the Method of Moments and the celebrated Maximum Likelihood Estimation.

i Finnish Terminology Reference

For Finnish-speaking students, here’s a reference table of key terms in this chapter:

English	Finnish	Context
Parametric model	Parametrisen malli	Models with finite parameters
Parameter of interest	Kiennostuksen parametri	The quantity we want to estimate
Nuisance parameter	Kiusaparametri	Parameters not of primary interest
Method of Moments (MoM)	Momenttimenetelmä	Estimation by matching moments
Moment	Momentti	Expected value of powers
Sample moment	Otosmomentti	Empirical average of powers
Maximum Likelihood Estimation (MLE)	Suurimman uskottavuuden menetelmä	Most common estimation method
Likelihood function	Uskottavuusfunktio	Joint density as function of parameters
Log-likelihood function	Log-uskottavuusfunktio	Logarithm of likelihood
Maximum Likelihood Estimator	SU-estimaattori	Parameter maximizing likelihood
Numerical optimization	Numeerinen optimointi	Computational methods for finding optima
Gradient	Gradientti	Vector of partial derivatives
Gradient descent	Gradienttimenetelmä	Iterative optimization algorithm

5.3 Parametric Models

We introduced parametric models in Chapter 3 when discussing statistical inference frameworks. Recall that in the world of statistical inference, we often make assumptions about the structure of our data-generating process. A **parametric model** is one such assumption – it postulates that our data comes from a distribution that can be fully characterized by a finite number of parameters.

Now that we’re diving into estimation methods, let’s revisit this concept with a focus on how we actually *find* these parameters.

A **parametric model** is a set of distributions

$$\mathfrak{F} = \{f(x; \theta) : \theta \in \Theta\}$$

where:

- $\theta = (\theta_1, \dots, \theta_k)$ is the **parameter** (possibly vector-valued)
- $\Theta \subseteq \mathbb{R}^k$ is the **parameter space** (the set of all possible parameter values)
- $f(x; \theta)$ is the density or distribution function indexed by θ

The key insight: We assume the data-generating process belongs to a specific family of distributions, and our job is just to find the right parameter θ within that family.

In other words, the problem of inference reduces to estimating the parameter(s) θ .

i All Models Are Wrong, But Some Are Useful

Parametric models are widely used although the underlying models are usually not perfect. As the statistician George Box famously said – in what is possibly the most repeated quote in statistics – “*All models are wrong, but some are useful.*”

When the model is good enough, parametric models can be very useful because they offer a simple representation for potentially complex phenomena. The art of statistical modeling is finding a model that is wrong in acceptable ways while still capturing the essential features of your data.

Examples of Parametric Models:

- **Simple distributions:**
 - Bernoulli(p): One parameter determining success probability
 - Poisson(λ): One parameter determining both mean and variance
 - Normal(μ, σ^2): Two parameters for location and scale
- **Regression models:**
 - Linear regression: $Y = \beta_0 + \beta_1 X + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$
 - Logistic regression: $P(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$
- **Finite mixture models:**
 - Mixture of Gaussians: $f(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x; \mu_k, \sigma_k^2)$
 - Parameters include mixing weights π_k and component parameters (μ_k, σ_k)
- **Machine Learning models:**
 - Deep Neural Networks: Often millions of parameters (weights and biases)
 - Despite their complexity, these are still parametric models!

i Remember: Parameters of Interest vs. Nuisance Parameters

We introduced this distinction in Chapter 3, which it's crucial for estimation:

- **Parameter of interest:** The specific quantity $T(\theta)$ we want to estimate
- **Nuisance parameter:** Other parameters we must estimate but don't care about directly

Example: Mean Lifetime Estimation

Equipment lifetimes often follow a Gamma distribution. If $X_1, \dots, X_n \sim \text{Gamma}(\alpha, \beta)$, then:

$$f(x; \alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta}, \quad x > 0$$

If we want to estimate the mean lifetime:

- **Parameter of interest:** $T(\alpha, \beta) = \mathbb{E}(X) = \alpha\beta$
- **Nuisance parameters:** The individual shape (α) and scale (β) parameters

Note that we must estimate both parameters, but only their product matters for our question.

5.4 The Method of Moments (MoM)

The Method of Moments is a simple estimation technique that does not yield optimal estimators, but provides easy-to-compute values that can serve as good starting points for more sophisticated methods.

5.4.1 The Principle: Matching Moments

The Method of Moments is based on a straightforward idea: if our model is correct, then theoretical properties of the distribution (moments) should match their empirical counterparts in the data. It's like saying, "If this really is the right distribution, then the average I calculate from my model should match the average I see in my data."

For a model with parameter $\theta = (\theta_1, \dots, \theta_k)$:

- The j^{th} **theoretical moment** is: $\alpha_j(\theta) = \mathbb{E}_\theta(X^j)$
- The j^{th} **sample moment** is: $\hat{\alpha}_j = \frac{1}{n} \sum_{i=1}^n X_i^j$

The **Method of Moments estimator** $\hat{\theta}_n$ is the value of θ such that:

$$\alpha_1(\hat{\theta}_n) = \hat{\alpha}_1 \quad (5.1)$$

$$\alpha_2(\hat{\theta}_n) = \hat{\alpha}_2 \quad (5.2)$$

$$\vdots \quad (5.3)$$

$$\alpha_k(\hat{\theta}_n) = \hat{\alpha}_k \quad (5.4)$$

This gives us a system of k equations with k unknowns – exactly what we need to solve for k parameters!

5.4.2 MoM in Action: Examples

Example: Bernoulli Distribution

For $X_1, \dots, X_n \sim \text{Bernoulli}(p)$, we have one parameter to estimate.

- Theoretical first moment: $\alpha_1(p) = \mathbb{E}_p(X) = p$
- Sample first moment: $\hat{\alpha}_1 = \frac{1}{n} \sum_{i=1}^n X_i = \bar{X}_n$

Equating them: $p = \bar{X}_n$

Therefore, the MoM estimator is $\hat{p}_{\text{MoM}} = \bar{X}_n$ – simply the proportion of successes!

Example: Normal Distribution

For $X_1, \dots, X_n \sim \mathcal{N}(\mu, \sigma^2)$, we have two parameters, so we need two equations.

First moment equation:

- $\alpha_1(\theta) = \mathbb{E}(X) = \mu$
- $\hat{\alpha}_1 = \bar{X}_n$
- Setting equal: $\mu = \bar{X}_n$

Second moment equation:

- $\alpha_2(\theta) = \mathbb{E}(X^2) = \mathbb{V}(X) + (\mathbb{E}(X))^2 = \sigma^2 + \mu^2$
- $\hat{\alpha}_2 = \frac{1}{n} \sum_{i=1}^n X_i^2$
- Setting equal: $\sigma^2 + \mu^2 = \frac{1}{n} \sum_{i=1}^n X_i^2$

Solving this system:

- $\hat{\mu}_{\text{MoM}} = \bar{X}_n$
- $\hat{\sigma}_{\text{MoM}}^2 = \frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}_n^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2$

Example: Gamma Distribution

For $X_1, \dots, X_n \sim \text{Gamma}(\alpha, \beta)$, let's derive the MoM estimators.

The Gamma distribution has:

- First moment: $\mathbb{E}(X) = \alpha\beta$
- Second moment: $\mathbb{E}(X^2) = \mathbb{V}(X) + (\mathbb{E}(X))^2 = \alpha\beta^2 + (\alpha\beta)^2$

Setting up the moment equations:

$$\alpha\beta = \bar{X}_n$$

$$\alpha\beta^2 + (\alpha\beta)^2 = \frac{1}{n} \sum_{i=1}^n X_i^2$$

From the first equation: $\alpha = \bar{X}_n/\beta$

Substituting into the second equation:

$$\frac{\bar{X}_n}{\beta} \cdot \beta^2 + \bar{X}_n^2 = \frac{1}{n} \sum_{i=1}^n X_i^2$$

Simplifying:

$$\bar{X}_n\beta + \bar{X}_n^2 = \frac{1}{n} \sum_{i=1}^n X_i^2$$

Solving for β :

$$\beta = \frac{\frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}_n^2}{\bar{X}_n} = \frac{\text{sample variance}}{\bar{X}_n}$$

Therefore, the MoM estimators are:

- $\hat{\beta}_{\text{MoM}} = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2}{\bar{X}_n}$
- $\hat{\alpha}_{\text{MoM}} = \frac{\bar{X}_n}{\hat{\beta}_{\text{MoM}}} = \frac{\bar{X}_n^2}{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2}$

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Generate data from a Gamma distribution
np.random.seed(42)
true_alpha, true_beta = 3.0, 2.0 # True parameters
n = 100
data = stats.gamma.rvs(a=true_alpha, scale=true_beta, size=n)

# Method of Moments for Gamma distribution
# For Gamma( , ): E[X] = , E[X2] = (+1)2
sample_mean = np.mean(data)
sample_second_moment = np.mean(data**2)

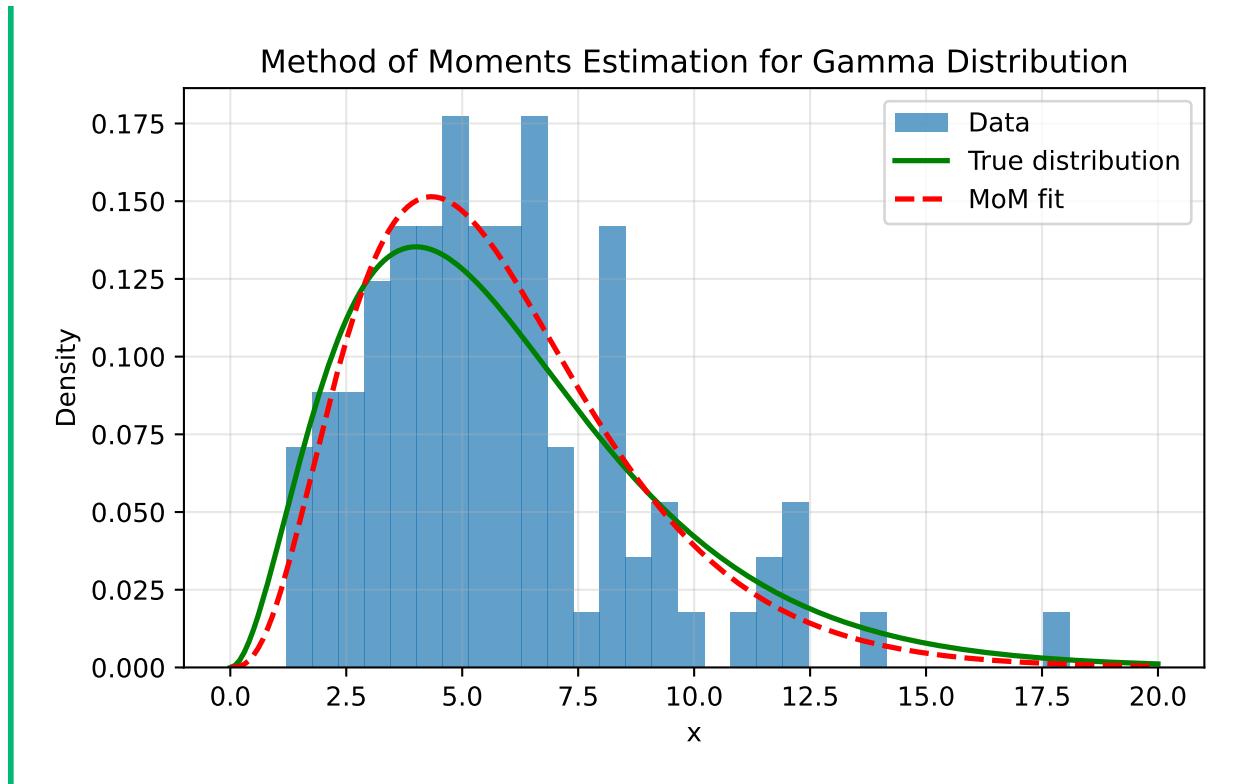
# Solve the system of equations
# mean = *
# second_moment = *2 * ( + 1) = *2 + *2
# This gives us: = (second_moment - mean2) / mean
mom_beta = (sample_second_moment - sample_mean**2) / sample_mean
mom_alpha = sample_mean / mom_beta

print(f"True parameters:   = {true_alpha},   = {true_beta}")
print(f"MoM estimates:     = {mom_alpha:.3f},   = {mom_beta:.3f}")

# Visualize the fit
x = np.linspace(0, 20, 200)
plt.figure(figsize=(7, 4))
plt.hist(data, bins=30, density=True, alpha=0.7, label='Data')
plt.plot(x, stats.gamma.pdf(x, a=true_alpha, scale=true_beta),
          'g-', linewidth=2, label='True distribution')
plt.plot(x, stats.gamma.pdf(x, a=mom_alpha, scale=mom_beta),
          'r--', linewidth=2, label='MoM fit')
plt.xlabel('x')
plt.ylabel('Density')
plt.title('Method of Moments Estimation for Gamma Distribution')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

True parameters:   = 3.0,   = 2.0
MoM estimates:     = 3.869,   = 1.511

```



5.4.3 Properties of Method of Moments Estimator

Under regular conditions, Method of Moments estimators have some desirable properties:

Let $\hat{\theta}_n$ denote the method of moments estimator. Under appropriate conditions on the model, the following statements hold:

1. **Existence:** The estimate $\hat{\theta}_n$ exists with probability tending to 1.

2. **Consistency:** The estimate is consistent:

$$\hat{\theta}_n \xrightarrow{P} \theta$$

3. **Asymptotic Normality:** The estimate is asymptotically Normal:

$$\sqrt{n}(\hat{\theta}_n - \theta) \rightsquigarrow N(0, \Sigma)$$

where

$$\Sigma = g\mathbb{E}_\theta(YY^T)g^T,$$

with $Y = (X, X^2, \dots, X^k)^T$ and $g = (g_1, \dots, g_k)$ where $g_j = \frac{\partial \alpha_j^{-1}(\theta)}{\partial \theta}$.

The last result can yield confidence intervals, but bootstrap is usually easier.

i How Good are MoM Estimators?

Strengths:

- Simple to compute – just solve algebraic equations!
- Guaranteed existence and consistency under mild conditions
- Asymptotically normal, enabling confidence intervals

Weaknesses:

- **Not efficient:** Other estimators (like MLE) typically have smaller variance

- **May give impossible values:** Can produce estimates outside the parameter space
- **Arbitrary:** Why use moments? Why not other features of the distribution?

Primary Use Case: MoM estimators are excellent starting values for more sophisticated methods like maximum likelihood estimation, which often require numerical optimization.

5.5 Maximum Likelihood Estimation (MLE)

5.5.1 The Principle: What Parameter Makes My Data Most Probable?

Maximum Likelihood Estimation is arguably the most important estimation method in statistics. While the Method of Moments asks “what parameters make the theoretical moments match the empirical ones?”, MLE asks a more direct question: “what parameter values make my observed data most plausible?”

The elegance of MLE is that it reverses our usual probability thinking:

- **Probability:** Given parameters, what data would we expect to see?
- **Likelihood:** Given data, which parameters make this data most probable?

5.5.2 The Likelihood Function

The mathematical object at the core of MLE is the likelihood function.

Let X_1, \dots, X_n be IID with PDF $f(x; \theta)$.

The **likelihood function** is:

$$\mathcal{L}_n(\theta) = \prod_{i=1}^n f(X_i; \theta)$$

For both mathematical and numerical convenience, in practice we often work with the **log-likelihood function**:

$$\ell_n(\theta) = \log \mathcal{L}_n(\theta) = \sum_{i=1}^n \log f(X_i; \theta)$$

The **Maximum Likelihood Estimator (MLE)** is:

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta \in \Theta} \mathcal{L}_n(\theta) = \arg \max_{\theta \in \Theta} \ell_n(\theta)$$

Note that maximizing the likelihood is the same as maximizing the log-likelihood, since the logarithm is a strictly increasing (monotonic) function.

Multiple Perspectives

Intuitive

Likelihood is a “what if” game. Imagine you have a coin and you flip it 10 times, getting 7 heads. You ask yourself:

“What if the probability of heads were 0.5? How likely would 7 heads in 10 flips be?” “What if the probability were 0.6? Would that make my data more or less likely?” “What if it were 0.7? 0.8?” For each possible value of the parameter (here, the probability of heads), you calculate how probable your exact observed data would be. The parameter value that maximizes this probability is your maximum likelihood estimate.

The likelihood function is this “what if” calculation formalized – it’s the probability (or probability density) of your observed data, treated as a function of the unknown parameters.

Mathematical

Why do we often prefer working with the log-likelihood?

$$\ell_n(\theta) = \log \mathcal{L}_n(\theta) = \sum_{i=1}^n \log f(x_i; \theta)$$

1. **Differentiation:** For [exponential family](#) distributions (normal, exponential, gamma, etc.), log derivatives eliminate exponentials. For example, for normal: $\frac{d}{d\mu} e^{-(x-\mu)^2/2}$ becomes just $(x - \mu)$ after taking logs
2. **Numerical stability:** Products of small probabilities underflow; sums of logs don't
3. **Additive structure:** Log-likelihood is a sum over observations, making derivatives and optimization more tractable

Computational

Let's visualize the likelihood function for a simple Bernoulli example:

```
import numpy as np
import matplotlib.pyplot as plt

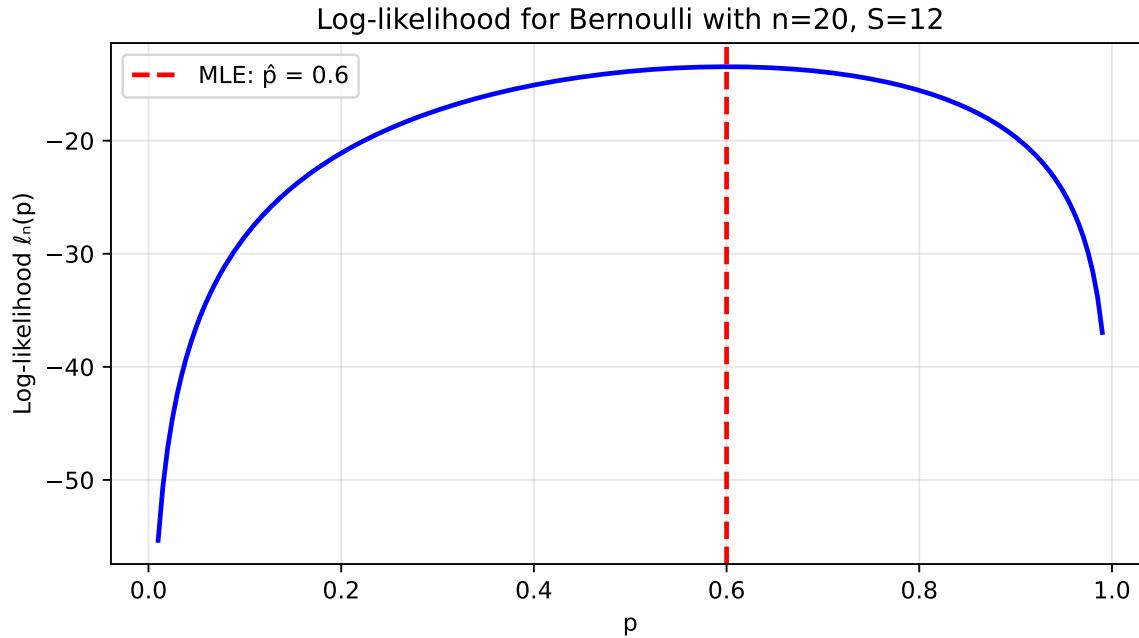
# Simulate coin flips: n=20, observed 12 successes
n = 20
successes = 12

# Define the log-likelihood function
p_values = np.linspace(0.01, 0.99, 200)
log_likelihood = successes * np.log(p_values) + (n - successes) * np.log(1 - p_values)

# Find the MLE (exact analytical solution)
p_mle = successes / n # Exact MLE for Bernoulli

# Create the plot
plt.figure(figsize=(7, 4))
plt.plot(p_values, log_likelihood, 'b-', linewidth=2)
plt.axvline(p_mle, color='red', linestyle='--', linewidth=2, label=f'MLE: \hat{p} = {p_mle}')
plt.xlabel('p')
plt.ylabel('Log-likelihood (p)')
plt.title(f'Log-likelihood for Bernoulli with n={n}, S={successes}')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Maximum likelihood estimate: \hat{p} = {successes/n}")
print(f"This makes intuitive sense: it's simply the observed proportion of successes!")
```



Maximum likelihood estimate: $\hat{p} = 0.6$

This makes intuitive sense: it's simply the observed proportion of successes!

Notice how the log-likelihood is maximized exactly at the observed proportion of successes. This is no coincidence – the MLE often has an intuitive interpretation.

⚠️ Important: Likelihood function is NOT a probability distribution!

The likelihood $\mathcal{L}_n(\theta)$ is NOT a probability distribution over θ . In general:

$$\int_{\Theta} \mathcal{L}_n(\theta) d\theta \neq 1$$

Why? The same mathematical expression $f(x; \theta)$ plays two different roles:

- **As a PDF:** Fix θ , vary $x \rightarrow \int f(x; \theta) dx = 1$ (proper probability distribution)
- **As a likelihood:** Fix x (observed data), vary $\theta \rightarrow \int \mathcal{L}_n(\theta) d\theta$ is usually not 1

Example: Observe one coin flip with result $X = 1$ (heads) from $\text{Bernoulli}(p)$:

- The likelihood is $\mathcal{L}(p) = p$ for $p \in [0, 1]$
- $\int_0^1 p dp = \frac{1}{2} \neq 1$

The likelihood tells us relative plausibility of parameter values, not their probabilities. This is why we need Bayesian methods if we want actual probability distributions over parameters!

5.5.3 Finding the MLE Analytically

For simple models, we can find the MLE by taking derivatives and setting them to zero. The general recipe:

1. Write down the likelihood $\mathcal{L}_n(\theta)$
2. Take the logarithm to get $\ell_n(\theta)$ (this simplifies products to sums)
3. Take the derivative with respect to θ
4. Set equal to zero and solve
5. Verify it's a maximum (not a minimum or saddle point)

i Simplifying MLE Calculations

When finding the MLE, multiplying the likelihood (or log-likelihood) by a positive constant or adding a constant doesn't change where the maximum occurs. This means we can often ignore:

- Normalization constants that don't depend on θ
- Terms like $\frac{1}{(2\pi)^{n/2}}$ in the normal distribution
- Factorials in discrete distributions

This greatly simplifies calculations – focus only on terms involving θ !

Let's work through some examples:

Example: Bernoulli Distribution

For $X_1, \dots, X_n \sim \text{Bernoulli}(p)$:

Step 1: The likelihood is

$$\mathcal{L}_n(p) = \prod_{i=1}^n p^{X_i} (1-p)^{1-X_i} = p^S (1-p)^{n-S}$$

where $S = \sum_{i=1}^n X_i$ is the total number of successes.

Step 2: The log-likelihood is

$$\ell_n(p) = S \log p + (n - S) \log(1 - p)$$

Step 3: Taking the derivative:

$$\frac{d\ell_n}{dp} = \frac{S}{p} - \frac{n - S}{1 - p}$$

Step 4: Setting to zero and solving:

$$\frac{S}{p} = \frac{n - S}{1 - p} \implies S(1 - p) = (n - S)p \implies S = np$$

Therefore, $\hat{p}_{\text{MLE}} = S/n = \bar{X}_n$.

Note: This is the same as the Method of Moments estimator!

Example: Normal Distribution

For $X_1, \dots, X_n \sim \mathcal{N}(\mu, \sigma^2)$:

The log-likelihood (ignoring constants) is:

$$\ell_n(\mu, \sigma) = -n \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^n (X_i - \mu)^2$$

Taking partial derivatives and setting to zero:

$$\frac{\partial \ell_n}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^n (X_i - \mu) = 0$$

$$\frac{\partial \ell_n}{\partial \sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^n (X_i - \mu)^2 = 0$$

Solving these equations:

- $\hat{\mu}_{\text{MLE}} = \bar{X}_n$

- $\hat{\sigma}_{MLE}^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2$

Again, these match the Method of Moments estimators!

Example: A Harder Case - Uniform(0, θ)

Not all MLEs can be found by differentiation! Consider $X_1, \dots, X_n \sim \text{Uniform}(0, \theta)$.

The PDF is:

$$f(x; \theta) = \begin{cases} 1/\theta & \text{if } 0 \leq x \leq \theta \\ 0 & \text{otherwise} \end{cases}$$

The likelihood is:

$$\mathcal{L}_n(\theta) = \begin{cases} (1/\theta)^n & \text{if } \theta \geq \max\{X_1, \dots, X_n\} \\ 0 & \text{if } \theta < \max\{X_1, \dots, X_n\} \end{cases}$$

This function:

- Is 0 when θ is less than the largest observation
- Decreases as $(1/\theta)^n$ for larger θ

Therefore, the likelihood is maximized at the boundary: $\hat{\theta}_{MLE} = X_{(n)} = \max\{X_1, \dots, X_n\}$.

This example shows that not all optimization problems are solved by calculus – sometimes we need to think more carefully about the function's behavior!

Let's visualize this unusual likelihood function:

```
# Uniform(0, ) MLE visualization
np.random.seed(42)
n = 10
true_theta = 2.0
data = np.random.uniform(0, true_theta, n)
x_max = np.max(data)

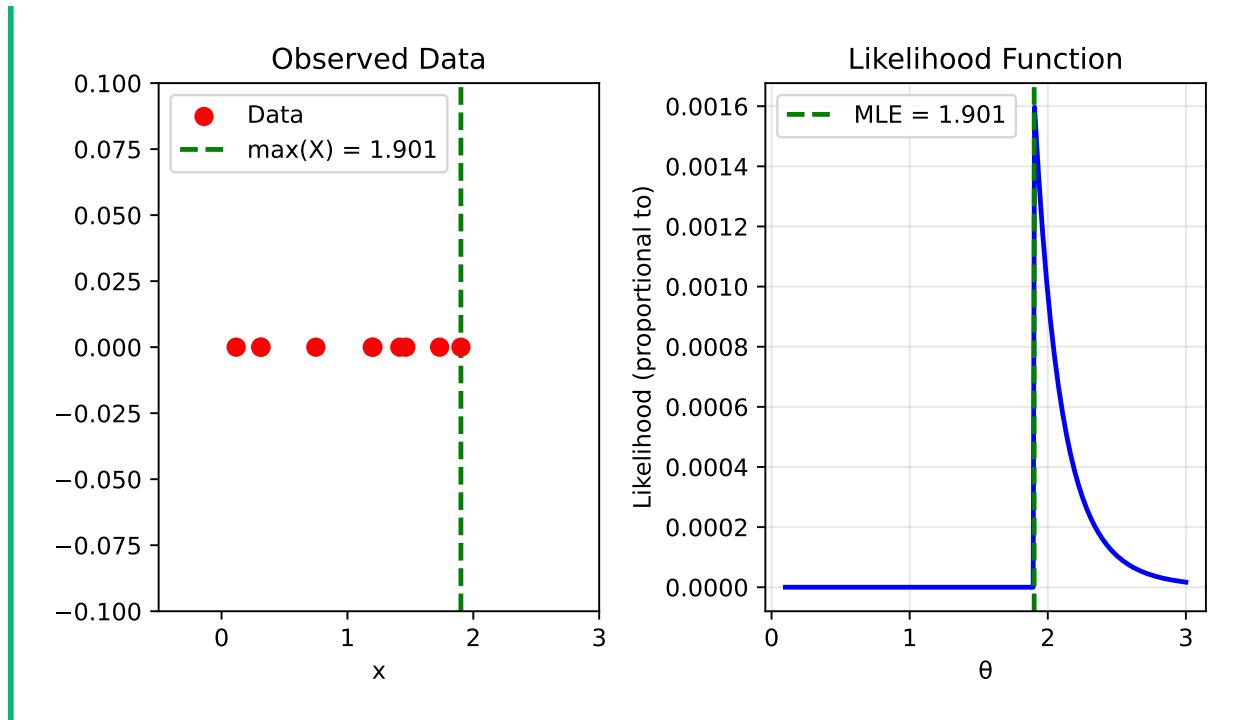
# Create theta values
theta_values = np.linspace(0.1, 3.0, 300)
likelihood = np.zeros_like(theta_values)

# Calculate likelihood (proportional to)
for i, theta in enumerate(theta_values):
    if theta >= x_max:
        likelihood[i] = (1/theta)**n
    else:
        likelihood[i] = 0

plt.figure(figsize=(7, 4))
plt.subplot(1, 2, 1)
# Plot data points
plt.scatter(data, np.zeros_like(data), color='red', s=50, zorder=5, label='Data')
plt.axvline(x_max, color='green', linestyle='--', linewidth=2, label=f'max(X) = {x_max:.3f}')
plt.xlim(-0.5, 3.0)
plt.ylim(-0.1, 0.1)
plt.xlabel('x')
plt.title('Observed Data')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(theta_values, likelihood, 'b-', linewidth=2)
plt.axvline(x_max, color='green', linestyle='--', linewidth=2, label=f'MLE = {x_max:.3f}')
plt.xlabel(' ')
plt.ylabel('Likelihood (proportional to)')
plt.title('Likelihood Function')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



5.6 MLE Via Numerical Optimization

Finding the maximum of the log-likelihood by taking a derivative and setting it to zero is clean and satisfying, but it only works for the simplest models. For most real-world problems, the log-likelihood function is a complex, high-dimensional surface, and we cannot find the peak analytically.

Instead, we must turn to **numerical optimization**. The core idea of most optimization algorithms is simple: we start with an initial guess for the parameters, θ_0 , and then iteratively take steps “uphill” on the likelihood surface until we can no longer find a higher point.

This section explores the concepts and tools behind this fundamental process.¹

i The Need for Numerical Methods

For many important models, we cannot solve for the MLE analytically:

- **Logistic regression:** No closed-form solution for the regression coefficients
- **Mixture models:** Complex likelihood with multiple local maxima
- **Most complex ML models:** Neural networks, random forests, etc.

We must use iterative numerical optimization algorithms to find the maximum of the likelihood function (or minimum of the negative log-likelihood).

5.6.1 The Optimization Setup for MLE

Finding the MLE requires solving an optimization problem to find the parameters θ that maximize the likelihood function $\mathcal{L}_n(\theta)$ or equivalently the log-likelihood function $\ell_n(\theta)$.

Since optimization methods in software libraries are conventionally written to perform **minimization**, our practical goal becomes:

¹In the next chapter, we’ll explore a special optimization algorithm for MLE called the EM (Expectation-Maximization) algorithm, which is particularly useful for models with latent variables or missing data.

$$\hat{\theta}_{\text{MLE}} = \arg \min_{\theta} [-\ell_n(\theta)]$$

This is a crucial point: we minimize the **negative** log-likelihood. Forgetting this minus sign is one of the most common programming errors in statistical computing!

Throughout this section, we'll follow the standard convention and present algorithms for minimization. We'll use examples from Python's `scipy.optimize`, but other languages and libraries (R's `optim`, Julia's `Optim.jl`, etc.) offer similar algorithms with comparable interfaces.

When in optimization we say that a problem is D -dimensional, we refer to the number of variables being optimized over – here, the number of elements of θ .

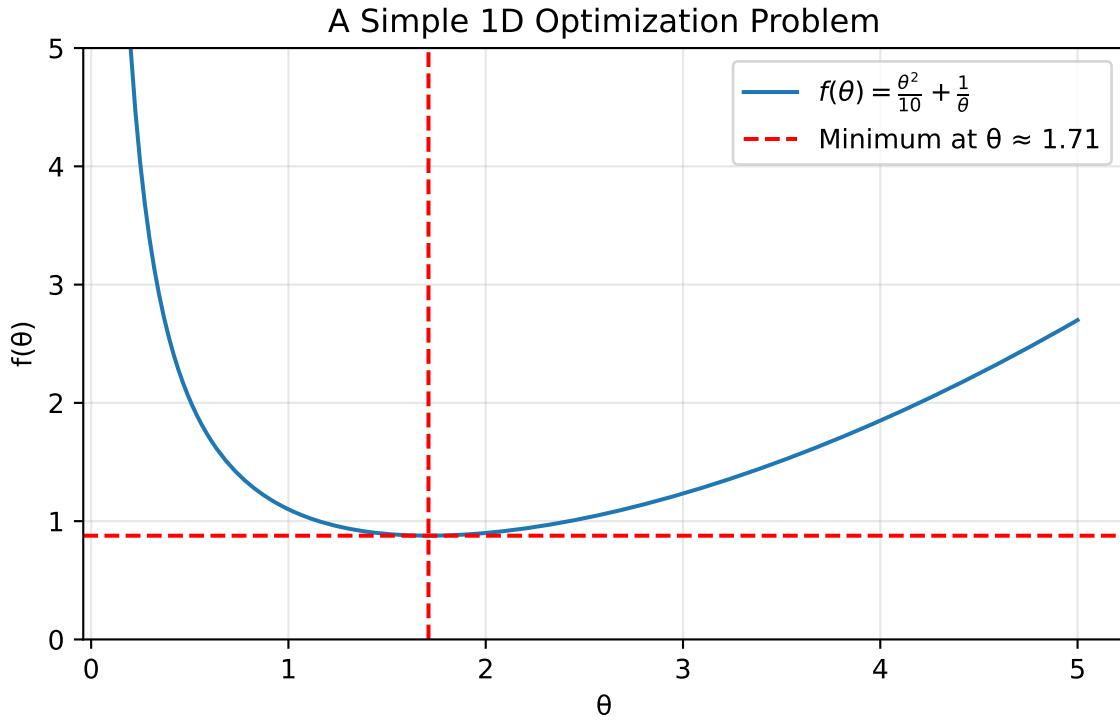
5.6.2 Numerical Optimization in 1D

To understand how numerical optimization works, it's helpful to start with the simplest case: finding the minimum of a function with a single parameter, $f(\theta)$.

Let's take a simple example function to minimize:

$$f(\theta) = \frac{\theta^2}{10} + \frac{1}{\theta}; \quad \theta > 0$$

whose global minimum is at $\theta^* \approx 1.71$.



How can a computer find the minimum of this function? There are two main families of approaches.

5.6.2.1 1D Optimization without Derivatives

The simplest algorithms work by bracketing the minimum. If you can find three points (a, c, b) such that $a < c < b$ and $f(c) < f(a)$ and $f(c) < f(b)$, you know a minimum lies somewhere in the interval (a, b) .

- **Golden-section search:** A simple but robust method that is analogous to binary search for finding a root. It progressively narrows the bracket until the desired precision is reached.
- **Brent's method:** A more sophisticated and generally preferred method. It combines the guaranteed (but slow) progress of golden-section search with the potentially faster convergence of fitting a parabola to the three points and jumping to the minimum of the parabola.

These methods are useful when the function's derivative is unavailable or unreliable.

5.6.2.2 1D Optimization with Derivatives: Newton's Method

If we can compute derivatives, we can often find the minimum much faster. The most famous derivative-based method is **Newton's method** (or Newton-Raphson).

The idea is to iteratively approximate the function with a quadratic and jump to the minimum of that quadratic. A second-order Taylor expansion of $f(\theta)$ around a point θ_t is:

$$f(\theta) \approx f(\theta_t) + f'(\theta_t)(\theta - \theta_t) + \frac{1}{2}f''(\theta_t)(\theta - \theta_t)^2.$$

To find the minimum of this quadratic approximation, we take its derivative with respect to θ and set it to zero:

$$\frac{d}{d\theta}(\text{approx}) = f'(\theta_t) + f''(\theta_t)(\theta - \theta_t) = 0$$

Solving for θ gives us the next point in our iteration, θ_{t+1} :

$$\theta_{t+1} = \theta_t - \frac{f'(\theta_t)}{f''(\theta_t)}.$$

This simple update rule is the core of Newton's method. When close to a well-behaved minimum, it converges very rapidly (quadratically). However, it can be unstable if the function is not “nice” or if the starting point is far from the minimum.

Example: 1D Optimization in `scipy`

Let's find the minimum of our example function using `scipy.optimize`. We'll use Brent's method, which only requires the function itself and a bracket.

```
import numpy as np
import scipy.optimize

def f(theta):
    return theta**2/10 + 1/theta

# Find the minimum using Brent's method
# We need to provide a bracket (a, b) or (a, c, b)
# where the minimum is expected to lie.
# Let's use (0.1, 10.0)
result = scipy.optimize.brent(f, brack=(0.1, 10.0))

print(f"The minimum found by Brent's method is at  = {result:.6f}")
print(f"The value of the function at the minimum is f() = {f(result):.6f}")

# Analytical minimum is at (5)^(1/3)
analytical_min = 5**(1/3)
print(f"The analytical minimum is at  = {analytical_min:.6f}")
```

The minimum found by Brent's method is at $\theta = -0.000000$
 The value of the function at the minimum is $f(\theta) = -456934496164.231018$
 The analytical minimum is at $\theta = 1.709976$
 The output shows that the numerical method finds the correct minimum with high precision.

5.6.2.3 Moving to Multiple Dimensions

The concepts from 1D optimization extend to the multi-dimensional case, but with added complexity.

Derivative-free methods:

Generalizing derivative-free methods to multiple dimensions, i.e., optimizing $f(\theta_1, \dots, \theta_n)$ with respect to $\theta_1, \dots, \theta_n$ is hard! As the dimensionality increases, the number of possible directions that need to be searched for the optimum grows very rapidly.

- **Classical approaches:** Very old algorithms like [direct search](#) methods are mostly inefficient and struggle as dimensionality grows. Searching in many directions at once becomes prohibitively expensive.
- **Modern approaches:** Recent methods can be more effective in specific scenarios:
 - [Bayesian optimization](#): Works well in low dimensions (typically < 20) and can be very sample-efficient when function evaluations are expensive
 - [CMA-ES](#) (Covariance Matrix Adaptation Evolution Strategy): If you can afford many function evaluations and don't have gradients, this method can work in surprisingly high dimensions – up to tens of thousands of parameters
- These methods are mainly used when gradients are unavailable or unreliable (e.g., noisy simulations, black-box models)

Derivative-based methods:

- **The standard choice:** When gradients are available and the function is relatively inexpensive to evaluate, gradient-based methods dominate
- **Scalability:** These methods can handle problems with millions or even billions of parameters (think deep neural networks)
- **Key transformation:** The first derivative (slope) becomes the **gradient** (∇f), and the second derivative (curvature) becomes the **Hessian matrix** (H)
- **Why they work:** Gradient information provides a principled direction for improvement at each step, making the search much more efficient than blind exploration

You can find visualizations of several optimization algorithms at work [here](#).

5.6.3 The Gradient: The Key to Multidimensional Optimization

The main tool for optimization of multidimensional functions

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

is the **gradient**:

$$\nabla f(\theta) = \left(\frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_n} \right)^T$$

Here $\frac{\partial f}{\partial \theta_i}$ is the **partial derivative** of f with respect to θ_i . It can be evaluated by computing the derivative w.r.t. θ_i while keeping the other variables constant.

Geometric interpretation: The gradient points in the direction where the function values grow fastest. Its magnitude measures the rate of change in that direction.

i Intuition: The Gradient as a Compass

Think of finding the MLE as hiking blindfolded on a hilly terrain (the negative log-likelihood surface). At each point, the gradient tells you which direction is steepest uphill. Since we want to minimize, we go in the opposite direction – downhill.

Let's visualize this concept:

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a 2D function with interesting topology (negative log-likelihood surface)
def f(x, y):
    """Example negative log-likelihood with two local minima"""
    return -2 * np.exp(-(x-1)**2 - y**2) + 3 * np.exp(-(x+1)**2 - y**2) + 0.1*(x**2 + y**2)

# Create grid
x = np.linspace(-3, 3, 50)
y = np.linspace(-2, 2, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Create figure with two subplots
fig = plt.figure(figsize=(7, 10))

# First subplot: 3D surface
ax1 = fig.add_subplot(2, 1, 1, projection='3d')
surf = ax1.plot_surface(X, Y, Z, cmap='viridis', alpha=0.8, edgecolor='none')
ax1.contour(X, Y, Z, levels=15, cmap='viridis', offset=Z.min(), alpha=0.4)

# Mark the global minimum
ax1.scatter([1], [0], [f(1, 0)], color='black', s=100, marker='*', label='Global minimum')

ax1.set_xlabel(' ')
ax1.set_ylabel(' ')
ax1.set_zlabel('Negative log-likelihood')
ax1.set_title('3D Optimization Landscape')
# Adjust viewing angle to swap visual perspective
ax1.view_init(elev=30, azim=-45)

# Second subplot: 2D contour with gradient field
ax2 = fig.add_subplot(2, 1, 2)

# Compute gradient (numerically)
dx = 0.01
dy = 0.01
grad_x = (f(X + dx, Y) - f(X - dx, Y)) / (2 * dx)
grad_y = (f(X, Y + dy) - f(X, Y - dy)) / (2 * dy)

# Normalize gradient vectors for better visualization
grad_norm = np.sqrt(grad_x**2 + grad_y**2)

```

```
grad_x_norm = -grad_x / (grad_norm + 1e-10) # Negative for descent
grad_y_norm = -grad_y / (grad_norm + 1e-10)

# Contour plot
contour = ax2.contour(X, Y, Z, levels=15, cmap='viridis', alpha=0.6)
ax2.clabel(contour, inline=True, fontsize=8)

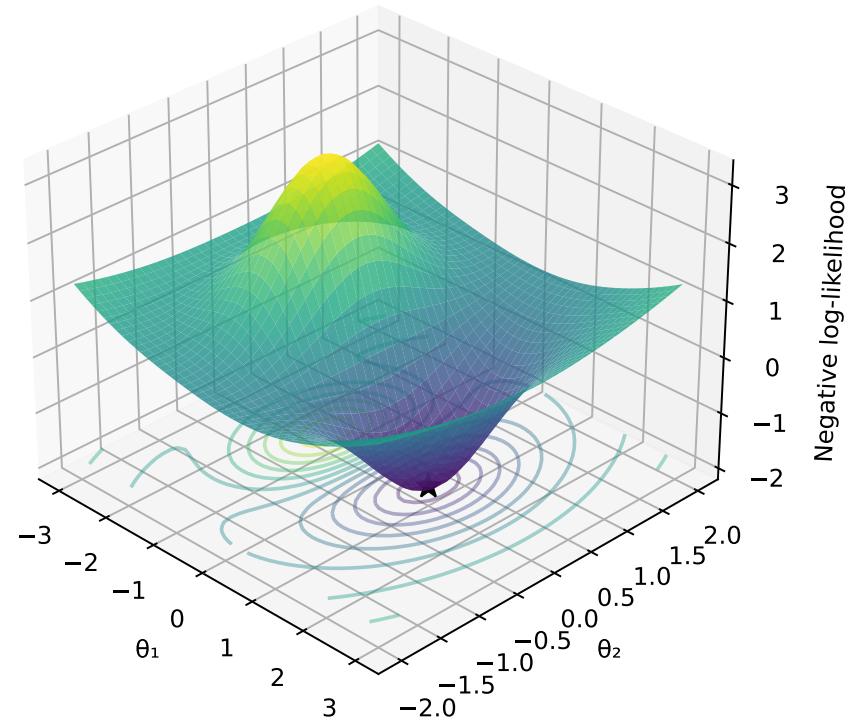
# Gradient vectors
skip = 3 # Show every 3rd arrow for clarity
ax2.quiver(X[::skip, ::skip], Y[::skip, ::skip],
            grad_x_norm[::skip, ::skip], grad_y_norm[::skip, ::skip],
            scale=20, alpha=0.7, width=0.003, color='red')

# Mark global minimum
ax2.plot([1], [0], 'k*', markersize=10, label='Global minimum')

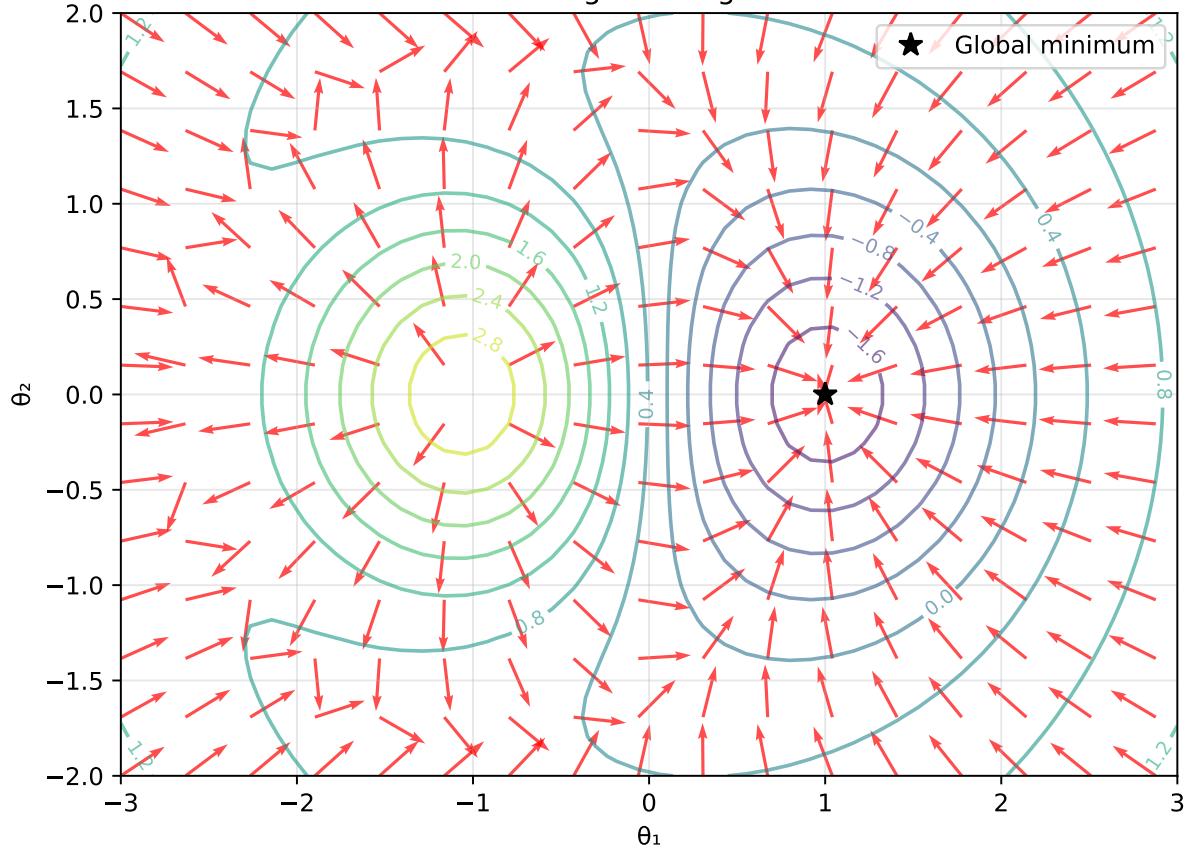
ax2.set_xlabel(' ')
ax2.set_ylabel(' ')
ax2.set_title('Gradient Field on Negative Log-Likelihood Surface')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

3D Optimization Landscape



Gradient Field on Negative Log-Likelihood Surface



These visualizations show:

- **Top (3D)**: The optimization landscape as a surface, showing the “hills and valleys” that optimization algorithms must navigate
- **Bottom (2D)**: The same landscape from above with:
 - **Contour lines** showing level sets of the negative log-likelihood
 - **Red arrows** pointing in the direction of steepest descent (negative gradient)
 - **Black star** marking the global minimum at $\theta_1 = 1, \theta_2 = 0$
- Note how the gradient vectors always point toward lower values, guiding optimization algorithms down-hill

5.6.4 Evaluating the Gradient

The first practical challenge in using gradients for optimization is how to evaluate the gradient. There are three main approaches:

1. **Analytically**: Derive by hand and implement – fast but error-prone
2. **Finite differences**: Approximate numerically by evaluating nearby points – easy but expensive² and less precise
3. **Automatic differentiation (autodiff)**: Best of both worlds – exact, fast and automatic

Modern frameworks like [PyTorch](#) and [JAX](#) make automatic differentiation the preferred choice. However, some older languages such as R still lack full autodiff support. Gradient computation usually uses **backward-mode autodiff**, which is suitable for computing derivatives of real-valued functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

Example: Gradients in Python with JAX

Here's how to compute gradients automatically using JAX:

```
import jax.numpy as jnp
import jax

def f(theta):
    return theta**2/10 + 1/theta

g = jax.grad(f)
g(1.5) # Evaluating at a scalar value

# For multiple variables:
def f2(theta):
    return (jnp.exp(-(theta[0]-1)**2 - theta[1]**2)
           - jnp.exp(-(theta[0]+1)**2 - theta[1]**2))

g2 = jax.grad(f2)
g2(jnp.array([-3.0, 0.5]))
```

5.6.5 Gradient-Based Optimization Methods

Once we have gradients, we can use various algorithms to find the optimum.

The simplest gradient-based algorithm is **gradient descent** (also called **steepest descent**), which iteratively takes steps in the direction of the negative gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

²Generally, evaluating a gradient in D dimensions with finite differences will need at least $D + 1$ evaluations.

where $\eta > 0$ (eta) is the *learning rate*, which defines the step length.

Gradient descent is inefficient, often zigzagging toward the optimum, but it's the foundation for understanding more sophisticated methods.

Advanced Gradient-Based Methods

Beyond basic gradient descent, there are many sophisticated optimization algorithms:

Classical improvements:

- **Conjugate gradient methods:** Choose search directions that are conjugate with respect to the Hessian matrix. For quadratic functions, this guarantees finding the exact minimum in at most n steps (where n is the number of dimensions). Unlike gradient descent which can take tiny steps in nearly orthogonal directions, conjugate gradient methods take larger, more efficient steps.
- **Quasi-Newton methods:** Approximate the Hessian matrix without computing second derivatives directly.
 - **BFGS** (Broyden-Fletcher-Goldfarb-Shanno): Builds up an approximation to the inverse Hessian using gradient information from previous steps
 - **L-BFGS** (Limited-memory BFGS): Stores only a few vectors instead of the full Hessian approximation, making it practical for high-dimensional problems

Modern methods:

- **Momentum methods** (heavy ball method): Add a fraction of the previous step to the current gradient step:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t) + \beta(\theta_t - \theta_{t-1})$$

where $\beta \in [0, 1]$ is the momentum coefficient. This helps the optimizer “roll through” small local variations and reduces zigzagging in narrow valleys.

- **Accelerated gradient methods:** Achieve provably faster convergence rates. Nesterov’s accelerated gradient has convergence rate $O(1/t^2)$ compared to $O(1/t)$ for standard gradient descent on convex functions.

These methods are especially important in deep learning where simple gradient descent would be too slow to navigate the complex, high-dimensional loss landscapes of neural networks.

5.6.6 Stochastic Gradient Methods

Many machine learning problems, especially in deep learning, involve optimizing functions of the form:

$$\min_{\theta} f(X, \theta) = \sum_{i=1}^n f(x_i, \theta)$$

When n is large and θ is high-dimensional, evaluating the full gradient becomes computationally prohibitive.

Stochastic gradient descent (SGD) approximates the gradient using a random subset of data:

$$\theta_{n+1} = \theta_n - \eta_n \nabla \sum_{i \in S_n} f(x_i, \theta_n)$$

where $S_n \subset \{1, \dots, n\}$ is a randomly selected mini-batch.

Convergence: SGD converges for well-behaved functions when the learning rate sequence satisfies:

$$\sum_{i=1}^{\infty} \eta_i = \infty, \quad \sum_{i=1}^{\infty} \eta_i^2 < \infty$$

Note that constant learning rates don't guarantee convergence to the exact optimum!

i Popular SGD Variants

The huge popularity of SGD has spawned many improved variants:

- **Adam** (Adaptive Moment Estimation) (Kingma and Ba 2015): Combines momentum with adaptive learning rates for each parameter
- **AdaGrad** (Adaptive Gradient): Adapts learning rate based on historical gradients - parameters with frequent updates get smaller learning rates
- **RMSprop** (Root Mean Square Propagation): Uses a running average of recent gradients to normalize the learning rate

5.6.7 Which Optimizer Should I Use?

For smaller datasets (< 10K observations):

- **L-BFGS** is usually the best first choice
- Fast convergence, reliable for smooth problems
- Standard choice in traditional statistical software

For large datasets or deep learning:

- **SGD** and variants (especially **Adam**) are often the only practical choice
- Require careful tuning of learning rates
- Can handle millions and even billions of parameters

For black-box problems (no gradients available):

- **CMA-ES**: Principled “population-based” method that can handle up to thousands of parameters (requiring many evaluations)
- **Bayesian Optimization**: Efficient for expensive functions with $D < 20$ parameters
- **BADS** (Bayesian Adaptive Direct Search): Combines Bayesian optimization with mesh adaptive direct search, good for noisy and mildly expensive functions with $D < 20$

i Advanced Topic: Constrained Optimization

Often parameters have natural constraints:

- Standard deviations must be positive: $\sigma > 0$
- Probabilities must be in $[0,1]$: $0 \leq p \leq 1$
- Correlation matrices must be positive definite

How do we enforce these? There are typically three different approaches:

1. **Reparameterization**: Optimize $\phi = \log \sigma$ and then use $\sigma = e^\phi$ to ensure positivity³
2. **Constrained optimization**: Use algorithms like L-BFGS-B that allows you to specify parameter bounds
3. **Barrier methods**: Add penalty terms that blow up at boundaries (can be unstable – not recommended)

i Example: 2D MLE with Numerical Optimization

Now let's apply the multidimensional optimization concepts to a real statistical problem. We'll estimate both parameters of a Gamma distribution, which provides an excellent illustration of:

- **True 2D optimization**: Unlike our 1D examples, we need to simultaneously find the shape parameter α and scale parameter β .
- **Constrained optimization in practice**: Both parameters must be positive, so we'll use L-BFGS-B with bounds.

³For numerical stability, it has become common to use the `softplus` instead of `exp` to ensure positivity.

- **The importance of starting values:** We'll launch optimization from multiple starting points to see if they converge to the same solution. We choose the first starting point using the Method of Moments, while the others arbitrarily (could have been random).
- **Visualizing the likelihood surface:** We'll create both 3D and contour plots to understand the optimization landscape.

The Gamma distribution is particularly interesting because its two parameters interact in the likelihood function, creating a curved valley in the likelihood surface rather than a simple bowl.

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats, optimize
from mpl_toolkits.mplot3d import Axes3D

# Generate data from Gamma distribution
np.random.seed(42)
true_alpha = 3.0 # shape parameter
true_beta = 2.0 # scale parameter
n = 100
data = stats.gamma.rvs(a=true_alpha, scale=true_beta, size=n)

# Define negative log-likelihood for Gamma( , )
def neg_log_likelihood(params, data):
    """Negative log-likelihood for Gamma distribution"""
    alpha, beta = params
    if alpha <= 0 or beta <= 0:
        return np.inf # Return infinity for invalid parameters
    return -np.sum(stats.gamma.logpdf(data, a=alpha, scale=beta))

# Method of Moments starting values
sample_mean = np.mean(data)
sample_var = np.var(data)
mom_beta = sample_var / sample_mean
mom_alpha = sample_mean / mom_beta

# Try optimization from different starting points
starting_points = [
    [mom_alpha, mom_beta], # MoM estimate
    [1.0, 1.0],           # Generic start
    [5.0, 5.0],           # Far from truth
]
]

# Store optimization paths
paths = []
results = []

for i, start in enumerate(starting_points):
    # Create a list to store the path for this optimization
    path = []

    # Callback function to record each iteration
    def callback(xk):
        path.append(xk.copy())

    result = optimize.minimize(
        fun=neg_log_likelihood,
        x0=start,
        args=(data,),
        method='L-BFGS-B',
        bounds=[(0.01, None), (0.01, None)], # Enforce positivity
        callback=callback
    )

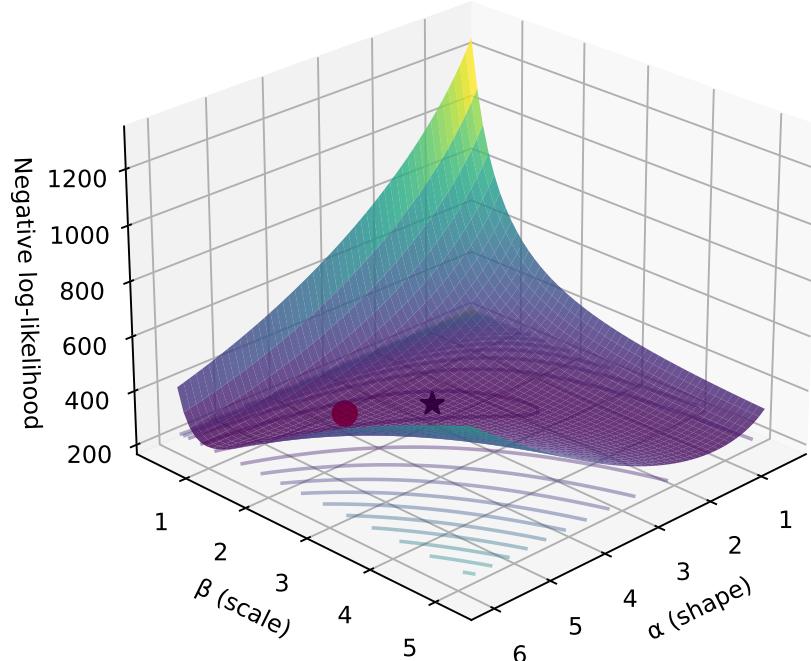
    # Add starting point to path
    full_path = [np.array(start)] + path
    paths.append(np.array(full_path))
    results.append(result)

```

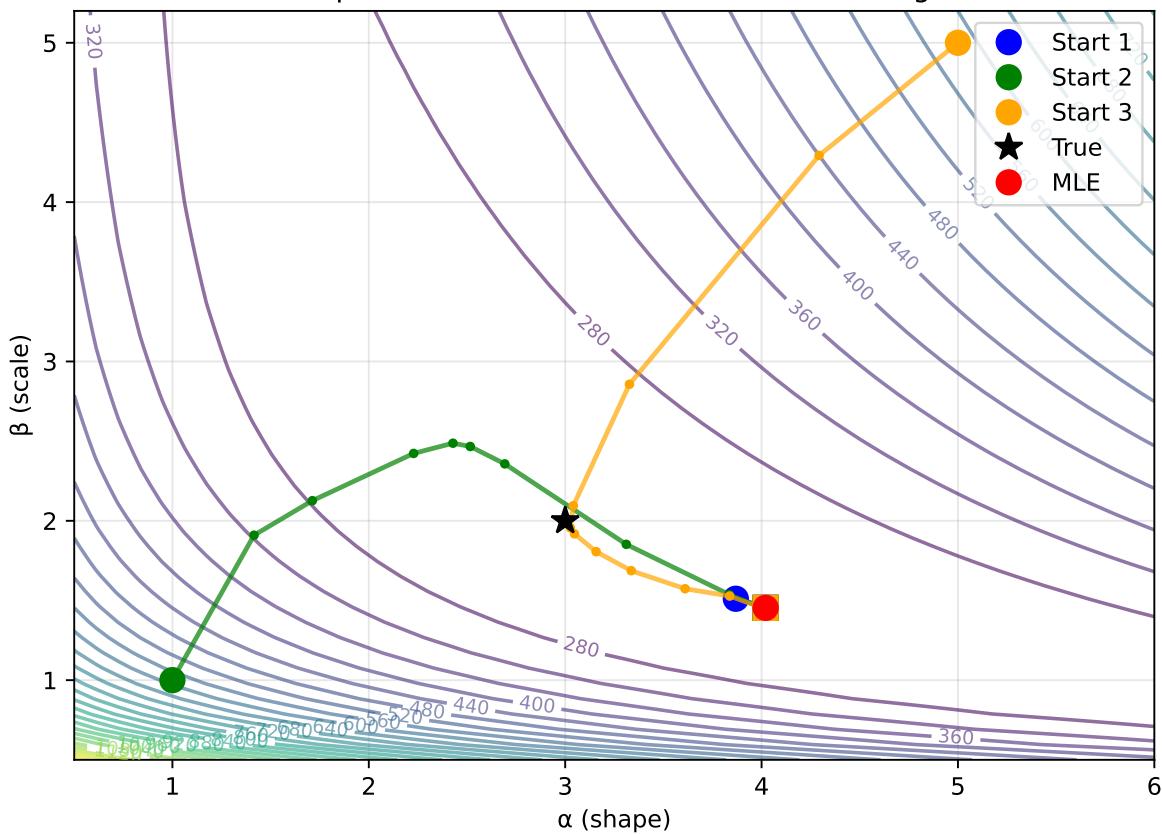
```
Start 1: [3.868907522031454, 1.510860197180884] → MLE: [4.021, 1.454], NLL: 240.066, Iterations: 6
Start 2: [1.0, 1.0] → MLE: [4.021, 1.454], NLL: 240.066, Iterations: 14
Start 3: [5.0, 5.0] → MLE: [4.021, 1.454], NLL: 240.066, Iterations: 13

True parameters:   = 3.0,   = 2.0
MLE estimates:     = 4.021,   = 1.454
MoM estimates:     = 3.869,   = 1.511
```

Likelihood Surface for Gamma Distribution



L-BFGS Optimization Traces from Different Starting Points



Notice how all three starting points converged to the same MLE, demonstrating that this likelihood surface is well-behaved with a single global optimum. The actual L-BFGS traces (with intermediate points marked as dots) reveal interesting optimization behavior:

- **Blue path (MoM start):** Converges in very few iterations since it starts close to the optimum
- **Green path (generic start):** Takes a curved path following the likelihood valley
- **Orange path (far start):** Makes larger initial steps, then follows the contours more carefully as it approaches the optimum

The 3D plot reveals the characteristic curved valley of the Gamma likelihood, explaining why the optimization paths curve rather than taking straight lines to the optimum.

5.6.8 FAQ: Common Issues in Numerical Optimization

i Frequently Asked Questions

Q: I used a minimizer, but I'm supposed to be doing *maximum* likelihood. What gives?

A: A classic source of confusion! Maximizing a function $f(x)$ is equivalent to minimizing its negative $-f(x)$. All standard optimization libraries are built as minimizers. Therefore, in practice, we always find the MLE by **minimizing the negative log-likelihood function**.

Q: I tried different starting points and the optimizer gave different answers. Is it broken?

A: No, this is expected behavior! The algorithms we use are **local optimizers**. They find the nearest valley (local minimum), but they have no guarantee of finding the deepest valley (global minimum). If your likelihood surface has multiple local optima, the result will depend on your starting point. This is why using a good initial value (like the Method of Moments estimate!) is so important.

Q: How do I know if the algorithm has actually converged to the right answer?

A: In general, you don't know with 100% certainty. Optimizers use heuristics, like stopping when the change in the parameter values or the likelihood is very small. Good practices include:

- **Always try different starting points!**
- Check the optimizer's status messages
- Verify that the gradient is near zero at the solution
- Compare with simpler methods (like MoM) as a sanity check

Let's demonstrate the importance of starting values:

```
# Example: Multiple local optima in a mixture model
# We'll create a bimodal likelihood surface

def bimodal_nll(theta):
    """A negative log-likelihood with two local minima"""
    # Artificial example with two valleys
    return -np.log(0.6 * np.exp(-2*(theta-1)**2) +
                  0.4 * np.exp(-3*(theta-4)**2) + 0.01)

# Try optimization from different starting points
theta_range = np.linspace(-2, 7, 200)
nll_surface = [bimodal_nll(t) for t in theta_range]

starting_points = [-1, 2.5, 5]
colors = ['red', 'green', 'blue']
results = []

plt.figure(figsize=(7, 4))
```

```

plt.plot(theta_range, nll_surface, 'k-', linewidth=2, label='Objective function')

for start, color in zip(starting_points, colors):
    result = optimize.minimize(bimodal_nll, x0=[start], method='L-BFGS-B')
    results.append(result.x[0])
    plt.scatter([start], [bimodal_nll(start)], color=color, s=100,
               marker='o', label=f'Start: {start:.1f}')
    plt.scatter([result.x[0]], [result.fun], color=color, s=100,
               marker='*', label=f'End: {result.x[0]:.2f}')

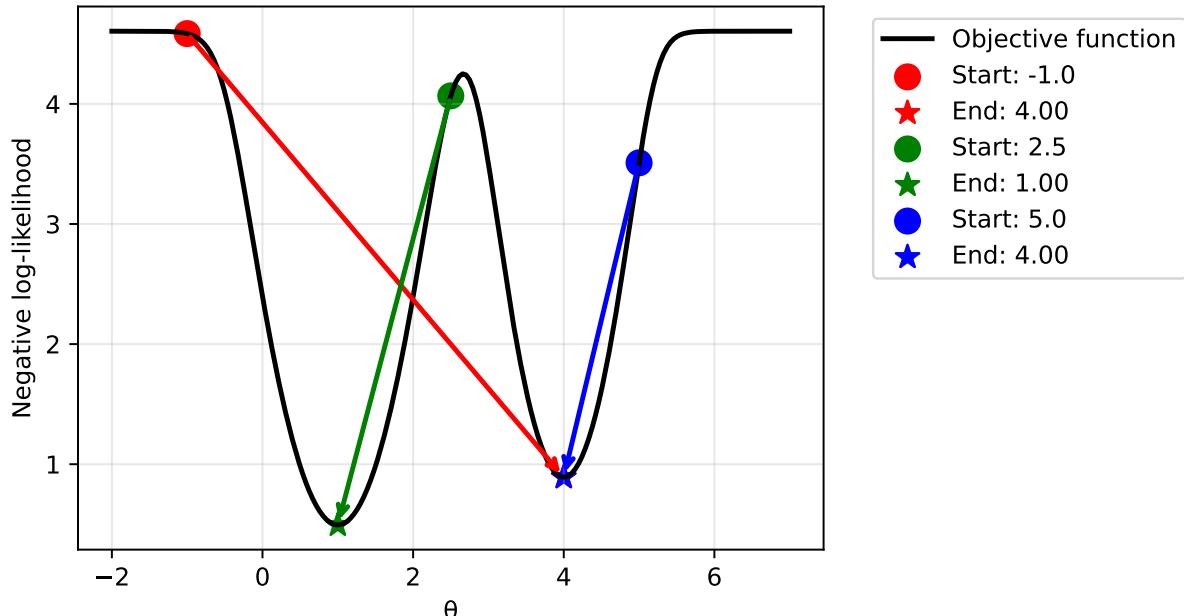
# Draw arrow from start to end
plt.annotate('', xy=(result.x[0], result.fun),
             xytext=(start, bimodal_nll(start)),
             arrowprops=dict(arrowstyle='->', color=color, lw=2))

plt.xlabel(' ')
plt.ylabel('Negative log-likelihood')
plt.title('Local Optimization: Different Starting Points → Different Solutions')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print("Starting points and their corresponding local optima:")
for start, end in zip(starting_points, results):
    print(f" Start: {start:.4f} → End: {end:.3f}")

```

Local Optimization: Different Starting Points → Different Solutions



Starting points and their corresponding local optima:

Start: -1.0 → End: 4.000
 Start: 2.5 → End: 1.000
 Start: 5.0 → End: 4.000

5.7 Chapter Summary and Connections

5.7.1 Key Concepts Review

We've explored two fundamental approaches to finding estimators in parametric models:

Parametric Models:

- Assume data comes from a specific family of distributions $\mathfrak{F} = \{f(x; \theta) : \theta \in \Theta\}$
- Our job reduces to estimating the finite-dimensional parameter θ
- Often have parameters of interest and nuisance parameters

Method of Moments (MoM):

- Match theoretical moments $\mathbb{E}(X^j)$ with sample moments $\frac{1}{n} \sum X_i^j$
- Simple to compute – just solve algebraic equations
- Consistent and asymptotically normal
- Not efficient, but excellent starting values for other methods

Maximum Likelihood Estimation (MLE):

- Find parameters that make observed data most probable
- Likelihood: $\mathcal{L}_n(\theta) = \prod_{i=1}^n f(X_i; \theta)$
- Often requires numerical optimization
- The gold standard of parametric estimation

Numerical Optimization:

- Most MLEs require iterative algorithms
- Gradient-based methods dominate (L-BFGS for small data, SGD/Adam for large)
- Automatic differentiation (JAX) makes implementation easier
- Local optima are a real concern – **always try multiple starting points!**

5.7.2 The Big Picture

This chapter revealed a fundamental connection: much of modern machine learning is secretly Maximum Likelihood Estimation:

- **Linear regression:** MLE with normal errors
- **Logistic regression:** MLE for Bernoulli responses
- **Neural networks:** MLE with complex parametric models
- **Deep learning:** MLE with stochastic optimization

The principles we've learned – likelihood, optimization, gradients – are the foundation of both classical statistics and modern ML.

5.7.3 Common Pitfalls to Avoid

1. **Confusing likelihood with probability:** The likelihood is NOT a probability distribution over parameters
2. **Forgetting the negative sign:** Optimizers minimize, so use negative log-likelihood
3. **Assuming analytical solutions exist:** Most real problems require numerical methods
4. **Trusting a single optimization run:** Did I mention that you should **always try multiple starting points?**
5. **Ignoring convergence warnings:** Check optimizer status and diagnostics

5.7.4 Chapter Connections

- **Previous:** Chapter 3 gave us convergence concepts and the framework for evaluating estimators. Now we know how to *find* estimators.

- **Next:** Chapter 6 will explore the *properties* of these estimators in detail – bias, variance, efficiency – and prove that MLEs have optimal properties.
- **Bootstrap (Chapter 4):** Provides a computational alternative to analytical standard errors for our estimators
- **Later chapters:** These estimation principles extend to more complex models (regression, time series, etc.)

5.7.5 Self-Test Problems

1. **Method of Moments:** For $X_1, \dots, X_n \sim \text{Uniform}(a, b)$, find the Method of Moments estimators for a and b .

Hint: Use $\mathbb{E}(X) = \frac{a+b}{2}$ and $\mathbb{V}(X) = \frac{(b-a)^2}{12}$.

2. **Maximum Likelihood:** For $X_1, \dots, X_n \sim \text{Poisson}(\lambda)$, find the Maximum Likelihood Estimator for λ .

Hint: The Poisson PMF is $f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$.

3. **Numerical Optimization:** The log-likelihood for logistic regression with one covariate is:

$$\ell(y, x; \beta_0, \beta_1) = \sum_{i=1}^n [y_i(\beta_0 + \beta_1 x_i) - \log(1 + e^{\beta_0 + \beta_1 x_i})]$$

Explain why you cannot find the MLE for (β_0, β_1) analytically and must use numerical optimization.

4. **Comparing Estimators:** For $X_1, \dots, X_n \sim \text{Exponential}(\lambda)$:

- Find the MoM estimator for λ
- Find the MLE for λ
- Are they the same? Why or why not?

5.7.6 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

5.7.7 Connections to Source Material

i Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Introduction: Machine Learning As Statistical Estimation Parametric Models	Expanded motivation from the slides and general context from AoS §9 introduction. AoS §9 (Introduction), AoS §9.1 (Parameter of Interest). The Gamma distribution example is from AoS Example 9.2.
The Method of Moments (MoM) The Principle: Matching Moments MoM Examples (Bernoulli, Normal, Gamma)	AoS §9.2 (Definition 9.3). AoS §9.2 (Examples 9.4, 9.5). The Gamma example is from AoS §9.14 (Exercise 1).

Properties of MoM Estimator	AoS §9.2 (Theorem 9.6).
Maximum Likelihood Estimation (MLE)	
The Principle and Likelihood Function	AoS §9.3 (Definitions 9.7, 9.8).
Finding the MLE Analytically	
General approach & examples (Bernoulli, Normal)	AoS §9.3 (Remark 9.9, Examples 9.10, 9.11).
Harder case: Uniform($0, \infty$)	AoS §9.3 (Example 9.12).
MLE Via Numerical Optimization	Expanded material from the slides and AoS §9.13.4 (Appendix), with a modern ML focus.
The Need for Numerical Methods & Setup	General optimization concepts, related to AoS §9.13.4.
Gradient-Based Optimization	Concepts related to Newton-Raphson from AoS §9.13.4, but expanded with modern methods (SGD, Adam, etc.).
Example: 2D MLE for Gamma distribution	Practical application. The Gamma MoM is from AoS §9.14 (Exercise 1), serving as a starting point.
Chapter Summary and Connections	New summary material.
Self-Test Problems	Problems inspired by AoS §9.14 (Exercises 2a, 5).

5.7.8 Further Materials

- **Bayesian Optimization:** An interactive exploration on [Distill](#) (Agnihotri & Batra; 2020)
- **Classic reference:** Casella & Berger, “Statistical Inference”, Chapter 7
- **Modern perspective:** Efron & Hastie, “Computer Age Statistical Inference”, Chapter 4

Remember: Parametric inference is about making assumptions (choosing a model) and then finding the best parameters within that model. The Method of Moments is simple but not optimal. Maximum Likelihood is the gold standard but often requires numerical optimization. Master these concepts – they’re the foundation of statistical modeling and machine learning!

Chapter 6

Parametric Inference II: Properties of Estimators

6.1 Learning Objectives

After completing this chapter, you will be able to:

- Explain the key properties of the MLE (consistency, equivariance, asymptotic normality, and efficiency) and their practical implications.
- Define Fisher Information and use it to quantify the precision of parameter estimates.
- Construct and interpret confidence intervals for parameters using the MLE and its standard error.
- Apply the Delta Method to find confidence intervals for transformed parameters.
- Implement the EM algorithm for finding MLEs in models with latent variables.

Note

This chapter explores the theoretical properties of Maximum Likelihood Estimators and provides practical tools for statistical inference. The material is adapted from Chapter 9 of Wasserman (2013), with additional computational examples and modern perspectives on optimization for latent variable models.

6.2 Introduction: How Good Are Our Estimators?

In Chapter 5, we learned how to *find* estimators for parametric models using the Method of Moments (MoM) and Maximum Likelihood Estimation (MLE). We saw that finding the MLE often requires numerical optimization, and we explored practical algorithms for this task.

But finding an estimator is only half the story. The next natural question is: **how good are these estimators?**

This chapter addresses this fundamental question by exploring the *properties* of estimators, with a special focus on the MLE. We'll discover why the MLE is considered the “gold standard” of parametric estimation – it turns out to have a remarkable collection of desirable properties that make it optimal in many senses.

Key Questions We'll Answer

- Does our estimator converge to the true value as we get more data? (Consistency)
- How much uncertainty is associated with our estimate? Can we quantify it? (Confidence)

Intervals)

- **Is our estimator the best possible one**, or could we do better? (Efficiency)
- **How do these properties extend** to complex models with multiple parameters or latent variables?

Understanding these properties is crucial for several reasons:

1. **Practical inference:** Knowing that $\hat{\theta}_n = 2.3$ is not very useful without understanding the uncertainty. Is the true value likely between 2.2 and 2.4, or between 1 and 4?
2. **Method selection:** When multiple estimation methods exist, these properties help us choose the best one.
3. **Foundation for advanced methods:** The concepts in this chapter – especially Fisher Information and the EM algorithm – are fundamental to modern statistical methods and machine learning.
4. **Connection to optimization:** We'll see that the same mathematical quantities that determine statistical uncertainty also appear in optimization algorithms.

Let's begin with a concrete example to bridge our previous work on finding MLEs with the new focus on analyzing their properties.

6.3 Warm-up: A Complete MLE Example with Numerical Optimization

Before diving into the theoretical properties, let's work through a complete example that bridges Chapter 5 (finding MLEs) and Chapter 6 (analyzing them). We'll find the MLE for a [Beta distribution](#), which requires numerical optimization.

The **likelihood** for the Beta distribution with parameters $\alpha, \beta > 0$ is:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

The **log-likelihood** for n observations is:

$$\begin{aligned} \ell_n(\alpha, \beta) = & \sum_{i=1}^n \left[\log \Gamma(\alpha + \beta) - \log \Gamma(\alpha) - \log \Gamma(\beta) \right. \\ & \left. + (\alpha - 1) \log(X_i) + (\beta - 1) \log(1 - X_i) \right] \end{aligned}$$

We will implement it below as the negative log-likelihood (`beta_nll`).

```
import numpy as np
import scipy.optimize
import jax
import jax.numpy as jnp
import jax.scipy.special as jsps

# Generate data from a Beta distribution
np.random.seed(43)
x = np.random.beta(1.5, 0.5, size=100)

# Define the negative log-likelihood for Beta( , )
def beta_nll(theta, x):
    """Negative log-likelihood for Beta distribution using JAX"""
    pass
```

```

alpha, beta = theta
# Use log-gamma function for numerical stability
return -jnp.sum(
    jsps.gammaln(alpha + beta)
    - jsps.gammaln(alpha)
    - jsps.gammaln(beta)
    + (alpha - 1) * jnp.log(x)
    + (beta - 1) * jnp.log(1 - x)
)

# Get the gradient function automatically
beta_grad = jax.grad(beta_nll)

# We specify bounds: , > 0
# - Bounds are *inclusive* so we set a small positive number as lower bound
# - None here denotes infinity
bounds = [(0.0001, None), (0.0001, None)]

# Optimize using L-BFGS-B with bounds to ensure positivity
result = scipy.optimize.minimize(
    beta_nll,
    x0=jnp.array([1.0, 1.0]), # Initial guess
    args=(x,), # Tuple of additional arguments to pass to jac (x is data)
    jac=beta_grad,
    method='L-BFGS-B',
    bounds=bounds
)

print(f"Raw optimization result:\n")
print(result)

print(f"\nExtracted results:")
print(f"  MLE: ^ = {result.x[0]:.3f}, ^ = {result.x[1]:.3f}")
print(f"  Negative log-likelihood: {result.fun:.3f}")
print(f"  Converged: {result.success}")
print(f"  Iterations: {result.nit}")

```

Raw optimization result:

```

message: CONVERGENCE: RELATIVE REDUCTION OF F <= FACTR*EPSMCH
success: True
status: 0
fun: -55.24228286743164
x: [ 1.501e+00  4.985e-01]
nit: 10
jac: [-1.030e-04 -9.155e-05]
nfev: 41
njev: 41
hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>

```

Extracted results:

```

MLE: ^ = 1.501, ^ = 0.499
Negative log-likelihood: -55.242
Converged: True

```

Iterations: 10

This example showcases several important points:

1. **Automatic differentiation:** We use JAX to automatically compute gradients, avoiding error-prone manual derivations
2. **Constrained optimization:** The L-BFGS-B method handles the constraint that both parameters must be positive
3. **Numerical stability:** We work with log-gamma functions rather than raw factorials

i The MLE as a Starting Point

Now that we have $\hat{\alpha}$ and $\hat{\beta}$, we can ask the crucial questions:

- How accurate are these estimates?
- What's their sampling distribution?
- Are they optimal?

These are exactly the questions this chapter will answer!

i Finnish-English Terminology Reference

For Finnish-speaking students, here are the key terms we'll use in this chapter:

English	Finnish	Context
Equivariant	Ekvivariantti	Property of MLE under reparameterization
Efficient	Tehokas	Optimal variance property
Score function	Rinnefunktio	Gradient of log-likelihood
Fisher information	Fisherin informaatio	Variance of score function
Delta method	Delta-menetelmä	Method for transformed parameters
Sufficient statistic	Tyhjentävä tunnusluku	Contains all information about parameter
Latent variable	Piilomuuttuja	Unobserved variable in model
EM Algorithm	EM-algoritmi	Expectation-Maximization algorithm
Asymptotic normality	Asympoottinen normaalisuus	Large-sample distribution property
Consistency	Konsistenssi/Tarkentuva	Convergence to true value

6.4 Core Properties of the Maximum Likelihood Estimator

Before exploring its properties, let's recall the definition of the **maximum likelihood estimator** (MLE) from Chapter 5:¹

Let X_1, \dots, X_n be i.i.d. with PDF (or PMF) $f(x; \theta)$.

The **likelihood function** is:

$$\mathcal{L}_n(\theta) = \prod_{i=1}^n f(X_i; \theta)$$

¹Note that the term MLE is used interchangeably to denote maximum-likelihood estimation (the technique), maximum-likelihood estimator (the rule to compute the estimate) and maximum-likelihood estimate (the result).

The **log-likelihood function** is:

$$\ell_n(\theta) = \log \mathcal{L}_n(\theta) = \sum_{i=1}^n \log f(X_i; \theta)$$

The **maximum likelihood estimator** (MLE), denoted by $\hat{\theta}_n$, is the value of θ that maximizes $\mathcal{L}_n(\theta)$ (or equivalently, $\ell_n(\theta)$).

6.4.1 Overview

In Chapter 5, we saw that the MLE is found by maximizing the likelihood function – a principle that seems intuitively reasonable. But intuition alone doesn't make for good statistics. We need to understand *why* the MLE works so well.

It turns out that the MLE has several remarkable properties that make it the “gold standard” of parametric estimation:

Property	Mathematical Statement	Intuitive Meaning
Consistency	$\hat{\theta}_n \xrightarrow{P} \theta_*$	The MLE converges to the true parameter value as $n \rightarrow \infty$
Equivariance	If $\hat{\theta}_n$ is the MLE of θ , then $g(\hat{\theta}_n)$ is the MLE of $g(\theta)$	The MLE behaves sensibly under parameter transformations
Asymptotic Normality	$\frac{(\hat{\theta}_n - \theta_*)}{\text{se}} \rightsquigarrow \mathcal{N}(0, 1)$	The MLE has an approximately normal distribution for large samples
Asymptotic Efficiency	$\text{Var}(\hat{\theta}_n)$ achieves the Cramér-Rao lower bound	The MLE has the smallest possible variance among consistent estimators
Approximate Bayes	$\hat{\theta}_n \approx \arg \max_{\theta} \pi(\theta X^n)$ with flat prior	The MLE approximates the Bayesian posterior mode

Let's explore each of these properties in detail.

6.4.2 Consistency: Getting It Right Eventually

The most fundamental property we could ask of any estimator is that it gets closer to the truth as we collect more data. This is the property of **consistency**.

An estimator $\hat{\theta}_n$ is **consistent** for θ_* if:

$$\hat{\theta}_n \xrightarrow{P} \theta_*$$

That is, for any $\epsilon > 0$:

$$\lim_{n \rightarrow \infty} P(|\hat{\theta}_n - \theta_*| > \epsilon) = 0$$

In words: as we collect more data, the probability that our estimate is far from the truth goes to zero. The MLE has this property under mild conditions.

i The Theory Behind Consistency

The consistency of the MLE is deeply connected to the Kullback-Leibler (KL) divergence. For two densities f and g , the KL divergence is:

$$D(f, g) = \int f(x) \log \frac{f(x)}{g(x)} dx$$

Key properties:

- $D(f, g) \geq 0$ always
- $D(f, g) = 0$ if and only if $f = g$ (almost everywhere)

The crucial insight is that maximizing the log-likelihood is equivalent to minimizing the KL divergence between the true distribution and our model. As $n \rightarrow \infty$, the empirical distribution converges to the true distribution, so the MLE converges to the parameter that makes the model distribution equal to the true distribution.

Identifiability: For this to work, the model must be **identifiable**: different parameter values must correspond to different distributions. That is, $\theta \neq \psi$ implies $D(f(\cdot; \theta), f(\cdot; \psi)) > 0$.

Example: Consistency in Action

Let's visualize how the MLE becomes more accurate with increasing sample size n . We'll use the Poisson distribution where the MLE for the rate parameter λ is simply the sample mean.²

```
import matplotlib.pyplot as plt

# True parameter
true_lambda = 3.0

# Sample sizes to consider
sample_sizes = [10, 50, 100, 500, 1000, 5000]
num_simulations = 100

# Store results
mle_estimates = {n: [] for n in sample_sizes}

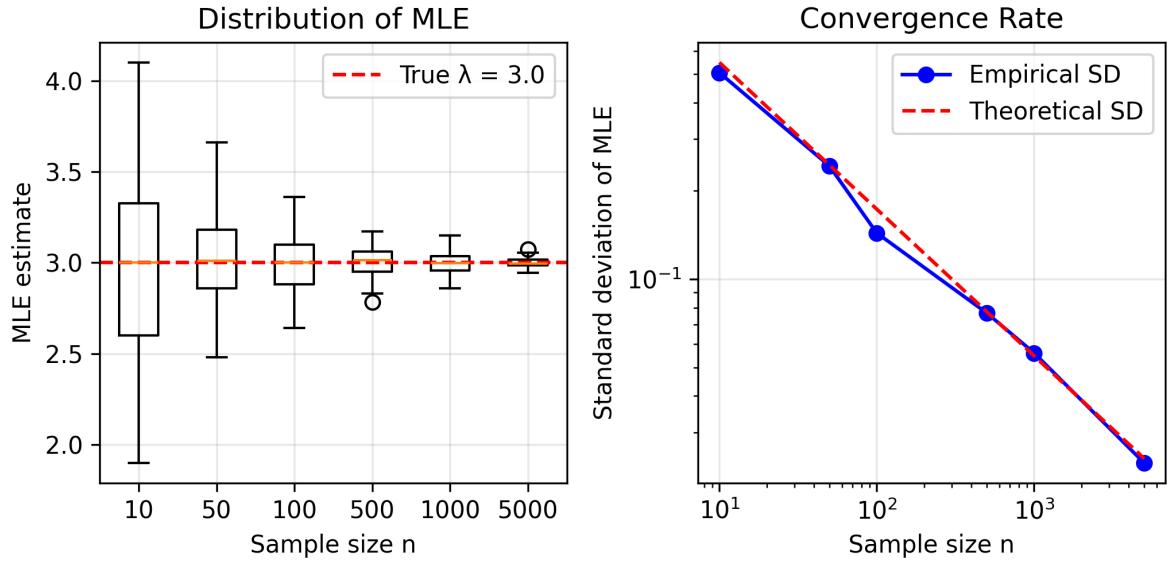
# Run simulations
np.random.seed(42)
for n in sample_sizes:
    for _ in range(num_simulations):
        # Generate Poisson data
        data = np.random.poisson(true_lambda, size=n)
        # MLE for Poisson is just the sample mean
        mle = np.mean(data)
        mle_estimates[n].append(mle)

# Create plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))

# Left plot: Box plots showing distribution at each sample size
positions = range(len(sample_sizes))
ax1.boxplot([mle_estimates[n] for n in sample_sizes],
            positions=positions,
            labels=[str(n) for n in sample_sizes])
ax1.axhline(y=true_lambda, color='red', linestyle='--', label=f'True = {true_lambda}')
ax1.set_xlabel('Sample size n')
ax1.set_ylabel('MLE estimate')
ax1.set_title('Distribution of MLE')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Right plot: Standard deviation vs sample size
std_devs = [np.std(mle_estimates[n]) for n in sample_sizes]
ax2.loglog(sample_sizes, std_devs, 'bo-', label='Empirical SD')
# Theoretical standard deviation: sqrt( /n)
theoretical_sd = [np.sqrt(true_lambda/n) for n in sample_sizes]
ax2.loglog(sample_sizes, theoretical_sd, 'r--', label='Theoretical SD')
ax2.set_xlabel('Sample size n')
ax2.set_ylabel('Standard deviation of MLE')
ax2.set_title('Convergence Rate')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



The plots demonstrate consistency: as n increases, the MLE concentrates more tightly around the true value, with standard deviation decreasing at rate $1/\sqrt{n}$.

6.4.3 Equivariance: Reparameterization Invariance

A subtle but important property of the MLE is **equivariance** (or “functional invariance”). This means that if we reparameterize our model, the MLE transforms in the natural way.

If $\hat{\theta}_n$ is the MLE of θ , then for any function g :

$$\widehat{g(\theta)} = g(\hat{\theta}_n)$$

That is, the MLE of $\tau = g(\theta)$ is $\hat{\tau}_n = g(\hat{\theta}_n)$.

Proof: Let $\tau = g(\theta)$ where g has inverse h , so $\theta = h(\tau)$. For any τ :

$$\mathcal{L}_n(\tau) = \prod_{i=1}^n f(X_i; h(\tau)) = \prod_{i=1}^n f(X_i; \theta) = \mathcal{L}_n(\theta)$$

where $\theta = h(\tau)$. Therefore:

$$\mathcal{L}_n(\tau) = \mathcal{L}_n(\theta) \leq \mathcal{L}_n(\hat{\theta}_n) = \mathcal{L}_n(\hat{\tau}_n)$$

Since this holds for any τ , we have $\hat{\tau}_n = g(\hat{\theta}_n)$.

Example: Equivariance in Practice

Consider $X_1, \dots, X_n \sim \mathcal{N}(\theta, 1)$ where we’re interested in both:

- The mean θ
- The parameter $\tau = e^\theta$ (perhaps θ is log-income and τ is income)

²In this example, we don’t need numerical optimization since we can compute the MLE analytically. We could still do numerical optimization and we would get the same results – it just would be slower and not very smart.

The MLE for θ is $\hat{\theta}_n = \bar{X}_n$. By equivariance:

$$\hat{\tau}_n = e^{\hat{\theta}_n} = e^{\bar{X}_n}$$

No need to rederive from scratch! This is particularly convenient when dealing with complex transformations.

Why Equivariance Matters:

1. **Convenience:** We can work in whatever parameterization is most natural for finding the MLE, then transform to the parameterization of interest.
2. **Consistency across parameterizations:** Different researchers might parameterize the same model differently (e.g., variance vs. precision in a normal distribution). Equivariance ensures they'll get equivalent results.
3. **Not universal:** This property is special to MLEs! Other estimators, like the Bayesian posterior mode (also known as MAP or *maximum a posteriori* estimate), generally lack this property. For instance, if θ has a uniform prior, $\tau = \theta^2$ does not have a uniform prior, leading to different posterior modes.



A Common Misconception

Equivariance does NOT mean that:

$$\mathbb{E}[g(\hat{\theta}_n)] = g(\mathbb{E}[\hat{\theta}_n])$$

In general, $g(\hat{\theta}_n)$ is a biased estimator of $g(\theta)$ even if $\hat{\theta}_n$ is unbiased for θ (unless g is linear). Equivariance is about what parameter value maximizes the likelihood, not about expected values.

6.4.4 Asymptotic Normality & Optimality

The consistency and equivariance properties are nice, but they don't tell us about the *distribution* of the MLE. How much uncertainty is there in our estimate? How efficient is it compared to other estimators?

The remarkable answer is that the MLE is approximately normally distributed with the smallest possible variance. We'll explore these twin properties – asymptotic normality and efficiency – in detail in the next section, as they require us to first understand a fundamental concept: the Fisher Information.

6.5 Fisher Information and Confidence Intervals

6.5.1 Fisher Information: Quantifying What Data Can Tell Us

To understand the precision of the MLE, we need to introduce one of the most important concepts in statistical theory: the **Fisher Information**. Named after statistician and polymath [R.A. Fisher](#), this quantity measures how much “information” about a parameter is contained in the data.

The Fisher Information is formally defined through the **score function** and its variance.

The **score function** is the gradient of the log-likelihood:

$$s(X; \theta) = \frac{\partial \log f(X; \theta)}{\partial \theta}$$

The **Fisher Information** is:

$$I_n(\theta) = \mathbb{V}_{\theta} \left(\sum_{i=1}^n s(X_i; \theta) \right) = \sum_{i=1}^n \mathbb{V}_{\theta}(s(X_i; \theta))$$

For a single observation ($n = 1$), we often write $I(\theta) = I_1(\theta)$.

The computation of the Fisher information can be simplified using the following results:

For an IID sample of size n :

1. $I_n(\theta) = n \cdot I(\theta)$ (information accumulates linearly)
2. $\mathbb{E}_\theta[s(X; \theta)] = 0$ (expected score is zero at the true parameter)
3. $I(\theta) = -\mathbb{E}_\theta \left[\frac{\partial^2 \log f(X; \theta)}{\partial \theta^2} \right]$ (expected negative curvature)

The last property shows that Fisher Information is literally the expected curvature of the log-likelihood – confirming our intuition about “sharpness”!

i Proof of Properties of Fisher Information and Score

Property 1: Linear accumulation for IID samples

For IID samples X_1, \dots, X_n :

$$I_n(\theta) = \mathbb{V}_\theta \left(\sum_{i=1}^n s(X_i; \theta) \right)$$

Since the X_i are independent and $\mathbb{V}(\sum Y_i) = \sum \mathbb{V}(Y_i)$ for independent random variables:

$$I_n(\theta) = \sum_{i=1}^n \mathbb{V}_\theta(s(X_i; \theta)) = n \cdot \mathbb{V}_\theta(s(X; \theta)) = n \cdot I(\theta)$$

Property 2: Expected score is zero

Under regularity conditions that allow interchange of derivative and integral:

$$\mathbb{E}_\theta[s(X; \theta)] = \mathbb{E}_\theta \left[\frac{\partial \log f(X; \theta)}{\partial \theta} \right] = \int \frac{\partial \log f(x; \theta)}{\partial \theta} f(x; \theta) dx$$

Using $\frac{\partial \log f}{\partial \theta} = \frac{1}{f} \frac{\partial f}{\partial \theta}$:

$$= \int \frac{\partial f(x; \theta)}{\partial \theta} dx = \frac{\partial}{\partial \theta} \int f(x; \theta) dx = \frac{\partial}{\partial \theta}(1) = 0$$

Property 3: Alternative formula using second derivative

We start with Property 2: $\mathbb{E}_\theta[s(X; \theta)] = 0$, which we can write explicitly as:

$$\int s(x; \theta) f(x; \theta) dx = \int \frac{\partial \log f(x; \theta)}{\partial \theta} f(x; \theta) dx = 0$$

Since this holds for all θ , we can differentiate both sides with respect to θ :

$$\frac{\partial}{\partial \theta} \int s(x; \theta) f(x; \theta) dx = 0$$

Under regularity conditions (allowing interchange of derivative and integral):

$$\int \frac{\partial}{\partial \theta} [s(x; \theta) f(x; \theta)] dx = 0$$

Using the product rule:

$$\int \left[\frac{\partial s(x; \theta)}{\partial \theta} f(x; \theta) + s(x; \theta) \frac{\partial f(x; \theta)}{\partial \theta} \right] dx = 0$$

Now, note that $\frac{\partial f(x; \theta)}{\partial \theta} = f(x; \theta) \cdot \frac{\partial \log f(x; \theta)}{\partial \theta} = f(x; \theta) \cdot s(x; \theta)$

Substituting:

$$\int \left[\frac{\partial s(x; \theta)}{\partial \theta} f(x; \theta) + s(x; \theta)^2 f(x; \theta) \right] dx = 0$$

This can be rewritten as:

$$\mathbb{E}_\theta \left[\frac{\partial s(X; \theta)}{\partial \theta} \right] + \mathbb{E}_\theta [s(X; \theta)^2] = 0$$

Since $s(X; \theta) = \frac{\partial \log f(X; \theta)}{\partial \theta}$, we have:

- $\frac{\partial s(X; \theta)}{\partial \theta} = \frac{\partial^2 \log f(X; \theta)}{\partial \theta^2}$
- $\mathbb{E}_\theta [s(X; \theta)^2] = \mathbb{V}_\theta [s(X; \theta)] = I(\theta)$ (since $\mathbb{E}_\theta [s(X; \theta)] = 0$)

Therefore:

$$\mathbb{E}_\theta \left[\frac{\partial^2 \log f(X; \theta)}{\partial \theta^2} \right] + I(\theta) = 0$$

Which gives us: $I(\theta) = -\mathbb{E}_\theta \left[\frac{\partial^2 \log f(X; \theta)}{\partial \theta^2} \right]$

Multiple Perspectives

Intuitive

Imagine you're trying to estimate a parameter by maximizing the log-likelihood function. Picture this function as a hill that you're climbing to find the peak (the MLE).

Now think about two scenarios:

1. **Sharp, pointy peak:** The log-likelihood drops off steeply as you move away from the maximum. Even a small change in the parameter makes the data much less likely. This means the data is very “informative” – it strongly prefers one specific parameter value.
2. **Flat, broad peak:** The log-likelihood changes slowly near the maximum. Many different parameter values give similar likelihoods. The data isn’t very informative about the exact parameter value.

The **Fisher Information** measures the “sharpness” or curvature of the log-likelihood at its peak:

- **Sharp peak** → High Fisher Information → Low variance for $\hat{\theta}$ → Narrow confidence interval
- **Flat peak** → Low Fisher Information → High variance for $\hat{\theta}$ → Wide confidence interval

This is the key insight: the same curvature that makes optimization easy (sharp peak = clear maximum) also makes estimation precise (sharp peak = low uncertainty)!

Mathematical

The meaning of $\mathbb{E}_\theta [s(X; \theta)] = 0$ is subtle and often misunderstood. It does NOT mean:

- The derivative is zero (that’s what happens at the MLE for a specific dataset)
- The log-likelihood is maximized at the true parameter

What it DOES mean:

- When data is generated from $f(x; \theta_*)$, the score evaluated at θ_* (true parameter) averages to zero across all possible datasets
- If you’re at the true parameter value and observe a random data point, it’s equally likely to suggest increasing or decreasing θ
- This is why the true parameter is “stable” – random samples don’t systematically pull the estimate away from the truth

Think of it as a balance point: at the true parameter, the data provides no systematic evidence to move in either direction, even though any individual sample might suggest moving up or down.

Computational

Let’s calculate and visualize the Fisher Information for a concrete example. We’ll examine the **Bernoulli distribution**, where the Fisher Information has a particularly interesting form, which we will derive later:

$$I(p) = 1/(p(1-p))$$

This shows how the precision of estimating a probability depends on the true probability value:

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Example: Fisher Information for Bernoulli(p)
# For Bernoulli: I(p) = 1/(p(1-p))

p_values = np.linspace(0.01, 0.99, 200)
fisher_info = 1 / (p_values * (1 - p_values))

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(7, 8))

# Top plot: Fisher Information
ax1.plot(p_values, fisher_info, 'b-', linewidth=2)
ax1.set_xlabel('p')
ax1.set_ylabel('I(p)')
ax1.set_title('Fisher Information for Bernoulli(p)')
ax1.grid(True, alpha=0.3)
ax1.set_ylim(0, 50)

# Middle plot: Standard error (1/sqrt(nI(p))) for different n
n_values = [10, 50, 100, 500]
colors = ['red', 'orange', 'green', 'blue']
for n, color in zip(n_values, colors):
    se = 1 / np.sqrt(n * fisher_info)
    ax2.plot(p_values, se, color=color, linewidth=2, label=f'n = {n}')
ax2.set_xlabel('p')
ax2.set_ylabel('Standard Error')
ax2.set_title('Standard Error of MLE')
ax2.legend()
ax2.grid(True, alpha=0.3)
ax2.set_ylim(0, 0.2)

# Bottom plot: Log-likelihood curves for different p values
n_obs = 20
successes = 12
theta_range = np.linspace(0.01, 0.99, 200)

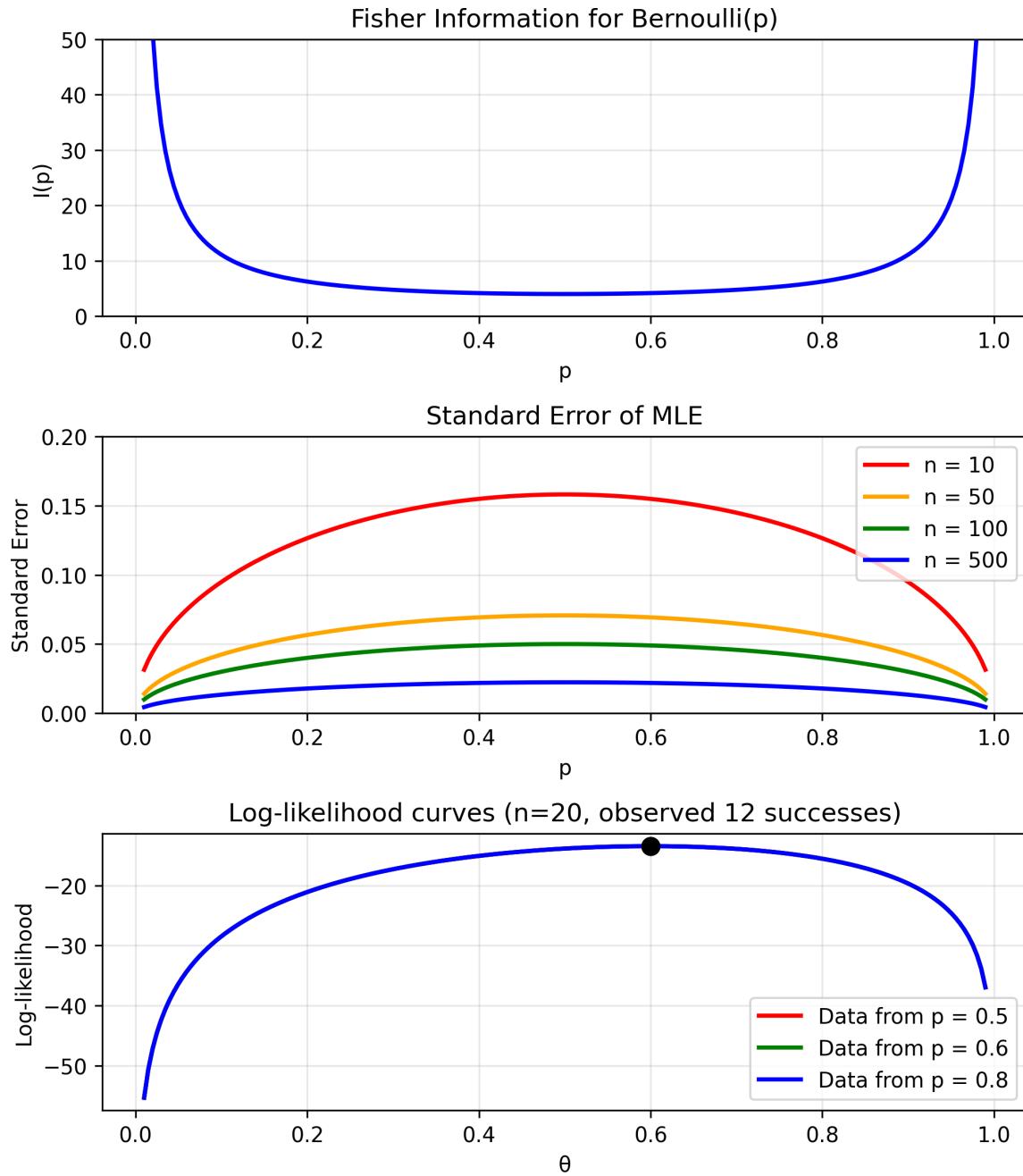
# Show log-likelihood for different true p values
for p_true, color in [(0.5, 'red'), (0.6, 'green'), (0.8, 'blue')]:
    log_lik = successes * np.log(theta_range) + (n_obs - successes) * np.log(1 - theta_range)
    ax3.plot(theta_range, log_lik, color=color, linewidth=2,
              label=f'Data from p = {p_true}')

# Mark the MLE
mle = successes / n_obs
mle_ll = successes * np.log(mle) + (n_obs - successes) * np.log(1 - mle)
ax3.plot(mle, mle_ll, 'ko', markersize=8)

ax3.set_xlabel('')
ax3.set_ylabel('Log-likelihood')
ax3.set_title(f'Log-likelihood curves (n={n_obs}, observed {successes} successes)')
ax3.legend()
ax3.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```



Key insights from the plots:

1. Fisher Information is lowest at $p = 0.5$ (hardest to estimate a fair coin)
2. Fisher Information $\rightarrow \infty$ as $p \rightarrow 0$ or $p \rightarrow 1$ (extreme probabilities are ‘easier’ to pin down)
3. Standard error decreases with both n and $I(p)$
4. The curvature of the log-likelihood reflects the Fisher Information

6.5.2 Asymptotic Normality of the MLE

Now we can state the key theorem that connects Fisher Information to the distribution of the MLE:

Let $se = \sqrt{\mathbb{V}(\hat{\theta}_n)}$. Under appropriate regularity conditions we have the following:

1. $\text{se} \approx \sqrt{1/I_n(\theta)}$ and

$$\frac{(\hat{\theta}_n - \theta)}{\text{se}} \rightsquigarrow \mathcal{N}(0, 1)$$

2. Let $\widehat{\text{se}} = \sqrt{1/I_n(\hat{\theta}_n)}$. Then

$$\frac{(\hat{\theta}_n - \theta)}{\widehat{\text{se}}} \rightsquigarrow \mathcal{N}(0, 1)$$

According to this theorem, the distribution of the MLE is approximately $\mathcal{N}(\theta, \widehat{\text{se}}^2)$. This is one of the most important results in statistics: it tells us not just that the MLE converges to the true value (consistency), but also gives us its approximate distribution for finite samples.

The key insight is that the same Fisher Information that measures how “sharp” the likelihood is also determines the precision of our estimate. Sharp likelihood \rightarrow High Fisher Information \rightarrow Small standard error \rightarrow Precise estimate.

6.5.3 Constructing Confidence Intervals for the MLE

With the asymptotic normality result in hand, we can now construct confidence intervals for our parameter estimates.

The interval

$$C_n = (\hat{\theta}_n - z_{\alpha/2} \widehat{\text{se}}, \hat{\theta}_n + z_{\alpha/2} \widehat{\text{se}})$$

is an asymptotically valid $(1 - \alpha)$ confidence interval for θ .

For a 95% confidence interval, $z_{0.025} \approx 1.96 \approx 2$, giving the simple rule:

$$95\% \text{ CI} \approx \hat{\theta}_n \pm 2 \cdot \widehat{\text{se}}$$

This type of confidence interval is used as the basis for statements about expected error in opinion polls and many other applications.

Example: Confidence Interval for a Proportion

Let's work through the complete derivation for the Bernoulli case, which gives us confidence intervals for proportions – one of the most common applications in practice.

For $X_1, \dots, X_n \sim \text{Bernoulli}(p)$, we have $f(x; p) = p^x(1-p)^{1-x}$ for $x \in \{0, 1\}$.

Step 1: Find the MLE The likelihood for n observations is:

$$\mathcal{L}_n(p) = \prod_{i=1}^n p^{X_i} (1-p)^{1-X_i} = p^S (1-p)^{n-S}$$

where $S = \sum_{i=1}^n X_i$ is the total number of successes.

Taking the log: $\ell_n(p) = S \log p + (n - S) \log(1 - p)$

Setting $\frac{d\ell_n}{dp} = \frac{S}{p} - \frac{n-S}{1-p} = 0$ gives $\hat{p}_n = \frac{S}{n} = \bar{X}_n$

Step 2: Compute the score function For a single observation:

$$\log f(X; p) = X \log p + (1 - X) \log(1 - p)$$

The score function is:

$$s(X; p) = \frac{\partial \log f(X; p)}{\partial p} = \frac{X}{p} - \frac{1-X}{1-p}$$

Let's verify $\mathbb{E}_p[s(X; p)] = 0$:

$$\mathbb{E}_p[s(X; p)] = \mathbb{E}_p \left[\frac{X}{p} - \frac{1-X}{1-p} \right] = \frac{p}{p} - \frac{1-p}{1-p} = 1 - 1 = 0 \checkmark$$

Step 3: Compute the second derivative

$$\frac{\partial^2 \log f(X; p)}{\partial p^2} = \frac{\partial}{\partial p} \left[\frac{X}{p} - \frac{1-X}{1-p} \right] = -\frac{X}{p^2} - \frac{1-X}{(1-p)^2}$$

Step 4: Find the Fisher Information Using the formula $I(p) = -\mathbb{E}_p \left[\frac{\partial^2 \log f(X; p)}{\partial p^2} \right]$:

$$I(p) = -\mathbb{E}_p \left[-\frac{X}{p^2} - \frac{1-X}{(1-p)^2} \right] = \mathbb{E}_p \left[\frac{X}{p^2} \right] + \mathbb{E}_p \left[\frac{1-X}{(1-p)^2} \right]$$

Since $\mathbb{E}_p[X] = p$ and $\mathbb{E}_p[1-X] = 1-p$:

$$I(p) = \frac{p}{p^2} + \frac{1-p}{(1-p)^2} = \frac{1}{p} + \frac{1}{1-p} = \frac{1-p+p}{p(1-p)} = \frac{1}{p(1-p)}$$

Step 5: Derive the standard error For n observations, $I_n(p) = n \cdot I(p) = \frac{n}{p(1-p)}$

The standard error of \hat{p}_n is:

$$\widehat{\text{se}} = \frac{1}{\sqrt{I_n(\hat{p}_n)}} = \frac{1}{\sqrt{\frac{n}{\hat{p}_n(1-\hat{p}_n)}}} = \sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}}$$

Step 6: Construct the confidence interval From the asymptotic normality theorem: $\frac{\hat{p}_n - p}{\widehat{\text{se}}} \rightsquigarrow \mathcal{N}(0, 1)$

For a $(1 - \alpha)$ confidence interval:

$$\mathbb{P} \left(-z_{\alpha/2} \leq \frac{\hat{p}_n - p}{\widehat{\text{se}}} \leq z_{\alpha/2} \right) \approx 1 - \alpha$$

Rearranging:

$$\mathbb{P} \left(\hat{p}_n - z_{\alpha/2} \cdot \widehat{\text{se}} \leq p \leq \hat{p}_n + z_{\alpha/2} \cdot \widehat{\text{se}} \right) \approx 1 - \alpha$$

Therefore, the $(1 - \alpha)$ confidence interval is:

$$\hat{p}_n \pm z_{\alpha/2} \sqrt{\frac{\hat{p}_n(1-\hat{p}_n)}{n}}$$

Example: Political Opinion Polling

Let's apply the confidence interval for a proportion from the previous worked example to understand how political polls work and what their "margin of error" really means.

```

# Political polling example
# Suppose we poll n=1000 randomly selected likely voters about a referendum

n = 1000 # Sample size
successes = 520 # Number who support the referendum
p_hat = successes / n # Sample proportion = 0.52 (52%)

# Standard error using the formula we derived
se = np.sqrt(p_hat * (1 - p_hat) / n)

# 95% confidence interval (using z_{0.025} = 1.96)
z_critical = 1.96
ci_lower = p_hat - z_critical * se
ci_upper = p_hat + z_critical * se

print("POLL RESULTS:")
print(f"Sample size: {n} voters")
print(f"Support for referendum: {p_hat*100:.1f}% ({successes} out of {n})")
print(f"Standard error: {se*100:.2f} percentage points")
print(f"95% confidence interval: ({ci_lower*100:.1f}%, {ci_upper*100:.1f}%)")
print(f"Margin of error: ±{z_critical*se*100:.1f} percentage points")

# Interpretation
print("\nINTERPRETATION:")
if ci_lower > 0.5:
    print("The referendum has statistically significant majority support.")
elif ci_upper < 0.5:
    print("The referendum has statistically significant minority support.")
else:
    print("The poll cannot determine if there's majority support (CI includes 50%).")

# Compare different scenarios to understand margin of error
print("\n" + "="*60)
print("HOW Margin OF ERROR VARIES:")
print("=".*60)

scenarios = [
    (0.50, 500, "50% support, n=500"),
    (0.50, 1000, "50% support, n=1000"),
    (0.50, 2000, "50% support, n=2000"),
    (0.20, 1000, "20% support, n=1000"),
    (0.10, 1000, "10% support, n=1000"),
    (0.01, 1000, "1% support, n=1000")
]

print(f"{'Scenario':<25} {'Margin of Error':>20} {'95% CI':>20}")
print("-" * 65)
for p, n, desc in scenarios:
    margin = 1.96 * np.sqrt(p * (1 - p) / n) * 100
    ci_low = (p - 1.96 * np.sqrt(p * (1 - p) / n)) * 100
    ci_high = (p + 1.96 * np.sqrt(p * (1 - p) / n)) * 100
    print(f"{desc:<25} ±{margin:>6.1f} percentage pts  ({ci_low:>5.1f}%, {ci_high:>5.1f}%)")

# Key assumptions and limitations
print("\nKEY ASSUMPTIONS:")
print("1. Random sampling from the population of interest")
print("2. Respondents answer truthfully")
print("3. The sample size is large enough for normal approximation (np > 10 and n(1-p) > 10)")
print("4. No systematic bias in who responds to the poll")

```

POLL RESULTS:

Sample size: 1000 voters
 Support for referendum: 52.0% (520 out of 1000)
 Standard error: 1.58 percentage points
 95% confidence interval: (48.9%, 55.1%)
 Margin of error: ± 3.1 percentage points

INTERPRETATION:

The poll cannot determine if there's majority support (CI includes 50%).

HOW MARGIN OF ERROR VARIES:

Scenario	Margin of Error	95% CI
50% support, n=500	± 4.4 percentage pts	(45.6%, 54.4%)
50% support, n=1000	± 3.1 percentage pts	(46.9%, 53.1%)
50% support, n=2000	± 2.2 percentage pts	(47.8%, 52.2%)
20% support, n=1000	± 2.5 percentage pts	(17.5%, 22.5%)
10% support, n=1000	± 1.9 percentage pts	(8.1%, 11.9%)
1% support, n=1000	± 0.6 percentage pts	(0.4%, 1.6%)

KEY ASSUMPTIONS:

1. Random sampling from the population of interest
2. Respondents answer truthfully
3. The sample size is large enough for normal approximation ($np \geq 10$ and $n(1-p) \geq 10$)
4. No systematic bias in who responds to the poll

Important insights about polling:

1. **The margin of error is largest when $p = 0.5$:** This is why close races are hardest to call. When support is near 50%, we have maximum uncertainty about which side has the majority.
2. **Sample size matters, but with diminishing returns:** To halve the margin of error, you need to quadruple the sample size (since margin $\propto 1/\sqrt{n}$).
3. **Extreme proportions have smaller margins:** If only 1% support something, we can estimate that quite precisely even with modest sample sizes.
4. **“Statistical ties”:** When the confidence interval includes 50%, we cannot conclude which side has majority support. This is often called a “statistical tie” in media reports.
5. **The stated margin usually assumes 95% confidence:** When polls report “ ± 3 percentage points,” they typically mean the 95% confidence interval extends 3 points in each direction.

 Advanced: Limitations of the Wald Interval

The confidence interval we derived above, known as the **Wald interval**, has poor coverage when p is near 0 or 1, or when n is small. In practice, consider using:

- **Agresti-Coull interval:** Add 2 successes and 2 failures before computing: $\tilde{p} = (S + 2)/(n + 4)$.
- **Wilson score interval:** More complex but widely recommended as the default choice.

The Wald interval remains important for understanding the theory, but these alternatives perform better in practice.

i Advanced: The Cramér-Rao Lower Bound

We mentioned that the MLE achieves the smallest possible variance. This is formalized by the **Cramér-Rao Lower Bound**:

For any unbiased estimator $\tilde{\theta}$:

$$\text{V}(\tilde{\theta}) \geq \frac{1}{I_n(\theta)}$$

Since the MLE asymptotically achieves variance $1/I_n(\theta)$, it is **asymptotically efficient** – no consistent estimator can do better!

This is why we call the MLE “optimal”: it extracts all possible information from the data about the parameter.

6.6 Additional Topics

6.6.1 The Delta Method: Confidence Intervals for Transformations

Often we’re not interested in the parameter θ itself, but in some transformation $\tau = g(\theta)$. For example:

- If θ is log-odds, we might want odds $\tau = e^\theta$
- If θ is standard deviation, we might want variance $\tau = \theta^2$
- If θ is a rate parameter, we might want mean lifetime $\tau = 1/\theta$

Due to the **equivariance property** seen earlier, if we know the MLE $\hat{\theta}$, we know also that the MLE of τ is $\hat{\tau} = g(\hat{\theta})$. However, what about the confidence intervals of $\hat{\tau}$?

The **Delta Method** provides a way to find confidence intervals for transformed parameters.

If $\tau = g(\theta)$ where g is differentiable and $g'(\theta) \neq 0$, then:

$$\widehat{\text{se}}(\hat{\tau}_n) \approx |g'(\hat{\theta}_n)| \cdot \widehat{\text{se}}(\hat{\theta}_n)$$

Therefore:

$$\frac{(\hat{\tau}_n - \tau)}{\widehat{\text{se}}(\hat{\tau}_n)} \rightsquigarrow \mathcal{N}(0, 1)$$

i Intuition Behind the Delta Method

If we zoom in enough, any smooth function looks linear. The Delta Method says that when we transform our estimate through a function g , the standard error gets multiplied by $|g'(\hat{\theta}_n)|$ – the absolute slope of the function at our estimate.

This approximation becomes exact as $n \rightarrow \infty$ because the variability of $\hat{\theta}_n$ shrinks, effectively “zooming in” on the region where g is effectively linear.

Example: Full Delta Method Workflow

Let’s work through estimating $\psi = \log \sigma$ for a Normal distribution with known mean. This example demonstrates every step of the Delta Method process.

Setup: $X_1, \dots, X_n \sim \mathcal{N}(\mu, \sigma^2)$ with μ known.

Step 1: Find MLE for σ

The log-likelihood is:

$$\ell(\sigma) = -n \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^n (X_i - \mu)^2$$

Taking the derivative and setting to zero:

$$\frac{d\ell}{d\sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^n (X_i - \mu)^2 = 0$$

Solving: $\hat{\sigma}_n = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2}$

Step 2: Find Fisher Information for σ

The log density for a single observation is:

$$\log f(X; \sigma) = -\log \sigma - \frac{(X - \mu)^2}{2\sigma^2}$$

First derivative:

$$\frac{\partial \log f(X; \sigma)}{\partial \sigma} = -\frac{1}{\sigma} + \frac{(X - \mu)^2}{\sigma^3}$$

Second derivative:

$$\frac{\partial^2 \log f(X; \sigma)}{\partial \sigma^2} = \frac{1}{\sigma^2} - \frac{3(X - \mu)^2}{\sigma^4}$$

Fisher Information:

$$I(\sigma) = -\mathbb{E} \left[\frac{\partial^2 \log f(X; \sigma)}{\partial \sigma^2} \right] = -\frac{1}{\sigma^2} + \frac{3\sigma^2}{\sigma^4} = \frac{2}{\sigma^2}$$

Step 3: Standard Error for $\hat{\sigma}$

$$\widehat{\text{se}}(\hat{\sigma}_n) = \frac{1}{\sqrt{nI(\hat{\sigma}_n)}} = \frac{\hat{\sigma}_n}{\sqrt{2n}}$$

Step 4: Apply the Delta Method

For $\psi = g(\sigma) = \log \sigma$, we have $g'(\sigma) = 1/\sigma$. Therefore:

$$\widehat{\text{se}}(\hat{\psi}_n) = |g'(\hat{\sigma}_n)| \cdot \widehat{\text{se}}(\hat{\sigma}_n) = \frac{1}{\hat{\sigma}_n} \cdot \frac{\hat{\sigma}_n}{\sqrt{2n}} = \frac{1}{\sqrt{2n}}$$

Step 5: Confidence Interval

A 95% confidence interval for $\psi = \log \sigma$ is:

$$\hat{\psi}_n \pm \frac{2}{\sqrt{2n}} = \log \hat{\sigma}_n \pm \frac{2}{\sqrt{2n}}$$

Notice something remarkable: the standard error for $\log \sigma$ doesn't depend on σ itself! This is one reason why log-transformations are often used for scale parameters.

Verification Via Simulation

Let's verify the Delta Method through simulation. We'll generate many samples, compute both $\hat{\sigma}$ and $\log \hat{\sigma}$ for each, and check if their empirical standard errors match the theoretical predictions:

```

# Demonstrate the Delta Method with simulations
np.random.seed(42)

# True parameters
mu_true = 5.0
sigma_true = 2.0
n = 100

# Generate data and compute MLEs
n_sims = 1000
sigma_mles = []
log_sigma_mles = []

for _ in range(n_sims):
    data = np.random.normal(mu_true, sigma_true, n)
    sigma_hat = np.sqrt(np.mean((data - mu_true)**2))
    sigma_mles.append(sigma_hat)
    log_sigma_mles.append(np.log(sigma_hat))

sigma_mles = np.array(sigma_mles)
log_sigma_mles = np.array(log_sigma_mles)

# Theoretical values
theoretical_se_sigma = sigma_true / np.sqrt(2*n)
theoretical_se_log_sigma = 1 / np.sqrt(2*n)

# Create plots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))

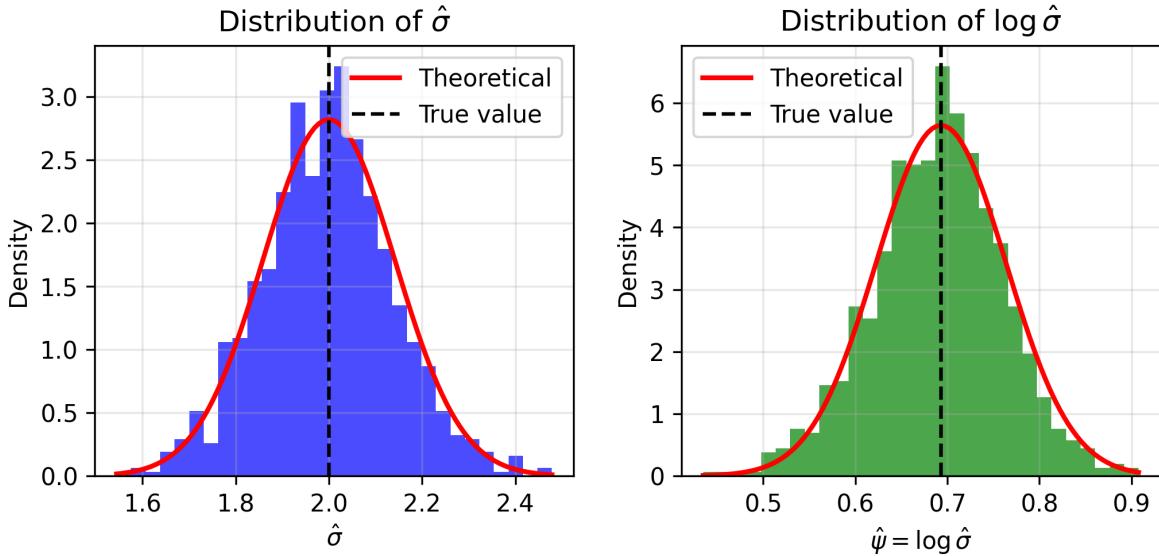
# Left: Distribution of sigma MLE
ax1.hist(sigma_mles, bins=30, density=True, alpha=0.7, color='blue')
x = np.linspace(sigma_mles.min(), sigma_mles.max(), 100)
ax1.plot(x, stats.norm.pdf(x, sigma_true, theoretical_se_sigma),
          'r-', linewidth=2, label='Theoretical')
ax1.axvline(sigma_true, color='black', linestyle='--', label='True value')
ax1.set_xlabel('$\hat{\sigma}$')
ax1.set_ylabel('Density')
ax1.set_title('Distribution of $\hat{\sigma}$')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Right: Distribution of log(sigma) MLE
ax2.hist(log_sigma_mles, bins=30, density=True, alpha=0.7, color='green')
x = np.linspace(log_sigma_mles.min(), log_sigma_mles.max(), 100)
ax2.plot(x, stats.norm.pdf(x, np.log(sigma_true), theoretical_se_log_sigma),
          'r-', linewidth=2, label='Theoretical')
ax2.axvline(np.log(sigma_true), color='black', linestyle='--', label='True value')
ax2.set_xlabel('$\hat{\psi} = \log \hat{\sigma}$')
ax2.set_ylabel('Density')
ax2.set_title('Distribution of $\log \hat{\sigma}$')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("Standard Error Comparison:")
print(f"Empirical SE($\hat{\sigma}$): {np.std(sigma_mles):.4f}")
print(f"Theoretical SE($\hat{\sigma}$): {theoretical_se_sigma:.4f}")
print(f"Empirical SE($\log \hat{\sigma}$): {np.std(log_sigma_mles):.4f}")

```



Standard Error Comparison:

Empirical SE($\hat{\sigma}$): 0.1403

Theoretical SE($\hat{\sigma}$): 0.1414

Empirical SE($\log \hat{\sigma}$): 0.0705

Theoretical SE($\log \hat{\sigma}$): 0.0707

SE ratio (empirical): 0.5027

SE ratio (Delta method): 0.5000

Key takeaways: The simulation confirms that the Delta Method works perfectly! The empirical standard error ratio between $\log \hat{\sigma}$ and $\hat{\sigma}$ matches exactly what the Delta Method predicts: $1/\sigma$. Both distributions are well-approximated by normal distributions, validating the asymptotic normality of MLEs.

6.6.2 Multiparameter Models

Real-world problems often involve multiple parameters. The theory we saw in this chapter extends naturally, with matrices replacing scalars.

For $\theta = (\theta_1, \dots, \theta_k)$, let $\hat{\theta} = (\hat{\theta}_1, \dots, \hat{\theta}_k)$ be the MLE. The key concepts become:

Let $\ell_n(\theta) = \sum_{i=1}^n \log f(X_i; \theta)$ and define:

$$H_{jk} = \frac{\partial^2 \ell_n}{\partial \theta_j \partial \theta_k}$$

The **Fisher Information Matrix** is:

$$I_n(\theta) = - \begin{pmatrix} \mathbb{E}_{\theta}(H_{11}) & \mathbb{E}_{\theta}(H_{12}) & \cdots & \mathbb{E}_{\theta}(H_{1k}) \\ \mathbb{E}_{\theta}(H_{21}) & \mathbb{E}_{\theta}(H_{22}) & \cdots & \mathbb{E}_{\theta}(H_{2k}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{E}_{\theta}(H_{k1}) & \mathbb{E}_{\theta}(H_{k2}) & \cdots & \mathbb{E}_{\theta}(H_{kk}) \end{pmatrix}$$

Under regularity conditions:

1. The MLE is approximately multivariate normal:

$$(\hat{\theta} - \theta) \approx \mathcal{N}(0, J_n)$$

where $J_n = I_n^{-1}(\theta)$ is the inverse Fisher Information matrix.

2. For parameter θ_j , the standard error is:

$$\widehat{\text{se}}_j = \sqrt{J_n(j, j)}$$

(the square root of the j -th diagonal element of J_n)

3. The covariance between $\hat{\theta}_j$ and $\hat{\theta}_k$ is $J_n(j, k)$.

Example: Multiparameter Normal Distribution

For $X_1, \dots, X_n \sim \mathcal{N}(\mu, \sigma^2)$ with both parameters unknown:

The Fisher Information matrix is:

$$I_n(\mu, \sigma) = n \begin{pmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{2}{\sigma^2} \end{pmatrix}$$

Therefore:

$$J_n = \frac{1}{n} \begin{pmatrix} \sigma^2 & 0 \\ 0 & \frac{\sigma^2}{2} \end{pmatrix}$$

This tells us:

- $\widehat{\text{se}}(\hat{\mu}) = \sigma/\sqrt{n}$ (familiar formula!)
- $\widehat{\text{se}}(\hat{\sigma}) = \sigma/\sqrt{2n}$
- $\text{Cov}(\hat{\mu}, \hat{\sigma}) = 0$ (they're independent!)

The zero off-diagonal terms mean that uncertainty about μ doesn't affect our estimate of σ , and vice versa. This orthogonality is special to the normal distribution.

i Advanced: Multiparameter Delta Method

For a function $\tau = g(\theta_1, \dots, \theta_k)$, the Delta Method generalizes to:

$$\widehat{\text{se}}(\hat{\tau}) = \sqrt{(\widehat{\nabla}g)^T \widehat{J}_n(\widehat{\nabla}g)}$$

where $\widehat{\nabla}g$ is the gradient of g evaluated at $\hat{\theta}$.

Example: For the Normal distribution, if we're interested in the coefficient of variation $\tau = \sigma/\mu$:

$$\nabla g = \begin{pmatrix} -\sigma/\mu^2 \\ 1/\mu \end{pmatrix}$$

The standard error involves both parameter uncertainties and their covariance (though the covariance is zero for the normal case).

6.6.3 Sufficient Statistics

A **statistic** is a function $T(X^n)$ of the data. A **sufficient statistic** is a statistic that contains all the information in the data about the parameter.

A statistic $T(X^n)$ is **sufficient** for parameter θ if the conditional distribution of the data given T doesn't depend on θ .

Let $X_1, \dots, X_n \sim \text{Bernoulli}(p)$. Then $\mathcal{L}(p) = p^S(1-p)^{n-S}$ where $S = \sum_{i=1}^n X_i$.

The likelihood depends on the data only through S , so S is sufficient. This means that once we know the total number of successes, the individual outcomes provide no additional information about p .

A sufficient statistic is **minimal** if it provides the most compressed summary possible while still retaining all information about the parameter. More precisely: any other sufficient statistic can be "reduced" to the minimal one, but not vice versa.

i Connection Between MLE and Sufficiency

When a non-trivial sufficient statistic exists, the MLE depends on the data only through that statistic.
Examples:

- **Bernoulli:** MLE $\hat{p} = S/n$ depends only on sufficient statistic $S = \sum X_i$
- **Normal:** MLE $(\hat{\mu}, \hat{\sigma}^2)$ depends only on sufficient statistics (\bar{X}, S^2)
- **Uniform(0,θ):** MLE is exactly the sufficient statistic $\max\{X_i\}$

This provides theoretical justification for data reduction: when sufficient statistics exist, we can compress our data without losing any information about the parameter. However, not all models have nice sufficient statistics beyond the trivial one (the entire dataset).

6.7 Connection to Machine Learning: Cross-Entropy as MLE

One of the most profound connections between statistics and modern machine learning is that many “loss functions” used in ML are actually negative log-likelihoods in disguise. This isn’t a coincidence – it reflects the deep statistical foundations of machine learning. Let’s consider the case of the cross-entropy loss used in classification tasks.

Consider a classification problem where we predict probabilities for K classes using a machine learning model $f(X; \theta)$ parameterized by θ .

The **cross-entropy loss** is:

$$H(p, q) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^K p(Y_i = j) \log q_\theta(Y_i = j)$$

where:

- $p(Y_i = j)$ is the observed distribution (1 if observation i is class j , 0 otherwise)
- $q_\theta(Y_i = j) = f(X_i; \theta)_j$ is the predicted probability from our model with parameters θ

Let’s show this is exactly the negative log-likelihood for a categorical distribution.

The likelihood for categorical data is:

$$\mathcal{L}_n(\theta) = \prod_{i=1}^n \text{Categorical}(Y_i; f(X_i; \theta))$$

The categorical probability for observation i is:

$$\text{Categorical}(Y_i; f(X_i; \theta)) = \prod_{j=1}^K f(X_i; \theta)_j^{p(Y_i=j)}$$

Taking the log:

$$\ell_n(\theta) = \sum_{i=1}^n \sum_{j=1}^K p(Y_i = j) \log f(X_i; \theta)_j$$

This is exactly $-n \cdot H(p, q)$. Thus: **Minimizing cross-entropy = Maximizing likelihood!**

i Other Common ML Loss Functions as MLEs

This pattern appears throughout machine learning:

ML Loss Function	Statistical Model	MLE Interpretation
Mean Squared Error (MSE)	$Y \sim \mathcal{N}(\mu(x), \sigma^2)$	Gaussian likelihood
Mean Absolute Error (MAE)	$Y \sim \text{Laplace}(\mu(x), b)$	Laplace likelihood
Huber Loss	Hybrid of L_1 and L_2	Robust to outliers

Understanding that common ML losses are negative log-likelihoods provides:

- Principled loss design:** When facing a new problem, you can derive an appropriate loss function by specifying a probabilistic model for your data.
- Historical context:** It explains why these particular loss functions became standard – they weren't arbitrary choices but emerged from statistical principles.
- Intuition:** Knowing the probabilistic interpretation helps understand when each loss is appropriate (e.g., MAE for heavy-tailed errors, MSE for Gaussian noise).

Example: Deriving a Custom Loss

Suppose you believe errors in your regression problem approximately follow a [Student-t distribution](#) (heavy tails for robustness). The negative log-likelihood gives you the loss function:

$$\text{Loss}(y, \hat{y}) = \frac{\nu + 1}{2} \log \left(1 + \frac{(y - \hat{y})^2}{\nu s^2} \right)$$

where ν is degrees of freedom and s is scale. This naturally down-weights outliers!

6.8 MLE for Latent Variable Models: The EM Algorithm

6.8.1 The Challenge of Latent Variables

So far, we've assumed that maximizing the likelihood is straightforward – we take derivatives, set them to zero, and solve (possibly numerically). But what if the likelihood itself is intractable?

This often happens when our model involves **latent (unobserved) variables** – hidden quantities that would make the problem easy if only we could observe them.

Common examples with latent variables:

- **Mixture models:** Which component generated each observation?
- **Hidden Markov models:** What's the hidden state sequence?
- **Factor analysis:** What are the values of the latent factors?
- **Missing data:** What would the missing values have been?

The canonical example is the **Gaussian Mixture Model (GMM)**:³

$$f(y; \theta) = \sum_{k=1}^K \pi_k \mathcal{N}(y; \mu_k, \sigma_k^2)$$

where $\theta = (\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \sigma_1, \dots, \sigma_K)$.

³Also called **Mixture of Gaussians (MoG)**.

The likelihood involves a sum inside the logarithm:

$$\ell_n(\theta) = \sum_{i=1}^n \log \left[\sum_{k=1}^K \pi_k \mathcal{N}(y_i; \mu_k, \sigma_k^2) \right]$$

This is a nightmare to differentiate! The derivatives involve ratios of sums – messy and hard to work with.

The key insight: If we knew which component $Z_i \in \{1, \dots, K\}$ generated each observation Y_i , the problem would become trivial – we'd just fit separate Gaussians to each group.

i Example: Two-Component Gaussian Mixture ($K = 2$)

For the special case of $K = 2$ (two components), the mixture model simplifies to:

$$f(y; \theta) = \pi \mathcal{N}(y; \mu_1, \sigma_1^2) + (1 - \pi) \mathcal{N}(y; \mu_2, \sigma_2^2)$$

where:

- $\pi \in [0, 1]$ is the mixing weight (probability of component 1)
- $(1 - \pi)$ is the probability of component 2
- μ_1, μ_2 are the means of the two Gaussian components
- σ_1^2, σ_2^2 are the variances of the two components
- The parameter vector is $\theta = (\pi, \mu_1, \mu_2, \sigma_1, \sigma_2)$

Interpretation: Each observation Y_i is generated by:

1. First, flip a biased coin with probability π of heads
2. If heads: draw $Y_i \sim \mathcal{N}(\mu_1, \sigma_1^2)$
3. If tails: draw $Y_i \sim \mathcal{N}(\mu_2, \sigma_2^2)$

The **latent variables** $Z_1, \dots, Z_n \in \{1, 2\}$ indicate which component generated each observation. If we knew these Z_i values, estimation would be trivial – we'd simply:

- Estimate π as the proportion of observations from component 1
- Estimate μ_1, σ_1 using only observations where $Z_i = 1$
- Estimate μ_2, σ_2 using only observations where $Z_i = 2$

But since the Z_i are unobserved, we need a method to estimate both the component assignments and the parameters.

6.8.2 The Expectation-Maximization (EM) Algorithm

The EM algorithm is a clever iterative procedure that alternates between:

1. Guessing the values of the latent variables (E-step)
2. Updating parameters as if those guesses were correct (M-step)

Multiple Perspectives

Intuitive

Think of the EM algorithm as a “chicken and egg” problem solver for mixture models.

The dilemma:

- If we knew which cluster each point belonged to, we could easily estimate the cluster parameters (just compute means and variances for each group)
- If we knew the cluster parameters, we could easily assign points to clusters (pick the most likely cluster for each point)

The EM solution: Start with a guess and alternate!

1. **E-step (Expectation):** Given current cluster parameters, compute “soft assignments” – the probability that each point belongs to each cluster. These are called “responsibilities.”
 - Point near cluster 1 center: Maybe 90% probability for cluster 1, 10% for cluster 2

- Point between clusters: Maybe 50-50 split
2. **M-step (Maximization):** Update cluster parameters using weighted calculations, where weights are the responsibilities.
- New mean for cluster 1: Weighted average of all points, using their cluster 1 responsibilities as weights
 - Points with high responsibility contribute more to the cluster's parameters
3. **Iterate:** Keep alternating until convergence.

It's like a dance where clusters and assignments gradually find their proper configuration, each step making the other more accurate.

Mathematical

The EM algorithm maximizes the expected complete-data log-likelihood. Let:

- Y = observed data
- Z = latent/missing data
- (Y, Z) = complete data

The algorithm iterates between:

E-step: Compute the expected complete-data log-likelihood:

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_{Z|Y,\theta^{(t)}}[\log P(Y, Z|\theta)]$$

This expectation is over the distribution of Z given the observed data Y and current parameters $\theta^{(t)}$.

M-step: Maximize with respect to θ :

$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^{(t)})$$

Key property: The likelihood is guaranteed to increase (or stay the same) at each iteration:

$$\mathcal{L}(\theta^{(t+1)}) \geq \mathcal{L}(\theta^{(t)})$$

This monotonic improvement ensures convergence to a local maximum.

Computational

Let's implement and visualize the EM algorithm for a 2D Gaussian mixture:

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal

# Generate synthetic data from a mixture of 3 Gaussians
np.random.seed(42)
n_samples = 300
n_components = 3

# True parameters
true_means = np.array([[-2, -2], [0, 2], [3, -1]])
true_covs = [np.array([[0.5, 0.2], [0.2, 0.5]]),
             np.array([[0.8, -0.3], [-0.3, 0.8]]),
             np.array([[0.6, 0], [0, 0.3]])]
true_weights = np.array([0.3, 0.5, 0.2])

# Generate data
data = []
true_labels = []
for i in range(n_samples):
    # Choose component
    k = np.random.choice(n_components, p=true_weights)
    true_labels.append(k)
    # Generate point from that component
    data.append(np.random.multivariate_normal(true_means[k], true_covs[k]))
data = np.array(data)
true_labels = np.array(true_labels)

# Initialize EM with random parameters
means = np.random.randn(n_components, 2) * 2
covs = [np.eye(2) for _ in range(n_components)]
weights = np.ones(n_components) / n_components

# Function to plot current state
def plot_em_state(data, means, covs, weights, responsibilities, iteration, log_likelihood_history):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3.5))

    # Left plot: data colored by responsibilities
    colors = responsibilities @ np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    ax1.scatter(data[:, 0], data[:, 1], c=colors, alpha=0.6, s=30)

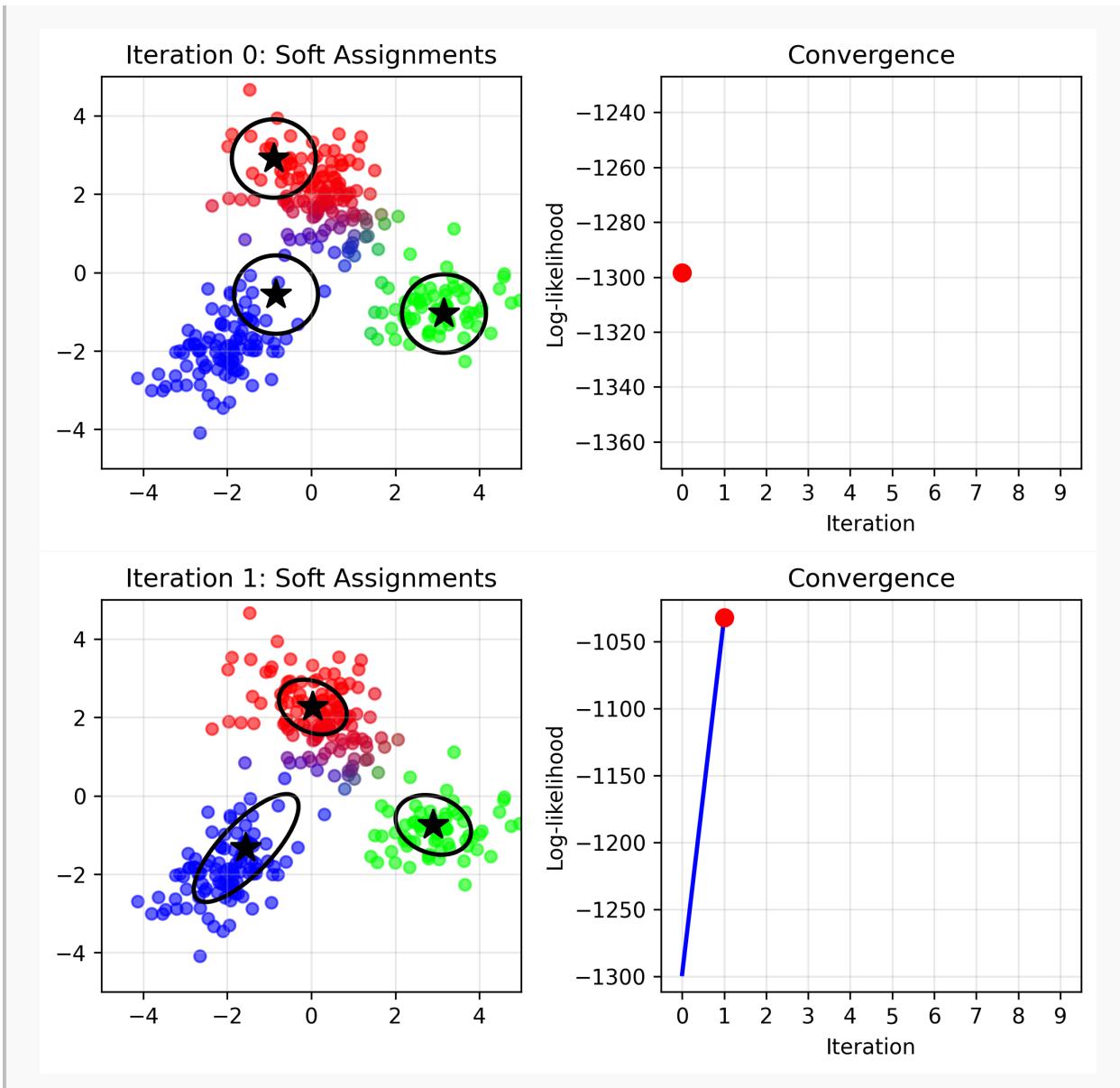
    # Plot component centers and ellipses
    for k in range(n_components):
        ax1.plot(means[k, 0], means[k, 1], 'k*', markersize=15)

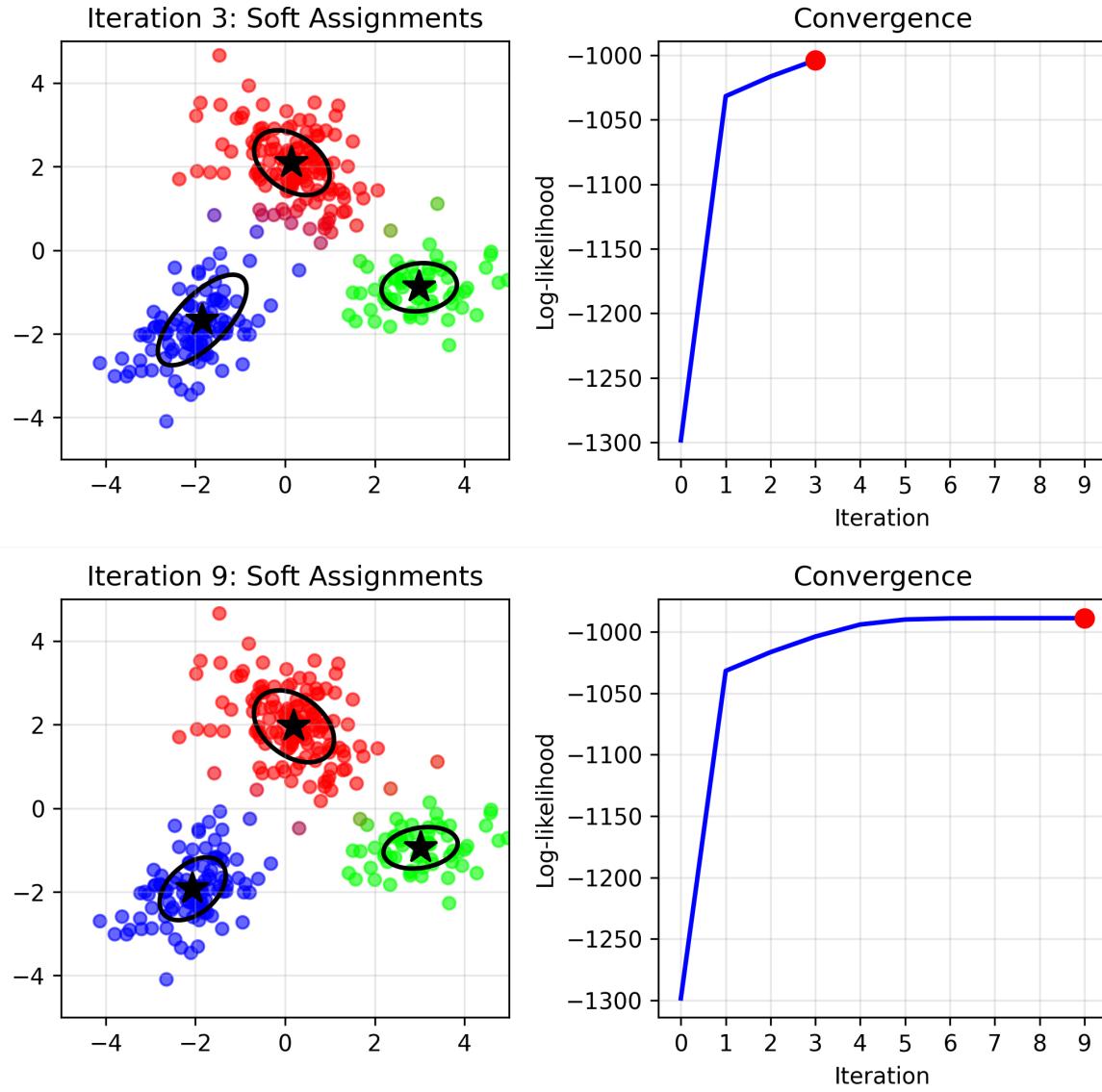
        # Draw ellipse representing covariance
        eigenvalues, eigenvectors = np.linalg.eigh(covs[k])
        angle = np.degrees(np.arctan2(eigenvectors[1, 0], eigenvectors[0, 0]))
        width, height = 2 * np.sqrt(eigenvalues)
        ellipse = Ellipse(means[k], width, height, angle=angle,
                           facecolor='none', edgecolor='black', linewidth=2)
        ax1.add_patch(ellipse)

    ax1.set_title(f'Iteration {iteration}: Soft Assignments')
    ax1.set_xlim(-5, 5)
    ax1.set_ylim(-5, 5)
    ax1.grid(True, alpha=0.3)

    # Right plot: log likelihood history
    ax2.plot(log_likelihood_history)
    ax2.set_title('Log Likelihood History')
    ax2.set_xlabel('Iteration')
    ax2.set_ylabel('Log Likelihood')

```





Final parameters:

Weights: [0.45669616 0.211968 0.33133584]

Means:

```
[[ 0.19903644  1.95778334]
 [ 3.02378239 -0.94719336]
 [-2.06041818 -1.92642337]]
```

The visualizations show:

- **Left plots:** Data points colored by their soft assignments (RGB = probabilities for 3 clusters)
- **Black stars:** Cluster centers
- **Ellipses:** Covariance structure of each component
- **Right plot:** Log-likelihood increasing monotonically

Notice how the algorithm gradually discovers the true cluster structure!

6.8.3 Properties and Practical Considerations

Strengths of EM:

1. **Guaranteed improvement:** The likelihood never decreases
2. **No step size tuning:** Unlike gradient descent, no learning rate to choose
3. **Naturally handles constraints:** Probabilities automatically sum to 1
4. **Interpretable intermediate results:** Soft assignments have meaning

Limitations and solutions:

1. **Local optima:** EM only finds a local maximum
 - **Solution:** Run from multiple random initializations
 - **Smart initialization:** Use [k-means++](#) or similar
2. **Slow convergence:** Can take many iterations near the optimum
 - **Solution:** Switch to Newton's method near convergence
 - **Early stopping:** Monitor log-likelihood changes
3. **Choosing number of components:** How many clusters?
 - **Solution:** Use information criteria (AIC, BIC)
 - **Cross-validation:** Evaluate on held-out data



The Initialization Trap

The EM algorithm is extremely sensitive to initialization. Poor starting values can lead to:

- Convergence to inferior local optima
- One component “eating” all the data
- Empty components (singularities)

Always run EM multiple times with different initializations and choose the result with the highest likelihood!

6.9 Chapter Summary and Connections

6.9.1 Key Concepts Review

We've explored the remarkable properties that make the Maximum Likelihood Estimator the “gold standard” of parametric estimation:

Core Properties of the MLE:

- **Consistency:** $\hat{\theta}_n \xrightarrow{P} \theta_*$ – the MLE converges to the truth as $n \rightarrow \infty$.
- **Equivariance:** $\widehat{g(\theta)} = g(\hat{\theta}_n)$ – reparameterization doesn't affect the MLE.
- **Asymptotic Normality:** $\hat{\theta}_n \approx \mathcal{N}(\theta, 1/I_n(\theta))$ – enabling confidence intervals.
- **Asymptotic Efficiency:** Achieves the Cramér-Rao lower bound – optimal variance.

Fisher Information:

- Measures the “information” about a parameter contained in data.
- Equals the expected curvature of the log-likelihood: $I(\theta) = -\mathbb{E}\left[\frac{\partial^2 \log f(X; \theta)}{\partial \theta^2}\right]$.
- Determines the precision of the MLE: $\text{Var}(\hat{\theta}_n) \approx 1/(nI(\theta))$.
- Sharp likelihood peak → High information → Precise estimates.

Deep Connections:

- Cross-entropy loss in ML = Negative log-likelihood.
- Many ML algorithms are secretly performing MLE.

Practical Tools:

- **Confidence Intervals:** $\hat{\theta}_n \pm z_{\alpha/2} \cdot \widehat{\text{se}}$ where $\widehat{\text{se}} = 1/\sqrt{I_n(\hat{\theta}_n)}$.
- **Delta Method:** For transformed parameters $\tau = g(\theta)$: $\widehat{\text{se}}(\hat{\tau}) \approx |g'(\hat{\theta})| \cdot \widehat{\text{se}}(\hat{\theta})$.
- **EM Algorithm:** Iterative method for MLEs with latent variables – alternates between E-step (soft assignments) and M-step (parameter updates).

6.9.2 The Big Picture

We have moved from simply *finding* estimators (Chapter 5) to *evaluating their quality*. This chapter revealed why the MLE is so widely used:

1. **It works:** Consistency ensures we get the right answer with enough data.
2. **It's optimal:** No other estimator has smaller asymptotic variance.
3. **It's practical:** To a degree, we can quantify uncertainty through Fisher Information.⁴
4. **It's flexible:** Equivariance means we can work in convenient parameterizations.
5. **It extends:** The EM algorithm handles complex models with latent structure.

6.9.3 Common Pitfalls to Avoid

1. **Misinterpreting Confidence Intervals:** A 95% CI does NOT mean:
 - “95% probability the parameter is in this interval” (that’s Bayesian!).
 - “95% of the data falls in this interval” (that’s a prediction interval).
 - Correct: “95% of such intervals constructed from repeated samples would contain the true parameter”.
2. **Assuming Asymptotic Results for Small Samples:**
 - Asymptotic normality may not hold for small n .
 - Fisher Information formulas are approximate for finite samples.
 - Bootstrap or exact methods may be preferable when $n < 30$.⁵
3. **Getting Stuck in Local Optima with EM:**
 - EM only guarantees local, not global, maximum.
 - Always run from multiple starting points.
 - Monitor log-likelihood to ensure proper convergence.
4. **Forgetting About Model Assumptions:**
 - MLE theory assumes the model is correctly specified (i.e., the true distribution belongs to our model family).
 - We briefly mentioned that parameters must be identifiable (Section on Consistency) – different parameter values must give different distributions.
 - The asymptotic results (normality, efficiency) require “regularity conditions” that we haven’t detailed but include smoothness of the likelihood.
5. **Over-interpreting Asymptotic Results:**
 - All our results (asymptotic normality, efficiency) are for large samples.
 - With small samples, the MLE might not be normally distributed and confidence intervals may be inaccurate.
 - The MLE’s optimality is theoretical – in practice, we might prefer more robust methods when data contains outliers or model assumptions are questionable.

6.9.4 Chapter Connections

Building on Previous Chapters:

- **Chapter 3 (Statistical Inference):** Provided the framework for evaluating estimators. Now we have seen that MLEs have optimal properties within this framework.
- **Chapter 4 (Bootstrap):** Offers a computational alternative to the analytical standard errors we derived here. When Fisher Information is hard to compute, bootstrap it!
- **Chapter 5 (Finding Estimators):** Showed how to find MLEs. This chapter justifies why we bother – they have provably good properties.

Looking Ahead:

- **Bayesian Inference:** While MLE obtains point estimates and confidence intervals, Bayesian methods provide full probability distributions over parameters. The MLE appears as the mode of the posterior

⁴As we will see in future lectures, a Bayesian approach is often more suitable for proper uncertainty quantification.

⁵This is just a rule of thumb.

with uniform priors.

- **Regression Models:** The theory in this chapter extends directly – least squares is MLE for normal errors, logistic regression is MLE for binary outcomes.
- **Model Selection:** Information criteria (AIC, BIC) build on the likelihood framework we've developed.
- **Robust Statistics:** When MLE assumptions fail, we need methods that sacrifice some efficiency for robustness.

The properties we've studied – consistency, asymptotic normality, efficiency – will reappear throughout statistics. Whether you're fitting neural networks or analyzing clinical trials, you're likely using some form of MLE, and the theory in this chapter explains why it works.

6.9.5 Self-Test Problems

1. **Fisher Information:** For $X_1, \dots, X_n \sim \text{Exponential}(\lambda)$ with PDF $f(x; \lambda) = \lambda e^{-\lambda x}$ for $x > 0$:
 - Find the Fisher Information $I(\lambda)$ for a single observation.
 - What is the approximate standard error of the MLE $\hat{\lambda}_n$?
2. **Confidence Intervals:** You flip a coin 100 times and observe 58 heads. Use Fisher Information to construct an approximate 95% confidence interval for the probability of heads.
3. **Delta Method:** If $\hat{p} = 0.4$ with standard error 0.05, find the standard error of $\hat{\tau} = \log(p/(1-p))$ using the Delta Method.
4. **EM Intuition:** In a two-component Gaussian mixture, the E-step computes “responsibilities” for each data point. What do these represent? Why can't we just assign each point to its most likely cluster?
5. **MLE Properties:** True or False (with brief explanation):
 - If $\hat{\theta}_n$ is the MLE of θ , then $\hat{\theta}_n^2$ is the MLE of θ^2 .
 - The MLE is always unbiased.
 - Fisher Information measures how “peaked” the likelihood function is.

6.9.6 Python and R Reference

Python and R Reference Code

Python and R code examples for Fisher Information, confidence intervals, the Delta Method, and the EM algorithm can be found in the HTML version of these notes. The code includes:

- Computing Fisher Information both symbolically and numerically
- Constructing confidence intervals using Fisher Information
- Implementing the Delta Method for transformed parameters
- Using built-in packages for Gaussian mixture models (`sklearn` in Python, `mclust` in R)
- Manual implementations of the EM algorithm

Key packages: - **Python:** `statsmodels`, `sympy`, `sklearn.mixture`, `jax` - **R:** `MASS`, `mclust`, `msm`, `numDeriv`

6.9.7 Connections to Source Material

Mapping to “All of Statistics”

This table maps sections in these lecture notes to the corresponding sections in Wasserman (2013) (“All of Statistics” or AoS).

Lecture Note Section	Corresponding AoS Section(s)
Introduction: How Good Are Our Estimators?	From slides and general context from AoS §9.4 introduction on properties of estimators.
Warm-up: Beta Distribution MLE	Example from slides; similar numerical optimization examples in AoS §9.13.4.
Core Properties of the MLE	
Overview	AoS §9.4 (list of properties).
Consistency	AoS §9.5 (Theorem 9.13 and related discussion).
Equivariance	AoS §9.6 (Theorem and Example).
Asymptotic Normality & Optimality	AoS §9.7 introduction, §9.8.
Fisher Information and Confidence Intervals	
Fisher Information	AoS §9.7 (Definitions and Theorem 9.17).
Constructing Confidence Intervals	AoS §9.7 (Theorem 9.19 and Examples).
Cramér-Rao Lower Bound	Mentioned in AoS §9.8.
Additional Topics	
The Delta Method	AoS §9.9 (Theorem 9.24 and Examples).
Multiparameter Models	AoS §9.10 (Fisher Information Matrix and related theorems).
Sufficient Statistics	AoS §9.13.2 (brief treatment from slides).
Connection to Machine Learning:	Expanded from slides.
Cross-Entropy as MLE	
MLE for Latent Variable Models: The EM Algorithm	AoS §9.13.4 (Appendix on computing MLEs).
Self-Test Problems	Inspired by AoS §9.14 exercises.

6.9.8 Further Materials

- **Connection to machine learning:** Murphy (2022), “Probabilistic Machine Learning: An Introduction”, Chapter 4

Remember: The MLE isn't just a computational procedure – it's a principled approach to estimation with deep theoretical foundations. The properties we've studied explain why maximum likelihood appears everywhere from simple coin flips to complex neural networks. These concepts represent the statistical foundations of modern machine learning!

Chapter 7

Hypothesis Testing and p-values

7.1 Learning Objectives

After completing this chapter, you will be able to:

- **Explain the core framework of null-hypothesis significance testing (NHST)**, including the roles of null/alternative hypotheses, Type I/II errors, and statistical power.
- **Define the p-value and correctly interpret its meaning**, recognizing its limitations and common misinterpretations.
- **Apply and interpret common hypothesis tests**, such as the Wald test and permutation test, and understand the connection between tests and confidence intervals.
- **Explain the multiple testing problem** and apply standard correction methods like the Bonferroni and Benjamini-Hochberg procedures.
- **Critically evaluate the use of NHST and p-values** in data analysis and scientific research.

i Note

This chapter covers null-hypothesis significance testing (NHST) and p-values, fundamental concepts in statistical inference. The material is adapted from Chapter 10 of Wasserman (2013), supplemented with computational examples and critical perspectives on the use of NHST in modern data science.

7.2 Introduction: Is an Observed Effect Real or Just Random Chance?

A drug company is running a clinical trial of a new drug. Patients are randomly assigned to either a treatment group that receives the new drug (100 subjects), or a control group that receives a placebo (other 100 subjects). After a suitable period of observation, the results are tallied:

	Better	Not Better
Treated	50	50
Control	40	60

Looking at these numbers, we see that 50% of the treated patients improved, compared to only 40% of the control patients. But is this 10 percentage point difference large enough to represent a *real* effect of the drug, or could it simply be due to random chance in how we happened to assign individuals to groups?

Similarly, an online retailer might run an A/B test comparing two different user interfaces. Users are randomly shown either the old system or a new design, and we track their purchase behavior:

	Purchase	No Purchase
New System	850	150
Old System	800	200

The new system appears to have a higher conversion rate (85% vs 80%), but again we face the fundamental question: is this difference statistically meaningful, or just random variation?

These questions lie at the heart of **Null-Hypothesis Significance Testing (NHST)**, one of the most widely used – and most debated¹ – frameworks in statistics. The core idea is to start by assuming there is *no effect* (the “null hypothesis”) and then ask how surprising our observed data would be under that assumption. If the data would be very surprising under the null, we have evidence against it.

This chapter builds the NHST framework from the ground up, introducing key concepts like p-values, statistical power, and the critical issue of multiple testing. We’ll see how these tools help us distinguish real effects from random noise, while also understanding their limitations and why they’re often misused in practice.

i Finnish Terminology Reference

For Finnish-speaking students, here’s a reference table of key terms in this chapter:

English	Finnish	Context
Null Hypothesis	Nollahypoteesi	The default assumption of no effect
Alternative Hypothesis	Vastahypoteesi, vaihtoehtoinen hypoteesi	What we hope to find evidence for
Simple Hypothesis	Yksinkertainen hypoteesi, pistehypoteesi	Hypothesis that completely specifies the distribution
Composite Hypothesis	Yhdistetty hypoteesi	Hypothesis that specifies a range of values
Two-sided Test	Kaksisuuntainen testi	Test detecting differences in either direction
One-sided Test	Yksitahoinen testi, yksisuuntainen testi	Test detecting differences in one direction
Rejection Region	Hylkäysalue	Set of outcomes leading to rejection
Test Statistic	Testisuure	Summary of evidence against null
Critical Value	Kriittinen arvo	Threshold for rejection
Type I Error	Hylkäysvirhe	False positive rejection
Type II Error	Hyväksymisvirhe	False negative (failure to detect)
Power	Voima	Probability of detecting true effect
Power function	Voimafunktio	Power as function of parameter
Size of a test	Testin koko	Maximum Type I error rate
Statistically significant	Tilastollisesti merkitsevä	Result unlikely under null

¹NHST has been central to scientific research for decades, but has also been subject to criticism regarding its misuse and misinterpretation. We’ll explore both its strengths and limitations throughout this chapter, particularly in the sections on interpreting p-values and the critical view of NHST in practice.

Wald Test	Waldin testi	Test based on asymptotic normality
Paired test	Parittainen testi	Test for dependent samples
Permutation test	Permutaatiotesti, satunnaistamistesti	Non-parametric test
Likelihood ratio statistic	Uskottavuusosamäärsuure	Ratio of likelihoods
Multiple testing	Monitestaus	Running many tests simultaneously
False discovery rate (FDR)	Väärien löydösten osuus	Expected proportion of false positives

7.3 The Framework of Hypothesis Testing

7.3.1 Null and Alternative Hypotheses

When we observe a difference between two groups or a pattern in data, we need a systematic way to determine whether this observation represents a genuine phenomenon or could simply be due to chance. Hypothesis testing provides this framework by setting up two competing explanations and evaluating the evidence against one of them.

Null Hypothesis (H_0): A statement of “no effect” or “no difference.” It’s the default assumption we seek to find evidence against.

Alternative Hypothesis (H_1): The statement we hope to find evidence for, typically representing the presence of an effect or difference.

For example, in our drug trial:

- H_0 : The drug has the same efficacy as the placebo (no effect)
- H_1 : The drug’s efficacy differs from the placebo

In the parametric framework we studied in Chapters 5–6, we can often formalize this by partitioning the parameter space Θ into two disjoint sets Θ_0 and Θ_1 , and testing:

$$H_0 : \theta \in \Theta_0 \quad \text{versus} \quad H_1 : \theta \in \Theta_1$$

The nature of the hypotheses determines the type of test:

Simple hypothesis: A hypothesis that completely specifies the distribution, such as $\theta = \theta_0$.

Composite hypothesis: A hypothesis that specifies a range of values, such as $\theta > \theta_0$ or $\theta < \theta_0$.

Two-sided test: Tests $H_0 : \theta = \theta_0$ versus $H_1 : \theta \neq \theta_0$ (detects differences in either direction).

One-sided test: Tests $H_0 : \theta \leq \theta_0$ versus $H_1 : \theta > \theta_0$ (or the reverse), detecting differences in a specific direction.

Most scientific applications use two-sided tests, as we’re typically interested in detecting any difference from the null, not just differences in a predetermined direction.

7.3.2 The Machinery of a Test

Once we’ve specified our hypotheses, we need a systematic procedure for deciding between them based on the observed data. This involves defining what outcomes would lead us to reject the null hypothesis.

Let X be a random variable with range \mathcal{X} .

Rejection Region ($R \subset \mathcal{X}$): The subset of outcomes for which we will reject H_0 . If $X \in R$, we reject the null hypothesis; otherwise, we retain it.²

Test Statistic ($T(X^n)$): A function of the data that summarizes the evidence against H_0 . Common examples include differences in sample means, ratios of variances, or correlation coefficients.

Critical Value (c): A threshold used to define the rejection region, often in terms of a test statistic, such as

$$R = \{x : T(x) > c\} \quad \text{or} \quad R = \{x : |T(x)| > c\}$$

Example: Testing Equality of Means

Suppose we have samples $X_1, \dots, X_n \sim F_X$ and $Y_1, \dots, Y_m \sim F_Y$ and want to test:

$$H_0 : \mathbb{E}(X) = \mathbb{E}(Y) \quad \text{versus} \quad H_1 : \mathbb{E}(X) \neq \mathbb{E}(Y)$$

The test statistic might be (a scaled version of) the difference in sample means:

$$T = \bar{X}_n - \bar{Y}_m$$

If $|T|$ is large, we have evidence against H_0 . The rejection region would be $R = \{(x_1, \dots, x_n, y_1, \dots, y_m) : |T| > c\}$ for some critical value c chosen to control the error rates, as explained in the next section.

7.3.3 Two Ways to Be Wrong: Type I and Type II Errors

When we make a decision based on data, we can make two types of errors. Understanding these errors is crucial for properly designing and interpreting hypothesis tests.

Type I Error: Rejecting H_0 when H_0 is true (false positive). The probability of Type I error is denoted α .

Type II Error: Failing to reject H_0 when H_1 is true (false negative). The probability of Type II error is denoted β .

The possible outcomes of a hypothesis test can be summarized as:

	H_0 True	H_0 False
H_0 Retained	Correct (True Negative)	Type II Error (False Negative)
H_0 Rejected	Type I Error (False Positive)	Correct (True Positive)

Key quantities for characterizing test performance:

Power Function: For a test with rejection region R , the power function is:

$$\beta(\theta) = \mathbb{P}_{\theta}(X \in R)$$

Size of a Test: The maximum probability of Type I error:

$$\alpha = \sup_{\theta \in \Theta_0} \beta(\theta)$$

Level of a Test: A test has level α if its size is at most α .

Power of a Test: The probability of correctly rejecting H_0 when it's false:

$$\text{Power} = 1 - \beta = \mathbb{P}_{\theta}(\text{Reject } H_0 \mid \theta \in \Theta_1)$$

²In classical hypothesis testing, we **never** “accept” the null hypothesis – we can only keep it because we haven’t found enough evidence to reject it.

There's an inherent trade-off between Type I and Type II errors: making it harder to commit a Type I error (lowering α) makes it easier to commit a Type II error (increasing β), thus reducing power. Standard practice is to fix the Type I error rate at a conventional level (typically $\alpha = 0.05$) and design the study to have adequate power (typically 80% or higher).

The relationship between these errors explains why we use asymmetric language: we "reject" or "fail to reject" H_0 , never "accept" it. Failing to reject doesn't prove H_0 is true – we might simply lack power to detect the effect!

Multiple Perspectives

Intuitive

Think of hypothesis testing like a criminal trial. The null hypothesis is "the defendant is innocent" – our default assumption until proven otherwise. We only reject this assumption (convict) if the evidence against innocence is very strong.

Just as a trial can go wrong in two ways, so can a hypothesis test:

- **Type I Error:** Convicting an innocent person (false positive)
- **Type II Error:** Acquitting a guilty person (false negative)

A common illustration uses the [Voight-Kampff test](#), which detects replicants (human-looking androids):

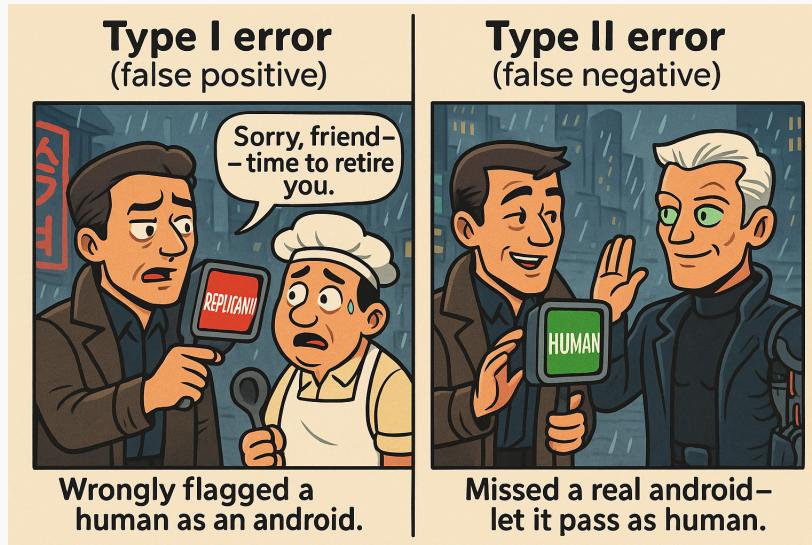


Figure 7.1: Type I and Type II Errors illustrated with replicant detection. Credits: [gpt-4o](#).

With H_0 : "The subject is human":

- **Type I Error:** Test says REPLICANT but subject is actually human (wrongly flagged as android).
- **Type II Error:** Test says HUMAN but subject is actually an android (missed detection).

Mathematical

Let's derive the power function for a concrete example. Consider testing the mean of a normal distribution:

Setup: We have $X_1, \dots, X_n \sim \mathcal{N}(\mu, \sigma^2)$ with known variance σ^2 , testing:

$$H_0 : \mu = 0 \quad \text{versus} \quad H_1 : \mu \neq 0$$

Test statistic: The sample mean $\bar{X} \sim \mathcal{N}(\mu, \sigma^2/n)$. Under H_0 where $\mu = 0$:

$$Z = \frac{\sqrt{n}\bar{X}}{\sigma} \sim \mathcal{N}(0, 1)$$

Decision rule: For a level α test, we reject H_0 when $|Z| > z_{\alpha/2}$, where $z_{\alpha/2}$ is the $(1 - \alpha/2)$ quantile of the standard normal distribution.

Power function: For any true value μ , the power is:

$$\beta(\mu) = \mathbb{P}_{\mu}(\text{Reject } H_0) = \mathbb{P}_{\mu}(|Z| > z_{\alpha/2})$$

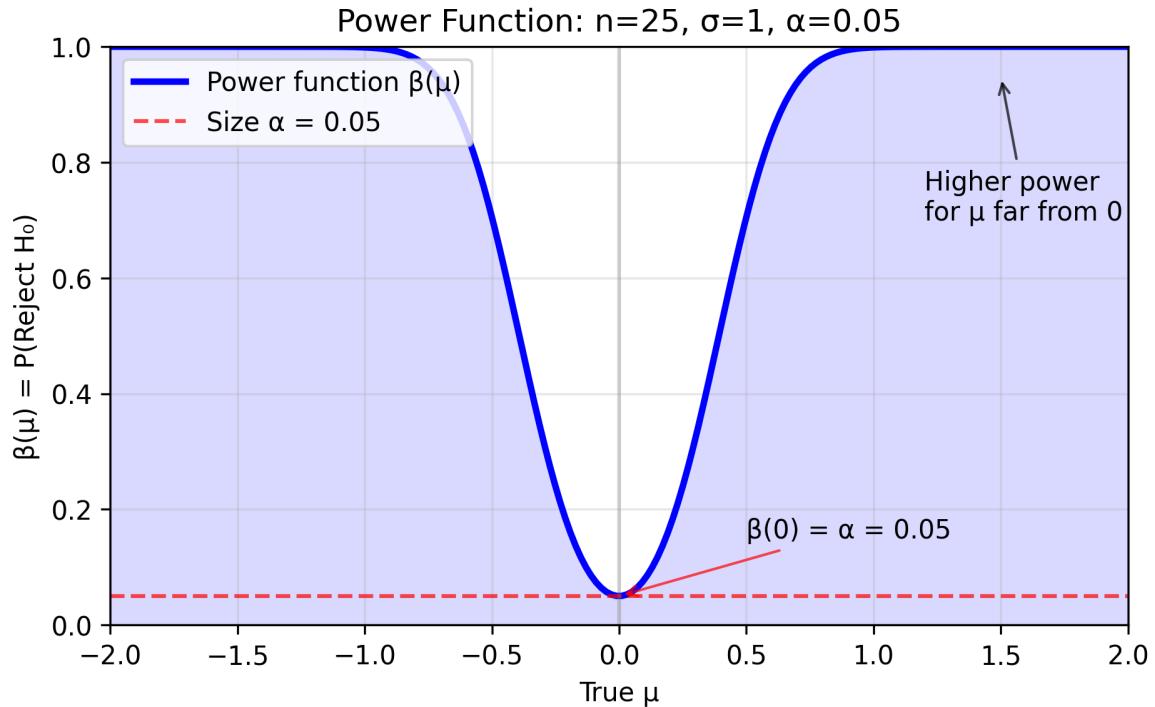
Under the true μ , the test statistic follows $Z \sim \mathcal{N}(\sqrt{n}\mu/\sigma, 1)$. Therefore:

$$\beta(\mu) = \mathbb{P} \left(\left| \mathcal{N} \left(\frac{\sqrt{n}\mu}{\sigma}, 1 \right) \right| > z_{\alpha/2} \right)$$

This probability depends on three key factors:

- **Effect size:** $\delta = \mu/\sigma$ (standardized distance from null)
- **Sample size:** Larger $n \rightarrow$ higher power
- **Significance level:** Larger $\alpha \rightarrow$ higher power (but more Type I errors)

Let's visualize this power function for $H_0 : \mu = 0, \sigma = 1, n = 25, \alpha = 0.05$:



The mirrored S-shaped curve demonstrates that power is minimized at the boundary of H_0 (here, $\mu = 0$) and increases monotonically as the true parameter moves away from the null value, making detecting the difference easier.

Computational

Let's simulate hypothesis testing to see Type I and Type II errors in action. We'll test $H_0 : \mu = 0$ using one-sample t-tests on data from normal distributions. By generating many datasets under different true means, we can observe the empirical error rates:

```

import numpy as np
from scipy import stats

# Simulate multiple hypothesis tests
np.random.seed(42)
n_tests = 10000
n_samples = 30
alpha = 0.05

# Scenario 1: H0 is true ( = 0)
# We should reject H0 about 5% of the time (Type I error rate)
type_i_errors = 0
for _ in range(n_tests):
    sample = np.random.normal(0, 1, n_samples) # H0 true:   = 0
    t_stat, p_value = stats.ttest_1samp(sample, 0)
    if p_value < alpha:
        type_i_errors += 1

print(f"Type I Error Rate (H true, =0):")
print(f"  Theoretical: {alpha:.3f}")
print(f"  Observed:   {type_i_errors/n_tests:.3f}")

# Scenario 2: H0 is false ( = 0.5)
# Power = probability of correctly rejecting H0
true_mu = 0.5
rejections = 0
for _ in range(n_tests):
    sample = np.random.normal(true_mu, 1, n_samples) # H0 false:   = 0.5
    t_stat, p_value = stats.ttest_1samp(sample, 0)
    if p_value < alpha:
        rejections += 1

power = rejections / n_tests
type_ii_error_rate = 1 - power

print(f"\nWhen H is false (true ={true_mu}):")
print(f"  Power (correct rejection):   {power:.3f}")
print(f"  Type II Error Rate (miss):   {type_ii_error_rate:.3f}")

Type I Error Rate (H true, =0):
Theoretical: 0.050
Observed:   0.056

When H is false (true =0.5):
Power (correct rejection):   0.750
Type II Error Rate (miss):   0.250
Factors affecting power:


- Effect size: Larger true differences from  $H_0 \rightarrow$  higher power
- Sample size: More data  $\rightarrow$  higher power
- Variability: Lower variance  $\rightarrow$  higher power
- Significance level: Higher  $\alpha \rightarrow$  higher power (but more Type I errors)

```

7.3.4 The Wald Test

The Wald test is one of the most widely used hypothesis tests in statistics. It leverages the asymptotic normality of estimators that we studied in Chapters 5-6, providing a general framework for testing hypotheses about parameters.

The key insight: If our estimator $\hat{\theta}$ is approximately normal (which many estimators are for large samples), we can use this normality to construct a test statistic with a known distribution under the null hypothesis.

The Wald Test: For testing $H_0 : \theta = \theta_0$ versus $H_1 : \theta \neq \theta_0$:

Assume that $\hat{\theta}$ is asymptotically normal:

$$\frac{(\hat{\theta} - \theta_0)}{\widehat{\text{se}}} \rightsquigarrow \mathcal{N}(0, 1)$$

The size α Wald test rejects H_0 when $|W| > z_{\alpha/2}$, where:

$$W = \frac{\hat{\theta} - \theta_0}{\widehat{\text{se}}}$$

Key Properties of the Wald Test:

1. **Correct error rate:** For large samples, the test has (approximately) the desired Type I error rate α
2. **Power increases with:**
 - Larger effect size (bigger difference from θ_0)
 - Larger sample size (more data)
 - Smaller variance (less noise)
3. **Duality with confidence intervals:** The Wald test rejects $H_0 : \theta = \theta_0$ if and only if θ_0 is outside the $(1 - \alpha)$ confidence interval

i Mathematical Details

1. **Asymptotic size:** As $n \rightarrow \infty$, $\mathbb{P}_{\theta_0}(|W| > z_{\alpha/2}) \rightarrow \alpha$
2. **Power function:** When the true parameter is $\theta_* \neq \theta_0$, the power is:

$$\text{Power}(\theta_*) = 1 - \Phi\left(\frac{\theta_0 - \theta_*}{\widehat{\text{se}}} + z_{\alpha/2}\right) + \Phi\left(\frac{\theta_0 - \theta_*}{\widehat{\text{se}}} - z_{\alpha/2}\right)$$

where Φ is the standard normal CDF.

3. **Confidence interval duality:** Reject H_0 if and only if $\theta_0 \notin [\hat{\theta} - z_{\alpha/2} \cdot \widehat{\text{se}}, \hat{\theta} + z_{\alpha/2} \cdot \widehat{\text{se}}]$

This last property provides a powerful duality: every confidence interval can be inverted to give a hypothesis test, and vice versa. This is why confidence intervals are often more informative than p-values alone – they show all parameter values that wouldn't be rejected.

Example: Comparing Two Means

The Wald test extends naturally to comparing means from two populations. For independent samples X_1, \dots, X_m and Y_1, \dots, Y_n with means μ_1 and μ_2 :

Testing $H_0 : \mu_1 = \mu_2$ (or $\delta = \mu_1 - \mu_2 = 0$):

- Estimator: $\hat{\delta} = \bar{X} - \bar{Y}$
- Standard error: $\widehat{\text{se}} = \sqrt{s_1^2/m + s_2^2/n}$
- Test statistic: $W = \hat{\delta}/\widehat{\text{se}}$

This is the basis for the famous two-sample t-test (which uses a t-distribution for small samples instead)

of the normal approximation).

7.3.5 Statistical and Scientific Significance

When H_0 is rejected, the result is called **statistically significant**. This does **not** mean that the result has any practical or scientific relevance! The effect found could be very tiny and negligible for all practical purpose.

Statistical vs. Scientific Significance

Statistical significance Scientific significance

Let's assume $\alpha = 0.05$ and $\theta_0 = 0$. In this case, we reject the null if the 95 % confidence interval (CI) excludes 0. A result can be:

- **Statistically significant but scientifically trivial:** The CI excludes 0 but is very close to it (where “very close” depends on the domain).
- **Statistically non-significant but scientifically important:** The CI includes 0 but also includes large, meaningful effects.

Always report confidence intervals alongside p-values (described in the next section) to show both statistical and practical significance!

Let's visualize this distinction:

```
import matplotlib.pyplot as plt

# Two example confidence intervals
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 4))

# Example 1: Statistically significant but small effect
theta0 = 0
ci1_lower, ci1_upper = 0.001, 0.003
ci1_center = (ci1_lower + ci1_upper) / 2

ax1.axvline(theta0, color='red', linestyle='--', linewidth=2, label='H :   = 0')
ax1.barh(0, ci1_upper - ci1_lower, left=ci1_lower, height=0.3,
         color='blue', alpha=0.5, label='95% CI')
ax1.plot(ci1_center, 0, 'ko', markersize=8, label='Estimate')
ax1.set_xlim(-0.01, 0.01)
ax1.set_ylim(-0.5, 0.5)
ax1.set_xlabel('Parameter value')
ax1.set_title('Statistically Significant\nbut Tiny Effect')
ax1.legend(loc='upper right')
ax1.set_yticks([])
ax1.grid(True, alpha=0.3)

# Example 2: Not significant but large potential effect
ci2_lower, ci2_upper = -0.5, 2.5
ci2_center = (ci2_lower + ci2_upper) / 2

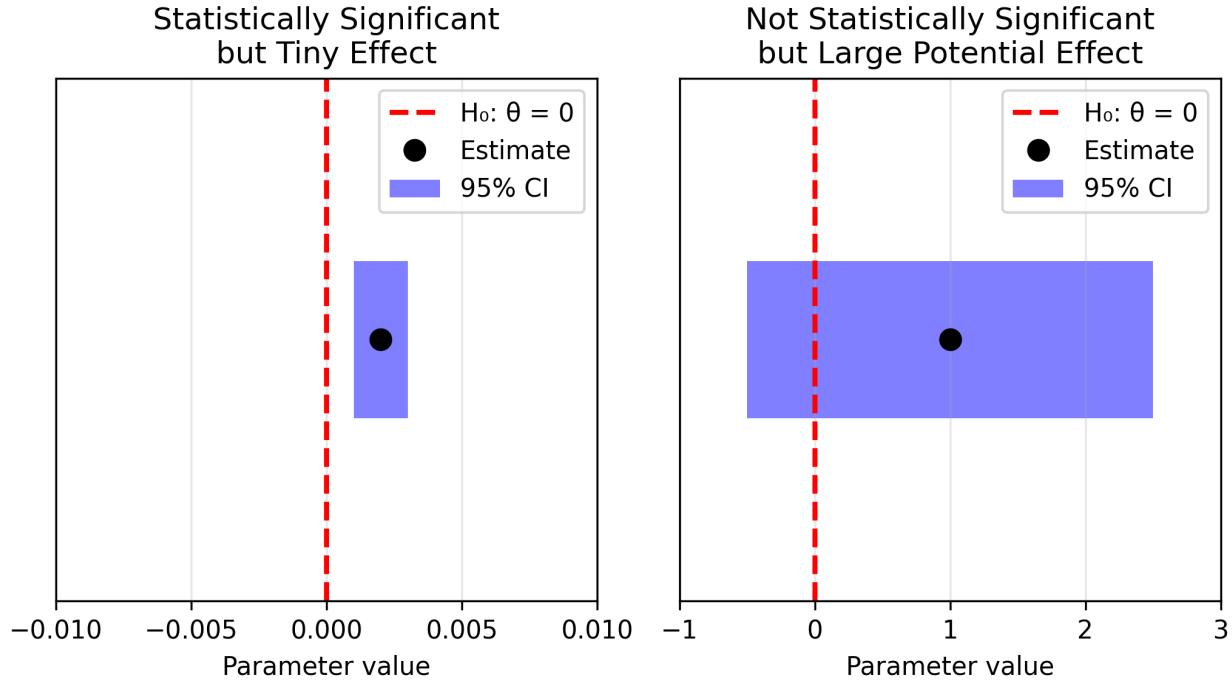
ax2.axvline(theta0, color='red', linestyle='--', linewidth=2, label='H :   = 0')
ax2.barh(0, ci2_upper - ci2_lower, left=ci2_lower, height=0.3,
         color='blue', alpha=0.5, label='95% CI')
ax2.plot(ci2_center, 0, 'ko', markersize=8, label='Estimate')
```

```

ax2.set_xlim(-1, 3)
ax2.set_ylim(-0.5, 0.5)
ax2.set_xlabel('Parameter value')
ax2.set_title('Not Statistically Significant\nbut Large Potential Effect')
ax2.legend(loc='upper right')
ax2.set_yticks([])
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```



The visualization demonstrates the CI-test duality: using significance level $\alpha = 0.05$, we construct a 95% CI (since $1 - \alpha = 0.95$). We reject $H_0 : \theta = 0$ if and only if 0 lies outside this interval.

Left panel: The 95% CI excludes 0, so we reject H_0 at the 5% level. But the effect is tiny (0.001 to 0.003) – statistically significant but practically negligible.

Right panel: The 95% CI includes 0, so we fail to reject H_0 . However, the interval also includes large values (up to 2.5) – we simply lack precision to determine if there's an effect.

7.4 The p-value: A Continuous Measure of Evidence

7.4.1 Understanding the p-value

Simply reporting “reject H_0 ” or “retain H_0 ” isn’t very informative – the result depends entirely on the chosen significance level α . What if we could provide a continuous measure of the strength of evidence against the null hypothesis? This is exactly what the p-value provides.

The **p-value** is the probability, calculated under H_0 , of observing a test statistic value at least as extreme as the one actually observed:

$$\text{p-value} = \mathbb{P}_{H_0}(T \text{ as extreme or more extreme than } T_{\text{obs}})$$

Equivalently, it's the smallest significance level α at which we would reject H_0 :

$$\text{p-value} = \inf\{\alpha : T(X^n) \in R_\alpha\}$$

where R_α is the rejection region for a size α test.

i Why are these two definitions equivalent?

The rejection region R_α contains the most extreme α proportion of possible test statistics. If our observed statistic is “more extreme than 2.1% of possible values” ($p = 0.021$), then:

- We'd reject for any $\alpha \geq 0.021$ (our observation is extreme enough)
- We'd fail to reject for any $\alpha < 0.021$ (our observation isn't extreme enough)

- Thus 0.021 is exactly the boundary – the smallest α for which we'd reject

The p-value acts as a threshold: it tells us both how extreme our observation is AND the critical significance level where our decision switches.

Key properties of p-values:

1. **Range:** p-values lie between 0 and 1
2. **Interpretation:** Small p-value = strong evidence against H_0
3. **Decision rule:** Reject H_0 if p-value $< \alpha$
4. **Under H_0 :** The p-value follows a Uniform(0, 1) distribution

Example: p-value for the Wald Test

For the Wald test we studied earlier, if $w = (\hat{\theta} - \theta_0)/\widehat{\text{se}}$ is the observed Wald statistic, then:

$$\text{p-value} = \mathbb{P}(|Z| > |w|) = 2\Phi(-|w|)$$

where $Z \sim \mathcal{N}(0, 1)$ and Φ is the standard normal CDF.

This formula works because under H_0 , the Wald statistic $W \sim \mathcal{N}(0, 1)$ asymptotically, so we calculate the probability of seeing a value as extreme as our observed w from a standard normal distribution.

Multiple Perspectives

Intuitive

The p-value is often called a “surprise index.” It answers the question: “If there were truly no effect (if H_0 were true), how likely would we be to see a result at least as extreme as the one we actually observed?”

Think of it this way: Imagine you suspect a coin is biased. You flip it 10 times and get 9 heads. The p-value asks: “If this were actually a fair coin, what's the probability of getting 9 or more heads in 10 flips?” If that probability is very small, you have strong evidence the coin isn't fair.

A small p-value means our data would be very surprising under the null hypothesis, providing evidence against it. A large p-value means our data is consistent with the null hypothesis (though this doesn't prove the null is true! Maybe we simply didn't collect enough data).

Mathematical

Let $T(X^n)$ be the test statistic and $t_{\text{obs}} = T(x^n)$ be its observed value from the data. The p-value formalizes the “surprise index” under the null hypothesis H_0 .

1. Simple Null Hypothesis ($H_0 : \theta = \theta_0$)

For a simple null, the p-value is the probability of observing a test statistic at least as extreme as t_{obs} . The definition of “extreme” depends on the alternative hypothesis H_1 :

- **Right-tailed test** ($H_1 : \theta > \theta_0$): Extreme means large values of T .

$$\text{p-value} = \mathbb{P}_{\theta_0}(T(X^n) \geq t_{\text{obs}})$$

- **Left-tailed test** ($H_1 : \theta < \theta_0$): Extreme means small values of T .

$$\text{p-value} = \mathbb{P}_{\theta_0}(T(X^n) \leq t_{\text{obs}})$$

- **Two-tailed test** ($H_1 : \theta \neq \theta_0$): Extreme means large values of $|T|$ (or similar symmetric measure).

$$\text{p-value} = \mathbb{P}_{\theta_0}(|T(X^n)| \geq |t_{\text{obs}}|)$$

2. Composite Null Hypothesis ($H_0 : \theta \in \Theta_0$)

When the null hypothesis is composite (e.g., $H_0 : \mu \leq 0$), there isn't a single distribution under H_0 . We need to find the probability of an extreme result under the “worst-case” scenario within Θ_0 – the one that makes our data look least surprising. This is achieved by taking the supremum (least upper bound) of the probability over all possible parameter values in Θ_0 .

For a right-tailed test, the formula is:

$$\text{p-value} = \sup_{\theta \in \Theta_0} \mathbb{P}_\theta(T(X^n) \geq t_{\text{obs}})$$

This ensures that if we reject when p-value $< \alpha$, the Type I error rate is controlled and does not exceed α for any $\theta \in \Theta_0$. For many standard tests, this supremum occurs at the boundary of Θ_0 (e.g., at $\mu = 0$ for $H_0 : \mu \leq 0$).

Computational

Let's visualize how the p-value is calculated for a two-sided test. We'll use the Wald test as our example, where the test statistic follows a standard normal distribution under H_0 .

For a two-sided test (e.g., $H_0 : \theta = \theta_0$ vs $H_1 : \theta \neq \theta_0$), “extreme” means far from zero in either direction. The p-value is the total probability in both tails beyond our observed test statistic:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Example: Observed test statistic value
w_obs = 2.3 # Our observed Wald statistic

# Create the standard normal distribution
z_values = np.linspace(-4, 4, 300)
pdf_values = stats.norm.pdf(z_values)

# Calculate p-value
p_value = 2 * stats.norm.cdf(-abs(w_obs))

plt.figure(figsize=(7, 4))
# Plot the distribution
plt.plot(z_values, pdf_values, 'b-', linewidth=2, label='Standard Normal')

# Shade the rejection regions (tails)
tail_right = z_values[z_values >= abs(w_obs)]
tail_left = z_values[z_values <= -abs(w_obs)]

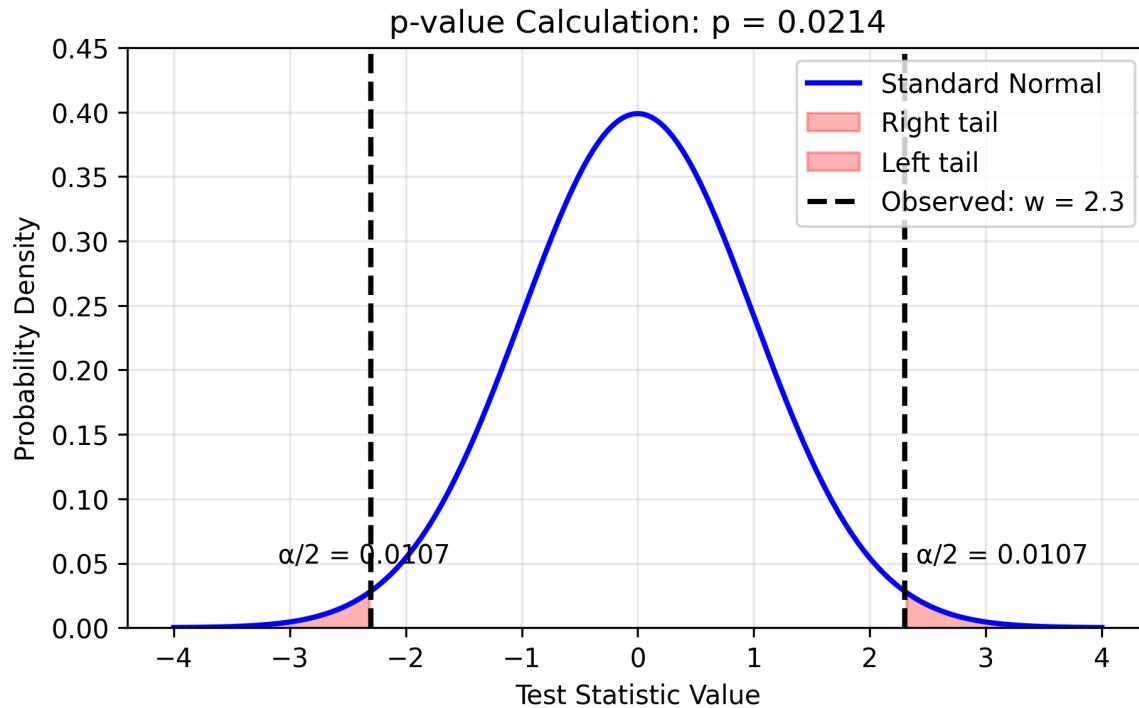
plt.fill_between(tail_right, 0, stats.norm.pdf(tail_right),
                 color='red', alpha=0.3, label=f'Right tail')
plt.fill_between(tail_left, 0, stats.norm.pdf(tail_left),
                 color='red', alpha=0.3, label=f'Left tail')

# Mark the observed values
plt.axvline(w_obs, color='black', linestyle='--', linewidth=2, label=f'Observed: w = {w_obs}')
plt.axvline(-w_obs, color='black', linestyle='--', linewidth=2)

# Add annotations
plt.text(w_obs + 0.1, 0.05, f'/2 = {p_value/2:.4f}', fontsize=10)
plt.text(-w_obs - 0.8, 0.05, f'/2 = {p_value/2:.4f}', fontsize=10)

plt.xlabel('Test Statistic Value')
plt.ylabel('Probability Density')
plt.title(f'p-value Calculation: p = {p_value:.4f}')
plt.legend(loc='upper right')
plt.grid(True, alpha=0.3)
plt.ylim(0, 0.45)
plt.show()

print(f"For observed test statistic w = {w_obs}:")
print(f"p-value = 2 * P(Z > |{w_obs}|) = {p_value:.4f}")
print(f"\nInterpretation: If H were true, we'd see a test statistic")
print(f"this extreme or more extreme only {p_value*100:.2f}% of the time.")
```



For observed test statistic $w = 2.3$:
 $p\text{-value} = 2 * P(Z > |2.3|) = 0.0214$

Interpretation: If H_0 were true, we'd see a test statistic this extreme or more extreme only 2.14% of the time.

7.4.2 How to Interpret p-values (and How Not To)

The p-value is one of the most misunderstood concepts in statistics. Let's clarify what it is and isn't.

What the p-value IS:

- The p-value IS the probability, computed under H_0 , of observing data as extreme or more extreme than what we actually observed
- The p-value IS a measure of evidence against H_0 – smaller values indicate stronger evidence
- The p-value IS the answer to: “If H_0 were true, how surprising would our data be?”
- The p-value IS useful for deciding whether to reject H_0 at any given significance level

A p-value of 0.02 means: If the null hypothesis were true, we'd see data this extreme or more extreme only 2% of the time. That's fairly surprising, suggesting the null might be false.

⚠ Common p-value Misinterpretations

A p-value is NOT the probability that the null hypothesis is true.

- Wrong: “ $p = 0.03$ means there's a 3% chance the null hypothesis is true”
- The p-value is $P(\text{data}|H_0)$, not $P(H_0|\text{data})$
- Computing $P(H_0|\text{data})$ requires Bayesian methods (Chapter 8)

A large p-value is NOT strong evidence that the null hypothesis is true.

- A large p-value could mean:
 1. H_0 is true, or
 2. H_0 is false but our test has low power to detect the effect

- Never conclude “we accept H_0 ” based on a large p-value

Statistical significance is NOT the same as practical significance.

- With enough data, tiny meaningless effects can become “statistically significant”
- Always examine the effect size (e.g., via confidence intervals) to judge practical importance
- Example: A drug that lowers blood pressure by 0.1 mmHg might be statistically significant with $n=10,000$ but clinically meaningless

i Standard p-value Interpretation Scales

The interpretation of p-values varies significantly by field and context. Here's a common scale used in many fields:

p-value	Evidence against H_0
< 0.01	Strong evidence
0.01 - 0.05	Positive evidence
> 0.05	Little or no evidence

Field-specific standards:

- **Medicine/Psychology:** Often use $\alpha = 0.05$ as the standard.
- **Genomics:** Use much stricter thresholds (e.g., 5×10^{-8}) due to multiple testing, as we will see below.
- **Particle Physics:** Extremely strict standards for discoveries:
 - The [Higgs boson discovery](#) required a “5-sigma” result ($p < 3 \times 10^{-7}$).
 - This corresponds to less than 1 in 3.5 million chance of a false positive.

These thresholds are conventions, not laws of nature. The appropriate threshold depends on the consequences of Type I and Type II errors in your specific context.

If the test statistic has a continuous distribution, then under $H_0 : \theta = \theta_0$, the p-value has a $\text{Uniform}(0,1)$ distribution. Therefore, if we reject H_0 when the p-value is less than α , the probability of a Type I error is exactly α .

This property means that p-values “work correctly” – under the null hypothesis, you'll get a p-value less than 0.05 exactly 5% of the time, a p-value less than 0.01 exactly 1% of the time, and so on.

7.4.3 Applying p-values: Classification Algorithm Comparison

Let's apply the Wald test and p-value concepts to a practical problem that appears frequently in machine learning and data science: comparing the performance of two classification algorithms. This example (from AoS Example 10.7) illustrates both independent and paired testing scenarios.

Example: Comparing Algorithms with Independent Test Sets

Suppose we test two classification algorithms on different, independent test sets:

- Algorithm 1: Test set of size m , makes X errors
- Algorithm 2: Test set of size n , makes Y errors

Then $X \sim \text{Binomial}(m, p_1)$ and $Y \sim \text{Binomial}(n, p_2)$, where p_1 and p_2 are the true error rates.

We want to test:

$$H_0 : p_1 = p_2 \quad \text{versus} \quad H_1 : p_1 \neq p_2$$

Or equivalently, $H_0 : \delta = 0$ versus $H_1 : \delta \neq 0$, where $\delta = p_1 - p_2$.

The Wald Test Approach:

The MLE is $\hat{\delta} = \hat{p}_1 - \hat{p}_2$ where $\hat{p}_1 = X/m$ and $\hat{p}_2 = Y/n$.

The estimated standard error is:

$$\widehat{se} = \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{m} + \frac{\hat{p}_2(1-\hat{p}_2)}{n}}$$

The Wald test statistic is:

$$W = \frac{\delta - 0}{\widehat{se}} = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{m} + \frac{\hat{p}_2(1-\hat{p}_2)}{n}}}$$

For a size α test, we reject H_0 when $|W| > z_{\alpha/2}$, and the p-value is $2\Phi(-|W|)$.

Numerical Example: Let $m = n = 500$, with Algorithm 1 making 75 errors and Algorithm 2 making 100 errors:

```
import numpy as np
from scipy import stats

# Independent test sets
m, n = 500, 500
p1_hat, p2_hat = 0.15, 0.20

# Standard error
se = np.sqrt(p1_hat*(1-p1_hat)/m + p2_hat*(1-p2_hat)/n)

# Wald test statistic
W = (p1_hat - p2_hat) / se

# p-value (two-sided)
p_value = 2 * stats.norm.cdf(-abs(W))

print(f"Error rates: Algorithm 1 = {p1_hat:.2%}, Algorithm 2 = {p2_hat:.2%}")
print(f"Difference: {p1_hat - p2_hat:.2%}")
print(f"Standard error: {se:.4f}")
print(f"Wald statistic: {W:.3f}")
print(f"p-value: {p_value:.4f}")
print(f"\nConclusion: {'Significant difference' if p_value < 0.05 else 'No significant difference'}")
```

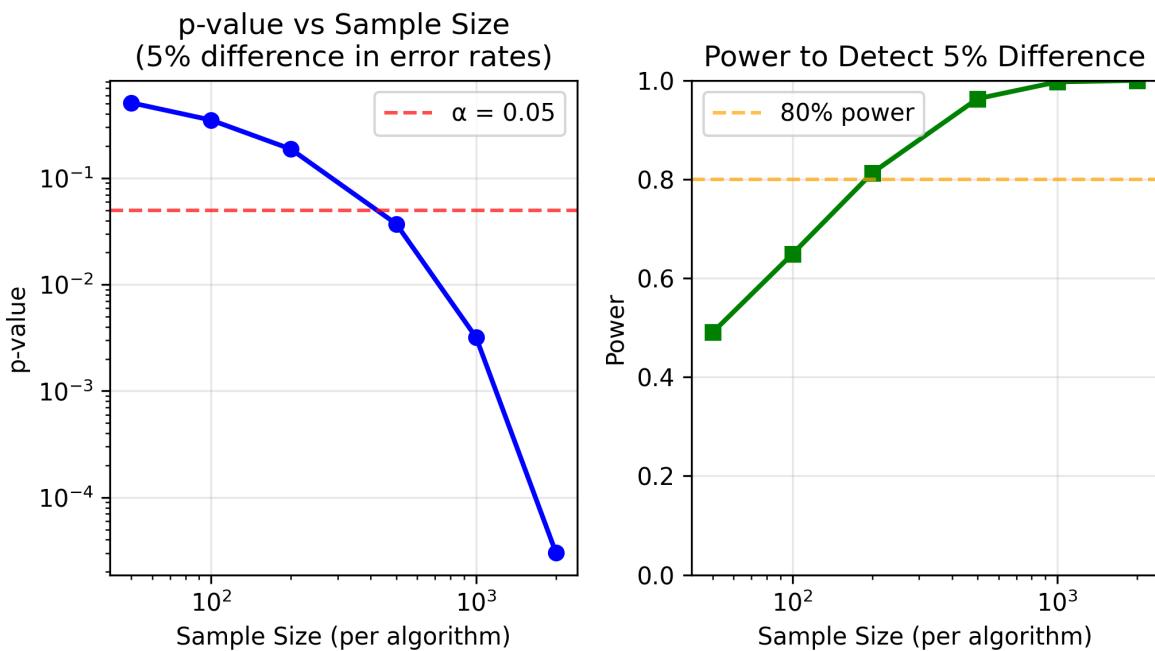
Error rates: Algorithm 1 = 15.00%, Algorithm 2 = 20.00%
Difference: -5.00%
Standard error: 0.0240
Wald statistic: -2.085
p-value: 0.0371

Conclusion: Significant difference at $= 0.05$

This 5 percentage point difference is statistically significant at $= 0.05$. But the significance depends critically on sample size. As a numerical illustration:

- With $m = n = 100$: Same 5% difference gives $|W| \approx 0.93$, p-value ≈ 0.35 (not significant)
- With $m = n = 500$: $|W| \approx 2.09$, p-value ≈ 0.037 (significant)
- With $m = n = 1000$: $|W| \approx 2.95$, p-value ≈ 0.003 (highly significant)

Let's visualize how sample size affects our ability to detect this difference:



Key Insight: The same 5% difference in error rates can be:

- Non-significant with small samples (low power)
- Highly significant with large samples (high power)

This illustrates why reporting effect sizes (the actual difference) alongside p-values is crucial!

Example: Comparing Algorithms with Paired Test Sets

Often we test both algorithms on the **same** test set. This is more efficient but requires a different analysis because the results are no longer independent.

Data Structure: For each test instance $i = 1, \dots, n$:

Test Case	Algorithm 1 (X_i)	Algorithm 2 (Y_i)	Difference ($D_i = X_i - Y_i$)
1	1 (correct)	0 (incorrect)	1
2	1 (correct)	1 (correct)	0
3	0 (incorrect)	1 (correct)	-1
...
n	X_n	Y_n	D_n

The key insight: We can no longer treat X and Y as independent because they're tested on the same instances.

The Paired Test Approach:

Define $D_i = X_i - Y_i$ for each test instance. Then:

$$\delta = \mathbb{E}(D_i) = \mathbb{E}(X_i) - \mathbb{E}(Y_i) = \mathbb{P}(X_i = 1) - \mathbb{P}(Y_i = 1)$$

We test $H_0 : \delta = 0$ versus $H_1 : \delta \neq 0$.

The nonparametric plug-in estimate is $\hat{\delta} = \bar{D} = \frac{1}{n} \sum_{i=1}^n D_i$.

The standard error is $\widehat{\text{se}}(\hat{\delta}) = S/\sqrt{n}$, where $S^2 = \frac{1}{n} \sum_{i=1}^n (D_i - \bar{D})^2$.

The Wald test statistic is $W = \hat{\delta}/\widehat{\text{se}}$ and we reject H_0 if $|W| > z_{\alpha/2}$.

This is called a **paired comparison** or **paired test**.

```
# Simulate paired comparison
np.random.seed(42)
n = 1000 # test set size

# True probabilities of correct classification
p1_true = 0.85 # Algorithm 1
p2_true = 0.80 # Algorithm 2

# Simulate correlated outcomes (algorithms often agree)
correlation = 0.7
from scipy.stats import multivariate_normal

# Generate correlated binary outcomes
cov = [[1, correlation], [correlation, 1]]
latent = multivariate_normal.rvs([0, 0], cov, size=n)
X = (latent[:, 0] < stats.norm.ppf(p1_true)).astype(int)
Y = (latent[:, 1] < stats.norm.ppf(p2_true)).astype(int)

# Compute differences
D = X - Y
D_bar = np.mean(D)
S = np.std(D, ddof=1)
se = S / np.sqrt(n)

# Wald test
W = D_bar / se
p_value = 2 * stats.norm.cdf(-abs(W))

print(f"Sample mean difference: {D_bar:.4f}")
print(f"Standard error: {se:.4f}")
print(f"Wald statistic: {W:.3f}")
print(f"p-value: {p_value:.4f}")

# Show contingency table
from pandas import crosstab, DataFrame
ct = crosstab(X, Y, rownames=['Alg1'], colnames=['Alg2'])
print("\nContingency table:")
print(ct)
print(f"\nAlgorithms agree on {np.sum(X == Y)/n:.1%} of instances")
Sample mean difference: 0.0580
```

```
Standard error: 0.0125
Wald statistic: 4.632
p-value: 0.0000
```

Contingency table:

	Alg2	0	1
Alg1	0	74	51
	1	109	766

Algorithms agree on 84.0% of instances

⚠ Paired vs Independent Tests

Paired tests are generally **more powerful** than independent tests when:

1. The same subjects/instances are measured twice
2. There's positive correlation between measurements

The paired test removes the between-subject variability, focusing only on within-subject differences.

7.5 Constructing Statistical Tests

Now that we understand p-values and have seen the Wald test in action, let's step back and consider the general problem: How do we construct a statistical test?

Given a test statistic $T(X^n)$ and null hypothesis H_0 , we need to determine the distribution of T under H_0 to calculate p-values and make decisions. There are three main approaches:

1. **Exact Distribution:** Sometimes we can calculate the exact distribution of T under H_0
 - Example: Fisher's exact test for 2×2 tables
 - Advantage: Exact p-values, valid for any sample size
 - Disadvantage: Only possible for simple cases
2. **Asymptotic Approximation:** Use limiting distributions as $n \rightarrow \infty$
 - Examples: Wald test (normal), likelihood ratio test (chi-squared)
 - Advantage: Widely applicable, computationally simple
 - Disadvantage: May be inaccurate for small samples
3. **Simulation/Resampling:** Simulate the distribution by resampling
 - Examples: Permutation test, bootstrap test
 - Advantage: Minimal assumptions, works for any test statistic
 - Disadvantage: Computationally intensive

The choice depends on:

- Sample size (small samples \rightarrow avoid asymptotics)
- Distributional assumptions (violated \rightarrow use resampling)
- Test statistic complexity (complex \rightarrow simulation may be only option)
- Computational resources (limited \rightarrow prefer analytical methods)

Let's now explore specific tests that exemplify each approach: the permutation test (simulation), Fisher's exact test (exact distribution), and the likelihood ratio test (asymptotic).

7.5.1 The Permutation Test: A Simulation Approach

The permutation test is a powerful non-parametric method for testing whether two distributions are the same. It exemplifies the simulation approach to test construction, making minimal assumptions – only requiring that the distributions are identical under the null hypothesis.

Permutation Test: A nonparametric method for testing whether two distributions are the same.

Let $X_1, \dots, X_m \sim F_X$ and $Y_1, \dots, Y_n \sim F_Y$ be independent samples.

Hypotheses: $H_0 : F_X = F_Y$ versus $H_1 : F_X \neq F_Y$

Test statistic: Choose any statistic $T(x_1, \dots, x_m, y_1, \dots, y_n)$, such as

$$T = |\bar{X}_m - \bar{Y}_n|$$

Key principle: Under H_0 , all $N! = (m + n)!$ permutations of the combined data are equally likely.

Permutation distribution: The distribution that puts mass $1/N!$ on each value T_j obtained from the $N!$ permutations.

Permutation p-value:

$$\text{p-value} = \mathbb{P}_0(T \geq t_{\text{obs}}) = \frac{1}{N!} \sum_{j=1}^{N!} I(T_j \geq t_{\text{obs}})$$

where t_{obs} is the observed test statistic and T_j is the statistic for permutation j .

The key insight: If the null hypothesis is true (both groups come from the same distribution), then the specific assignment of observations to groups was just one random possibility among many. We can simulate the null distribution by considering all possible reassessments.

When to use permutation tests:

- Small sample sizes where asymptotic approximations may fail
- Non-standard or unknown distributions
- Complex test statistics without known distributions
- When exact p-values are needed for critical decisions

i Practical Permutation Test Algorithm

Since evaluating all $(m + n)!$ permutations is computationally infeasible for realistic sample sizes, we use Monte Carlo sampling:

1. **Compute observed test statistic:** $t_{\text{obs}} = T(X_1, \dots, X_m, Y_1, \dots, Y_n)$
2. **Generate permutation distribution:** For B iterations (typically 10,000+):
 - Randomly permute the combined data
 - Split into groups of original sizes m and n
 - Recompute the test statistic T_i
3. **Calculate p-value:**

$$\text{p-value} \approx \frac{1 + \sum_{i=1}^B I(T_i \geq t_{\text{obs}})}{B + 1}$$

The “+1” corrections in both numerator and denominator:

- Prevent p-values of exactly 0 (which would be misleading)
- Make the estimate slightly conservative but consistent (as $B \rightarrow \infty$, it converges to the true p-value)
- Can be interpreted as treating the observed data arrangement as one of $B+1$ total permutations (since it's a valid permutation that always has $T = t_{\text{obs}}$)

Multiple Perspectives

Intuitive

The core idea of the permutation test is quite simple: if the group labels (e.g., “Treatment” vs

“Control”) don’t actually matter – that is, if the null hypothesis is true – then shuffling these labels randomly shouldn’t change the distribution of our test statistic.

Here’s the logic:

1. Under H_0 , the treatment and control groups come from the same distribution
2. So the specific assignment of units to groups was just one random possibility
3. We can simulate other equally likely assignments by shuffling the labels
4. If our observed difference is unusual compared to these shuffled differences, we have evidence against H_0

Mathematical

The mathematical foundation of the permutation test is the principle of **exchangeability**. Under the null hypothesis $H_0 : F_X = F_Y$, all observations $(X_1, \dots, X_m, Y_1, \dots, Y_n)$ are independent and identically distributed (IID) from the same underlying distribution. This implies that the group labels (“Treatment”, “Control”) are arbitrary; any permutation of the combined data is equally likely to have occurred.

The test leverages this property to construct an **exact, non-parametric reference distribution** for a chosen test statistic T under H_0 :

1. **The Permutation Distribution:** Conceptually, we consider the set of all $N! = (m + n)!$ possible permutations of the combined data. For each permutation, we compute the test statistic. The set of these $N!$ values forms the *exact* distribution of T under the null hypothesis, conditional on the observed data values.
2. **Exactness:** Because this distribution is derived directly from the data without asymptotic approximations, the permutation test is an **exact test**. This means that for a chosen significance level α , the Type I error rate is controlled at exactly α (with proper handling of discrete data). This is a significant advantage over asymptotic tests like the Wald test, which are only guaranteed to have the correct size as $n \rightarrow \infty$.
3. **Formal p-value:** The exact p-value is the proportion of permutations that yield a test statistic value as extreme or more extreme than the one observed with the original data labelling:

$$\text{p-value} = \frac{\#\{\text{permutations } \pi : T(\pi(\text{data})) \geq t_{\text{obs}}\}}{N!}$$

4. **Monte Carlo Approximation:** Since calculating all $N!$ statistics is computationally infeasible, the algorithm in the “Computational” tab uses Monte Carlo sampling to approximate this exact p-value. By drawing a large number of random permutations (B), we create an empirical distribution that converges to the true permutation distribution as $B \rightarrow \infty$.

Computational

Let’s implement a permutation test and see it in action. We’ll test whether two groups have different means, comparing our permutation test results with the standard parametric t-test. This demonstrates both how the test works and when it differs from classical approaches.

Since evaluating all $N!$ permutations is computationally infeasible for realistic sample sizes (recall that $20! \approx 2.4 \times 10^{18}$), we’ll use Monte Carlo sampling to approximate the permutation distribution with $B = 10,000$ random permutations.

```

import numpy as np
from scipy import stats

def permutation_test(x, y, n_permutations=10000):
    """
    Perform a two-sample permutation test.
    H0: The two samples come from the same distribution.
    Test statistic: Absolute difference in means.
    """
    # Observed test statistic
    t_obs = abs(np.mean(x) - np.mean(y))

    # Combine the data
    combined = np.concatenate([x, y])
    n_x = len(x)

    # Generate permutations and compute test statistics
    t_perm = []
    np.random.seed(42) # For reproducibility

    for _ in range(n_permutations):
        # Randomly permute the combined data
        permuted = np.random.permutation(combined)
        # Split into two groups of original sizes
        x_perm = permuted[:n_x]
        y_perm = permuted[n_x:]
        # Compute test statistic for this permutation
        t = abs(np.mean(x_perm) - np.mean(y_perm))
        t_perm.append(t)

    # Calculate p-value (with +1 correction)
    # Adding 1 to numerator and denominator for unbiased estimate
    p_value = (np.sum(np.array(t_perm) >= t_obs) + 1) / (n_permutations + 1)

    return t_obs, t_perm, p_value

# Example: Compare two small samples
np.random.seed(123)
group1 = np.random.normal(100, 15, 12) # Mean 100
group2 = np.random.normal(110, 15, 10) # Mean 110

t_obs, t_perm, p_value = permutation_test(group1, group2)

print(f"Observed difference in means: {t_obs:.2f}")
print(f"Permutation p-value: {p_value:.4f}")

# For comparison, also run parametric t-test
t_stat, p_parametric = stats.ttest_ind(group1, group2)
print(f"Parametric t-test p-value: {p_parametric:.4f}")

Observed difference in means: 26.31
Permutation p-value: 0.0016
Parametric t-test p-value: 0.0017

```

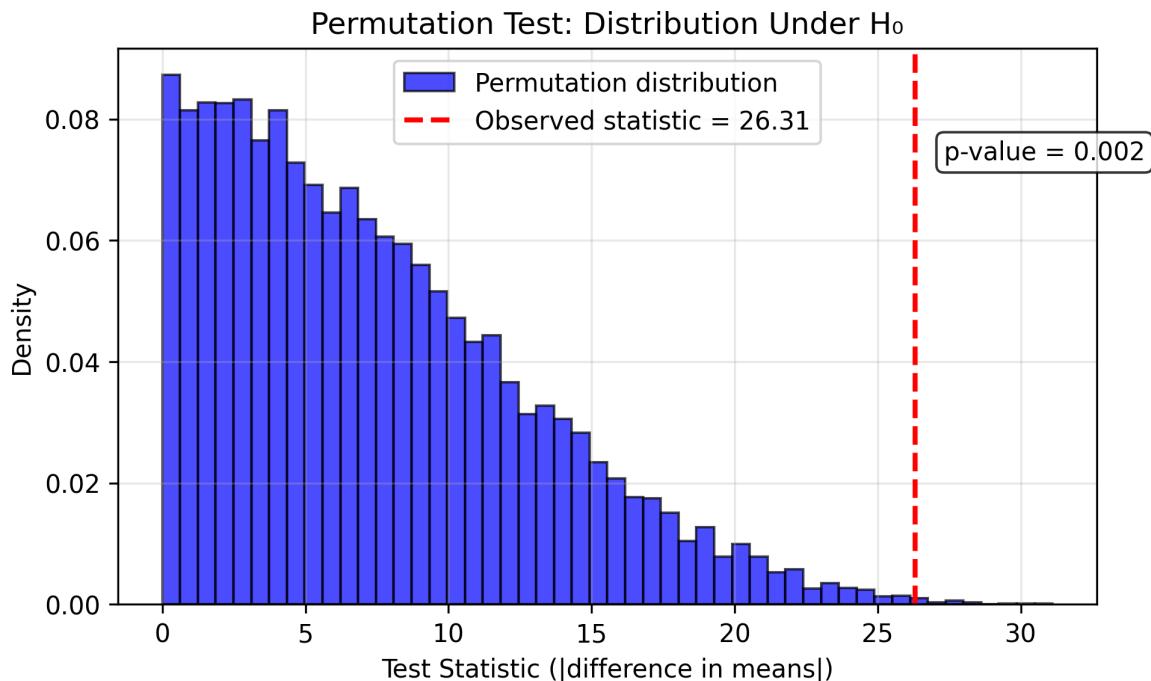
Let's visualize the permutation distribution to see what we've created:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(7, 4))
plt.hist(t_perm, bins=50, density=True, alpha=0.7,
         color='blue', edgecolor='black', label='Permutation distribution')
plt.axvline(t_obs, color='red', linestyle='--', linewidth=2,
            label=f'Observed statistic = {t_obs:.2f}')

# Add text annotation for p-value
plt.text(t_obs + 1, plt.ylim()[1] * 0.8,
          f'p-value = {p_value:.3f}',
          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.xlabel('Test Statistic (|difference in means|)')
plt.ylabel('Density')
plt.title('Permutation Test: Distribution Under H0')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



What this demonstration shows:

1. **The permutation distribution shows typical chance differences:** Under H_0 , we see what absolute differences in means we'd expect purely by chance when randomly assigning labels. Since we use the absolute difference, all values are positive, with most falling between 0 and 20.
2. **Our observed statistic falls in the far right tail:** The red dashed line shows our actual observed difference is larger than what we'd typically see by chance. The p-value annotation shows that only about 0.2% of permutations produce a difference this large or larger, providing evidence against H_0 .

3. **Agreement with the parametric test:** Both the permutation test ($p = 0.016$) and the t-test ($p = 0.017$) reach similar conclusions. This is reassuring when assumptions hold, but the permutation test would still be valid even if normality assumptions were violated.
4. **The Monte Carlo approximation works well:** With 10,000 permutations, we get a smooth estimate of the true permutation distribution, making the test both computationally feasible and statistically reliable. The distribution has the characteristic shape for an absolute difference statistic – starting near zero and skewing right – but the permutation test makes no assumptions about this shape.

⚠️ Use \geq not $>$ for discrete distributions in permutation tests

Wasserman (2013) uses $I(T_j > t_{\text{obs}})$ but this is incorrect for discrete distributions. The definition of p-value includes equality: the probability of observing a value “as extreme or more extreme.” For continuous distributions this makes no difference, but for discrete cases (including permutation tests) we must use \geq .

Example illustrating why this matters:

Consider a tiny dataset: $X = (1, 9)$ and $Y = (3)$. We want to test if the means are different using $T = |\bar{X} - \bar{Y}|$ as our test statistic.

- Observed: $\bar{X} = 5$, $\bar{Y} = 3$, so $t_{\text{obs}} = |5 - 3| = 2$

Now let’s enumerate all 6 possible permutations:

Permutation	X values	Y value	\bar{X}	\bar{Y}	$T = \bar{X} - \bar{Y} $
Original	(1, 9)	(3)	5	3	2
2	(9, 1)	(3)	5	3	2
3	(1, 3)	(9)	2	9	7
4	(3, 1)	(9)	2	9	7
5	(3, 9)	(1)	6	1	5
6	(9, 3)	(1)	6	1	5

Computing the p-value:

- **Correct** (using \geq): $P(T \geq 2) = 6/6 = 1.0$ (all permutations have $T \geq 2$)
- **Incorrect** (using $>$): $P(T > 2) = 4/6 = 0.67$ (only 4 permutations have $T > 2$)

The correct p-value of 1.0 tells us our observed difference is the **smallest possible** – not evidence against H_0 at all! The incorrect formula would suggest some evidence against the null, which is completely wrong.

This example shows why the correct formula must use $I(T_j \geq t_{\text{obs}})$.

7.5.2 Fisher’s Exact Test: An Exact Approach

Fisher’s exact test exemplifies the exact distribution approach. It provides the exact probability of observing data as extreme or more extreme than what we observed, given fixed marginal totals in a 2×2 contingency table. This is particularly valuable for small sample sizes where asymptotic approximations may fail.

To illustrate, let’s return to the motivating drug trial problem from the chapter introduction:

i Application: The Drug Trial

Recall our 2×2 table of outcomes:

	Better	Not Better
Treated	50	50
Control	40	60

Many different statistical tests could be applied in this setting – we could use a two-sample test of proportions (Wald test), a chi-squared test, or a permutation test. However, for 2×2 contingency tables with modest sample sizes, **Fisher's Exact Test** is one of the more attractive alternatives. It calculates the exact probability of observing a table as extreme or more extreme than this, given fixed marginal totals.

```
import scipy.stats as stats

# Create the contingency table
table = [[50, 50], # Treated: 50 better, 50 not better
          [40, 60]] # Control: 40 better, 60 not better

# Fisher's exact test
# alternative="greater" tests if treatment has higher "better" rate
odds_ratio, p_value = stats.fisher_exact(table, alternative="greater")

print(f"Contingency table:")
print(f"      Better  Not Better")
print(f"Treated:  50      50")
print(f"Control:  40      60")
print(f"\nOdds ratio: {odds_ratio:.3f}")
print(f"One-sided p-value: {p_value:.4f}")

# Also try two-sided test
_, p_two_sided = stats.fisher_exact(table, alternative="two-sided")
print(f"Two-sided p-value: {p_two_sided:.4f}")

if p_value < 0.05:
    print("\nConclusion: Significant evidence of treatment effect (p < 0.05)")
else:
    print("\nConclusion: No significant evidence of treatment effect at = 0.05")

Contingency table:
      Better  Not Better
Treated:  50      50
Control:  40      60

Odds ratio: 1.500
One-sided p-value: 0.1004
Two-sided p-value: 0.2007

Conclusion: No significant evidence of treatment effect at = 0.05
With  $p = 0.10$ , we don't have strong evidence to reject the null hypothesis at the conventional  $\alpha = 0.05$  level. The observed 10 percentage point difference could plausibly arise by chance.
This example illustrates a key point: even seemingly large differences (50% vs 40% success rate) may not be statistically significant with modest sample sizes. Power analysis before conducting studies is crucial!
```

Many other applications share this same 2×2 structure:

- **A/B testing:** Comparing conversion rates between website designs
- **Demographics:** Testing differences in proportions between groups
- **Medical screening:** Comparing test accuracy between methods
- **Quality control:** Comparing defect rates between processes

Whenever you're comparing binary outcomes between two groups, you face the same statistical question: is the observed difference real or just chance?

7.5.3 The Likelihood Ratio Test: A General Asymptotic Approach

The Wald test is useful for testing a scalar parameter. The likelihood ratio test is more general and can be used for testing a vector-valued parameter, making it one of the most important tools in statistical inference.

Likelihood Ratio Test: For testing $H_0 : \theta \in \Theta_0$ versus $H_1 : \theta \notin \Theta_0$:

The **likelihood ratio statistic** is:

$$\lambda = 2 \log \left(\frac{\sup_{\theta \in \Theta} \mathcal{L}(\theta)}{\sup_{\theta \in \Theta_0} \mathcal{L}(\theta)} \right) = 2[\ell(\hat{\theta}) - \ell(\hat{\theta}_0)]$$

where:

- $\hat{\theta}$ is the MLE over the entire parameter space Θ
- $\hat{\theta}_0$ is the MLE under the constraint $\theta \in \Theta_0$

i Why use Θ instead of Θ_0^c in the numerator?

You might expect to maximize over Θ_0^c (the alternative hypothesis space) in the numerator. However:

- Using Θ has little practical effect on the test statistic
- The theoretical properties are much simpler with this definition
- It ensures the statistic is always non-negative

The LRT is most useful when Θ_0 is defined by constraining some parameters to fixed values. For example, if $\theta = (\theta_1, \dots, \theta_r)$ and we want to test that the last $r - q$ components equal specific values:

$$\Theta_0 = \{\theta : (\theta_{q+1}, \dots, \theta_r) = (\theta_{0,q+1}, \dots, \theta_{0,r})\}$$

Under $H_0 : \theta \in \Theta_0$, the likelihood ratio statistic has an asymptotic [chi-squared distribution](#):

$$\lambda \rightsquigarrow \chi_{r-q}^2$$

where $r - q$ is the difference in dimensionality between Θ and Θ_0 (the number of constraints imposed by H_0).

The p-value is: $\mathbb{P}(\chi_{r-q}^2 \geq \lambda)$

Example: Counting Degrees of Freedom

If $\theta = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$ and we test:

$$H_0 : \theta_4 = \theta_5 = 0$$

Then:

- Dimension of $\Theta = 5$ (all parameters free)
- Dimension of $\Theta_0 = 3$ (only $\theta_1, \theta_2, \theta_3$ free)
- Degrees of freedom = $5 - 3 = 2$

The test statistic $\lambda \rightsquigarrow \chi_2^2$ under H_0 .

7.6 The Multiple Testing Problem: The Peril of Many Tests

7.6.1 The Problem

Modern data science often involves testing many hypotheses simultaneously. Consider these scenarios:

- **Genomics:** Testing thousands of genes for association with disease
- **A/B testing:** Running dozens of experiments across a website
- **Neuroscience:** Testing brain activity at thousands of voxels
- **Feature selection:** Testing which of hundreds of features predict an outcome

The problem is simple but severe: If you perform many tests, you're virtually guaranteed to get false positives by chance alone.

Let's quantify this:

- **One test** at $\alpha = 0.05$: 5% chance of a Type I error
- **20 independent tests** at $\alpha = 0.05$ each:
 - Probability of at least one Type I error = $1 - (0.95)^{20} \approx 0.64$
 - That's a 64% chance of at least one false positive!
- **1000 tests**: Virtually certain to get many false positives

```
import numpy as np
import matplotlib.pyplot as plt

# Simulate the multiple testing problem
np.random.seed(42)
n_tests = [1, 5, 10, 20, 50, 100, 200, 500, 1000]
alpha = 0.05

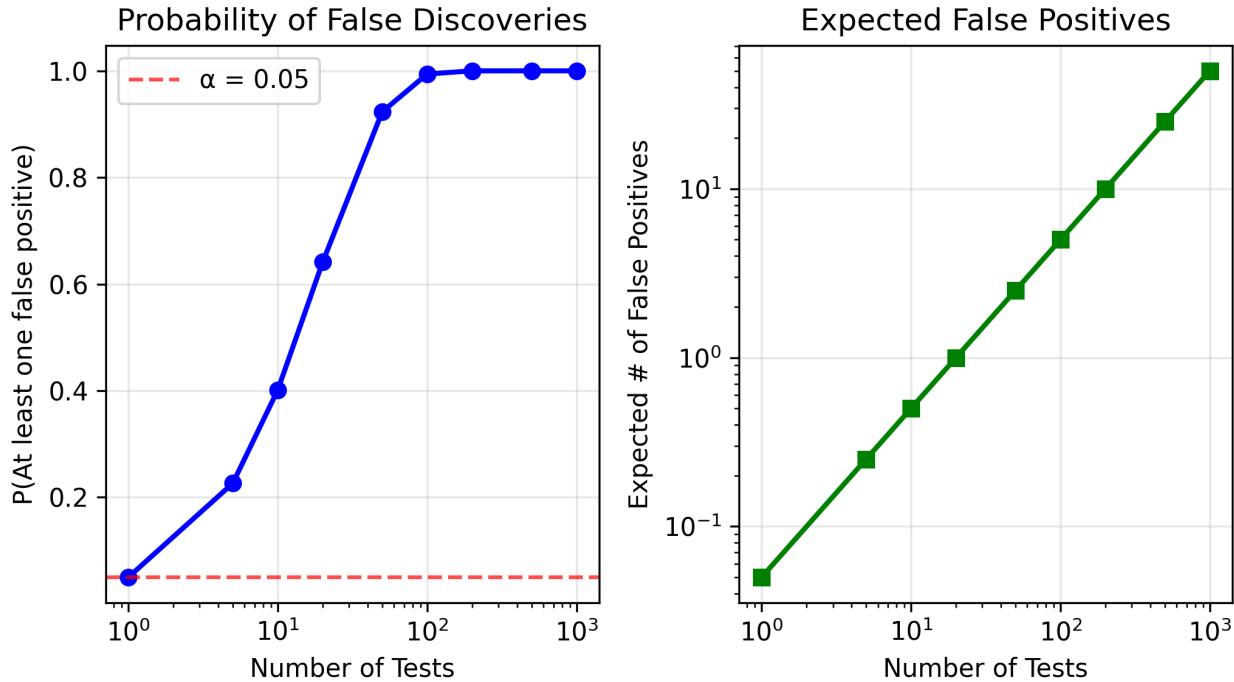
# Calculate probability of at least one false positive
prob_no_false_positive = [(1 - alpha)**m for m in n_tests]
prob_at_least_one_fp = [1 - p for p in prob_no_false_positive]

# Expected number of false positives
expected_fp = [m * alpha for m in n_tests]

plt.figure(figsize=(7, 4))
plt.subplot(1, 2, 1)
plt.plot(n_tests, prob_at_least_one_fp, 'b-', linewidth=2, marker='o')
plt.axhline(y=alpha, color='r', linestyle='--', alpha=0.7, label=f' = {alpha}')
plt.xlabel('Number of Tests')
plt.ylabel('P(At least one false positive)')
plt.title('Probability of False Discoveries')
plt.xscale('log')
plt.grid(True, alpha=0.3)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(n_tests, expected_fp, 'g-', linewidth=2, marker='s')
plt.xlabel('Number of Tests')
plt.ylabel('Expected # of False Positives')
plt.title('Expected False Positives')
plt.xscale('log')
plt.yscale('log')
plt.grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```



7.6.2 Key Concepts for Multiple Testing

When conducting m hypothesis tests, we need to understand what types of errors we want to control. Consider this classification table of test outcomes:

	H_0 Not Rejected	H_0 Rejected	Total
H_0 True	U	V	m_0
H_0 False	T	S	m_1
Total	$m - R$	R	m

Where:

- V = Number of false positives (Type I errors)
- S = Number of true positives (correct rejections)
- $R = V + S$ = Total rejections
- m_0 = Number of true null hypotheses

Family-Wise Error Rate (FWER): The probability of making at least one Type I error among all tests:

$$\text{FWER} = \mathbb{P}(V \geq 1)$$

False Discovery Proportion (FDP): The proportion of rejections that are false:

$$\text{FDP} = \begin{cases} \frac{V}{R} & \text{if } R > 0 \\ 0 & \text{if } R = 0 \end{cases}$$

False Discovery Rate (FDR): The expected false discovery proportion:

$$\text{FDR} = \mathbb{E}[\text{FDP}]$$

These quantities represent different philosophies for error control:

- **FWER control** is very conservative, aiming to avoid any false positives.
- **FDR control** is more liberal and often more sensible in practical applications, accepting some false positives but controlling their overall proportion.

7.6.3 The Bonferroni Method: Controlling FWER

The Bonferroni correction provides the simplest and most conservative approach to multiple testing.

Bonferroni Method: Given p-values P_1, \dots, P_m , reject null hypothesis H_{0i} if:

$$P_i < \frac{\alpha}{m}$$

Intuition: We divide our total error budget α equally among all m tests. If each test has Type I error rate α/m , the total error rate can't exceed α .

Using the Bonferroni method, the probability of falsely rejecting any null hypothesis is at most α :

$$\text{FWER} \leq \alpha$$

Proof: Let A_i be the event that test i rejects when H_{0i} is true. Then:

$$\text{FWER} = \mathbb{P}\left(\bigcup_{i \in I_0} A_i\right) \leq \sum_{i \in I_0} \mathbb{P}(A_i) \leq \sum_{i \in I_0} \frac{\alpha}{m} \leq \alpha$$

where I_0 is the set of true null hypotheses.

Pros:

- Simple to implement
- Provides strong control of Type I errors
- Works for any dependency structure among tests

Cons:

- **Extremely conservative:** Dramatically reduces power
- Often fails to detect true effects
- Becomes nearly useless with thousands of tests

7.6.4 The Benjamini-Hochberg Method: Controlling FDR

The Benjamini-Hochberg (BH) procedure offers a more powerful alternative by controlling the proportion of false discoveries rather than the probability of any false discovery.

Benjamini-Hochberg (BH) Procedure:

1. Let $P_{(1)} \leq P_{(2)} \leq \dots \leq P_{(m)}$ denote the ordered p-values
2. Define:

$$\ell_i = \frac{i\alpha}{C_m m} \quad \text{and} \quad R = \max\{i : P_{(i)} \leq \ell_i\}$$

where C_m is defined as:

- $C_m = 1$ if the p-values are independent

- $C_m = \sum_{i=1}^m \frac{1}{i}$ otherwise (for dependent tests)
- Let $T = P_{(R)}$ be the **BH rejection threshold**
 - Reject all null hypotheses H_{0i} for which $P_i \leq T$

Intuition: The procedure finds the largest set of rejections such that the expected proportion of false discoveries is controlled. In practice, $C_m = 1$ is almost always used (assuming independence), so the threshold for the i -th smallest p-value is $\ell_i = i\alpha/m$. The procedure looks for the rightmost p-value that falls below this sloped line.

The procedure can be visualized by plotting ordered p-values against their threshold line:

```
# Visualize BH procedure with a clearer example
np.random.seed(42)
m = 20 # Number of tests
alpha_fdr = 0.05

# Generate p-values designed to show BH advantage
# With =0.05 and m=20: Bonferroni threshold = 0.0025, BH thresholds = i*0.0025
p_values = np.array([
    0.0008, 0.0045, 0.0068, # BH will reject these 3 (below 0.0025, 0.005, 0.0075)
    0.012, 0.024, 0.041, 0.063, 0.089, # Won't be rejected
    0.115, 0.181, 0.224, 0.301, # Medium p-values
    0.412, 0.501, 0.578, 0.656, # Larger p-values
    0.721, 0.812, 0.888, 0.951 # Very large p-values
])
p_sorted = np.sort(p_values)

# BH threshold line (using C_m = 1 for independent tests)
k_values = np.arange(1, m+1)
bh_threshold = k_values * alpha_fdr / m # This is _i with C_m = 1

# Find BH cutoff using the standard definition
# R = max{i: P_(i) <= _i}
bh_check = p_sorted <= bh_threshold
if np.any(bh_check):
    R = np.max(np.where(bh_check)[0]) + 1 # +1 because Python uses 0-indexing
    T = p_sorted[R-1] # BH rejection threshold
else:
    R = 0
    T = 0

plt.figure(figsize=(7, 4))

# Plot p-values and thresholds
plt.scatter(k_values, p_sorted, color='blue', s=60, label='Sorted p-values', zorder=3)
plt.plot(k_values, bh_threshold, 'r--', linewidth=2, label=f'BH threshold (= {alpha_fdr})', zorder=2)
plt.plot(k_values, [alpha_fdr/m]*m, 'g--', alpha=0.7, linewidth=1.5, label='Bonferroni threshold', zorder=1)

if R > 0:
    # Highlight rejected p-values
    plt.scatter(k_values[:R], p_sorted[:R],
                color='red', s=100, marker='s', label='Rejected', zorder=4)
    # Mark the cutoff point R
    plt.axvline(R + 0.5, color='black', linestyle=':', alpha=0.4, linewidth=1)
```

```

plt.text(R + 0.7, 0.15, f'R = {R}', fontsize=9, color='black')

# Formatting
plt.xlabel('Rank i')
plt.ylabel('p-value (log scale)')
plt.title('Benjamini-Hochberg Procedure')
plt.legend(loc='upper left', framealpha=0.95)
plt.grid(True, alpha=0.3, linewidth=0.5, which='both')

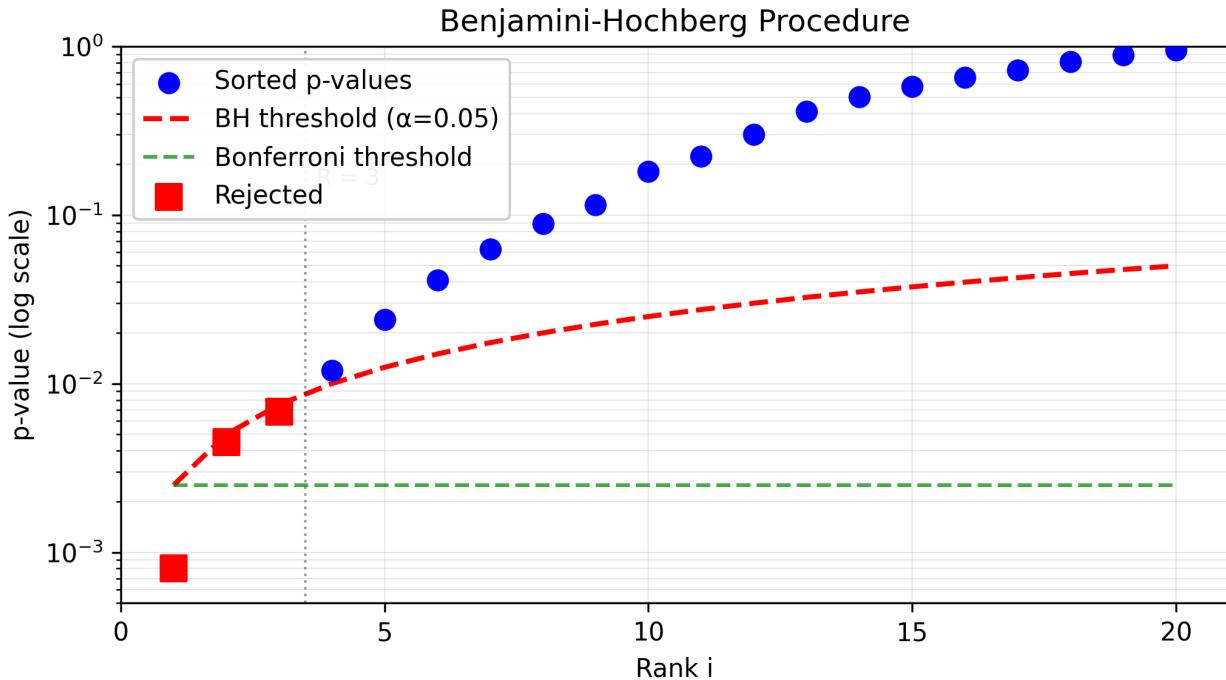
# Fix axes
plt.xticks(range(0, m+1, 5)) # Show every 5th rank
plt.xlim(0, m+1) # Proper bounds for discrete ranks

# Use log scale for y-axis
plt.yscale('log')
plt.ylim(0.0005, 1.0) # Start just below smallest p-value

plt.tight_layout()
plt.show()

if R > 0:
    print(f"\"BH procedure:")
    print(f"  R = {R} (largest i where P_(i) < i· /m)")
    print(f"  Rejection threshold T = P_({R}) = {T:.4f}")
    print(f"  Rejects {R} hypotheses (p-values {T:.4f})")
    print(f"\nBonferroni would reject {np.sum(p_sorted < alpha_fdr/m)} hypotheses")
    print(f"  (only those with p < {alpha_fdr/m:.4f}))")
else:
    print("No hypotheses rejected by either method")

```



```

BH procedure:
R = 3 (largest i where P_(i) < i· /m)

```

```
Rejection threshold T = P_(3) = 0.0068
Rejects 3 hypotheses (p-values 0.0068)
```

Bonferroni would reject 1 hypotheses
(only those with $p < 0.0025$)

The visualization shows the key difference between methods:

- **Bonferroni** (flat green line): Same strict threshold for all tests
- **BH** (sloped red line): More lenient threshold for higher-ranked p-values
- The BH procedure finds the rightmost crossing of p-values below the sloped line

When the BH procedure is applied with the appropriate C_m :

$$\text{FDR} = \mathbb{E}[\text{FDP}] \leq \frac{m_0}{m}\alpha \leq \alpha$$

where m_0 is the number of true null hypotheses.

This guarantee holds:

- With $C_m = 1$ when the test statistics are independent
- With $C_m = \sum_{i=1}^m \frac{1}{i}$ for arbitrary dependence structures

In practice, $C_m = 1$ is almost always used as the procedure is remarkably robust to many forms of dependence.

Comparison of Methods:

Method	Controls	Decision Rule	Power	Use Case
Bonferroni	$\text{FWER} \leq \alpha$	Reject if $P_i < \alpha/m$	Low	Critical applications
Benjamini-Hochberg	$\text{FDR} \leq \alpha$	Reject if $P_i \leq T = P_{(R)}^*$	Higher	Large-scale testing

*Where $R = \max\{i : P_{(i)} \leq i\alpha/(C_m m)\}$ with $C_m = 1$ for independent tests

i Adjusted p-values: An Alternative Presentation

The BH procedure can also be presented using **adjusted p-values**, which is how many software packages (including `scipy.stats.false_discovery_control`) implement it. This might seem like a completely different method, but it's just a reformulation of the same procedure.

Adjusted p-values: For the BH method, the adjusted p-value is computed in two steps:

1. First, scale each p-value: $\tilde{P}'_{(i)} = \min\left(1, \frac{m}{i} P_{(i)}\right)$
2. Then enforce monotonicity (working from largest to smallest):

$$\tilde{P}_{(i)} = \min_{j \geq i} \tilde{P}'_{(j)}$$

This ensures adjusted p-values are non-decreasing: $\tilde{P}_{(1)} \leq \tilde{P}_{(2)} \leq \dots \leq \tilde{P}_{(m)}$.

After computing adjusted p-values, reject all hypotheses where $\tilde{P}_i \leq \alpha$.

This is **mathematically equivalent** to the threshold approach but ensures the adjusted p-values maintain proper ordering.

```

import scipy.stats as stats

# Example p-values
pvals = [0.003, 0.02, 0.04, 0.08, 0.15, 0.25]
m = len(pvals)

# Using scipy's function (returns adjusted p-values)
adjusted = stats.false_discovery_control(pvals, method='bh')

print("Original vs Adjusted p-values:")
print("-" * 40)
for i, (p, adj) in enumerate(zip(pvals, adjusted)):
    # Step 1: Scale the p-value
    scaled = min(1.0, (m/(i+1)) * p)
    print(f"p[{i+1}] = {p:.3f} → scaled = {scaled:.3f} → adjusted = {adj:.3f}")

print("\nNote: adjusted values enforce monotonicity")
print("(each adjusted p-value previous one)")

print(f"\nAt = 0.05:")
print(f"Reject hypotheses where adjusted p-value 0.05")
print(f"Result: Reject first {sum(adjusted <= 0.05)} hypotheses")

Original vs Adjusted p-values:
-----
p[1] = 0.003 → scaled = 0.018 → adjusted = 0.018
p[2] = 0.020 → scaled = 0.060 → adjusted = 0.060
p[3] = 0.040 → scaled = 0.080 → adjusted = 0.080
p[4] = 0.080 → scaled = 0.120 → adjusted = 0.120
p[5] = 0.150 → scaled = 0.180 → adjusted = 0.180
p[6] = 0.250 → scaled = 0.250 → adjusted = 0.250

Note: adjusted values enforce monotonicity
(each adjusted p-value previous one)

At = 0.05:
Reject hypotheses where adjusted p-value 0.05
Result: Reject first 1 hypotheses
The adjusted p-values tell you: "This is the smallest level at which this hypothesis would be rejected by the BH procedure."

```

7.6.5 Practical Recommendations

When to Use Which Method

Use Bonferroni when:

- You have a small number of tests (< 20)
- The cost of any false positive is very high
- You need to convince skeptical reviewers

Use Benjamini-Hochberg when:

- You have many tests (genomics, neuroimaging, etc.)
- Some false positives are acceptable

- You want to generate hypotheses for follow-up

Use no correction when:

- Tests are explicitly exploratory
- You're doing hypothesis generation, not confirmation
- But always report that no correction was applied!

7.7 NHST in Practice: A Critical View

Despite its ubiquity in scientific research, NHST has fundamental limitations that have contributed to what many call the **replication crisis** – the inability to reproduce many published scientific findings.

7.7.1 Fundamental Problems with NHST

The Focus on the Null: NHST tells us whether to reject H_0 , but doesn't require us to specify a realistic alternative. We test against "no effect" without having to say what effect we actually expect. This leads to vague research questions and post-hoc rationalization of any significant finding.

Ignoring Prior Plausibility: NHST treats all hypotheses equally – the same $p < 0.05$ is required whether testing a well-established biological mechanism or claiming telepathy exists. As the saying goes, "extraordinary claims require extraordinary evidence," but NHST doesn't account for this. This is a key limitation addressed by Bayesian methods (Chapter 8).

The Dichotomy Problem: The division into "significant" and "non-significant" at $\alpha = 0.05$ is entirely arbitrary. As statistician Andrew Gelman notes:

"The difference between 'significant' and 'not significant' is not itself statistically significant!"

A result with $p = 0.049$ is treated fundamentally differently from $p = 0.051$, despite being practically identical.

7.7.2 The Low Power Trap

Low-powered studies create a vicious cycle:

- They often fail to detect true effects (high Type II error)
- When they do find "significant" results, these are more likely to be false positives
- **Crucially:** In low-power settings, most published "significant" findings are false positives

This counterintuitive result occurs because with low power, the few significant results that emerge are disproportionately likely to be statistical flukes rather than real effects.

7.7.3 Common Misuses

p-hacking (Data Dredging):

- Testing multiple hypotheses until finding $p < 0.05$
- Stopping data collection when significance is reached
- Trying different analyses until one "works"
- Excluding "outliers" post-hoc to achieve significance

HARKing (Hypothesizing After Results are Known): Looking at the data first, then pretending the observed pattern was the hypothesis all along.

Publication Bias: Journals preferentially publish "significant" results, creating a distorted scientific literature where negative results disappear. This file-drawer problem means the published record overestimates effect sizes and underestimates uncertainty.

7.7.4 Moving Forward: Better Practices

Practical Recommendations:

- Always report effect sizes and confidence intervals, not just p-values
- Conduct power analyses before collecting data
- Pre-register hypotheses and analysis plans to prevent p-hacking
- Consider Bayesian methods when prior information exists
- Use appropriate multiple testing corrections when testing many hypotheses
- Report all analyses attempted, not just the “significant” ones

Alternative Approaches:

- Bayesian inference (Chapter 8): Incorporates prior knowledge and provides probability statements about hypotheses
- Estimation-focused analysis: Emphasize effect sizes and uncertainty rather than binary decisions
- Replication studies: The ultimate test of a finding’s validity

7.8 Chapter Summary

7.8.1 Key Concepts Review

We’ve explored the foundations of null-hypothesis significance testing (NHST):

The Framework:

- Null and alternative hypotheses: H_0 (no effect) vs H_1 (effect exists)
- Type I and Type II errors: False positives vs false negatives
- Power and size: Probability of detecting true effects vs controlling false positives
- Test statistic and rejection region: Summarizing evidence and decision rules

The p-value:

- Measures how surprising data would be under H_0
- NOT the probability that H_0 is true
- Small p-value = evidence against H_0 , not proof
- Statistical significance – practical significance

Key Tests:

- Wald test: Uses asymptotic normality of estimators
- Permutation test: Non-parametric alternative requiring minimal assumptions
- Fisher’s exact test: Exact test for contingency tables
- Likelihood ratio test: General framework for testing constraints on parameters

Multiple Testing:

- Running many tests inflates Type I error rate
- Bonferroni: Controls FWER (conservative but safe)
- Benjamini-Hochberg: Controls FDR (more powerful, modern standard)

7.8.2 Common Pitfalls to Avoid

1. Misinterpreting p-values: Remember, p-value = $P(H_0 \text{ is true})$
2. Multiple testing without correction: Always consider how many tests you’re running
3. Confusing statistical and practical significance: A tiny effect can be “significant” with enough data
4. Ignoring assumptions: Tests like the Wald test require large samples
5. Post-hoc hypothesis formulation: Don’t look at data, then formulate hypotheses to test

7.8.3 Chapter Connections

- **Previous chapters:**
 - Chapters 5-6 gave us estimators and their properties; now we test hypotheses about them
 - Chapter 4 (Bootstrap) provides an alternative to asymptotic tests
- **This chapter:** Core framework for statistical inference and decision-making
- **Next chapter:** Bayesian inference offers an alternative paradigm that:
 - Provides probabilities for hypotheses
 - Incorporates prior information
 - Avoids some NHST pitfalls

7.8.4 Self-Test Problems

1. Understanding Type I and Type II Errors

A medical test for a disease has the following properties:

- If a person has the disease, the test is positive 95% of the time
- If a person doesn't have the disease, the test is negative 98% of the time

In hypothesis testing terms (where H_0 : person is healthy):

- a) What is the Type I error rate?
- b) What is the Type II error rate?
- c) What is the power of the test?

i Solution

- a) Type I error = rejecting H_0 when true = false positive = $1 - 0.98 = 0.02$
- b) Type II error = failing to reject H_0 when false = false negative = $1 - 0.95 = 0.05$
- c) Power = $1 - \text{Type II error} = 0.95$

2. Wald Test Calculation

Death times around Passover (from AoS Exercise 10.6): Of 1919 deaths, 922 occurred the week before Passover and 997 the week after. Test $H_0 : p = 0.5$ where p is the probability of death in the week before.

Calculate the Wald test statistic and p-value.

i Solution

```

import numpy as np
from scipy import stats

n = 1919
x = 922
p_hat = x / n
p_0 = 0.5

# Wald test statistic
se = np.sqrt(p_0 * (1 - p_0) / n)
W = (p_hat - p_0) / se

# p-value (two-sided)
p_value = 2 * stats.norm.cdf(-abs(W))

print(f"Sample proportion: {p_hat:.4f}")
print(f"Wald statistic: {W:.3f}")
print(f"p-value: {p_value:.4f}")
print(f"Conclusion: {'Reject H' if p_value < 0.05 else 'Fail to reject H'}")

```

Sample proportion: 0.4805
 Wald statistic: -1.712
 p-value: 0.0869
 Conclusion: Fail to reject H

3. Multiple Testing Correction

You run 10 hypothesis tests and get these p-values:

0.001, 0.004, 0.012, 0.025, 0.041, 0.053, 0.074, 0.135, 0.246, 0.531

At $\alpha = 0.05$, which hypotheses are rejected using:

- a) No correction?
- b) Bonferroni correction?
- c) Benjamini-Hochberg correction?

i Solution

```

p_values = np.array([0.001, 0.004, 0.012, 0.025, 0.041, 0.053, 0.074, 0.135, 0.246, 0.531])
alpha = 0.05
m = len(p_values)

# No correction
no_correction = p_values <= alpha
print(f"No correction: Reject hypotheses {np.where(no_correction)[0] + 1}")
print(f"  ({np.sum(no_correction)}) rejections")

# Bonferroni
bonferroni_threshold = alpha / m
bonferroni = p_values <= bonferroni_threshold
print(f"\nBonferroni (threshold = {bonferroni_threshold:.4f}):")
print(f"  Reject hypotheses {np.where(bonferroni)[0] + 1}")
print(f"  ({np.sum(bonferroni)}) rejections")

# Benjamini-Hochberg
sorted_idx = np.argsort(p_values)
sorted_p = p_values[sorted_idx]
bh_threshold = (np.arange(1, m+1) / m) * alpha
bh_reject_sorted = sorted_p <= bh_threshold
if np.any(bh_reject_sorted):
    k_max = np.max(np.where(bh_reject_sorted)[0])
    bh_reject = np.zeros(m, dtype=bool)
    bh_reject[sorted_idx[:k_max+1]] = True
else:
    bh_reject = np.zeros(m, dtype=bool)

print(f"\nBenjamini-Hochberg:")
print(f"  Reject hypotheses {np.where(bh_reject)[0] + 1}")
print(f"  ({np.sum(bh_reject)}) rejections")

No correction: Reject hypotheses [1 2 3 4 5]
(5 rejections)

Bonferroni (threshold = 0.0050):
Reject hypotheses [1 2]
(2 rejections)

Benjamini-Hochberg:
Reject hypotheses [1 2 3]
(3 rejections)

```

4. Permutation Test vs Parametric Test

When would you prefer a permutation test over a Wald test? Give at least three scenarios.

i Solution

Prefer permutation tests when:

1. **Small sample size:** Asymptotic approximations may not hold
2. **Non-standard distributions:** Data is heavily skewed or has outliers

3. **Complex test statistics:** No known distribution for the statistic
4. **Exact p-values needed:** Critical decisions requiring exact inference
5. **Assumptions violated:** Independence or normality assumptions fail

7.8.5 Python and R Reference

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

7.8.6 Connections to Source Material

i Mapping to “All of Statistics”

Lecture Note Section	Corresponding Source(s)
Introduction	Lecture slides; drug trial example from slides
Framework of Hypothesis Testing	AoS Ch 10 Intro
Null and Alternative Hypotheses	AoS Ch 10 Intro
Type I and Type II Errors	AoS Table 10.1, Definition 10.1
Power and Size	AoS Definition 10.1
The p-value	AoS §10.2
Understanding the p-value	AoS §10.2 (Definition 10.11, Theorem 10.12)
Interpretation and Misinterpretation	AoS §10.2, expanded with modern critiques
Hypothesis Tests	
The Wald Test	AoS §10.1 (Definition 10.3, Theorem 10.4)
Statistical and Scientific Significance	AoS §10.1, Theorem 10.10 (CI duality)
Comparing Proportions/Means	AoS §10.1 (Examples 10.7, 10.8)
The Permutation Test	AoS §10.5 (Example 10.19)
Fisher’s Exact Test	Expanded from slides
The Likelihood Ratio Test	AoS §10.6
Multiple Testing Problem	AoS §10.7
Bonferroni Correction	AoS §10.7 (Theorem 10.24)
Benjamini-Hochberg	AoS §10.7 (Theorem 10.26)
NHST in Practice: A Critical View	Expanded from slides
Self-Test Problems	Based on AoS Exercise 10.6 and similar examples

7.8.7 Further Reading

- **Modern perspective:** Wasserstein & Lazar (2016), “The ASA Statement on p-Values”
- **Critical view:** Ioannidis (2005), “Why Most Published Research Findings Are False”

Remember: Hypothesis testing is a tool for making decisions under uncertainty. Use it wisely – report effect sizes and confidence intervals, correct for multiple testing, and never forget that statistical significance is not the same as practical importance!

Chapter 8

Bayesian Inference and Statistical Decision Theory

8.1 Learning Objectives

After completing this chapter, you will be able to:

- **Apply Bayes' theorem to compute posterior distributions** from prior and likelihood, and interpret credible intervals vs. confidence intervals.
- **Work with conjugate models** (Beta-Bernoulli, Normal-Normal) to derive posteriors and understand how data and prior beliefs combine.
- **Choose appropriate priors** and explain their impact on inference, particularly as sample size increases.
- **Use decision theory to compare estimators** via loss functions and risk, understanding why we need scalar summaries (Bayes risk, maximum risk).
- **Identify optimal estimators** by connecting posterior summaries to Bayes estimators, finding minimax estimators via constant risk, and determining admissibility.

Note

This chapter introduces two deeply connected topics: Bayesian inference, which provides a principled way to update beliefs with data, and statistical decision theory, which gives us a formal framework for comparing any statistical procedure. The material is adapted from Chapters 11 and 12 of Wasserman (2013) and supplemented with modern perspectives and computational examples.

8.2 Introduction: A Different Way of Thinking

8.2.1 The Search for Air France Flight 447

On June 1, 2009, Air France Flight 447 vanished over the Atlantic Ocean. The Airbus A330, carrying 228 people, disappeared from radar while flying from Rio de Janeiro to Paris, leaving behind only automated messages indicating system failures. What followed was one of the most challenging search operations in aviation history – and ultimately, a powerful demonstration of how Bayesian inference succeeds by integrating multiple sources of uncertain information. This remarkable story is documented in detail in Stone et al. (2014), from which the figures and search details in this section are taken.

Modern airliners transmit their position every 10 minutes via satellite. When AF447's transmissions stopped at 2:14 AM, it created a circular search area with a 40 nautical mile (74 km) radius – still covering over 5,000 square nautical miles of ocean.

The depth in this region reaches 14,000 feet, with underwater mountains and valleys making detection extremely challenging. The flight data and cockpit voice recorders, crucial for understanding what happened, emit acoustic beacons that function for about 40 days and have a detection range of about 2,000 meters.

This wasn't just a search problem – it was a problem of combining uncertain, conflicting information from multiple sources.

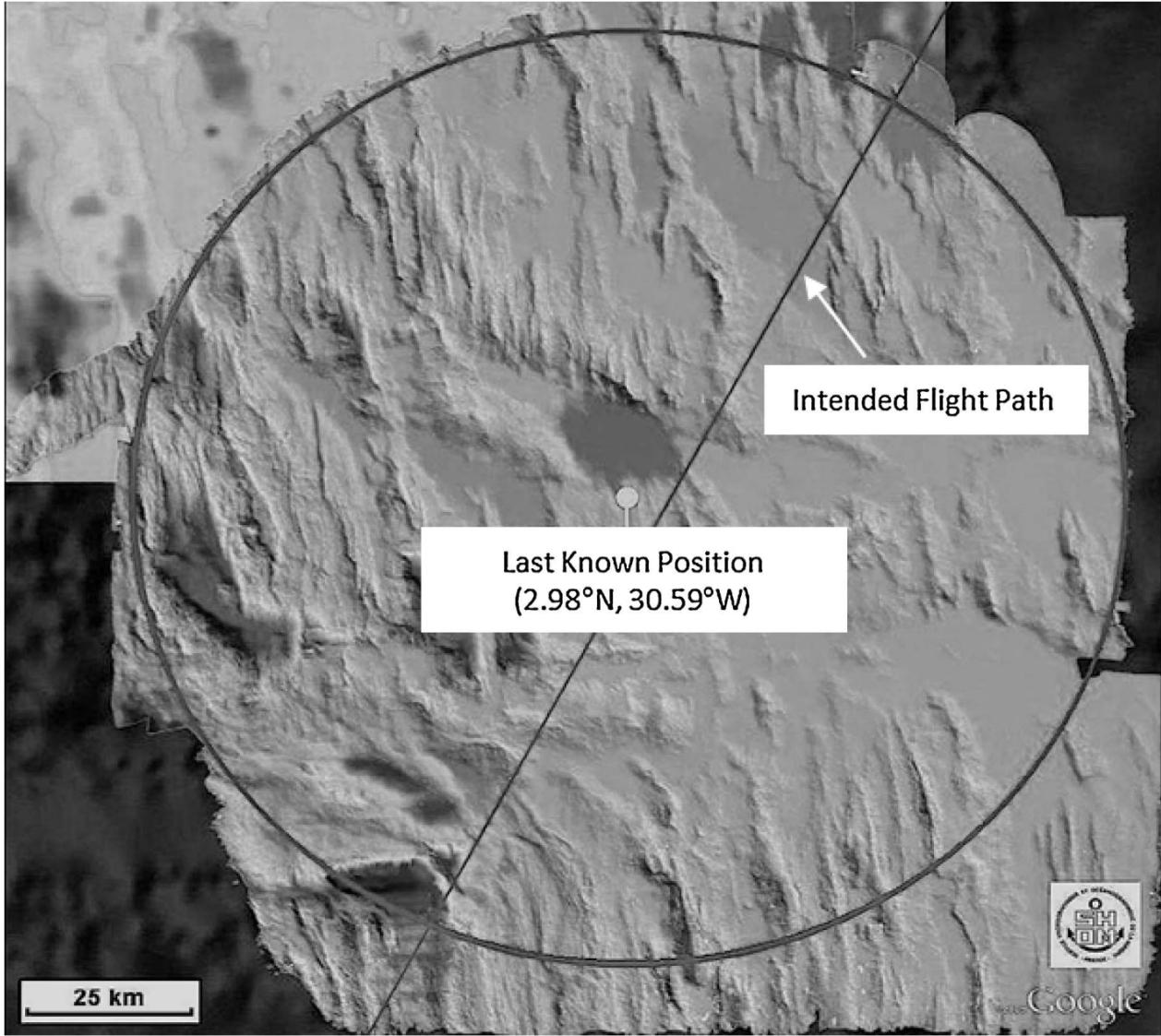


FIG. 1. *Last known position of the aircraft, intended flight path and the 40 NM circle.*

Figure 8.1: The intended flight path of AF447 and the 40 NM radius circle centered on the last known position (LKP). The circle represents the maximum distance the aircraft could have traveled after its last transmission. Figure from Stone et al. (2014).

8.2.1.1 The Initial Search Efforts

Throughout 2009 and 2010, search teams employed sophisticated statistical models including oceanographic drift analysis and confidence regions. However, each search operation focused on a single line of evidence rather than integrating all available information. These efforts, while extensive and expensive, all failed to

locate the wreckage:

Multiple Perspectives

Surface Search (June 2009)

The first phase focused on finding floating debris. After six days, search aircraft spotted debris and bodies approximately 38 NM north of the last known position. Scientists then used reverse drift modeling (working backwards from where debris was found, using ocean current data to estimate where it originated) to predict where the wreckage might be.

Result: No wreckage found in the predicted areas.

Acoustic Search (June 2009)

Teams deployed sensitive hydrophones to listen for the flight recorders' acoustic beacons. They concentrated along the intended flight path, reasoning the aircraft was likely on course when it crashed.

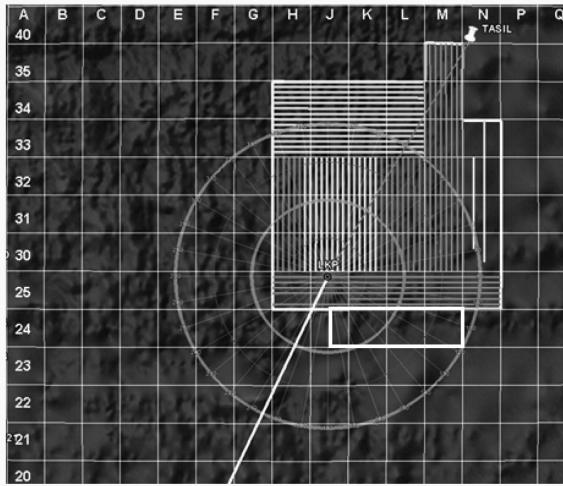


FIG. 5. The vertical and horizontal lines show the search paths for the passive acoustic search. The circles are the 20 and 40 NM circles about the last known position. The white rectangle in row 24 was searched by side-looking sonar in August 2009.

Figure 8.2: The vertical and horizontal search lines showing the passive acoustic search paths for the flight recorder beacons. The circles show the 20 and 40 NM radius from the last known position. Figure from Stone et al. (2014).

Result: No signals detected. The search assumed the beacons were functioning – a reasonable but ultimately incorrect assumption.

Active Sonar (August 2009)

A limited side-scan sonar search was conducted south of the last known position in areas not covered in June.

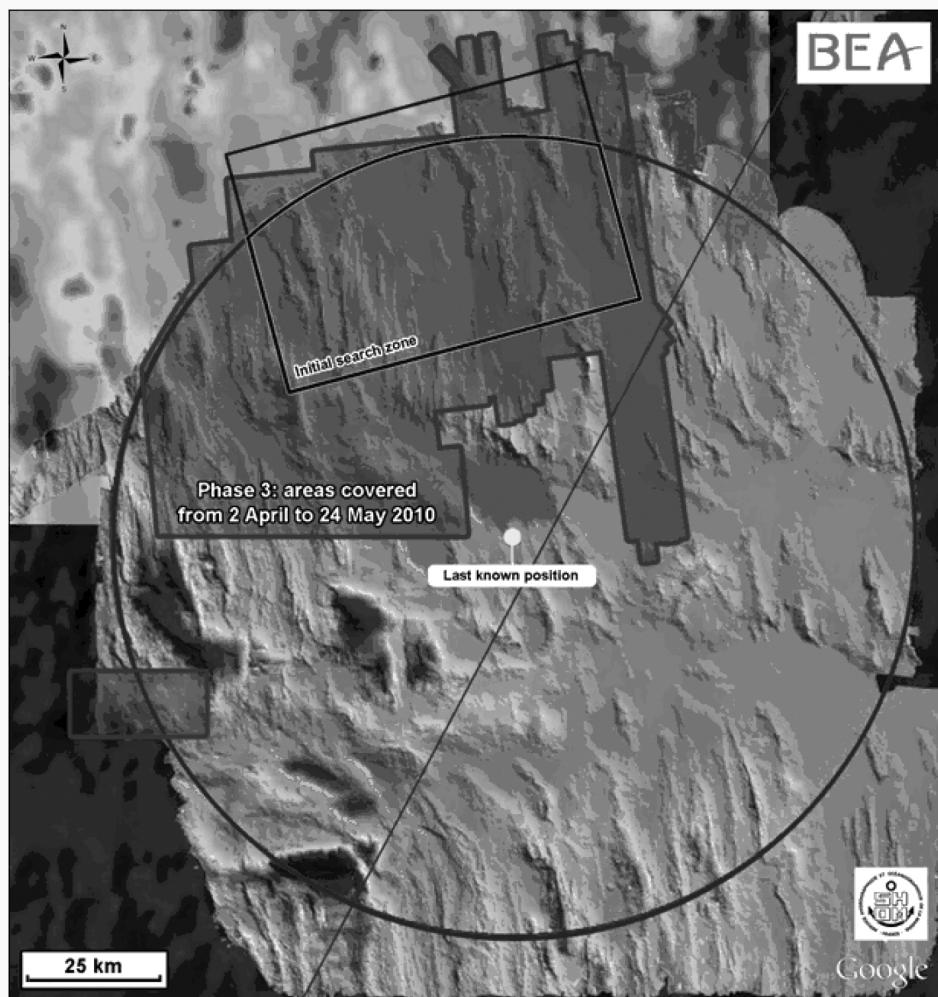


FIG. 6. Regions searched by active side-looking sonar in April–May 2010.

Figure 8.3: Regions searched by active side-looking sonar. The small rectangle shows the limited August 2009 coverage, while the larger areas show April–May 2010 coverage. Figure from Stone et al. (2014).

Result: No wreckage found.

Confidence Region (April–May 2010)

Scientists computed a 95% confidence region by reverse drift modeling from where bodies and debris were recovered. By simulating ocean currents backwards in time, they estimated where the crash most likely occurred, producing a search zone north and west of the last known position.

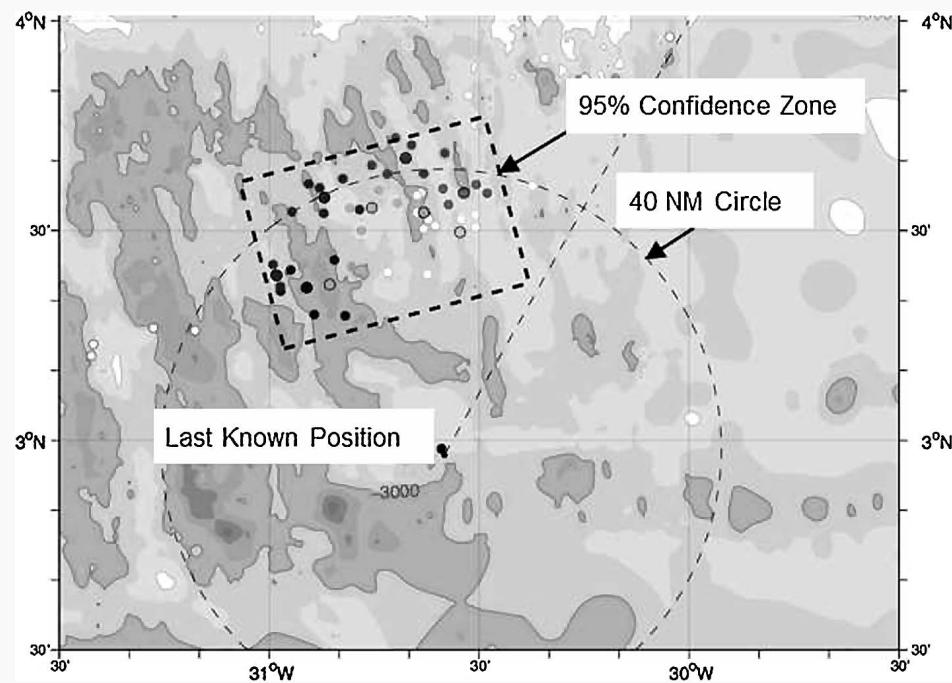


FIG. 2. The 95% confidence zone recommended in [6] for the 2010 search area.

Figure 8.4: The 95% confidence zone recommended for the 2010 search, located north and west of the LKP, based on reverse drift modeling. Figure from Stone et al. (2014).

Result: No wreckage found. The confidence region, while statistically valid, relied heavily on ocean current models and didn't integrate other sources of evidence like historical crash locations or search effectiveness.

⚠ Why the Initial Approaches Failed

Each search used valid and sophisticated statistical reasoning but treated evidence in isolation:

- Drift models didn't account for prior crash locations
- Passive acoustic searches couldn't distinguish between beacon failure and absence of wreckage
- Search patterns didn't incorporate the probability of missing the wreckage
- No unified framework was used to combine these different sources of uncertainty

8.2.1.2 The Bayesian Strategy

In July 2010, after four unsuccessful search operations, the French aviation authority (BEA) assembled a new team of statisticians to design a search strategy for 2011. This team took a fundamentally different approach: instead of treating each piece of evidence separately, they used Bayesian inference to combine *all* sources of information into a single probability distribution.

The Bayesian Framework

The team constructed a posterior probability distribution for the wreckage location by combining:

1. **Prior Distribution:** Historical data showed that aircraft are usually found close to their last known position. This gave higher prior probability to areas near the center of the circle.
2. **Drift Model Likelihood:** Bodies found north of the LKP implied certain starting positions were more likely than others – but with significant uncertainty.

3. **Search Effectiveness:** Previous searches weren't perfect. The team modeled the probability of missing the wreckage in searched areas, particularly accounting for terrain difficulty.
4. **Beacon Failure Possibility:** The lack of acoustic signals could mean either the wreckage wasn't in searched areas OR the beacons had failed. Bayesian analysis could incorporate both possibilities.

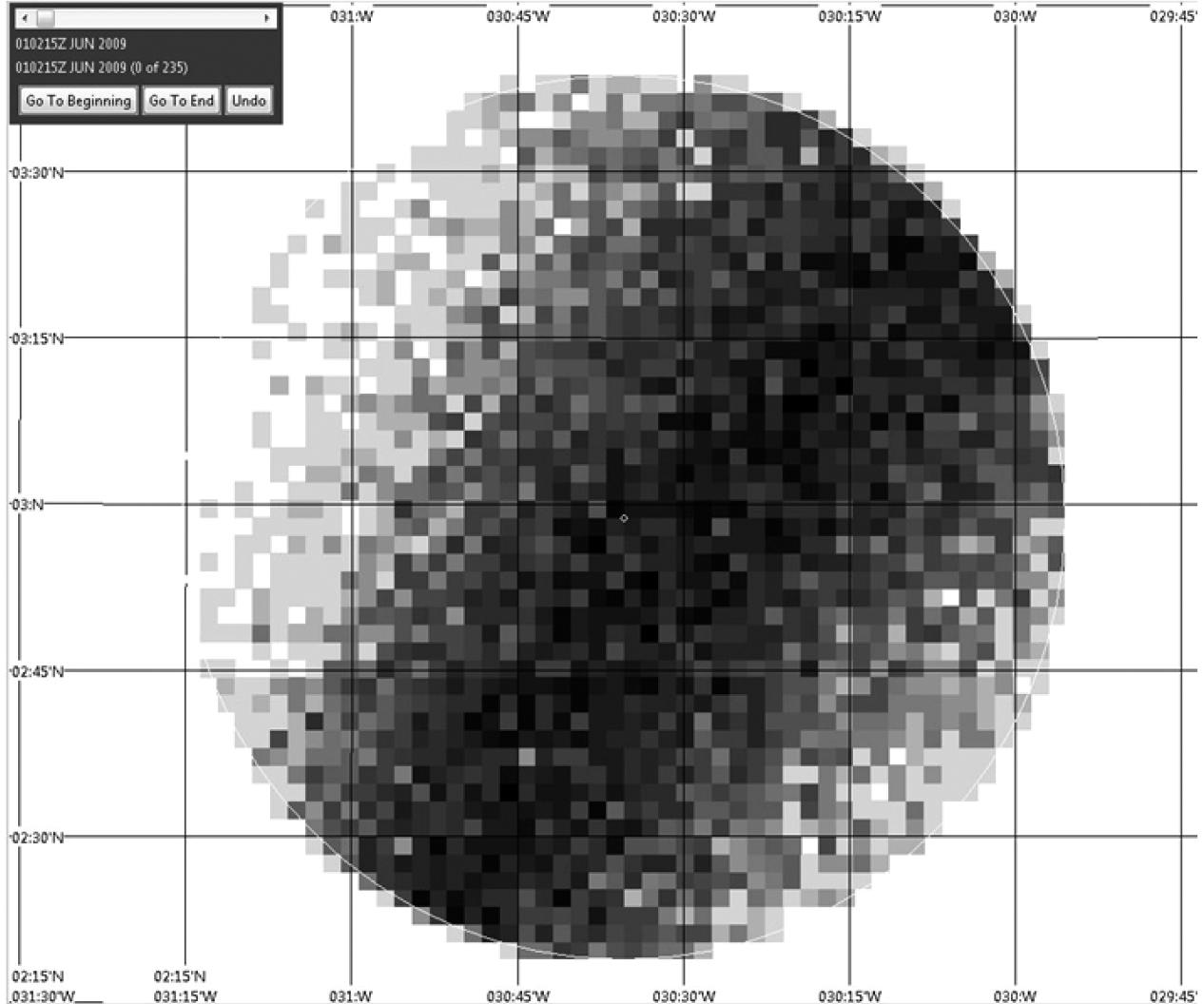


FIG. 3. *Reverse drift distribution D_3 .*

Figure 8.5: Reverse drift distribution showing the probability density of potential crash locations based on where bodies and debris were found. This was one key input to the Bayesian analysis. Figure from Stone et al. (2014).

i Technical Detail: Computing the Posterior

The posterior distribution was computed using:

$$P(\text{location} | \text{all evidence}) \propto P(\text{all evidence} | \text{location}) \times P(\text{location})$$

Where the evidence included:

- Negative search results (no detection in searched areas)

- Positive drift data (bodies found at specific locations)
- Timing constraints (time between crash and debris discovery)

The likelihood $P(\text{all evidence}|\text{location})$ was itself a product of multiple conditional probabilities, each capturing different aspects of the search problem. Monte Carlo methods were used to integrate over unknown parameters like ocean current variations and detection probabilities.

8.2.1.3 The Breakthrough

The Bayesian analysis produced a surprising result: the highest probability areas were very close to the last known position. Although these areas had been covered by passive acoustic searches in 2009, the active sonar efforts in 2009-2010 had focused elsewhere based on drift models.

The Key Insight

The Bayesian approach revealed that multiple pieces of weak evidence all pointed to the same conclusion:

- Historical data suggested searching near the LKP
- Debris drift models had high uncertainty and conflicting predictions
- The failure to find wreckage in extensively searched areas increased relative probability elsewhere
- Beacon failure was historically more likely than initially assumed

No single piece of evidence was conclusive, but together they pointed strongly to areas near the last known position.

8.2.1.4 Discovery and Vindication

The new search began in 2011, focusing on the high-probability areas identified by the Bayesian analysis. After just one week of searching, on April 3, 2011, the wreckage was found at a depth of approximately 14,000 feet, very close to the last known position.

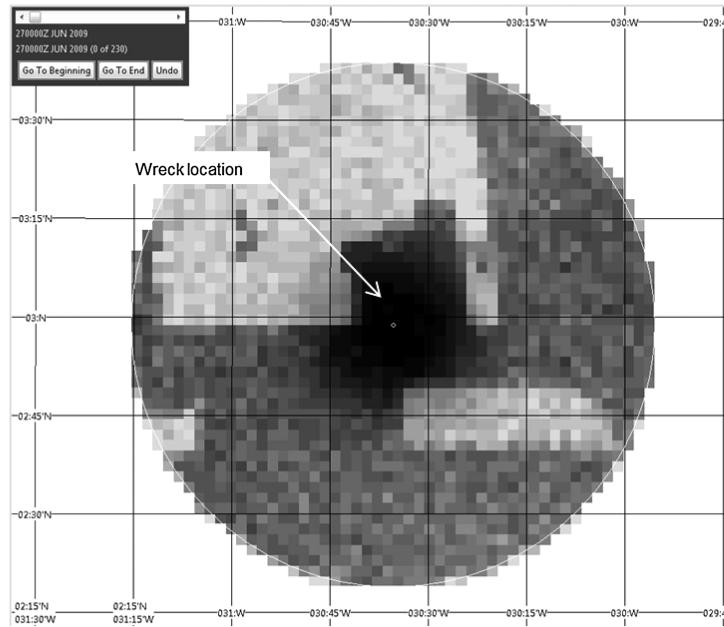


FIG. 8. Posterior distribution which assumes both beacons failed; produced because of doubts about the survivability of the beacons.

Figure 8.6: Posterior distribution from the Bayesian analysis, showing the actual wreck location marked. The dark area near the center shows the highest probability zone, which correctly identified the area where the wreckage was ultimately found. Figure from Stone et al. (2014).

i Why Bayesian Methods Succeeded

The Bayesian approach succeeded where the initial methods failed for three fundamental reasons:

1. **Coherent Information Integration:** While the initial searches treated each piece of evidence separately, Bayesian inference combined them into a single, coherent picture.
2. **Uncertainty Quantification:** The approach explicitly modeled multiple sources of uncertainty – from ocean currents to sensor reliability – rather than assuming point estimates were correct.
3. **Prior Knowledge Utilization:** Historical data about crash locations provided valuable information that pure data-driven approaches ignored.

This case demonstrates the power of Bayesian thinking: when faced with multiple sources of imperfect information, Bayesian methods provide the mathematical framework to combine them optimally.

8.2.2 The Two Philosophies of Statistics

In the world of statistical inference, there are two major philosophical schools of thought about probability, parameters, and how we should make inferences from data. These aren't just abstract philosophical debates – they lead to fundamentally different methods, interpretations, and answers. Understanding both perspectives is crucial for modern data scientists.

Multiple Perspectives

Frequentist View

We've been working primarily within the frequentist framework throughout this course. Let's formalize its key principles:

- F1. Probability as Frequency:** Probability refers to limiting relative frequencies in repeated experiments. Probabilities are objective properties of the real world. When we say a coin has probability 0.5 of landing heads, we mean that in an infinite sequence of flips, exactly half would be heads.
- F2. Fixed Parameters:** Parameters are fixed, unknown constants. They are not random variables. Because they don't vary, we cannot make probability statements about them. We can't say "there's a 95% probability that μ is between 2 and 4" – either it is or it isn't.
- F3. Long-Run Performance:** Statistical methods should have well-defined long-run frequency properties. A 95% confidence interval should trap the true parameter in 95% of repeated experiments. This is a statement about the procedure, not about any particular interval.
- F4. Point-Conditioned Prediction:** Predictions are typically conditioned on a single parameter value, often an estimate like the MLE. We predict future data assuming our estimate is correct.
- F5. Separate Theories:** There's no single, overarching theory unifying all aspects of frequentist inference. Estimation theory, hypothesis testing, and prediction each have their own frameworks and optimality criteria.

Bayesian View

The Bayesian approach starts from fundamentally different assumptions:

- B1. Probability as Belief:** Probability describes degree of belief or confidence. Probabilities can be subjective and represent our uncertainty about anything – including fixed events. We can meaningfully say "I'm 70% confident it rained in Paris on January 1, 1850" even though this is a fixed historical fact.
- B2. Probabilistic Parameters:** We can make probability statements about parameters, treating our uncertainty about them as something to be described by a probability distribution. Even though θ is fixed, our knowledge about it is uncertain, and we quantify this uncertainty with probabilities.
- B3. Inference as Belief Updating:** The core of inference is updating our beliefs about parameters by producing a posterior probability distribution after observing data. This posterior encapsulates everything we know about the parameter.

B4. Averaged Prediction: Predictions are made by averaging over all parameter values, weighted by their posterior probability. Instead of picking one “best” parameter value, we consider all plausible values.

B5. Unified Theory: The framework has a strong, unified theoretical foundation based on the rules of probability. Bayes’ theorem provides a single coherent approach to all inference problems.

8.2.3 This Chapter’s Goal

We will explore two deeply connected topics:

1. **Bayesian Inference:** The machinery for updating our beliefs about parameters using data. We’ll see how prior knowledge combines with observed data to produce posterior distributions.
2. **Statistical Decision Theory:** A formal framework for choosing the “best” estimator under any paradigm. This theory, which applies to both frequentist and Bayesian methods, gives us a rigorous way to compare different statistical procedures.

These topics are connected because Bayesian inference naturally leads to optimal estimators under decision theory, while decision theory helps us understand when and why Bayesian methods work well.

i Finnish Terminology Reference

For Finnish-speaking students, here’s a reference table of key terms in this chapter:

English	Finnish	Context
Bayesian inference	Bayesiläinen päätteily	Main inferential framework
Prior distribution	Priorijakauma	Beliefs before seeing data
Posterior distribution	Posteriorijakauma	Updated beliefs after data
Likelihood	Uskottavuus	Probability of data given parameters
Credible interval	Uskottavuusväli	Bayesian confidence interval
Loss function	Tappiofunktio	Measure of estimation error
Risk	Riski	Expected loss
Bayes estimator	Bayes-estimaattori	Minimizes Bayes risk
Minimax estimator	Minimax-estimaattori	Minimizes maximum risk
Admissible	Käypä, kelvollinen	Cannot be uniformly improved

8.3 The Bayesian Method: Updating Beliefs with Data

8.3.1 The Engine: Bayes’ Theorem for Inference

The Bayesian method centers on a fundamental question: **how do we make predictions about unknown quantities** when we have uncertain knowledge about the parameters that govern them?

Consider predicting some unknown quantity x^* (which could be future data, or properties of the parameter itself) when we have:

- A model with unknown parameter θ
- Observed data $x^n = (x_1, \dots, x_n)$ that provides information about θ

Using the rules of probability, we can write:

$$f(x^*|x^n) = \int f(x^*|\theta, x^n) f(\theta|x^n) d\theta$$

If x^* depends on the data only through θ (a common assumption), this simplifies to:

$$f(x^*|x^n) = \int f(x^*|\theta)f(\theta|x^n)d\theta$$

This equation reveals the key insight: **to make predictions, we need the posterior distribution $f(\theta|x^n)$.** The posterior tells us which parameter values are plausible given the data, and we average our predictions over all these plausible values.

The Components of Bayesian Inference

To compute the posterior distribution $f(\theta|x^n)$, we need:

- **Prior Distribution $f(\theta)$:** What we believe about θ *before* seeing the data. This encodes our initial knowledge or assumptions. We will see later how the prior is chosen.
- **Likelihood $f(x^n|\theta)$ or $\mathcal{L}_n(\theta)$:** The probability of observing our data given different parameter values. This is the same likelihood function used in maximum likelihood estimation.
- **Posterior Distribution $f(\theta|x^n)$:** Our updated belief about θ *after* seeing the data, obtained via Bayes' theorem.

The posterior distribution is computed as:

$$f(\theta|x^n) = \frac{f(x^n|\theta)f(\theta)}{\int f(x^n|\theta)f(\theta)d\theta}$$

The denominator $\int f(x^n|\theta)f(\theta)d\theta$ is called the **marginal likelihood** or **evidence**. It's a normalizing constant that ensures the posterior integrates to 1.¹

We often do not specifically care about the normalizing constant, and write:

$$f(\theta|x^n) \propto f(x^n|\theta)f(\theta)$$

denoting that the posterior is **proportional** to Likelihood times Prior.

When the observations X_1, \dots, X_n are IID given θ , the likelihood factorizes:

$$f(\theta|x^n) \propto \mathcal{L}_n(\theta)f(\theta) = \left[\prod_{i=1}^n f(x_i|\theta) \right] f(\theta)$$

This product structure is what allows evidence to accumulate across independent observations.

i Why Do We Care About the Posterior?

The posterior distribution serves two distinct purposes:

1. **Direct Parameter Inference:** Sometimes the parameters themselves are what we want to know:
 - What's the true efficacy of a vaccine?
 - What's the rate of climate change?
 - What's a manufacturing process's defect rate?

Here, we examine the posterior directly to understand the parameter values.

2. **Prediction:** Other times, parameters are just a means to predict future observations:
 - Estimating weather model parameters to forecast tomorrow's conditions
 - Learning user preferences to recommend movies
 - Estimating volatility to predict financial risk

For prediction, we integrate over the posterior, incorporating parameter uncertainty into our forecasts

¹The marginal likelihood is not *just* a constant. Since it encodes the probability of the data under a specific statistical model, it can be used as a metric for comparing *different models*.

rather than conditioning on a single estimate.

8.3.2 Summarizing the Posterior

The posterior distribution $f(\theta|x^n)$ contains all our knowledge about θ after seeing the data. It's the complete Bayesian answer to an inference problem. However, we often need to summarize this distribution with a single point – a **point estimate** – for communication or decision-making.

Point Estimates:

- **Posterior Mean:** $\bar{\theta}_n = \mathbb{E}[\theta|x^n] = \int \theta f(\theta|x^n) d\theta$

The center of our posterior beliefs, weighting all possible values by their posterior probability.

- **Posterior Median:** The value θ_m such that $\mathbb{P}(\theta \leq \theta_m|x^n) = 0.5$

The value that splits the posterior distribution in half.

- **Posterior Mode (MAP):** $\hat{\theta}_{MAP} = \arg \max_{\theta} f(\theta|x^n)$

The most probable value according to the posterior. MAP stands for “Maximum A Posteriori.”²

Interval Estimates:

- **Credible Interval:** A $(1 - \alpha)$ credible interval³ is a range (a, b) such that:

$$\mathbb{P}(a < \theta < b|x^n) = 1 - \alpha$$

Typically computed as an equal-tailed interval by finding a and b where $\int_{-\infty}^a f(\theta|x^n) d\theta = \int_b^{\infty} f(\theta|x^n) d\theta = \alpha/2$.

⚠ Credible vs. Confidence Intervals

A crucial distinction:

- **Credible interval** (Bayesian): “Given the data, there’s a 95% probability that θ lies in this interval.”
- **Confidence interval** (Frequentist): “This procedure produces intervals that trap the true θ in 95% of repeated experiments.”

The credible interval makes a direct probability statement about the parameter, which is what most people incorrectly think confidence intervals do!

Multiple Perspectives

Intuitive

Imagine you’re trying to estimate the average height in a population. You take a sample and compute an interval.

Confidence Interval (Frequentist): “If I repeated this sampling procedure 100 times, about 95 of those intervals would contain the true average height.” It’s a statement about the reliability of the *method*, not about any specific interval. Once computed, the true value is either in it or not – there’s no probability involved.

Credible Interval (Bayesian): “Based on the data I observed and my prior knowledge, I’m 95% confident the true average height is in this interval.” It’s a direct probability statement about where

²The term “a posteriori” is Latin meaning “from what comes after” or “from the latter,” referring to knowledge that comes *after* observing evidence. This contrasts with “a priori” meaning “from what comes before” – knowledge *before* seeing data.

³Also called a **posterior interval** in some texts, particularly older or more theoretical works. Both terms are correct and refer to the same concept.

the parameter lies, given what we've learned.

The confidence interval is like a fishing net manufacturer's guarantee: "95% of our nets catch fish." The credible interval is like a weather forecast: "95% chance of rain tomorrow." One describes a long-run property of a procedure; the other describes belief about a specific unknown.

Mathematical

Let θ be the parameter and X^n the observed data.

Confidence Interval: Find functions $L(X^n)$ and $U(X^n)$ such that:

$$\mathbb{P}_\theta(L(X^n) \leq \theta \leq U(X^n)) = 1 - \alpha \text{ for all } \theta$$

The probability is over the random data X^n , with θ fixed. Different data gives different intervals.

Credible Interval: Find constants a and b such that:

$$\int_a^b f(\theta|X^n) d\theta = 1 - \alpha$$

The probability is over the parameter θ given fixed, observed data X^n . The interval quantifies our posterior uncertainty about θ .

Key difference: In confidence intervals, data is random and parameter is fixed. In credible intervals, data is fixed (observed) and parameter is treated as random (uncertain).

Computational

Let's simulate both types of intervals to see their fundamental difference. We'll generate many datasets to show the frequentist coverage property, then compute a single credible interval to show the Bayesian probability statement:

```

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Simulate the difference between confidence and credible intervals
np.random.seed(42)

# True parameter
true_mean = 5.0
true_std = 1.0
n = 30

# Generate many datasets to show confidence interval behavior
n_simulations = 100
confidence_intervals = []

for i in range(n_simulations):
    # Generate a dataset
    data = np.random.normal(true_mean, true_std, n)
    sample_mean = np.mean(data)
    sample_se = true_std / np.sqrt(n) # Known variance case

    # 95% Confidence interval
    ci_lower = sample_mean - 1.96 * sample_se
    ci_upper = sample_mean + 1.96 * sample_se
    confidence_intervals.append((ci_lower, ci_upper))

# Count how many contain the true parameter
coverage = sum(1 for (l, u) in confidence_intervals if l <= true_mean <= u)
print(f"Confidence Interval Coverage: {coverage}/{n_simulations} = {coverage/n_simulations:.2%}")
print("This demonstrates the frequentist guarantee: ~95% coverage in repeated sampling")

# Visualize all 100 confidence intervals
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 8))

# Top panel: Show all confidence intervals
ax1.axhline(true_mean, color='black', linestyle='-', label='True mean', linewidth=1.5, zorder=5)

# Plot all intervals, colored by whether they contain the true mean
for i in range(n_simulations):
    l, u = confidence_intervals[i]
    contains_true = l <= true_mean <= u
    color = '#0173B2' if contains_true else '#DE8F05' # Blue vs Orange (high contrast, colorblind safe)
    # Use thinner lines and transparency for better visualization
    ax1.plot([i, i], [l, u], color=color, linewidth=0.8, alpha=0.7)
    # Small dots for interval centers
    ax1.plot(i, (l+u)/2, '.', color=color, markersize=2, alpha=0.8)

# Add summary statistics
n_containing = sum(1 for (l,u) in confidence_intervals if l <= true_mean <= u)
ax1.set_xlabel('Dataset number')
ax1.set_ylabel('Parameter value')
ax1.set_title(f'All {n_simulations} Confidence Intervals\n{n_containing}/{n_simulations} ({n_containing/n_simulations:.1%}) contain true mean\nBlue = contains true mean, Orange = misses')
ax1.grid(True, alpha=0.3)
ax1.set_xlim(-1, n_simulations)

# Now show a single Bayesian credible interval

```

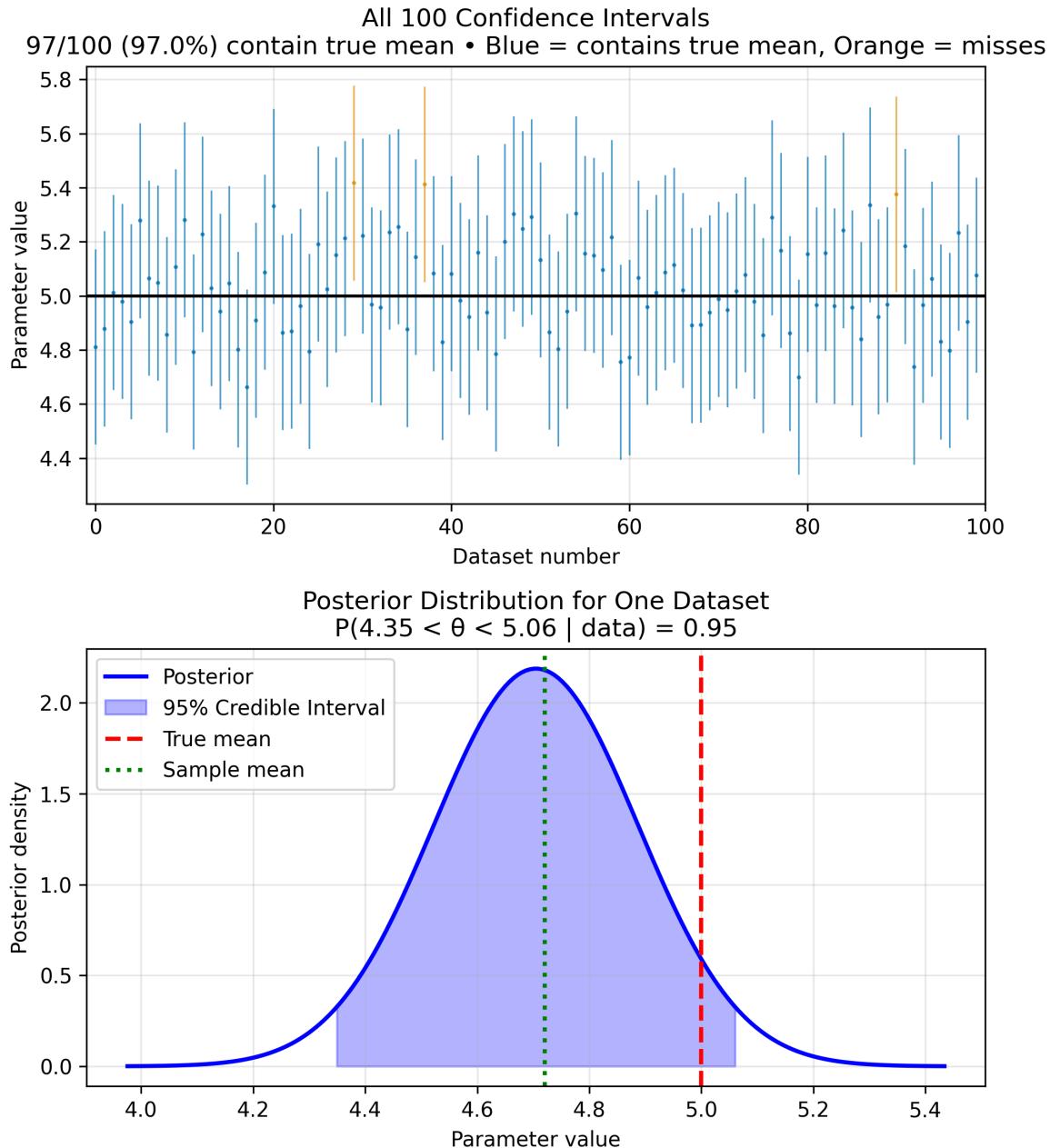
Confidence Interval Coverage: $97/100 = 97.00\%$
 This demonstrates the frequentist guarantee: ~95% coverage in repeated sampling

For this specific dataset:

Sample mean: 4.72

95% Credible Interval: [4.35, 5.06]

This is a direct probability statement about the parameter!



Key Takeaway: Confidence intervals achieve 95% coverage *across many experiments* (a procedure property), while credible intervals give 95% probability *for this specific dataset* (a parameter property). Same numbers, fundamentally different meanings.

8.4 Bayesian Inference in Action

8.4.1 Conjugate Models and Conjugate Priors

In principle, Bayesian inference requires us to compute integrals to normalize the posterior distribution. In practice, these integrals are often intractable. However, for certain combinations of priors and likelihoods, the posterior has a nice closed form. These special cases are called **conjugate models**.

i What is a Conjugate Prior?

A prior is **conjugate** to a likelihood if the resulting posterior distribution is in the same family as the prior. This means:

- If the prior is Beta, the posterior is also Beta
- If the prior is Normal, the posterior is also Normal

Conjugacy provides a convenient analytical shortcut, though modern computational methods have reduced its importance.

8.4.1.1 Example: The Bernoulli-Beta Model

Consider the fundamental problem of estimating a probability from binary data. Let $X_i \sim \text{Bernoulli}(p)$ for $i = 1, \dots, n$, where we observe s successes out of n trials.

Starting with a Uniform Prior:

Since p is a probability, it must lie in $[0, 1]$. If we have no prior information, a natural choice is the uniform prior: $f(p) = 1$ for $p \in [0, 1]$.

Likelihood: With s successes in n trials, the likelihood is:

$$\mathcal{L}_n(p) \propto p^s (1-p)^{n-s}$$

Posterior calculation:

$$f(p|x^n) \propto f(p) \times \mathcal{L}_n(p) = 1 \times p^s (1-p)^{n-s} = p^{(s+1)-1} (1-p)^{(n-s+1)-1}$$

This has the form of a **Beta distribution**! Specifically, if we match the parameters:

$$p|x^n \sim \text{Beta}(s+1, n-s+1)$$

The mean of $\text{Beta}(\alpha, \beta)$ is $\alpha/(\alpha + \beta)$, so the posterior mean here is $\bar{p} = \frac{s+1}{n+2}$, which can be written as:

$$\bar{p} = \frac{n}{n+2} \cdot \frac{s}{n} + \frac{2}{n+2} \cdot \frac{1}{2}$$

This is a weighted average of the MLE $\hat{p} = s/n$ and the prior mean $1/2$, with the data getting more weight as n increases.

The General Beta Prior:

The uniform prior is actually a special case of the Beta distribution. In general, if we use a $\text{Beta}(\alpha, \beta)$ prior:

$$f(p) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1} (1-p)^{\beta-1}$$

Then the posterior is:

$$p|x^n \sim \text{Beta}(\alpha + s, \beta + n - s)$$

Key insights:

- The Beta distribution is **conjugate** to the Bernoulli likelihood - the posterior stays in the Beta family
- The parameters α and β act as “pseudo-counts”: α prior successes, β prior failures
- The uniform prior is $\text{Beta}(1, 1)$ - one pseudo-success and one pseudo-failure
- The posterior mean $\bar{p} = \frac{\alpha+s}{\alpha+\beta+n}$ combines prior pseudo-counts with observed counts
- As $n \rightarrow \infty$, the data dominates and the prior’s influence vanishes

Let’s visualize how the posterior evolves with data:

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Set up the figure
fig, axes = plt.subplots(2, 1, figsize=(7, 7))

# Top panel: Effect of sample size
ax1 = axes[0]
p_true = 0.7 # True probability
alpha_prior, beta_prior = 1, 1 # Uniform prior

# Different sample sizes
sample_sizes = [0, 10, 50, 200]
colors = ['gray', 'blue', 'green', 'red']

p_range = np.linspace(0, 1, 200)

# Set seed once outside the loop for reproducible results
np.random.seed(42)

for n, color in zip(sample_sizes, colors):
    if n == 0:
        # Just the prior
        y = stats.beta.pdf(p_range, alpha_prior, beta_prior)
        label = 'Prior'
    else:
        # Simulate data
        s = np.random.binomial(n, p_true)
        # Posterior parameters
        alpha_post = alpha_prior + s
        beta_post = beta_prior + n - s
        y = stats.beta.pdf(p_range, alpha_post, beta_post)
        label = f'n={n}, s={s}'

    ax1.plot(p_range, y, color=color, linewidth=2, label=label)

ax1.axvline(p_true, color='black', linestyle='--', alpha=0.5, label='True p')
ax1.set_xlabel('p')
ax1.set_ylabel('Density')
ax1.set_title('Posterior Becomes More Concentrated with More Data')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Bottom panel: Effect of different priors
ax2 = axes[1]

```

```
n = 20
s = 10 # 50% success rate in data

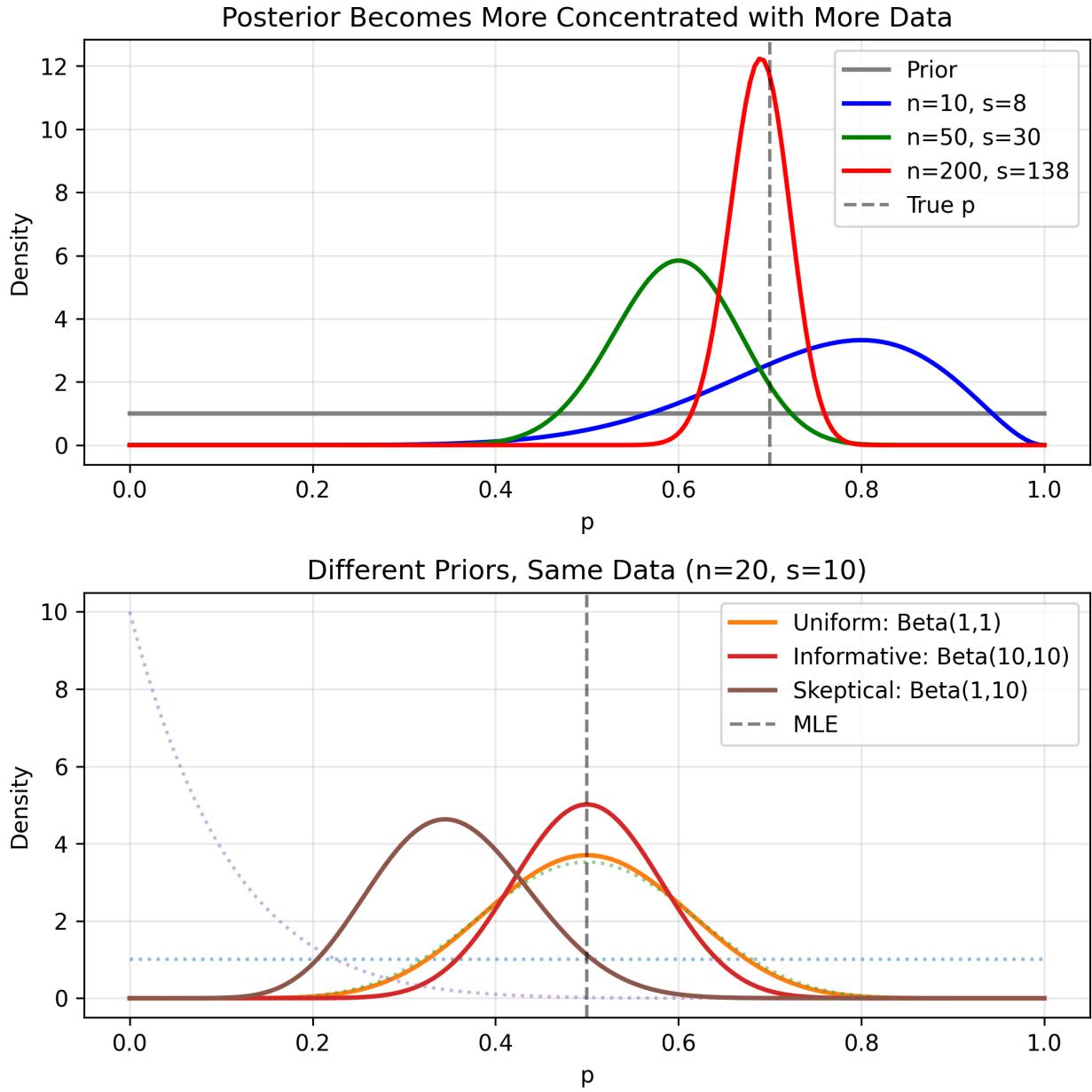
priors = [
    (1, 1, 'Uniform: Beta(1,1)'),
    (10, 10, 'Informative: Beta(10,10)'),
    (1, 10, 'Skeptical: Beta(1,10)')
]

for (alpha, beta, label) in priors:
    # Prior
    prior_y = stats.beta.pdf(p_range, alpha, beta)
    ax2.plot(p_range, prior_y, linestyle=':', alpha=0.5)

    # Posterior
    alpha_post = alpha + s
    beta_post = beta + n - s
    post_y = stats.beta.pdf(p_range, alpha_post, beta_post)
    ax2.plot(p_range, post_y, linewidth=2, label=label)

ax2.axvline(0.5, color='black', linestyle='--', alpha=0.5, label='MLE')
ax2.set_xlabel('p')
ax2.set_ylabel('Density')
ax2.set_title(f'Different Priors, Same Data (n={n}, s={s})')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



The plots illustrate two key principles:

- Top panel:** As we collect more data, the posterior becomes increasingly concentrated around the true value, regardless of the prior.
- Bottom panel:** Different priors lead to different posteriors (but this effect diminishes with larger sample sizes).

8.4.1.2 Example: The Normal-Normal Model

Now consider estimating the mean of a Normal distribution with known variance. Let $X_i \sim \mathcal{N}(\theta, \sigma^2)$ where σ^2 is known (this assumption simplifies the math and is commonly used in introductory examples).

Prior: $\theta \sim \mathcal{N}(\theta_0, \sigma_0^2)$

Likelihood: For IID data, the sufficient statistic is the sample mean \bar{x} , and:

$$\bar{x}|\theta \sim \mathcal{N}(\theta, \sigma^2/n)$$

Given a likelihood $X_i|\theta \sim \mathcal{N}(\theta, \sigma^2)$ (known σ^2) and a prior $\theta \sim \mathcal{N}(\theta_0, \sigma_0^2)$, the posterior is:

$$\theta|x^n \sim \mathcal{N}(\theta_*, \sigma_*^2)$$

where:

- **Posterior Precision:** $\frac{1}{\sigma_*^2} = \frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}$
- **Posterior Mean:** $\theta_* = \sigma_*^2 \left(\frac{\theta_0}{\sigma_0^2} + \frac{n\bar{x}}{\sigma^2} \right)$

Interpretation:

- Precision (inverse variance) is additive: posterior precision = prior precision + data precision
- The posterior mean is a precision-weighted average of prior mean and sample mean
- More precise information gets more weight
- The posterior mean interpolates between the prior mean and sample mean, with the weight given to the data increasing as n increases
- Posterior is always more precise than either prior or likelihood alone

```
def normal_normal_posterior(prior_mean, prior_var, data_mean, data_var, n):
    """
    Calculate posterior parameters for Normal-Normal conjugate model.

    Parameters:
    -----------
    prior_mean : Prior mean
    prior_var : Prior variance **2
    data_mean : Sample mean x̄
    data_var : Known data variance **2
    n : Sample size

    Returns:
    -----------
    post_mean : Posterior mean *
    post_var : Posterior variance **2
    """
    # Convert to precisions (inverse variances)
    prior_precision = 1 / prior_var
    data_precision = n / data_var

    # Posterior precision is sum of precisions
    post_precision = prior_precision + data_precision
    post_var = 1 / post_precision

    # Posterior mean is precision-weighted average
    post_mean = post_var * (prior_precision * prior_mean +
                           data_precision * data_mean)

    return post_mean, post_var

# Example calculation
prior_mean, prior_var = 0, 4 # Prior: N(0, 4)
data_var = 1 # Known variance
n = 10
data_mean = 2.3 # Observed sample mean
```

```

post_mean, post_var = normal_normal_posterior(
    prior_mean, prior_var, data_mean, data_var, n
)

print(f"Prior: N({prior_mean}, {prior_var})")
print(f"Data: n={n}, \bar{x}={data_mean}, \sigma^2={data_var}")
print(f"Posterior: N({post_mean:.3f}, {post_var:.3f})")
print(f"\nPosterior mean is {post_mean:.3f}, between prior mean {prior_mean} and MLE {data_mean}")

Prior: N(0, 4)
Data: n=10, \bar{x}=2.3, \sigma^2=1
Posterior: N(2.244, 0.098)

Posterior mean is 2.244, between prior mean 0 and MLE 2.3

```

8.4.2 The Art and Science of Choosing Priors

One of the most debated topics in Bayesian statistics is how to choose the prior distribution. Critics argue that priors introduce subjectivity; advocates counter that they make assumptions explicit. The reality is nuanced: prior choice is both an art requiring judgment and a science with established principles.

Conjugate Priors: We've seen these in action – Beta for Bernoulli, Normal for Normal. They're computationally convenient and have nice interpretations (like pseudo-counts), but they may not reflect genuine prior beliefs. Using them just for convenience can lead to misleading results.

Non-Informative Priors: These attempt to be “objective” by letting the data speak for itself. Common choices include:

- Uniform priors: $f(\theta) = \text{constant}$
- Jeffreys' prior: $f(\theta) \propto \sqrt{I(\theta)}$ where $I(\theta)$ is the Fisher information

⚠ The Flat Prior Fallacy

A uniform prior is not “uninformative”! Consider:

1. **Scale matters:** A uniform prior on $[-10^6, 10^6]$ says $|\theta|$ is almost certainly large
2. **Not transformation invariant:** If $p \sim \text{Uniform}(0, 1)$, then $\log(p/(1-p))$ is not uniform
3. **Can encode strong beliefs:** Uniform on $[0, 1000]$ for a rate parameter implies most mass is on very large values (highly informative!)

The notion of “no information” is not well-defined mathematically.

ℹ Advanced: Jeffreys' Prior

Jeffreys proposed using $f(\theta) \propto \sqrt{I(\theta)}$ where $I(\theta)$ is the Fisher information. This prior has a key property: it's invariant to reparameterization. If we transform θ to $\varphi = g(\theta)$, the Jeffreys prior for φ is what we'd get by transforming the Jeffreys prior for θ .

For Bernoulli(p), the Jeffreys prior is Beta(1/2, 1/2), which is U-shaped, putting more mass near 0 and 1 than at 0.5 – hardly “uninformative”!

Weakly Informative Priors: This is the recommended approach nowadays which balances several goals:

- Wide enough to not exclude plausible values
- Tight enough to exclude absurd values
- Regularize estimation to prevent overfitting

For example, for a logistic regression coefficient, a $\mathcal{N}(0, 2.5^2)$ prior (mean 0, standard deviation 2.5) allows large effects but prevents numerical instability. Note: We use the $\mathcal{N}(\mu, \sigma^2)$ parameterization throughout these notes.

🔥 The Challenge of High-Dimensional Priors

Placing sensible priors becomes increasingly difficult as dimensionality grows:

- **The Gaussian bubble:** In high dimensions, a multivariate standard normal $\mathcal{N}(0, I)$ concentrates its mass in a thin shell at radius \sqrt{d} from the origin – almost no mass is near zero despite this being the “center” of the distribution. This concentration of measure phenomenon means our intuitions about priors break down (see this [blog post](#)).
- **Deep learning:** Specifying priors for millions of neural network weights remains an open problem in Bayesian deep learning. Most practitioners resort to simple priors like $\mathcal{N}(0, \sigma^2 I)$ that don’t capture the true structure, or avoid fully Bayesian approaches altogether.

High-dimensional Bayesian inference requires careful thought about what the prior actually implies when there are many parameters.

ℹ️ Advanced: The Bayesian Central Limit Theorem

A remarkable result shows that Bayesian and frequentist methods converge with enough data. Under suitable regularity conditions, as $n \rightarrow \infty$, the posterior distribution can be approximated by:

$$f(\theta|x^n) \approx \mathcal{N}(\hat{\theta}_{MLE}, \widehat{se}^2)$$

where $\hat{\theta}_{MLE}$ is the maximum likelihood estimate and \widehat{se} is its standard error.

Why this matters:

1. With enough data, the prior becomes irrelevant
2. Bayesian credible intervals Frequentist confidence intervals
3. Both approaches give essentially the same answer
4. The likelihood dominates both approaches in large samples

This is reassuring: two philosophically different approaches converge to the same practical conclusions when we have sufficient evidence.

8.4.3 Implementing Bayesian Inference

For the conjugate models in this chapter, we can solve for the posterior distribution analytically. But what happens in more complex, real-world models where this is not possible?

The modern Bayesian workflow relies on powerful computational algorithms, most commonly **Markov chain Monte Carlo (MCMC)**. These algorithms allow us to generate a large collection of samples that are representative of the posterior distribution, even when we cannot solve for it directly. Once we have these samples, we can approximate any summary we need (like the mean or a credible interval) and easily get posteriors for transformed parameters.

Modern **probabilistic programming frameworks** such as [Stan](#) or [PyMC](#) allow users to perform Bayesian inference relatively easily, exploiting modern machinery. This computational approach is incredibly powerful and flexible, and we will explore it in detail in Chapter 10. For now, the key takeaway is that the goal of Bayesian inference is always to obtain the posterior; the methods in this chapter do it with math, while later methods will do it with computation.

8.5 Statistical Decision Theory: A Framework for “Best”

We now shift from Bayesian inference to a more general question: given multiple ways to estimate a parameter, how do we choose the best one? Statistical decision theory provides a formal framework for comparing any

estimators – Bayesian, frequentist, or otherwise.

8.5.1 The Ingredients: Loss and Risk

To compare estimators formally, we need to quantify how “wrong” an estimate is.

Loss Function $L(\theta, \hat{\theta})$: Quantifies the penalty for estimating θ with $\hat{\theta}$.

Common examples:

- **Squared Error:** $L_2(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$
- **Absolute Error:** $L_1(\theta, \hat{\theta}) = |\theta - \hat{\theta}|$
- **L_p Loss:** $L_p(\theta, \hat{\theta}) = |\theta - \hat{\theta}|^p$ for $p \geq 1$ (generalizes the above)
- **Zero-One Loss:** $L_{0-1}(\theta, \hat{\theta}) = \begin{cases} 0 & \text{if } \theta = \hat{\theta} \\ 1 & \text{otherwise} \end{cases}$

Loss tells us the cost of a specific estimate for a specific parameter value. But estimators are random – they depend on data. So we need to average:

Risk $R(\theta, \hat{\theta})$: The expected loss over all possible datasets, for a given loss function \mathcal{L} and fixed θ .

$$R(\theta, \hat{\theta}) = \mathbb{E}_{\theta}[L(\theta, \hat{\theta}(X))] = \int L(\theta, \hat{\theta}(x))f(x; \theta)dx$$

For squared error loss, the risk equals the MSE:

$$R(\theta, \hat{\theta}) = \mathbb{E}_{\theta}[(\theta - \hat{\theta})^2] = \text{MSE} = \mathbb{V}(\hat{\theta}) + \text{Bias}^2(\hat{\theta})$$

Multiple Perspectives

Intuitive

Think of risk as the “average wrongness” of an estimator, for a specific definition of “wrongness” (loss function). Imagine you could repeat your experiment many times with the same true parameter value. Each time, you’d get different data and thus a different estimate. The risk tells you the average penalty you’d pay across all these repetitions.

It’s like evaluating a weather forecaster: you don’t judge them on one prediction, but on their average performance over many days. An estimator with low risk is consistently good, even if it’s not perfect on any single dataset.

Mathematical

Risk is the frequentist expectation of the loss function, treating the estimator as a random variable (through its dependence on random data) while holding the parameter fixed:

$$R(\theta, \hat{\theta}) = \mathbb{E}_{X \sim f(x; \theta)}[L(\theta, \hat{\theta}(X))]$$

This contrasts with Bayes risk, which averages over θ according to a prior. The risk function $R(\theta, \cdot)$ maps each parameter value to a real number, creating a curve that characterizes the estimator’s performance across the parameter space.

For squared error loss, the bias-variance decomposition shows that risk combines systematic error (bias) with variability (variance), revealing the fundamental tradeoff in estimation.

Computational

We can understand risk concretely through simulation. The following code demonstrates what risk actually means by repeatedly generating datasets from the same distribution and computing the squared loss for each one:

```

import numpy as np

def estimate_risk_by_simulation(true_theta, estimator_func,
                                n_samples=20, n_simulations=10000):
    """
    Estimate risk via Monte Carlo simulation.

    This shows what risk really means: the average loss
    over many possible datasets.
    """
    losses = []

    for _ in range(n_simulations):
        # Generate a dataset (example: Normal distribution)
        data = np.random.normal(true_theta, 1, n_samples)

        # Compute the estimate for this dataset
        estimate = estimator_func(data)

        # Compute the loss (using squared error)
        loss = (estimate - true_theta)**2
        losses.append(loss)

    # Risk is the average loss
    risk = np.mean(losses)

    print(f"True parameter: {true_theta}")
    print(f"Estimated risk: {risk:.4f}")
    print(f"Min loss seen: {np.min(losses):.4f}")
    print(f"Max loss seen: {np.max(losses):.4f}")

    return risk

# Example: Risk of sample mean estimator
risk = estimate_risk_by_simulation(
    true_theta=5.0,
    estimator_func=lambda data: np.mean(data),
    n_samples=20
)
True parameter: 5.0
Estimated risk: 0.0503
Min loss seen: 0.0000
Max loss seen: 0.8506

```

Key Takeaway: Risk is the *average* loss across all possible datasets. The simulation shows that while individual losses vary widely (min to max), risk captures the expected performance of an estimator.

8.5.2 The Challenge of Comparing Risk Functions

Risk functions are curves – one risk value for each possible θ . This creates a fundamental problem: estimators rarely dominate uniformly.

Example: A Simple Case

Let $X \sim \mathcal{N}(\theta, 1)$ and let's assume squared error loss. Compare:

- $\hat{\theta}_1 = X$ (the sensible estimator)
- $\hat{\theta}_2 = 3$ (a silly constant estimator)

Risk calculations:

- For $\hat{\theta}_1 = X$:

$$R(\theta, \hat{\theta}_1) = \mathbb{E}_{\theta}[(X - \theta)^2] = \text{Var}(X) = 1$$

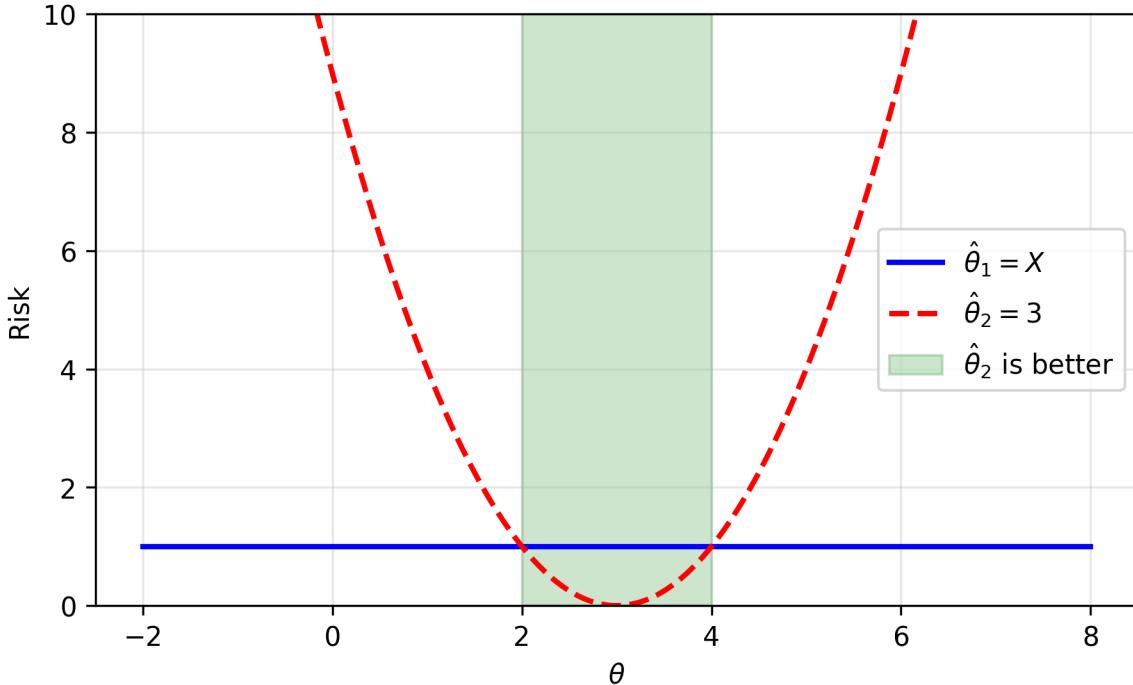
This is constant for all θ .

- For $\hat{\theta}_2 = 3$:

$$R(\theta, \hat{\theta}_2) = \mathbb{E}_{\theta}[(3 - \theta)^2] = (3 - \theta)^2$$

This depends on θ since the estimator is non-random.

Risk Functions Can Cross



The constant estimator is actually better when θ is near 3 (“a broken clock is right twice a day”), but terrible elsewhere. Still, neither *uniformly* dominates the other.

Example: Bernoulli Estimation

Consider $X_1, \dots, X_n \sim \text{Bernoulli}(p)$ with squared error loss. Let $S = \sum_{i=1}^n X_i$ be the number of successes.

We'll compare two natural estimators by computing their risk functions.

Estimator 1: MLE

The MLE is $\hat{p}_1 = \bar{X} = \frac{S}{n}$

Since $S \sim \text{Binomial}(n, p)$, we have:

- $\mathbb{E}[S] = np$, so $\mathbb{E}[\hat{p}_1] = p$ (unbiased)
- $\text{Var}(S) = np(1-p)$, so $\text{Var}(\hat{p}_1) = \frac{p(1-p)}{n}$

The risk under squared error loss is:

$$R(p, \hat{p}_1) = \mathbb{E}[(\hat{p}_1 - p)^2] = \text{Var}(\hat{p}_1) + \text{Bias}^2(\hat{p}_1) = \frac{p(1-p)}{n} + 0 = \frac{p(1-p)}{n}$$

This is a parabola with maximum at $p = 1/2$.

Estimator 2: Bayesian posterior mean with Beta(,) prior

Using Bayes' theorem with prior $p \sim \text{Beta}(\alpha, \beta)$ and observing S successes:

$$p|S \sim \text{Beta}(\alpha + S, \beta + n - S)$$

The posterior mean⁴ is:

$$\hat{p}_2 = \frac{\alpha + S}{\alpha + \beta + n}$$

To find the risk, we compute bias and variance:

- Expected value: $\mathbb{E}[\hat{p}_2] = \frac{\alpha + np}{\alpha + \beta + n}$
- Bias: $\text{Bias}(\hat{p}_2) = \frac{\alpha + np}{\alpha + \beta + n} - p = \frac{\alpha - p(\alpha + \beta)}{\alpha + \beta + n}$
- Variance: $\text{Var}(\hat{p}_2) = \text{Var}\left(\frac{S}{\alpha + \beta + n}\right) = \frac{np(1-p)}{(\alpha + \beta + n)^2}$

Therefore, the general risk formula is:

$$R(p, \hat{p}_2) = \frac{np(1-p)}{(\alpha + \beta + n)^2} + \left(\frac{\alpha - p(\alpha + \beta)}{\alpha + \beta + n}\right)^2$$

Special case: Uniform prior Beta(1, 1)

For $\alpha = \beta = 1$, the posterior mean becomes:

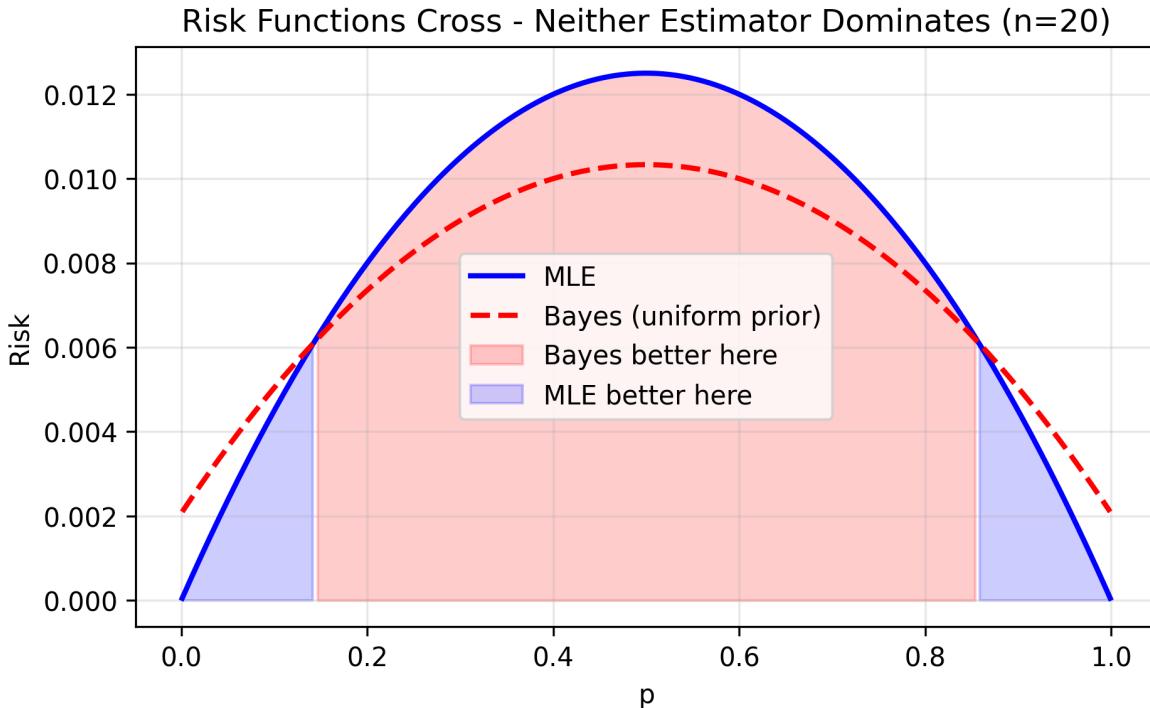
$$\hat{p}_2 = \frac{1 + S}{2 + n} = \frac{n}{n + 2} \cdot \frac{S}{n} + \frac{2}{n + 2} \cdot \frac{1}{2}$$

This is a weighted average of the MLE and the prior mean $1/2$.

The risk specializes to:

$$R(p, \hat{p}_2) = \frac{np(1-p)}{(n+2)^2} + \left(\frac{1 - 2p}{n+2}\right)^2$$

Let's plot both risk functions to see how they compare:



The risk functions cross! The MLE is better near the extremes (p near 0 or 1), while the Bayes estimator is better near the middle (p near 1/2). Neither estimator uniformly dominates the other.

Both examples above illustrate a fundamental challenge in decision theory: when risk functions cross, we cannot declare one estimator uniformly better than another. Different estimators excel in different regions of the parameter space.

To make a decision on the estimator to use, we must reduce these risk curves to single numbers that we can compare. But how should we summarize an entire function? Should we care most about average performance or worst-case performance? Different answers to this question lead to two distinct optimality criteria: **Bayes estimators** (optimizing average risk) and **minimax estimators** (optimizing worst-case risk), detailed in the next section.

8.6 Optimal Estimators: Bayes and Minimax Rules

8.6.1 The Bayesian Approach: Minimizing Average Risk

The Bayesian approach to decision theory averages the risk over a prior distribution, giving us a single number to minimize.

Bayes Risk: The expected risk, averaged over the prior distribution $f(\theta)$:

$$r(f, \hat{\theta}) = \int R(\theta, \hat{\theta}) f(\theta) d\theta$$

Bayes Estimator: The estimator $\hat{\theta}^B$ that minimizes the Bayes risk:

$$\hat{\theta}^B = \arg \min_{\hat{\theta}} r(f, \hat{\theta})$$

The remarkable connection between Bayesian inference and decision theory:⁴

⁴The posterior mean is the Bayes estimator under squared error loss, as we will see in the following section.

The Bayes estimator can be found by minimizing the *posterior* expected loss for each observed x . Specifically:

- For **Squared Error Loss**: The Bayes estimator is the **Posterior Mean**
- For **Absolute Error Loss**: The Bayes estimator is the **Posterior Median**
- For **Zero-One Loss**: The Bayes estimator is the **Posterior Mode (MAP)**

i Proof for Different Loss Functions

For any loss function $L(\theta, \hat{\theta})$, the Bayes estimator minimizes the posterior expected loss:

$$\hat{\theta}^B(x) = \arg \min_a \mathbb{E}[L(\theta, a)|X = x] = \arg \min_a \int L(\theta, a)f(\theta|x)d\theta$$

Squared Error Loss: $L(\theta, a) = (\theta - a)^2$

We need to minimize:

$$\int (\theta - a)^2 f(\theta|x)d\theta$$

Taking the derivative with respect to a and setting to zero:

$$\frac{d}{da} \int (\theta - a)^2 f(\theta|x)d\theta = -2 \int (\theta - a)f(\theta|x)d\theta = 0$$

This gives:

$$\int \theta f(\theta|x)d\theta = a \int f(\theta|x)d\theta = a$$

Therefore: $\hat{\theta}^B(x) = \int \theta f(\theta|x)d\theta = \mathbb{E}[\theta|X = x]$ (posterior mean)

Absolute Error Loss: $L(\theta, a) = |\theta - a|$

We need to minimize:

$$\int |\theta - a|f(\theta|x)d\theta = \int_{-\infty}^a (a - \theta)f(\theta|x)d\theta + \int_a^{\infty} (\theta - a)f(\theta|x)d\theta$$

Taking the derivative with respect to a :

$$\frac{d}{da} = \int_{-\infty}^a f(\theta|x)d\theta - \int_a^{\infty} f(\theta|x)d\theta = F(a|x) - (1 - F(a|x)) = 2F(a|x) - 1$$

Setting to zero: $F(a|x) = 1/2$, so $\hat{\theta}^B(x)$ is the posterior median.

Zero-One Loss: $L(\theta, a) = \mathbb{1}\{\theta \neq a\}$

The expected loss is:

$$\mathbb{E}[L(\theta, a)|X = x] = P(\theta \neq a|X = x) = 1 - P(\theta = a|X = x)$$

This is minimized when $P(\theta = a|X = x)$ is maximized, which occurs at the posterior mode.

This theorem reveals a profound insight: Bayesian inference naturally produces optimal estimators! The posterior summaries we compute for inference are exactly the estimators that minimize expected loss.

8.6.2 The Frequentist Approach: Minimizing Worst-Case Risk

The minimax approach takes a pessimistic view: prepare for the worst case.

Maximum Risk: The worst-case risk over the entire parameter space Θ :

$$\bar{R}(\hat{\theta}) = \sup_{\theta \in \Theta} R(\theta, \hat{\theta})$$

Minimax Estimator: The estimator $\hat{\theta}^{MM}$ with the smallest maximum risk:

$$\hat{\theta}^{MM} = \arg \min_{\hat{\theta}} \sup_{\theta} R(\theta, \hat{\theta})$$

It's the “best of the worst-case” estimators.

Finding minimax estimators directly is usually difficult. However, there's a powerful connection to Bayes estimators:

If a Bayes estimator has constant risk (the same risk for all θ), then it is minimax.

This gives us a recipe: find a prior such that the resulting Bayes estimator has constant risk.

Example: Minimax Estimator for Bernoulli

Consider $X_1, \dots, X_n \sim \text{Bernoulli}(p)$ with squared error loss. We know from the previous example that the Bayes estimator with prior $\text{Beta}(\alpha, \beta)$ has risk:

$$R(p, \hat{p}) = \frac{np(1-p)}{(\alpha + \beta + n)^2} + \left(\frac{\alpha - p(\alpha + \beta)}{\alpha + \beta + n} \right)^2$$

The key insight: Can we choose α and β to make this risk constant (independent of p)? If so, the constant risk theorem tells us the resulting estimator would be minimax.

It turns out that setting $\alpha = \beta = \sqrt{n/4}$ does exactly this!

i Derivation of the minimax prior

If we set $\alpha = \beta$, the risk becomes:

$$R(p, \hat{p}) = \frac{1}{(2\alpha + n)^2} [np(1-p) + \alpha^2(1-2p)^2]$$

Expanding the term in brackets:

$$\begin{aligned} np(1-p) + \alpha^2(1-2p)^2 &= np - np^2 + \alpha^2(1-4p+4p^2) \\ &= \alpha^2 + (n-4\alpha^2)p + (4\alpha^2-n)p^2 \end{aligned}$$

This is constant if and only if the coefficients of p and p^2 are both zero:
- Coefficient of p : $n - 4\alpha^2 = 0 \Rightarrow \alpha^2 = n/4$
- Coefficient of p^2 : $4\alpha^2 - n = 0 \Rightarrow \alpha^2 = n/4$
Both conditions give the same answer: $\alpha = \sqrt{n/4}$.

With $\alpha = \beta = \sqrt{n/4}$:

- The Bayes estimator is: $\hat{p} = \frac{S+\sqrt{n/4}}{n+\sqrt{n}}$
- The risk is constant: $R(p, \hat{p}) = \frac{n}{4(n+\sqrt{n})^2}$ for all p
- By the theorem above, this makes it minimax!

How does this minimax estimator compare to the MLE?

By maximum risk (worst-case criterion):

- MLE: Maximum risk is $\frac{1}{4n}$ (at $p = 1/2$)
- Minimax: Constant risk $\frac{n}{4(n+\sqrt{n})^2} < \frac{1}{4n}$

The minimax estimator wins on worst-case performance - that's what it was designed for!

But here's the interesting part: even though the minimax estimator was derived from a $\text{Beta}(\sqrt{n/4}, \sqrt{n/4})$ prior, we can ask how it performs on average under *any* prior. For instance, under a uniform prior:

- MLE: Bayes risk = $\frac{1}{6n}$
- Minimax: Bayes risk = $\frac{n}{4(n+\sqrt{n})^2}$

For $n \geq 20$, the MLE has lower average risk under the uniform prior. This illustrates a key principle: **the minimax estimator optimizes worst-case performance, but may sacrifice average-case performance to achieve this robustness.**

Example: Minimax Estimator for Normal Mean

For $X_1, \dots, X_n \sim \mathcal{N}(\theta, 1)$, the sample mean \bar{X} has risk:

$$R(\theta, \bar{X}) = \mathbb{E}[(\bar{X} - \theta)^2] = \text{Var}(\bar{X}) = \frac{1}{n}$$

This risk is constant (doesn't depend on θ). Furthermore, \bar{X} can be shown to be admissible (as it is the limit of admissible Bayes estimators for Normal priors). An admissible estimator with constant risk is minimax. Therefore, \bar{X} is minimax.

This result holds for any “well-behaved” loss function (convex and symmetric about the origin).

Example: Large Sample MLE

In most parametric models with large n , the MLE is approximately minimax. The intuition: as $n \rightarrow \infty$, the MLE becomes approximately Normal with variance $1/(nI(\theta))$ where $I(\theta)$ is the Fisher information. In many regular models, this leads to approximately constant risk.

Important caveat: This breaks down when the number of parameters grows with n . For example, in the “many Normal means” problem where we estimate n means from n observations, the MLE is far from minimax.

8.7 Admissibility: Ruling Out Bad Estimators

Minimax and Bayes estimators tell us about optimality according to specific criteria. But there's a more basic requirement: an estimator shouldn't be uniformly worse than another.

8.7.1 Defining Admissibility

An estimator $\hat{\theta}$ is **inadmissible** if there exists another estimator $\hat{\theta}'$ such that:

1. $R(\theta, \hat{\theta}') \leq R(\theta, \hat{\theta})$ for all θ
2. $R(\theta, \hat{\theta}') < R(\theta, \hat{\theta})$ for at least one θ

Otherwise, $\hat{\theta}$ is **admissible**.

An inadmissible estimator is dominated – there's another estimator that's never worse and sometimes better. Using an inadmissible estimator is irrational.

i Admissibility Good

The constant estimator $\hat{\theta} = 3$ for $X \sim \mathcal{N}(\theta, 1)$ is admissible! Why? Any estimator that beats it at $\theta = 3$ must be worse elsewhere. But it's still a terrible estimator for most purposes.

Admissibility is a necessary but not sufficient condition for a good estimator.

8.7.2 Key Properties and Connections

A Bayes estimator for a prior with full support (positive density everywhere) is always admissible.

This is powerful: Bayesian methods automatically avoid inadmissible estimators.

Other connections:

- **Constant Risk and Admissibility:** An admissible estimator with constant risk is minimax
- **Minimax and Admissibility:** Minimax estimators are usually admissible or “nearly” admissible
- **MLE and Admissibility:** The MLE is not always admissible, especially in high dimensions

i Advanced: Stein’s Paradox and Shrinkage

Consider estimating $k \geq 3$ Normal means simultaneously. Let $Y_i \sim \mathcal{N}(\theta_i, 1)$ for $i = 1, \dots, k$.

The Setup: We want to estimate $\theta = (\theta_1, \dots, \theta_k)$ with total squared error loss:

$$L(\theta, \hat{\theta}) = \sum_{i=1}^k (\theta_i - \hat{\theta}_i)^2$$

The Paradox: The “obvious” estimator $\hat{\theta}_i = Y_i$ (using each observation to estimate its own mean) is inadmissible when $k \geq 3$!

The Solution: The James-Stein estimator

$$\hat{\theta}_i^{JS} = \left(1 - \frac{k-2}{\sum_j Y_j^2}\right)^+ Y_i$$

“shrinks” estimates toward zero and has uniformly lower risk than the MLE.

The Importance: This counterintuitive result revolutionized high-dimensional statistics. It shows that when estimating many parameters simultaneously, we can improve by “borrowing strength” across parameters. This is the foundation of modern regularization methods in machine learning.

The key insight: in high dimensions, the MLE can be improved by shrinkage toward a common value.

8.8 Chapter Summary and Connections

8.8.1 Key Concepts Review

Bayesian Inference:

- **Posterior Likelihood \times Prior:** Bayes’ theorem provides the recipe for updating beliefs
- **Conjugate models:** Beta-Bernoulli and Normal-Normal give closed-form posteriors
- **Prior choice matters:** Conjugate (convenient), non-informative (problematic), weakly informative (recommended)
- **Credible intervals:** Direct probability statements about parameters

Statistical Decision Theory:

- **Loss functions:** Quantify the cost of estimation errors
- **Risk functions:** Expected loss – curves that are hard to compare
- **Bayes estimators:** Minimize average risk over a prior
- **Minimax estimators:** Minimize worst-case risk

Key Connections:

- Posterior mean = Bayes estimator for squared error loss
- Constant risk Bayes rules are minimax
- Bayes rules are admissible
- In large samples, Bayesian and frequentist methods converge

8.8.2 The Big Picture

This chapter revealed two fundamental insights:

1. **Bayesian inference provides a unified, probabilistic framework** for learning from data. By treating parameters as random variables, we can make direct probability statements and naturally incorporate prior knowledge.
2. **Decision theory provides a formal language** for evaluating and comparing any statistical procedure. The posterior mean is not just an arbitrary summary – it's the optimal estimator under squared error loss.

The connection runs deeper: Bayesian methods naturally produce optimal estimators, while decision theory helps us understand when and why different approaches work well. Even frequentist stalwarts use decision theory, and the best frequentist estimators often have Bayesian interpretations.

8.8.3 Common Pitfalls to Avoid

1. **Confusing credible and confidence intervals:** A 95% credible interval contains θ with probability 0.95 given the data.⁵ A 95% confidence interval is produced by a procedure that traps θ in 95% of repeated experiments.
2. **Thinking uniform priors are “uninformative”:** They encode specific beliefs and aren't transformation invariant.
3. **Using conjugate priors blindly:** Convenience shouldn't override reasonable prior beliefs.
4. **Forgetting the prior's influence diminishes:** With enough data, different reasonable priors lead to similar posteriors.⁶
5. **Assuming admissible = good:** The constant estimator $\hat{\theta} = 3$ is admissible but useless.

8.8.4 Chapter Connections

- **Previous (Ch. 5-7):** We learned frequentist methods for finding and evaluating estimators. Now we have a completely different paradigm (Bayesian) and a unified theory (decision theory) for comparing estimators from any paradigm.
- **This Chapter:** Introduced Bayesian thinking and formal decision theory. These provide alternative and complementary approaches to the frequentist methods we've studied.
- **Next (Ch. 10):** We'll see how modern computational methods (MCMC, Stan) make Bayesian inference practical for complex models where conjugacy doesn't help.
- **Applications:** Bayesian methods shine in hierarchical models, missing data problems, and anywhere prior information is valuable.

8.8.5 Self-Test Problems

1. **Bayesian Calculation:** Given $n = 10$ observations from a $\text{Poisson}(\lambda)$ distribution with $\sum x_i = 30$, and a prior $\lambda \sim \text{Gamma}(2, 1)$, find the posterior distribution for λ . What is the Bayes estimator under squared error loss?

 Solution Hint

The Gamma distribution is conjugate to the Poisson. Using the shape-rate parameterization (where β is the rate parameter), if $\lambda \sim \text{Gamma}(\alpha, \beta)$ and we observe data with sum S , then $\lambda|data \sim \text{Gamma}(\alpha + S, \beta + n)$. The posterior mean (Bayes estimator) is $(\alpha + S)/(\beta + n)$.

2. **Decision Theory Concepts:** Let $X_1, \dots, X_n \sim \mathcal{N}(\mu, 1)$. The MLE $\hat{\mu} = \bar{X}$ has risk $1/n$ under squared error loss.
 - (a) Is this risk constant?

⁵For a given model and prior.

⁶In regular, identifiable parametric models; this can fail in high dimensions or weakly identified settings.

- (b) How does $\hat{\mu}$ relate to the Bayes estimator under a Normal prior? (One sentence.)
- (c) Is $\hat{\mu}$ minimax? Give a one-line justification.
- (d) Is $\hat{\mu}$ admissible? Give a one-line justification.

i Solution Hint

- (a) Yes, $R(\mu, \bar{X}) = \text{Var}(\bar{X}) = 1/n$ is constant.
- (b) With prior $\mu \sim \mathcal{N}(a, b^2)$, the Bayes estimator is the posterior mean $w\bar{X} + (1-w)a$, where $w = \frac{b^2}{b^2+1/n}$; as $b^2 \rightarrow \infty$ (very diffuse prior), this approaches \bar{X} .
- (c) Constant risk + admissibility minimax (by results in the notes).
- (d) Yes, in the 1D Normal-mean problem with squared error, \bar{X} is admissible (classical result), even though it's a limit of Bayes rules.

3. **Prior Choice:** You're estimating the probability p that a new medical treatment works. You're skeptical because most new treatments fail. What would be:

- A weakly informative prior for p ?
- A strong prior reflecting your skepticism?

i Solution Hint

Weakly informative: Beta(1, 3) or Beta(1, 5) - allows all values but slightly favors lower success rates. Strong skeptical prior: Beta(1, 10) or Beta(1, 20) - strongly concentrates mass near 0, reflecting belief that the treatment likely doesn't work. Remember: Beta parameters can be interpreted as pseudo-counts of successes and failures.

4. **Conceptual Understanding:** Why is the James-Stein estimator's improvement over the MLE considered paradoxical? What does it tell us about estimating multiple parameters simultaneously?

i Solution Hint

The paradox: When estimating three or more unrelated means (e.g., baseball batting average, physics constants, and rainfall), using information from all of them together (via shrinkage) gives better estimates than treating them separately. This violates our intuition that unrelated problems should be solved independently. The lesson: In high dimensions, "borrowing strength" across parameters through shrinkage reduces overall risk, even for unrelated parameters.

8.8.6 Python and R Reference

i Python and R Reference Code

Python and R code examples for Bayesian inference and decision theory can be found in the HTML version of these notes.

8.8.7 Connections to Source Material

i Mapping to “All of Statistics”

Lecture Note Section	Corresponding Source(s)
Introduction: A Different Way of Thinking	From slides and AF447 case study from Stone et al. (2014)
The Two Philosophies of Statistics	AoS §11.1 and slides
The Bayesian Method: Updating Beliefs with Data	AoS §11.2
The Engine: Bayes’ Theorem for Inference	AoS §11.2 plus slides
Summarizing the Posterior	AoS §11.2; AoS §12.3 (for median/mode)
Credible vs. Confidence Intervals	AoS §11.9 and expanded from lecture material
Bayesian Inference in Action	
Conjugate Models and Conjugate Priors	AoS §11.2
Example: The Bernoulli-Beta Model	AoS Example 11.1 and slides
Example: The Normal-Normal Model	AoS Example 11.2 and slides
The Art and Science of Choosing Priors	AoS §11.6 expanded with modern views
Implementing Bayesian Inference	AoS §11.4 expanded with modern tools
Statistical Decision Theory	AoS Ch 12
The Ingredients: Loss and Risk	AoS §12.1
The Challenge of Comparing Risk Functions	AoS §12.2 (Examples 12.2, 12.3)
Optimal Estimators: Bayes and Minimax Rules	
The Bayesian Approach: Minimizing Average Risk	AoS §12.3
The Frequentist Approach: Minimizing Worst-Case Risk	AoS §12.4
Example: Minimax Estimator for Bernoulli	AoS Example 12.12
Example: Minimax Estimator for Normal Mean	AoS Theorem 12.14, Theorem 12.22
Example: Large Sample MLE	AoS §12.5
Admissibility: Ruling Out Bad Estimators	AoS §12.6
Defining Admissibility	AoS Definition 12.17
Key Properties and Connections	AoS Theorem 12.19, Theorem 12.21
Advanced: Stein’s Paradox and Shrinkage	AoS §12.7
Self-Test Problems	Based on AoS Ch 11/12 exercises and concepts

8.8.8 Further Materials

- **Foundational Text:** Gelman et al., “Bayesian Data Analysis” (3rd ed.)
- **The Air France Search Case Study:** Stone et al. (2014).
- **Prior Choice:** See the [Stan wiki](#).

Remember: Bayesian inference is about updating beliefs with data. Decision theory is about choosing the best estimator. Together, they provide a complete framework for statistical inference that complements and enriches the frequentist approach. Master both paradigms – they each have their place in the modern statistician’s toolkit!

Chapter 9

Linear and Logistic Regression

9.1 Learning Objectives

After completing this chapter, you will be able to:

- **Build and interpret linear regression models** to understand relationships between variables, connecting the geometric intuition of least squares with the statistical framework of maximum likelihood estimation.
- **Evaluate regression coefficients meaningfully**, including their practical interpretation, statistical significance via confidence intervals and hypothesis tests, and the crucial distinction between association and causation.
- **Diagnose and address model inadequacies** using residual plots to check assumptions, applying transformations when relationships are nonlinear, and selecting predictors using principled criteria (AIC, BIC, cross-validation).
- **Extend regression to binary outcomes** through logistic regression, understanding how the logit link enables probability modeling and interpreting coefficients as odds ratios.
- **Apply regression for both prediction and explanation**, recognizing when linear models excel (interpretability needs, moderate sample sizes) versus when more complex methods are warranted.

Note

This chapter introduces the two most fundamental and widely used models in statistics: linear and logistic regression. While modern machine learning offers powerful black-box predictors, linear models remain essential for their interpretability and foundational role in statistical inference. The material is adapted from Chapter 13 of Wasserman (2013), supplemented with modern perspectives on model interpretability and practical examples.

9.2 Introduction: Why Linear Models Still Matter

9.2.1 The Power of Interpretability

In an age dominated by complex machine learning models – neural networks with millions of parameters, random forests with thousands of trees, gradient boosting machines with intricate interactions – one might wonder: why dedicate an entire chapter to something as simple as linear regression?

The answer lies in a fundamental trade-off in statistical modeling: **complexity versus interpretability**. While a deep neural network might achieve better prediction accuracy on a complex dataset, it operates as a “black box”. We feed in inputs, we get outputs, but the mechanism connecting them remains opaque. This opacity becomes problematic when we need to:

- **Explain predictions to stakeholders:** “Why was this loan application denied?”
- **Identify which features matter most:** “Which factors most strongly predict patient readmission?”
- **Ensure fairness and avoid bias:** “Is our model discriminating based on protected attributes?”
- **Debug unexpected behavior:** “Why did the model fail on this particular case?”

Linear models excel at all these tasks. Every coefficient has a clear interpretation: it tells us exactly how a one-unit change in a predictor affects the outcome, holding all else constant. This interpretability has made linear models the reference tools in fields where understanding relationships is as important as making predictions – from economics to medicine to social sciences.

i Advanced: Are Neural Networks Black Boxes?

The story we presented above of neural networks being completely black boxes is a simplification nowadays. The field of *interpretability research* has developed sophisticated methods to understand neural networks, including techniques like mechanistic interpretability, activation analysis, and circuit discovery. These approaches have yielded notable insights, for example for [large language models](#).

Still, there's a crucial distinction: linear models are transparent by construction – each coefficient directly tells us how a unit change in input affects output. Neural networks must be **reverse-engineered** through complex analysis requiring specialized tools and expertise. Think of it as the difference between reading a recipe versus doing forensic analysis on a finished cake.

In short, while interpretability research continues to advance, linear models remain uniquely valuable when interpretability is a primary requirement rather than an afterthought – and linear models can be used as tools to understand more complex models, as seen in the next paragraph.

9.2.2 Linear Models as Building Blocks

Beyond their direct application, linear models serve as the foundation for building and understanding more complex methods:

- **Generalized Linear Models (GLMs)** extend linear regression to handle non-normal outcomes
- **Mixed Effects Models** add random effects to account for hierarchical data structures
- **Regularized Regression** (Ridge, LASSO, Elastic Net) adds penalties to handle high-dimensional data
- **Local Linear Methods** like LOESS use linear regression in small neighborhoods for flexible curve fitting

Even in the realm of “black-box” machine learning, linear models play a crucial role in model interpretation. Consider **LIME** (Local Interpretable Model-Agnostic Explanations) (Ribeiro, Singh, and Guestrin 2016), a popular technique for explaining individual predictions from any complex model. LIME works by:

1. Taking a prediction from a complex model (e.g., “This image is 92% likely to contain a cat”)
2. Generating perturbed samples around the input of interest
3. Getting predictions from the complex model for these perturbed samples
4. **Fitting a simple linear model** to approximate the complex model’s behavior locally
5. Using the linear model’s coefficients to explain **which features drove the prediction**

In essence, LIME uses the interpretability of linear models to shed light on the darkness of black-box predictions. When we can’t understand the global behavior of a complex model, we can at least understand its local behavior through the lens of linear approximation.

9.2.3 This Chapter’s Goal

Our goal in this chapter is to master both the theory and practice of linear and logistic regression. We’ll develop the mathematical framework, explore the connections to maximum likelihood estimation, and learn how to implement and interpret these models in practice. Along the way, we’ll address critical questions like:

- How do we estimate regression coefficients and quantify our uncertainty about them?

- What assumptions underlie linear regression, and what happens when they’re violated?
- How do we choose which predictors to include when we have many candidates?
- How do we extend the framework to binary outcomes?

By the end of this chapter, you’ll have a thorough understanding of these fundamental models – knowledge that will serve you whether you’re building interpretable models for scientific research or using LIME to explain neural network predictions.

i Historical Note: The Origins of “Regression”

The term “regression” might seem odd for a method that predicts one variable from others. Its origin lies in the work of [Sir Francis Galton](#) (1822-1911), who studied the relationship between parents’ heights and their children’s heights. Galton observed that while tall parents tended to have tall children and short parents short children, the children’s heights tended to be closer to the population mean than their parents’ heights.

He called this phenomenon “regression towards mediocrity” (later “regression to the mean”). Though the name stuck, modern regression analysis is far more general than Galton’s original application – it’s a comprehensive framework for modeling relationships between variables.

i Finnish Terminology Reference

For Finnish-speaking students, here’s a concise reference table of key terms in this chapter:

English	Finnish	Context
Regression	Regressio	General statistical method
Linear regression	Lineaarinen regressio	Predicting continuous outcomes
Simple linear regression	Yhden selittäjän lineaarinen regressio	One predictor variable
Multiple regression	Usean selittäjän lineaarinen regressio	Multiple predictor variables
Logistic regression	Logistinen regressio	Predicting binary outcomes
Regression function	Regressiofunktio	$r(x) = \mathbb{E}(Y X = x)$
Response variable	Vastemuuttuja	Dependent variable
Predictor variable	Selittävä muuttuja (myös: kovariaatti, piirre)	Independent variable
Least squares	Pienimmän neliösumman menetelmä	Parameter estimation method
Residual Sum of Squares (RSS)	Jäännösneliösumma	Fit criterion minimized by OLS
Residual	Residuaali, jäännös	Difference between observed and predicted
Fitted value	Sovitettu arvo (myös: sovite; ennustettu arvo)	Model prediction for a data point
Fitted line	Sovitettu suora	Line of best fit
Coefficient	Kerroin	Parameter in regression equation
Intercept	Vakiotermi	Constant term in regression
Slope	Kulmakerroin	Rate of change parameter
Standard error	Keskivirhe	Uncertainty of an estimate
Conditional likelihood	Ehdollinen uskottavuus	Likelihood given covariates
R-squared	Selitysaste (R^2)	Proportion of variance explained
Training error	Opetusvirhe	In-sample error
Overfitting	Ylisovitus (tai: liikasovitus)	Model too complex for data

Underfitting	Vajaasovitus	Model too simple for data
AIC	Akaiken informaatiokriteeri	Model selection criterion
BIC	Bayes-informaatiokriteeri	Model selection criterion
Forward search/selection	Etenevä haku (myös: eteenpäin valinta)	Greedy model search
Statistical control	Vakiointi	Including background variables
Cross-validation	Ristiinvalidointi	Model evaluation technique
Leave-one-out cross-validation	Yksi-pois-ristiinvalidointi (myös: yksittäisristiinvalidointi)	CV with one held-out point
p-value	p-arvo	Significance measure for tests
Confidence interval	Luottamusväli	Uncertainty quantification
Odds ratio	Vetosuhde	Effect measure in logistic regression
Logit (log-odds)	Logit-muunnos (logaritmisen vetosuhde)	Link in logistic regression
Interaction term	Vuorovaikutustermi (interaktiotermi)	Effect modification between predictors
Homoscedasticity / Heteroscedasticity	Homo-/heteroskedastisuus	(Non-)constant error variance
Multicollinearity	Multikollinearisuus	Correlation among predictors

9.3 Simple Linear Regression

9.3.1 Regression Models

Regression is a method for studying the relationship between a **response variable** Y and a **covariate** X . The response variable (also called the dependent variable) is what we're trying to understand or predict – exam scores, blood pressure, sales revenue. The covariate is also called a **predictor variable** or **feature** – these are the variables we use to explain or predict the response, such as study hours, weight, or advertising spend.

When we observe pairs of these variables, we naturally ask: how does Y tend to change as X varies? The answer lies in the **regression function**:

$$r(x) = \mathbb{E}(Y | X = x) = \int y f(y | x) dy$$

This function tells us the expected (average) value of Y for any given value of X . Think of it as the systematic part of the relationship – the signal beneath the noise. If we knew $r(x)$ perfectly, we could say “when $X = x$, we expect Y to be around $r(x)$, though individual observations will vary.”

Our goal is to estimate the regression function $r(x)$ from data of the form:

$$(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n) \sim F_{X,Y}$$

where each pair represents one observation drawn from some joint distribution. In this chapter, we take a parametric approach and assume that r has a specific functional form – namely, that it's linear.

9.3.2 Regression Notation

Before diving into linear regression specifically, let's establish a basic notation. *Any* regression model can be written as:

$$Y = \underbrace{r(X)}_{\text{signal}} + \underbrace{\epsilon}_{\text{noise}}$$

where $\mathbb{E}(\epsilon|X) = 0$. This decomposition is fundamental: the observed response equals the **signal** (the systematic component we can predict from X) plus **noise** or **error** (the random variation we cannot predict).¹

i Why can we always write it this way?

The proof is simple. Define $\epsilon = Y - r(X)$, then:

$$Y = Y + r(X) - r(X) = r(X) + \epsilon.$$

Moreover, since $r(X) = \mathbb{E}(Y | X)$ by definition, we have:

$$\mathbb{E}(\epsilon | X) = \mathbb{E}(Y - r(X) | X) = \mathbb{E}(Y | X) - r(X) = 0$$

This shows that the decomposition isn't just a notational convenience – it's a mathematical fact that any joint distribution of (X, Y) can be decomposed into a predictable part (the conditional expectation) and an unpredictable part (the zero-mean error).

This notation separates what's predictable (the regression function) from what's unpredictable (the error term). The art and science of regression lies in finding good estimates for $r(x)$ from finite data.

9.3.3 The Simple Linear Regression Model

In **simple linear regression**, we assume $r(x)$ is a linear function of one-dimensional X :

$$r(x) = \beta_0 + \beta_1 x$$

This defines the simple linear regression model:

Simple Linear Regression Model

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

where $\mathbb{E}(\epsilon_i | X_i) = 0$ and $\mathbb{V}(\epsilon_i | X_i) = \sigma^2$.

The parameters have specific interpretations:

- β_0 is the **intercept**: the expected value of Y when $X = 0$
- β_1 is the **slope**: the expected change in Y for a one-unit increase in X
- σ^2 is the **error variance**: how much individual observations vary around the line

This model makes a bold claim: the relationship between X and Y can be adequately captured by a straight line, with all deviations from this line being random noise with constant variance. The assumption $\mathbb{E}(\epsilon_i | X_i) = 0$ ensures the line goes through the “middle” of the data at each value of X , while $\mathbb{V}(\epsilon_i | X_i) = \sigma^2$ (homoscedasticity) means the scatter around the line is equally variable across all values of X .

i When is Linearity Reasonable?

More often than you might think! Many relationships are approximately linear, at least over the range of observed data. Even when the true relationship is nonlinear, linear regression often provides a useful first-order approximation – much like how we can approximate curves with tangent lines in calculus.

¹Note that the distribution of ϵ may depend on X , though its mean is always zero.

Consider height and weight, income and spending, or temperature and ice cream sales. While none of these relationships are perfectly linear across all possible values, they're often reasonably linear within the range we observe. And sometimes that's all we need for useful predictions and insights.

Once we've estimated the parameters – which we'll discuss next –, we can make predictions and assess our model:

Regression Terminology

Let $\hat{\beta}_0$ and $\hat{\beta}_1$ denote estimates of β_0 and β_1 . Then:

- The **fitted line** is: $\hat{r}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$
- The **predicted values** or **fitted values** are: $\hat{Y}_i = \hat{r}(X_i) = \hat{\beta}_0 + \hat{\beta}_1 X_i$
- The **residuals** are: $\hat{\epsilon}_i = Y_i - \hat{Y}_i = Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)$

The residuals are crucial – they represent what our model doesn't explain. Small residuals mean our line fits well; large residuals suggest either a poor fit or inherent variability in the relationship.

9.3.4 Estimating Parameters: The Method of Least Squares

Now comes the central question: given our data, how do we find the “best” line? What values should we choose for $\hat{\beta}_0$ and $\hat{\beta}_1$?

The most common answer is the **method of least squares**. The idea is to find the line that minimizes the **Residual Sum of Squares (RSS)**:

$$\text{RSS} = \sum_{i=1}^n \hat{\epsilon}_i^2 = \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2$$

The RSS measures how well the line fits the data – it's the sum of squared vertical distances from the points to the line. The least squares estimates are the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that make RSS as small as possible.

Multiple Perspectives

Intuitive

Imagine you're trying to draw a line through a cloud of points on a scatter plot. You want the line to be “close” to all the points simultaneously. But what does “close” mean?

For each point, we have an error: how far the point lies above the line (positive error) or below the line (negative error). We need to aggregate these errors into a single measure of fit. We have several options:

1. **Sum of errors:** Won't work – positive and negative errors cancel out. A terrible line far above half the points and far below the others could have zero total error!
2. **Sum of absolute errors:** This works (no cancellation), but absolute values are mathematically inconvenient – they're not differentiable at zero, making optimization harder.
3. **Sum of squared errors:** This is the winner! Squaring prevents cancellation, penalizes large errors more than small ones (outliers matter), and is mathematically convenient allowing closed-form solutions.

The least squares line is the one that minimizes this sum of squared errors. It's the line that best “threads through” the cloud of points in the squared-error sense.

Mathematical

To find the least squares estimates, we minimize RSS with respect to $\hat{\beta}_0$ and $\hat{\beta}_1$. Taking partial derivatives and setting them to zero:

$$\frac{\partial \text{RSS}}{\partial \hat{\beta}_0} = -2 \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i) = 0$$

$$\frac{\partial \text{RSS}}{\partial \hat{\beta}_1} = -2 \sum_{i=1}^n X_i (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i) = 0$$

These are called the **normal equations**. From the first equation:

$$\sum_{i=1}^n Y_i = n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^n X_i$$

Dividing by n gives: $\bar{Y}_n = \hat{\beta}_0 + \hat{\beta}_1 \bar{X}_n$, which shows the fitted line passes through (\bar{X}_n, \bar{Y}_n) .

From the second equation and some algebra (expanding the products and using the first equation), we get:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n X_i Y_i - n \bar{X}_n \bar{Y}_n}{\sum_{i=1}^n X_i^2 - n \bar{X}_n^2} = \frac{\sum_{i=1}^n (X_i - \bar{X}_n)(Y_i - \bar{Y}_n)}{\sum_{i=1}^n (X_i - \bar{X}_n)^2}$$

This is the sample covariance of X and Y divided by the sample variance of X .

Computational

Let's see least squares in action. We'll generate some synthetic data with a known linear relationship plus noise, then find the least squares line. The visualization will show two key perspectives:

1. **The fitted line with residuals:** How well the line fits the data and what the residuals look like
2. **The optimization landscape:** How RSS changes as we vary the slope, showing that our formula finds the minimum

```

import numpy as np
import matplotlib.pyplot as plt

# Generate some example data
np.random.seed(42)
n = 30
X = np.random.uniform(0, 10, n)
true_beta0, true_beta1 = 2, 1.5
Y = true_beta0 + true_beta1 * X + np.random.normal(0, 2, n)

# Calculate least squares estimates
X_mean = np.mean(X)
Y_mean = np.mean(Y)
beta1_hat = np.sum((X - X_mean) * (Y - Y_mean)) / np.sum((X - X_mean)**2)
beta0_hat = Y_mean - beta1_hat * X_mean

# Create visualization with stacked subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 8))

# Top panel: Show the residuals
ax1.scatter(X, Y, alpha=0.6, s=50)
X_plot = np.linspace(0, 10, 100)
Y_plot = beta0_hat + beta1_hat * X_plot
ax1.plot(X_plot, Y_plot, 'r-', linewidth=2, label=f'Fitted line: Y = {beta0_hat:.2f} + {beta1_hat:.2f}X')
for i in range(n):
    Y_pred = beta0_hat + beta1_hat * X[i]
    ax1.plot([X[i], X[i]], [Y[i], Y_pred], 'g--', alpha=0.5, linewidth=0.8)

ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_title('Least Squares Minimizes Squared Residuals')
ax1.legend()
ax1.grid(True, alpha=0.3)

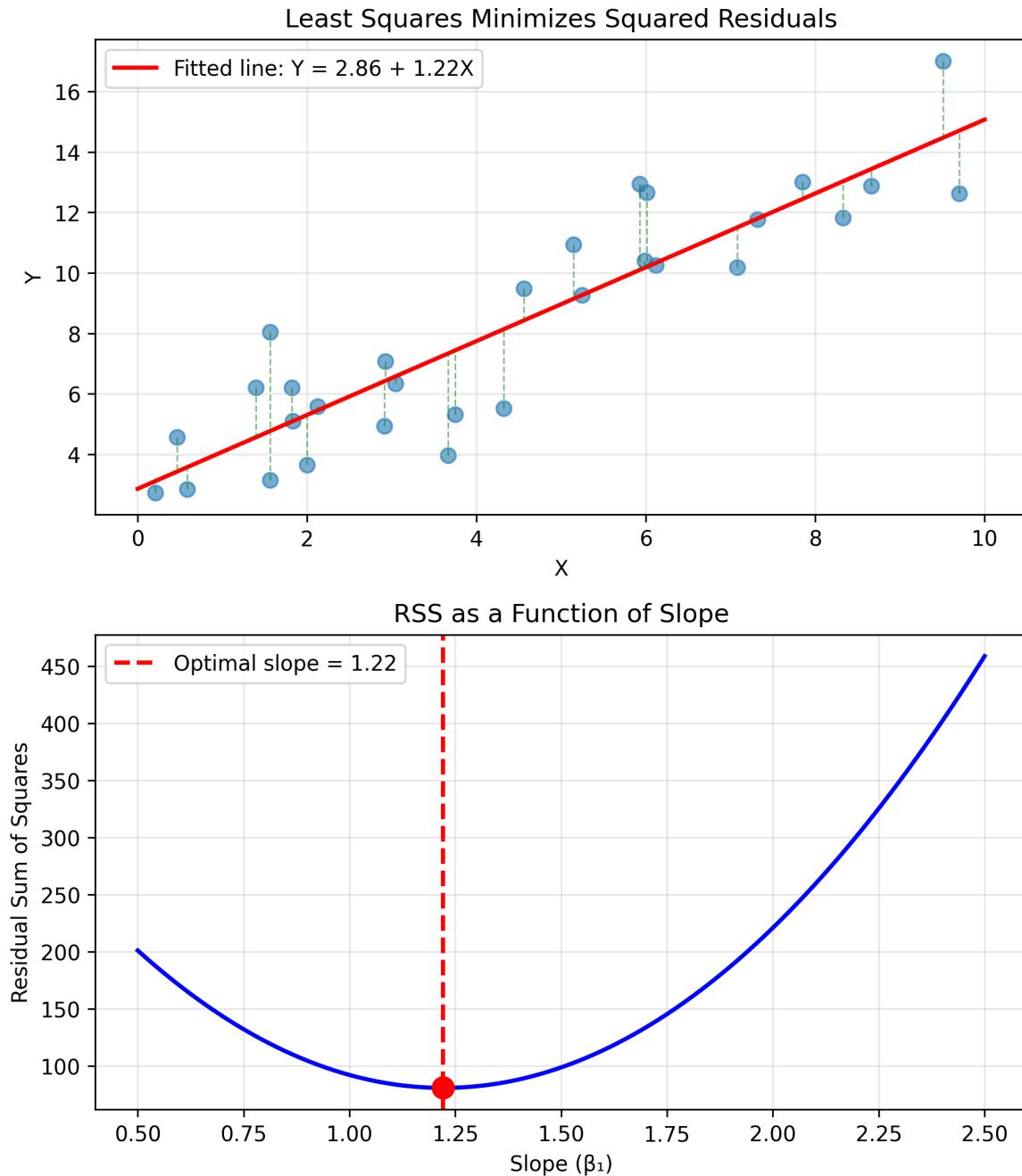
# Bottom panel: Show RSS as a function of slope
slopes = np.linspace(0.5, 2.5, 100)
rss_values = []
for slope in slopes:
    intercept = Y_mean - slope * X_mean # Best intercept given slope
    residuals = Y - (intercept + slope * X)
    rss = np.sum(residuals**2)
    rss_values.append(rss)

ax2.plot(slopes, rss_values, 'b-', linewidth=2)
ax2.axvline(beta1_hat, color='r', linestyle='--', linewidth=2, label=f'Optimal slope = {beta1_hat:.2f}')
ax2.scatter([beta1_hat], [min(rss_values)], color='r', s=100, zorder=5)
ax2.set_xlabel('Slope ( )')
ax2.set_ylabel('Residual Sum of Squares')
ax2.set_title('RSS as a Function of Slope')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Least squares estimates:  = {beta0_hat:.3f},  = {beta1_hat:.3f}")
print(f"Minimum RSS = {min(rss_values):.2f}")

```



Least squares estimates: $\hat{\beta}_0 = 2.858$, $\hat{\beta}_1 = 1.221$

Minimum RSS = 80.61

The top panel shows the fitted line and the residuals (green dashed lines). Notice how the line “threads through” the cloud of points, balancing the errors above and below. The residuals visualize what we’re minimizing – we want these vertical distances (squared) to be as small as possible in total.

The bottom panel reveals the optimization landscape. As we vary the slope β_1 (while adjusting β_0 optimally for each slope to maintain the constraint that the line passes through (\bar{X}, \bar{Y})), the RSS forms a parabola with a clear minimum. The red dashed line marks where our formula places us – exactly at the minimum! This confirms that the least squares formula genuinely finds the best fit in terms of minimizing squared errors.

The values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the RSS are:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (X_i - \bar{X}_n)(Y_i - \bar{Y}_n)}{\sum_{i=1}^n (X_i - \bar{X}_n)^2} \quad (\text{slope})$$

$$\hat{\beta}_0 = \bar{Y}_n - \hat{\beta}_1 \bar{X}_n \quad (\text{intercept})$$

where $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$ and $\bar{Y}_n = \frac{1}{n} \sum_{i=1}^n Y_i$ are the sample means.

These formulas have intuitive interpretations:

- The slope $\hat{\beta}_1$ is essentially the sample covariance between X and Y divided by the sample variance of X
- The intercept $\hat{\beta}_0$ is chosen so the fitted line passes through the point (\bar{X}_n, \bar{Y}_n) – the center of the data

Least squares is very convenient because it has a closed-form solution. We don't need iterative algorithms or numerical optimization² – just plug the data into our formulas and we get the optimal answer.

i Estimating Error Variance

We also need to estimate the error variance σ^2 . An unbiased estimator is:

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

Why divide by $n-2$ instead of n ? We've estimated two parameters (β_0 and β_1), which costs us two degrees of freedom. This adjustment ensures our variance estimate is unbiased.

9.3.5 Connection to Maximum Likelihood Estimation

So far, we've motivated least squares geometrically – it finds the line that minimizes squared distances. But there's a deeper connection to the likelihood principle we studied in previous chapters.

Adding the Normality Assumption

Suppose we strengthen our assumptions by specifying that the errors are normally distributed:

$$\epsilon_i | X_i \sim \mathcal{N}(0, \sigma^2)$$

This implies that:

$$Y_i | X_i \sim \mathcal{N}(\beta_0 + \beta_1 X_i, \sigma^2)$$

Each observation follows a normal distribution centered at the regression line, with constant variance σ^2 .

The Likelihood Function

Under these assumptions, we can write the likelihood function. The joint density of all observations is:

$$\prod_{i=1}^n f(X_i, Y_i) = \prod_{i=1}^n f_X(X_i) \times \prod_{i=1}^n f_{Y|X}(Y_i | X_i)$$

The first term doesn't involve our parameters β_0 and β_1 , so we focus on the second term – the **conditional likelihood**:

²Though for very large datasets, iterative methods like stochastic gradient descent may still be used for computational efficiency, even when closed-form solutions exist. Also, when we add regularization (Ridge, LASSO) or have constraints, we lose the closed-form solution and must use iterative methods.

$$\mathcal{L}(\beta_0, \beta_1, \sigma) = \prod_{i=1}^n f_{Y|X}(Y_i | X_i) \propto \sigma^{-n} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i)^2 \right\}$$

The conditional log-likelihood is:

$$\ell(\beta_0, \beta_1, \sigma) = -n \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i)^2$$

i The Key Insight

To maximize the log-likelihood with respect to β_0 and β_1 , we must minimize the sum:

$$\sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_i)^2$$

But this is exactly the RSS! Therefore, under the normality assumption, **the least squares estimators are also the maximum likelihood estimators**.

This is a profound connection: the simple geometric idea of minimizing squared distances coincides with the principled statistical approach of maximum likelihood, at least when errors are normal.

i What about estimating σ^2 ?

From the conditional log-likelihood above we can also derive the MLE for the error variance:

$$\hat{\sigma}_{\text{MLE}}^2 = \frac{1}{n} \sum_{i=1}^n \hat{\epsilon}_i^2$$

However, this estimator is biased – it systematically underestimates the true variance. In practice, we use the unbiased estimator:

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

The difference is small for large n , but the unbiased version provides more accurate confidence intervals and hypothesis tests, which is why it's standard in linear regression.

9.3.6 Properties of the Least Squares Estimators

Understanding the statistical properties of our estimates is crucial for inference. How accurate are they? How does accuracy improve with more data? Can we quantify our uncertainty?

Finite Sample Properties

Let $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1)^T$ denote the least squares estimators. Assume that:

1. $\mathbb{E}(\epsilon_i | X^n) = 0$ for all i
2. $\mathbb{V}(\epsilon_i | X^n) = \sigma^2$ for all i (homoscedasticity)
3. $\text{Cov}(\epsilon_i, \epsilon_j | X^n) = 0$ for $i \neq j$ (uncorrelated errors)

Then, conditional on $X^n = (X_1, \dots, X_n)$:

$$\mathbb{E}(\hat{\beta} | X^n) = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

$$\mathbb{V}(\hat{\beta} | X^n) = \frac{\sigma^2}{ns_X^2} \begin{pmatrix} \frac{1}{n} \sum_{i=1}^n X_i^2 & -\bar{X}_n \\ -\bar{X}_n & 1 \end{pmatrix}$$

where $s_X^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X}_n)^2$ is the sample variance of X .

This theorem tells us that the least squares estimators are **unbiased** – on average, they hit the true values. The variance formula allows us to compute standard errors:

$$\widehat{\text{se}}(\hat{\beta}_0) = \frac{\hat{\sigma}}{s_X \sqrt{n}} \sqrt{\frac{\sum_{i=1}^n X_i^2}{n}} \quad \text{and} \quad \widehat{\text{se}}(\hat{\beta}_1) = \frac{\hat{\sigma}}{s_X \sqrt{n}}$$

These standard errors quantify the uncertainty in our estimates. Notice that both decrease with \sqrt{n} – more data means more precision.

i Advanced: Derivation of Standard Errors

The variance formula comes from the fact that $\hat{\beta}_1$ is a linear combination of the Y_i 's:

$$\hat{\beta}_1 = \sum_{i=1}^n w_i Y_i$$

where $w_i = \frac{X_i - \bar{X}_n}{\sum_{j=1}^n (X_j - \bar{X}_n)^2}$.

Since the Y_i 's are independent with variance σ^2 :

$$\text{Var}(\hat{\beta}_1 | X^n) = \sigma^2 \sum_{i=1}^n w_i^2 = \frac{\sigma^2}{\sum_{i=1}^n (X_i - \bar{X}_n)^2} = \frac{\sigma^2}{ns_X^2}$$

The derivation for $\hat{\beta}_0$ is similar but more involved due to its dependence on both \bar{Y}_n and $\hat{\beta}_1$.

Asymptotic Properties and Inference

Under appropriate regularity conditions, as $n \rightarrow \infty$:

1. **Consistency:** $\hat{\beta}_0 \xrightarrow{P} \beta_0$ and $\hat{\beta}_1 \xrightarrow{P} \beta_1$
2. **Asymptotic Normality:**

$$\frac{\hat{\beta}_0 - \beta_0}{\widehat{\text{se}}(\hat{\beta}_0)} \rightsquigarrow \mathcal{N}(0, 1) \quad \text{and} \quad \frac{\hat{\beta}_1 - \beta_1}{\widehat{\text{se}}(\hat{\beta}_1)} \rightsquigarrow \mathcal{N}(0, 1)$$

3. **Confidence Intervals:** An approximate $(1 - \alpha)$ confidence interval for β_j is:

$$\hat{\beta}_j \pm z_{\alpha/2} \cdot \widehat{\text{se}}(\hat{\beta}_j)$$

4. **Hypothesis Testing:** The Wald test for $H_0 : \beta_1 = 0$ vs. $H_1 : \beta_1 \neq 0$ is: reject H_0 if $|W| > z_{\alpha/2}$ where $W = \hat{\beta}_1 / \widehat{\text{se}}(\hat{\beta}_1)$

i Finite Sample Refinement

For finite n , a more accurate test uses the Student's t -distribution with $n - 2$ degrees of freedom rather than the normal distribution. This accounts for the additional uncertainty from estimating σ^2 . Most statistical software uses the t -distribution by default for regression inference.

9.3.7 Simple Linear Regression in Practice

Let's see how these concepts work with real data. We'll use the classic Framingham Heart Study dataset to explore the relationship between weight and blood pressure, building our understanding step by step.

Example: The Framingham Heart Study

The Framingham Heart Study is a long-term cardiovascular study that began in 1948 in Framingham, Massachusetts. This landmark study has provided crucial insights into cardiovascular disease risk factors. We'll examine the relationship between **relative weight** (FRW - a normalized weight measure where 100 represents median weight for height) and **systolic blood pressure** (SBP - the pressure when the heart beats).

9.3.7.1 Step 1: Loading and Exploring the Data

First, let's load the data and understand what we're working with:

Multiple Perspectives

Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import statsmodels.api as sm
from scipy import stats

# Load the Framingham data
fram = pd.read_csv('../data/fram.txt', sep='\t', index_col=0)

# Display first few rows to understand the structure
print("First 6 rows of the Framingham data:")
print(fram.head(6))
print(f"\nDataset shape: {fram.shape}")
print(f"Columns: {list(fram.columns)}")

First 6 rows of the Framingham data:
   SEX  AGE  FRW  SBP  SBP10  DBP  CHOL  CIG  CHD  YRS_CHD  DEATH \
ID
4988  female    57   135   186     NaN   120    150    0    1      pre     7
3001  female    60   123   165     NaN   100    167    25    0      16    10
5079  female    54   115   140     NaN    90    213     5    0      8     8
5162  female    52   102   170     NaN   104    280    15    0     10     7
4672  female    45    99   185     NaN   105    326    20    0      8    10
5822  female    51    93   142     NaN    90    234    35    0      4     8

   YRS_DTH      CAUSE
ID
4988      11  unknown
3001      17  unknown
5079      13  unknown
5162      11  unknown
```

```

4672      17  unknown
5822      13  unknown

Dataset shape: (1394, 13)
Columns: ['SEX', 'AGE', 'FRW', 'SBP', 'SBP10', 'DBP', 'CHOL', 'CIG', 'CHD', 'YRS_CHD', 'DEATH', 'YRS_]

# Focus on our variables of interest
print("\nSummary statistics for key variables:")
print(fram[['FRW', 'SBP']].describe())
Summary statistics for key variables:
          FRW        SBP
count  1394.000000  1394.000000
mean   105.365136  148.086083
std    17.752489  28.022062
min    52.000000  90.000000
25%    94.000000  130.000000
50%   103.000000  142.000000
75%   114.000000  160.000000
max   222.000000  300.000000

```

R

```

# Load the data
fram <- read.csv('../data/fram.txt', sep='\t', row.names = 1)

# Display first few rows to understand the structure
cat("First 6 rows of the Framingham data:\n")
head(fram)

cat("\nDataset dimensions:", dim(fram), "\n")
cat("Column names:", names(fram), "\n")

# Summary statistics for key variables
summary(fram[c("FRW", "SBP")])

```

i Understanding the Variables

- The dataset contains 1,394 observations with 13 variables including demographics, behaviors, and health outcomes.
- FRW (Framingham Relative Weight)**: A normalized measure where 100 represents the median weight for a given height. Values above 100 indicate above-median weight.
- SBP (Systolic Blood Pressure)**: Measured in mmHg, normal range is typically below 120. Values 140 indicate hypertension.

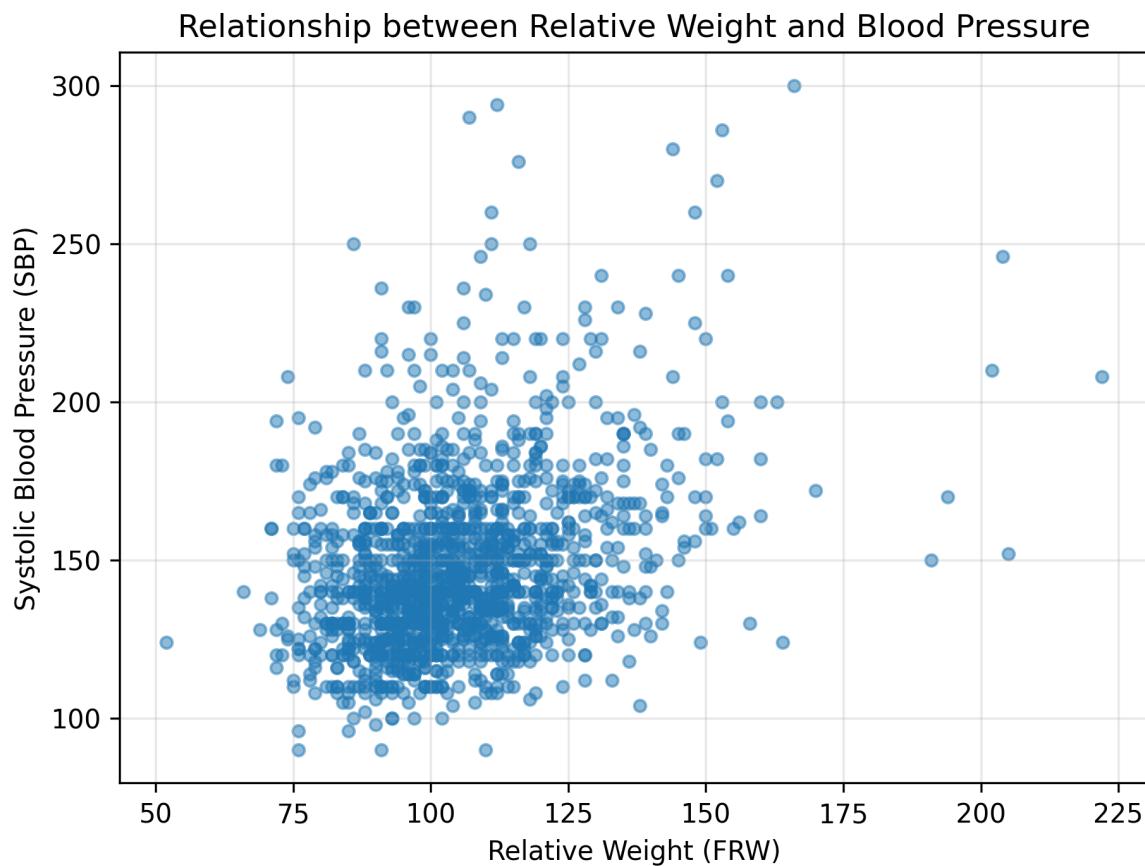
9.3.7.2 Step 2: Initial Visualization

Before fitting any model, let's visualize the relationship between weight and blood pressure:

Multiple Perspectives

Python

```
# Create scatter plot to explore the relationship
plt.figure(figsize=(7, 5))
plt.scatter(fram['FRW'], fram['SBP'], alpha=0.5, s=20)
plt.xlabel('Relative Weight (FRW)')
plt.ylabel('Systolic Blood Pressure (SBP)')
plt.title('Relationship between Relative Weight and Blood Pressure')
plt.grid(True, alpha=0.3)
plt.show()
```



R

```
# Create scatter plot to explore the relationship
plot(fram$FRW, fram$SBP,
      xlab = "Relative Weight (FRW)",
      ylab = "Systolic Blood Pressure (SBP)",
      main = "Relationship between Relative Weight and Blood Pressure",
      pch = 16, col = rgb(0, 0, 0, 0.3))
grid()
```

i What the Visualization Tells Us

Looking at this scatter plot, we can observe:

- There appears to be a **positive relationship**: As relative weight increases, blood pressure tends to increase.
- There's **substantial variation**: The wide spread of points suggests that weight alone won't perfectly predict blood pressure - other factors must also be important.
- The pattern looks **compatible with the linearity hypothesis**, in the sense that there's no obvious curve or nonlinear pattern that would suggest a straight line is inappropriate.

This visualization motivates our next step: fitting a linear model to quantify this relationship.

9.3.7.3 Step 3: Fitting the Linear Regression Model

Now let's fit the simple linear regression model: $SBP = \beta_0 + \beta_1 \cdot FRW + \epsilon$.

Linear regression with a quadratic loss is also called "ordinary least squares" (OLS), a term which can be found in statistical software like in Python's `statsmodel` package.

Multiple Perspectives

Python

```
# Fit a linear regression model using statsmodels
model = smf.ols('SBP ~ FRW', data=fram)
results = model.fit()

# Show the fitted equation
print(f"Fitted equation: SBP = {results.params['Intercept']):.2f} + {results.params['FRW']):.3f} * FRW")
Fitted equation: SBP = 92.87 + 0.524 * FRW
```

R

```
# Fit the linear regression model
fit <- lm(SBP ~ FRW, data = fram)

# Show the fitted equation
cat(sprintf("Fitted equation: SBP = %.2f + %.3f * FRW\n",
            coef(fit)[1], coef(fit)[2]))
```

We've now fitted our linear regression model. The fitted equation shows the estimated relationship between relative weight and blood pressure. In the following steps, we'll use this model to make predictions, visualize the fit, check our assumptions, and interpret the statistical significance of our findings.

9.3.7.4 Step 4: Making Predictions

Let's use our model to make predictions for specific weight values:

Multiple Perspectives

Python

```

# Make predictions for specific FRW values
test_weights = pd.DataFrame({'FRW': [80, 100, 120]})
predictions = results.predict(test_weights)

print("Predictions for new data points:")
print("*"*40)
for frw, sbp in zip(test_weights['FRW'], predictions):
    print(f"FRW = {frw:.3d} → Predicted SBP = {sbp:.1f}")

# Manual calculation to show the mechanics
print("\nManual calculation (verifying our understanding):")
beta0 = results.params['Intercept']
beta1 = results.params['FRW']
for frw in [80, 100, 120]:
    manual_pred = beta0 + beta1 * frw
    print(f"FRW = {frw:.3d} → {beta0:.3f} + {beta1:.3f} × {frw} = {manual_pred:.1f}")

Predictions for new data points:
=====
FRW = 80 → Predicted SBP = 134.8
FRW = 100 → Predicted SBP = 145.3
FRW = 120 → Predicted SBP = 155.8

Manual calculation (verifying our understanding):
FRW = 80 → 92.866 + 0.524 × 80 = 134.8
FRW = 100 → 92.866 + 0.524 × 100 = 145.3
FRW = 120 → 92.866 + 0.524 × 120 = 155.8

```

R

```

# Make predictions for specific FRW values
test_weights <- data.frame(FRW = c(80, 100, 120))
predictions <- predict(fit, newdata = test_weights)

cat("Predictions for new data points:\n")
cat(rep("=", 40), collapse="", "\n")
for(i in 1:nrow(test_weights)) {
    cat(sprintf("FRW = %3d → Predicted SBP = %.1f\n",
               test_weights$FRW[i], predictions[i]))
}

# Manual calculation
beta0 <- coef(fit)[1]
beta1 <- coef(fit)[2]
cat("\nManual calculation (verifying our understanding):\n")
for(frw in c(80, 100, 120)) {
    manual_pred <- beta0 + beta1 * frw
    cat(sprintf("FRW = %3d → %.3f + %.3f × %d = %.1f\n",
               frw, beta0, beta1, frw, manual_pred))
}

```

9.3.7.5 Step 5: Visualizing the Fitted Model

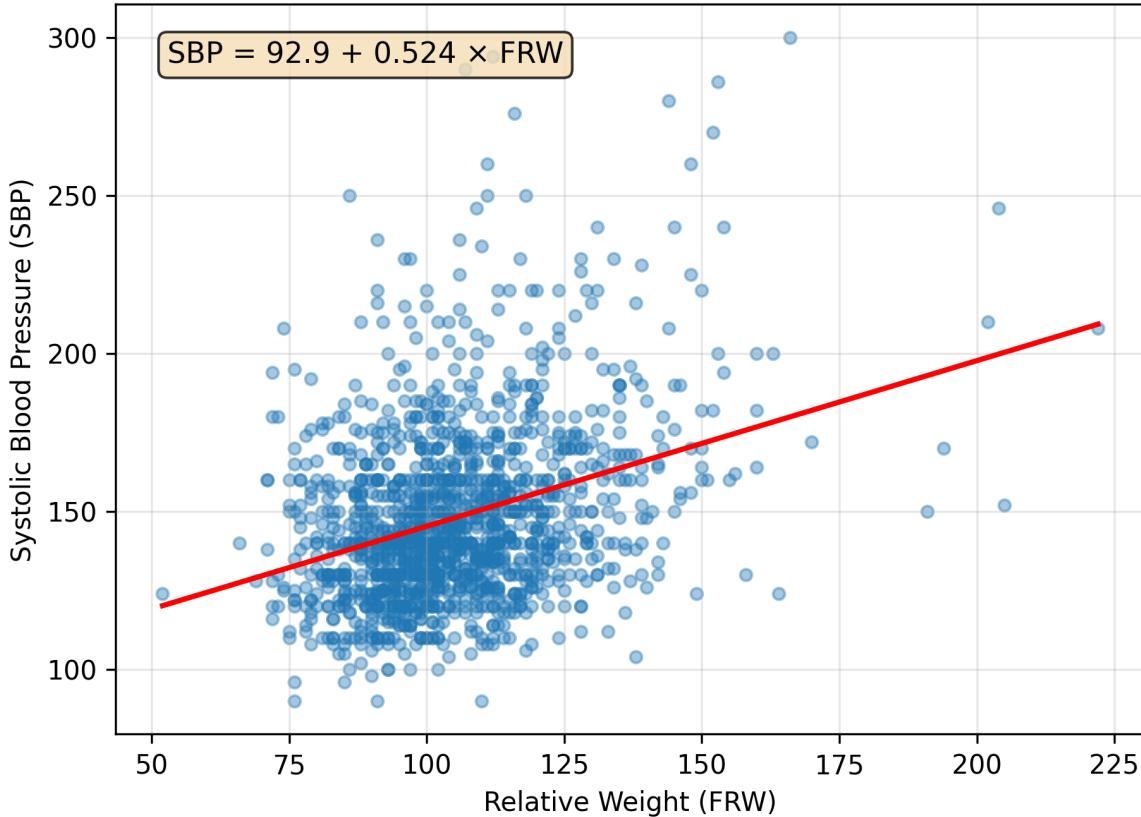
Let's plot the fitted model:

Multiple Perspectives

Python

```
# Basic fitted line visualization
plt.figure(figsize=(7, 5))
plt.scatter(fram['FRW'], fram['SBP'], alpha=0.4, s=20)
x_range = np.linspace(fram['FRW'].min(), fram['FRW'].max(), 100)
y_pred = results.params['Intercept'] + results.params['FRW'] * x_range
plt.plot(x_range, y_pred, 'r-', linewidth=2, label='Fitted line')
plt.xlabel('Relative Weight (FRW)')
plt.ylabel('Systolic Blood Pressure (SBP)')
plt.title('Fitted Regression Line')
# Add equation to the plot
equation = f'SBP = {results.params["Intercept"]:.1f} + {results.params["FRW"]:.3f} × FRW'
plt.text(0.05, 0.95, equation, transform=plt.gca().transAxes, fontsize=11,
         verticalalignment='top', bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
plt.grid(True, alpha=0.3)
plt.show()
```

Fitted Regression Line



R

```
# Basic fitted line visualization
plot(fram$FRW, fram$SBP,
      xlab = "Relative Weight (FRW)",
      ylab = "Systolic Blood Pressure (SBP)",
      main = "Fitted Regression Line",
      pch = 16, col = rgb(0, 0, 0, 0.4))
abline(fit, col = "red", lwd = 2)
# Add equation to the plot
equation <- paste("SBP = ", round(coef(fit)[1], 1), "+",
                  round(coef(fit)[2], 3), "x FRW")
legend("topleft", legend = equation,
       bty = "n", cex = 1.1,
       text.col = "black", bg = rgb(1, 0.96, 0.8, 0.8))
grid()
```

The plot above shows our fitted regression line. The equation in the box gives us the specific relationship: for each unit increase in relative weight (FRW), systolic blood pressure increases by approximately half a mmHg on average (see the exact coefficient in the equation above).

The plot below shows the geometric interpretation of intercept (β_0) and slope (β_1) parameters:

```
# Geometric interpretation of intercept and slope
plt.figure(figsize=(7, 6))
plt.scatter(fram['FRW'], fram['SBP'], alpha=0.4, s=20)

# Extend x range to show intercept
x_extended = np.linspace(0, fram['FRW'].max(), 100)
y_extended = results.params['Intercept'] + results.params['FRW'] * x_extended
plt.plot(x_extended, y_extended, 'r-', linewidth=2)

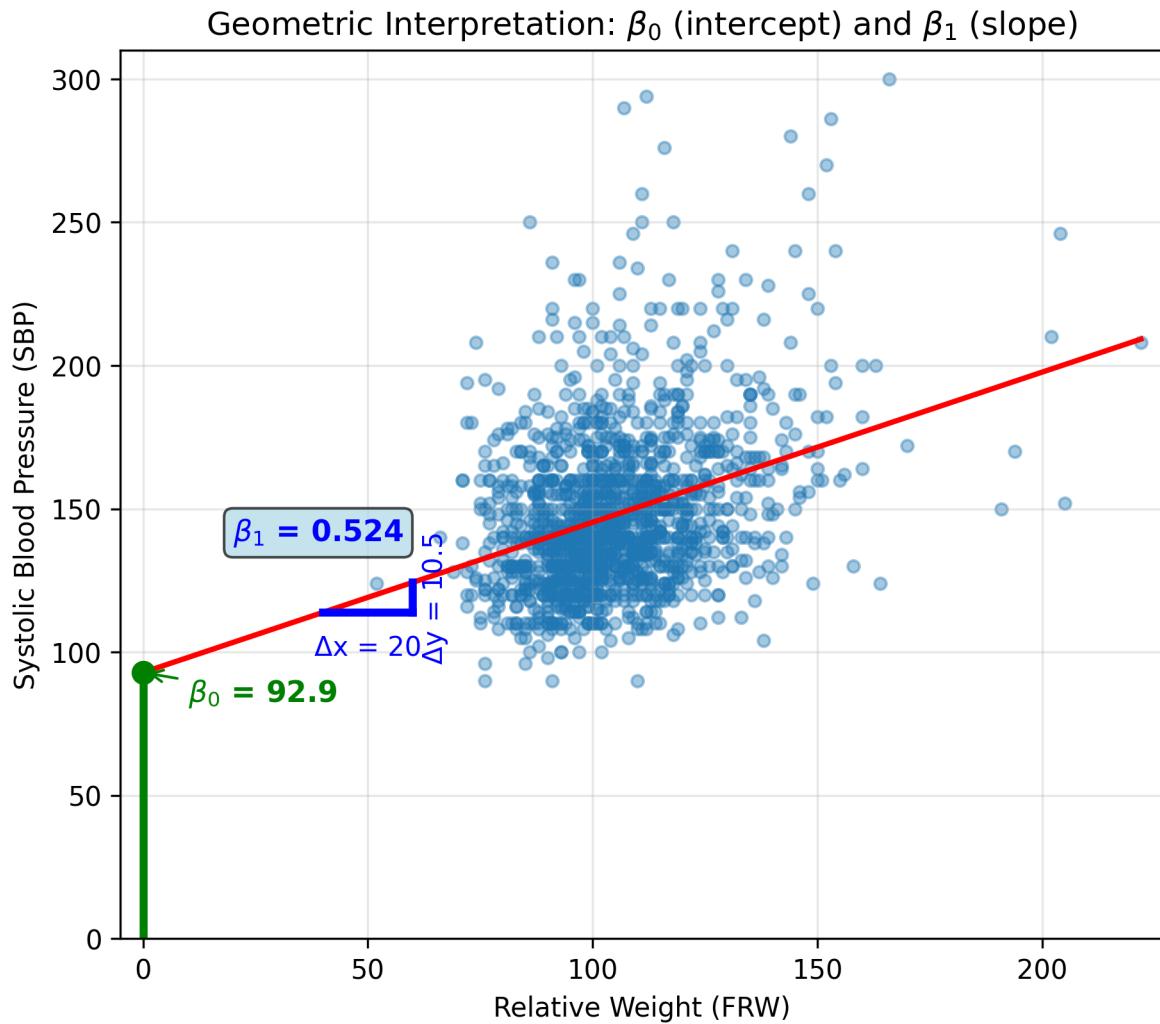
# Show intercept ( ) at x=0
plt.plot([0, 0], [0, results.params['Intercept']], 'green', linewidth=3)
plt.plot(0, results.params['Intercept'], 'go', markersize=8)
plt.annotate(r'$\beta_0$' + f' = {results.params["Intercept"]:.1f}',
            xy=(0, results.params['Intercept']), xytext=(10, results.params['Intercept'] - 10),
            fontsize=11, color='green', fontweight='bold',
            arrowprops=dict(arrowstyle='->', color='green', lw=1))

# Show slope ( ) interpretation
x_demo = [40, 60]
y_demo = [results.params['Intercept'] + results.params['FRW'] * x for x in x_demo]
plt.plot(x_demo, [y_demo[0], y_demo[0]], 'b-', linewidth=3)
plt.plot([x_demo[1], x_demo[1]], y_demo, 'b-', linewidth=3)
plt.annotate('Δx = 20', xy=(50, y_demo[0] - 15), fontsize=10, color='blue', ha='center')
plt.annotate(f'Δy = {20 * results.params["FRW"]:.1f}', xy=(x_demo[1] + 2, np.mean(y_demo)),
            fontsize=10, color='blue', rotation=90, va='center')
plt.text(20, y_demo[0] + 25, r'$\beta_1$' + f' = {results.params["FRW"]:.3f}',
        fontsize=11, color='blue', fontweight='bold',
        bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.7))
```

```

plt.xlim(-5, fram['FRW'].max() + 5)
plt.ylim(0, fram['SBP'].max() + 10)
plt.xlabel('Relative Weight (FRW)')
plt.ylabel('Systolic Blood Pressure (SBP)')
plt.title(r'Geometric Interpretation: $\beta_0$ (intercept) and $\beta_1$ (slope)')
plt.grid(True, alpha=0.3)
plt.show()

```



9.3.7.6 Step 6: Model Diagnostics

Before interpreting our regression results, let's check if our model assumptions are satisfied using diagnostic plots:

Multiple Perspectives

Python

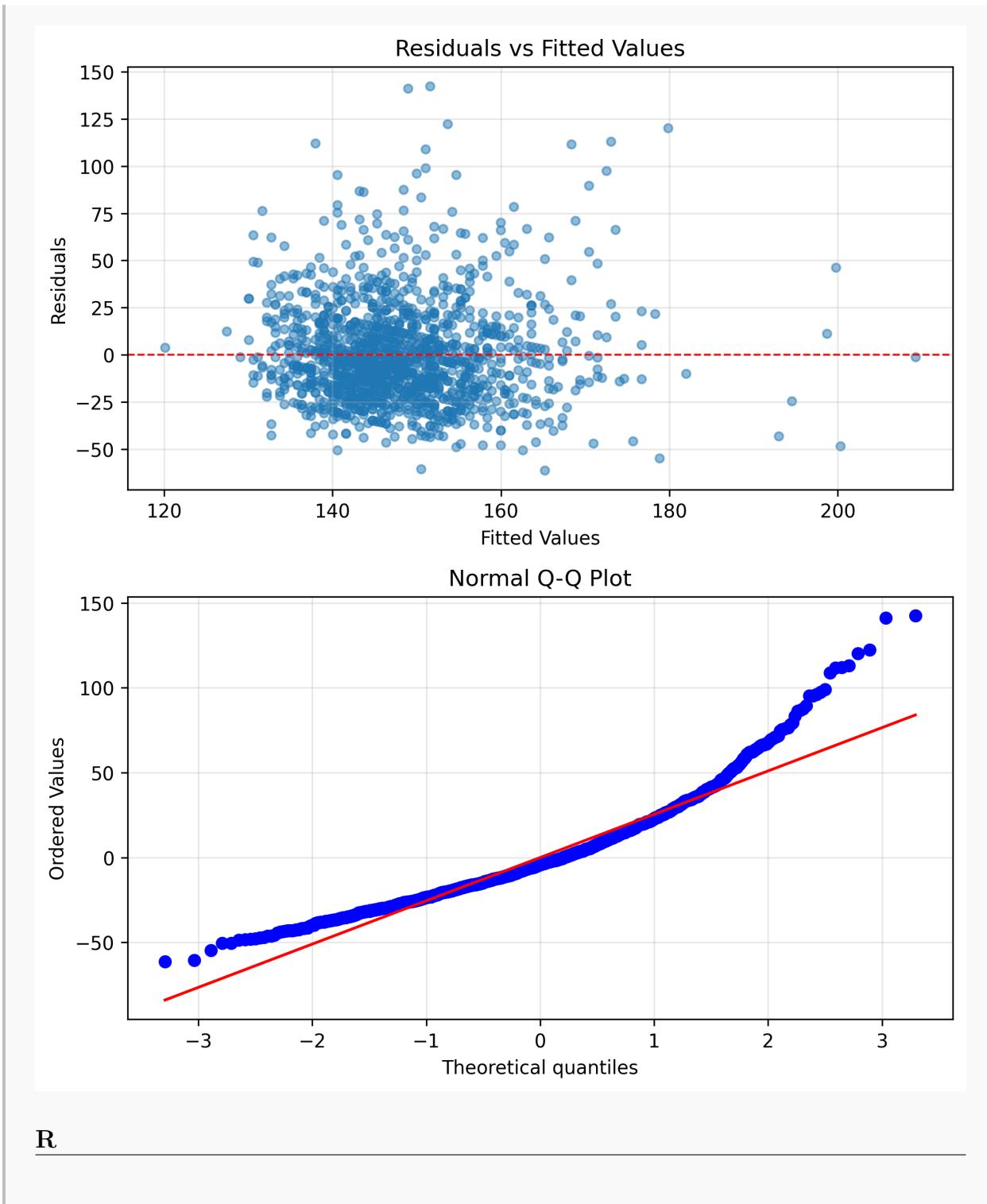
```
# Create diagnostic plots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 8))

# Get residuals and fitted values
residuals = results.resid
fitted = results.fittedvalues

# Plot 1: Residuals vs Fitted Values
ax1.scatter(fitted, residuals, alpha=0.5, s=20)
ax1.axhline(y=0, color='r', linestyle='--', linewidth=1)
ax1.set_xlabel('Fitted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Residuals vs Fitted Values')
ax1.grid(True, alpha=0.3)

# Plot 2: Q-Q plot for normality check
stats.probplot(residuals, dist="norm", plot=ax2)
ax2.set_title('Normal Q-Q Plot')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



```
# Create diagnostic plots
par(mfrow = c(2, 1))

# Plot 1: Residuals vs Fitted
plot(fit, which = 1)

# Plot 2: Q-Q plot
plot(fit, which = 2)

# Reset plot layout
par(mfrow = c(1, 1))
```

What the Diagnostic Plots Tell Us

Residuals vs Fitted: The residuals are approximately centered around zero with reasonably random scatter, though we can see some evidence of skew (asymmetry) in the distribution.

Q-Q Plot (Quantile-Quantile Plot): This plot compares the distribution of our residuals to a normal distribution. If residuals were perfectly normal, all points would fall exactly on the diagonal line. Here we see the points roughly follow the line but with some deviation at the extremes (the tails), confirming that the residuals are not perfectly normally distributed.

Important: These minor violations of assumptions are common with real data and not necessarily dealbreakers. Linear regression is quite robust to modest departures from normality, especially with large samples like ours ($n=1,394$). However, we should keep these limitations in mind when interpreting results and consider transformations or robust methods if violations become severe.

9.3.7.7 Step 7: Model Interpretation

Now let's examine and interpret the full regression output:

Multiple Perspectives

Python

```
# Display the full regression summary
print(results.summary())
                OLS Regression Results
=====
Dep. Variable:          SBP    R-squared:     0.110
Model:                 OLS    Adj. R-squared:  0.110
Method:                Least Squares   F-statistic:   172.5
Date:      Sun, 07 Sep 2025   Prob (F-statistic): 3.18e-37
Time:          17:29:20    Log-Likelihood: -6542.3
No. Observations:      1394    AIC:         1.309e+04
Df Residuals:          1392    BIC:         1.310e+04
Df Model:                  1
Covariance Type:    nonrobust
=====
            coef      std err          t      P>|t|      [0.025      0.975]
-----
```

Intercept	92.8658	4.264	21.778	0.000	84.501	101.231
FRW	0.5241	0.040	13.132	0.000	0.446	0.602
<hr/>						
Omnibus:		338.464	Durbin-Watson:		1.756	
Prob(Omnibus):		0.000	Jarque-Bera (JB):		883.998	
Skew:		1.271	Prob(JB):		1.10e-192	
Kurtosis:		5.959	Cond. No.		643.	
<hr/>						

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Note: Python's statsmodels output is more verbose than R's. Focus on these key sections:

- **Coefficients table** (middle): Shows estimates, standard errors, t-statistics, and p-values
- **R-squared** (top-right): Proportion of variance explained
- **F-statistic** (top-right): Tests overall model significance

R

```
# Display the regression summary
summary(fit)
```

i Interpreting the Regression Output

Key Statistical Tests:

1. The reported p-values for individual coefficients use the **Wald Test**:
 - $H_0 : \beta_j = 0$ (coefficient equals zero, no effect)
 - Test statistic: $t = \hat{\beta}_j / \text{SE}(\hat{\beta}_j)$
 - Our result: FRW p-value < 0.001 → reject H_0 → weight significantly affects blood pressure
2. **F-Test** is used for overall model significance
 - $H_0 : \beta_1 = \beta_2 = \dots = \beta_k = 0$ (no predictors have any effect)
 - Tests if **any** coefficient is non-zero
 - Our result: F = 172.5, p < 0.001 → reject H_0 → model is useful
3. **R-Squared** (R^2 , coefficient of determination)
 - Proportion of variance in Y explained by the model (typically between 0 and 1)
 - Measures accuracy on training data
 - In our example: model explains roughly 11% of blood pressure variation (see exact value in output above)
 - Low R^2 common when many unmeasured factors affect outcome, which happens for many real-world data

Reading the Coefficients Table:

- **Intercept** (β_0): Expected SBP when FRW = 0 (extrapolation - not meaningful here)
- **FRW coefficient** (β_1): Each unit increase in relative weight increases SBP by approximately 0.5 mmHg on average (see exact value in output)
- **Standard errors**: Quantify uncertainty in estimates
- **95% CI for β_1** : Check the confidence interval in the output - it quantifies our uncertainty about the true slope

Model Quality Assessment:

- **Residual Standard Error**: Typical prediction error (see output for exact value)
- **Statistical vs. Practical Significance**: While statistically significant ($p < 0.001$), the effect size is modest

- **Predictive Power:** The relatively low R^2 indicates that weight alone is not a strong predictor of blood pressure

Key Takeaway: The regression confirms a statistically significant positive relationship between weight and blood pressure. However, the modest R^2 reminds us that blood pressure is multifactorial – diet, exercise, genetics, stress, and many other factors play important roles.

9.3.7.8 Summary: What We've Learned

Through this comprehensive example, we've followed a complete regression workflow:

1. **Data Exploration:** Understood our variables and their context
2. **Initial Visualization:** Examined the relationship visually before modeling
3. **Model Fitting:** Applied least squares to find the best-fitting line
4. **Making Predictions:** Used the model to predict new values
5. **Visualizing the Fit:** Showed the fitted line
6. **Model Diagnostics:** Checked assumptions through residual plots
7. **Model Interpretation:** Understood the statistical tests and what they tell us

The Framingham data reveals an important finding: weight has a statistically significant effect on blood pressure ($p < 0.001$), with each unit increase in relative weight associated with approximately a 0.5 mmHg increase in systolic blood pressure (see the exact coefficient in the regression output above).

The modest R^2 (around 11% of variance explained) doesn't diminish the medical importance of this relationship. Rather, it reminds us that:

- Blood pressure is multifactorial - weight is one important factor among many (age, diet, exercise, genetics, stress)
- Even when individual predictors explain modest variance, they can still be clinically meaningful
- Simple linear regression effectively quantifies real-world relationships and their uncertainty

9.4 Multiple Linear Regression

9.4.1 Extending the Model to Multiple Predictors

Real-world phenomena rarely depend on just one predictor. A person's blood pressure isn't determined solely by their weight – age, diet, exercise, genetics, and countless other factors play roles. Multiple linear regression extends our framework to handle multiple predictors simultaneously.

9.4.1.1 The Model

With k predictors (covariates), the multiple linear regression model becomes:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dots + \beta_k X_{ik} + \epsilon_i, \quad i = 1, \dots, n$$

where:

- Y_i is the response for observation i
- X_{ij} is the value of the j -th covariate for observation i
- β_0 is the intercept
- β_j is the coefficient for the j -th covariate (for $j = 1, \dots, k$)
- ϵ_i is the error term with $\mathbb{E}(\epsilon_i | X_i) = 0$ and $\mathbb{V}(\epsilon_i | X_i) = \sigma^2$

! Convention: Incorporating the Intercept

The intercept β_0 is often incorporated directly into the covariate notation by defining $X_{i0} = 1$ for all observations $i = 1, \dots, n$. This allows us to write the model more compactly as:

$$Y_i = \sum_{j=0}^k \beta_j X_{ij} + \epsilon_i$$

This convention simplifies matrix notation and many derivations. When you see design matrices or covariate vectors, check whether the intercept is handled explicitly (with a column of ones) or implicitly (assumed but not shown).

i Indexing Confusion: Starting from 0 vs 1

You'll encounter two indexing conventions in the literature:

Convention 1 (0-indexed, used here in the lecture notes):

- $X_{i0} = 1$ for the intercept
- $X_{i1}, X_{i2}, \dots, X_{ik}$ for the k actual covariates
- Design matrix \mathbf{X} is $n \times (k + 1)$
- Model: $Y_i = \beta_0 + \beta_1 X_{i1} + \dots + \beta_k X_{ik} + \epsilon_i$

Convention 2 (1-indexed, common in some texts):

- $X_{i1} = 1$ for the intercept
- $X_{i2}, X_{i3}, \dots, X_{i,k+1}$ for the k actual covariates
- Design matrix \mathbf{X} is still $n \times (k + 1)$
- Model: $Y_i = \beta_1 + \beta_2 X_{i2} + \dots + \beta_{k+1} X_{i,k+1} + \epsilon_i$

Both are correct! The key is consistency within a given analysis. Software typically handles this transparently – R's `lm()` and Python's `statsmodels` automatically add the intercept column regardless of your indexing preference.

Multiple Linear Regression in Matrix Form

The model is more elegantly expressed using matrix notation:

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon$$

where:

- \mathbf{Y} is an $n \times 1$ vector of responses
- \mathbf{X} is an $n \times (k + 1)$ design matrix (often written as $n \times k$ when the intercept is implicit)
- β is a $(k + 1) \times 1$ vector of coefficients (or $k \times 1$ when intercept is implicit)
- ϵ is an $n \times 1$ vector of errors

Explicitly (showing the column of 1s separately from the actual covariates):

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1k} \\ 1 & X_{21} & X_{22} & \cdots & X_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{nk} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

Here:

- The first column of 1s corresponds to what we defined as $X_{i0} = 1$ for the intercept
- $X_{i1}, X_{i2}, \dots, X_{ik}$ are the values of the k actual covariates for observation i
- The n rows correspond to different observations
- The $k + 1$ columns correspond to the intercept and k covariates

9.4.1.2 Least Squares in Matrix Form

The least squares criterion remains the same – minimize RSS – but the solution is now expressed in matrix form:

Assuming $\mathbf{X}^T \mathbf{X}$ is invertible, the least squares estimate is:

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

with variance-covariance matrix:

$$\mathbb{V}(\hat{\beta} | \mathbf{X}) = \sigma^2 (\mathbf{X}^T \mathbf{X})^{-1}$$

Key Results from the Least Squares Solution:

1. The estimated regression function is:

$$\hat{r}(x) = \hat{\beta}_0 + \sum_{j=1}^k \hat{\beta}_j x_j$$

2. An unbiased estimate of the error variance σ^2 is:

$$\hat{\sigma}^2 = \frac{1}{n - k - 1} \sum_{i=1}^n \hat{\epsilon}_i^2 = \frac{1}{n - k - 1} \|\mathbf{Y} - \mathbf{X}\hat{\beta}\|^2$$

We divide by $n - k - 1$ because we've estimated $k + 1$ parameters (including the intercept).

3. Confidence intervals for individual coefficients:

$$\hat{\beta}_j \pm z_{\alpha/2} \cdot \widehat{\text{se}}(\hat{\beta}_j)$$

where $\widehat{\text{se}}^2(\hat{\beta}_j)$ is the j -th diagonal element of $\hat{\sigma}^2 (\mathbf{X}^T \mathbf{X})^{-1}$.

The Meaning of a Coefficient

In multiple regression, the interpretation of coefficients becomes more subtle. The coefficient β_j represents the expected change in Y for a one-unit change in X_j , **holding all other covariates constant**. This is the crucial concept of **statistical control** or **adjustment**. We're estimating the partial effect of each predictor, after accounting for the linear effects of all other predictors in the model.

This interpretation assumes:

1. The other variables can actually be held constant (may not be realistic)
2. The relationship is truly linear
3. No important interactions exist between predictors

9.4.2 Multiple Regression in Practice

We'll now add two biologically relevant predictors to our weight-only model:

- **SEX:** Categorical variable (“female”/“male” in the data)
 - R and Python automatically encode with female = 0 (reference), male = 1
 - Output shows as “SEX[T.male]” (Python) or “SEXmale” (R)
 - Coefficient interpretation: difference in mean SBP for males vs females
- **CHOL:** Total serum cholesterol in mg/dL (typical range: 150-250 mg/dL)
 - A known cardiovascular risk factor

- Coefficient interpretation: change in SBP per 1 mg/dL increase in cholesterol

We'll build three models:

1. **Model 1:** $SBP \sim FRW$ (baseline simple regression)
2. **Model 2:** $SBP \sim FRW + SEX + CHOL$ (multiple regression)
 - Adds SEX and CHOL predictors
3. **Model 3:** $SBP \sim FRW + SEX + CHOL + FRW:SEX$ (with interaction/cross term)
 - $FRW:SEX$ is an **interaction term** - that is the product $FRW \times SEX$ (weight $\times 0/1$ for female/male)
 - This tests: "Does weight affect blood pressure differently for males vs females?"
 - For females ($SEX = 0$): Effect of weight = β_{FRW}
 - For males ($SEX = 1$): Effect of weight = $\beta_{FRW} + \beta_{FRW:SEX}$
 - If $\beta_{FRW:SEX} > 0$: weight increases BP more for males than females

Example: From Simple to Multiple Regression

Let's build on our simple regression model by adding sex and cholesterol as predictors:

Multiple Perspectives

Python

```
# Load the same Framingham data
fram = pd.read_csv('../data/fram.txt', sep='\t', index_col=0)

# Model 1: Simple regression (for comparison)
model1 = smf.ols('SBP ~ FRW', data=fram).fit()

# Model 2: Multiple regression
model2 = smf.ols('SBP ~ FRW + SEX + CHOL', data=fram).fit()

# Model 3: With interaction (does weight affect BP differently for males/females?)
model3 = smf.ols('SBP ~ FRW + SEX + CHOL + FRW:SEX', data=fram).fit()

print("Simple Regression (Model 1):")
print(f"  FRW coefficient: {model1.params['FRW']:.3f} (SE={model1.bse['FRW']:.3f})")
print(f"  R-squared: {model1.rsquared:.3f}")

print("\nMultiple Regression (Model 2):")
print(f"  FRW coefficient: {model2.params['FRW']:.3f} (SE={model2.bse['FRW']:.3f})")
print(f"  SEX[T.male] coefficient: {model2.params['SEX[T.male]']:.3f}")
print(f"  CHOL coefficient: {model2.params['CHOL']:.3f}")
print(f"  R-squared: {model2.rsquared:.3f}")

print("\nWith Interaction (Model 3):")
if 'FRW:SEX[T.male]' in model3.params:
    interaction_coef = model3.params['FRW:SEX[T.male]']
    interaction_pval = model3.pvalues['FRW:SEX[T.male]']
    print(f"  FRW:SEX interaction: {interaction_coef:.3f} (p={interaction_pval:.3f})")
    if interaction_pval > 0.05:
        print(f"    Interpretation: No significant interaction - weight affects BP similarly for both sexes")
    else:
        print(f"    Interpretation: Weight has {'stronger' if interaction_coef > 0 else 'weaker'} effect on BP")
Simple Regression (Model 1):
  FRW coefficient: 0.524 (SE=0.040)
  R-squared: 0.110

Multiple Regression (Model 2):
  FRW coefficient: 0.499 (SE=0.040)
  SEX[T.male] coefficient: -4.066
  CHOL coefficient: 0.053
  R-squared: 0.125

With Interaction (Model 3):
  FRW:SEX interaction: -0.004 (p=0.967)
  Interpretation: No significant interaction - weight affects BP similarly for both sexes
```

R

```
# Load data
fram <- read.csv('../data/fram.txt', sep='\t', row.names = 1)

# Build models
model1 <- lm(SBP ~ FRW, data = fram)
model2 <- lm(SBP ~ FRW + SEX + CHOL, data = fram)
model3 <- lm(SBP ~ FRW + SEX + CHOL + FRW:SEX, data = fram)
```

Key Observations from the Multiple Regression Results:

1. **Minimal coefficient change:** The FRW coefficient changes only slightly when adding other predictors (compare Model 1 vs Model 2 outputs above). This stability suggests weight has a robust relationship with blood pressure that isn't confounded by sex or cholesterol.
2. **Sex effect in this cohort:** In this Framingham cohort, the model shows males have somewhat lower blood pressure than females (see the SEX coefficient in Model 2 output). This pattern in the 1950s-60s data (mean age 52 years) may reflect post-menopausal effects in women.
3. **Modest R-squared improvement:** Adding sex and cholesterol only marginally improves R^2 (compare the R-squared values between Model 1 and Model 2). This teaches us that more predictors don't always mean much better predictions.
4. **No meaningful interaction:** The interaction term in Model 3 is near zero (see FRW:SEX coefficient), suggesting weight affects blood pressure similarly for both sexes. Not all hypothesized interactions turn out to be important!

Real Data Lessons

These Framingham results illustrate important realities of data analysis:

- **Effects can be counterintuitive:** Women having higher blood pressure in this 1950s-60s cohort (mean age 52) surprises many, but it's consistent across all age groups in the data. Historical context and demographics matter!
- **More predictors much better fit:** Despite adding two predictors and an interaction, we only explained an additional 1.5% of variance.
- **Most interactions are null:** We often hypothesize interactions that don't materialize. That's fine – testing and rejecting hypotheses is part of science.
- **Biological systems are complex:** Even our best model explains only 12.5% of blood pressure variation. The remaining 87.5% comes from genetics, lifestyle, measurement error, and countless other factors.

9.4.3 Model Selection: Choosing the Right Predictors

With many potential predictors, a critical question arises: which ones should we include? Including too few predictors (**underfitting**) leads to bias; including too many (**overfitting**) increases variance and reduces interpretability. **Model selection** seeks the sweet spot.

9.4.3.1 The Core Problem

When we have k potential predictors, there are 2^k possible models (each predictor is either in or out). With just 10 predictors, that's 1,024 models; with 20 predictors, over a million! We need:

1. A way to score each model's quality
2. An efficient search strategy to find the best model

9.4.3.2 Scoring Models: The Bias-Variance Trade-off

The fundamental challenge is that training error – how well the model fits the data used to build it – is a bad guide to how well it will predict new data. Complex models always fit training data better, but they may perform poorly on new data. This is a manifestation of the bias-variance tradeoff we studied in Chapter 3: as we add more predictors to a regression, bias decreases (better fit to the true relationship) but variance increases (more sensitivity to the particular sample). Too few predictors leads to **underfitting** (high bias), while too many leads to **overfitting** (high variance).

Prediction Risk (Quadratic Loss)

For a model S with predictors \mathcal{X}_S , the prediction risk under quadratic loss is:

$$R(S) = \sum_{i=1}^n \mathbb{E}[(\hat{Y}_i(S) - Y_i^*)^2]$$

where Y_i^* is a future observation at covariate value X_i , and $\hat{Y}_i(S)$ is the prediction from model S .

i Why Squared Error Loss?

We use quadratic loss throughout model selection because:

1. It matches our least squares estimation method (consistency across model fitting and model evaluation)
2. It leads to tractable bias-variance decompositions
3. It penalizes large errors more than small ones (often desirable in practice)

Other loss functions (absolute error, 0-1 loss) are valid but lead to different optimal models.

Since we can't directly compute prediction risk (we don't have future data!), we need estimates. Many model selection criteria follow a similar form which we want to maximize:

$$\text{Model Score} = \underbrace{\text{Goodness of Fit}}_{\text{how well model fits data}} - \underbrace{\text{Complexity Penalty}}_{\text{penalty for too many parameters}}$$

Equivalently, some model selection metrics aim to minimize:

$$\text{Model Score Loss} = \text{Training Error} + \text{Complexity Penalty}$$

This fundamental trade-off appears in different guises across the methods we'll examine. The key insight is that we must balance how well we fit the current data against the danger of overfitting.

Multiple Perspectives

Mallow's Cp

Mallow's C_p Statistic provides an estimate of prediction risk:

$$\hat{R}(S) = \text{RSS}(S) + 2|S|\hat{\sigma}^2$$

where:

- $\text{RSS}(S)$ = residual sum of squares (training error)
- $|S|$ = number of parameters in model S
- $\hat{\sigma}^2$ = error variance estimate from the full model

Interpretation: The first term measures lack of fit, the second penalizes complexity. Named after statistician [Colin Mallows](#) who developed it.

When to use: Linear regression with normal errors when you want an unbiased estimate of prediction risk.

AIC

AIC (Akaike Information Criterion) takes an information-theoretic approach. The standard definition used in statistical software is:

$$\text{AIC}(S) = -2\ell_S + 2|S|$$

where: - ℓ_S is the log-likelihood at the MLE - $|S|$ is the number of parameters in model S (including the intercept)

We **minimize** AIC to select the best model. Conceptually, this is equivalent to maximizing “goodness of fit minus complexity penalty” since minimizing $-2\ell_S + 2|S|$ is the same as maximizing $\ell_S - |S|$.

Key insight: For linear regression with normal errors, AIC is equivalent to Mallows's C_p (they select the same model).

Philosophy: AIC was developed by statistician [Hirotugu Akaike](#) to approximate the Kullback-Leibler divergence between the true and fitted models. It aims to minimize prediction error, not find the “true” model – recognizing that the true model might be too complex to express mathematically.

When to use: When prediction accuracy is the primary goal and you believe the true model may be complex.

BIC

BIC (Bayesian Information Criterion) adds a stronger complexity penalty. The standard definition is:

$$\text{BIC}(S) = -2\ell_S + |S| \log n$$

where n is the sample size. We **minimize** BIC to select the best model.

Note that the penalty $|S| \log n$ grows with sample size, making BIC more conservative than AIC (which has a fixed penalty of $2|S|$). For $n > e^2 \approx 7.4$, BIC penalizes complexity more heavily than AIC.

Philosophy: BIC, developed by statistician Gideon E. Schwarz, has a Bayesian interpretation – it approximates the log posterior probability of the model. As $n \rightarrow \infty$, BIC selects the true model with probability 1 (consistency), *if the true model belongs to the candidate model set*.

Key difference from AIC: Stronger penalty leads to simpler models. BIC assumes a true, relatively simple model exists among the candidates.

When to use: When you believe a relatively simple true model exists and want consistency.

Cross-Validation

Alternatively, we can approximate prediction error by training our model on a **subset** of the data, and testing on the remaining (held-out) set. Repeating this procedure multiple times for different partitions of the data is called **cross-validation** (CV).

Leave-one-out Cross-Validation (LOO-CV) directly estimates prediction error on one held-out data point at a time:

$$\hat{R}_{CV}(S) = \sum_{i=1}^n (Y_i - \hat{Y}_{(i)})^2$$

where $\hat{Y}_{(i)}$ is the prediction for observation i from a model fit without observation i . LOO-CV can be very expensive (requires refitting the model n times!) but for linear models it can be computed efficiently.

k -fold CV: Divide data into k groups called “folds” (often $k = 5$ or $k = 10$), train on $k - 1$ folds, test on the held-out fold, repeat and average. This only requires retraining the model k times.

When to use: When you want a direct, model-agnostic estimate of prediction performance. Essential for complex models where AIC/BIC aren't available. CV is a common evaluation technique in machine learning.

9.4.3.3 Search Strategies

Even with a scoring criterion, we can't check all 2^k models when k is large. Common search strategies include:

- **Forward Stepwise Selection:** Start with no predictors, add the best one at each step

- **Backward Stepwise Selection:** Start with all predictors, remove the worst one at each step
- **Best Subset Selection:** Check all models of each size (computationally intensive)

🔥 Greedy Search Limitations

Stepwise methods are greedy algorithms – they make locally optimal choices without considering the global picture. They may miss the best model. For example, two predictors might be useless alone but powerful together due to interaction effects.

9.4.3.4 Comparing Predictor Importance

Once we've selected a model, we often want to know: which predictors have the most impact? The raw regression coefficients can be misleading because predictors are on different scales. A predictor measured in millimeters will have a much smaller coefficient than one measured in kilometers, even if they have the same actual importance.

The solution is to **standardize predictors before comparing coefficients**. After standardization:

- All predictors have mean 0 and the same spread
- Coefficients become directly comparable
- The coefficient magnitude indicates relative importance

Gelman & Hill's recommendation: Gelman and Hill (2007) recommend dividing continuous predictors by 2 standard deviations (not 1). This makes binary and continuous predictors more comparable, since a binary predictor's standard deviation is at most 0.5, so dividing by 2 SDs puts it on a similar scale.

After standardization, predictors with larger coefficient magnitudes have stronger effects on the outcome (in standard deviation units). However, when predictors are correlated, standardized coefficients don't directly measure the unique variance explained by each predictor. For that, consider partial R^2 or other variance decomposition methods.

💡 Heuristics for Model Selection

Beyond automated criteria, Gelman and Hill (2007) suggest these practical guidelines:

1. **Include predictors you expect to be important** based on subject knowledge
2. **Consider creating composite predictors:** Not all related variables need separate inclusion – you can combine multiple covariates into meaningful composites (e.g., a socioeconomic index from income, education, and occupation)
3. **Add interaction terms for strong predictors:** When predictors have large effects, their interactions often matter too
4. **Use statistical significance and sign to guide decisions:**
 - **Significant with expected sign:** Keep these predictors
 - **Significant with unexpected sign:** Investigate further – may indicate model misspecification, confounding, or data issues
 - **Non-significant with expected sign:** Often worth keeping if theoretically important
 - **Non-significant with unexpected sign:** Generally drop these

Remember: statistical significance isn't everything. A predictor's theoretical importance and practical significance matter too.

9.4.3.5 Controlling for Background Variables

Many studies **control** for background variables (age, sex, education, socioeconomic status, etc.). This simply means **including these variables as predictors** in the model to “remove their impact” on the relationship of interest.

For example, in our [earlier Framingham analysis](#), Model 2 controlled for sex and cholesterol when examining the weight-blood pressure relationship:

- Model 1: SBP ~ FRW (simple regression with weight only)
- Model 2: SBP ~ FRW + SEX + CHOL (controlling for sex and cholesterol)

The weight coefficient changed only slightly between models (see the outputs above), suggesting the relationship isn't confounded by these variables.

Limitation of Statistical Control

Controlling only captures **linear effects** of the control variables. If age has a nonlinear effect on the outcome (e.g., quadratic), simply including age as a linear term won't fully control for it. Background variables can still affect inferences if their effects are nonlinear.

Consider including polynomial terms or splines for control variables when you suspect nonlinear relationships.

9.4.4 Regression Assumptions and Diagnostics

Linear regression makes strong assumptions. When violated, our inferences may be invalid.

The Five Assumptions of Linear Regression

In decreasing order of importance (per Gelman and Hill 2007):

1. **Validity:** Are the data relevant to your research question? Does the outcome measure what you think it measures? Are all important predictors included? Missing key variables can invalidate all conclusions.
2. **Additivity and Linearity:** The model assumes $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots$. If relationships are nonlinear, consider transformations (log, square root), polynomial terms, or interaction terms.
3. **Independence of Errors:** Each observation's error should be independent of others. Watch out for time series (temporal correlation), spatial data (geographic clustering), or grouped data (students within schools).
4. **Equal Variance (Homoscedasticity):** Error variance should be constant across all predictor values. Violation makes standard errors and confidence intervals unreliable.
5. **Normality of Errors:** Errors should follow a normal distribution. This is the least important – with large samples, the Central Limit Theorem ensures valid inference even with non-normal errors. Outliers matter more than the exact distribution shape.

9.4.4.1 Checking Assumptions: Residual Plots

Since the statistical assumptions of linear regression focus on the errors $\epsilon_i = Y_i - \hat{Y}_i$, visualizing the residuals provides our primary diagnostic tool. The most important plot is **residuals versus fitted values**, which can reveal multiple assumption violations at once.

We demonstrated this in [Step 6 of our Framingham analysis](#), where we created two key diagnostic plots:

- **Residuals vs Fitted:** Reveals problems with linearity and constant variance assumptions.
- **Q-Q Plot:** Checks whether residuals follow a normal distribution.

Interpreting Residual Patterns

Good residuals look like **random noise** around zero – no patterns, just scatter. Specific patterns reveal specific problems:

- **Curved patterns** (U-shape, waves): Nonlinearity detected. The true relationship isn't straight. Try transformations or polynomial terms.
- **Funnel shape** (variance changes with fitted values): Heteroscedasticity. Errors have unequal

variance. Consider log-transforming Y or weighted least squares.

- **Outliers or extreme points:** Can dominate the entire regression. Check if they're data errors or reveal model limitations.

Correlation vs. Causation

A significant regression coefficient does **not** imply causation! Regression finds associations, not causal relationships. For example, ice cream sales and swimming pool drownings are positively correlated (both increase in summer), but ice cream doesn't cause drowning.

Establishing causation requires theoretical justification, temporal precedence, ruling out confounders, and ideally randomized experiments. We'll explore causal inference in detail in Chapter 11.

9.5 Logistic Regression

9.5.1 Modeling Binary Outcomes

So far, we've assumed the response variable Y is continuous. But what if Y is binary? Consider:

- Does a patient have the disease? (Yes/No)
- Will a customer churn? (Yes/No)³
- Did the email get clicked? (Yes/No)
- Will a loan default? (Yes/No)

For these binary outcomes, we need **logistic regression**.

9.5.2 The Logistic Regression Model

The Logistic Regression Model

For a binary outcome $Y_i \in \{0, 1\}$ and predictors X_i , the logistic regression model specifies:

$$p_i \equiv \mathbb{P}(Y_i = 1 | X_i) = \frac{e^{\beta_0 + \sum_{j=1}^k \beta_j X_{ij}}}{1 + e^{\beta_0 + \sum_{j=1}^k \beta_j X_{ij}}}$$

Equivalently, using the **logit** (log-odds) transformation:

$$\text{logit}(p_i) = \log \left(\frac{p_i}{1 - p_i} \right) = \beta_0 + \sum_{j=1}^k \beta_j X_{ij}$$

Multiple Perspectives

Intuitive

Where does this formula come from? Why “logistic” regression?

Point is, when Y is binary, linear regression produces continuous predictions – not the 0s and 1s we observe with binary data. The natural approach is to model the probability $p = \mathbb{P}(Y = 1 | X)$ instead of Y .

We could try to model the probability p with a linear model such as $p = \beta_0 + \beta_1 X$. However, we would immediately hit two problems:

³Customer churn refers to when customers stop doing business with a company or cancel their subscription to a service. For example, a mobile phone customer “churns” when they switch to a different provider or cancel their contract. Predicting churn helps companies identify at-risk customers and take preventive action.

1. Linear functions are unbounded: when X is large, $\beta_0 + \beta_1 X$ can exceed 1; when X is small, it can fall below 0. Both give impossible “probabilities.”
2. Binary data strongly violates the homoscedasticity assumption – a Bernoulli (i.e., binary) variable with probability p has variance $p(1 - p)$, which depends on X .

The trick is that instead of modelling directly p , we model the **log-odds** (logarithm of the probability ratio), that is $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$, which is an unbounded quantity that lives in $(-\infty, \infty)$.

The **logistic function** is then the function that allows us to map the logit values back to $(0, 1)$, ensuring valid probabilities regardless of predictor values.

Mathematical

The logistic function emerges naturally from maximum likelihood with Bernoulli data. Since $Y_i \sim \text{Bernoulli}(p_i)$, the likelihood is:

$$\mathcal{L} = \prod_{i=1}^n p_i^{Y_i} (1 - p_i)^{1 - Y_i}$$

The log-likelihood becomes:

$$\ell = \sum_{i=1}^n [Y_i \log p_i + (1 - Y_i) \log(1 - p_i)]$$

We need to connect p_i to the predictors X_i . We could try various transformations, but the **logit** turns out to be special – it’s the “canonical link” for Bernoulli distributions, meaning it makes the mathematics particularly elegant. Specifically, if we set:

$$\log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_1 X_i$$

then the log-likelihood becomes **concave** in the parameters β_0, β_1 . This is crucial: a concave function has a unique maximum, so we’re guaranteed to find the best fit without worrying about local maxima. Solving the logit equation for p_i gives us the logistic function.

Computational

Let’s visualize the logistic function to build intuition for how it transforms linear predictors into probabilities:

```

import numpy as np
import matplotlib.pyplot as plt

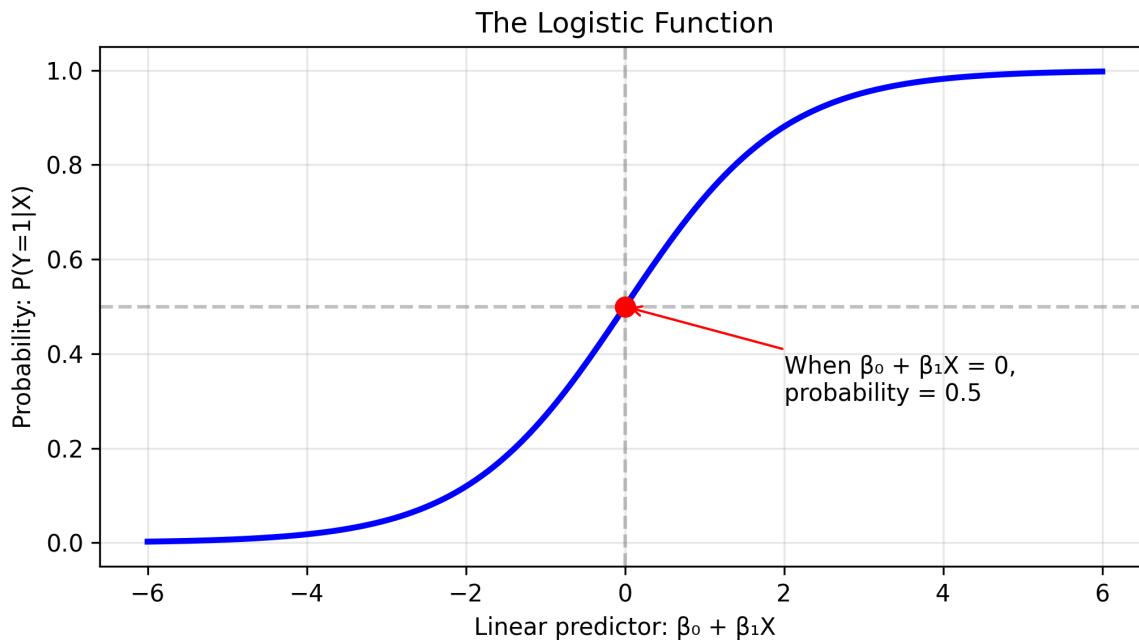
x = np.linspace(-6, 6, 200)
p = 1 / (1 + np.exp(-x))

plt.figure(figsize=(7, 4))
plt.plot(x, p, 'b-', linewidth=2.5)
plt.axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
plt.axvline(x=0, color='gray', linestyle='--', alpha=0.5)
plt.xlabel('Linear predictor: + X')
plt.ylabel('Probability: P(Y=1|X)')
plt.title('The Logistic Function')
plt.grid(True, alpha=0.3)
plt.ylim(-0.05, 1.05)

# Mark key point
plt.plot(0, 0.5, 'ro', markersize=8)
plt.annotate('When + X = 0,\nprobability = 0.5',
             xy=(0, 0.5), xytext=(2, 0.3),
             arrowprops=dict(arrowsize=10, color='red'))

plt.tight_layout()
plt.show()

```



The S-shaped curve is the logistic function in action. Notice three critical features:

1. **Bounded output:** No matter how extreme the input ($+ X$), the output stays strictly between 0 and 1
2. **Decision boundary:** When the linear predictor equals 0, the probability equals 0.5 – this is the natural decision threshold
3. **Smooth transitions:** Unlike a hard step function, the logistic provides gradual probability changes, reflecting uncertainty near the boundary

This smooth mapping from $(-\infty, \infty) \rightarrow (0, 1)$ is what makes logistic regression both mathematically tractable and practically interpretable.

Unlike linear regression, logistic regression has no closed-form solution. The parameters are estimated via maximum likelihood using numerical optimization algorithms, as implemented in common statistical packages.

i Interpreting Coefficients: Odds Ratios

In logistic regression, coefficients have a specific interpretation:

- β_j is the change in **log-odds** for a one-unit increase in X_j , holding other variables constant
- e^{β_j} is the **odds ratio**: the factor by which odds are multiplied for a one-unit increase in X_j

For example, if $\beta_{\text{age}} = 0.05$, then:

- Each additional year of age increases log-odds by 0.05
- Each additional year multiplies odds by $e^{0.05} \approx 1.051$ (5.1% increase)

Remember: odds = $p/(1-p)$. If $p = 0.2$, odds = 0.25. If $p = 0.8$, odds = 4.

9.5.3 Logistic Regression in Practice

Let's apply logistic regression to the Framingham data to predict high blood pressure.

Example: Predicting High Blood Pressure

We'll create a **binary outcome** for high blood pressure using the clinical definition: systolic blood pressure (**SBP**) 140 mmHg or diastolic blood pressure (**DBP**) 90 mmHg. This is the standard threshold used in medical practice.

To model the probability of high blood pressure, we'll:

1. Standardize continuous predictors (weight, age, cholesterol) by dividing by 2 standard deviations – this makes coefficients comparable across predictors.
2. Fit logistic regression models, starting with a single predictor then adding multiple variables.
3. Interpret the results through odds ratios.

Multiple Perspectives

Python

```

import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt

# Load the Framingham data
fram = pd.read_csv('../data/fram.txt', sep='\t', index_col=0)

# Define high blood pressure (standard clinical threshold)
fram['HIGH_BP'] = ((fram['SBP'] >= 140) | (fram['DBP'] >= 90)).astype(int)

print(f"Prevalence of high BP: {fram['HIGH_BP'].mean():.1%} ({fram['HIGH_BP'].sum()} of {len(fram)})")

# Standardize predictors (following Gelman & Hill's recommendation)
# Dividing by 2*SD makes binary and continuous predictors comparable
def standardize(x):
    return (x - x.mean()) / (2 * x.std())

fram['sFRW'] = standardize(fram['FRW'])
fram['sAGE'] = standardize(fram['AGE'])
fram['sCHOL'] = standardize(fram['CHOL'])

# Fit a simple logistic regression with standardized weight
model = smf.logit('HIGH_BP ~ sFRW', data=fram).fit(disp=0)
print("\n" + "="*50)
print("Logistic Regression: HIGH_BP ~ sFRW")
print("="*50)
print(model.summary2().tables[1])

# Visualize the fitted model
plt.figure(figsize=(7, 5))

# Plot the data points (with slight jitter for visibility)
y_jitter = fram['HIGH_BP'] + np.random.normal(0, 0.02, len(fram))
plt.scatter(fram['sFRW'], y_jitter, alpha=0.3, s=20)

# Plot the fitted probability curve
x_range = np.linspace(fram['sFRW'].min(), fram['sFRW'].max(), 200)
X_pred = pd.DataFrame({'sFRW': x_range})
y_pred = model.predict(X_pred)
plt.plot(x_range, y_pred, 'r-', linewidth=2, label='Fitted probability')

plt.xlabel('Standardized Weight (sFRW)')
plt.ylabel('P(High BP = 1)')
plt.title('Probability of High Blood Pressure vs Weight')
plt.ylim(-0.05, 1.05)
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()

# Interpret the coefficient as odds ratio
beta_sfrw = model.params['sFRW']
or_sfrw = np.exp(beta_sfrw)
print(f"\nCoefficient for sFRW: {beta_sfrw:.4f}")
print(f"Odds ratio: {or_sfrw:.4f}")
print(f"Interpretation: A 2-SD increase in weight multiplies odds of high BP by {or_sfrw:.4f}")

```

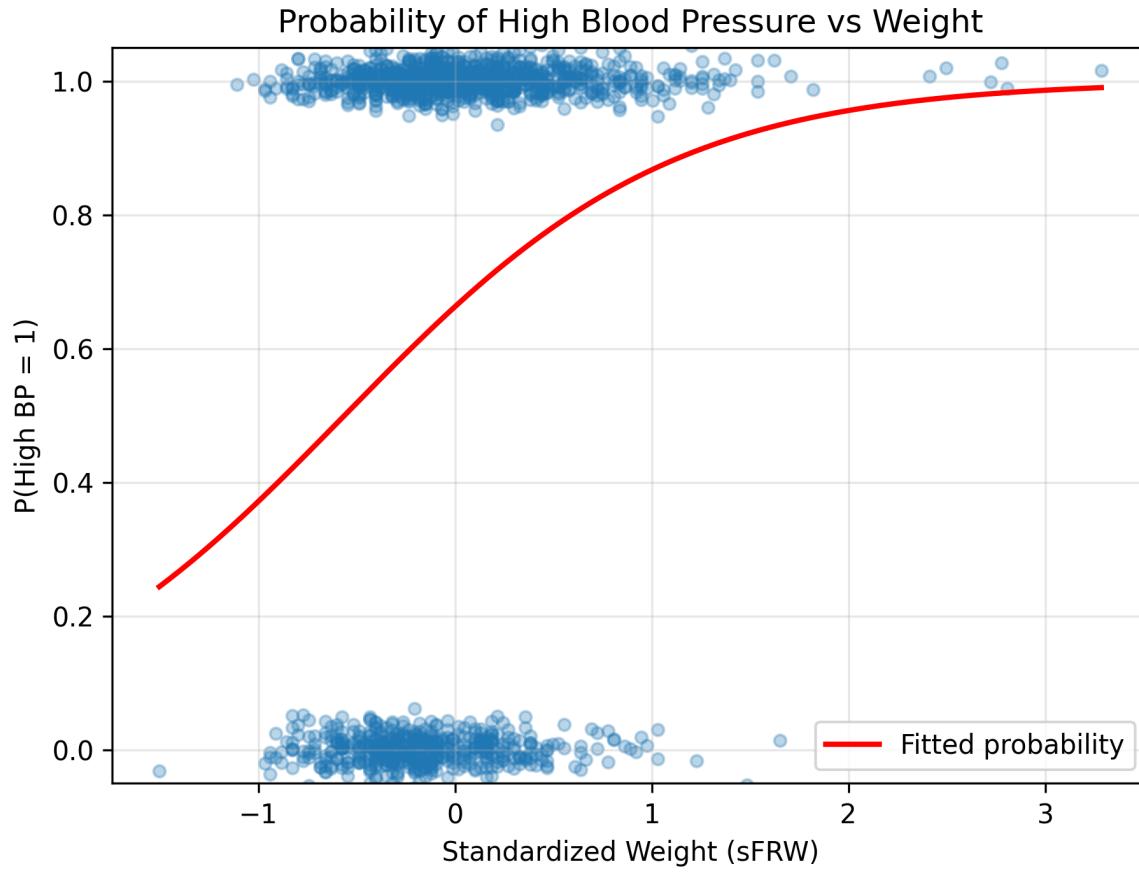
```

Prevalence of high BP: 65.0% (906 of 1394)

=====
Logistic Regression: HIGH_BP ~ sFRW
=====
      Coef.  Std.Err.      z     P>|z|      [0.025    0.975]
Intercept  0.675629  0.059255  11.402124  4.080404e-30  0.559492  0.791766
sFRW       1.201906  0.138943   8.650323  5.135384e-18  0.929582  1.474230

Coefficient for sFRW: 1.2019
Odds ratio: 3.3265
Interpretation: A 2-SD increase in weight multiplies odds of high BP by 3.3265

```



R

```

library(arm) # For rescale and invlogit functions

# Load data
fram <- read.csv('../data/fram.txt', sep='\t', row.names = 1)

# Define high blood pressure
fram$HIGH_BP <- (fram$SBP >= 140) | (fram$DBP >= 90)

cat(sprintf("Prevalence of high BP: %.1f%% (%d of %d)\n",
            mean(fram$HIGH_BP)*100, sum(fram$HIGH_BP), nrow(fram)))

# Standardize predictors (rescale divides by 2*SD as recommended by Gelman & Hill)
fram$sFRW <- rescale(fram$FRW)
fram$sAGE <- rescale(fram$AGE)
fram$sCHOL <- rescale(fram$CHOL)

# Fit logistic regression with standardized weight
fit <- glm(HIGH_BP ~ sFRW, data = fram, family = binomial(link = 'logit'))
summary(fit)

# Visualize the fitted model
plot(fram$sFRW, jitter(as.numeric(fram$HIGH_BP), 0.05),
      xlab = "Standardized Weight (sFRW)",
      ylab = "P(High BP = 1)",
      main = "Probability of High Blood Pressure vs Weight",
      pch = 16, col = rgb(0, 0, 0, 0.3))

# Add fitted curve
curve(invlogit(coef(fit)[1] + coef(fit)[2]*x),
      add = TRUE, col = "red", lwd = 2)
legend("topleft", "Fitted probability", col = "red", lwd = 2)

# Interpret as odds ratio
beta_sfrw <- coef(fit)["sFRW"]
or_sfrw <- exp(beta_sfrw)
cat(sprintf("\nCoefficient for sFRW: %.4f\n", beta_sfrw))
cat(sprintf("Odds ratio: %.4f\n", or_sfrw))
cat(sprintf("Interpretation: A 2-SD increase in weight multiplies odds of high BP by %.4f\n", or_sfrw))

```

The fitted logistic curve shows how the probability of high blood pressure increases with weight. Unlike linear regression, the relationship is nonlinear – the effect of weight on probability is strongest in the middle range where probabilities are near 0.5.

9.5.3.1 Adding Multiple Predictors

Now let's include age and sex to improve our model:

Multiple Perspectives

Python

```

# Fit model with multiple predictors (using standardized variables)
model2 = smf.logit('HIGH_BP ~ sFRW + sAGE + SEX', data=fram).fit(disp=0)

print("=="*50)
print("Multiple Logistic Regression")
print("=="*50)
print(model2.summary2().tables[1])

# Calculate and display odds ratios
print("\n" + "=="*50)
print("ODDS RATIOS")
print("=="*50)
for var in model2.params.index:
    or_val = np.exp(model2.params[var])
    ci = model2.conf_int().loc[var]
    ci_low, ci_high = np.exp(ci[0]), np.exp(ci[1])
    if var != 'Intercept':
        print(f'{var}: OR = {or_val:5.3f} (95% CI: {ci_low:5.3f}-{ci_high:5.3f})')

# Visualize the effect of sex on the weight-BP relationship
plt.figure(figsize=(7, 5))

# Plot the actual data points with jitter, separated by sex
females = fram[fram['SEX'] == 'female']
males = fram[fram['SEX'] == 'male']

# Add jitter to binary outcome for visibility
jitter_f = females['HIGH_BP'] + np.random.normal(0, 0.02, len(females))
jitter_m = males['HIGH_BP'] + np.random.normal(0, 0.02, len(males))

plt.scatter(females['sFRW'], jitter_f, alpha=0.3, s=20, color='red', label='Female (data)')
plt.scatter(males['sFRW'], jitter_m, alpha=0.3, s=20, color='blue', label='Male (data)')

# Create prediction data: vary weight, hold age at mean (0 for standardized)
weight_range = np.linspace(fram['sFRW'].min(), fram['sFRW'].max(), 100)

# Predictions for females
pred_data_f = pd.DataFrame({'sFRW': weight_range, 'sAGE': 0, 'SEX': 'female'})
pred_f = model2.predict(pred_data_f)

# Predictions for males
pred_data_m = pd.DataFrame({'sFRW': weight_range, 'sAGE': 0, 'SEX': 'male'})
pred_m = model2.predict(pred_data_m)

# Plot the fitted curves
plt.plot(weight_range, pred_f, 'r-', linewidth=2.5, label='Female (fitted)')
plt.plot(weight_range, pred_m, 'b-', linewidth=2.5, label='Male (fitted)')

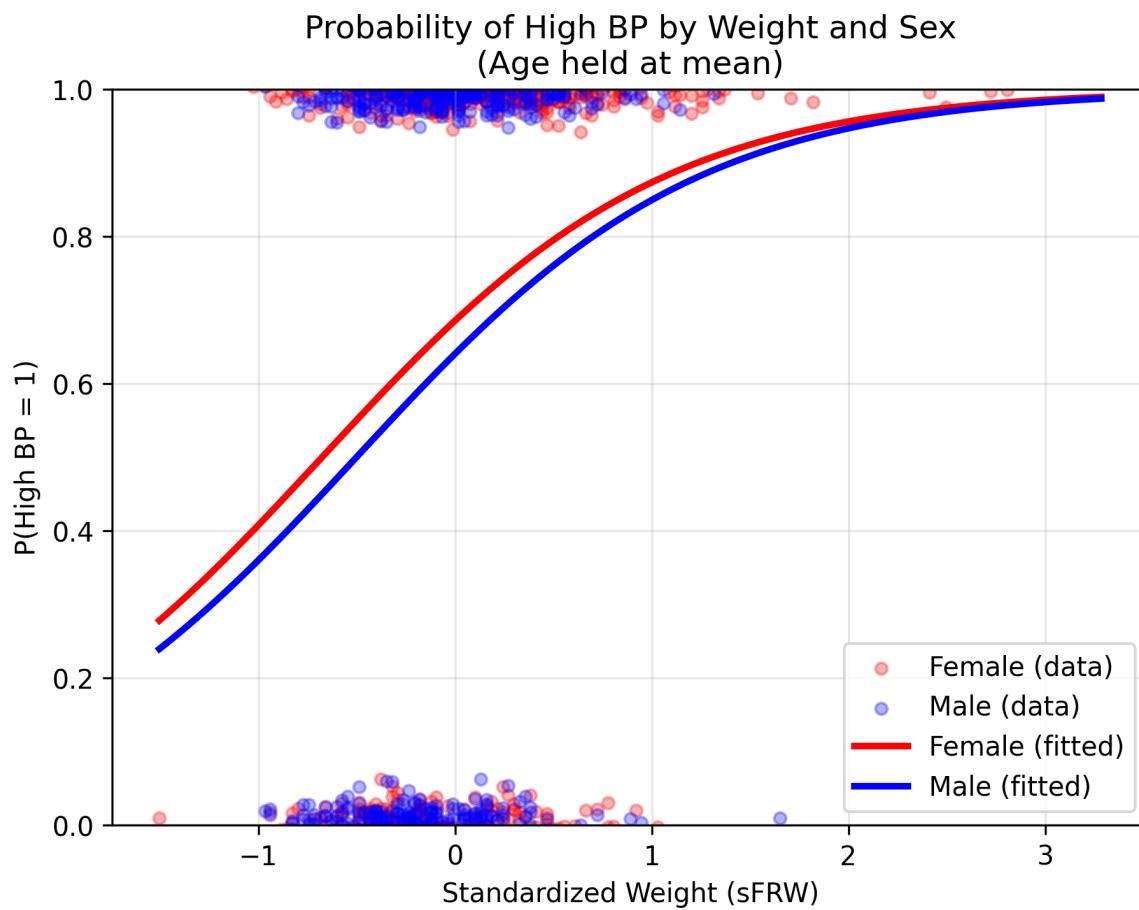
plt.xlabel('Standardized Weight (sFRW)')
plt.ylabel('P(High BP = 1)')
plt.title('Probability of High BP by Weight and Sex\n(Age held at mean)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.ylim(0, 1)
plt.show()

```

```
=====
Multiple Logistic Regression
=====
      Coef.  Std.Err.      z      P>|z|      [0.025    0.975]
Intercept  0.780125  0.083319  9.363138  7.740317e-21  0.616823  0.943426
SEX[T.male] -0.201763  0.116912 -1.725767  8.438929e-02 -0.430907  0.027380
sFRW       1.153445  0.140824  8.190707  2.596960e-16  0.877436  1.429454
sAGE       0.373807  0.117219  3.188958  1.427867e-03  0.144062  0.603552
```

```
=====
ODDS RATIOS
=====
```

```
SEX[T.male]: OR = 0.817 (95% CI: 0.650-1.028)
sFRW      : OR = 3.169 (95% CI: 2.405-4.176)
sAGE      : OR = 1.453 (95% CI: 1.155-1.829)
```



R

```

# Fit model with multiple predictors (using standardized variables)
fit2 <- glm(HIGH_BP ~ sFRW + sAGE + SEX, data = fram,
             family = binomial(link = 'logit'))
summary(fit2)

# Calculate odds ratios with confidence intervals
or_table <- exp(cbind(OR = coef(fit2), confint(fit2)))
print(or_table)

# Visualize the effect of sex on the weight-BP relationship
# Plot the actual data points with jitter, separated by sex
females <- fram[fram$SEX == "female", ]
males <- fram[fram$SEX == "male", ]

plot(females$sFRW, jitter(as.numeric(females$HIGH_BP), 0.05),
      col = rgb(1, 0, 0, 0.3), pch = 16,
      xlab = "Standardized Weight (sFRW)",
      ylab = "P(High BP = 1)",
      main = "Probability of High BP by Weight and Sex\n(Age held at mean)",
      ylim = c(-0.1, 1.1))
points(males$sFRW, jitter(as.numeric(males$HIGH_BP), 0.05),
       col = rgb(0, 0, 1, 0.3), pch = 16)

# Add fitted curves for females and males (age at mean = 0 for standardized)
# Female curve
curve(invlogit(coef(fit2)[1] + coef(fit2)["sFRW"]*x + coef(fit2)["sAGE"]*0),
      col = "red", lwd = 2.5, add = TRUE)

# Male curve
curve(invlogit(coef(fit2)[1] + coef(fit2)["SEXmale"] + coef(fit2)["sFRW"]*x + coef(fit2)["sAGE"]*0),
      col = "blue", lwd = 2.5, add = TRUE)

legend("topleft",
       c("Female (data)", "Male (data)", "Female (fitted)", "Male (fitted)"),
       col = c(rgb(1, 0, 0, 0.5), rgb(0, 0, 1, 0.5), "red", "blue"),
       pch = c(16, 16, NA, NA),
       lwd = c(NA, NA, 2.5, 2.5))
grid()

```

Interpretation of Results:

- **sFRW (Weight):** A 2-SD increase in weight multiplies the odds of high BP by 3.17 (95% CI: 2.41-4.18)
– a very strong effect
- **sAGE (Age):** A 2-SD increase in age multiplies the odds of high BP by 1.45 (95% CI: 1.16-1.83) – significant but weaker than weight
- **SEX:** Being male decreases the odds by about 18% (OR = 0.82), but this is not statistically significant (95% CI: 0.65-1.03 includes 1)

The visualization shows how the probability curves are parallel on the logit scale – males have consistently lower probability across all weight values (the blue curve sits below the red curve). The standardization allows direct comparison: weight has the strongest association with high blood pressure in this model.

i Remarks About Logistic Regression

1. **No R-squared:** The usual R^2 doesn't apply. Pseudo- R^2 measures exist but are less interpretable.
2. **Classification vs. Probability:** Logistic regression estimates probabilities. Classification (yes/no) requires choosing a threshold (often 0.5, but domain-specific considerations matter).
3. **Separation Problem:** If predictors perfectly separate the classes, the MLE doesn't exist (coefficients go to $\pm\infty$). Regularization or Bayesian methods can help.
4. **Sample Size Requirements:** Need more data than linear regression. Rule of thumb: 10-20 events per predictor for the less common outcome.
5. **Link Functions:** The logit is just one choice, another choice is for example the [probit](#) (normal CDF).

9.6 Chapter Summary and Connections

9.6.1 Key Concepts Review

We've covered the two fundamental models in statistical learning:

Linear Regression:

- Models the expected value of a continuous response as a linear function of predictors
- Estimated via least squares, which coincides with MLE under normality
- Provides interpretable coefficients with well-understood inference procedures
- Extends naturally to multiple predictors, with matrix formulation
- Requires careful attention to assumptions and model selection

Logistic Regression:

- Extends the linear framework to binary outcomes via the logit link
- Models probabilities, not the outcomes directly
- Coefficients represent changes in log-odds, interpretable as odds ratios
- Estimated via maximum likelihood with iterative algorithms
- Shares the interpretability advantages of linear models

Key Connections:

Both models exemplify the fundamental statistical modeling workflow:

1. Specify a model (assumptions about the data-generating process)
2. Estimate parameters (least squares or maximum likelihood)
3. Quantify uncertainty (standard errors, confidence intervals)
4. Check assumptions (diagnostic plots)
5. Make predictions and interpret results

9.6.2 The Big Picture

This chapter has taken you through the fundamentals of linear and logistic regression, from basic concepts to advanced applications. These models exemplify the core statistical modeling workflow: specify a model, estimate parameters, quantify uncertainty, check assumptions, and make predictions.

But why do these simple linear models remain so important in the era of deep learning? The answer lies in their unique ability to explain complex predictions. **LIME (Local Interpretable Model-Agnostic Explanations)** (Ribeiro, Singh, and Guestrin 2016) demonstrates this perfectly: it explains any complex model's predictions by fitting simple linear models locally around points of interest.

This principle – that complex functions are locally linear – makes linear models useful for understanding predictions from any model, no matter how complex. The techniques you've learned in this chapter (least squares, coefficient interpretation, diagnostics) aren't just historical artifacts; they're the foundation for both classical statistical analysis and modern interpretable machine learning.

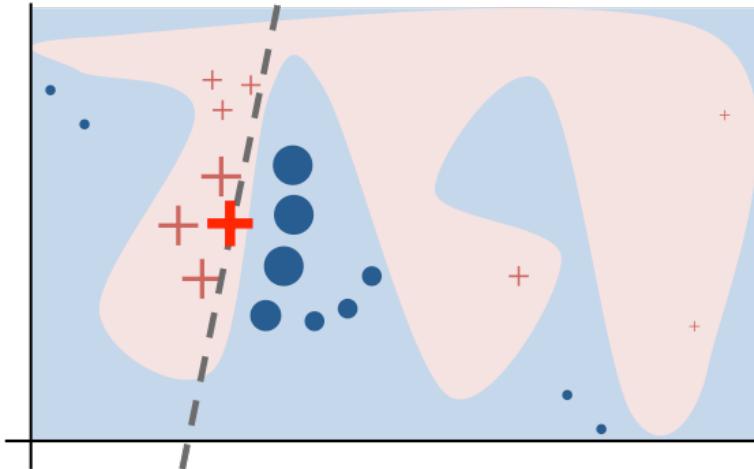


Figure 9.1: LIME: Local linear models explain complex predictions by approximating them in small neighborhoods. Figure reproduced from Ribeiro, Singh, and Guestrin (2016).

9.6.3 Common Pitfalls to Avoid

When working with linear and logistic regression, watch out for these critical mistakes:

1. Interpretation Errors

- **Correlation Causation:** A significant coefficient shows association, not causation (that's Chapter 11's topic!)
- **Statistical Practical significance:** With $n=10,000$, even tiny effects become "significant"
- **Logistic coefficients are log-odds:** A coefficient of 0.5 doesn't mean "50% increase in probability"

2. Model Selection Traps

- **Overfitting:** Using training error to select models guarantees disappointment on new data
- **Automation without thinking:** If your model says ice cream sales *decrease* temperatures, something's wrong
- **Ignoring validation:** Always hold out data – a perfect training fit often means terrible generalization

3. Technical Violations

- **Ignoring diagnostic plots:** That funnel-shaped residual plot? Your model needs help
- **Multicollinearity chaos:** When predictors correlate highly, coefficients become unstable and standard errors explode

4. The Big One: Context Blindness

- **Extrapolation:** Linear trends rarely continue forever (no, humans won't be 20 feet tall in year 3000)
- **Domain knowledge matters:** Statistical criteria (AIC/BIC) are guides, not gospel

9.6.4 Chapter Connections

Previous (Chapters 5-8):

- Chapters 5-6 introduced parametric inference – linear regression is the quintessential parametric model, with least squares achieving the Cramér-Rao bound under normality
- Chapter 7's hypothesis testing framework directly applies to testing regression coefficients via Wald tests and F-tests
- Chapter 8's Bayesian paradigm offers an alternative: Bayesian linear regression incorporates prior knowledge about parameters

This Chapter: Introduced the two workhorses of statistical modeling: linear and logistic regression. We saw how least squares connects to maximum likelihood, how to handle multiple predictors, select models, and extend to binary outcomes. The focus on interpretability makes these models essential even in the era of complex machine learning.

Next (Ch. 10-11):

- Chapter 10 will show Bayesian regression in practice using probabilistic programming languages, with complex priors and hierarchical structures
- Chapter 11 will distinguish association from causation – regression finds associations, not causal effects

Applications: Linear models remain indispensable across fields: economics (modeling market relationships), medicine (risk factor analysis), social sciences (understanding social determinants), machine learning (LIME and interpretability), and A/B testing (variance reduction through regression adjustment).

9.6.5 Self-Test Problems

1. **Centroid Property:** Show that the least squares regression line always passes through the point (\bar{X}, \bar{Y}) .

i Solution Hint

Use the normal equations: from $\frac{\partial \text{RSS}}{\partial \hat{\beta}_0} = 0$ you get $\sum_i (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i) = 0$, which gives $\bar{Y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{X}$.

2. **Multicollinearity Inflates Standard Errors:** Explain why high correlation between X_j and the other predictors increases the standard error of $\hat{\beta}_j$.

i Solution Hint

In multiple regression, $\text{Var}(\hat{\beta}_j) \propto (1 - R_j^2)^{-1}$, where R_j^2 is from regressing X_j on the other X 's. As $R_j^2 \rightarrow 1$, the variance (and SE) blows up.

3. **Binary Outcomes and Heteroscedasticity:** For $Y \sim \text{Bernoulli}(p)$, compute $\text{Var}(Y)$ and explain why applying linear regression to binary Y violates the constant-variance assumption.

i Solution Hint

$\text{Var}(Y) = p(1 - p)$, which depends on X if $p = \mathbb{P}(Y = 1 | X)$ changes with X . The variance is maximized at $p = 0.5$ and approaches 0 as $p \rightarrow 0$ or $p \rightarrow 1$.

4. **Odds Ratio to Probability:** Baseline probability is $p_0 = 0.20$. A predictor has an odds ratio OR = 2.4. What is the new probability?

i Solution Hint

Convert baseline to odds: $o_0 = \frac{p_0}{1-p_0} = \frac{0.20}{0.80} = 0.25$. Multiply by OR to get $o_1 = 2.4 \times 0.25 = 0.6$. Convert back: $p_1 = \frac{o_1}{1+o_1} = \frac{0.6}{1.6} = 0.375$.

5. **Interpreting an Interaction:** In the model $Y = \beta_0 + \beta_1 \cdot \text{FRW} + \beta_2 \cdot \text{SEX} + \beta_3 \cdot (\text{FRW} \times \text{SEX}) + \epsilon$ with SEX = 0 (female) and SEX = 1 (male), what is the effect (slope) of FRW on Y for females vs males? What does β_2 represent?

i Solution Hint

Plug in $\text{SEX} = 0$ and $\text{SEX} = 1$. The FRW slope is β_1 for females and $\beta_1 + \beta_3$ for males. β_2 is the male-female intercept difference when FRW = 0.

9.6.6 Python and R Reference

This section provides a quick reference for the main functions used in linear and logistic regression.

i Python and R Reference Code

Python and R code examples for this chapter can be found in the HTML version of these notes.

9.6.7 Connections to Source Material

i Mapping to Course Materials

Lecture Note Section	Corresponding Source(s)
Introduction: Why Linear Models Still Matter	Lecture 9 slides intro on interpretable ML and LIME
The Power of Interpretability	Expanded from lecture motivation
Linear Models as Building Blocks	New material connecting to GLMs, mixed models, LIME
Simple Linear Regression	AoS §13.1
Regression Models	AoS §13.1
The Simple Linear Regression Model	AoS Definition 13.1
Estimating Parameters: Method of Least Squares	AoS §13.1, Theorem 13.4
Connection to Maximum Likelihood	AoS §13.2
Properties of the Least Squares Estimators	AoS §13.3, Theorems 13.8-13.9
Simple Linear Regression in Practice	New Framingham example from lecture slides, applying concepts from AoS §13.1-13.3
Multiple Linear Regression	AoS §13.5
Extending the Model to Multiple Predictors	AoS §13.5
Least Squares in Matrix Form	AoS Theorem 13.13
Multiple Regression in Practice	New Framingham example from lecture slides, applying concepts from AoS §13.5
Model Selection: Choosing the Right Predictors	AoS §13.6
Scoring Models: The Bias-Variance Trade-off	AoS §13.6
Mallow's Cp, AIC, BIC, Cross-Validation	AoS §13.6
Search Strategies	AoS §13.6
Comparing Predictor Importance	Lecture 9 slides + Gelman & Hill
Controlling for Background Variables	Lecture 9 slides
Regression Assumptions and Diagnostics	Lecture 9 slides (sourcing Gelman & Hill)
The Five Assumptions	Lecture 9 slides (sourcing Gelman & Hill)
Checking Assumptions: Residual Plots	Lecture 9 slides + expanded examples
Logistic Regression	AoS §13.7
Modeling Binary Outcomes	AoS §13.7
The Logistic Regression Model	AoS §13.7

Logistic Regression in Practice

New Framingham example from lecture slides,
applying concepts from AoS §13.7

Chapter Summary and Connections
Python and R Reference

New comprehensive summary
New - added Python alongside R implementations

9.6.8 Further Materials

- Gelman and Hill (2007) - Practical regression guidance with excellent intuition and real-world advice
- Ribeiro, Singh, and Guestrin (2016) - The LIME paper demonstrating how local linear models can explain any classifier's predictions ([GitHub repo](#))

Remember: Start with linear regression – many problems that get a neural network thrown at them could be solved with these simple models. Always establish a linear/logistic baseline first: if your complex deep network only improves accuracy by 2%, is the added complexity, computation, and loss of interpretability really worth it? Master these fundamental methods – they're used daily by practitioners worldwide and remain indispensable as baselines, interpretable models, and building blocks for more complex systems.

References

- Breiman, Leo. 2001. "Statistical Modeling: The Two Cultures (with Comments and a Rejoinder by the Author)." *Statistical Science* 16 (3): 199–231.
- Gelman, Andrew, and Jennifer Hill. 2007. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge: Cambridge University Press.
- Kingma, Diederik P, and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." In *International Conference on Learning Representations (ICLR)*.
- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. 2016. "Why Should i Trust You?: Explaining the Predictions of Any Classifier." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–44. ACM. <https://doi.org/10.1145/2939672.2939778>.
- Stone, Lawrence D, Colleen M Keller, Thomas M Kratzke, and Johan P Strumpfer. 2014. "Search for the Wreckage of Air France Flight AF 447." *Statistical Science* 29 (1): 69–80. <https://doi.org/10.1214/13-STS420>.
- Wasserman, Larry. 2013. *All of Statistics: A Concise Course in Statistical Inference*. Springer Science & Business Media.

Download Complete PDF

About the PDF Version

You are currently reading the PDF version of these lecture notes. This document contains all chapters compiled into a single file for convenient offline reading and printing.

For the interactive HTML version with tabs, code folding, and better navigation features, please visit:
<https://lacerbi.github.io/stats-for-ds-website/>

The HTML version is the recommended format for the best learning experience, especially when working through code examples.

