



Fundamentals of Python for Econometrics

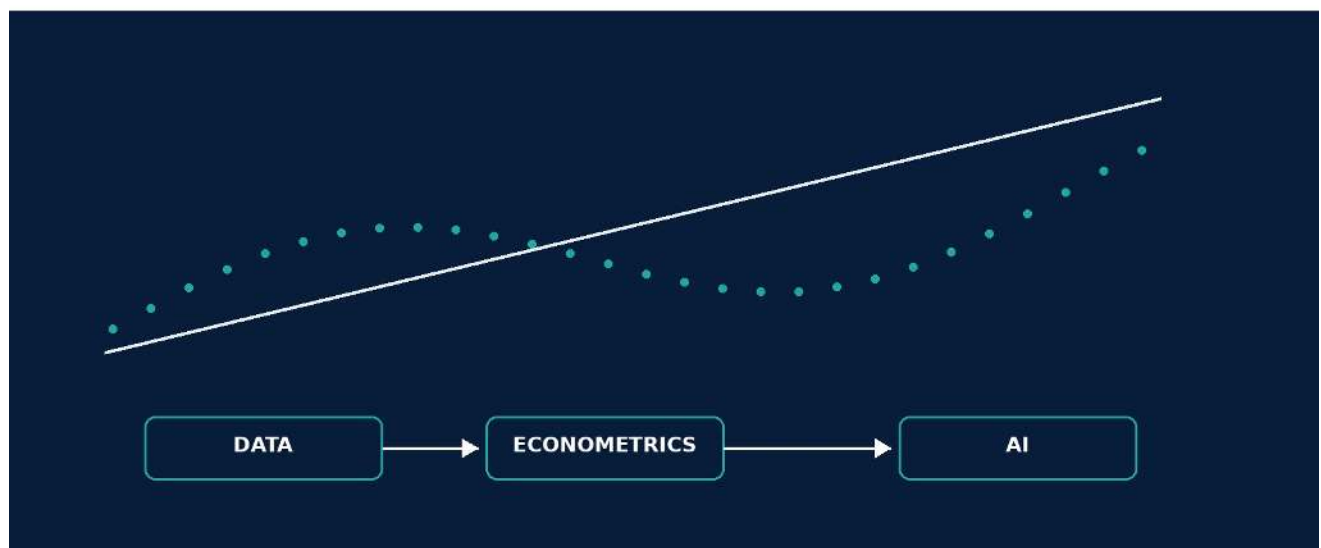
Economics, Data Analysis, Causal Inference, Forecasting, Machine Learning, and Deep Learning

CETERIS LAB STUDENT NOTES

Mohammad Safavi, Ph.D.

Version 1.0 - August 2026

econometrics.safavim.com



Publication Information



Fundamentals of Python for Econometrics

Economics, Data Analysis, Causal Inference, Forecasting, Machine Learning, and Deep Learning

Prepared for **Ceteris LAB** by **Mohammad Safavi, Ph.D.**

Version 1.0, August 2026

Website: <https://econometrics.safavim.com/>

Copyright © 2026 Mohammad Safavi and Ceteris LAB for original Ceteris LAB text, code, exercises, and figures. No public licence is granted for original Ceteris LAB material unless a separate licence file states otherwise. Third-party materials retain their own rights and licences. The supplied IntroAI materials are credited under CC BY 4.0; source financial-time-series lectures are cited and are not redistributed in this package.

This publication is an educational resource. Financial, investment, risk, and option-pricing examples are not personalized investment, legal, accounting, or risk-management advice. AI examples are instructional demonstrations and must be verified before consequential use.

Preface

Python is now a common language across data analysis, economics, finance, econometrics, and artificial intelligence. That breadth is both its advantage and its trap. A beginner can import a powerful model before learning what the data mean, and an experienced analyst can inherit an attractive notebook whose hidden state makes the result impossible to reproduce. This book therefore treats programming as part of analytical reasoning rather than as a separate technical ornament.

The publication begins with variables, collections, control flow, functions, projects, NumPy, pandas, and visualization. It then develops a full econometrics sequence covering regression, inference, endogeneity, instrumental variables, panel data, limited dependent variables, causal inference, and causal machine learning before moving to time series, financial econometrics, machine learning, deep learning, NLP, and AI-assisted research.

The financial-econometrics spine is adapted and substantially modernized from Ruey S. Tsay's 2013 *Analysis of Financial Time Series* lecture notes (Tsay 2013). The AI spine incorporates selected, legally reusable ideas and datasets from Ali Sharifi-Zarchi and contributors' ten-session IntroAI repository under CC BY 4.0 (Sharifi-Zarchi and contributors 2026). Every source figure has been replaced with a new Python-generated visual, and every substantive detected R segment is accounted for in the separate conversion ledger.

How to Use This Book

Each chapter follows the sequence **question, concept, Python code, verified output, visual evidence, interpretation, limitation, and practice**. Students new to programming should complete Parts I and II in order. Readers with prior Python experience may begin with Part III, but should review Chapters 7, 10, and 16 because the project uses strict path, validation, and provenance conventions.

The downloadable source package contains one notebook per chapter. Run notebooks from the project root in numerical order. The student package also includes the frozen datasets that may be legally redistributed, data-retrieval guidance, Python scripts, environment files, and audit reports. Optional advanced sections can be postponed without breaking the core learning path.

Reproducibility and AI Use

All required examples are designed to run without paid APIs. Random examples use documented seeds. External data calls include timeouts and frozen fallbacks where redistribution is permitted. Students may use AI tools to explain errors or propose code, but the submitted analytical work should identify AI assistance, preserve sources, run the code independently, and verify numerical and conceptual claims. A generated citation, coefficient interpretation, or policy claim is not acceptable until it has been checked against evidence.

List of Figures

1. Python connects numerical computing, data management, econometrics, visualization, and artificial intelligence.
2. The reproducible analysis loop used throughout the book.
3. Common Python scalar types and example values.
4. Lists, tuples, dictionaries, and sets encode different structural promises.
5. Conditional logic and iteration turn a static script into a decision process.
6. A well-designed function validates inputs, performs one clear task, and returns a documented result.
7. Recommended project layout using relative paths rather than personal working directories.
8. A local benchmark comparing element-wise squaring with a Python loop and a NumPy operation.
9. A DataFrame organizes observations by rows and variables by labelled columns.
10. A missing-value map helps reveal whether gaps are isolated, clustered, or systematic.
11. Joining two tables requires explicit keys and duplicate-key diagnostics.
12. Four common chart forms and the analytical question each answers.
13. Kernel-density estimates for normal and heavy-tailed samples.
14. Distribution of 2,500 sample means, each based on 40 observations.
15. A fitted linear relationship with simulated business data.
16. Residual plots reveal nonlinearity, changing variance, and distributional departures.
17. A resilient data-retrieval pipeline validates and caches official observations.
18. A price-and-volume panel generated from a deterministic financial simulation.
19. Simulated returns display volatility clustering and heavier tails than a normal benchmark.
20. Autocorrelation and partial autocorrelation summarize lag dependence.
21. A stationary process fluctuates around a stable distribution; a random walk carries shocks forward.
22. A model forecast is a distribution, not merely a line.
23. Differencing removes the accumulated component of a random walk.
24. A monthly series decomposed into observed, trend, seasonal, and irregular components.
25. A SARIMA forecast for a synthetic monthly series with yearly seasonality.
26. A custom educational Gaussian GARCH(1,1) fit to simulated heavy-tailed returns.
27. Returns may have little linear autocorrelation while squared returns remain serially dependent.
28. News-impact curves for symmetric and asymmetric volatility models.
29. Rolling, exponentially weighted, and realized-volatility proxies for the same return series.

30. Close-to-close, Parkinson, and Garman-Klass volatility estimates from simulated OHLC prices.
31. A simulated two-regime threshold autoregressive process.
32. A two-regime Markov model separates periods with different mean and variance patterns.
33. Ordered models map a latent continuous score into ranked observed categories.
34. Predicted category probabilities from a fitted ordered-probit model.
35. Six simulated standard Brownian-motion paths.
36. Forty one-year geometric Brownian-motion price paths.
37. European call and put payoffs at a strike of 100.
38. Black-Scholes European call prices across stock prices and volatilities.
39. Historical 99% Value at Risk and Expected Shortfall for simulated portfolio losses.
40. A rolling historical VaR backtest with exceedances marked as exceptions.
41. Mean excess above candidate thresholds for a heavy-tailed sample.
42. A synthetic lead-lag relationship peaks near a lag of two periods.
43. A bivariate VAR forecast for U.S. GDP growth and changes in unemployment.
44. Illustrative response of GDP growth to the second VAR innovation.
45. Simulated cointegrated prices and their mean-reverting linear combination.
46. A cointegration-spread z-score with illustrative entry thresholds.
47. A simplified relationship among AI, machine learning, deep learning, and generative AI.
48. An honest predictive workflow isolates the final test set until model choices are complete.
49. Iterative parameter updates on a simple quadratic loss surface.
50. Area and listed price in the Tehran housing dataset from the IntroAI course archive.
51. The linear baseline leaves large, asymmetric errors, motivating feature engineering and robust evaluation.
52. Survival proportions by sex and passenger class in the IntroAI Titanic dataset.
53. A depth-three decision tree for the Titanic classification problem.
54. Out-of-sample confusion matrix for a logistic-regression pipeline.
55. Precision-recall performance for the Titanic logistic-regression pipeline.
56. A feed-forward neural network transforms inputs through learned weighted connections.
57. Training and cross-validation learning curves for a small neural classifier.
58. A small convolution kernel highlights vertical edges in an image.
59. Different convolution filters extract different local image structures.
60. A text-classification workflow from raw language to evaluated predictions.
61. A two-dimensional projection of TF-IDF document vectors.
62. A synthetic attention-weight matrix for a five-token sentence.
63. Retrieval-augmented generation grounds a response in selected source passages.
64. A tool-using agent alternates planning, action, observation, and verification.
65. Official Bank of Canada FXUSDCAD observations from 15 April to 15 May 2024.

- 66. USD/CAD log changes and a short-window annualized volatility estimate.
- 67. Out-of-sample mean absolute error for three illustrative GDP-growth forecasts.
- 68. A simulated three-asset opportunity set generated from random portfolio weights.

List of Tables

1. Welcome to Python and Ceteris LAB: chapter reference table.
2. Installing and Running Python: chapter reference table.
3. Variables, Values, Types, Strings, and Dates: chapter reference table.
4. Collections, Indexing, and Slicing: chapter reference table.
5. Decisions, Loops, and Iteration: chapter reference table.
6. Functions, Modules, Errors, and Testing: chapter reference table.
7. Files, Paths, Projects, and Reproducibility: chapter reference table.
8. NumPy and Numerical Computing: chapter reference table.
9. pandas Fundamentals: chapter reference table.
10. Importing, Cleaning, Validating, and Exporting Data: chapter reference table.
11. Manipulating, Grouping, Reshaping, and Joining Data: chapter reference table.
12. Data Visualization with Python: chapter reference table.
13. Descriptive Statistics and Exploratory Data Analysis: chapter reference table.
14. Probability, Simulation, and Statistical Inference: chapter reference table.
15. Regression and Econometrics with Python: chapter reference table.
16. Obtaining Economic and Financial Data: chapter reference table.
17. Introduction to Time-Series Data: chapter reference table.
18. Dependence, Stationarity, and White Noise: chapter reference table.
19. AR, MA, ARMA, ARIMA, and Forecasting: chapter reference table.
20. Unit Roots, Seasonality, and Dynamic Regression: chapter reference table.
21. ARCH and GARCH Models: chapter reference table.
22. Asymmetric, Advanced, and Realized Volatility: chapter reference table.
23. Nonlinear Models, Regimes, Market Microstructure, and Ordered Outcomes: chapter reference table.
24. Stochastic Processes and Option Pricing: chapter reference table.
25. Financial Risk Management: chapter reference table.
26. Multiple Time Series: chapter reference table.
27. What Artificial Intelligence and Machine Learning Actually Do: chapter reference table.
28. The Machine-Learning Workflow and Gradient Descent: chapter reference table.
29. Applied Regression and Classification Projects: chapter reference table.
30. Neural Networks and Deep Learning: chapter reference table.
31. Computer Vision: chapter reference table.
32. Natural-Language Processing and Transformers: chapter reference table.
33. Language Models, Retrieval, and AI Agents: chapter reference table.

34. Capstone Projects and Student Portfolio: chapter reference table.

Table of Contents

Publication Information.....	1
Preface.....	2
How to Use This Book.....	3
Reproducibility and AI Use.....	4
List of Figures.....	5
List of Tables.....	8
Part I: Python Foundations.....	10
1 Welcome to Python and Ceteris LAB.....	23
1.1 Learning objectives.....	23
1.2 A language and an ecosystem.....	23
1.3 From question to evidence.....	23
1.4 What Python cannot decide for you.....	24
1.5 Python demonstrations.....	24
1.8 Guided practice.....	26
1.9 Exercises.....	26
1.10 Chapter summary.....	26
2 Installing and Running Python.....	27
2.1 Learning objectives.....	27
2.2 Choose a path that matches the learner.....	27
2.3 Virtual environments are small fences.....	27
2.4 Troubleshooting by layers.....	27
2.5 Python demonstrations.....	28
2.8 Guided practice.....	30
2.9 Exercises.....	30
2.10 Chapter summary.....	30
3 Variables, Values, Types, Strings, and Dates.....	31
3.1 Learning objectives.....	31
3.2 Names point to objects.....	31

3.3 Types are analytical constraints.....	31
3.4 Dates carry frequency and meaning.....	31
3.5 Python demonstrations.....	32
3.8 Guided practice.....	34
3.9 Exercises.....	34
3.10 Chapter summary.....	34
4 Collections, Indexing, and Slicing.....	35
4.1 Learning objectives.....	35
4.2 Four structures, four promises.....	35
4.3 Indexing is positional, slicing is a window.....	35
4.4 Comprehensions compress a transformation.....	35
4.5 Python demonstrations.....	36
4.8 Guided practice.....	38
4.9 Exercises.....	38
4.10 Chapter summary.....	38
5 Decisions, Loops, and Iteration.....	39
5.1 Learning objectives.....	39
5.2 Branches encode explicit rules.....	39
5.3 Loops reveal sequence.....	39
5.4 Vectorization is not a moral command.....	39
5.5 Python demonstrations.....	40
5.8 Guided practice.....	42
5.9 Exercises.....	42
5.10 Chapter summary.....	42
6 Functions, Modules, Errors, and Testing.....	43
6.1 Learning objectives.....	43
6.2 A function is a contract.....	43
6.3 Errors should explain the violated assumption.....	43
6.4 Tests protect meaning.....	43

6.5 Python demonstrations.....	44
6.8 Guided practice.....	45
6.9 Exercises.....	46
6.10 Chapter summary.....	46
7 Files, Paths, Projects, and Reproducibility.....	47
7.1 Learning objectives.....	47
7.2 A project is a map, not a pile.....	47
7.3 Relative paths travel.....	47
7.4 Reproducibility includes provenance.....	47
7.5 Python demonstrations.....	48
7.8 Guided practice.....	50
7.9 Exercises.....	50
7.10 Chapter summary.....	50
8 NumPy and Numerical Computing.....	51
8.1 Learning objectives.....	51
8.2 Arrays are homogeneous numerical structures.....	51
8.3 Vectorization states the calculation directly.....	51
8.4 Randomness should be local and reproducible.....	51
8.5 Core equations.....	52
8.6 Python demonstrations.....	52
8.9 Guided practice.....	54
8.10 Exercises.....	54
8.11 Chapter summary.....	54
Part II: Data Analysis, Statistics, and Econometrics.....	55
9 pandas Fundamentals.....	56
9.1 Learning objectives.....	56
9.2 Labels are part of the data model.....	56
9.3 Inspection comes before transformation.....	56
9.4 Selection should reveal intent.....	56

9.5 Python demonstrations.....	57
9.8 Guided practice.....	58
9.9 Exercises.....	59
9.10 Chapter summary.....	59
10 Importing, Cleaning, Validating, and Exporting Data.....	60
10.1 Learning objectives.....	60
10.2 Import is an inference step.....	60
10.3 Cleaning is a sequence of justified decisions.....	60
10.4 A schema makes assumptions executable.....	60
10.5 Python demonstrations.....	61
10.8 Guided practice.....	63
10.9 Exercises.....	63
10.10 Chapter summary.....	63
11 Manipulating, Grouping, Reshaping, and Joining Data.....	64
11.1 Learning objectives.....	64
11.2 Grouping changes the unit of analysis.....	64
11.3 Wide and long are views of the same logic.....	64
11.4 Joins are models of relationships.....	64
11.5 Python demonstrations.....	65
11.8 Guided practice.....	66
11.9 Exercises.....	67
11.10 Chapter summary.....	67
12 Data Visualization with Python.....	68
12.1 Learning objectives.....	68
12.2 A chart is an argument with visual grammar.....	68
12.3 Labels protect meaning.....	68
12.4 Accessibility is part of accuracy.....	68
12.5 Python demonstrations.....	69
12.8 Guided practice.....	71

12.9 Exercises.....	71
12.10 Chapter summary.....	71
13 Descriptive Statistics and Exploratory Data Analysis.....	72
13.1 Learning objectives.....	72
13.2 Location and spread answer different questions.....	72
13.3 Shape matters in finance.....	72
13.4 Dependence is not a single coefficient.....	72
13.5 Core equations.....	73
13.6 Python demonstrations.....	73
13.9 Guided practice.....	75
13.10 Exercises.....	75
13.11 Chapter summary.....	75
14 Probability, Simulation, and Statistical Inference.....	77
14.1 Learning objectives.....	77
14.2 Probability describes a model of uncertainty.....	77
14.3 Simulation turns assumptions into observable consequences.....	77
14.4 Inference concerns repeated-sample behaviour.....	77
14.5 Core equations.....	78
14.6 Python demonstrations.....	78
14.9 Guided practice.....	80
14.10 Exercises.....	80
14.11 Chapter summary.....	80
15 Regression and Econometrics with Python.....	81
15.1 Learning objectives.....	81
15.2 Regression is a conditional comparison.....	81
15.3 Uncertainty depends on the error structure.....	81
15.4 Prediction and inference optimize different questions.....	81
15.5 Core equations.....	82
15.6 Python demonstrations.....	82

15.9 Guided practice.....	84
15.10 Exercises.....	84
15.11 Chapter summary.....	84
16 Obtaining Economic and Financial Data.....	85
16.1 Learning objectives.....	85
16.2 Prefer authoritative providers.....	85
16.3 A live call is not a reproducibility plan.....	85
16.4 Validate response semantics.....	85
16.5 Python demonstrations.....	86
16.8 Guided practice.....	88
16.9 Exercises.....	88
16.10 Chapter summary.....	88
Part III: Time Series and Financial Econometrics.....	89
17 Introduction to Time-Series Data.....	89
17.1 Learning objectives.....	166
17.2 Time order is part of the model.....	166
17.3 Levels and changes answer different questions.....	166
17.4 Aggregation requires a stock-flow decision.....	166
17.5 Core equations.....	166
17.6 Python demonstrations.....	167
17.9 Guided practice.....	169
17.10 Exercises.....	169
17.11 Chapter summary.....	169
18 Dependence, Stationarity, and White Noise.....	170
18.1 Learning objectives.....	170
18.2 Stationarity stabilizes the probabilistic frame.....	170
18.3 ACF and PACF are diagnostic fingerprints.....	170
18.4 Uncorrelated does not mean independent.....	170
18.5 Core equations.....	171

18.6 Python demonstrations.....	171
18.9 Guided practice.....	173
18.10 Exercises.....	173
18.11 Chapter summary.....	173
19 AR, MA, ARMA, ARIMA, and Forecasting.....	174
19.1 Learning objectives.....	174
19.2 Autoregression describes persistence.....	174
19.3 Moving averages describe shock memory.....	174
19.4 Forecasting is an evaluation design.....	174
19.5 Core equations.....	175
19.6 Python demonstrations.....	175
19.9 Guided practice.....	177
19.10 Exercises.....	177
19.11 Chapter summary.....	177
20 Unit Roots, Seasonality, and Dynamic Regression.....	178
20.1 Learning objectives.....	178
20.2 Differencing removes a stochastic trend, not every trend.....	178
20.3 Seasonality is dependence at a calendar rhythm.....	178
20.4 Regression errors can remember the past.....	178
20.5 Core equations.....	179
20.6 Python demonstrations.....	179
20.9 Guided practice.....	181
20.10 Exercises.....	182
20.11 Chapter summary.....	182
21 ARCH and GARCH Models.....	183
21.1 Learning objectives.....	183
21.2 Volatility is latent and conditional.....	183
21.3 GARCH compresses a long memory.....	183
21.4 Model checking moves to standardized residuals.....	183

21.5 Core equations.....	184
21.6 Python demonstrations.....	184
21.9 Guided practice.....	186
21.10 Exercises.....	186
21.11 Chapter summary.....	186
22 Asymmetric, Advanced, and Realized Volatility.....	187
22.1 Learning objectives.....	187
22.2 Asymmetry belongs in the variance equation.....	187
22.3 Realized measures use more of the day.....	187
22.4 OHLC estimators trade assumptions for efficiency.....	187
22.5 Core equations.....	188
22.6 Python demonstrations.....	188
22.9 Guided practice.....	190
22.10 Exercises.....	191
22.11 Chapter summary.....	191
23 Nonlinear Models, Regimes, Market Microstructure, and Ordered Outcomes.....	192
23.1 Learning objectives.....	192
23.2 Threshold models make dynamics piecewise.....	192
23.3 Markov switching treats the regime as latent.....	192
23.4 Market data are generated by trading mechanisms.....	192
23.5 Core equations.....	193
23.6 Python demonstrations.....	193
23.9 Guided practice.....	196
23.10 Exercises.....	196
23.11 Chapter summary.....	196
24 Stochastic Processes and Option Pricing.....	197
24.1 Learning objectives.....	197
24.2 Brownian motion accumulates random increments.....	197
24.3 Ito's lemma adds a variance correction.....	197

24.4 Black-Scholes is a model, not a market law.....	197
24.5 Core equations.....	198
24.6 Python demonstrations.....	198
24.9 Guided practice.....	201
24.10 Exercises.....	201
24.11 Chapter summary.....	201
25 Financial Risk Management.....	203
25.1 Learning objectives.....	203
25.2 The sign convention comes first.....	203
25.3 Expected Shortfall looks beyond the threshold.....	203
25.4 Backtesting checks frequency, not every dimension of risk.....	203
25.5 Core equations.....	204
25.6 Python demonstrations.....	204
25.9 Guided practice.....	206
25.10 Exercises.....	207
25.11 Chapter summary.....	207
26 Multiple Time Series.....	208
26.1 Learning objectives.....	208
26.2 A vector model uses shared information.....	208
26.3 Predictive ordering is not structural causation.....	208
26.4 Cointegration separates short and long horizons.....	208
26.5 Core equations.....	209
26.6 Python demonstrations.....	209
26.9 Guided practice.....	212
26.10 Exercises.....	213
26.11 Chapter summary.....	213
Part IV: Machine Learning and Artificial Intelligence.....	214
27 What Artificial Intelligence and Machine Learning Actually Do.....	215
27.1 Learning objectives.....	215

27.2 AI is an umbrella, not a single mechanism.....	215
27.3 Learning means optimizing an objective.....	215
27.4 Fluency is not verification.....	215
27.5 Python demonstrations.....	216
27.8 Guided practice.....	218
27.9 Exercises.....	218
27.10 Chapter summary.....	218
28 The Machine-Learning Workflow and Gradient Descent.....	219
28.1 Learning objectives.....	219
28.2 The split defines the claim.....	219
28.3 Overfitting is a gap between memory and generalization.....	219
28.4 Gradient descent follows local slope information.....	219
28.5 Core equations.....	220
28.6 Python demonstrations.....	220
28.9 Guided practice.....	222
28.10 Exercises.....	222
28.11 Chapter summary.....	222
29 Applied Regression and Classification Projects.....	223
29.1 Learning objectives.....	223
29.2 Project A: Tehran house prices.....	223
29.3 Project B: Titanic classification.....	223
29.4 Baselines make improvement meaningful.....	223
29.5 Core equations.....	224
29.6 Python demonstrations.....	224
29.9 Guided practice.....	228
29.10 Exercises.....	228
29.11 Chapter summary.....	228
30 Neural Networks and Deep Learning.....	230
30.1 Learning objectives.....	230

30.2 A neural network composes transformations.....	230
30.3 Backpropagation is organized chain-rule accounting.....	230
30.4 Learning curves reveal optimization and generalization.....	230
30.5 Core equations.....	231
30.6 Python demonstrations.....	231
30.9 Guided practice.....	233
30.10 Exercises.....	233
30.11 Chapter summary.....	233
31 Computer Vision.....	235
31.1 Learning objectives.....	235
31.2 Images are structured tensors.....	235
31.3 Convolution searches locally with shared weights.....	235
31.4 Transfer learning is borrowed representation, not borrowed truth.....	235
31.5 Core equations.....	235
31.6 Python demonstrations.....	236
31.9 Guided practice.....	238
31.10 Exercises.....	238
31.11 Chapter summary.....	238
32 Natural-Language Processing and Transformers.....	239
32.1 Learning objectives.....	239
32.2 Tokenization chooses the model's alphabet.....	239
32.3 Representations move from counts to geometry.....	239
32.4 Attention connects tokens conditionally.....	239
32.5 Core equations.....	240
32.6 Python demonstrations.....	240
32.9 Guided practice.....	243
32.10 Exercises.....	243
32.11 Chapter summary.....	243
33 Language Models, Retrieval, and AI Agents.....	244

33.1 Learning objectives.....	244
33.2 A language model predicts continuations.....	244
33.3 Retrieval creates an evidence channel.....	244
33.4 Agents are loops with authority.....	244
33.5 Python demonstrations.....	245
33.8 Guided practice.....	247
33.9 Exercises.....	247
33.10 Chapter summary.....	247
34 Capstone Projects and Student Portfolio.....	249
34.1 Learning objectives.....	249
34.2 Capstone A: Canadian exchange-rate forecasting.....	249
34.3 Capstone B: inflation and interest-rate dashboard.....	249
34.4 Capstones C and D: risk and AI.....	249
34.5 Python demonstrations.....	250
34.8 Guided practice.....	253
34.9 Exercises.....	253
34.10 Chapter summary.....	253
35 Appendix A. Python Quick Reference.....	255
36 Appendix B. Installation and Troubleshooting.....	Error: Reference source not found
37 Appendix C. R-to-Python Crosswalk.....	257
38 Appendix D. Mathematical Notation.....	259
39 Appendix E. Glossary.....	Error: Reference source not found
40 Appendix F. Selected Exercise Guidance.....	263
41 Appendix G. Data Sources, Licences, and Attribution.....	268
42 Appendix H. Software and Reproducibility.....	269
43 Appendix I. Accessibility Statement.....	270
44 Appendix J. Acknowledgments.....	271
References.....	272



PART I

Python Foundations

LEARN • ANALYZE • UNDERSTAND

Ceteris LAB | econometrics.safavim.com

1 Welcome to Python and Ceteris LAB

Opening question: **How can one language carry an economic question from raw observations to a reproducible conclusion?**

1.1 Learning objectives

- Explain why Python is useful in economics, finance, econometrics, and AI.
- Distinguish scripts, notebooks, packages, and environments.
- Run a small end-to-end analysis.
- Recognize the difference between computation and interpretation.

Prerequisites: No programming experience is required.

Key terms: Python, open source, notebook, script, package, reproducibility.

1.2 A language and an ecosystem

Python is both a general-purpose programming language and the centre of a large scientific ecosystem. The language supplies readable syntax, functions, objects, and modules. Libraries such as NumPy, pandas, Matplotlib, statsmodels, scikit-learn, and PyTorch add numerical computing, data management, visualization, econometrics, machine learning, and deep learning. The result is a connected workshop rather than a single-purpose calculator. A student can clean a Statistics Canada table, estimate a regression, visualize uncertainty, and document the workflow without changing languages. Current Python documentation remains the authority for syntax and standard-library behaviour (Python Software Foundation 2026).

1.3 From question to evidence

A useful analysis begins with a question, not with a package. The Ceteris LAB workflow used throughout this book is: define the question, identify data, validate inputs, transform variables, estimate or simulate, diagnose the result, communicate the evidence, and preserve enough metadata for another person to repeat the work. Reproducibility does not mean that every future run will be numerically identical. It means that the code, data provenance, assumptions, software versions, and random seeds are visible rather than hidden in a maze of clicks.

1.4 What Python cannot decide for you

Software can calculate quickly and still answer the wrong question. Python does not decide whether a variable measures the intended concept, whether a relationship is causal, whether a forecast is useful, or whether a financial result is suitable for a real decision. Those judgements remain analytical tasks. Throughout the book, each result is followed by interpretation and limitation. This separation is especially important when AI tools generate code: fluent syntax is not evidence that a model is correctly specified.

1.5 Python demonstrations

1.5.1 Demonstration 1.1: A first miniature analysis

```
import pandas as pd

inflation = pd.DataFrame({
    "month": ["Jan", "Feb", "Mar", "Apr"],
    "rate": [2.9, 2.8, 2.7, 2.6],
})

print(inflation["rate"].mean())
print(inflation.tail(2))
```

Verified output

```
2.7499999999999996
  month rate
2  Mar  2.7
3  Apr  2.6
```

Interpretation. The mean summarizes the four observations, while the final rows preserve the temporal detail. A statistic and the underlying data answer different questions.

1.5.2 Demonstration 1.2: A named calculation

```
principal = 1_000
annual_rate = 0.045
years = 3
future_value = principal * (1 + annual_rate) ** years
print(f'Future value: ${future_value:,.2f}')
```

Verified output

```
Future value: $1,141.17
```

Interpretation. Clear names make the calculation readable as a small model rather than a string of unexplained numbers.

1.6 Visual evidence

A practical Python ecosystem for economic and financial analysis

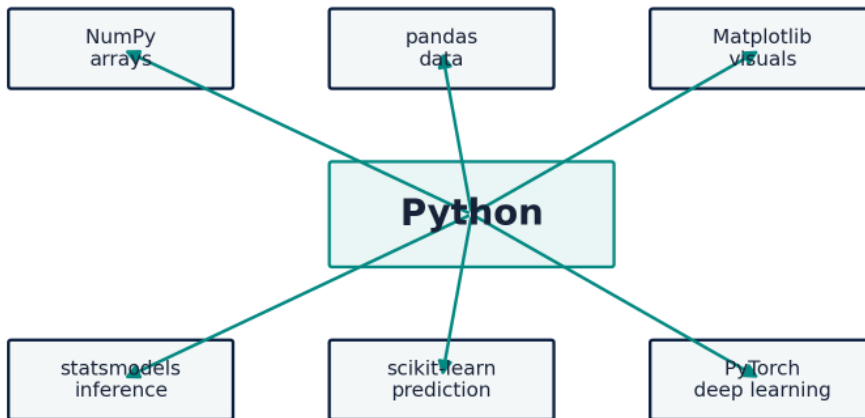
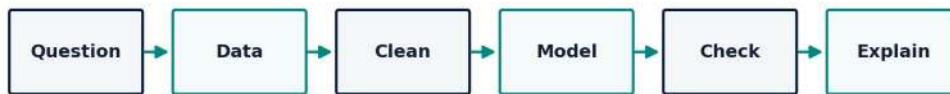


Figure 1. Python connects numerical computing, data management, econometrics, visualization, and artificial intelligence.



Reproducibility means another learner can rerun the path.

Figure 2. The reproducible analysis loop used throughout the book.

1.7 Reference table

Table 1. Chapter reference.

Tool	Primary role	Typical classroom use
Python language	Logic and reusable instructions	Functions and simulations
Jupyter notebook	Narrative plus executable cells	Labs and demonstrations
Package	Specialized functionality	pandas, statsmodels, scikit-learn
Environment	Controlled package versions	Rebuilding the book

Why This Matters

Python makes the complete chain of evidence visible. That visibility is the foundation of auditability, collaboration, and careful learning.

Common Mistake

Treating a successful run as proof that the analysis is correct. Code execution checks syntax and computation, not research design.

Ceteris LAB Tip

Before importing a package, write the question in one sentence and name the output that would answer it.

R-to-Python / Source Bridge

The opening financial time-series lecture emphasized obtaining, processing, and interpreting data. This chapter preserves that objective while replacing the 2013 R-first workflow with a modern Python-first path (Tsay 2013).

1.8 Guided practice

1. Re-run Demonstration 1.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

1.9 Exercises

1. Change the four inflation rates and recompute the mean.
2. Add a fifth month and calculate the minimum and maximum.
3. Write one sentence explaining why the mean alone may hide important variation.
4. Create a compound-growth example using your own values.

1.10 Chapter summary

- Python combines a readable language with specialized scientific libraries.
- A reproducible workflow connects question, data, computation, interpretation, and documentation.
- Analytical judgement cannot be delegated to software.

1.11 Key Python commands

```
import pandas as pd
inflation = pd.DataFrame({
print(inflation["rate"].mean())
print(inflation.tail(2))
future_value = principal * (1 + annual_rate) ** years
print(f"Future value: ${future_value:,.2f}")
```

2 Installing and Running Python

Opening question: What is the least fragile way to move from “I have a computer” to a working, reproducible Python environment?

2.1 Learning objectives

- Choose between local Python, JupyterLab, VS Code, and Google Colab.
- Create and activate a virtual environment.
- Install packages once in a controlled setup.
- Diagnose common kernel and path problems.

Prerequisites: Chapter 1.

Key terms: interpreter, virtual environment, kernel, package manager, terminal, Colab.

2.2 Choose a path that matches the learner

Students need more than one entry door. Google Colab is the lowest-friction route when installation rights are limited. A local environment is preferable when students need files, privacy, larger datasets, or repeatable package versions. JupyterLab supports exploratory notebooks; VS Code combines notebooks, scripts, testing, and version control. The book is written so that core examples run in either Colab or a local environment. As of August 2026, Python 3.14 is the current feature series, but the reproducibility files record the exact tested environment rather than assuming that “latest” always means “compatible” (Python Software Foundation 2026).

2.3 Virtual environments are small fences

A virtual environment gives one project its own package directory. This prevents a course update from silently breaking another project and makes it possible to state which versions produced the published results. The environment does not isolate the operating system or guarantee security; it simply separates Python dependencies. The recommended sequence is create, activate, install from a requirements file, register the environment as a Jupyter kernel, and test a short import script.

2.4 Troubleshooting by layers

Installation problems become easier when separated into layers. First verify that the terminal can locate Python. Next verify that the intended environment is active. Then confirm that Jupyter is using the same interpreter. Finally import packages one at a time. A common

problem is installing pandas into one Python and launching a notebook with another. Printing `sys.executable` reveals the interpreter behind the current kernel and often turns a foggy problem into a precise path mismatch.

2.5 Python demonstrations

2.5.1 Demonstration 2.1: Inspect the running interpreter

```
import platform
import sys

print(sys.version.split()[0])
print(sys.executable)
print(platform.system())
```

Verified output

```
3.13.5
/opt/pyvenv/bin/python
Linux
```

Interpretation. Your path will differ. The important point is that the notebook and the environment should refer to the same interpreter.

2.5.2 Demonstration 2.2: Create a cross-platform project path

```
from pathlib import Path

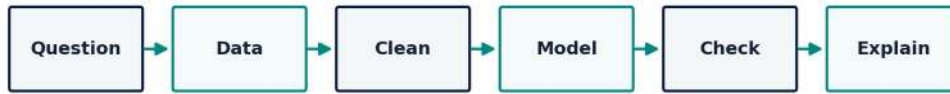
project = Path.cwd()
data_dir = project / "data" / "frozen"
print(data_dir.relative_to(project).as_posix())
print(data_dir.exists())
```

Verified output

```
data/frozen
True
```

Interpretation. `pathlib` joins folders without hard-coded slashes and makes the intended location explicit.

2.6 Visual evidence



Reproducibility means another learner can rerun the path.

Figure 2. The reproducible analysis loop used throughout the book.

A reproducible project separates inputs, transformations, and outputs

folder project/

- folder data/
 - folder raw/
 - folder processed/
- folder notebooks/
- folder scripts/
- file README.md

Figure 7. Recommended project layout using relative paths rather than personal working directories.

2.7 Reference table

Table 2. Chapter reference.

Symptom	Likely cause	First check
ModuleNotFoundError	Package absent from active environment	Print sys.executable
Notebook uses old package	Wrong kernel selected	Kernel menu and executable path
Command not found	Python not on PATH	python -version or py -version
Permission error	Restricted installation location	Use venv or Colab

Why This Matters

A clean environment converts “it worked on my laptop” into a documented computational setting that another learner can reconstruct.

Common Mistake

Running `pip install` inside random notebook cells. It creates hidden differences between runs and makes the notebook dependent on cell order.

Ceteris LAB Tip

Keep installation commands in one setup file and analysis commands in notebooks or scripts.

R-to-Python / Source Bridge

The source notes taught graphical installation of R packages and manual working-directory changes. The modernized workflow centralizes dependencies and uses project-relative paths rather than personal directories (Tsay 2013).

2.8 Guided practice

1. Re-run Demonstration 2.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

2.9 Exercises

1. Identify which Python interpreter your notebook uses.
2. Create a new virtual environment and install NumPy and pandas.
3. Explain the difference between an environment and a Jupyter kernel.
4. Write a two-step troubleshooting checklist for `ModuleNotFoundError`.

2.10 Chapter summary

- Colab provides a no-install route; local environments provide more control.
- Virtual environments separate project dependencies.
- Interpreter and kernel mismatches are a frequent source of confusion.

2.11 Key Python commands

```
import platform
import sys
print(sys.version.split()[0])
print(sys.executable)
print(platform.system())
from pathlib import Path
```

3 Variables, Values, Types, Strings, and Dates

Opening question: How does Python know whether a value is a price, a label, a date, or a logical condition?

3.1 Learning objectives

- Create variables with meaningful names.
- Work with integers, floats, strings, booleans, and None.
- Format text and numeric results.
- Parse and compare dates safely.

Prerequisites: Chapter 2.

Key terms: variable, object, type, boolean, None, datetime.

3.2 Names point to objects

Assignment binds a name to an object. The statement `rate = 0.045` does not put a value into a permanent box called `rate`; it creates a floating-point object and lets the name refer to it. This matters when objects are mutable, but the beginner rule is simpler: choose names that carry economic meaning, avoid spaces and punctuation, and reserve uppercase names for constants only when that convention genuinely helps. Python is dynamically typed, so the value determines the type at runtime.

3.3 Types are analytical constraints

A string such as `"2.5"` looks like a number to a human but cannot be averaged until it is converted. A date stored as plain text does not automatically understand calendar order. Type conversion is therefore part of data validation. Floating-point numbers also have finite binary precision, so exact equality can be unreliable for some decimal calculations. Financial accounting may require `Decimal`; most econometric work uses floating point and numerical tolerances.

3.4 Dates carry frequency and meaning

Calendar data require explicit parsing. The same text can be interpreted differently across countries, and a monthly observation may represent an average, a period end, or a reference month. Python's `datetime` and `pandas` date tools support arithmetic, sorting, and resampling,

but the analyst must still document the observation convention. A timestamp is not merely a label: it determines ordering, lags, trading-day alignment, and forecast origin.

3.5 Python demonstrations

3.5.1 Demonstration 3.1: Types and formatted output

```
country = "Canada"
policy_rate = 0.0275
observations = 120
is_monthly = True

print(type(policy_rate).__name__)
print(f"{country}: {policy_rate:.2%}, n={observations}, monthly={is_monthly}")
```

Verified output

```
float
Canada: 2.75%, n=120, monthly=True
```

Interpretation. The format specification changes presentation, not the stored value.

3.5.2 Demonstration 3.2: Parse and compare dates

```
from datetime import date

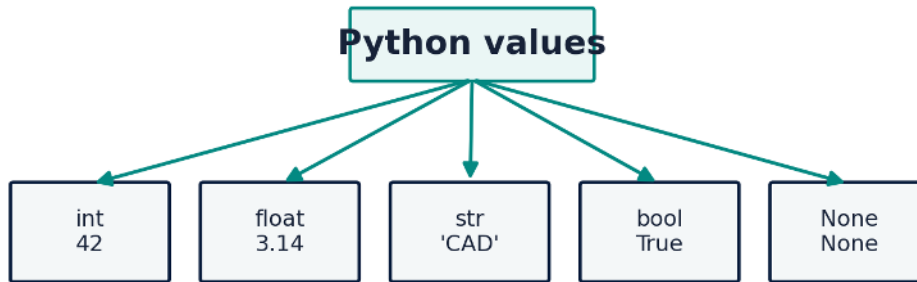
start = date.fromisoformat("2026-01-01")
end = date.fromisoformat("2026-08-15")
print((end - start).days)
print(start < end)
```

Verified output

```
226
True
```

Interpretation. ISO dates avoid day-month ambiguity and support calendar arithmetic.

3.6 Visual evidence



A variable is a name that points to a value; the value carries the type.

Figure 3. Common Python scalar types and example values.

3.7 Reference table

Table 3. Chapter reference.

Type	Example	Use
int	12	counts and discrete quantities
float	0.045	rates and measurements
str	"Canada"	labels and identifiers
bool	True	conditions and filters
None	None	intentional absence
datetime	2026-08-15	calendar operations

Why This Matters

Correct types prevent quiet errors in sorting, aggregation, filtering, and model construction.

Common Mistake

Using a numeric-looking identifier, such as a postal code, as a number. Leading zeros and categorical meaning may be lost.

Ceteris LAB Tip

Inspect `type(value)` when an operator behaves strangely. The error often belongs to the input type, not the formula.

R-to-Python / Source Bridge

The introductory source material used R assignments and raw numeric dates. This chapter preserves the idea of named objects while adding explicit type and calendar discipline.

3.8 Guided practice

1. Re-run Demonstration 3.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

3.9 Exercises

1. Create variables for a bond price, maturity year, issuer name, and default flag.
2. Convert the string "3.25" to a float and divide it by 100.
3. Format 1234567.891 as currency with commas and two decimals.
4. Calculate the days between two ISO dates.

3.10 Chapter summary

- Variables name objects; types determine valid operations.
- Formatting controls display without changing the underlying value.
- Dates should be parsed explicitly and documented.

3.11 Key Python commands

```
print(type(policy_rate).__name__)
print(f'{country}: {policy_rate:.2%}, n={observations}, monthly={is_monthly}')
from datetime import date
start = date.fromisoformat("2026-01-01")
end = date.fromisoformat("2026-08-15")
print((end - start).days)
```

4 Collections, Indexing, and Slicing

Opening question: How should related values be organized so that the structure communicates what operations are legitimate?

4.1 Learning objectives

- Use lists, tuples, dictionaries, and sets.
- Index and slice ordered collections.
- Build nested structures.
- Choose a collection based on meaning rather than habit.

Prerequisites: Chapter 3.

Key terms: list, tuple, dictionary, set, index, slice.

4.2 Four structures, four promises

A list promises order and permits change. A tuple promises order and is normally treated as fixed. A dictionary maps unique keys to values. A set stores unique elements without meaningful position. These distinctions are analytical: a portfolio of ticker weights is naturally a dictionary, a fixed coordinate may be a tuple, and a sequence of monthly returns is a list until it becomes a NumPy array or pandas Series.

4.3 Indexing is positional, slicing is a window

Python begins indexing at zero. The first element is therefore position 0, while -1 means the last element. Slices use a half-open interval: the starting position is included and the stopping position is excluded. That design makes lengths predictable, because `values[a:b]` contains `b-a` positions when the bounds are valid. The same convention appears in strings, arrays, and many pandas operations.

4.4 Comprehensions compress a transformation

List and dictionary comprehensions express a simple transformation or filter in one readable line. They are useful when the logic is short and transparent. A comprehension with several nested conditions becomes a puzzle and should be replaced by a loop or function. Readability is not cosmetic; it allows the analyst to inspect whether a data transformation matches the intended research rule.

4.5 Python demonstrations

4.5.1 Demonstration 4.1: Index and slice a return series

```
returns = [0.012, -0.004, 0.009, 0.003, -0.011]
print(returns[0])
print(returns[-1])
print(returns[1:4])
```

Verified output

```
0.012
-0.011
[-0.004, 0.009, 0.003]
```

Interpretation. The slice includes positions 1, 2, and 3, but not 4.

4.5.2 Demonstration 4.2: Map assets to weights

```
weights = {"BOND": 0.50, "EQUITY": 0.35, "CASH": 0.15}
print(sum(weights.values()))
large_positions = [asset for asset, w in weights.items() if w >= 0.30]
print(large_positions)
```

Verified output

```
1.0
['BOND', 'EQUITY']
```

Interpretation. A dictionary preserves the relationship between each asset label and its weight.

4.6 Visual evidence

Choose the collection that matches the job

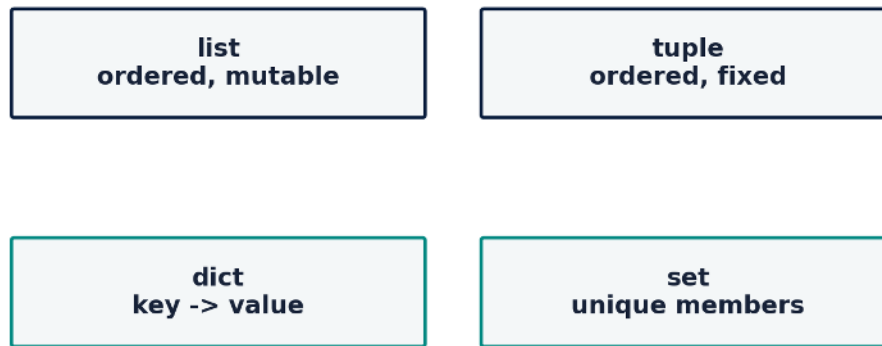


Figure 4. Lists, tuples, dictionaries, and sets encode different structural promises.

4.7 Reference table

Table 4. Chapter reference.

Structure	Ordered?	Duplicates?	Best use
list	Yes	Yes	changeable sequence
tuple	Yes	Yes	fixed record or coordinate
dict	Insertion order	Keys unique	named mapping
set	No positional meaning	No	membership and uniqueness

Why This Matters

Collection choice makes assumptions visible before the data reach a model.

Common Mistake

Using a set when order matters. A set is excellent for membership tests but cannot represent a time sequence.

Ceteris LAB Tip

Say the structure aloud: “a mapping from series name to unit” suggests a dictionary; “an ordered sequence of returns” suggests a list or array.

R-to-Python / Source Bridge

The old notes introduced vectors and matrix-like objects early. Python separates general-purpose collections from numerical arrays, which are introduced in Chapter 8.

4.8 Guided practice

1. Re-run Demonstration 4.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

4.9 Exercises

1. Create a list of five GDP growth rates and print the middle three.
2. Create a tuple containing a series ID and frequency.
3. Create a dictionary mapping three variables to units.
4. Use a set to remove duplicate province codes.

4.10 Chapter summary

- Lists, tuples, dictionaries, and sets encode different structural assumptions.
- Zero-based indexing and half-open slicing recur throughout Python.
- Comprehensions are best kept simple.

4.11 Key Python commands

```
print(returns[0])
print(returns[-1])
print(returns[1:4])
print(sum(weights.values()))
large_positions = [asset for asset, w in weights.items() if w >= 0.30]
print(large_positions)
```

5 Decisions, Loops, and Iteration

Opening question: How can a script apply a rule repeatedly while remaining easy to inspect?

5.1 Learning objectives

- Write conditional branches.
- Iterate with for and while loops.
- Use enumerate and zip.
- Recognize when vectorization is clearer.

Prerequisites: Chapters 3 and 4.

Key terms: if, for, while, iteration, enumerate, vectorization.

5.2 Branches encode explicit rules

An if statement selects a path based on a Boolean expression. In finance, a branch might classify a return as a loss, gain, or no change. In data cleaning, it might reject a negative quantity that should never be negative. Conditions should be mutually understandable, and boundary values deserve attention. A rule using `>` differs from one using `>=`; that single character may decide which risk bucket receives an observation.

5.3 Loops reveal sequence

A for loop is appropriate when the script should visit each element of a known iterable. A while loop repeats until a condition changes and therefore requires special care to avoid infinite execution. `enumerate` supplies both position and value; `zip` walks through aligned sequences. These tools are valuable for file inventories, simulation replications, and diagnostic reports.

5.4 Vectorization is not a moral command

NumPy and pandas often replace Python loops with vectorized operations that are faster and shorter. Yet a transparent loop may be preferable for teaching a recursive forecast or validating a complex rule. The right question is not “Can I remove the loop?” but “Which form makes the algorithm easiest to verify?” Performance matters after correctness and clarity are established.

5.5 Python demonstrations

5.5.1 Demonstration 5.1: Classify returns

```
returns = [0.02, -0.01, 0.0]
for r in returns:
    if r > 0:
        label = "gain"
    elif r < 0:
        label = "loss"
    else:
        label = "flat"
    print(label)
```

Verified output

```
gain
loss
flat
```

Interpretation. The zero case is handled explicitly rather than being absorbed into gain or loss.

5.5.2 Demonstration 5.2: Iterate over aligned series

```
months = ["Jan", "Feb", "Mar"]
inflation = [2.9, 2.8, 2.7]
for position, (month, rate) in enumerate(zip(months, inflation), start=1):
    print(position, month, rate)
```

Verified output

```
1 Jan 2.9
2 Feb 2.8
3 Mar 2.7
```

Interpretation. `zip` assumes the sequences align. In real data, joins based on keys are usually safer than positional alignment.

5.6 Visual evidence

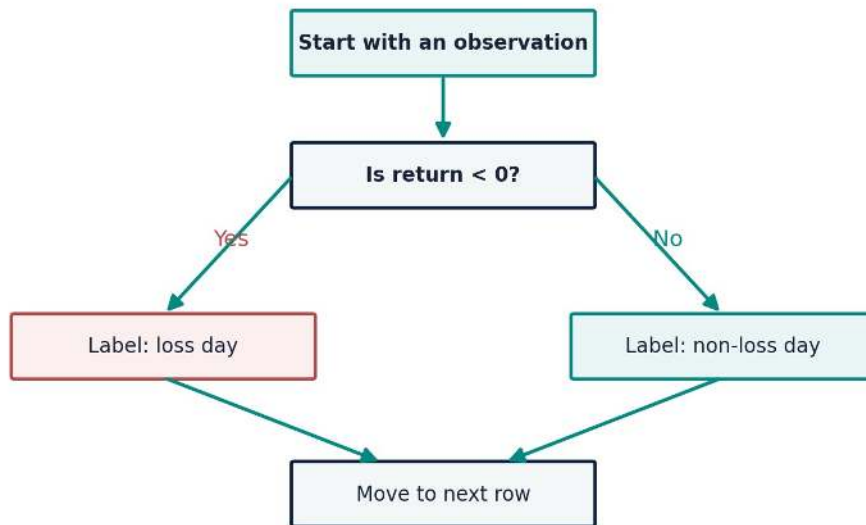


Figure 5. Conditional logic and iteration turn a static script into a decision process.

5.7 Reference table

Table 5. Chapter reference.

Pattern	Use	Risk
if/elif/else	exclusive decisions	unclear boundaries
for	known sequence	unnecessary mutation
while	condition-driven repetition	infinite loop
comprehension	short transform/filter	compressed complexity
vectorized expression	array calculation	hidden alignment assumptions

Why This Matters

Control flow turns a static formula into a transparent decision procedure.

Common Mistake

Using `while` when the stopping condition is not guaranteed to change.

Ceteris LAB Tip

Test boundary values separately: exactly zero, exactly at the cutoff, and immediately on either side.

R-to-Python / Source Bridge

Later source examples rely on recursive forecasts and volatility updates. This chapter establishes the control-flow tools needed to understand those recursions.

5.8 Guided practice

1. Re-run Demonstration 5.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

5.9 Exercises

1. Write a rule that labels inflation below 2, between 2 and 3, and above 3.
2. Use a loop to compound a value for five years.
3. Use enumerate to print observation numbers.
4. Rewrite a simple loop as a list comprehension.

5.10 Chapter summary

- Conditional branches express rules and boundaries.
- Loops repeat work over sequences or until conditions change.
- Vectorization is useful when it improves clarity and performance.

5.11 Key Python commands

```
print(label)
print(position, month, rate)
```

6 Functions, Modules, Errors, and Testing

Opening question: How can a calculation be trusted when it is reused in several chapters or projects?

6.1 Learning objectives

- Define reusable functions.
- Validate arguments and raise meaningful errors.
- Import modules without hidden state.
- Use assertions and tests for expected behaviour.

Prerequisites: Chapters 3 to 5.

Key terms: function, argument, return value, exception, module, test.

6.2 A function is a contract

A function has a name, inputs, a task, and a return value. A good function performs one coherent job and documents the units or conventions it expects. In a return calculation, for example, the analyst should state whether prices must be positive and whether the output is a decimal or percentage. Small functions reduce duplication and make it possible to test the logic independently from the surrounding notebook.

6.3 Errors should explain the violated assumption

Python exceptions are not enemies to suppress. A `ValueError` can tell the student that a maturity is negative or that a price series contains zero. Catch an exception only when the program can respond meaningfully. A bare `except` hides programming errors and may allow an invalid analysis to continue. Input validation belongs close to the function boundary, before a long calculation compounds the mistake.

6.4 Tests protect meaning

A test checks a property that should remain true when code changes. Exact expected values are useful for deterministic functions; inequalities and tolerances are better for floating-point or simulation results. Tests should cover normal cases, boundaries, and invalid inputs. The objective is not to prove a program perfect but to make its key assumptions executable and visible.

6.5 Python demonstrations

6.5.1 Demonstration 6.1: A validated return function

```
def simple_return(start_price: float, end_price: float) -> float:
    """Return the decimal holding-period return."""
    if start_price <= 0:
        raise ValueError("start_price must be positive")
    return end_price / start_price - 1

print(simple_return(100, 105))
```

Verified output

```
0.0500000000000000044
```

Interpretation. The tiny binary representation difference is normal. Formatting or `math.isclose` is appropriate when comparing floating-point values.

6.5.2 Demonstration 6.2: A lightweight test

```
import math

assert math.isclose(simple_return(100, 105), 0.05)
try:
    simple_return(0, 105)
except ValueError as err:
    print(err)
```

Verified output

```
start_price must be positive
```

Interpretation. The first assertion protects the formula; the second check demonstrates that an invalid denominator is rejected.

6.6 Visual evidence

Functions create testable, reusable analytical building blocks



Figure 6. A well-designed function validates inputs, performs one clear task, and returns a documented result.

6.7 Reference table

Table 6. Chapter reference.

Test type	Question	Example
unit test	Does one function behave correctly?	simple_return(100, 105)
boundary test	What happens at an edge?	zero years
error test	Is invalid input rejected?	negative price
smoke test	Does the workflow run?	build all figures

Why This Matters

Functions and tests convert isolated notebook cells into reusable analytical components.

Common Mistake

Catching every exception and continuing. The notebook may finish while the analysis silently loses observations or substitutes bad values.

Ceteris LAB Tip

Write the test before optimizing the function. A faster wrong answer remains wrong at higher speed.

R-to-Python / Source Bridge

Several original lecture examples depended on external custom R scripts. This book replaces opaque dependencies with documented Python functions and validation tests.

6.8 Guided practice

1. Re-run Demonstration 6.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.

3. Add one validation check that would prevent a plausible error.

6.9 Exercises

1. Write a function that annualizes monthly volatility.
2. Raise an error when frequency is not positive.
3. Test the function at a known value.
4. Move two related functions into a module and import them.

6.10 Chapter summary

- Functions define reusable contracts.
- Meaningful exceptions expose violated assumptions.
- Tests make key expectations executable.

6.11 Key Python commands

```
def simple_return(start_price: float, end_price: float) -> float:
    raise ValueError("start_price must be positive")
print(simple_return(100, 105))
import math
assert math.isclose(simple_return(100, 105), 0.05)
simple_return(0, 105)
```

7 Files, Paths, Projects, and Reproducibility

Opening question: How can a project find its data tomorrow, on another computer, and after being uploaded to a website?

7.1 Learning objectives

- Build a portable project tree.
- Use pathlib and relative paths.
- Separate raw, frozen, processed, and generated files.
- Record versions, sources, and random seeds.

Prerequisites: Chapter 6.

Key terms: pathlib, relative path, project root, metadata, version control, seed.

7.2 A project is a map, not a pile

A reproducible project separates source data, processed data, scripts, notebooks, figures, tables, and documentation. The directory names act as a map for both humans and code. Raw or frozen data should not be overwritten by a cleaning script. Generated figures should be reproducible from code. Temporary caches and credentials should not be included in a public package.

7.3 Relative paths travel

A hard-coded path such as `C:/Users/name/Desktop/data.csv` describes one computer. A relative path such as `data/frozen/series.csv` describes the file's role within the project. `pathlib.Path` provides cross-platform path construction and file operations. The project root should be established once rather than changed repeatedly with working-directory commands.

7.4 Reproducibility includes provenance

Saving code is not enough. The analyst should record where each dataset came from, the retrieval date, the series identifier, the transformations applied, and whether redistribution is permitted. Environment files record package versions. Random seeds stabilize demonstrations. Git records the evolution of text and code, although large or restricted data may require a separate governance plan.

7.5 Python demonstrations

7.5.1 Demonstration 7.1: Build portable paths

```
from pathlib import Path

root = Path.cwd()
figure_path = root / "figures" / "example.png"
print(figure_path.suffix)
print(figure_path.parent.name)
```

Verified output

```
.png
figures
```

Interpretation. The path expresses the file type and project role without assuming an operating system.

7.5.2 Demonstration 7.2: Write metadata beside data

```
import json
from pathlib import Path

metadata = {
    "provider": "Bank of Canada",
    "series": "FXUSDCAD",
    "retrieved": "2026-08-15",
}
path = Path("data/metadata/example.json")
path.parent.mkdir(parents=True, exist_ok=True)
path.write_text(json.dumps(metadata, indent=2), encoding="utf-8")
print(path.exists())
```

Verified output

```
True
```

Interpretation. Metadata becomes a first-class project artifact rather than a memory held by one analyst.

7.6 Visual evidence

A reproducible project separates inputs, transformations, and outputs

```

folder project/
├── folder data/
│   ├── folder raw/
│   └── folder processed/
├── folder notebooks/
├── folder scripts/
└── file README.md
  
```

Figure 7. Recommended project layout using relative paths rather than personal working directories.

7.7 Reference table

Table 7. Chapter reference.

Folder	Purpose	Rule
data/raw	untouched acquisition	never edit manually
data/frozen	redistributable snapshot	record source and date
data/processed	analysis-ready outputs	rebuild from code
scripts	reusable programs	no credentials
notebooks	guided analysis	run top to bottom
figures/tables	generated products	do not hand-edit

Why This Matters

A portable project can be rebuilt, inspected, and taught without reconstructing someone else's desktop.

Common Mistake

Changing the working directory inside multiple notebook cells. The result depends on execution order and becomes difficult to diagnose.

Ceteris LAB Tip

Design the folder tree before downloading data. A clear destination discourages accidental mixing of raw and processed files.

R-to-Python / Source Bridge

The source notes recommended `setwd()` and personal directories. The new workflow eliminates that brittle dependency and underpins every notebook in the public package.

7.8 Guided practice

1. Re-run Demonstration 7.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

7.9 Exercises

1. Create the project tree shown in the table.
2. Write a JSON metadata file for a dataset.
3. Explain why frozen and processed data are different.
4. Initialize a Git repository and identify files that should be ignored.

7.10 Chapter summary

- Project structure is part of the analytical method.
- Relative paths make workflows portable.
- Provenance and environment records are necessary for reproducibility.

7.11 Key Python commands

```
from pathlib import Path
root = Path.cwd()
print(figure_path.suffix)
print(figure_path.parent.name)
import json
from pathlib import Path
```

8 NumPy and Numerical Computing

Opening question: Why are numerical arrays faster and more expressive than repeatedly updating Python lists?

8.1 Learning objectives

- Create NumPy arrays.
- Understand shape, dtype, broadcasting, and vectorization.
- Generate reproducible random samples.
- Perform matrix and linear-algebra operations.

Prerequisites: Chapters 3 to 7.

Key terms: ndarray, shape, dtype, vectorization, broadcasting, random generator.

8.2 Arrays are homogeneous numerical structures

A NumPy array stores values of a common data type in a regular shape. That constraint enables compact memory representation and compiled numerical operations. A one-dimensional array can represent returns; a two-dimensional array can represent observations by variables; higher dimensions appear in images and neural networks. Shape is part of meaning, so analysts should inspect it before matrix multiplication or model fitting (NumPy Developers 2026).

8.3 Vectorization states the calculation directly

The expression `prices[1:] / prices[:-1] - 1` computes a complete return vector without manually managing an index. Vectorization often improves speed because the loop occurs in optimized compiled code. It also exposes the mathematical operation. Broadcasting extends compatible shapes automatically, for example subtracting a column mean from every row. Convenience must be paired with shape checks because an unintended broadcast can produce a plausible but wrong array.

8.4 Randomness should be local and reproducible

Modern NumPy uses a random generator object. Creating `rng = np.random.default_rng(seed)` keeps the random state explicit and avoids changing a global generator used elsewhere. A seed reproduces a demonstration, not the uncertainty of the real world. Simulation results should still report Monte Carlo error and be checked across additional seeds when conclusions might depend on one stream.

8.5 Core equations

Portfolio return

$$r_{p,t} = w^\top r_t$$

A weight vector and asset-return vector are combined through a dot product.

8.6 Python demonstrations

8.6.1 Demonstration 8.1: Vectorized return calculation

```
import numpy as np

prices = np.array([100.0, 102.0, 101.0, 104.0])
returns = prices[1:] / prices[:-1] - 1
print(np.round(returns, 4))
```

Verified output

```
[ 0.02 -0.0098 0.0297]
```

Interpretation. Each return compares adjacent prices. The output has one fewer observation than the price array.

8.6.2 Demonstration 8.2: A reproducible portfolio simulation

```
import numpy as np

rng = np.random.default_rng(41202)
asset_returns = rng.normal([0.0003, 0.0002], [0.012, 0.007], size=(5, 2))
weights = np.array([0.6, 0.4])
portfolio = asset_returns @ weights
print(np.round(portfolio, 4))
```

Verified output

```
[-0.0097 0.004 0.0097 -0.0022 -0.002 ]
```

Interpretation. Matrix multiplication applies the same portfolio weights to every simulated observation.

8.7 Visual evidence

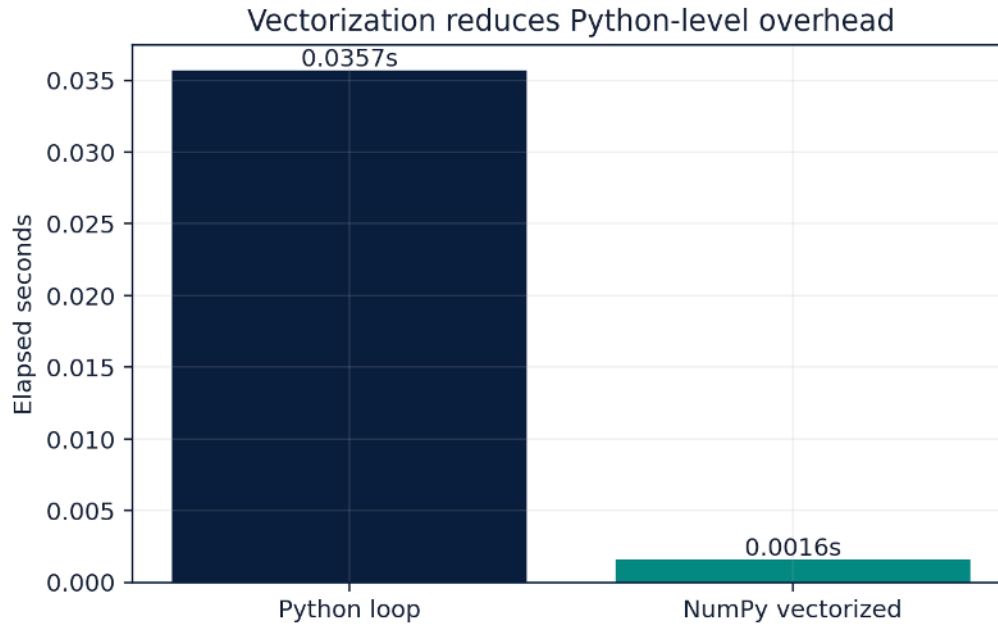


Figure 8. A local benchmark comparing element-wise squaring with a Python loop and a NumPy operation.

8.8 Reference table

Table 8. Chapter reference.

Concept	Question to ask	Typical check
shape	How many axes and elements?	<code>array.shape</code>
dtype	How are values stored?	<code>array.dtype</code>
broadcast	Which dimensions are expanded?	compare shapes
seed	Can the simulation be repeated?	record generator seed

Why This Matters

NumPy is the numerical foundation beneath pandas, statsmodels, scikit-learn, and much of scientific Python.

Common Mistake

Assuming two arrays align because their lengths match. Variables may be in a different order even when the shapes are compatible.

Ceteris LAB Tip

Print shapes at analytical boundaries: after loading, after feature construction, and before model fitting.

R-to-Python / Source Bridge

The source lectures relied on R vectors and matrix operations. NumPy provides the equivalent numerical foundation while making shape and dtype explicit.

8.9 Guided practice

1. Re-run Demonstration 8.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

8.10 Exercises

1. Create a 3 by 2 array and inspect its shape.
2. Calculate column means and demean the array using broadcasting.
3. Simulate 1,000 standard-normal values with a generator.
4. Compute a weighted portfolio return using a dot product.

8.11 Chapter summary

- NumPy arrays combine regular shape with efficient numerical operations.
- Vectorization and broadcasting can clarify mathematical structure.
- Local random generators improve reproducibility.

8.12 Key Python commands

```
import numpy as np
prices = np.array([100.0, 102.0, 101.0, 104.0])
print(np.round(returns, 4))
import numpy as np
rng = np.random.default_rng(41202)
asset_returns = rng.normal([0.0003, 0.0002], [0.012, 0.007], size=(5, 2))
```



PART II

Data Analysis, Statistics, and Econometrics

LEARN • ANALYZE • UNDERSTAND

Ceteris LAB | econometrics.safavim.com

9 pandas Fundamentals

Opening question: How can a rectangular dataset preserve labels, dates, and missing values while supporting fast analysis?

9.1 Learning objectives

- Create Series and DataFrames.
- Inspect rows, columns, indexes, and dtypes.
- Select, filter, sort, and assign variables.
- Use method chains without hiding intermediate meaning.

Prerequisites: Chapter 8.

Key terms: Series, DataFrame, index, column, dtype, method chain.

9.2 Labels are part of the data model

pandas adds row and column labels to NumPy-style arrays. A Series represents one labelled vector; a DataFrame represents columns that may have different data types. Labels make code readable, but they also introduce alignment rules. Arithmetic between two Series matches index labels rather than blindly matching position. This is powerful when dates align and dangerous when duplicated or inconsistent keys slip through (pandas Development Team 2026).

9.3 Inspection comes before transformation

A disciplined first look includes shape, column names, sample rows, data types, missingness, and summary statistics. `head()` is useful but insufficient because the first rows may not reveal late missing values or inconsistent categories. The analyst should inspect both structure and content before selecting a model. A DataFrame is not “clean” merely because it displays neatly.

9.4 Selection should reveal intent

Use column names for variables and Boolean masks for conditions. `.loc` selects by labels; `.iloc` selects by positions. Method chains can read as a workflow when each step is short: filter, assign, group, summarize. A long chain that mixes validation, transformation, and modeling is difficult to debug. Temporary named objects are helpful when they expose an important analytical stage.

9.5 Python demonstrations

9.5.1 Demonstration 9.1: Build and inspect a DataFrame

```
import pandas as pd

df = pd.DataFrame({
    "province": ["ON", "QC", "BC", "ON"],
    "income": [62_000, 58_000, 65_000, 71_000],
    "employed": [True, True, False, True],
})

print(df.shape)
print(df.dtypes.astype(str).to_dict())
```

Verified output

```
(4, 3)
{'province': 'object', 'income': 'int64', 'employed': 'bool'}
```

Interpretation. The structure reveals four observations and three variables with different data types.

9.5.2 Demonstration 9.2: Filter and assign

```
high_income = (
    df.loc[df["income"] >= 60_000]
    .assign(income_thousands=lambda x: x["income"] / 1_000)
    .sort_values("income", ascending=False)
)

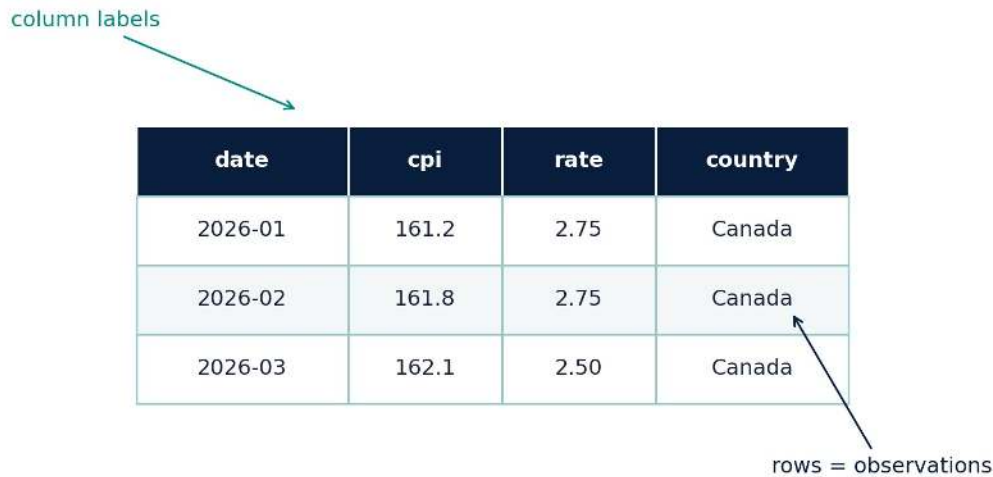
print(high_income[["province", "income_thousands"]])
```

Verified output

	province	income_thousands
3	ON	71.0
2	BC	65.0
0	ON	62.0

Interpretation. The chain reads as a sequence of analytical decisions and leaves the original DataFrame unchanged.

9.6 Visual evidence



date	cpi	rate	country
2026-01	161.2	2.75	Canada
2026-02	161.8	2.75	Canada
2026-03	162.1	2.50	Canada

Figure 9. A DataFrame organizes observations by rows and variables by labelled columns.

9.7 Reference table

Table 9. Chapter reference.

Operation	Preferred form	Meaning
Select columns	<code>df[["a", "b"]]</code>	retain named variables
Filter rows	<code>df.loc[df["x"] > 0]</code>	retain observations satisfying rule
Position	<code>df.iloc[:5]</code>	first five positions
Assign	<code>df.assign(growth=...)</code>	create variable without hidden mutation
Sort	<code>df.sort_values("date")</code>	establish order

Why This Matters

pandas makes data structure, labels, and transformation rules visible in code.

Common Mistake

Using chained indexing such as `df[df.x > 0]["y"] = ....` It can create ambiguous copies and warnings.

Ceteris LAB Tip

Use `.loc` for label-based selection and assignment, then inspect the resulting shape.

R-to-Python / Source Bridge

The source notes used R matrices and data frames for financial series. pandas supplies labelled, date-aware structures while retaining vectorized computation.

9.8 Guided practice

1. Re-run Demonstration 9.1 and change one input while keeping the analytical question fixed.

2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

9.9 Exercises

1. Create a DataFrame with five observations.
2. Filter it using two conditions.
3. Add a standardized variable.
4. Compare `.loc` and `.iloc` on the same data.

9.10 Chapter summary

- Series and DataFrames add labels and heterogeneous columns to numerical arrays.
- Inspection must cover structure, types, and missingness.
- Explicit selection and assignment reduce ambiguity.

9.11 Key Python commands

```
import pandas as pd
df = pd.DataFrame({
print(df.shape)
print(df.dtypes.astype(str).to_dict())
high_income = (
.assign(income_thousands=lambda x: x["income"] / 1_000)
```

10 Importing, Cleaning, Validating, and Exporting Data

Opening question: How can messy input be transformed without erasing the evidence of what was changed?

10.1 Learning objectives

- Read common tabular formats.
- Parse dates and numeric columns explicitly.
- Handle missing, duplicate, invalid, and extreme values.
- Write validation checks and export clean data.

Prerequisites: Chapter 9.

Key terms: CSV, schema, missing value, duplicate, outlier, validation.

10.2 Import is an inference step

When pandas reads a file, it infers delimiters, headers, types, and missing-value markers. Inference is convenient, not infallible. A column containing one nonnumeric note may become text. A date can be interpreted as month-day or day-month. Robust workflows specify critical dtypes, parse dates intentionally, and retain the original file. Parquet is useful for preserving types and compressing data, while CSV remains portable and inspectable.

10.3 Cleaning is a sequence of justified decisions

Missingness, duplicates, and outliers are different problems. A missing value may be expected because a market was closed; a duplicate may be a genuine repeated transaction; an extreme observation may be the event of greatest interest. Cleaning rules therefore require domain knowledge. Every deletion or replacement should be reproducible, counted, and described. A validation report is often more informative than a single “cleaned” file.

10.4 A schema makes assumptions executable

Validation checks can assert unique keys, allowed categories, positive prices, monotonic dates, and plausible ranges. Failing early is better than fitting a model to invalid inputs. The goal is not to force data into a preferred story, but to test whether the file satisfies the conditions required by the next analytical step.

10.5 Python demonstrations

10.5.1 Demonstration 10.1: Clean a small table

```
import pandas as pd
from io import StringIO

raw = StringIO("date,value\n2026-01-01,2.1\n2026-02-01,NA\n2026-02-01,2.4")
df = pd.read_csv(raw, parse_dates=["date"], na_values=["NA"])
df["is_duplicate_date"] = df.duplicated("date", keep=False)
print(df.isna().sum().to_dict())
print(df["is_duplicate_date"].sum())
```

Verified output

```
{'date': 0, 'value': 1, 'is_duplicate_date': 0}
2
```

Interpretation. The missing-value count and duplicate flag describe separate quality concerns.

10.5.2 Demonstration 10.2: Validate an economic series

```
def validate_series(frame):
    assert frame["date"].notna().all(), "dates must be present"
    assert frame["date"].is_monotonic_increasing, "dates must be sorted"
    assert frame["value"].dropna().between(-100, 100).all(), "value outside range"
    return True

print(validate_series(df.sort_values("date")))
```

Verified output

```
True
```

Interpretation. The range is intentionally broad for illustration. Real thresholds should reflect the variable and units.

10.6 Visual evidence

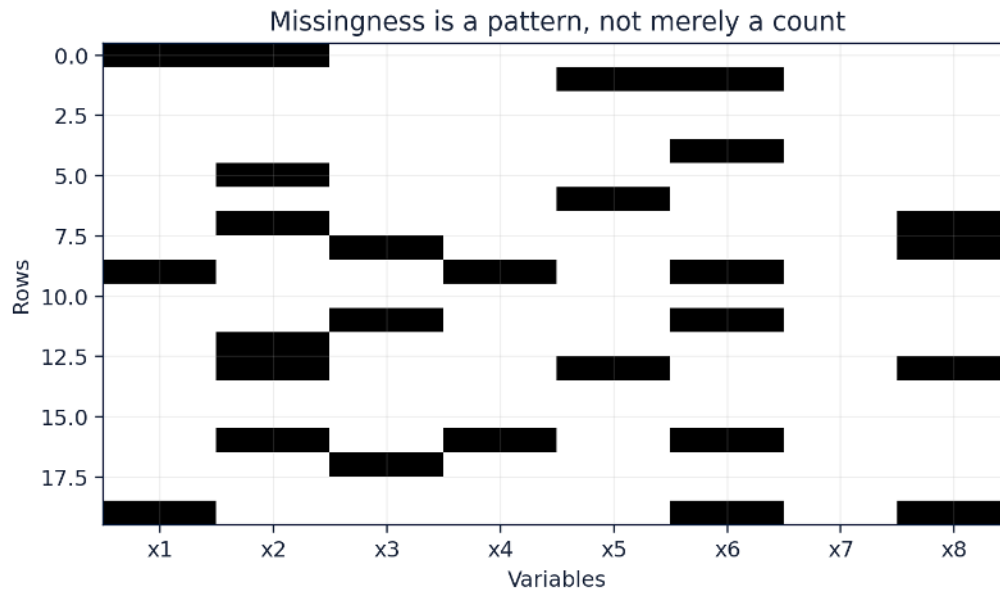


Figure 10. A missing-value map helps reveal whether gaps are isolated, clustered, or systematic.

10.7 Reference table

Table 10. Chapter reference.

Issue	Diagnostic	Possible response
Missing value	<code>isna().sum()</code>	investigate mechanism; impute only with justification
Duplicate key	<code>duplicated(keys)</code>	resolve or aggregate with documented rule
Invalid range	<code>between(low, high)</code>	correct source or reject row
Wrong dtype	<code>dtypes / to_numeric</code>	parse with explicit error handling
Outlier	robust summaries and plots	retain, flag, winsorize only with rationale

Why This Matters

Data validation prevents errors from becoming polished charts and confident coefficients.

Common Mistake

Dropping all missing rows without asking why they are missing or how the deletion changes the sample.

Ceteris LAB Tip

Create a quality table before cleaning and another after cleaning. The difference is your audit trail.

R-to-Python / Source Bridge

Older examples loaded text files with implicit assumptions. This chapter replaces them with explicit schemas, validation, and an audit trail.

10.8 Guided practice

1. Re-run Demonstration 10.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

10.9 Exercises

1. Import a CSV with explicit date parsing.
2. Count missing values by column.
3. Check uniqueness of a composite key.
4. Export a processed table and a metadata file.

10.10 Chapter summary

- Import decisions affect types and missingness.
- Cleaning rules require substantive justification.
- Validation turns assumptions into checks.

10.11 Key Python commands

```
import pandas as pd
from io import StringIO
raw = StringIO("date,value\n2026-01-01,2.1\n2026-02-01,NA\n2026-02-01,2.4")
df = pd.read_csv(raw, parse_dates=["date"], na_values=["NA"])
df["is_duplicate_date"] = df.duplicated("date", keep=False)
print(df.isna().sum().to_dict())
```

11 Manipulating, Grouping, Reshaping, and Joining Data

Opening question: How can information be reorganized without accidentally changing the number or meaning of observations?

11.1 Learning objectives

- Aggregate with groupby.
- Move between wide and long forms.
- Merge tables using keys.
- Diagnose duplicate keys and many-to-many joins.

Prerequisites: Chapters 9 and 10.

Key terms: groupby, aggregation, pivot, melt, merge, key.

11.2 Grouping changes the unit of analysis

A grouped summary collapses observations within categories or time periods. The output unit must be stated: an average across households is not a household-level variable, and a monthly average of daily rates is not a month-end rate. groupby is most useful when the grouping keys, aggregation function, and output names are explicit.

11.3 Wide and long are views of the same logic

Wide data place repeated measures in separate columns; long data place the measure name in one column and values in another. Long form is often convenient for plotting and grouped operations, while wide form is useful for matrix models such as VAR. Reshaping should preserve a key that uniquely identifies each observation. A failed pivot often reveals duplicates that need substantive resolution.

11.4 Joins are models of relationships

A merge states how two tables are related. One-to-one and many-to-one joins are usually easiest to verify. An unintended many-to-many join can multiply rows and distort totals. pandas can validate expected cardinality. Analysts should compare row counts, unmatched keys, and duplicate keys before and after every important merge.

11.5 Python demonstrations

11.5.1 Demonstration 11.1: Group and summarize

```
import pandas as pd

df = pd.DataFrame({
    "province": ["ON", "ON", "QC", "QC"],
    "year": [2025, 2026, 2025, 2026],
    "income": [60, 63, 55, 58],
})
summary = df.groupby("province", as_index=False).agg(mean_income=("income", "mean"))
print(summary)
```

Verified output

	province	mean_income
0	ON	61.5
1	QC	56.5

Interpretation. The result has one row per province, so its unit differs from the original province-year table.

11.5.2 Demonstration 11.2: Validate a merge

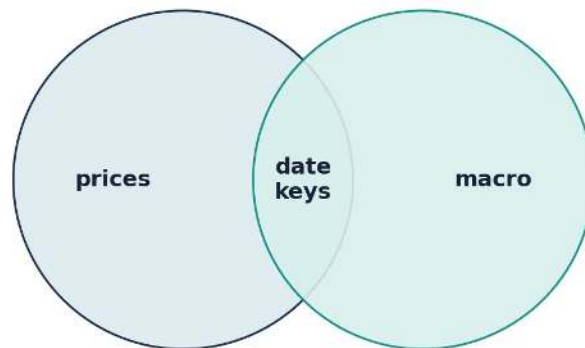
```
meta = pd.DataFrame({"province": ["ON", "QC"], "region": ["Central", "Central"]})
merged = df.merge(meta, on="province", how="left", validate="many_to_one",
indicator=True)
print(merged.shape)
print(merged["_merge"].value_counts().to_dict())
```

Verified output

```
(4, 5)
{'both': 4, 'left_only': 0, 'right_only': 0}
```

Interpretation. Cardinality validation and the merge indicator confirm that metadata attached without multiplying rows.

11.6 Visual evidence



A join is only valid when the key is unique at the intended level.

Figure 11. Joining two tables requires explicit keys and duplicate-key diagnostics.

11.7 Reference table

Table 11. Chapter reference.

Join	Keeps	Use case
inner	matching keys only	complete overlap
left	all left rows	attach metadata
right	all right rows	less common mirror of left
outer	all keys	reconciliation and coverage audit

Why This Matters

Many large analytical errors begin with an unnoticed change in observational unit during aggregation or merging.

Common Mistake

Joining on a variable that is not unique in the lookup table. The output may expand without an obvious error message.

Ceteris LAB Tip

Write the expected join cardinality in words before writing the merge code.

R-to-Python / Source Bridge

Financial and macroeconomic source examples combine dates, identifiers, and multiple series. This chapter supplies the safe restructuring tools needed before time-series modeling.

11.8 Guided practice

1. Re-run Demonstration 11.1 and change one input while keeping the analytical question fixed.

2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

11.9 Exercises

1. Create a grouped mean and count.
2. Reshape a two-year wide table to long form.
3. Perform a left join with validation.
4. Find keys that do not match across two tables.

11.10 Chapter summary

- Grouping changes the unit of analysis.
- Wide and long formats support different tasks.
- Join cardinality and unmatched keys must be audited.

11.11 Key Python commands

```
import pandas as pd
df = pd.DataFrame({
summary = df.groupby("province", as_index=False).agg(mean_income=("income", "mean"))
print(summary)
meta = pd.DataFrame({"province": ["ON", "QC"], "region": ["Central", "Central"]})
merged = df.merge(meta, on="province", how="left", validate="many_to_one",
indicator=True)
```

12 Data Visualization with Python

Opening question: What should a figure reveal that a table or coefficient cannot show as clearly?

12.1 Learning objectives

- Construct line, scatter, distribution, and categorical charts.
- Label units, sources, and transformations.
- Design for accessibility and grayscale printing.
- Recognize misleading scales and overplotting.

Prerequisites: Chapters 8 to 11.

Key terms: figure, axes, encoding, scale, annotation, accessibility.

12.2 A chart is an argument with visual grammar

A figure maps variables to position, length, shape, and sometimes colour. Position along a common scale is usually easier to compare than area or angle. Time belongs on an ordered horizontal axis; distributions need enough bins or a density representation; uncertainty should be shown when it affects interpretation. Matplotlib exposes the figure and axes objects that control these choices (Matplotlib Development Team 2026).

12.3 Labels protect meaning

A title should state the subject, not merely repeat a variable name. Axes need units, transformations, and frequency. A source note should identify the provider and whether values were calculated by the author. Legends are useful when direct labels are impractical. An annotation can point to a recession, policy change, or data break, but decoration should not compete with the evidence.

12.4 Accessibility is part of accuracy

Colour should not be the only channel distinguishing categories. Contrasting line styles, markers, labels, and sufficient luminance differences help readers with colour-vision differences and support grayscale printing. Truncated axes can exaggerate small changes, while dual axes can imply relationships that depend on arbitrary scaling. The default chart should be honest before it is beautiful.

12.5 Python demonstrations

12.5.1 Demonstration 12.1: A labelled time-series chart

```
import matplotlib.pyplot as plt
import pandas as pd

series = pd.Series([1.2, 1.4, 1.3, 1.6], index=pd.date_range("2026-01-01", periods=4,
freq="MS"))
fig, ax = plt.subplots()
ax.plot(series.index, series.values, marker="o")
ax.set(title="Illustrative Monthly Index", xlabel="Month", ylabel="Index points")
fig.tight_layout()
plt.close(fig)
print(len(series))
```

Verified output

```
4
```

Interpretation. The code creates a reusable figure object and labels both the time dimension and the unit.

12.5.2 Demonstration 12.2: A distribution check

```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(12)
values = rng.standard_t(df=5, size=1_000)
fig, ax = plt.subplots()
ax.hist(values, bins=35, density=True)
ax.set(title="Heavy-tailed simulated observations", xlabel="Value", ylabel="Density")
plt.close(fig)
print(round(values.std(), 3))
```

Verified output

```
1.263
```

Interpretation. The histogram reveals tails and concentration that a standard deviation alone cannot convey.

12.6 Visual evidence

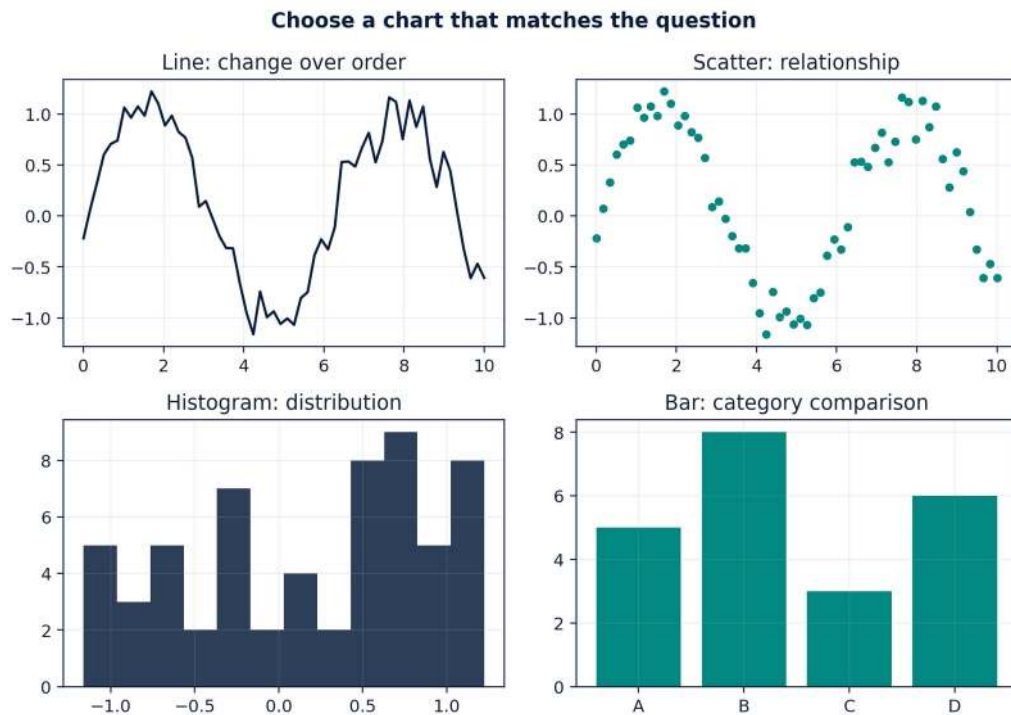


Figure 12. Four common chart forms and the analytical question each answers.

12.7 Reference table

Table 12. Chapter reference.

Question	Recommended chart	Common caution
How does a variable evolve?	line chart	gaps and frequency
How are two variables related?	scatterplot	overplotting and causality
What is the distribution?	histogram/density/boxplot	bin and bandwidth choices
How do categories compare?	bar or dot plot	zero baseline for bars

Why This Matters

Visualization is a diagnostic instrument as well as a communication device.

Common Mistake

Selecting a chart type because it looks impressive rather than because it matches the analytical question.

Ceteris LAB Tip

Before exporting, inspect the chart at the size students will actually see in the PDF or website.

R-to-Python / Source Bridge

All low-resolution source figures were recreated as original Python visuals. The new figure library uses consistent captions, labels, and Ceteris LAB styling rather than slide screenshots.

12.8 Guided practice

1. Re-run Demonstration 12.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

12.9 Exercises

1. Create a line chart with units and source note.
2. Compare a histogram under two bin choices.
3. Create a scatterplot and add a fitted line.
4. Redesign a misleading truncated-axis chart.

12.10 Chapter summary

- Charts map data to visual channels.
- Titles, units, sources, and transformations protect interpretation.
- Accessible design improves accuracy for every reader.

12.11 Key Python commands

```
import matplotlib.pyplot as plt
import pandas as pd
fig, ax = plt.subplots()
ax.plot(series.index, series.values, marker="o")
ax.set(title="Illustrative Monthly Index", xlabel="Month", ylabel="Index points")
fig.tight_layout()
```

13 Descriptive Statistics and Exploratory Data Analysis

Opening question: How can a dataset be summarized without letting one number erase its shape?

13.1 Learning objectives

- Calculate location, spread, shape, and dependence statistics.
- Compare conventional and robust summaries.
- Interpret skewness and kurtosis.
- Use plots and tables together.

Prerequisites: Chapters 9 to 12.

Key terms: mean, median, variance, skewness, kurtosis, correlation.

13.2 Location and spread answer different questions

The mean balances all observations and is sensitive to extreme values. The median identifies the middle and is more robust. Variance and standard deviation describe dispersion around the mean, while the interquartile range describes the middle half of the data. These measures should be selected according to the distribution and the decision, not reported mechanically.

13.3 Shape matters in finance

Asset returns often exhibit skewness and heavier tails than a Gaussian model. Skewness describes asymmetry; kurtosis describes the weight of tails and concentration relative to a reference convention. Software differs on whether reported kurtosis subtracts three. The book uses Fisher excess kurtosis when the normal reference is zero and labels the convention explicitly (Tsay 2010).

13.4 Dependence is not a single coefficient

Pearson correlation measures linear association and can be dominated by outliers. Spearman and Kendall correlations use ranks and capture monotonic relationships. None proves causality, and correlation can vary across regimes or frequencies. Exploratory analysis should combine scatterplots, subgroup summaries, and time ordering before a dependence measure is interpreted.

13.5 Core equations

Sample variance

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

The denominator $n-1$ estimates population variance under standard independent-sampling assumptions.

13.6 Python demonstrations

13.6.1 Demonstration 13.1: Conventional and robust summaries

```
import pandas as pd
from scipy import stats

x = pd.Series([10, 11, 12, 13, 55])
print({
    "mean": x.mean(),
    "median": x.median(),
    "std": x.std(),
    "iqr": stats.iqr(x),
})
```

Verified output

```
{'mean': np.float64(20.2), 'median': np.float64(12.0), 'std': np.float64(19.485892332659542),
'iqr': np.float64(2.0)}
```

Interpretation. The extreme value pulls the mean and standard deviation upward, while the median and IQR remain close to the central cluster.

13.6.2 Demonstration 13.2: Distribution shape

```
import numpy as np
from scipy import stats

rng = np.random.default_rng(13)
returns = rng.standard_t(df=5, size=5_000)
print(round(stats.skew(returns), 3))
print(round(stats.kurtosis(returns, fisher=True), 3))
```

Verified output

```
0.302
10.268
```

Interpretation. The excess kurtosis is well above zero, consistent with heavier tails than a normal distribution.

13.7 Visual evidence

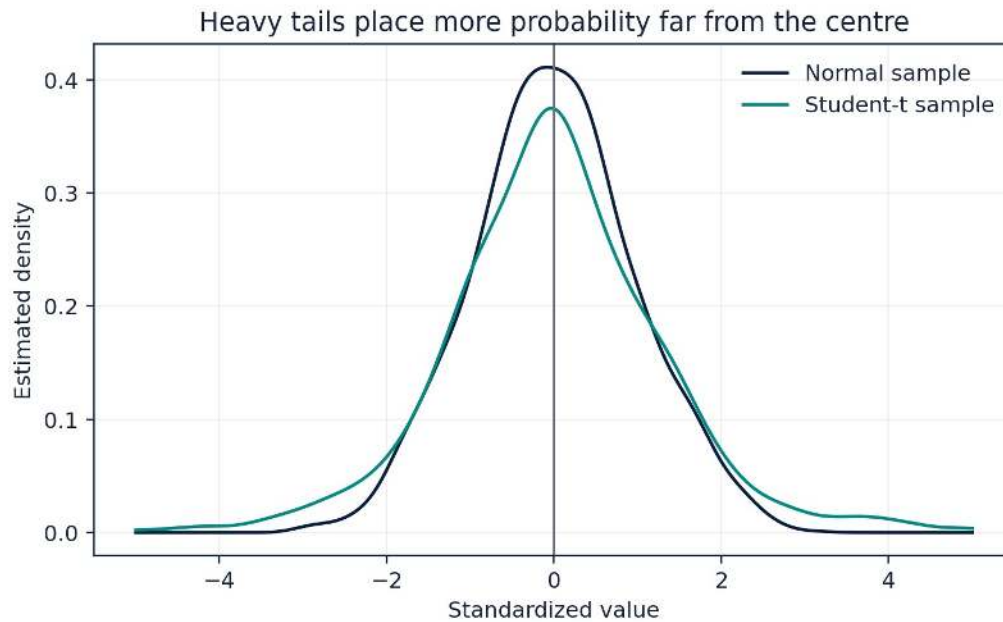


Figure 13. Kernel-density estimates for normal and heavy-tailed samples.

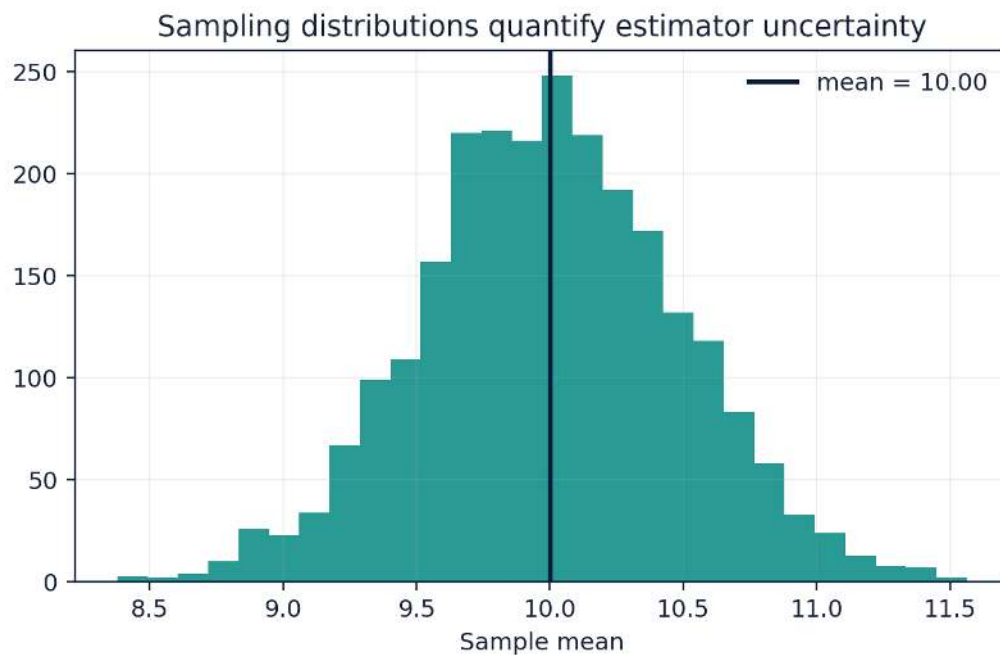


Figure 14. Distribution of 2,500 sample means, each based on 40 observations.

13.8 Reference table

Table 13. Chapter reference.

Statistic	Sensitive to extremes?	Primary meaning
mean	Yes	arithmetic centre
median	Less	middle observation
standard deviation	Yes	dispersion around mean
IQR	Less	middle-half spread
skewness	Yes	asymmetry
excess kurtosis	Very	tail weight and peakedness

Why This Matters

Exploration reveals which summaries, transformations, and models are plausible before formal estimation begins.

Common Mistake

Reporting correlation to three decimals without showing the scatterplot, subgroup structure, or time ordering.

Ceteris LAB Tip

Pair every important summary statistic with a plot that can reveal why it takes that value.

R-to-Python / Source Bridge

The source lecture on financial time series emphasized skewness, heavy tails, and dependence. This chapter preserves those foundations and adds explicit convention checks and robust alternatives (Tsay 2013).

13.9 Guided practice

1. Re-run Demonstration 13.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

13.10 Exercises

1. Compare mean and median in a skewed sample.
2. Calculate Fisher and Pearson kurtosis and explain the difference.
3. Compare Pearson and Spearman correlations.
4. Build a grouped descriptive-statistics table.

13.11 Chapter summary

- Location, spread, shape, and dependence describe different features.
- Robust statistics reduce sensitivity to extremes.
- Financial returns frequently require heavy-tail diagnostics.

13.12 Key Python commands

```
import pandas as pd
from scipy import stats
x = pd.Series([10, 11, 12, 13, 55])
print({
    "mean": x.mean(),
    "median": x.median(),
```

14 Probability, Simulation, and Statistical Inference

Opening question: How can uncertainty be represented, simulated, and summarized without pretending that one sample is the population?

14.1 Learning objectives

- Work with random variables and common distributions.
- Use Monte Carlo simulation.
- Explain sampling distributions and standard errors.
- Interpret confidence intervals, tests, and power.

Prerequisites: Chapter 13 and basic algebra.

Key terms: random variable, distribution, Monte Carlo, standard error, confidence interval, p-value.

14.2 Probability describes a model of uncertainty

A probability distribution assigns relative plausibility to possible outcomes under stated assumptions. The normal distribution is useful for averages and measurement errors; the binomial for counts of successes; the Student-t for heavier-tailed standardized quantities; and the lognormal for positive multiplicative processes. Real data need not follow any textbook distribution exactly. A distribution is a working model whose implications should be checked.

14.3 Simulation turns assumptions into observable consequences

Monte Carlo simulation draws repeated outcomes from a model and approximates quantities that may be difficult to calculate analytically. The method supports option pricing, forecast intervals, power analysis, and risk measurement. More replications reduce simulation noise at a square-root rate, so ten times more computation does not produce ten times more precision. Seeds make examples repeatable; multiple seeds test whether a conclusion is fragile.

14.4 Inference concerns repeated-sample behaviour

A standard error estimates how a statistic would vary across hypothetical repeated samples. A confidence interval is constructed by a procedure with a long-run coverage property; it is not automatically a probability statement about a fixed parameter. A p-value measures

compatibility between data and a null model, not the probability that the null is true. Statistical significance should be paired with effect size, uncertainty, design, and practical relevance.

14.5 Core equations

Standard error of the mean

$$SE(\bar{X}) = \frac{s}{\sqrt{n}}$$

Under standard independent-sampling conditions, precision improves with the square root of sample size.

14.6 Python demonstrations

14.6.1 Demonstration 14.1: Monte Carlo estimate of a probability

```
import numpy as np
rng = np.random.default_rng(1401)
draws = rng.normal(0, 1, 100_000)
probability = np.mean(draws > 1.96)
print(round(probability, 4))
```

Verified output

```
0.0249
```

Interpretation. The estimate is close to the theoretical upper-tail probability of 0.025, with small Monte Carlo error.

14.6.2 Demonstration 14.2: A confidence interval for a mean

```
import numpy as np
from scipy import stats
sample = np.array([2.1, 2.4, 2.0, 2.7, 2.3, 2.5])
mean = sample.mean()
se = stats.sem(sample)
interval = stats.t.interval(0.95, df=len(sample)-1, loc=mean, scale=se)
print(round(mean, 3), tuple(round(v, 3) for v in interval))
```

Verified output

```
2.333 (np.float64(2.062), np.float64(2.604))
```

Interpretation. The interval is wider than a normal-based interval because the small sample uses a Student-t critical value.

14.7 Visual evidence

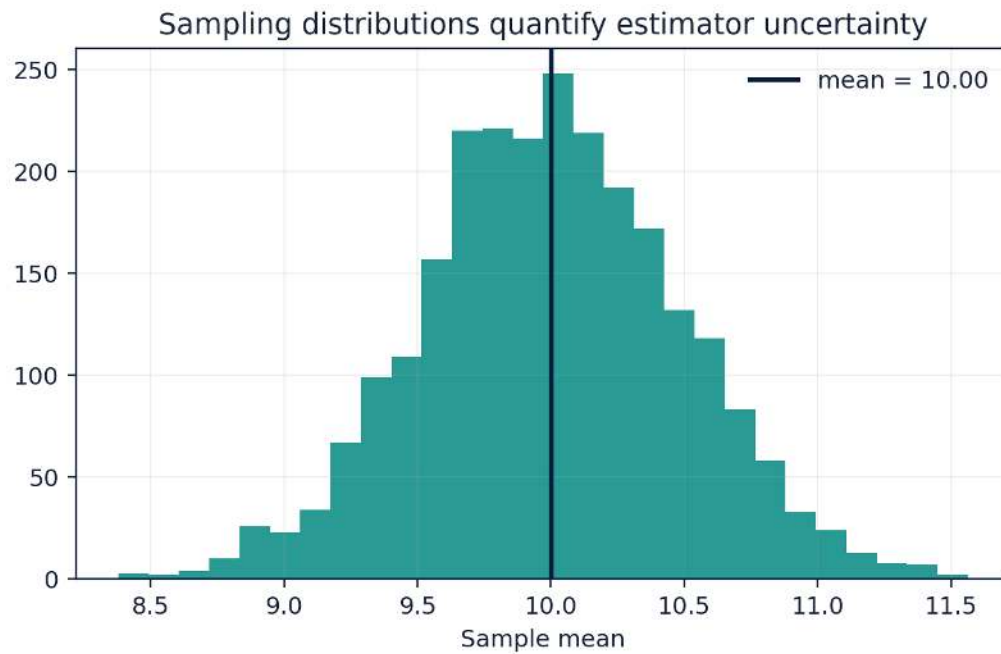


Figure 14. Distribution of 2,500 sample means, each based on 40 observations.

14.8 Reference table

Table 14. Chapter reference.

Quantity	What varies?	Interpretation
standard deviation	individual observations	spread in data or model
standard error	statistic across samples	estimation precision
confidence interval	procedure across samples	coverage under assumptions
p-value	test statistic under null	compatibility with null model
power	test decision under alternative	probability of detecting specified effect

Why This Matters

Inference organizes uncertainty around estimates, forecasts, and decisions.

Common Mistake

Interpreting a p-value of 0.03 as a 3 percent probability that the null hypothesis is true.

Ceteris LAB Tip

Report effect size and interval before discussing a significance threshold.

R-to-Python / Source Bridge

The book adds probability and inference prerequisites that the advanced source notes often assumed, allowing beginners to enter later econometric chapters safely.

14.9 Guided practice

1. Re-run Demonstration 14.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

14.10 Exercises

1. Simulate a binomial proportion.
2. Show how Monte Carlo error changes with replications.
3. Compute a t confidence interval.
4. Explain statistical versus practical significance in one example.

14.11 Chapter summary

- Probability models uncertainty under assumptions.
- Simulation approximates model implications.
- Inference requires careful interpretation of standard errors, intervals, and tests.

14.12 Key Python commands

```
import numpy as np
rng = np.random.default_rng(1401)
draws = rng.normal(0, 1, 100_000)
probability = np.mean(draws > 1.96)
print(round(probability, 4))
import numpy as np
```

15 Regression and Econometrics with Python

Opening question: What does a regression coefficient mean, and which assumptions are needed before it can support an economic claim?

15.1 Learning objectives

- Estimate simple and multiple linear regressions.
- Interpret coefficients, interactions, and uncertainty.
- Diagnose residual patterns and heteroskedasticity.
- Distinguish explanatory inference from predictive evaluation.

Prerequisites: Chapters 13 and 14.

Key terms: OLS, coefficient, residual, robust standard error, interaction, omitted variable.

15.2 Regression is a conditional comparison

Ordinary least squares chooses coefficients that minimize the sum of squared residuals. A slope describes the estimated change in the conditional mean of the outcome associated with a one-unit change in a predictor, holding included variables fixed. That language is descriptive unless the research design supports a causal interpretation. The list of included variables does not by itself eliminate omitted-variable bias.

15.3 Uncertainty depends on the error structure

Classical OLS standard errors assume a particular variance and dependence structure. Heteroskedasticity changes the conditional variance across observations; serial or clustered dependence links errors. Robust covariance estimators can improve inference under specified forms of misspecification, but they do not repair biased coefficients or a poorly defined sample. Diagnostics should examine residual plots, leverage, influential observations, and functional form.

15.4 Prediction and inference optimize different questions

statsmodels emphasizes statistical models, coefficient tables, tests, and diagnostics (statsmodels Developers 2026). scikit-learn emphasizes out-of-sample prediction, pipelines, and model comparison (scikit-learn Developers 2026). The same linear formula can serve either purpose, but evaluation differs. A model with interpretable coefficients may not be the best forecaster; a strong predictive model may not identify structural effects.

15.5 Core equations

Linear regression

$$y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki} + u_i$$

The residual u captures unobserved components relative to the specified conditional mean.

15.6 Python demonstrations

15.6.1 Demonstration 15.1: Estimate OLS with robust standard errors

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

rng = np.random.default_rng(15)
x = np.linspace(0, 10, 120)
y = 1.5 + 0.8*x + rng.normal(0, 0.4 + 0.08*x)
df = pd.DataFrame({"y": y, "x": x})
model = smf.ols("y ~ x", data=df).fit(cov_type="HC3")
print(round(model.params["x"], 3), round(model.bse["x"], 3))
```

Verified output

```
0.848 0.028
```

Interpretation. The slope is close to the data-generating value. HC3 standard errors account for heteroskedasticity in a large-sample approximation.

15.6.2 Demonstration 15.2: Interpret an interaction

```
df["group"] = (df["x"] > 5).astype(int)
interaction = smf.ols("y ~ x * group", data=df).fit(cov_type="HC3")
print(interaction.params.round(3).to_dict())
```

Verified output

```
{'Intercept': 1.503, 'x': 0.809, 'group': -0.565, 'x:group': 0.099}
```

Interpretation. The interaction allows the slope to differ after x exceeds five. Here the estimated difference is small because the simulated process has one common slope.

15.7 Visual evidence

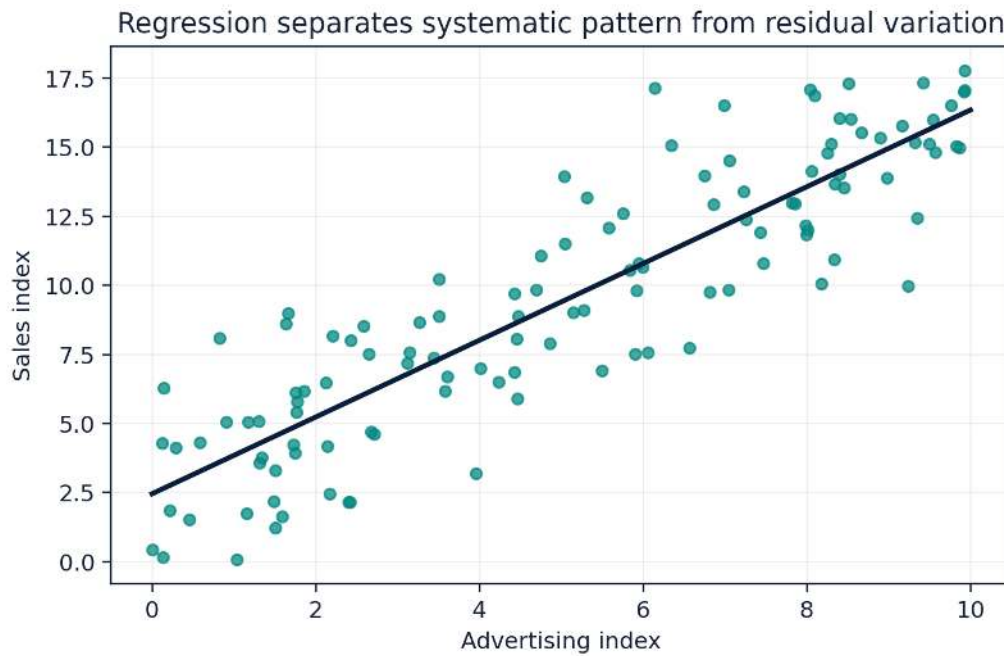


Figure 15. A fitted linear relationship with simulated business data.

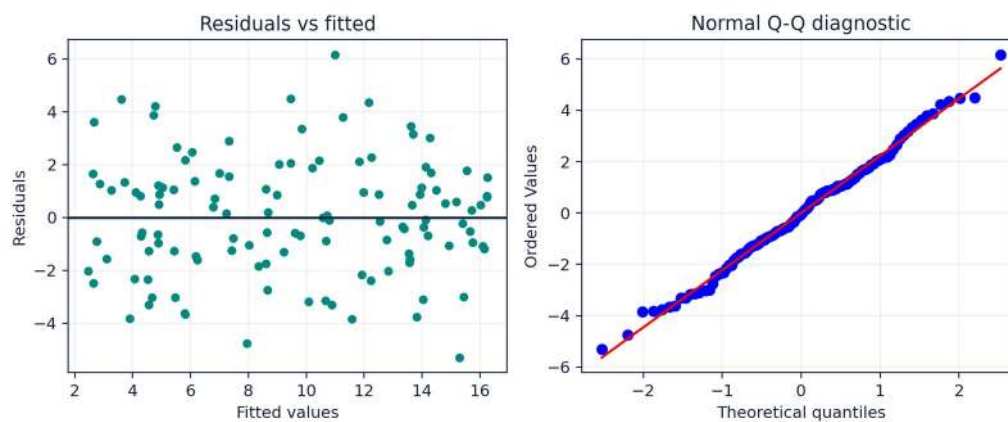


Figure 16. Residual plots reveal nonlinearity, changing variance, and distributional departures.

15.8 Reference table

Table 15. Chapter reference.

Element	Interpretation	Caution
coefficient	conditional mean difference	units and coding matter
standard error	sampling uncertainty estimate	depends on covariance assumptions
R-squared	in-sample variance share	not causality or forecast quality
residual	observed minus fitted	not the true structural error
interaction	effect depends on another variable	include lower-order terms

Why This Matters

Regression provides a disciplined language for conditional relationships, uncertainty, and model diagnostics.

Common Mistake

Describing an observational coefficient as “the impact” without a design that addresses confounding and reverse causality.

Ceteris LAB Tip

Write the coefficient interpretation with units and the phrase “holding included variables fixed” before discussing causality.

R-to-Python / Source Bridge

Regression with serially correlated errors appears later in the source seasonal lecture. This chapter builds the cross-sectional foundation and emphasizes identification before time-series extensions.

15.9 Guided practice

1. Re-run Demonstration 15.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

15.10 Exercises

1. Estimate a simple regression and interpret the slope.
2. Add a categorical predictor and explain the reference group.
3. Compare conventional and HC3 standard errors.
4. Design a train-test evaluation for a predictive regression.

15.11 Chapter summary

- OLS estimates a conditional linear approximation.
- Inference depends on covariance assumptions and research design.
- Explanatory and predictive goals require different evaluation.

15.12 Key Python commands

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
rng = np.random.default_rng(15)
x = np.linspace(0, 10, 120)
y = 1.5 + 0.8*x + rng.normal(0, 0.4 + 0.08*x)
```

16 Obtaining Economic and Financial Data

Opening question: How can a live data source be used without making the analysis disappear when the network or provider changes?

16.1 Learning objectives

- Retrieve data from authoritative APIs.
- Record series IDs, units, frequencies, and retrieval dates.
- Cache legally redistributable snapshots.
- Build graceful fallbacks and error handling.

Prerequisites: Chapters 7, 9, and 10.

Key terms: API, endpoint, series identifier, provenance, cache, licence.

16.2 Prefer authoritative providers

The Bank of Canada Valet API, Statistics Canada Web Data Service, FRED, and the World Bank provide documented programmatic access to economic series (Bank of Canada 2026; Statistics Canada 2026; Federal Reserve Bank of St. Louis 2026; World Bank 2026). An API request should name the series and date range, validate the response, and preserve provider metadata. Market-data interfaces require extra caution because unofficial endpoints may change and redistribution rights may differ from access rights.

16.3 A live call is not a reproducibility plan

A future request can return revised data, a changed schema, or an error. When licensing permits, store a frozen snapshot with its retrieval date and keep the retrieval script. When redistribution is restricted, store code and metadata rather than raw observations. Essential teaching examples should include a small deterministic fallback so that the lesson remains usable offline.

16.4 Validate response semantics

A successful HTTP status does not prove that the requested series, frequency, or units are correct. Inspect column names, date coverage, missing values, and observation counts. Economic series can be revised and may represent seasonally adjusted, annualized, index, level, or growth-rate concepts. The data dictionary belongs beside the code.

16.5 Python demonstrations

16.5.1 Demonstration 16.1: Load the frozen Bank of Canada example

```
from pathlib import Path
import pandas as pd

path = Path("data/frozen/boc_fxusdcad_2024-04-15_2024-05-15.csv")
fx = pd.read_csv(path, skiprows=8)
value_col = [c for c in fx.columns if "FXUSDCAD" in c][-1]
fx[value_col] = pd.to_numeric(fx[value_col], errors="coerce")
print(fx.shape[0], round(fx[value_col].mean(), 4))
```

Verified output

```
23 1.3708
```

Interpretation. The frozen file makes the example independent of a live request while preserving the official series identifier.

16.5.2 Demonstration 16.2: A guarded network pattern

```
import requests

url = "https://www.bankofcanada.ca/vallet/observations/FXUSDCAD/json"
try:
    response = requests.get(url, params={"start_date": "2024-04-15", "end_date": "2024-04-19"},
    timeout=20)
    response.raise_for_status()
    observations = response.json().get("observations", [])
    print(len(observations))
except requests.RequestException:
    # Offline fallback: the same five business-day observations are kept locally.
    observations = [None] * 5
print(len(observations))
```

Verified output

```
5
```

Interpretation. The timeout and exception handling prevent an indefinite hang. Production code should also validate schema and use the frozen fallback.

16.6 Visual evidence



Provenance records provider, identifier, retrieval date, units, and licence.

Figure 17. A resilient data-retrieval pipeline validates and caches official observations.

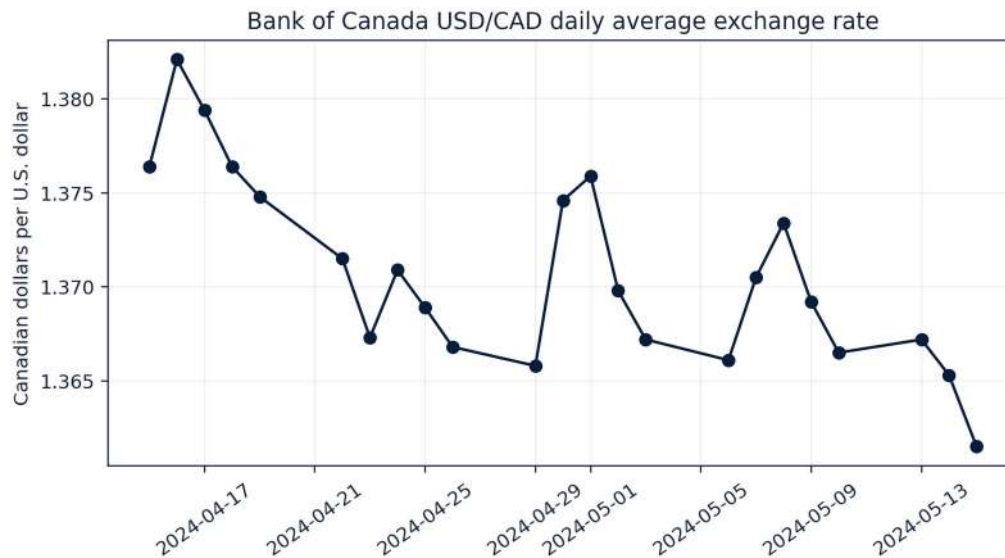


Figure 65. Official Bank of Canada FXUSDCAD observations from 15 April to 15 May 2024.

16.7 Reference table

Table 16. Chapter reference.

Provider	Strength	Required metadata
Bank of Canada	Canadian rates and exchange rates	series code, units, date range
Statistics Canada	official Canadian tables	table/product ID and vector IDs
FRED	U.S. and international macro series	series ID and frequency
World Bank	cross-country indicators	indicator code and country coverage
Market-data interface	prices and volumes	provider status, adjustments, licence

Why This Matters

Reproducible data access links an empirical result to an identifiable provider, series, transformation, and retrieval date.

Common Mistake

Using a ticker or series code without documenting whether the values are adjusted, seasonally adjusted, annualized, or revised.

Ceteris LAB Tip

Save the raw response headers or metadata alongside the observations whenever the provider supplies them.

R-to-Python / Source Bridge

The 2013 pre-class notes used `quantmod` and now-obsolete Google Finance examples. This chapter replaces them with current official APIs, documented market-data cautions, and a frozen Bank of Canada example (Tsay 2013).

16.8 Guided practice

1. Re-run Demonstration 16.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

16.9 Exercises

1. Retrieve a short official series and record metadata.
2. Write a fallback that loads a frozen file.
3. Validate expected date coverage.
4. Explain the difference between access permission and redistribution permission.

16.10 Chapter summary

- Authoritative APIs improve provenance.
- Frozen snapshots and retrieval scripts support reproducibility.
- Schema, units, frequency, and licensing require explicit validation.

16.11 Key Python commands

```
from pathlib import Path
import pandas as pd
path = Path("data/frozen/boc_fxusdcad_2024-04-15_2024-05-15.csv")
fx = pd.read_csv(path, skiprows=8)
fx[value_col] = pd.to_numeric(fx[value_col], errors="coerce")
print(fx.shape[0], round(fx[value_col].mean(), 4))
```



PART III

Time Series and Financial Econometrics

LEARN • ANALYZE • UNDERSTAND

Ceteris LAB | econometrics.safavim.com

Part III: Core Econometrics and Causal Inference

From association to identification, from simple regression to causal machine learning

17 Econometric Thinking and Research Design

What exactly are we trying to learn from economic data?

17.1 Learning objectives

Explain and apply economic model versus econometric model.

Explain and apply population, sample, parameter, estimate, and estimand.

Explain and apply data-generating processes.

Explain and apply exogeneity and endogeneity.

Explain and apply identification.

Explain and apply prediction versus causal inference.

Explain and apply cross-sectional, panel, and time-series designs.

Concept in one sentence: A regression coefficient can summarize an association without identifying a causal effect.

17.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about econometric thinking and research design. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

17.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

17.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

17.5 Python demonstration

```
import numpy as np
import statsmodels.api as sm
rng = np.random.default_rng(17)
x = rng.normal(size=300)
y = 1 + 0.8*x + rng.normal(size=300)
model = sm.OLS(y, sm.add_constant(x)).fit()
print(model.params)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

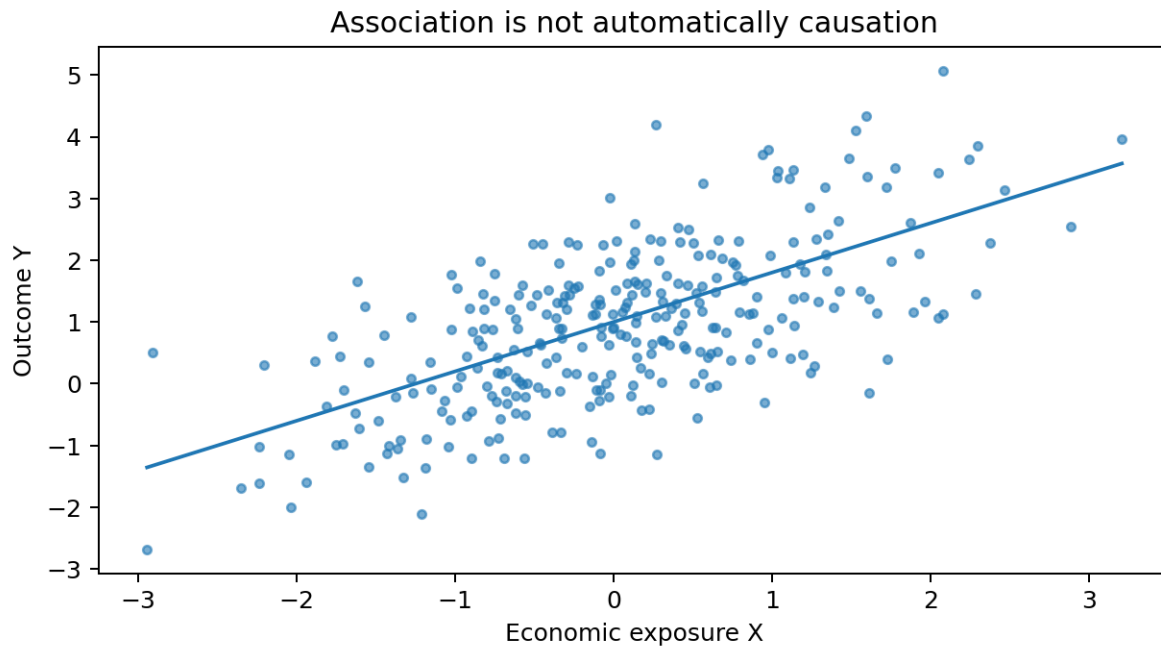


Figure 17.1. What exactly are we trying to learn from economic data?

17.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

17.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

17.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

17.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

17.10 Chapter summary

Econometric Thinking and Research Design is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

17.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

18 Simple Linear Regression

How does consumption change with income in a simple model?

18.1 Learning objectives

Explain and apply population regression function.

Explain and apply ordinary least squares.

Explain and apply intercept and slope.

Explain and apply fitted values and residuals.

Explain and apply R-squared.

Explain and apply confidence intervals.

Explain and apply prediction intervals.

Explain and apply units and economic interpretation.

Concept in one sentence: OLS chooses the line that minimizes the sum of squared residuals.

18.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about simple linear regression. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

18.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

18.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

18.5 Python demonstration

```
import numpy as np
import statsmodels.api as sm
rng=np.random.default_rng(18)
income=np.linspace(20,100,120)
cons=8+0.55*income+rng.normal(0,8,120)
res=sm.OLS(cons,sm.add_constant(income)).fit()
print(res.summary())
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

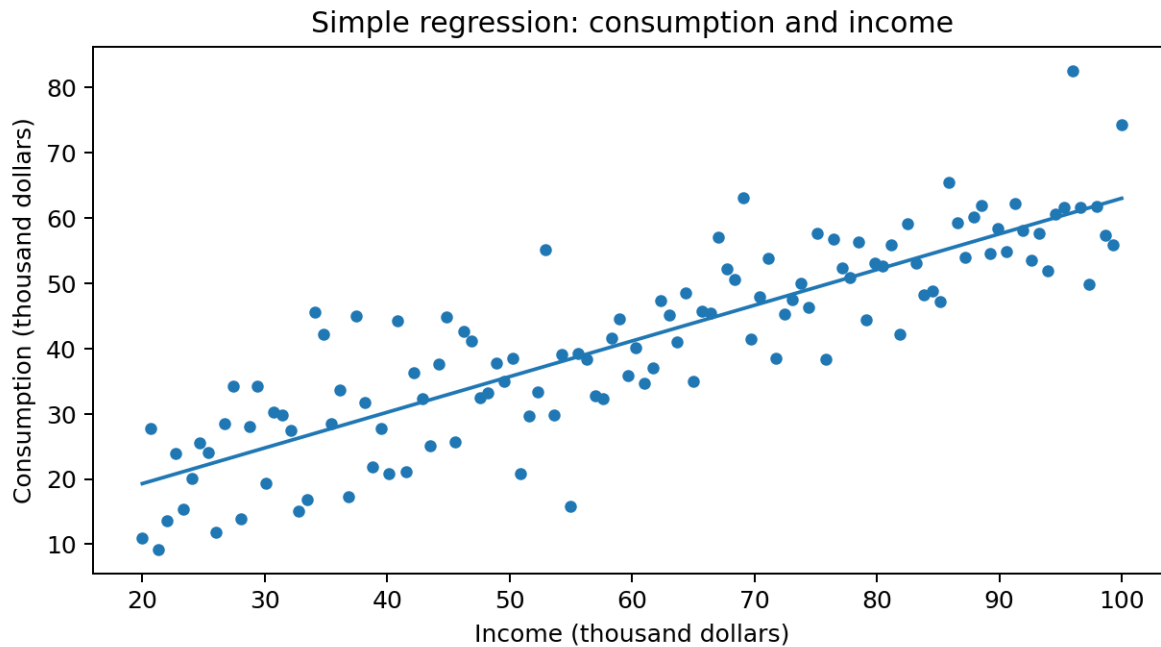


Figure 18.1. How does consumption change with income in a simple model?

18.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

18.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

18.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

18.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

18.10 Chapter summary

Simple Linear Regression is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

18.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

19 Multiple Regression and the Ceteris Paribus Interpretation

How can we compare two people while holding other observed characteristics constant?

19.1 Learning objectives

Explain and apply multiple explanatory variables.

Explain and apply partial effects.

Explain and apply ceteris paribus interpretation.

Explain and apply confounding.

Explain and apply Frisch-Waugh-Lovell intuition.

Explain and apply adjusted versus unadjusted relationships.

Explain and apply nested specifications.

Concept in one sentence: Multiple regression estimates partial associations conditional on the controls included in the model.

19.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about multiple regression and the ceteris paribus interpretation. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

19.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

19.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

19.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(19)
ability=rng.normal(size=300)
educ=.7*ability+rng.normal(size=300)
wage=2+1.2*educ+1.5*ability+rng.normal(size=300)
naive=sm.OLS(wage, sm.add_constant(educ)).fit()
controlled=sm.OLS(wage, sm.add_constant(np.c_[educ,ability])).fit()
print('naive',naive.params[1], 'controlled',controlled.params[1])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

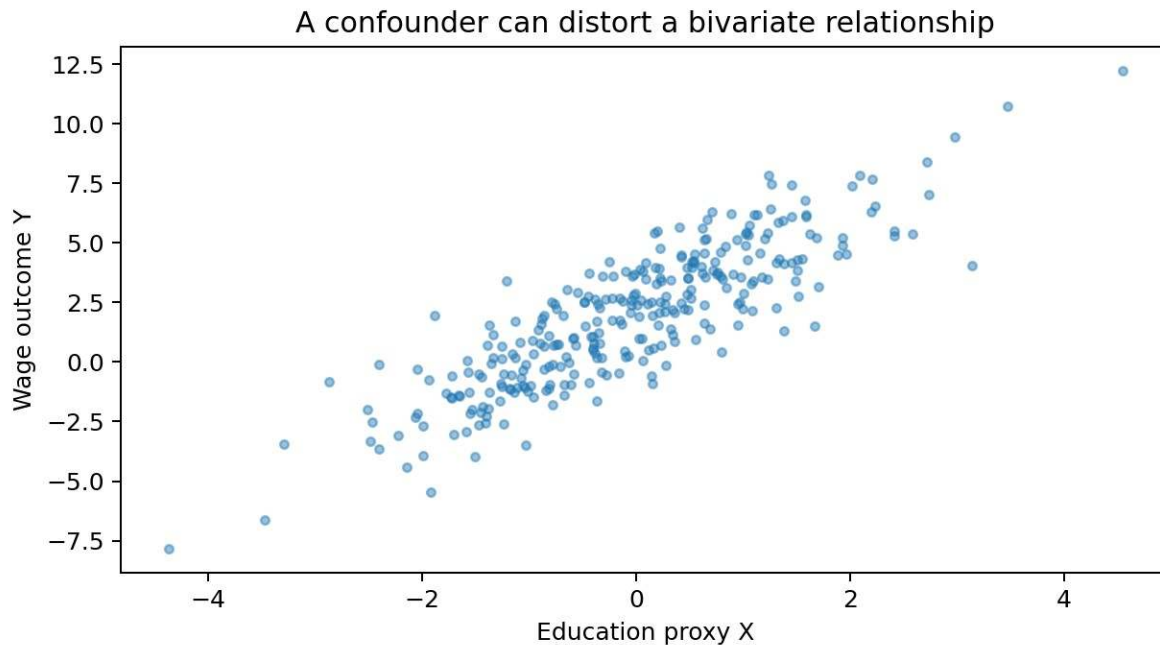


Figure 19.1. How can we compare two people while holding other observed characteristics constant?

19.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

19.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

19.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

19.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

19.10 Chapter summary

Multiple Regression and the Ceteris Paribus Interpretation is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

19.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

20 Functional Forms, Dummy Variables, and Interactions

When is a one-unit change not the right way to describe an economic relationship?

20.1 Learning objectives

Explain and apply log-level, level-log, and log-log models.

Explain and apply elasticities.

Explain and apply quadratic terms.

Explain and apply dummy variables.

Explain and apply reference groups.

Explain and apply interaction terms.

Explain and apply marginal effects.

Concept in one sentence: Functional form determines what a coefficient means, so interpretation must match the transformation.

20.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about functional forms, dummy variables, and interactions. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

20.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

20.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

20.5 Python demonstration

```
import numpy as np, pandas as pd, statsmodels.formula.api as smf
rng=np.random.default_rng(20)
n=300
experience=rng.uniform(0,10,n); group=rng.integers(0,2,n)
wage=20+2*experience+5*group+1.2*experience*group+rng.normal(0,3,n)
df=pd.DataFrame({'wage':wage, 'experience':experience, 'group':group})
print(smf.ols('wage ~ experience * group',data=df).fit().summary())
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

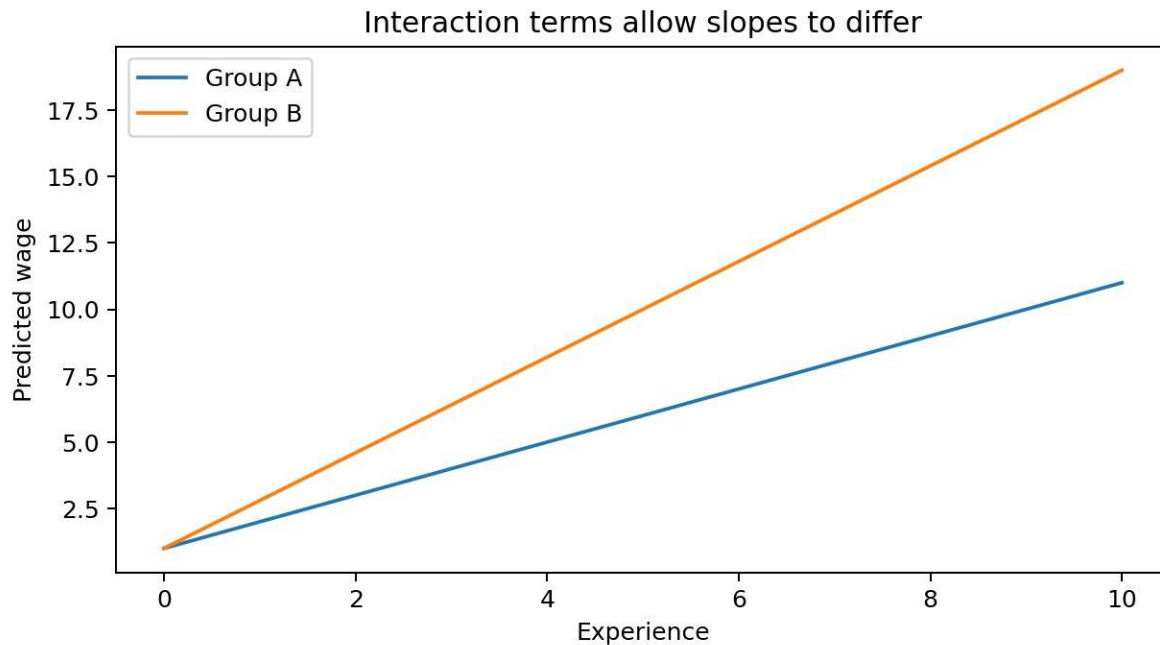


Figure 20.1. When is a one-unit change not the right way to describe an economic relationship?

20.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

20.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

20.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

20.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

20.10 Chapter summary

Functional Forms, Dummy Variables, and Interactions is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

20.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

21 Inference, Robust Standard Errors, and Diagnostics

How certain should we be about an estimated coefficient?

21.1 Learning objectives

Explain and apply sampling uncertainty.

Explain and apply standard errors.

Explain and apply t tests and confidence intervals.

Explain and apply joint F tests.

Explain and apply heteroskedasticity.

Explain and apply HC robust standard errors.

Explain and apply clustered standard errors.

Explain and apply leverage and influence.

Explain and apply multicollinearity.

Concept in one sentence: A coefficient estimate without a defensible uncertainty estimate is only half an empirical result.

21.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about inference, robust standard errors, and diagnostics. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

21.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

21.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

21.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(21)
x=np.linspace(0,10,300)
y=2+.8*x+rng.normal(0,.3+.35*x,300)
ols=sm.OLS(y,sm.add_constant(x)).fit()
robust=ols.get_robustcov_results(cov_type='HC3')
print('classical SE',ols.bse[1], 'HC3 SE',robust.bse[1])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

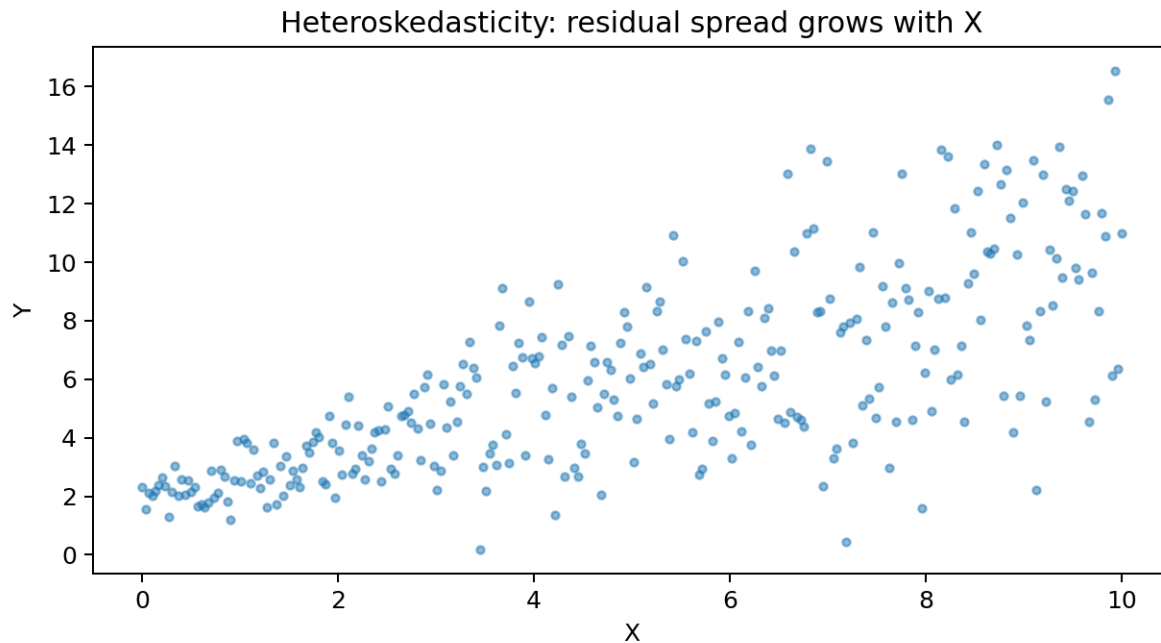


Figure 21.1. How certain should we be about an estimated coefficient?

21.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

21.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

21.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

21.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

21.10 Chapter summary

Inference, Robust Standard Errors, and Diagnostics is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

21.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

22 Omitted Variables, Measurement Error, Simultaneity, and Endogeneity

Why can a beautifully estimated regression still answer the wrong question?

22.1 Learning objectives

Explain and apply omitted-variable bias.

Explain and apply bad controls.

Explain and apply measurement error.

Explain and apply reverse causality.

Explain and apply simultaneity.

Explain and apply selection.

Explain and apply endogeneity.

Explain and apply why adding more controls is not a universal cure.

Concept in one sentence: Endogeneity means the regressor is statistically connected to the structural error, threatening causal interpretation.

22.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about omitted variables, measurement error, simultaneity, and endogeneity. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

22.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

22.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

22.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(22)
z=rng.normal(size=400); v=rng.normal(size=400)
x=.8*z+v; e=.7*v+rng.normal(size=400); y=1+1.5*x+e
print(sm.OLS(y,sm.add_constant(x)).fit().params)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

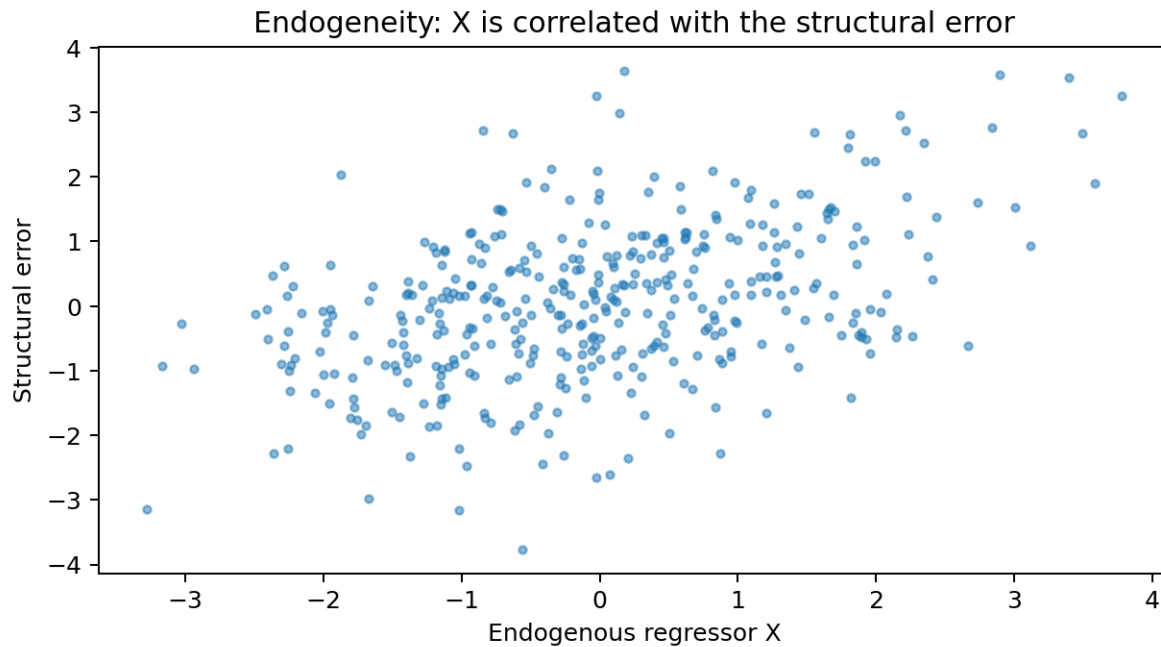


Figure 22.1. Why can a beautifully estimated regression still answer the wrong question?

22.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

22.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

22.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

22.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

22.10 Chapter summary

Omitted Variables, Measurement Error, Simultaneity, and Endogeneity is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

22.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

23 Instrumental Variables and Two-Stage Least Squares

Can we isolate useful variation in an endogenous explanatory variable?

23.1 Learning objectives

Explain and apply instrument relevance.

Explain and apply exclusion restriction.

Explain and apply first stage.

Explain and apply reduced form.

Explain and apply 2SLS.

Explain and apply weak instruments.

Explain and apply local average treatment effects.

Explain and apply diagnostics and interpretation.

Concept in one sentence: An instrument is useful only if it moves the endogenous variable and affects the outcome through the permitted channel.

23.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about instrumental variables and two-stage least squares. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

23.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

23.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

23.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(23)
z=rng.normal(size=500); u=rng.normal(size=500); x=.9*z+u
e=.7*u+rng.normal(size=500); y=1+2*x+e
first=sm.OLS(x,sm.add_constant(z)).fit(); xhat=first.fittedvalues
second=sm.OLS(y,sm.add_constant(xhat)).fit()
print('first-stage slope',first.params[1]); print('educational 2SLS slope',second.params[1])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

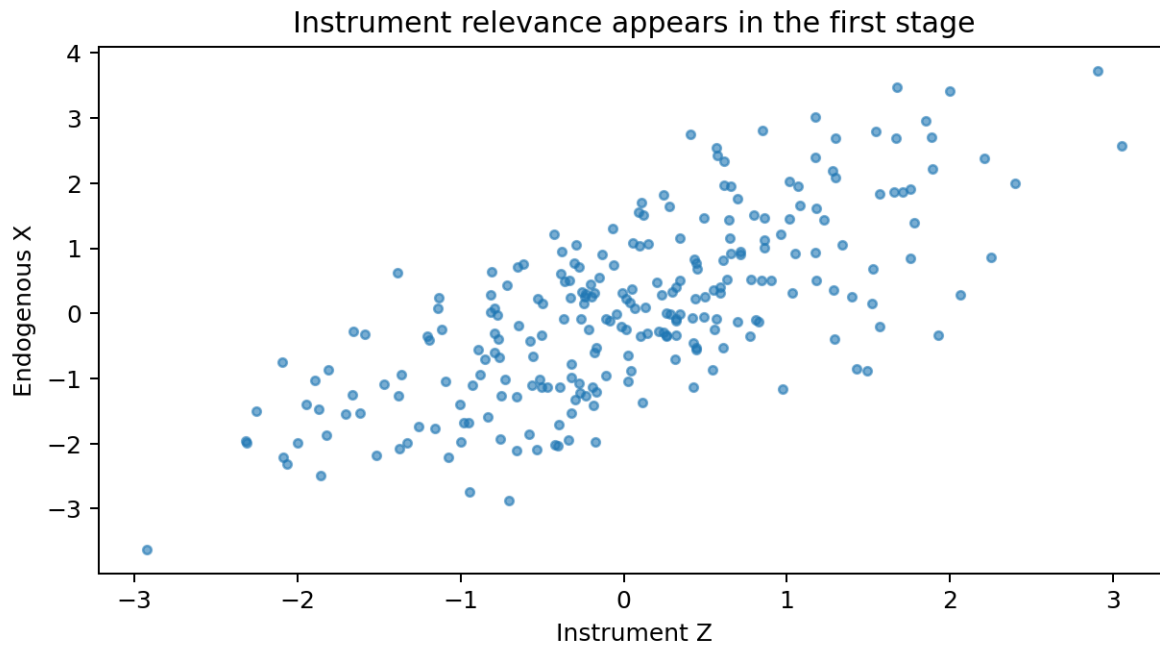


Figure 23.1. Can we isolate useful variation in an endogenous explanatory variable?

23.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

23.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

23.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

23.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

23.10 Chapter summary

Instrumental Variables and Two-Stage Least Squares is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

23.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

24 Panel Data and Fixed Effects

What can repeated observations teach us that a single cross-section cannot?

24.1 Learning objectives

Explain and apply entity and time effects.

Explain and apply pooled OLS.

Explain and apply first differences.

Explain and apply fixed effects.

Explain and apply within transformation.

Explain and apply random effects.

Explain and apply two-way fixed effects.

Explain and apply clustered inference.

Explain and apply unbalanced panels.

Concept in one sentence: Fixed effects use within-entity variation to remove time-invariant unobserved heterogeneity.

24.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about panel data and fixed effects. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

24.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

24.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

24.5 Python demonstration

```
import numpy as np, pandas as pd, statsmodels.formula.api as smf
rng=np.random.default_rng(24); n,t=30,8
id=np.repeat(np.arange(n),t); year=np.tile(np.arange(t),n); a=rng.normal(0,2,n)
x=rng.normal(size=n*t); y=a[id]+.7*x+.2*year+rng.normal(size=n*t)
df=pd.DataFrame({'y':y, 'x':x, 'id':id, 'year':year})
fit=smf.ols('y ~ x + C(id) + C(year)', data=df).fit(cov_type='cluster', cov_kws={'groups':df['id']})
print(fit.params['x'], fit.bse['x'])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

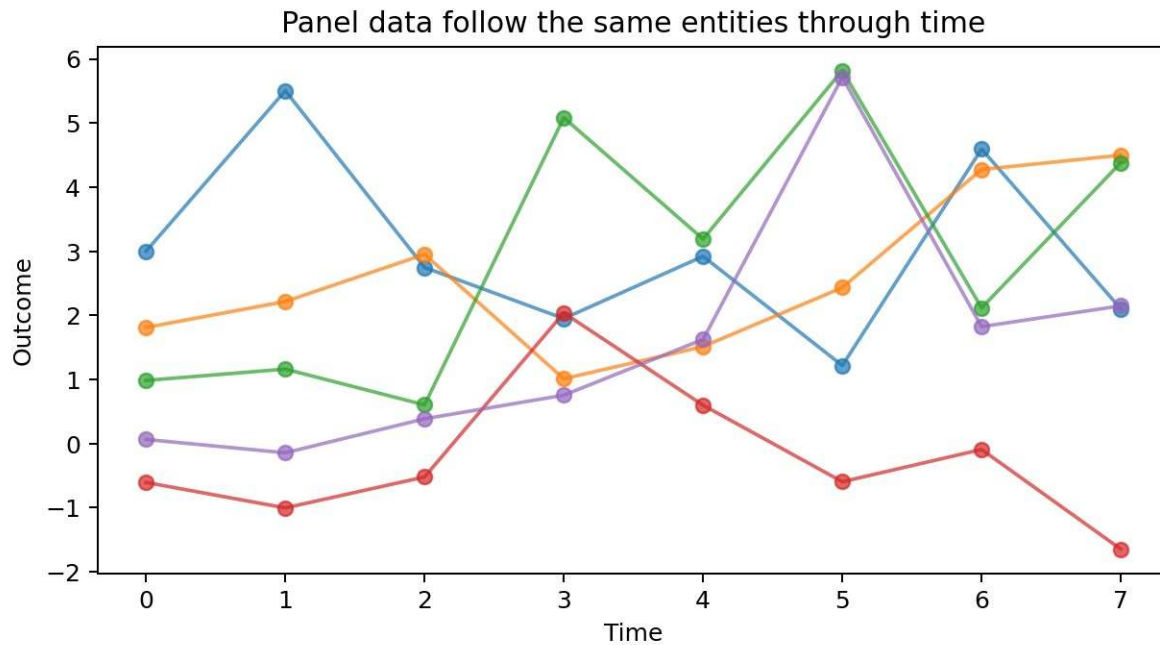


Figure 24.1. What can repeated observations teach us that a single cross-section cannot?

24.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

24.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

24.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

24.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

24.10 Chapter summary

Panel Data and Fixed Effects is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

24.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

25 Binary, Ordered, and Count Outcomes

How should we model outcomes that are probabilities, categories, or event counts?

25.1 Learning objectives

Explain and apply linear probability model.

Explain and apply logit and probit.

Explain and apply odds and marginal effects.

Explain and apply ordered logit and probit.

Explain and apply threshold parameters.

Explain and apply Poisson regression.

Explain and apply negative binomial regression.

Explain and apply overdispersion.

Concept in one sentence: When the dependent variable is discrete, the conditional mean is constrained and interpretation shifts from slopes to probabilities or expected counts.

25.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about binary, ordered, and count outcomes. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

25.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

25.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

25.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(25); x=rng.normal(size=800); p=1/(1+np.exp(-(-.5+1.2*x)));
y=rng.binomial(1,p)
fit=sm.Logit(y,sm.add_constant(x)).fit(dis=False)
print(fit.params); print(fit.get_margeff().summary())
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

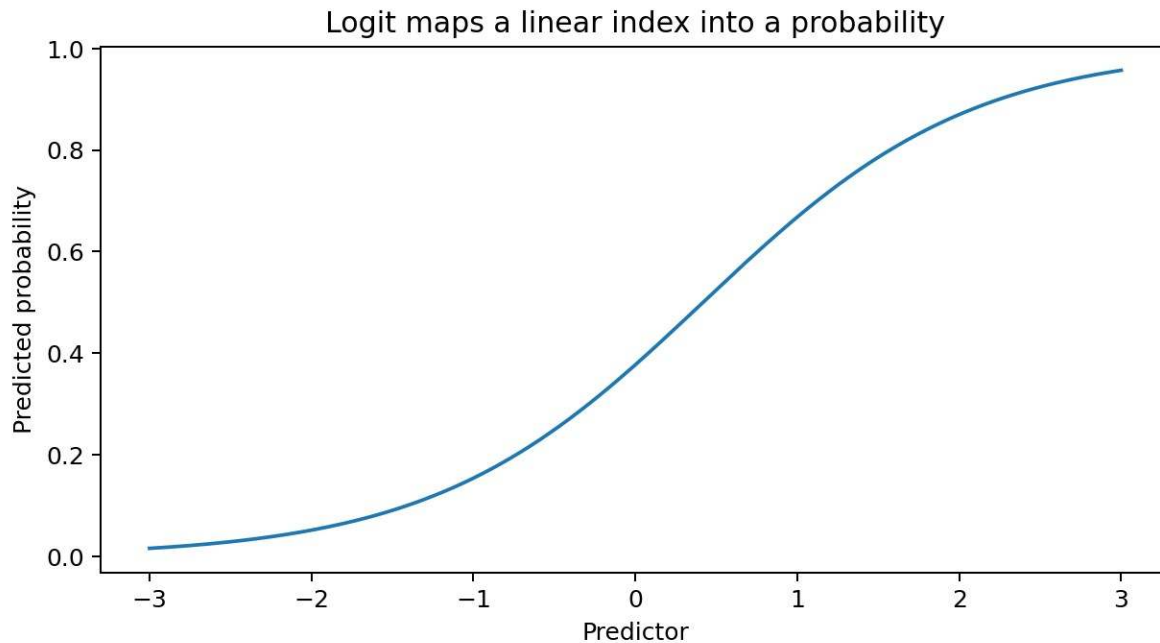


Figure 25.1. How should we model outcomes that are probabilities, categories, or event counts?

25.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

25.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

25.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

25.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

25.10 Chapter summary

Binary, Ordered, and Count Outcomes is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

25.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

26 Censoring, Selection, Quantiles, and Robust Methods

What if the mean is not the whole story, or part of the outcome is unobserved?

26.1 Learning objectives

Explain and apply censoring and truncation.

Explain and apply sample selection.

Explain and apply missing-data mechanisms.

Explain and apply survey weights.

Explain and apply robust regression.

Explain and apply quantile regression.

Explain and apply conditional quantiles.

Explain and apply distributional effects.

Concept in one sentence: Quantile regression asks how covariates relate to different points of the conditional outcome distribution, not just its mean.

26.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about censoring, selection, quantiles, and robust methods. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

26.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

26.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

26.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(26); x=rng.uniform(0,10,500);
y=2+.7*x+rng.standard_t(3,500)*(1+.2*x)
X=sm.add_constant(x)
for q in [.25, .5, .75]:
    fit=sm.QuantReg(y,X).fit(q=q)
    print(q, fit.params)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

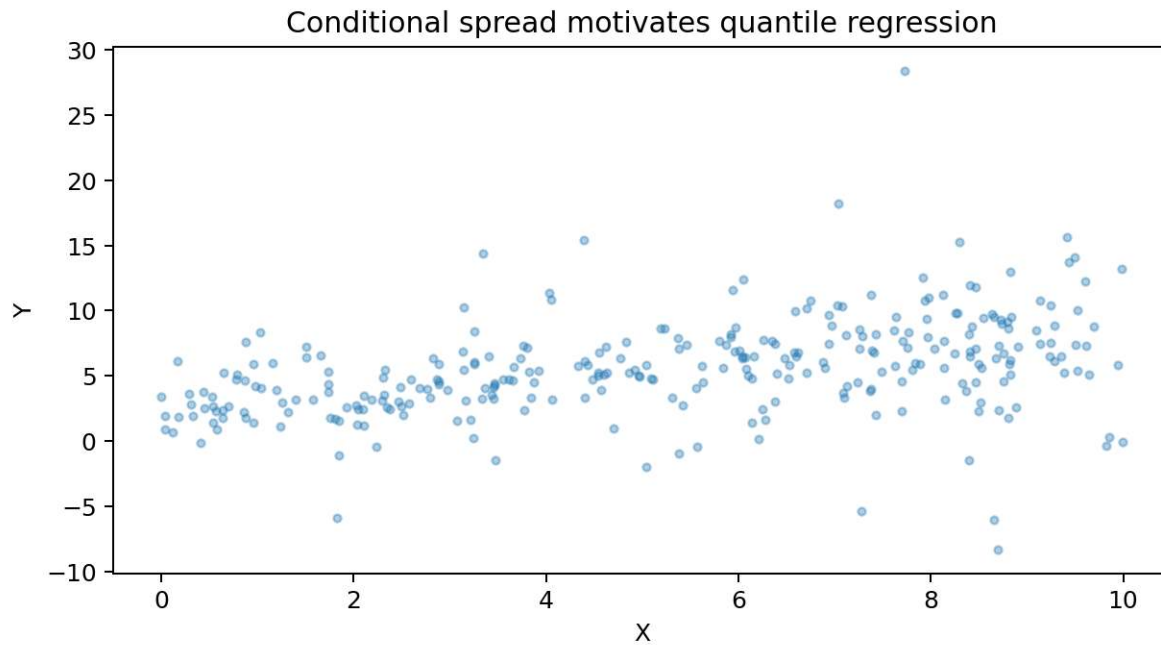


Figure 26.1. What if the mean is not the whole story, or part of the outcome is unobserved?

26.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

26.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

26.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

26.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

26.10 Chapter summary

Censoring, Selection, Quantiles, and Robust Methods is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

26.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

27 Potential Outcomes and Randomized Experiments

What does a causal effect mean for one unit, and why can we never observe both potential outcomes?

27.1 Learning objectives

Explain and apply potential outcomes.

Explain and apply individual and average treatment effects.

Explain and apply fundamental problem of causal inference.

Explain and apply random assignment.

Explain and apply balance.

Explain and apply intention to treat.

Explain and apply compliance and attrition.

Explain and apply internal and external validity.

Concept in one sentence: A causal effect compares two potential outcomes for the same unit, but only one is observed.

27.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about potential outcomes and randomized experiments. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

27.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

27.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

27.5 Python demonstration

```
import numpy as np
rng=np.random.default_rng(27); n=2000; treatment=rng.binomial(1,.5,n);
y0=rng.normal(50,10,n); y=y0+5*treatment+rng.normal(0,3,n)
ate=y[treatment==1].mean()-y[treatment==0].mean()
print('difference in means',ate)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

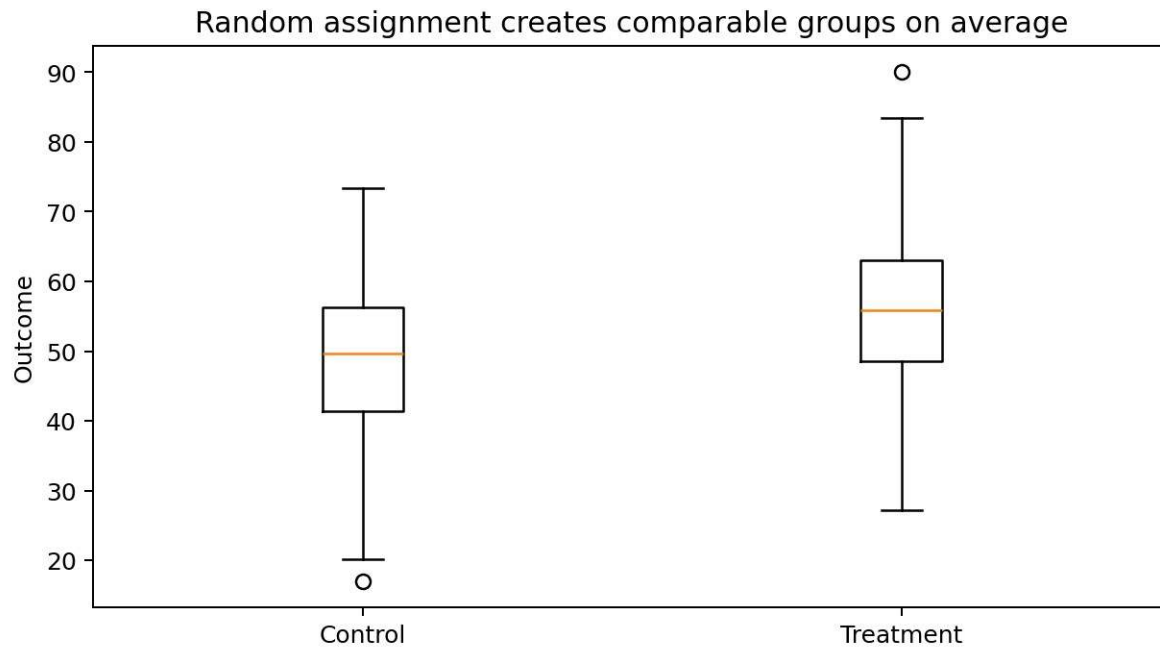


Figure 27.1. What does a causal effect mean for one unit, and why can we never observe both potential outcomes?

27.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

27.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

27.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

27.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

27.10 Chapter summary

Potential Outcomes and Randomized Experiments is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

27.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

28 Matching, Propensity Scores, Weighting, and Doubly Robust Estimation

How can observational studies improve comparability when treatment is not randomized?

28.1 Learning objectives

Explain and apply selection on observables.

Explain and apply propensity score.

Explain and apply overlap.

Explain and apply common support.

Explain and apply matching.

Explain and apply inverse-probability weighting.

Explain and apply covariate balance.

Explain and apply doubly robust estimation.

Explain and apply sensitivity to unobserved confounding.

Concept in one sentence: Propensity methods rebalance observed covariates; they do not eliminate bias from unobserved confounding.

28.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about matching, propensity scores, weighting, and doubly robust estimation. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

28.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

28.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

28.5 Python demonstration

```
import numpy as np
from sklearn.linear_model import LogisticRegression
rng=np.random.default_rng(28); n=1500; x=rng.normal(size=(n,3)); ps=1/(1+np.exp(-(x[:,0]-.5*x[:,1]))); d=rng.binomial(1,ps); y=2*d+x[:,0]+rng.normal(size=n)
logit=LogisticRegression().fit(x,d); phat=logit.predict_proba(x)[:,1]
w=d/phat+(1-d)/(1-phat)
print('IPW ATE', np.average(d*y/phat)-np.average((1-d)*y/(1-phat)))
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

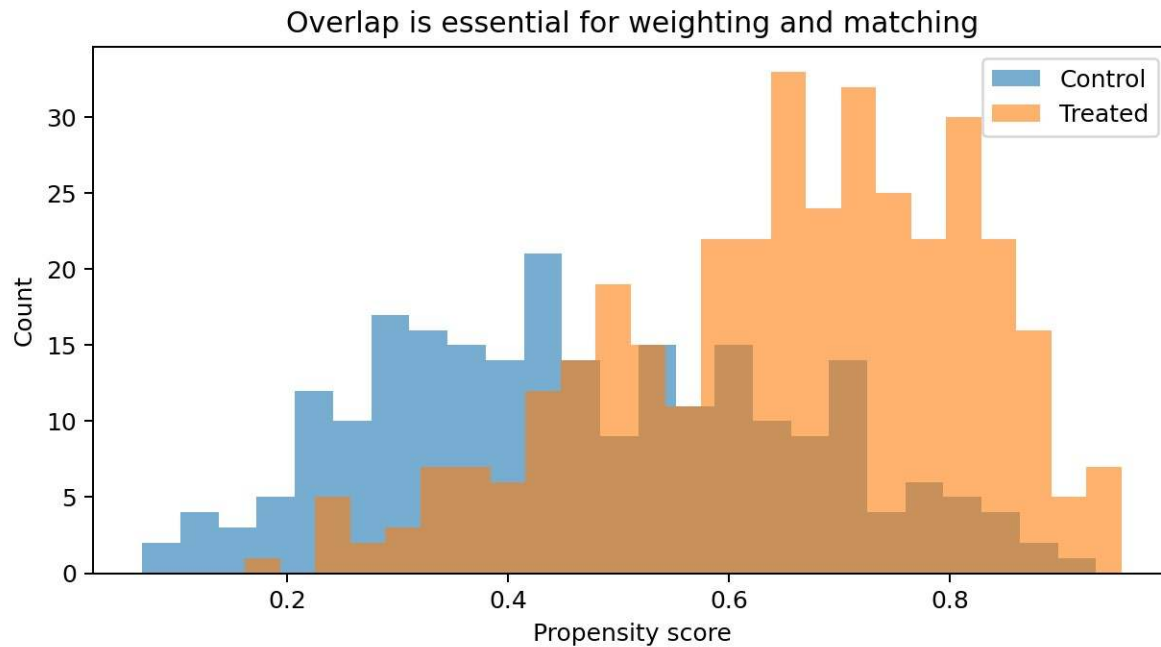


Figure 28.1. How can observational studies improve comparability when treatment is not randomized?

28.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

28.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

28.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

28.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

28.10 Chapter summary

Matching, Propensity Scores, Weighting, and Doubly Robust Estimation is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

28.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

29 Difference-in-Differences and Event Studies

How can policy evaluation use changes over time when randomized assignment is unavailable?

29.1 Learning objectives

Explain and apply treated and comparison groups.

Explain and apply parallel trends.

Explain and apply two-period DID.

Explain and apply regression implementation.

Explain and apply event studies.

Explain and apply pre-trend diagnostics.

Explain and apply staggered adoption caution.

Explain and apply clustered inference.

Concept in one sentence: Difference-in-differences identifies a treatment effect from differential changes under a parallel-trends assumption.

29.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about difference-in-differences and event studies. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

29.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

29.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

29.5 Python demonstration

```
import pandas as pd, statsmodels.formula.api as smf
rows=[]
for g in [0,1]:
    for t in range(8):
        for i in range(40): rows.append((g,t,10+.5*t+g+(3 if (g==1 and t>=4) else 0)))
df=pd.DataFrame(rows,columns=['treated','time','y']); df['post']=(df.time>=4).astype(int)
fit=smf.ols('y ~ treated * post + C(time)',data=df).fit()
print('DID',fit.params['treated:post'])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

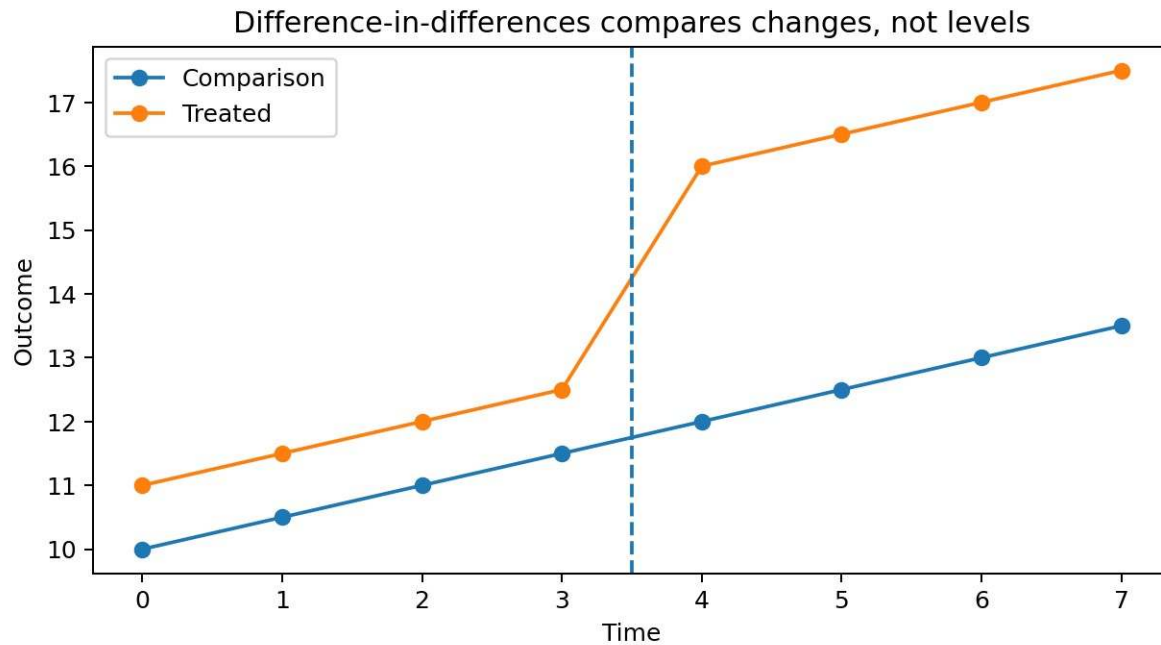


Figure 29.1. How can policy evaluation use changes over time when randomized assignment is unavailable?

29.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

29.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

29.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

29.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

29.10 Chapter summary

Difference-in-Differences and Event Studies is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

29.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

30 Regression Discontinuity

What can a policy cutoff reveal about causal effects near the threshold?

30.1 Learning objectives

Explain and apply running variable.

Explain and apply cutoff.

Explain and apply sharp and fuzzy designs.

Explain and apply local polynomial regression.

Explain and apply bandwidth.

Explain and apply continuity assumption.

Explain and apply manipulation tests.

Explain and apply covariate balance.

Explain and apply local interpretation.

Concept in one sentence: RDD compares units just above and below a threshold where treatment assignment changes discontinuously.

30.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about regression discontinuity. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

30.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

30.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

30.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
rng=np.random.default_rng(30); x=rng.uniform(-1,1,1000); d=(x>=0).astype(int);
y=2+.8*x+2*d+rng.normal(0, .5,1000)
mask=np.abs(x)<=.35; X=sm.add_constant(np.c_[x[mask],d[mask],x[mask]*d[mask]])
print(sm.OLS(y[mask],X).fit().params)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

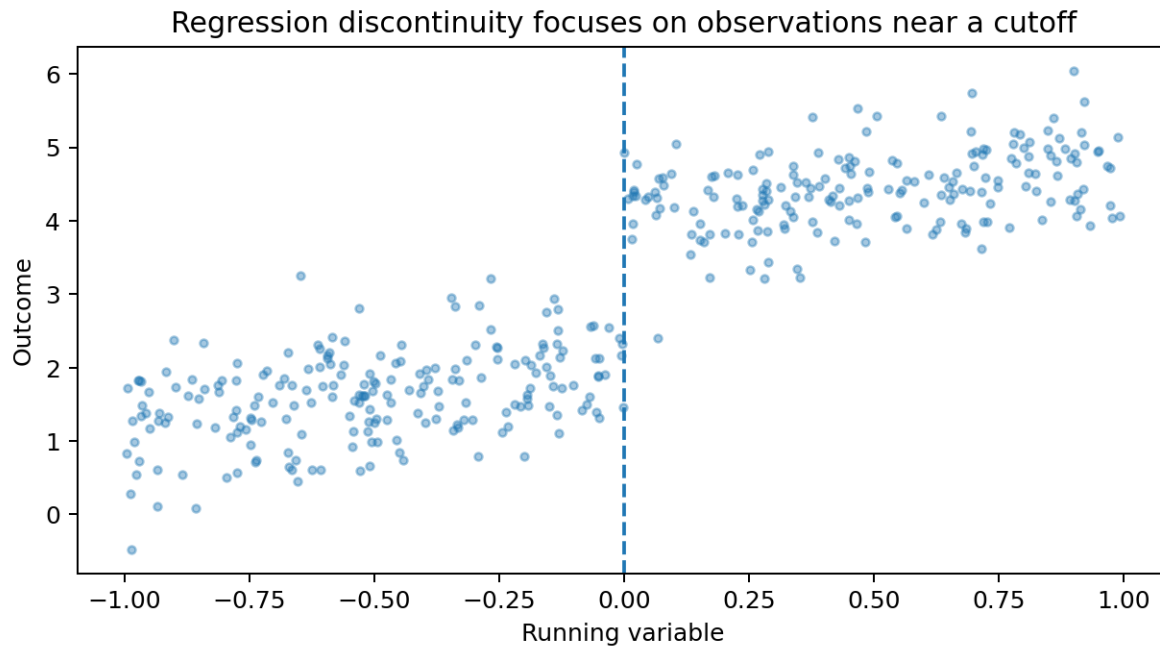


Figure 30.1. What can a policy cutoff reveal about causal effects near the threshold?

30.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

30.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

30.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

30.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

30.10 Chapter summary

Regression Discontinuity is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

30.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

31 Synthetic Control and Comparative Case Studies

How can one treated region be compared with a data-driven combination of untreated regions?

31.1 Learning objectives

Explain and apply treated unit and donor pool.

Explain and apply pre-treatment fit.

Explain and apply synthetic weights.

Explain and apply predictor balance.

Explain and apply post-treatment gaps.

Explain and apply placebo tests.

Explain and apply inference challenges.

Explain and apply policy interpretation.

Concept in one sentence: Synthetic control constructs a transparent counterfactual from weighted comparison units chosen to reproduce the treated unit before intervention.

31.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about synthetic control and comparative case studies. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

31.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made.

Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

31.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

31.5 Python demonstration

```
import numpy as np
rng=np.random.default_rng(31); T=20; donors=np.vstack([10+.2*np.arange(T)+rng.normal(0, .3,T)
for _ in range(4)])
synth=donors.mean(axis=0); treated=synth+rng.normal(0, .15,T); treated[12:]+=2.5
print('average post-treatment gap', (treated[12:]-synth[12:]).mean())
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

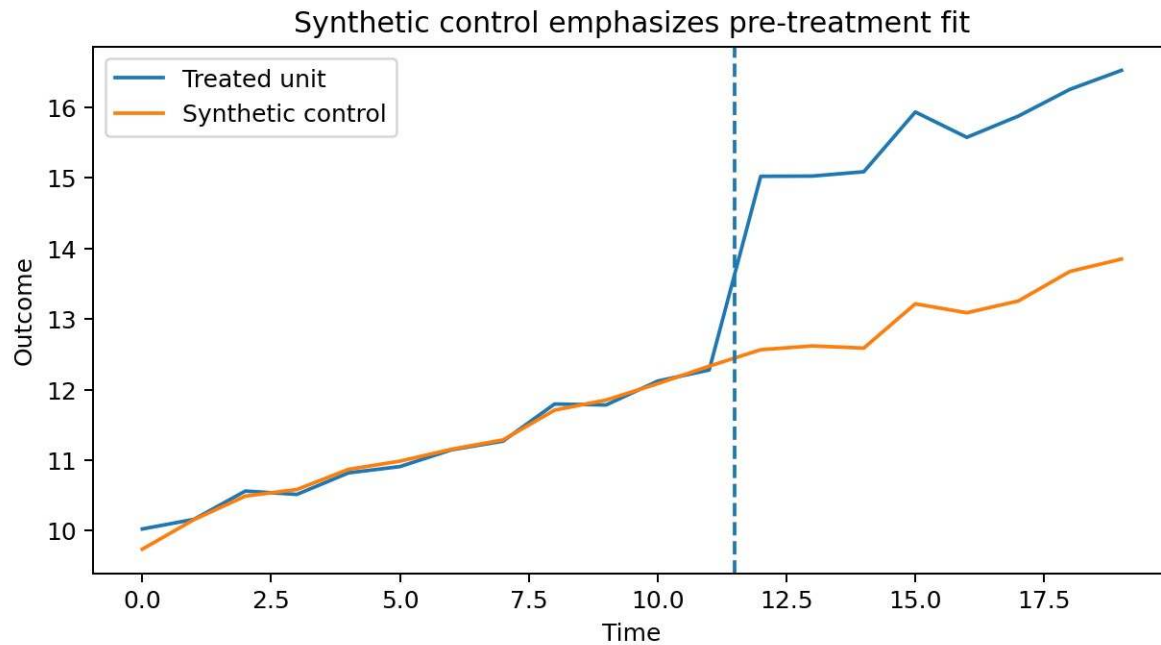


Figure 31.1. How can one treated region be compared with a data-driven combination of untreated regions?

31.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

31.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

31.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

31.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

31.10 Chapter summary

Synthetic Control and Comparative Case Studies is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

31.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

32 Regularization and High-Dimensional Economic Models

How do we build stable predictions when the number of candidate predictors becomes large?

32.1 Learning objectives

Explain and apply ridge regression.

Explain and apply lasso.

Explain and apply elastic net.

Explain and apply standardization.

Explain and apply cross-validation.

Explain and apply shrinkage.

Explain and apply variable selection.

Explain and apply post-selection caution.

Explain and apply prediction versus inference.

Concept in one sentence: Regularization accepts some bias in exchange for lower variance and more stable out-of-sample prediction.

32.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about regularization and high-dimensional economic models. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

32.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

32.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

32.5 Python demonstration

```
import numpy as np
from sklearn.linear_model import RidgeCV, LassoCV
rng=np.random.default_rng(32); X=rng.normal(size=(500,30)); b=np.zeros(30); b[:5]=[2, -1.5, 1, .5, .3]; y=X@b+rng.normal(size=500)
print('ridge alpha',RidgeCV(alphas=np.logspace(-3,3,40)).fit(X,y).alpha_)
print('lasso alpha',LassoCV(cv=5,random_state=1).fit(X,y).alpha_)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

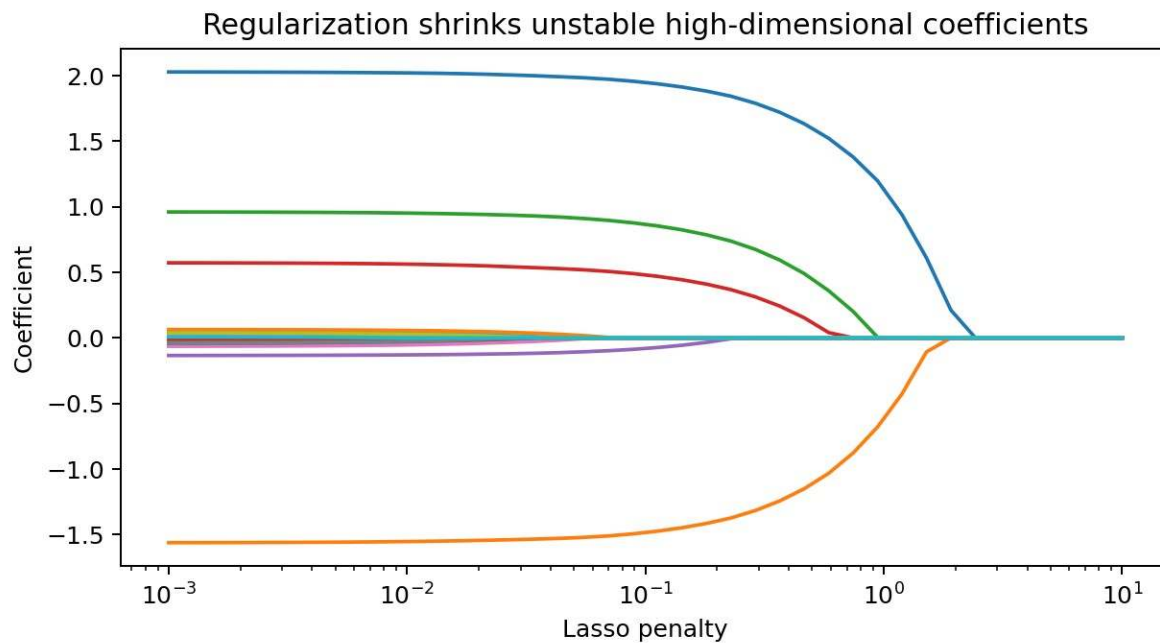


Figure 32.1. How do we build stable predictions when the number of candidate predictors becomes large?

32.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

32.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

32.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

32.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

32.10 Chapter summary

Regularization and High-Dimensional Economic Models is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

32.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

33 Decision Trees, Random Forests, and Boosting for Economic Prediction

Can flexible nonlinear models improve prediction without pretending to identify causal effects?

33.1 Learning objectives

Explain and apply recursive partitioning.

Explain and apply tree depth and pruning.

Explain and apply bagging.

Explain and apply random forests.

Explain and apply gradient boosting.

Explain and apply hyperparameter tuning.

Explain and apply partial dependence.

Explain and apply permutation importance.

Explain and apply extrapolation limits.

Concept in one sentence: Tree ensembles can approximate complex nonlinear prediction functions, but their feature importance is not a causal effect.

33.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about decision trees, random forests, and boosting for economic prediction. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

33.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

33.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

33.5 Python demonstration

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
rng=np.random.default_rng(33); X=rng.normal(size=(1000,8)); y=2*X[:,0]**2+np.sin(X[:,1])
+X[:,2]+rng.normal(size=1000)
Xt,Xv,yt,yv=train_test_split(X,y,test_size=.3,random_state=33)
for m in
[RandomForestRegressor(n_estimators=200,random_state=1),GradientBoostingRegressor(random_stat
e=1)]:
    m.fit(Xt,yt); print(type(m).__name__,mean_squared_error(yv,m.predict(Xv))**.5)
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

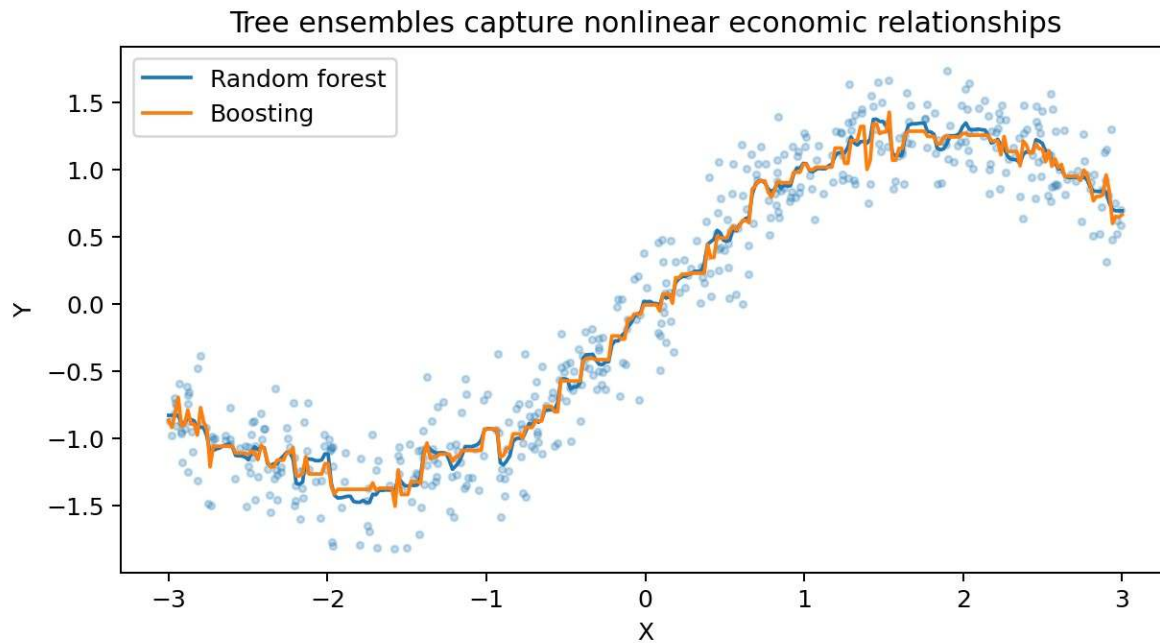


Figure 33.1. Can flexible nonlinear models improve prediction without pretending to identify causal effects?

33.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

33.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

33.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

33.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

33.10 Chapter summary

Decision Trees, Random Forests, and Boosting for Economic Prediction is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

33.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

34 Causal Machine Learning and Double Machine Learning

How can flexible prediction tools help estimate causal parameters without replacing identification assumptions?

34.1 Learning objectives

Explain and apply nuisance functions.

Explain and apply orthogonal scores.

Explain and apply residualization.

Explain and apply cross-fitting.

Explain and apply double machine learning.

Explain and apply heterogeneous treatment effects.

Explain and apply causal forests.

Explain and apply overlap and honest estimation.

Concept in one sentence: Double machine learning uses prediction to remove nuisance structure while protecting a low-dimensional causal target through orthogonalization.

34.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about causal machine learning and double machine learning. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

34.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

34.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

34.5 Python demonstration

```
import numpy as np, statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
rng=np.random.default_rng(34); Z=rng.normal(size=(1000,6)); d=Z[:,0]-
Z[:,1]+rng.normal(size=1000); y=2*d+Z[:,0]+.5*Z[:,2]+rng.normal(size=1000)
md=RandomForestRegressor(n_estimators=200,random_state=1,min_samples_leaf=10).fit(Z,d);
my=RandomForestRegressor(n_estimators=200,random_state=2,min_samples_leaf=10).fit(Z,y)
rd=d-md.predict(Z); ry=y-my.predict(Z); print(sm.OLS(ry,sm.add_constant(rd)).fit().params[1])
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

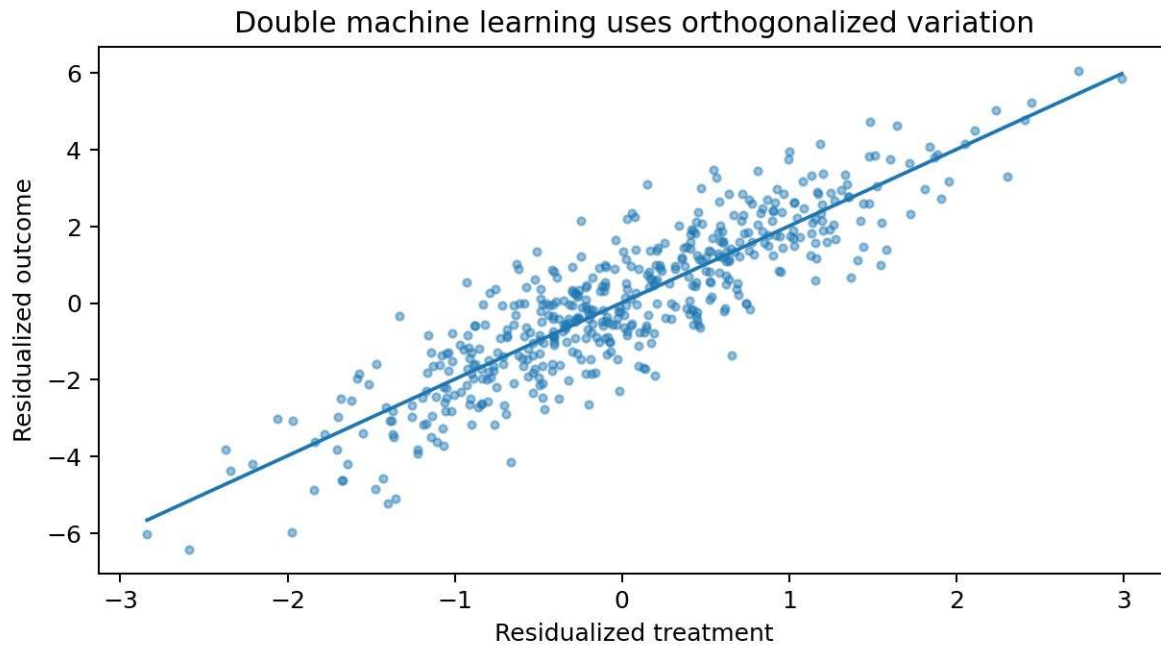


Figure 34.1. How can flexible prediction tools help estimate causal parameters without replacing identification assumptions?

34.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

34.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

34.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

34.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

34.10 Chapter summary

Causal Machine Learning and Double Machine Learning is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

34.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

35 Explainability, Uncertainty, and Model Governance

How do we decide whether a powerful prediction model is trustworthy enough to use?

35.1 Learning objectives

Explain and apply coefficients versus feature importance.

Explain and apply permutation importance.

Explain and apply partial dependence.

Explain and apply SHAP as an optional tool.

Explain and apply calibration.

Explain and apply bootstrap uncertainty.

Explain and apply conformal prediction.

Explain and apply distribution shift.

Explain and apply model cards and audit trails.

Concept in one sentence: Responsible model use requires error analysis, uncertainty, documentation, and governance, not just a high validation score.

35.2 Economic intuition

This chapter begins from the economic question rather than from software syntax. The goal is to understand what information the data can legitimately provide about explainability, uncertainty, and model governance. Python is used as a transparent laboratory: we state the question, define the target, inspect the data, estimate the model, and then challenge the result.

A recurring distinction in econometrics is the difference between describing a pattern, predicting an outcome, and estimating a causal effect. These goals can use similar equations but require different assumptions. A model may forecast well while offering little causal interpretation, and a credible causal design may deliberately use a simple estimator because identification comes from the research design rather than algorithmic complexity.

Throughout the chapter, pay attention to units, timing, sample construction, and the information set available to the analyst. Economic data are produced by institutions, markets, households, and firms. They are not abstract arrays. A good empirical workflow therefore combines statistical discipline with substantive economic reasoning.

35.3 A small numerical example

Start with a tiny hypothetical sample that can be checked by hand. Write the unit of observation, outcome, explanatory variable or treatment, and the comparison being made. Before estimating anything, ask what would make the comparison informative and what could make it misleading.

Observation	X / Treatment	Outcome Y	Interpretive note
1	0	10	baseline
2	0	12	comparison
3	1	15	exposed
4	1	17	exposed

The difference in group averages is easy to compute, but whether it is a causal effect depends on how exposure was assigned and whether the groups are otherwise comparable. This distinction is the heartbeat of modern applied econometrics.

35.4 Formal framework

Write the empirical model in a form that makes the target explicit. A generic conditional-mean representation is

$$E[Y | X] = m(X), \quad \text{or in a linear approximation,} \quad Y_i = \beta_0 + \beta_1 X_i + u_i.$$

Here Y is the outcome, X denotes observed information, β parameters summarize the chosen model, and u collects other determinants of the outcome. The crucial question is not only whether β can be estimated, but what β means under the design and assumptions.

35.5 Python demonstration

```
import numpy as np
rng=np.random.default_rng(35); pred=rng.normal(100,10,300); actual=pred+rng.normal(0,5,300)
err=actual-pred
print('mean error',err.mean()); print('RMSE',np.sqrt(np.mean(err**2))); print('90% empirical
error interval',np.quantile(err,[.05,.95]))
```

Run the code in a clean notebook and inspect both the numerical output and the data-generating assumptions. The associated notebook in the student package reproduces the example with a fixed random seed.

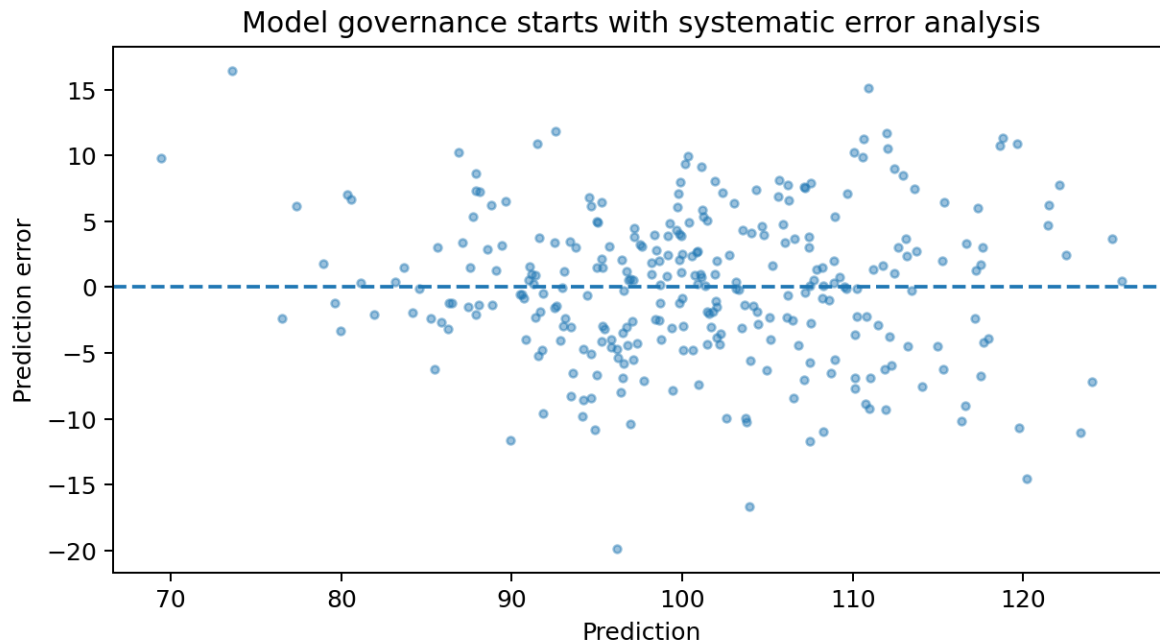


Figure 35.1. How do we decide whether a powerful prediction model is trustworthy enough to use?

35.6 Interpretation

Interpret results in the units of the variables. Report sign, magnitude, uncertainty, and the population or time period to which the estimate applies. If a variable is logged, categorical, interacted, standardized, or transformed, translate the coefficient accordingly rather than reading it as a raw one-unit effect.

Then separate statistical significance from economic significance. A precise estimate may be too small to matter economically; an economically important estimate may be imprecise. Both dimensions belong in the conclusion.

35.7 Assumptions, diagnostics, and threats to validity

Is the unit of observation appropriate for the question?

Are timing and information sets respected?

Could selection, confounding, measurement error, or reverse causality matter?

Does the uncertainty estimator match the sampling or dependence structure?

Would the result survive reasonable alternative specifications?

Is the task predictive, descriptive, or causal?

Causal caution: High predictive accuracy, feature importance, or a small p-value does not by itself identify a causal effect. Causal language requires a defensible identification strategy.

Ceteris LAB Tip: Write the research question and estimand in words before opening Python. If you cannot state what quantity you want to learn, adding packages will not rescue the analysis.

Common mistake: Treating a software default as an econometric assumption. Defaults are conveniences; assumptions must be chosen and defended.

35.8 Guided practice

Modify the notebook so that one assumption is deliberately violated. Re-estimate the model, compare the result, and explain why the estimate changed. This turns diagnostics from a checklist into an experiment.

35.9 Exercises

Concept: In one paragraph, distinguish association, prediction, and causation for the chapter topic.

Python: Reproduce the demonstration with a different documented random seed and verify that the qualitative conclusion is stable.

Applied econometrics: Replace the simulated outcome with a legally shareable economic variable and document its units, source, and sample.

Interpretation: Write a 150-word results paragraph that reports magnitude and uncertainty without exaggerating the evidence.

Research challenge: Propose one alternative design or robustness check that would address the most important threat to validity.

35.10 Chapter summary

Explainability, Uncertainty, and Model Governance is most useful when the economic question, statistical target, assumptions, code, and interpretation point in the same direction. Python makes the workflow reproducible; econometric reasoning determines what the output can mean.

35.11 Key Python tools

Core tools used across these chapters include NumPy for simulation, pandas for data structures, statsmodels for statistical inference, scikit-learn for predictive modeling, and Matplotlib for transparent visual diagnostics. Use the simplest maintained tool that answers the question.

36 Introduction to Time-Series Data

Opening question: **What changes when observations are ordered in time and yesterday can influence today?**

36.1 Learning objectives

- Create and validate time indexes.
- Distinguish levels, changes, growth rates, and returns.
- Resample data with appropriate aggregation.
- Handle missing dates and trading calendars.

Prerequisites: Chapters 9, 10, and 16.

Key terms: time index, frequency, resampling, simple return, log return, adjusted price.

36.2 Time order is part of the model

A time series combines values with an ordered calendar or event index. Shuffling rows destroys lags, seasonality, and forecast origins. Before calculating changes, sort the index, inspect duplicates, and determine whether gaps represent missing data, weekends, holidays, or genuine non-observation. pandas date indexes support resampling, rolling windows, and aligned arithmetic, but the analyst must choose the economically meaningful frequency.

36.3 Levels and changes answer different questions

A price level describes value at a point in time; a simple return measures proportional change; a log return measures the difference in log prices and adds across adjacent periods. Growth rates for macroeconomic variables may be month-over-month, year-over-year, annualized, nominal, or real. These transformations are not interchangeable. The label and formula should travel together.

36.4 Aggregation requires a stock-flow decision

Monthly data derived from daily observations may use a period-end value, average, sum, maximum, or compounded return. Interest rates are often averaged or sampled at period end; transaction volumes are summed; price returns are compounded rather than averaged. Resampling is therefore an economic operation, not merely a date convenience.

36.5 Core equations

Simple return

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

The proportional price change over one period.

Log return

$$r_t = \ln(P_t) - \ln(P_{t-1})$$

Log returns add across adjacent periods and approximate simple returns when changes are small.

36.6 Python demonstrations

36.6.1 Demonstration 17.1: Calculate simple and log returns

```
import numpy as np
import pandas as pd

prices = pd.Series([100, 102, 101, 104], index=pd.date_range("2026-01-01", periods=4,
freq="D"))
simple = prices.pct_change()
log_return = np.log(prices).diff()
print(simple.dropna().round(4).tolist())
print(log_return.dropna().round(4).tolist())
```

Verified output

```
[0.02, -0.0098, 0.0297]
[0.0198, -0.0099, 0.0293]
```

Interpretation. The two measures are close for small changes but are not numerically identical.

36.6.2 Demonstration 17.2: Resample with explicit rules

```
import pandas as pd

idx = pd.date_range("2026-01-01", periods=60, freq="D")
daily = pd.DataFrame({"rate": range(60), "volume": [100]*60}, index=idx)
monthly = daily.resample("ME").agg(rate=("rate", "last"), volume=("volume", "sum"))
print(monthly)
```

Verified output

```
rate volume
2026-01-31  30  3100
2026-02-28  58  2800
2026-03-31  59  100
```

Interpretation. The rate uses the last observation while volume is summed, reflecting different measurement concepts.

36.7 Visual evidence



Figure 18. A price-and-volume panel generated from a deterministic financial simulation.

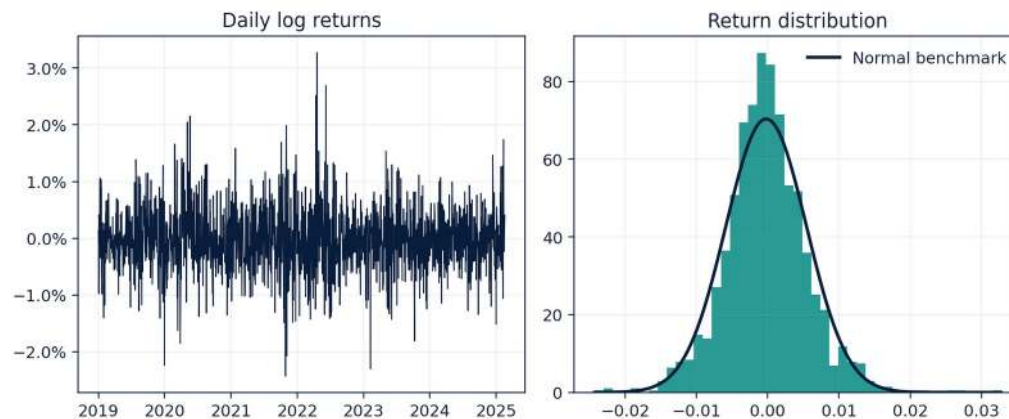


Figure 19. Simulated returns display volatility clustering and heavier tails than a normal benchmark.

36.8 Reference table

Table 17. Chapter reference.

Quantity	Formula or rule	Aggregation
price level	P_t	last or average, depending question
simple return	$P_t/P_{t-1}-1$	compound geometrically
log return	$\log(P_t)-\log(P_{t-1})$	sum

Quantity	Formula or rule	Aggregation
volume	count or quantity	sum
rate	percentage level	average or period end

Why This Matters

Time-series transformations determine what a model is trying to explain or forecast.

Common Mistake

Averaging daily returns to obtain a monthly return. Compounding or summing log returns is the coherent aggregation.

Ceteris LAB Tip

Write the frequency and transformation directly in the variable name or metadata, such as `cpi_yoy_pct`.

R-to-Python / Source Bridge

The first financial time-series lecture introduced prices, returns, exchange rates, interest rates, earnings, claims, and transactions. This chapter modernizes those examples with pandas time indexes and explicit aggregation (Tsay 2013).

36.9 Guided practice

1. Re-run Demonstration 17.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

36.10 Exercises

1. Calculate simple and log returns for a price series.
2. Verify that summed log returns equal the total log return.
3. Resample daily volume to monthly totals.
4. Explain how a missing trading day differs from a missing quote.

36.11 Chapter summary

- Time order and frequency are analytical structure.
- Levels, changes, simple returns, and log returns serve different purposes.
- Aggregation rules depend on whether a variable is a stock, flow, rate, or return.

36.12 Key Python commands

```
import numpy as np
import pandas as pd
simple = prices.pct_change()
log_return = np.log(prices).diff()
print(simple.dropna().round(4).tolist())
print(log_return.dropna().round(4).tolist())
```

37 Dependence, Stationarity, and White Noise

Opening question: **How can we tell whether a time series contains predictable linear structure rather than random fluctuation?**

37.1 Learning objectives

- Define weak stationarity and white noise.
- Compute and interpret ACF and PACF.
- Apply Ljung-Box and unit-root diagnostics.
- Recognize volatility clustering and nonlinear dependence.

Prerequisites: Chapter 17 and basic probability.

Key terms: stationarity, white noise, autocorrelation, partial autocorrelation, Ljung-Box, volatility clustering.

37.2 Stationarity stabilizes the probabilistic frame

A weakly stationary series has a constant mean, constant finite variance, and autocovariances that depend on lag rather than calendar date. Stationarity does not mean that every segment looks identical or that shocks are small. It supplies a stable reference for estimating dependence and long-run variance. Prices often appear nonstationary, while returns are closer to stationary, although their conditional variance may still change over time.

37.3 ACF and PACF are diagnostic fingerprints

The autocorrelation function measures linear association between observations separated by each lag. The partial autocorrelation removes the linear contribution of shorter lags. Sample values fluctuate even under white noise, so isolated spikes should be interpreted with approximate uncertainty bands and domain context. ACF patterns guide model specification but should not replace estimation and residual checks.

37.4 Uncorrelated does not mean independent

Financial returns may show little autocorrelation while squared or absolute returns show strong persistence. The mean can therefore be difficult to predict even when volatility is predictable. Ljung-Box tests summarize groups of autocorrelations, but p-values depend on lag choice and model estimation. Diagnostics should be applied to raw series, residuals, and transformed residuals according to the model question.

37.5 Core equations

Autocorrelation

$$\rho_k = \frac{\text{Cov}(X_t, X_{t-k})}{\text{Var}(X_t)}$$

For a stationary series, the denominator is constant and dependence is indexed by lag k .

37.6 Python demonstrations

37.6.1 Demonstration 18.1: ACF and Ljung-Box

```
import numpy as np
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.stattools import acf

rng = np.random.default_rng(18)
x = rng.normal(size=500)
print(np.round(acf(x, nlags=3, fft=True), 3))
print(round(acorr_ljungbox(x, lags=[10], return_df=True)["lb_pvalue"].iloc[0], 3))
```

Verified output

```
[1.  0.032 0.046 0.033]
0.844
```

Interpretation. The simulated white-noise series has small sample autocorrelations and does not reject joint independence at lag 10 in this run.

37.6.2 Demonstration 18.2: Levels versus differences

```
import numpy as np
from statsmodels.tsa.stattools import adfuller

rng = np.random.default_rng(1818)
level = np.cumsum(rng.normal(size=600))
difference = np.diff(level)
print(round(adfuller(level, regression="c")[1], 4))
print(f'{adfuller(difference, regression="c")[1]:.2e}')
```

Verified output

```
0.8381
0.00e+00
```

Interpretation. The random-walk level is consistent with a unit root, while its first difference is strongly stationary in the ADF diagnostic.

37.7 Visual evidence

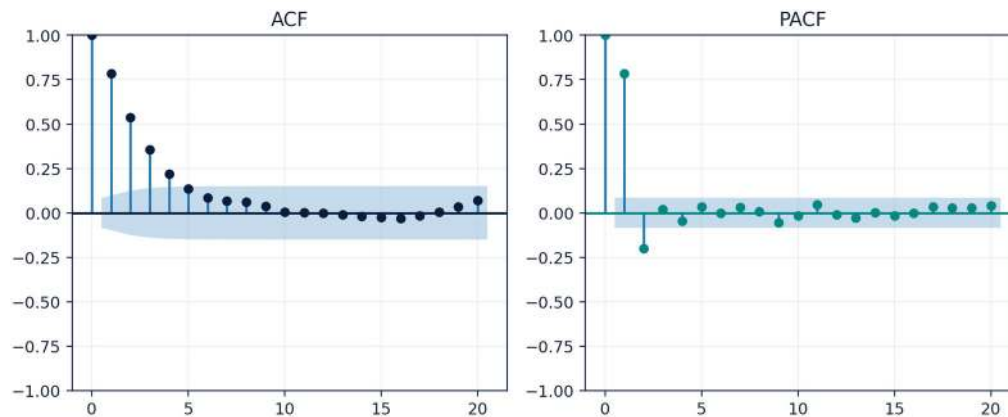


Figure 20. Autocorrelation and partial autocorrelation summarize lag dependence.

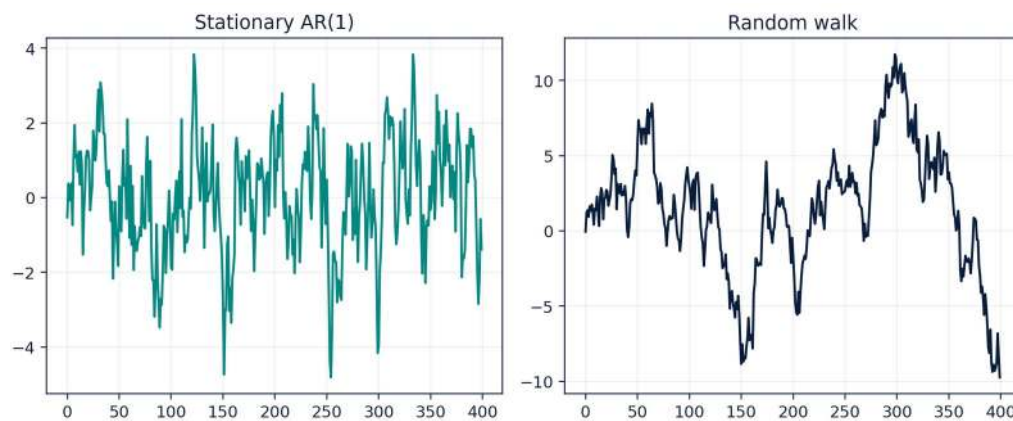


Figure 21. A stationary process fluctuates around a stable distribution; a random walk carries shocks forward.

37.8 Reference table

Table 18. Chapter reference.

Diagnostic	Null idea	Use
ACF	individual lag correlation near zero	pattern discovery
PACF	direct lag contribution near zero	AR order guidance
Ljung-Box	joint zero autocorrelation through lag m	residual adequacy
ADF	unit-root null	nonstationarity diagnostic
ACF of squared returns	no linear volatility dependence	ARCH screening

Why This Matters

Dependence diagnostics tell us whether a model has left systematic temporal structure unexplained.

Common Mistake

Calling a series white noise because its mean is near zero. White noise concerns dependence and variance, not only location.

Ceteris LAB Tip

Inspect the ACF of both returns and squared returns before choosing a conditional-mean or volatility model.

R-to-Python / Source Bridge

These concepts are central to the first and second source lectures. The Python version retains the diagnostic sequence while adding explicit distinction between linear and nonlinear dependence (Tsay 2013).

37.9 Guided practice

1. Re-run Demonstration 18.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

37.10 Exercises

1. Simulate an AR(1) process and compare its ACF with white noise.
2. Apply Ljung-Box at several lags and discuss multiple testing.
3. Compare ADF results for a random walk and its difference.
4. Explain why zero return autocorrelation does not imply independent returns.

37.11 Chapter summary

- Stationarity provides a stable dependence framework.
- ACF, PACF, and Ljung-Box diagnose linear serial structure.
- Volatility dependence may remain when return autocorrelation is weak.

37.12 Key Python commands

```
import numpy as np
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.stattools import acf
rng = np.random.default_rng(18)
x = rng.normal(size=500)
print(np.round(acf(x, nlags=3, fft=True), 3))
```

38 AR, MA, ARMA, ARIMA, and Forecasting

Opening question: **How can past values and past shocks be organized into a model that produces honest forecasts and uncertainty?**

38.1 Learning objectives

- Interpret AR and MA dynamics.
- Use ACF, PACF, AIC, and BIC for model guidance.
- Estimate ARIMA models with statsmodels.
- Evaluate point and interval forecasts out of sample.

Prerequisites: Chapter 18.

Key terms: AR, MA, ARMA, ARIMA, AIC, forecast interval.

38.2 Autoregression describes persistence

An AR model expresses the current value as a linear function of past values and a new shock. In a stationary AR(1), the coefficient controls persistence and the speed of mean reversion. Values near one produce slow decay; negative values produce alternating adjustment. Higher-order AR models can represent richer cycles but also create unstable roots if selected without care.

38.3 Moving averages describe shock memory

An MA model expresses the current observation as a combination of current and past innovations. Innovations are not observed directly and are recovered during estimation. Invertibility allows the model to be represented in terms of observed history. Software conventions differ in the sign assigned to MA coefficients, so equations and package parameterization must be checked before comparing R and Python output.

38.4 Forecasting is an evaluation design

ARIMA combines autoregressive, differencing, and moving-average components. AIC and BIC compare likelihood with complexity penalties, but the lowest in-sample criterion is not guaranteed to forecast best. A credible evaluation reserves later observations, fits only on available history, produces point and interval forecasts, and reports metrics such as MAE or RMSE alongside a naive benchmark. Rolling-origin evaluation better reflects repeated forecasting than one lucky split.

38.5 Core equations

AR(1)

$$X_t - \mu = \phi(X_{t-1} - \mu) + a_t, |\phi| < 1$$

The stationary process reverts toward mean μ at a rate controlled by ϕ .

Half-life

$$h = \frac{\ln(0.5)}{\ln(|\phi|)}$$

For $0 < |\phi| < 1$, h is the approximate number of periods required for a deviation to halve.

38.6 Python demonstrations

38.6.1 Demonstration 19.1: Simulate and fit an ARMA process

```
import numpy as np
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima.model import ARIMA

rng = np.random.default_rng(19)
process = ArmaProcess(ar=[1, -0.65], ma=[1, 0.35])
x = process.generate_sample(500, distrvs=rng.standard_normal)
fit = ARIMA(x, order=(1, 0, 1), trend="c").fit()
print(fit.params.round(3))
```

Verified output

```
[-0.055  0.619  0.394  0.932]
```

Interpretation. The estimates recover the simulated AR and MA structure approximately. Sampling variation and likelihood conventions prevent exact equality.

38.6.2 Demonstration 19.2: Forecast with an interval

```
forecast = fit.get_forecast(steps=5)
mean = forecast.predicted_mean
interval = forecast.conf_int(alpha=0.05)
print(np.round(mean, 3))
print(np.round(interval[0], 3))
```

Verified output

```
[-0.603 -0.394 -0.265 -0.185 -0.136]
[-2.495  1.29 ]
```

Interpretation. The forecast returns toward the estimated mean while uncertainty expands with horizon.

38.7 Visual evidence

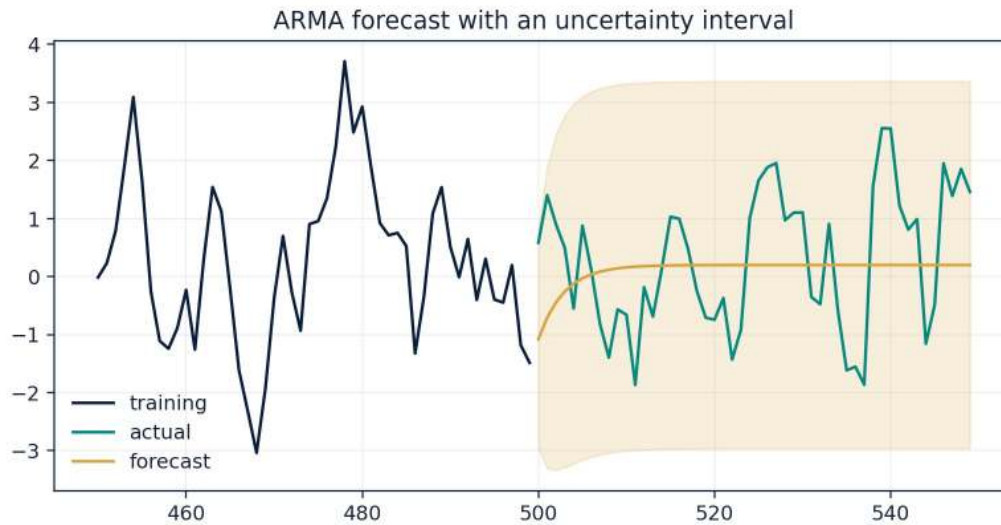


Figure 22. A model forecast is a distribution, not merely a line.

38.8 Reference table

Table 19. Chapter reference.

Pattern	AR suggestion	MA suggestion
ACF tails off; PACF cuts off	AR(p)	not primary
ACF cuts off; PACF tails off	not primary	MA(q)
both tail off	ARMA candidate	ARMA candidate
slow decay in levels	possible unit root	difference before ARMA

Why This Matters

Forecast models are useful only when their information set, benchmark, horizon, and evaluation window match the decision.

Common Mistake

Selecting p and q solely by scanning many specifications on the full sample, then reporting the best fit as an out-of-sample success.

Ceteris LAB Tip

Always compare an ARIMA forecast with a simple benchmark such as the historical mean, last value, or seasonal naive forecast.

R-to-Python / Source Bridge

The source AR lecture develops mean reversion, forecast-error variance, half-life, AIC/BIC, and Ljung-Box checks. The Python implementation uses statsmodels and explicitly documents intercept and MA-sign conventions (Tsay 2013).

38.9 Guided practice

1. Re-run Demonstration 19.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

38.10 Exercises

1. Simulate an AR(1) with ϕ 0.8 and calculate its half-life.
2. Fit ARIMA candidates and compare AIC and BIC.
3. Reserve the last 20 observations for evaluation.
4. Explain why forecast intervals widen with horizon.

38.11 Chapter summary

- AR models encode persistence; MA models encode shock memory.
- ARIMA adds differencing for integrated series.
- Model selection must be followed by residual diagnostics and genuine out-of-sample evaluation.

38.12 Key Python commands

```
import numpy as np
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima.model import ARIMA
rng = np.random.default_rng(19)
process = ArmaProcess(ar=[1, -0.65], ma=[1, 0.35])
x = process.generate_sample(500, distrvs=rng.standard_normal)
```

39 Unit Roots, Seasonality, and Dynamic Regression

Opening question: **How can a model distinguish persistent trend, recurring seasonality, and serially correlated errors?**

39.1 Learning objectives

- Diagnose unit roots and differencing needs.
- Apply seasonal differencing and decomposition.
- Estimate SARIMA and SARIMAX models.
- Interpret regression with time-series errors cautiously.

Prerequisites: Chapters 18 and 19.

Key terms: unit root, seasonality, seasonal difference, SARIMA, SARIMAX, dynamic regression.

39.2 Differencing removes a stochastic trend, not every trend

A unit-root process accumulates shocks, so its level does not revert to a fixed mean. First differencing converts a random walk into its innovations. Over-differencing can create unnecessary negative autocorrelation and discard long-run information. ADF results depend on deterministic terms, lag selection, sample length, and structural breaks; the test is evidence rather than a mechanical command.

39.3 Seasonality is dependence at a calendar rhythm

Monthly and quarterly series may repeat patterns because of weather, institutions, holidays, or reporting cycles. Seasonal decomposition separates estimated trend, seasonal, and residual components for exploration. Seasonal differencing compares an observation with the same season in the previous cycle. A multiplicative seasonal ARIMA model allows regular and seasonal AR and MA components to interact.

39.4 Regression errors can remember the past

OLS coefficients remain a conditional linear fit, but serially correlated residuals invalidate ordinary standard-error formulas and signal omitted dynamics. SARIMAX can estimate regression coefficients jointly with ARIMA errors. The regressors must be available at forecast time, and contemporaneous relationships may still be endogenous. A strong fit between two trending series can be spurious unless nonstationarity is addressed.

39.5 Core equations

Seasonal difference

$$\Delta_s X_t = X_t - X_{t-s}$$

The operator compares the same position in adjacent seasonal cycles.

Airline model

$$(1 - B)(1 - B^s)X_t = (1 - \theta B)(1 - \Theta B^s)a_t$$

A common regular and seasonal MA specification after regular and seasonal differencing.

39.6 Python demonstrations

39.6.1 Demonstration 20.1: Seasonal differencing

```
import pandas as pd

series = pd.Series(range(24), index=pd.date_range("2024-01-01", periods=24, freq="MS"))
seasonal_difference = series.diff(12)
print(seasonal_difference.dropna().unique().tolist())
```

Verified output

```
[12.0]
```

Interpretation. Each month is exactly 12 units above the same month one year earlier in this deterministic illustration.

39.6.2 Demonstration 20.2: Fit a seasonal model

```
import pandas as pd
from statsmodels.tsa.statespace.sarimax import SARIMAX

seasonal = pd.read_csv("data/processed/seasonal_monthly.csv", parse_dates=["date"],
index_col="date")["value"]
fit = SARIMAX(seasonal, order=(1,1,1), seasonal_order=(0,1,1,12), trend="n",
enforce_stationarity=False).fit(dispatch=False)
print(round(fit.aic, 2))
print(fit.get_forecast(3).predicted_mean.round(2).tolist())
```

Verified output

```
313.71
[61.79, 63.49, 64.73]
```

Interpretation. The exact values depend on the included processed series and estimation conventions. Residual diagnostics remain necessary.

39.7 Visual evidence

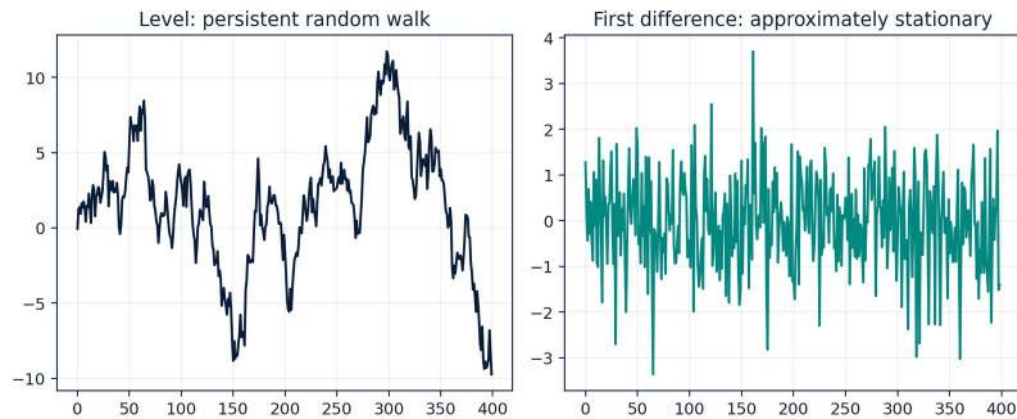


Figure 23. Differencing removes the accumulated component of a random walk.

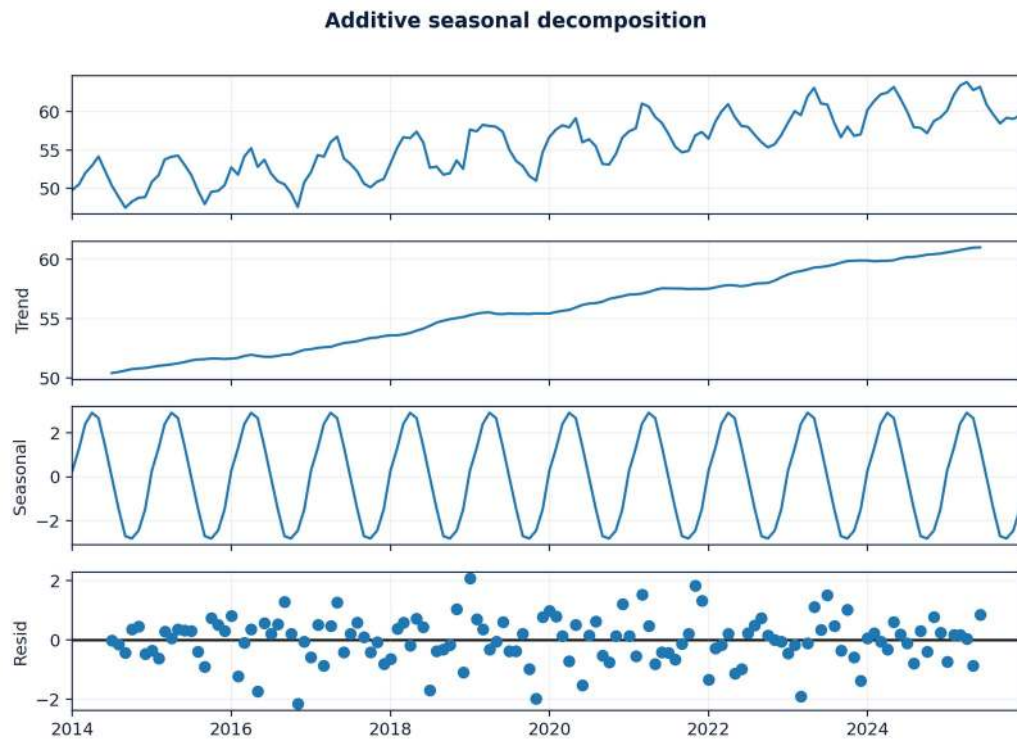


Figure 24. A monthly series decomposed into observed, trend, seasonal, and irregular components.

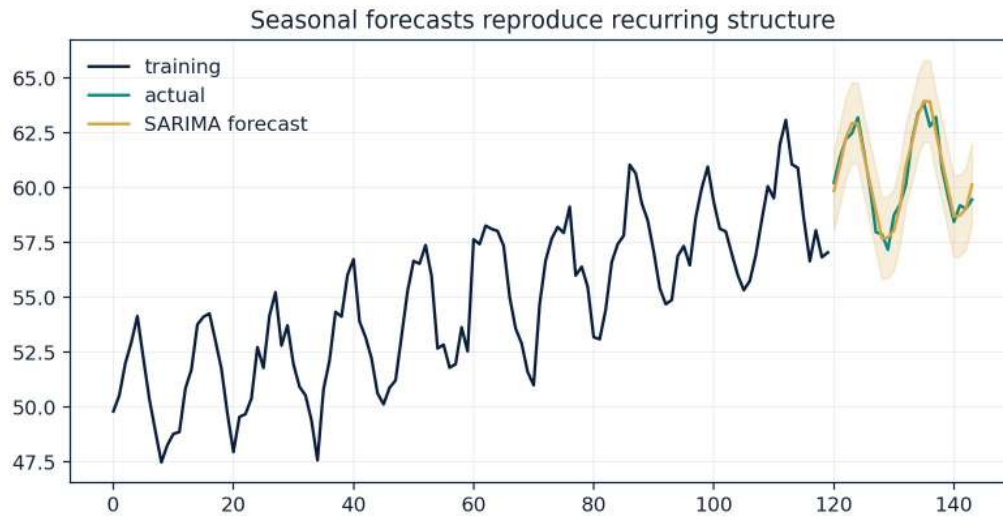


Figure 25. A SARIMA forecast for a synthetic monthly series with yearly seasonality.

39.8 Reference table

Table 20. Chapter reference.

Symptom	Candidate response	Check
random-walk level	first difference	ADF and forecast performance
stable yearly pattern	seasonal difference	seasonal plots and ACF
trend plus changing seasonal amplitude	log transform then seasonal model	residual variance
serial regression residuals	ARIMA errors/SARIMAX	Ljung-Box residuals

Why This Matters

Seasonal and nonstationary structure must be modeled before forecasts or regression standard errors can be trusted.

Common Mistake

Differencing until an ADF p-value crosses a threshold without checking whether the transformed series has an interpretable meaning.

Ceteris LAB Tip

Plot the original, transformed, and seasonally differenced series side by side before selecting a model.

R-to-Python / Source Bridge

The source seasonal lecture covers housing starts, quarterly earnings, the airline model, and regression with serially correlated errors. The new chapter translates those ideas to pandas and SARIMAX while adding benchmark evaluation (Tsay 2013).

39.9 Guided practice

1. Re-run Demonstration 20.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.

3. Add one validation check that would prevent a plausible error.

39.10 Exercises

1. Simulate a random walk and difference it.
2. Create monthly seasonal plots.
3. Fit a seasonal naive benchmark and SARIMA model.
4. Estimate a regression with AR(1) errors and compare residual diagnostics.

39.11 Chapter summary

- Unit roots create persistent stochastic trends.
- Seasonal differencing targets recurring calendar dependence.
- Dynamic regression models coefficients and serial errors jointly.

39.12 Key Python commands

```
import pandas as pd
series = pd.Series(range(24), index=pd.date_range("2024-01-01", periods=24, freq="MS"))
seasonal_difference = series.diff(12)
print(seasonal_difference.dropna().unique().tolist())
import pandas as pd
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

40 ARCH and GARCH Models

Opening question: **How can returns be difficult to predict while their volatility remains persistent and forecastable?**

40.1 Learning objectives

- Explain conditional variance and volatility clustering.
- Test for ARCH effects.
- Estimate ARCH and GARCH models.
- Diagnose standardized residuals and forecast volatility.

Prerequisites: Chapters 18 and 19.

Key terms: conditional variance, ARCH, GARCH, persistence, standardized residual, volatility forecast.

40.2 Volatility is latent and conditional

Volatility is not directly observed. In ARCH-type models it is the conditional standard deviation of a return innovation given past information. Large shocks tend to cluster because yesterday's squared innovation helps predict today's variance. Returns can be nearly uncorrelated while squared returns have substantial autocorrelation. This nonlinear dependence is one of the central facts of financial time series (Engle 1982; Bollerslev 1986).

40.3 GARCH compresses a long memory

An ARCH model uses lagged squared shocks. GARCH adds lagged conditional variance, allowing a parsimonious model to represent slowly decaying volatility. In GARCH(1,1), α measures the immediate reaction to news and β carries previous variance forward. The sum $\alpha + \beta$ summarizes persistence under common parameterizations. Values near one imply slow decay, but unit scaling and innovation distribution affect the numerical estimates.

40.4 Model checking moves to standardized residuals

After estimation, residuals are divided by their conditional standard deviations. An adequate mean and variance model should leave little autocorrelation in standardized residuals and their squares. Heavy tails may remain under Gaussian innovations, motivating a standardized Student-t distribution. A volatility forecast is conditional on the model, estimated parameters, return scale, and information available at the forecast origin.

40.5 Core equations

GARCH(1,1)

$$a_t = \sigma_t z_t, \sigma_t^2 = \omega + \alpha a_{t-1}^2 + \beta \sigma_{t-1}^2$$

With standardized innovations z_t , the conditional variance updates from news and prior variance.

Unconditional variance

$$\text{Var}(a_t) = \frac{\omega}{1 - \alpha - \beta}$$

This expression requires $\alpha + \beta < 1$ under the standard covariance-stationary specification.

40.6 Python demonstrations

40.6.1 Demonstration 21.1: Fit the educational GARCH implementation

```
import sys
from pathlib import Path
import pandas as pd

sys.path.insert(0, str(Path("scripts").resolve()))
from ceteris_examples import fit_garch11

returns = pd.read_csv("data/processed/simulated_garch_returns.csv")["return"].to_numpy()
fit = fit_garch11(returns)
print(round(fit["alpha"] + fit["beta"], 5), fit["success"])
```

Verified output

```
0.94347 True
```

Interpretation. The custom estimator is included for transparency and teaching. Production work should compare a maintained package and verify parameterization.

40.6.2 Demonstration 21.2: Annualize conditional volatility

```
import numpy as np

daily_sigma = np.array([0.008, 0.012, 0.010])
annualized = daily_sigma * np.sqrt(252)
print(np.round(annualized, 3))
```

Verified output

```
[0.127 0.19 0.159]
```

Interpretation. Square-root-of-time annualization assumes a daily variance scale and should not be applied blindly when dependence or horizon dynamics matter.

40.7 Visual evidence

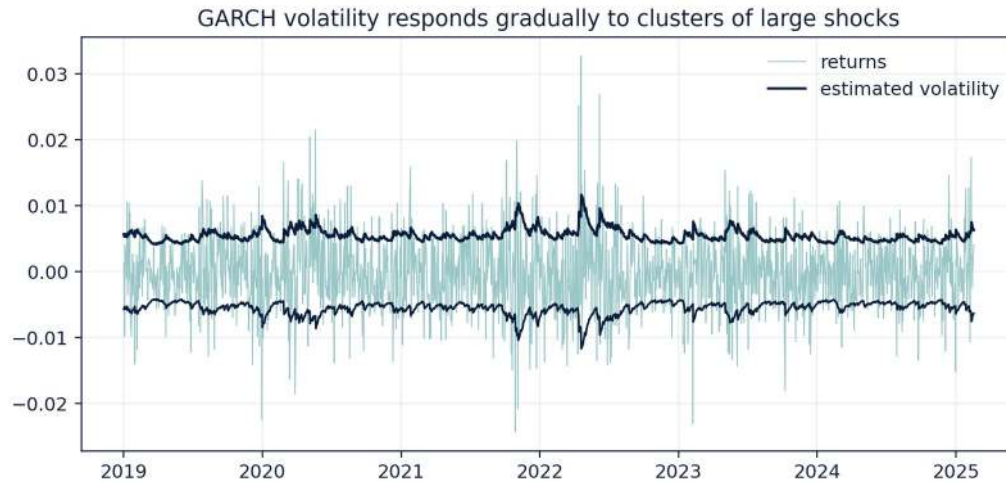


Figure 26. A custom educational Gaussian GARCH(1,1) fit to simulated heavy-tailed returns.

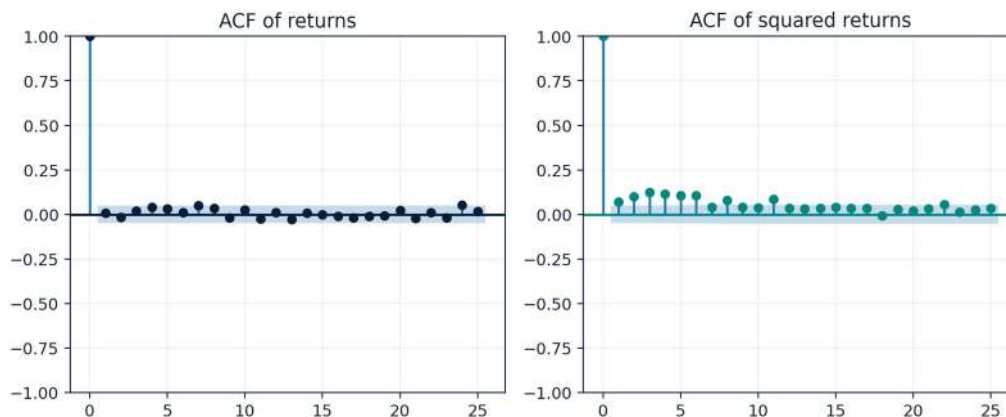


Figure 27. Returns may have little linear autocorrelation while squared returns remain serially dependent.

40.8 Reference table

Table 21. Chapter reference.

Parameter	Role	Interpretation caution
omega	long-run variance anchor	depends on return scaling
alpha	news response	not a causal effect of news sign
beta	variance persistence	can absorb misspecification
alpha+beta	overall persistence	near one may signal regime change
nu	Student-t tail shape	parameterization differs by package

Why This Matters

Volatility forecasts feed risk measures, derivative models, interval forecasts, and portfolio decisions.

Common Mistake

Fitting GARCH to raw prices rather than a stationary return or innovation series.

Ceteris LAB Tip

Scale daily returns to percentages when numerical optimization is unstable, then document and reverse the scale for interpretation.

R-to-Python / Source Bridge

The source volatility lecture builds ARCH/GARCH from squared-return diagnostics, Gaussian and Student-t innovations, and standardized-residual checks. The Python chapter preserves that structure and records scaling differences (Tsay 2013).

40.9 Guided practice

1. Re-run Demonstration 21.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

40.10 Exercises

1. Test squared returns for serial dependence.
2. Estimate Gaussian and Student-t GARCH models.
3. Calculate persistence and unconditional variance.
4. Inspect ACFs of standardized residuals and their squares.

40.11 Chapter summary

- ARCH and GARCH model conditional variance, not observed volatility itself.
- GARCH persistence combines reaction and carryover.
- Diagnostics focus on standardized residuals and squared standardized residuals.

40.12 Key Python commands

```
import sys
from pathlib import Path
import pandas as pd
sys.path.insert(0, str(Path("scripts").resolve()))
from ceteris_examples import fit_garch11
returns = pd.read_csv("data/processed/simulated_garch_returns.csv")["return"].to_numpy()
```

41 Asymmetric, Advanced, and Realized Volatility

Opening question: **Why can equally large negative and positive shocks have different volatility consequences, and what can intraday or OHLC data add?**

41.1 Learning objectives

- Compare EGARCH, GJR-GARCH, APARCH, and GARCH-M concepts.
- Interpret leverage and news-impact curves.
- Calculate rolling, EWMA, realized, and range-based volatility.
- Discuss microstructure noise and overnight information.

Prerequisites: Chapter 21.

Key terms: EGARCH, GJR-GARCH, APARCH, leverage effect, realized volatility, OHLC estimator.

41.2 Asymmetry belongs in the variance equation

Negative equity shocks often increase subsequent volatility more than positive shocks of the same magnitude. EGARCH models log variance and permits asymmetric standardized-shock effects without imposing positivity constraints directly (Nelson 1991). GJR-GARCH adds an indicator for negative innovations. APARCH generalizes the power applied to volatility and shock magnitude. Parameter names and signs differ across software, so interpretation should follow the exact implemented equation.

41.3 Realized measures use more of the day

Realized variance sums squared intraday returns. Under ideal continuous sampling it approximates quadratic variation, but extremely high frequency introduces bid-ask bounce and other microstructure noise. Overnight price changes carry information that an intraday sum can miss. Sampling intervals, market hours, and data cleaning therefore become part of the volatility estimator.

41.4 OHLC estimators trade assumptions for efficiency

High, low, open, and close prices contain more intraday information than close-to-close returns. Parkinson, Garman-Klass, Rogers-Satchell, and Yang-Zhang estimators use different combinations and assumptions about drift and overnight gaps (Garman and Klass 1980; Yang and Zhang 2000). Corporate actions must be handled consistently. Comparing several measures is usually more informative than declaring one universally best.

41.5 Core equations

Realized variance

$$RV_t = \sum_{j=1}^m r_{t,j}^2$$

Intraday squared returns are summed within a day.

Parkinson variance

$$\hat{\sigma}_{P,t}^2 = \frac{[\ln(H_t/L_t)]^2}{4 \ln 2}$$

The high-low range provides an efficient estimator under a driftless diffusion idealization.

41.6 Python demonstrations

41.6.1 Demonstration 22.1: EWMA variance recursion

```
import numpy as np
returns = np.array([0.01, -0.02, 0.005, 0.015])
lam = 0.94
variance = returns.var()
path = []
for r in returns:
    variance = lam * variance + (1-lam) * r**2
    path.append(variance)
print(np.round(np.sqrt(path), 4))
```

Verified output

```
[0.0133 0.0138 0.0134 0.0135]
```

Interpretation. Recent squared returns receive greater weight, while the recursion never fully forgets earlier variance.

41.6.2 Demonstration 22.2: Parkinson and Garman-Klass estimates

```
import numpy as np
open_, high, low, close = 100, 105, 98, 103
hl = np.log(high/low)
co = np.log(close/open_)
parkinson = hl**2 / (4*np.log(2))
gk = 0.5*hl**2 - (2*np.log(2)-1)*co**2
print(round(parkinson, 6), round(gk, 6))
```

Verified output

```
0.001717 0.002042
```

Interpretation. The estimators use the same OHLC day but weight its range and open-to-close movement differently.

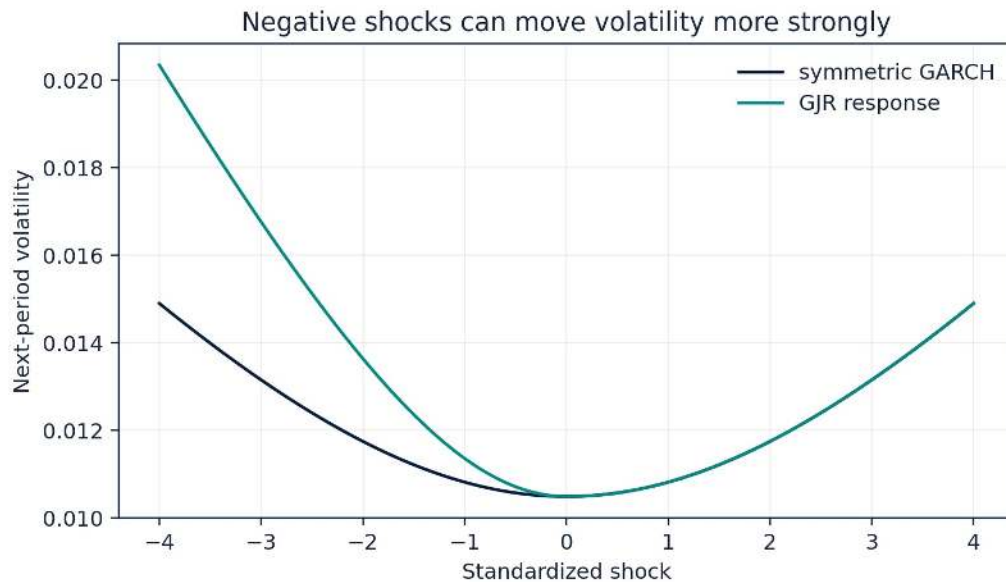
41.7 Visual evidence

Figure 28. News-impact curves for symmetric and asymmetric volatility models.

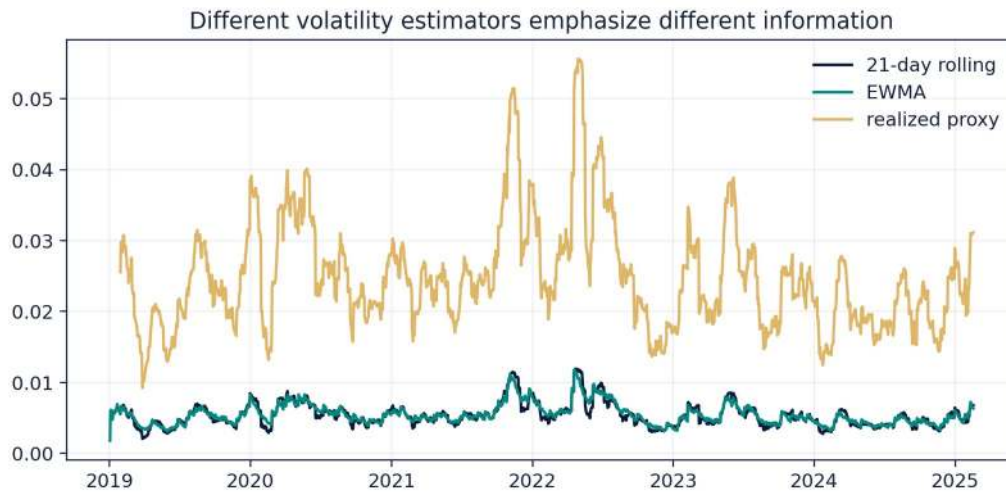


Figure 29. Rolling, exponentially weighted, and realized-volatility proxies for the same return series.

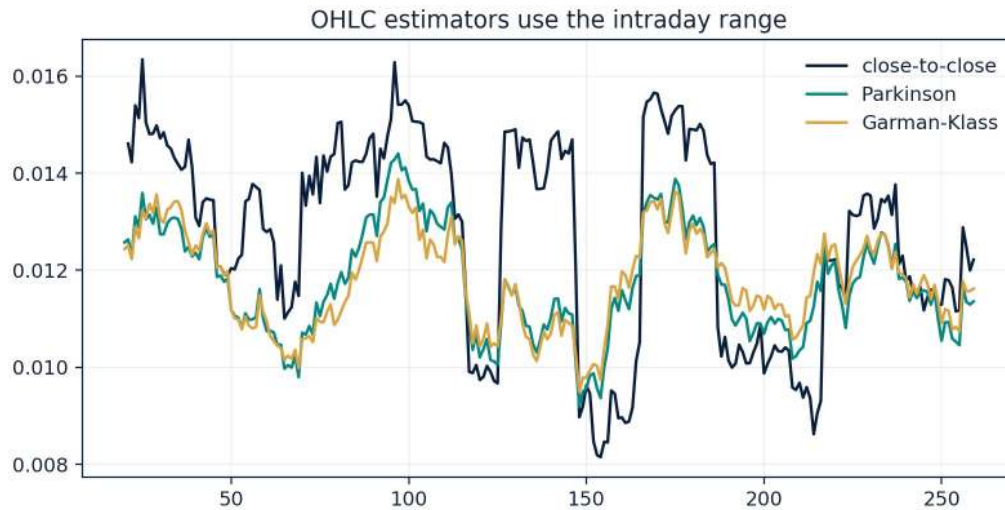


Figure 30. Close-to-close, Parkinson, and Garman-Klass volatility estimates from simulated OHLC prices.

41.8 Reference table

Table 22. Chapter reference.

Method	Information used	Main limitation
rolling SD	recent close returns	window choice
EWMA	all past returns with decay	fixed decay parameter
GARCH	model-based shocks and variance	specification risk
realized variance	intraday returns	microstructure noise and overnight gap
Yang-Zhang	open/high/low/close	corporate actions and assumptions

Why This Matters

Alternative volatility measures reveal how much the result depends on data frequency, asymmetry, and model assumptions.

Common Mistake

Calling every negative-return coefficient a “leverage effect” without checking the package equation and indicator definition.

Ceteris LAB Tip

Draw a news-impact curve from the fitted equation. The picture often clarifies asymmetry better than a coefficient table.

R-to-Python / Source Bridge

The source “More Volatility Models” and “Alternative Approaches” lectures cover GARCH-M, EGARCH, GJR/TGARCH, APARCH, realized volatility, and OHLC estimators. This chapter consolidates them into one tested Python workflow (Tsay 2013).

41.9 Guided practice

1. Re-run Demonstration 22.1 and change one input while keeping the analytical question fixed.

2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

41.10 Exercises

1. Calculate a rolling and EWMA volatility series.
2. Compare Gaussian GARCH and GJR-GARCH.
3. Compute realized variance at two sampling intervals.
4. Implement Parkinson and Yang-Zhang estimators on adjusted OHLC data.

41.11 Chapter summary

- Asymmetric models allow shock sign to affect future variance.
- Realized volatility uses intraday information but faces microstructure noise.
- Range-based estimators improve information use under specific assumptions.

41.12 Key Python commands

```
import numpy as np
returns = np.array([0.01, -0.02, 0.005, 0.015])
variance = returns.var()
variance = lam * variance + (1-lam) * r**2
path.append(variance)
print(np.round(np.sqrt(path), 4))
```

42 Nonlinear Models, Regimes, Market Microstructure, and Ordered Outcomes

Opening question: **What if the same lag has a different effect in calm and stressed regimes, or the observed price change is an ordered category?**

42.1 Learning objectives

- Simulate and interpret threshold autoregression.
- Explain Markov-switching regimes and expected duration.
- Describe market-microstructure effects.
- Estimate and interpret ordered probit probabilities.

Prerequisites: Chapters 18, 19, and 22.

Key terms: TAR, regime, Markov switching, bid-ask bounce, ordered probit, threshold.

42.2 Threshold models make dynamics piecewise

A threshold autoregression applies different linear equations depending on a lagged state variable. Each regime may be simple even when the combined process is nonlinear and asymmetric. Regime-specific coefficients can exceed one in magnitude without automatically making the complete process explosive, because switching rules constrain the path. Simulation and stability analysis are essential.

42.3 Markov switching treats the regime as latent

A Markov-switching model assumes an unobserved state that follows transition probabilities. The expected duration of a state is related to the probability of remaining in it. Regime labels are arbitrary and should be interpreted through estimated means, variances, and posterior probabilities. Apparent regimes can also reflect structural breaks or omitted variables rather than a persistent hidden process.

42.4 Market data are generated by trading mechanisms

Transaction prices are discrete, trades arrive irregularly, and bid-ask bounce can create negative short-lag dependence. Ordered probit models represent observed categories as intervals of a latent continuous variable. Thresholds and covariates determine category probabilities. The model describes ordering, not equal distance between categories, and a standard implementation assumes a common latent error scale unless extended.

42.5 Core equations

Ordered response

$$y_i = j \text{ if } \alpha_{j-1} < x_i \beta + \varepsilon_i \leq \alpha_j$$

Observed categories arise when a latent index crosses ordered thresholds.

42.6 Python demonstrations

42.6.1 Demonstration 23.1: Simulate a two-regime TAR process

```
import numpy as np

rng = np.random.default_rng(23)
x = np.zeros(300)
for t in range(1, len(x)):
    phi = -1.2 if x[t-1] < 0 else 0.55
    x[t] = phi*x[t-1] + rng.normal()
print(round(x.mean(), 3), round((x < 0).mean(), 3))
```

Verified output

```
0.726 0.263
```

Interpretation. The unconditional mean is positive even though neither regime includes an intercept, illustrating nonlinear asymmetry.

42.6.2 Demonstration 23.2: Fit an ordered probit

```
import numpy as np
import pandas as pd
from statsmodels.miscmodels.ordinal_model import OrderedModel

rng = np.random.default_rng(2301)
x = rng.normal(size=700)
latent = 0.8*x + rng.normal(size=700)
y = pd.cut(latent, [-np.inf, -0.7, 0.7, np.inf], labels=[0,1,2], ordered=True)
model = OrderedModel(y, pd.DataFrame({"x": x}), distr="probit").fit(method="bfgs",
disp=False)
print(round(model.params["x"], 3))
print(np.round(model.model.predict(model.params, exog=pd.DataFrame({"x": [0.0]}))[0], 3))
```

Verified output

```
0.805
[0.282 0.518 0.2 ]
```

Interpretation. At $x=0$, the central category is most probable. The probabilities sum to one and depend on both slope and thresholds.

42.7 Visual evidence

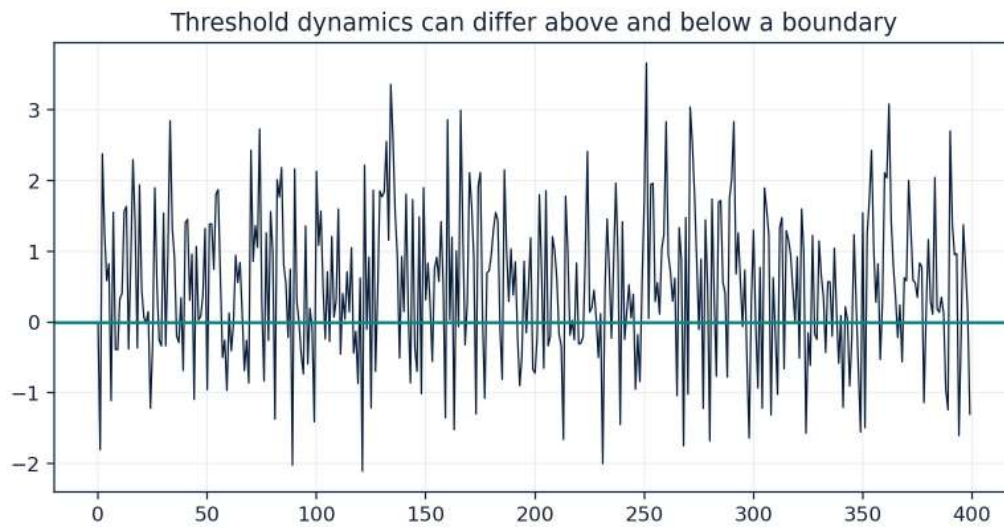


Figure 31. A simulated two-regime threshold autoregressive process.

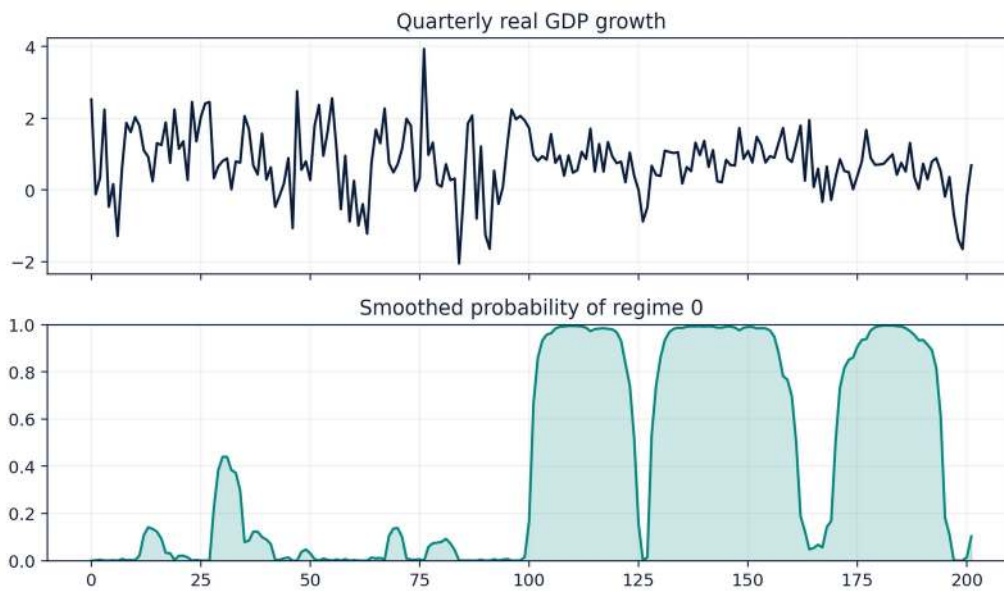


Figure 32. A two-regime Markov model separates periods with different mean and variance patterns.

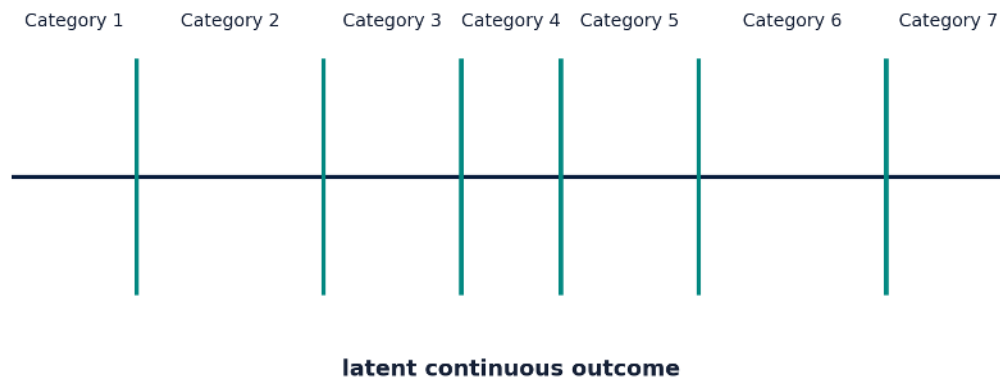


Figure 33. Ordered models map a latent continuous score into ranked observed categories.

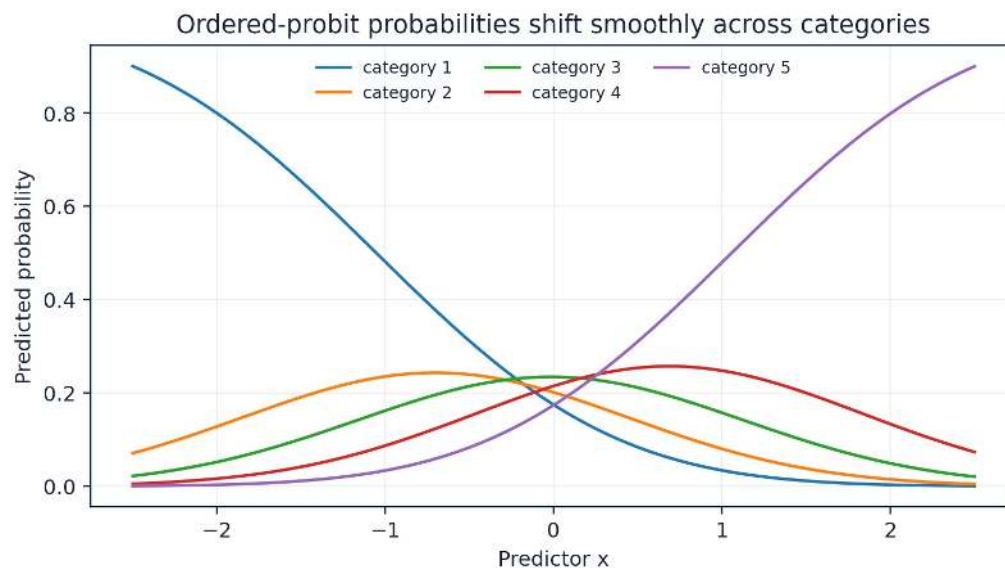


Figure 34. Predicted category probabilities from a fitted ordered-probit model.

42.8 Reference table

Table 23. Chapter reference.

Model	State observed?	Key interpretation
TAR	Yes, threshold variable	piecewise coefficients
Markov switching	No	transition probabilities and filtered states
ordered probit	Observed category only	latent index and thresholds
microstructure model	Trades/quotes observed	institutional price formation

Why This Matters

Nonlinear and microstructure models prevent a single linear equation from flattening state-dependent behaviour.

Common Mistake

Interpreting estimated regimes as known historical labels without examining posterior probabilities and uncertainty.

Ceteris LAB Tip

For ordered models, graph predicted category probabilities over a meaningful covariate range instead of reading thresholds in isolation.

R-to-Python / Source Bridge

The source nonlinear lecture and separate ordered-probit handout are integrated here. Python replaces custom R indicators and `polr()` with transparent feature construction and `statsmodels OrderedModel` (Tsay 2013).

42.9 Guided practice

1. Re-run Demonstration 23.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

42.10 Exercises

1. Simulate a TAR process under two threshold rules.
2. Fit a Markov-switching mean model.
3. Explain how bid-ask bounce affects short-lag returns.
4. Plot ordered-probit category probabilities.

42.11 Chapter summary

- Threshold models use observed regime rules; Markov models use latent states.
- Trading mechanisms can create dependence in transaction data.
- Ordered probit maps a latent index into ordered categories.

42.12 Key Python commands

```
import numpy as np
rng = np.random.default_rng(23)
x = np.zeros(300)
x[t] = phi*x[t-1] + rng.normal()
print(round(x.mean(), 3), round((x < 0).mean(), 3))
import numpy as np
```

43 Stochastic Processes and Option Pricing

Opening question: **How can continuous-time uncertainty be simulated and translated into a no-arbitrage option value?**

43.1 Learning objectives

- Simulate Brownian motion and geometric Brownian motion.
- Explain Ito's lemma intuitively.
- Calculate European option payoffs and Black-Scholes prices.
- Use Monte Carlo and sensitivity analysis.

Prerequisites: Probability, logarithms, derivatives, and Chapter 22.

Key terms: Brownian motion, diffusion, Ito's lemma, geometric Brownian motion, Black-Scholes, Greek.

43.2 Brownian motion accumulates random increments

A standard Brownian motion begins at zero, has independent normally distributed increments, and variance that grows with elapsed time. Its paths are continuous but nowhere classically differentiable. A generalized diffusion adds drift and a volatility scale. Geometric Brownian motion multiplies both by the current price, keeping simulated prices positive under the exact lognormal solution.

43.3 Ito's lemma adds a variance correction

Ordinary calculus ignores squared infinitesimal changes, but a Brownian increment has magnitude proportional to the square root of time, so its square contributes at order dt . Ito's lemma therefore adds a second-derivative term. Applied to log price under geometric Brownian motion, it produces drift μ minus one-half σ^2 . The result explains why expected arithmetic return and expected log growth differ.

43.4 Black-Scholes is a model, not a market law

The Black-Scholes-Merton framework derives a European option price by constructing a locally hedged portfolio under assumptions including continuous trading, frictionless markets, constant volatility and rate, and a diffusion price process (Black and Scholes 1973; Merton 1973). Real markets violate these assumptions. The formula remains a useful benchmark, and sensitivity measures such as delta and vega reveal how the benchmark changes when inputs move.

43.5 Core equations

Geometric Brownian motion

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

The exact log-price increment is normal with drift $(\mu - \sigma^2/2)dt$ and variance $\sigma^2 dt$.

European call

$$C = S_0 \Phi(d_1) - K e^{-rT} \Phi(d_2)$$

The Black-Scholes call price under the standard no-dividend assumptions.

43.6 Python demonstrations

43.6.1 Demonstration 24.1: Simulate geometric Brownian motion

```
import numpy as np

rng = np.random.default_rng(24)
S0, mu, sigma, T, steps = 100, 0.06, 0.20, 1, 252
dt = T/steps
z = rng.normal(size=steps)
log_path = np.log(S0) + np.cumsum((mu-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*z)
print(round(np.exp(log_path[-1]), 2))
```

Verified output

59.88

Interpretation. One path is a possible model realization, not a forecast. Many paths are needed for a distribution or Monte Carlo estimate.

43.6.2 Demonstration 24.2: Black-Scholes call price

```
import sys
from pathlib import Path
sys.path.insert(0, str(Path("scripts").resolve()))
from ceteris_examples import black_scholes_call

price = black_scholes_call(S=100, K=100, T=1, r=0.03, sigma=0.20)
print(round(float(price), 4))
```

Verified output

9.4134

Interpretation. The value is conditional on the model inputs and assumptions. It is not a trading recommendation or a guarantee of market price.

43.7 Visual evidence

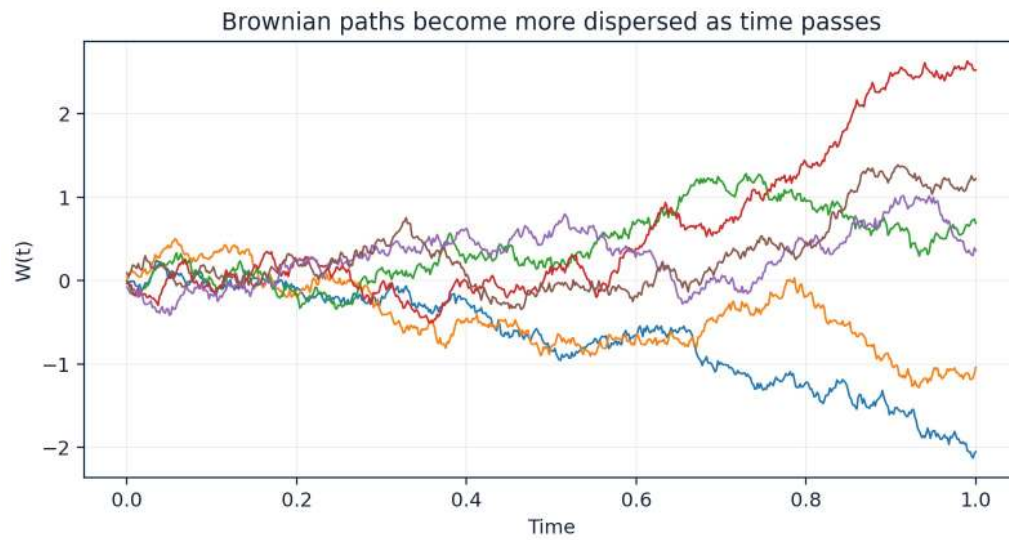


Figure 35. Six simulated standard Brownian-motion paths.

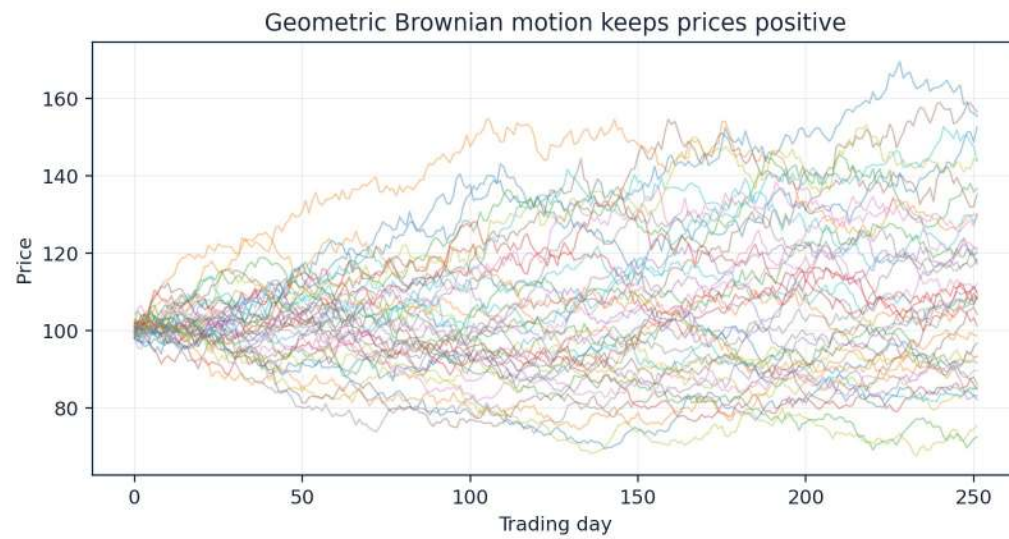


Figure 36. Forty one-year geometric Brownian-motion price paths.

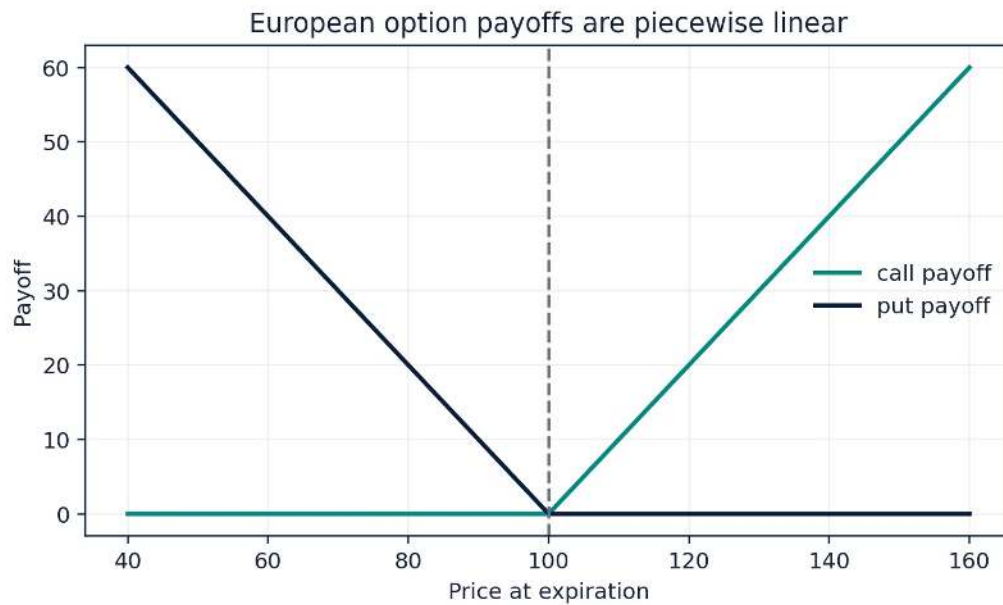


Figure 37. European call and put payoffs at a strike of 100.

Black-Scholes call value rises with price and volatility

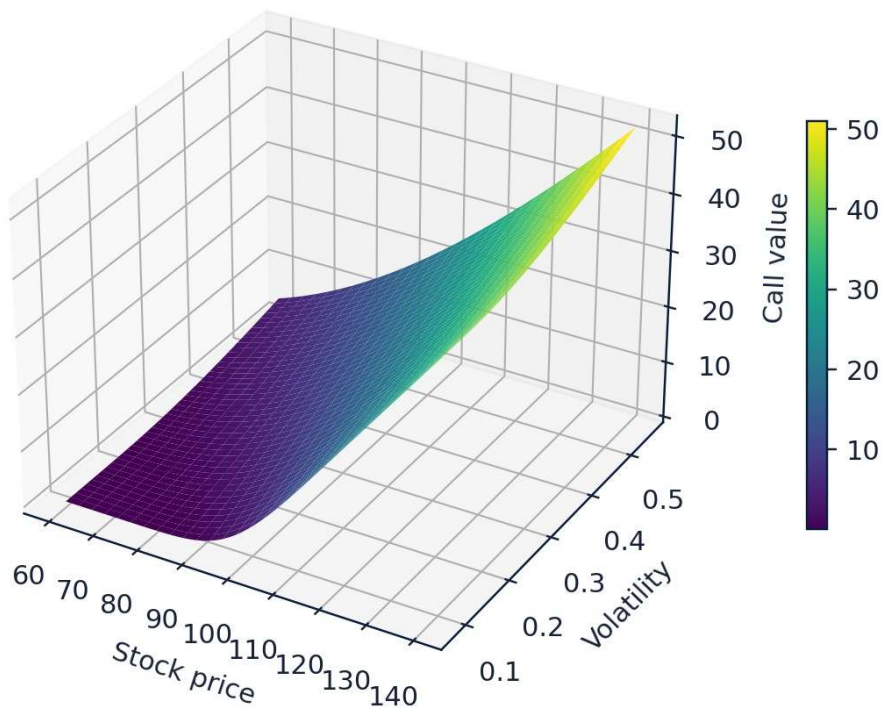


Figure 38. Black-Scholes European call prices across stock prices and volatilities.

43.8 Reference table

Table 24. Chapter reference.

Input	Effect on call price	Caution
spot S	usually positive	moneyness matters
strike K	usually negative	contract definition
time T	often positive	theta can vary
rate r	usually positive for calls	yield conventions
volatility σ	positive vega	implied volatility varies by strike and maturity

Why This Matters

Stochastic-process models connect return dynamics, derivative valuation, simulation, and risk management.

Common Mistake

Using an annual volatility with time measured in days, or a daily volatility with time measured in years.

Ceteris LAB Tip

Write all time and rate units beside the formula before calculating an option value.

R-to-Python / Source Bridge

The source option-pricing lecture develops Brownian motion, Ito's lemma, geometric Brownian motion, hedging, and Black-Scholes. Every R simulation is replaced by reproducible NumPy code and original sensitivity graphics (Tsay 2013).

43.9 Guided practice

1. Re-run Demonstration 24.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

43.10 Exercises

1. Simulate four Brownian paths.
2. Verify that Brownian variance increases with time.
3. Compare Monte Carlo and Black-Scholes call values.
4. Plot call value against volatility and time.

43.11 Chapter summary

- Brownian motion is the foundation of diffusion models.
- Ito's lemma introduces a second-derivative variance term.
- Black-Scholes is a benchmark conditional on restrictive assumptions.

43.12 Key Python commands

```
import numpy as np
rng = np.random.default_rng(24)
z = rng.normal(size=steps)
log_path = np.log(S0) + np.cumsum((mu-0.5*sigma**2)*dt + sigma*np.sqrt(dt)*z)
print(round(np.exp(log_path[-1]), 2))
import sys
```

44 Financial Risk Management

Opening question: **How much could be lost, how often should that threshold be exceeded, and what happens beyond it?**

44.1 Learning objectives

- Define a loss variable and coherent risk principles.
- Calculate VaR and Expected Shortfall.
- Compare historical, parametric, GARCH, and EVT methods.
- Backtest exceedances and discuss stress testing.

Prerequisites: Chapters 13, 14, 21, and 22.

Key terms: loss, Value at Risk, Expected Shortfall, backtesting, stress test, extreme-value theory.

44.2 The sign convention comes first

Risk calculations begin by defining a loss variable. For a long position, loss is often negative return times position value; for a short position the sign reverses. VaR is a high quantile of loss over a stated horizon and confidence level. Without the position, horizon, probability, currency, and sign convention, a VaR number is incomplete.

44.3 Expected Shortfall looks beyond the threshold

VaR identifies a quantile but does not describe the average severity of worse outcomes. Expected Shortfall averages losses in the tail beyond the VaR threshold under a continuous idealization. ES satisfies coherence properties under standard definitions, while VaR can fail subadditivity for some distributions (Artzner et al. 1999). Both remain model-dependent and can be unstable with limited tail data.

44.4 Backtesting checks frequency, not every dimension of risk

A 99 percent one-day VaR should be exceeded about one percent of the time under a correct conditional model. Too many or too few exceedances indicate calibration problems, while clustered exceedances reveal dependence. Backtests have limited power in short samples and do not validate scenario coverage. Stress testing asks what happens under severe specified conditions, including events absent from the estimation sample. Extreme-value methods focus directly on tail behaviour but introduce threshold and dependence choices.

44.5 Core equations

Value at Risk

$$VaR_q = F_L^{-1}(q)$$

The q quantile of the loss distribution.

Expected Shortfall

$$ES_q = E[L \mid L > VaR_q]$$

For a continuous loss distribution, the expected loss beyond VaR.

44.6 Python demonstrations

44.6.1 Demonstration 25.1: Historical VaR and ES

```
import numpy as np

rng = np.random.default_rng(25)
returns = 0.01 * rng.standard_t(df=5, size=2_000)
losses = -returns
q = 0.99
var = np.quantile(losses, q)
es = losses[losses >= var].mean()
print(round(var, 4), round(es, 4), int((losses >= var).sum()))
```

Verified output

```
0.0383 0.0518 20
```

Interpretation. At the 99th percentile, roughly 20 of 2,000 simulated observations lie in the tail used for ES, illustrating tail-sample scarcity.

44.6.2 Demonstration 25.2: Count VaR exceedances

```
threshold = np.full_like(losses, var)
exceed = losses > threshold
expected = len(losses) * (1-q)
print(exceed.sum(), round(expected, 1), round(exceed.mean(), 4))
```

Verified output

```
20 20.0 0.01
```

Interpretation. The unconditional exceedance rate matches by construction for an in-sample historical quantile. A real backtest must forecast each threshold using only prior information.

44.7 Visual evidence

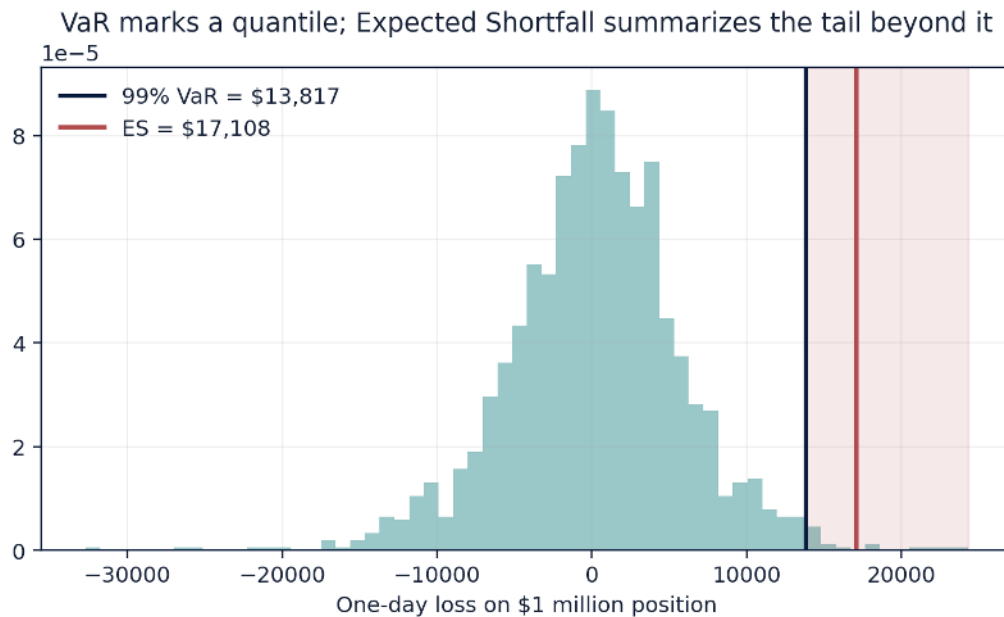


Figure 39. Historical 99% Value at Risk and Expected Shortfall for simulated portfolio losses.

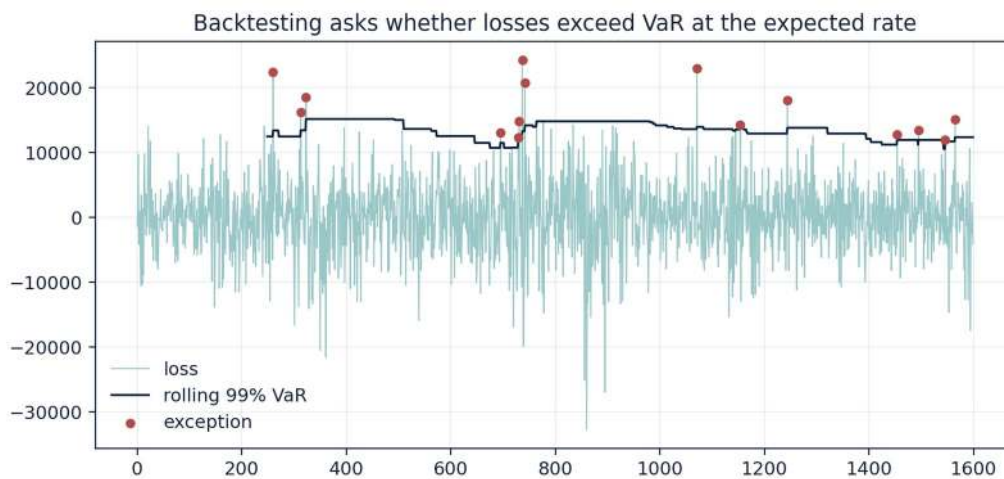


Figure 40. A rolling historical VaR backtest with exceedances marked as exceptions.

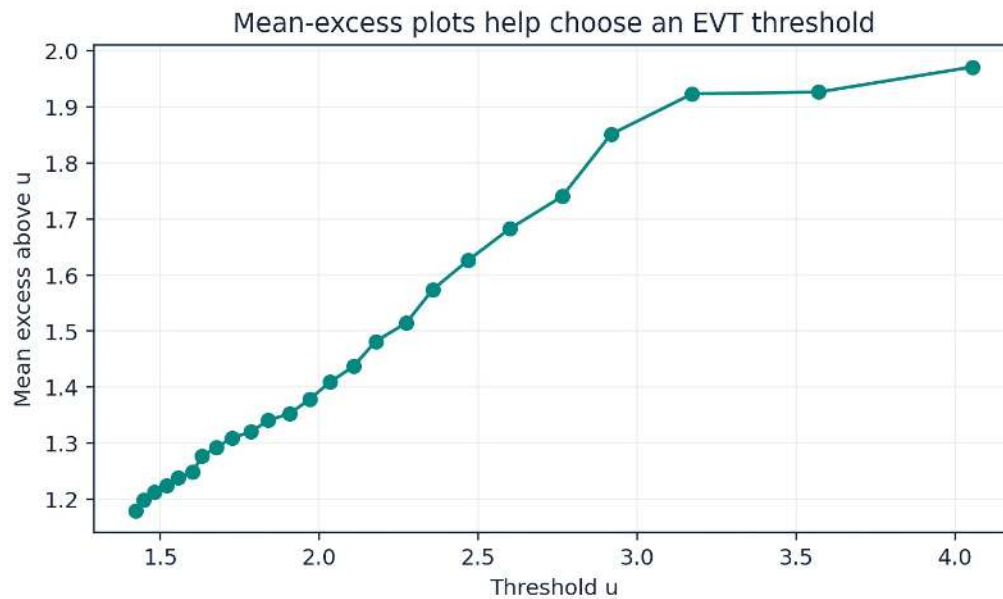


Figure 41. Mean excess above candidate thresholds for a heavy-tailed sample.

44.8 Reference table

Table 25. Chapter reference.

Method	Strength	Weakness
historical simulation	few distribution assumptions	limited to observed history
normal parametric	simple closed form	thin tails and symmetry
Student-t parametric	heavier tails	degrees-of-freedom sensitivity
GARCH VaR	time-varying volatility	model and innovation risk
POT EVT	tail-focused	threshold and dependence sensitivity

Why This Matters

Risk measures organize a conversation about tail loss, capital, limits, and model uncertainty, but they do not eliminate uncertainty.

Common Mistake

Backtesting a VaR model on the same observations used to estimate each day's threshold.

Ceteris LAB Tip

Keep losses positive and returns signed in separate variables. This prevents silent sign reversals.

R-to-Python / Source Bridge

The source risk lecture covers coherence, VaR, ES, RiskMetrics, GARCH risk, backtesting, stress testing, and EVT. The Python chapter preserves the mathematical core while strengthening sign, horizon, and out-of-sample controls (Tsay 2013).

44.9 Guided practice

1. Re-run Demonstration 25.1 and change one input while keeping the analytical question fixed.

2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

44.10 Exercises

1. Compute 95 and 99 percent historical VaR and ES.
2. Compare normal and Student-t parametric risk.
3. Design an expanding-window VaR backtest.
4. Create a stress scenario not present in the historical sample.

44.11 Chapter summary

- VaR is a loss quantile; ES summarizes severity beyond it.
- Backtesting must use ex ante thresholds.
- Tail estimation and stress design remain uncertain and model-dependent.

44.12 Key Python commands

```
import numpy as np
rng = np.random.default_rng(25)
returns = 0.01 * rng.standard_t(df=5, size=2_000)
var = np.quantile(losses, q)
es = losses[losses >= var].mean()
print(round(var, 4), round(es, 4), int((losses >= var).sum()))
```

45 Multiple Time Series

Opening question: **When several series move together, which relationships are short-run predictive and which are long-run equilibrating?**

45.1 Learning objectives

- Compute cross-correlation structures.
- Estimate and diagnose VAR models.
- Interpret Granger predictability and impulse responses cautiously.
- Test cointegration and describe VECM adjustment.

Prerequisites: Chapters 18 to 20.

Key terms: cross-correlation, VAR, Granger predictability, impulse response, cointegration, VECM.

45.2 A vector model uses shared information

A vector autoregression treats each variable as a function of lags of all variables in the system. This can improve forecasts when series contain complementary predictive information. The number of parameters grows quickly with variables and lags, making lag selection, sample size, and regularization important. Residual cross-correlation and serial diagnostics determine whether the system has captured its own dynamics.

45.3 Predictive ordering is not structural causation

A Granger test asks whether lags of one variable improve prediction of another conditional on the model. Rejection can reflect omitted common causes, measurement timing, policy anticipation, or data aggregation. Impulse responses trace a model's response to shocks, but contemporaneous identification requires assumptions such as variable ordering or structural restrictions. Those assumptions must be stated beside the graph.

45.4 Cointegration separates short and long horizons

Two or more unit-root series can share a stationary linear combination. Cointegration implies a long-run relation, while a vector error-correction model describes both short-run changes and adjustment toward that relation. Johansen methods estimate cointegrating rank and vectors under a specified deterministic and lag structure (Johansen 1988). Cointegration is not guaranteed arbitrage; trading costs, breaks, and data snooping can erase apparent opportunity.

45.5 Core equations

VAR(1)

$$y_t = c + A_1 y_{t-1} + \varepsilon_t$$

Each variable can depend on all variables at the previous lag.

VECM

$$\Delta y_t = \Pi y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \varepsilon_t$$

The rank of Π determines the number of cointegrating relations.

45.6 Python demonstrations

45.6.1 Demonstration 26.1: Estimate a small VAR

```
import numpy as np
import pandas as pd
from statsmodels.tsa.api import VAR

rng = np.random.default_rng(26)
y = np.zeros((400, 2))
for t in range(1, 400):
    y[t] = [0.55*y[t-1,0] + 0.20*y[t-1,1], -0.10*y[t-1,0] + 0.45*y[t-1,1]] + rng.normal(scale=0.6,
size=2)
df = pd.DataFrame(y, columns=["x", "z"])
fit = VAR(df).fit(1)
print(fit.coefs[0].round(2))
```

Verified output

```
[[ 0.55  0.15]
 [-0.13  0.47]]
```

Interpretation. The estimated lag matrix is close to the simulated system, with sampling variation.

45.6.2 Demonstration 26.2: Test a cointegrating spread

```
import numpy as np
from statsmodels.tsa.stattools import adfuller

rng = np.random.default_rng(2601)
common = np.cumsum(rng.normal(size=600))
x = common + rng.normal(scale=0.5, size=600)
y = 1.2*common + rng.normal(scale=0.5, size=600)
```

```
spread = y - 1.2*x
print(f"{adfuller(spread)[1]:.2e}")
```

Verified output

```
0.00e+00
```

Interpretation. The constructed spread is stationary even though the two levels inherit a shared stochastic trend.

45.7 Visual evidence

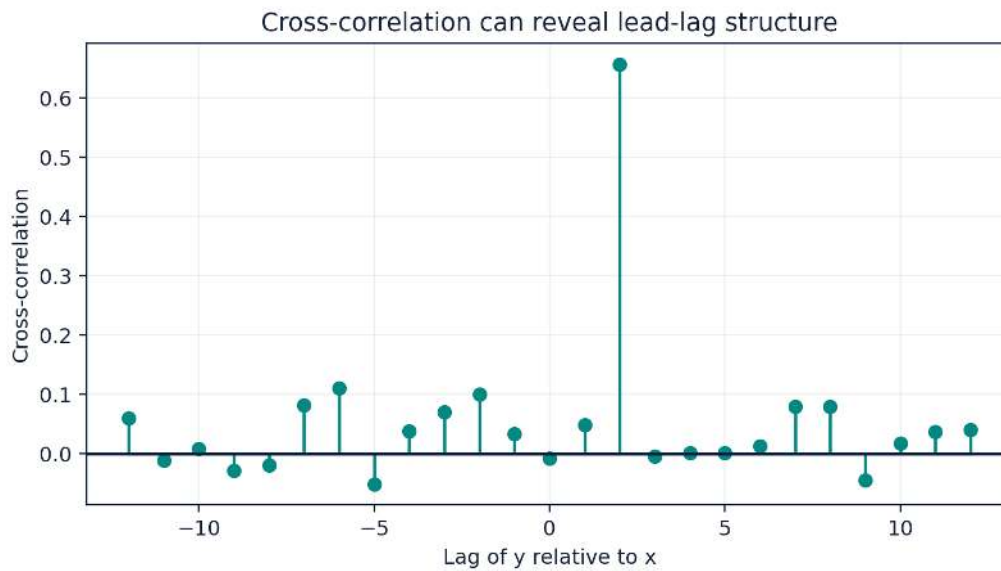


Figure 42. A synthetic lead-lag relationship peaks near a lag of two periods.

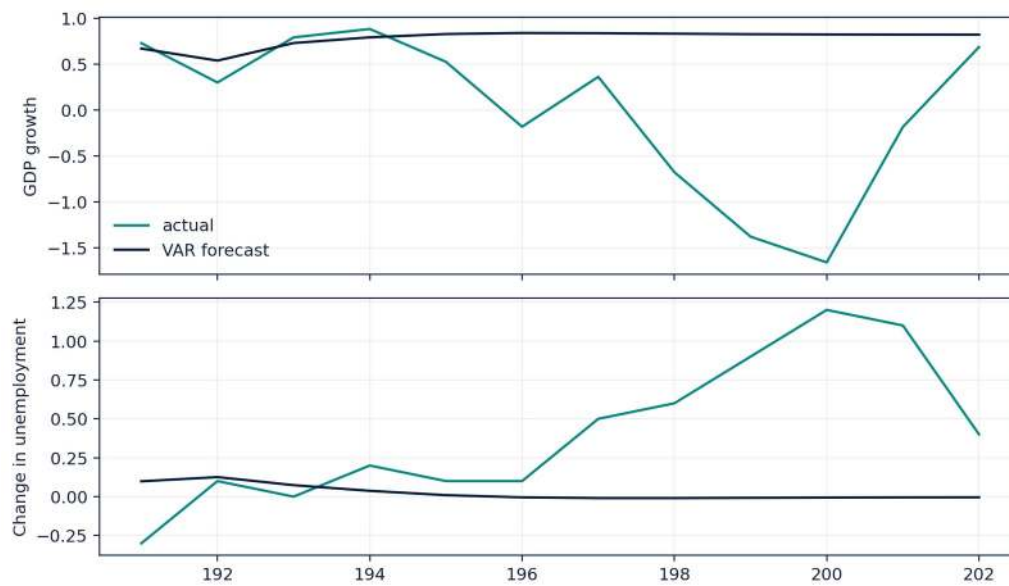


Figure 43. A bivariate VAR forecast for U.S. GDP growth and changes in unemployment.

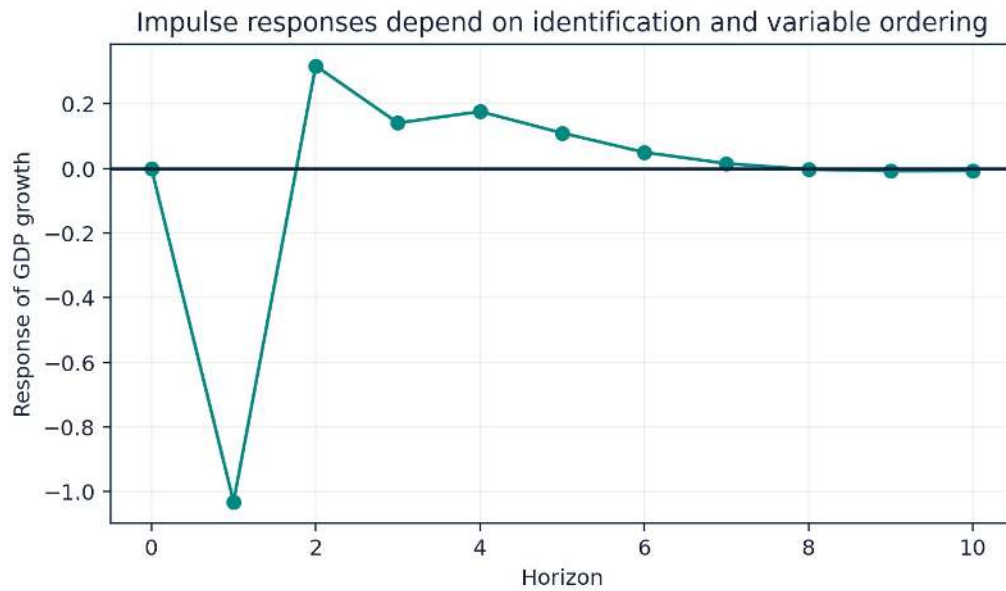


Figure 44. Illustrative response of GDP growth to the second VAR innovation.

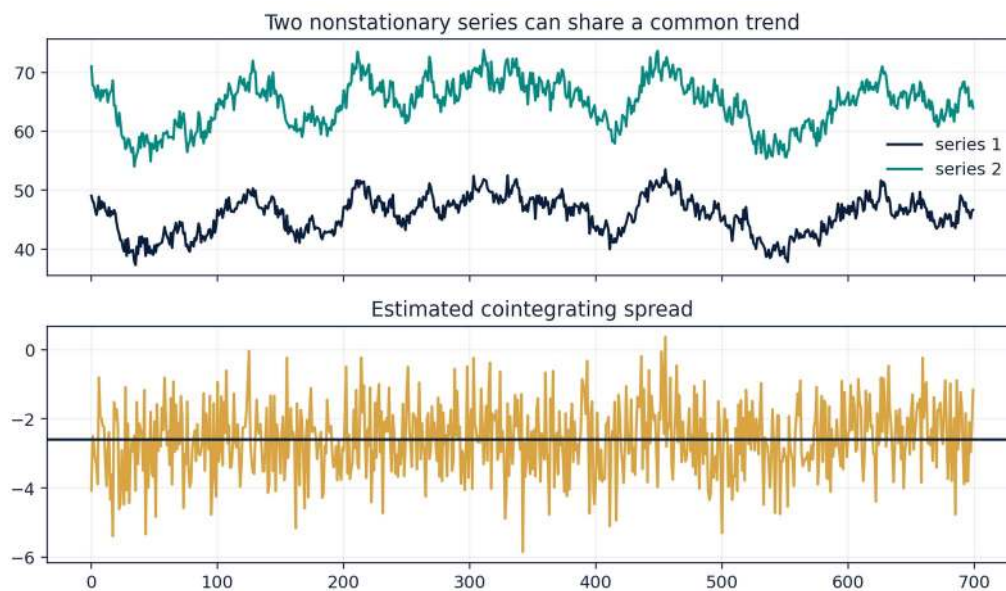


Figure 45. Simulated cointegrated prices and their mean-reverting linear combination.



Figure 46. A cointegration-spread z-score with illustrative entry thresholds.

45.8 Reference table

Table 26. Chapter reference.

Output	What it answers	Required caution
VAR coefficient	conditional lag relation	system scale and many tests
Granger test	incremental predictive content	not structural causality
impulse response	dynamic response to identified shock	ordering/restriction dependence
cointegrating vector	stationary long-run combination	normalization and breaks
error-correction loading	speed/direction of adjustment	model specification

Why This Matters

Multivariate models separate joint prediction, shock identification, and long-run adjustment into distinct analytical questions.

Common Mistake

Calling a Granger test evidence that one policy variable causes another outcome.

Ceteris LAB Tip

Write the contemporaneous identification assumption directly in the impulse-response caption.

R-to-Python / Source Bridge

The final source lecture develops cross-covariance, VAR, Granger relations, cointegration, Johansen testing, and pairs trading. Python replaces custom R scripts with statsmodels VAR/VECM tools and stronger identification cautions (Tsay 2013).

45.9 Guided practice

1. Re-run Demonstration 26.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.

3. Add one validation check that would prevent a plausible error.

45.10 Exercises

1. Fit a bivariate VAR and select lags by AIC and BIC.
2. Test each predictive direction.
3. Compare impulse responses under two variable orderings.
4. Simulate cointegrated series and estimate a VECM.

45.11 Chapter summary

- VAR models exploit cross-series lag information.
- Granger tests are predictive, not structural.
- Cointegration and VECM connect nonstationary levels to stationary long-run combinations.

45.12 Key Python commands

```
import numpy as np
import pandas as pd
from statsmodels.tsa.api import VAR
rng = np.random.default_rng(26)
y = np.zeros((400, 2))
df = pd.DataFrame(y, columns=["x", "z"])
```



PART IV

Machine Learning and Artificial Intelligence

LEARN • ANALYZE • UNDERSTAND

Ceteris LAB | econometrics.safavim.com

46 Econometrics and Machine Learning: Different Questions, Shared Tools

Opening question: Which tasks deserve the label AI, and which claims collapse when we ask what data, objective, and evaluation produced the result?

46.1 Learning objectives

- Distinguish AI, machine learning, deep learning, and generative AI.
- Separate rule-based systems from learned models.
- Explain regression, classification, generation, and retrieval.
- Identify hallucination, bias, privacy, and deepfake risks.

Prerequisites: Parts I and II.

Key terms: artificial intelligence, machine learning, deep learning, generative AI, hallucination, bias.

46.2 AI is an umbrella, not a single mechanism

Artificial intelligence is a broad label for systems designed to perform tasks associated with perception, language, prediction, planning, or decision support. Machine learning is a subset in which patterns are estimated from data rather than fully specified as hand-written rules. Deep learning uses layered neural networks. Generative models produce new text, images, audio, or code. A system can be useful and still be narrow: performance on one task does not imply general understanding.

46.3 Learning means optimizing an objective

A supervised model receives examples with features and targets and chooses parameters that reduce a loss function. A classifier does not “know” a class in the human sense; it maps inputs to scores or probabilities according to its training objective. Unsupervised methods search for structure without labelled outcomes. Reinforcement learning connects actions to rewards. Every approach inherits the information and omissions in its data and objective.

46.4 Fluency is not verification

Generative systems can produce coherent but unsupported statements, fabricated references, or subtly incorrect code. Hallucination is not cured by confident prose. Bias can enter through historical data, labels, sampling, model choices, or deployment context. Deepfakes and synthetic media make provenance checks increasingly important. Responsible use requires

disclosure, privacy protection, human review, and task-specific evaluation rather than vague trust in “AI” (Sharifi-Zarchi and contributors 2026).

46.5 Python demonstrations

46.5.1 Demonstration 27.1: Rules versus learned boundaries

```
def rule_based_income(income):
    return "high" if income >= 70_000 else "not high"

for value in [55_000, 72_000]:
    print(value, rule_based_income(value))
```

Verified output

```
55000 not high
72000 high
```

Interpretation. The threshold is fully specified by a person. A learned classifier would estimate a boundary from labelled examples.

46.5.2 Demonstration 27.2: A tiny learned classifier

```
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[20], [30], [45], [60], [75], [90]])
y = np.array([0, 0, 0, 1, 1, 1])
model = LogisticRegression().fit(X, y)
print(np.round(model.predict_proba([[50], [80]]), 3))
```

Verified output

```
[[0.76 0.24]
 [0.  1.  ]]
```

Interpretation. The probabilities arise from the fitted data and model, not from a hand-coded threshold, although the training labels still reflect human choices.

46.6 Visual evidence

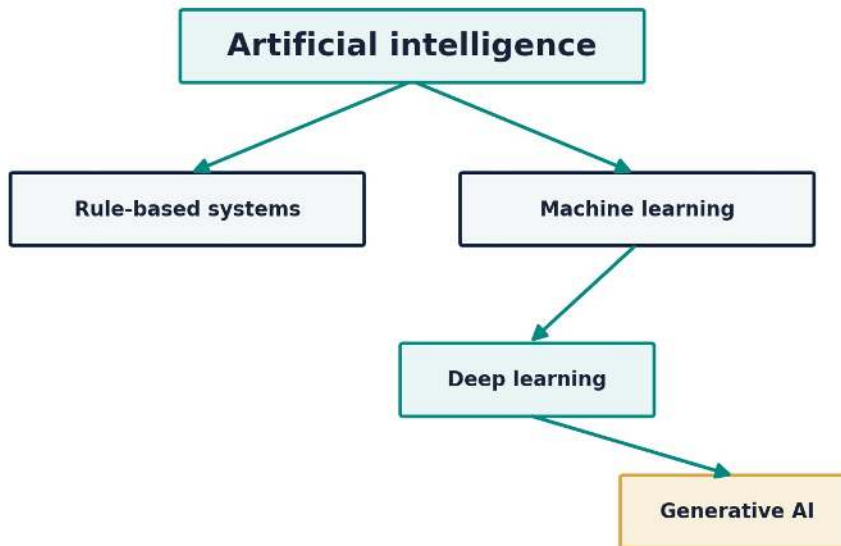


Figure 47. A simplified relationship among AI, machine learning, deep learning, and generative AI.

46.7 Reference table

Table 27. Chapter reference.

System type	Learns from examples?	Typical output
rule-based program	No, rules are coded	deterministic decision
regression model	Yes	continuous prediction
classifier	Yes	class score or probability
generative model	Yes	new content
retrieval system	Indexes examples/documents	ranked evidence
agent	May combine models and tools	multi-step action

Why This Matters

Clear definitions prevent AI enthusiasm from outrunning evidence, governance, and task-specific evaluation.

Common Mistake

Treating a fluent language-model answer as a cited fact or a successful code run as proof of conceptual correctness.

Ceteris LAB Tip

Ask four questions: What is the task? What data shaped the system? What objective was optimized? How was performance evaluated?

R-to-Python / Source Bridge

This chapter adapts the first three IntroAI sessions, which emphasize practical intuition, inflated claims, hallucination, bias, and the difference between programming rules and learning patterns (Sharifi-Zarchi and contributors 2026).

46.8 Guided practice

1. Re-run Demonstration 27.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

46.9 Exercises

1. Classify five everyday systems as rule-based, predictive, generative, retrieval, or agentic.
2. Give one example of label bias.
3. Design a verification checklist for AI-generated code.
4. Explain why narrow task performance is not evidence of general intelligence.

46.10 Chapter summary

- AI is a broad family of systems, not one method.
- Machine learning estimates patterns by optimizing objectives on data.
- Fluency, accuracy, fairness, and safety require separate evaluation.

46.11 Key Python commands

```
def rule_based_income(income):
    print(value, rule_based_income(value))
import numpy as np
from sklearn.linear_model import LogisticRegression
X = np.array([[20], [30], [45], [60], [75], [90]])
y = np.array([0, 0, 0, 1, 1, 1])
```

47 Machine-Learning Workflow and Gradient Descent for Economic Data

Opening question: **How can a model be trained without letting information from the future or test set leak into the learning process?**

47.1 Learning objectives

- Define features, labels, train, validation, and test sets.
- Recognize overfitting, underfitting, and leakage.
- Explain loss functions and gradient descent.
- Build reproducible preprocessing pipelines.

Prerequisites: Chapter 27 and basic calculus intuition.

Key terms: feature, label, generalization, data leakage, loss function, gradient descent.

47.2 The split defines the claim

A training set estimates model parameters. A validation set supports choices such as complexity and tuning. A test set is held back for the final unbiased evaluation of the chosen workflow. In time series, random splitting can let future observations train a model used to “predict” the past. The split must reflect deployment: future-from-past forecasting, new households, new countries, or new images.

47.3 Overfitting is a gap between memory and generalization

A highly flexible model can fit training noise and fail on new observations. Underfitting occurs when a model is too constrained to capture relevant structure. Learning curves compare training and validation performance across sample sizes or complexity. Regularization, simpler features, more representative data, and better validation can reduce overfitting, but no method rescues a mismatch between training data and deployment conditions.

47.4 Gradient descent follows local slope information

Gradient descent updates parameters in the direction that reduces a differentiable loss. The learning rate controls step size: too small can be slow, too large can oscillate or diverge. Feature scaling often improves numerical behaviour. Modern optimizers add momentum or adaptive scaling, yet the underlying idea remains iterative improvement of an objective. Optimization success does not guarantee a meaningful target or fair data.

47.5 Core equations

Gradient update

$$\theta_{k+1} = \theta_k - \eta \nabla_{\theta} L(\theta_k)$$

The learning rate eta scales a step opposite the loss gradient.

47.6 Python demonstrations

47.6.1 Demonstration 28.1: Gradient descent for one slope

```
import numpy as np
x = np.array([1., 2., 3., 4.])
y = 2.5 * x
slope = 0.0
learning_rate = 0.02
for _ in range(500):
    gradient = -2 * np.mean(x * (y - slope*x))
    slope -= learning_rate * gradient
print(round(slope, 4))
```

Verified output

```
2.5
```

Interpretation. The iterative slope converges to the least-squares solution for this noiseless one-parameter problem.

47.6.2 Demonstration 28.2: A leakage-safe pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge

pipeline = Pipeline([
    ("scale", StandardScaler()),
    ("model", Ridge(alpha=1.0)),
])
print([name for name, _ in pipeline.steps])
```

Verified output

```
['scale', 'model']
```

Interpretation. When the pipeline is fitted inside cross-validation, scaling parameters are learned only from each training fold.

47.7 Visual evidence



Figure 48. An honest predictive workflow isolates the final test set until model choices are complete.

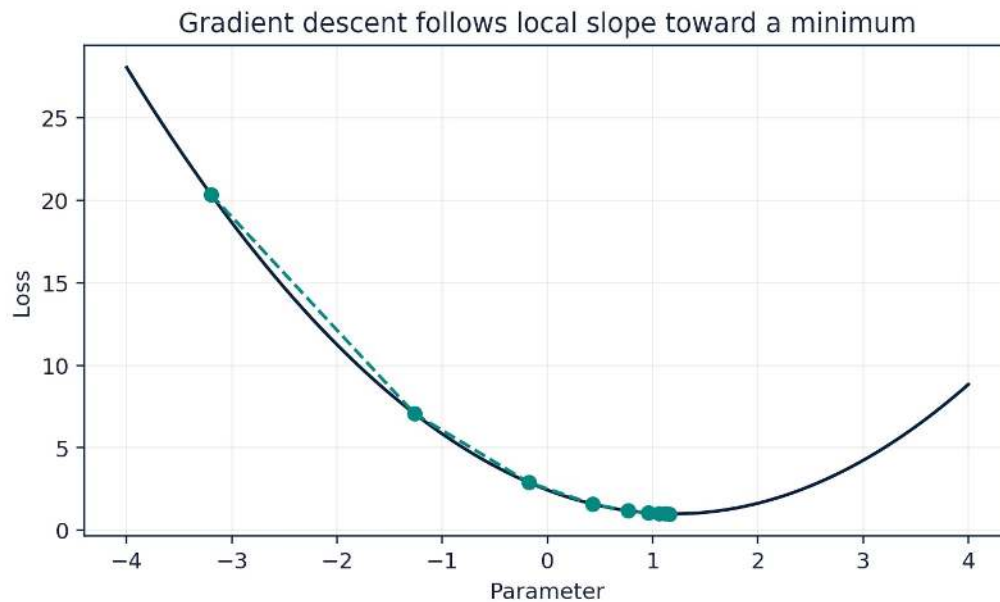


Figure 49. Iterative parameter updates on a simple quadratic loss surface.

47.8 Reference table

Table 28. Chapter reference.

Failure	How it appears	Prevention
overfitting	low train error, high validation error	simplify, regularize, improve validation
underfitting	high error on both	richer model or features
target leakage	implausibly strong validation	construct features using only available information
distribution shift	performance decays in deployment	representative data and monitoring
test-set tuning	test score improves after repeated choices	lock test set until final workflow

Why This Matters

The evaluation design is part of the model. A score without a credible split says little about real-world generalization.

Common Mistake

Scaling the full dataset before cross-validation. Information from validation folds then influences training transformations.

Ceteris LAB Tip

Place every learned preprocessing step inside a pipeline.

R-to-Python / Source Bridge

The chapter adapts IntroAI sessions on features, labels, memorization, data leakage, MAE/MSE, gradient descent, and learning rates while adding econometric time-order controls (Sharifi-Zarchi and contributors 2026).

47.9 Guided practice

1. Re-run Demonstration 28.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

47.10 Exercises

1. Design train, validation, and test splits for a household dataset.
2. Explain why random splitting is inappropriate for one-step-ahead forecasting.
3. Experiment with three learning rates.
4. Build a pipeline with imputation, scaling, and regression.

47.11 Chapter summary

- Generalization is measured on data not used for fitting or tuning.
- Leakage can make a weak model appear excellent.
- Gradient descent iteratively reduces a chosen loss.

47.12 Key Python commands

```
import numpy as np
x = np.array([1., 2., 3., 4.])
gradient = -2 * np.mean(x * (y - slope*x))
print(round(slope, 4))
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

48 Applied Economic Prediction and Classification Projects

Opening question: **How can an end-to-end workflow turn housing and passenger data into models that are evaluated honestly and interpreted carefully?**

48.1 Learning objectives

- Build a regression workflow with the Tehran housing data.
- Build a classification workflow with Titanic data.
- Use pipelines, cross-validation, and multiple metrics.
- Analyze residuals, confusion matrices, and subgroup performance.

Prerequisites: Chapters 10, 15, and 28.

Key terms: feature engineering, baseline, decision tree, confusion matrix, precision, recall.

48.2 Project A: Tehran house prices

The housing dataset from the IntroAI archive supports a beginner regression project. The target is price, while area, rooms, parking, warehouse, elevator, and location-related variables serve as features. The workflow begins with a data audit and a simple baseline, then compares linear and tree-based models. Price distributions and neighbourhood effects can be highly skewed, so residual plots and transformed targets deserve attention. The exercise is predictive and does not estimate a causal effect of area or amenities.

48.3 Project B: Titanic classification

The Titanic dataset illustrates categorical and numeric preprocessing, missing values, class probabilities, and threshold-dependent decisions. Accuracy can conceal failure on a minority class. Precision answers how often predicted positives are correct; recall answers how many actual positives were found; F1 balances the two harmonically. A confusion matrix grounds these metrics in counts. Historical social structures also mean that model patterns should not be treated as neutral prescriptions.

48.4 Baselines make improvement meaningful

A regression baseline may predict the training mean; a classification baseline may predict the majority class or observed prevalence. Complex models should outperform these baselines on held-out data and justify their additional complexity. Cross-validation estimates variability across splits, but the folds must respect groups or time when observations are related. Final analysis includes errors, not only averages: which houses are badly predicted and which passenger subgroups are misclassified?

48.5 Core equations

Precision

$$Precision = \frac{TP}{TP + FP}$$

Among predicted positives, the fraction that are true positives.

Recall

$$Recall = \frac{TP}{TP + FN}$$

Among actual positives, the fraction that are found.

48.6 Python demonstrations

48.6.1 Demonstration 29.1: Tehran housing regression

```
import pandas as pd
from sklearn.linear_model import LinearRegression

houses = pd.read_csv("data/frozen/tehran_house_prices.csv")
area_col = next(c for c in houses.columns if c.lower() == "area")
price_col = next(c for c in houses.columns if "price" in c.lower())
clean = houses[[area_col, price_col]].apply(pd.to_numeric, errors="coerce").dropna()
model = LinearRegression().fit(clean[[area_col]], clean[price_col])
print(round(float(model.coef_[0]), 4), len(clean))
```

Verified output

```
84427668.1238 3473
```

Interpretation. The slope is a predictive association in the dataset's currency units per square metre, not a causal valuation rule.

48.6.2 Demonstration 29.2: Titanic pipeline and metrics

```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

passengers = pd.read_csv("data/frozen/titanic.csv")
```

```

target = next(c for c in passengers.columns if c.lower() == "survived")
features = [c for c in ["Pclass", "Sex", "Age", "Fare"] if c in passengers.columns]
X_train, X_test, y_train, y_test = train_test_split(passengers[features], passengers[target],
test_size=.25, random_state=29, stratify=passengers[target])
num = [c for c in features if c not in ["Sex"]]; cat = [c for c in features if c in ["Sex"]]
prep = ColumnTransformer([("num", SimpleImputer(strategy="median"), num), ("cat",
Pipeline([("imp", SimpleImputer(strategy="most_frequent")), ("ohe",
OneHotEncoder(handle_unknown="ignore"))]), cat)])
pipe = Pipeline([("prep", prep), ("model", LogisticRegression(max_iter=1000))]).fit(X_train,
y_train)
pred = pipe.predict(X_test)
print(round(accuracy_score(y_test,pred),3), round(precision_score(y_test,pred),3),
round(recall_score(y_test,pred),3))

```

Verified output

```
0.789 0.719 0.744
```

Interpretation. The three metrics illuminate different error trade-offs. Exact values are tied to the documented split and preprocessing.

48.7 Visual evidence



Figure 50. Area and listed price in the Tehran housing dataset from the IntroAI course archive.

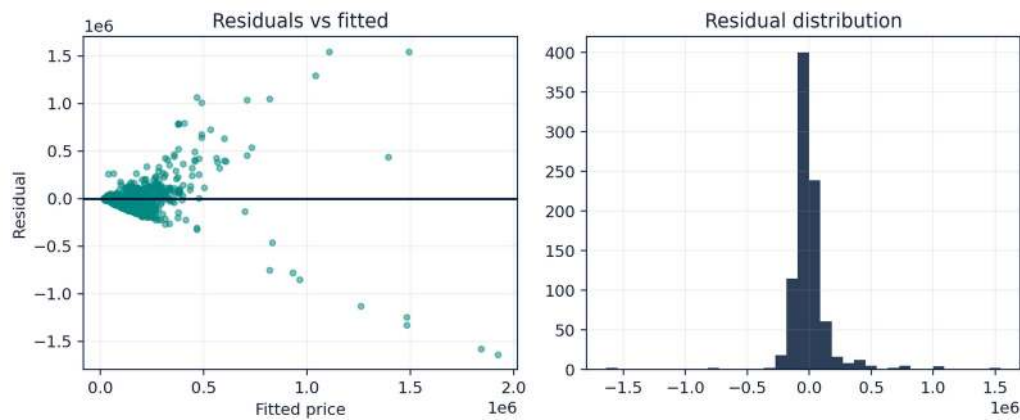


Figure 51. The linear baseline leaves large, asymmetric errors, motivating feature engineering and robust evaluation.

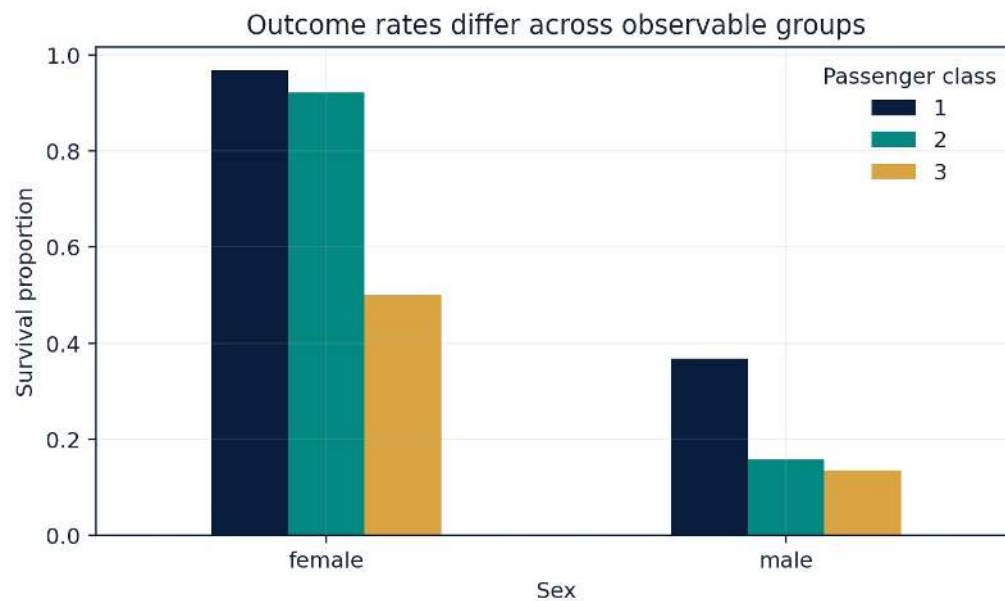


Figure 52. Survival proportions by sex and passenger class in the IntroAI Titanic dataset.

A shallow decision tree creates interpretable if-then partitions

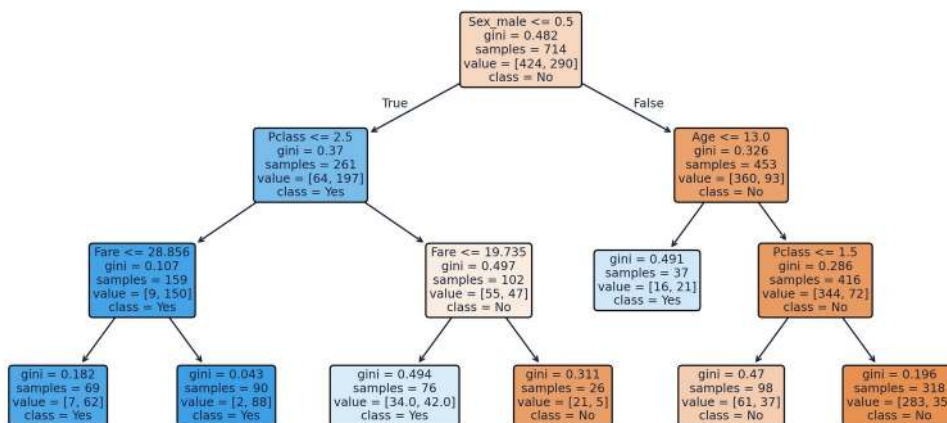


Figure 53. A depth-three decision tree for the Titanic classification problem.

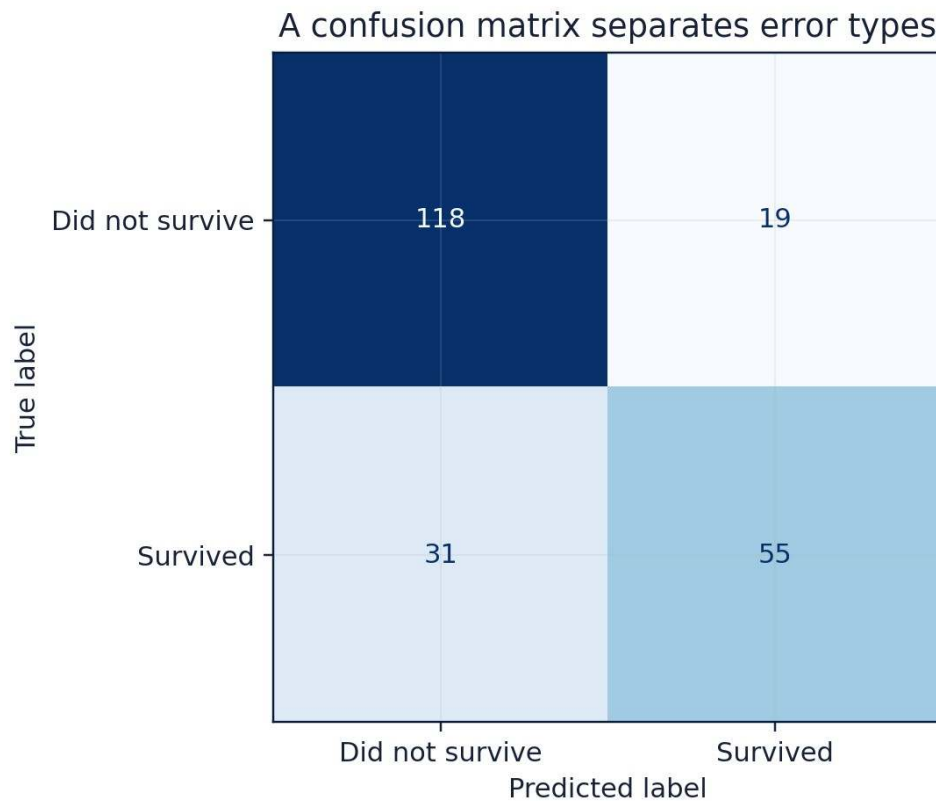


Figure 54. Out-of-sample confusion matrix for a logistic-regression pipeline.

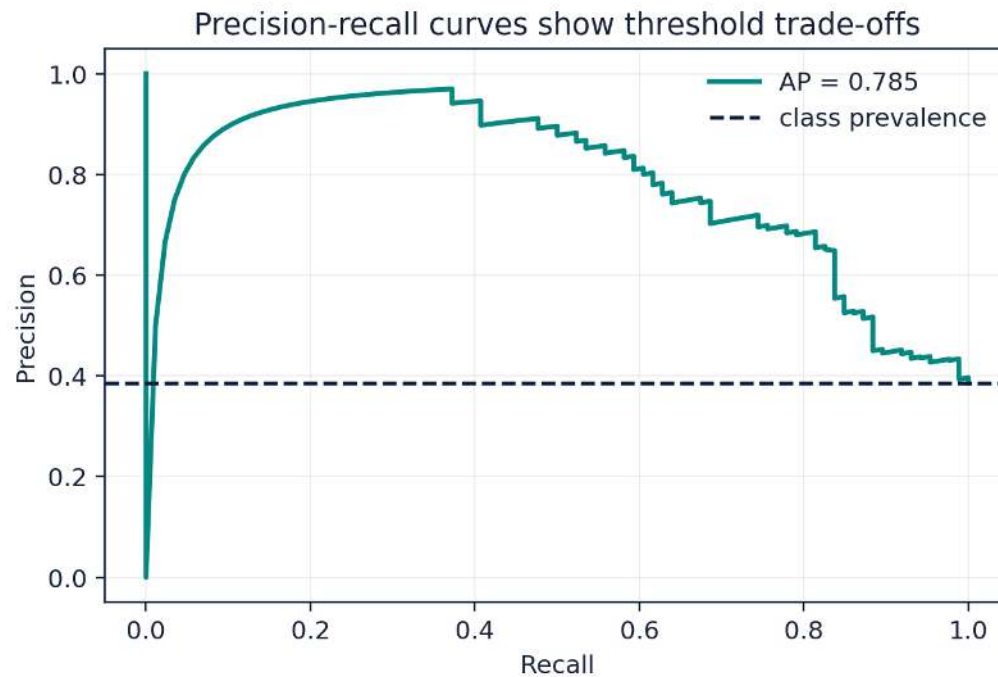


Figure 55. Precision-recall performance for the Titanic logistic-regression pipeline.

48.8 Reference table

Table 29. Chapter reference.

Metric	Best interpreted as	Limitation
MAE	average absolute prediction error	same scale but ignores direction
RMSE	square-error-sensitive error	strongly weights extremes
accuracy	overall correct fraction	misleading under imbalance
precision	reliability of positive predictions	depends on threshold/prevalence
recall	coverage of actual positives	can increase false positives
ROC-AUC	ranking across thresholds	can obscure class imbalance

Why This Matters

End-to-end projects teach that data preparation, baselines, evaluation design, and error analysis matter as much as the estimator.

Common Mistake

Reporting only the highest cross-validation score after trying many models and feature sets without preserving a final test set.

Ceteris LAB Tip

Keep a model comparison table that includes baseline, preprocessing, split, metric, and uncertainty.

R-to-Python / Source Bridge

These projects adapt the Tehran housing and Titanic notebooks from IntroAI sessions 4 to 6 under the archive's CC BY licence, with new pipelines, leakage controls, and interpretation (Sharifi-Zarchi and contributors 2026).

48.9 Guided practice

1. Re-run Demonstration 29.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

48.10 Exercises

1. Add categorical features to the housing model.
2. Compare linear, tree, and random-forest regressors.
3. Build a Titanic confusion matrix.
4. Move the classification threshold and trace precision-recall trade-offs.

48.11 Chapter summary

- Housing regression demonstrates continuous prediction and residual analysis.
- Titanic demonstrates classification, imbalance, and threshold metrics.
- Baselines and held-out evaluation make improvement interpretable.

48.12 Key Python commands

```
import pandas as pd
from sklearn.linear_model import LinearRegression
houses = pd.read_csv("data/frozen/tehran_house_prices.csv")
area_col = next(c for c in houses.columns if c.lower() == "area")
price_col = next(c for c in houses.columns if "price" in c.lower())
clean = houses[[area_col, price_col]].apply(pd.to_numeric, errors="coerce").dropna()
```

49 Neural Networks and Deep Learning for Economic Data

Opening question: **How does a stack of simple differentiable units learn a nonlinear mapping, and how do we know it has not merely memorized the training sample?**

49.1 Learning objectives

- Explain neurons, activations, layers, and forward passes.
- Describe backpropagation and optimization.
- Train a small PyTorch network.
- Use learning curves, regularization, and validation.

Prerequisites: Chapters 8, 14, and 28.

Key terms: neuron, activation, forward pass, backpropagation, epoch, regularization.

49.2 A neural network composes transformations

A neuron forms a weighted sum of inputs, adds a bias, and applies an activation function. A layer transforms a vector into another representation; multiple layers compose these transformations into a flexible nonlinear function. Without nonlinear activations, stacked linear layers collapse into one linear mapping. ReLU is common because it is simple and supports gradient-based optimization, while output activations depend on the task.

49.3 Backpropagation is organized chain-rule accounting

A forward pass computes predictions and loss. Backpropagation applies the chain rule from the loss back through each operation to calculate parameter gradients. An optimizer updates weights. Batches approximate the full-data gradient, and epochs count passes through the training data. Modern frameworks automate derivatives, but shape, scale, target coding, and loss selection remain human responsibilities (PyTorch Contributors 2026; Goodfellow, Bengio, and Courville 2016).

49.4 Learning curves reveal optimization and generalization

Training loss should generally fall, but validation loss determines whether performance transfers. A widening gap can indicate overfitting. Dropout, weight decay, early stopping, data augmentation, and simpler architectures can help. Deep models are not automatically superior on small tabular economic data, where linear or tree-based models often provide stronger baselines and clearer diagnostics.

49.5 Core equations

Neuron

$$h = \phi(w^T x + b)$$

The activation phi transforms a weighted input plus bias.

Cross-entropy

$$L = - \sum_k y_k \log(\hat{p}_k)$$

Classification loss penalizes low predicted probability for the observed class.

49.6 Python demonstrations

49.6.1 Demonstration 30.1: A small PyTorch network

```
import torch

torch.manual_seed(30)
X = torch.linspace(-1, 1, 200).reshape(-1, 1)
y = 2*X + 0.3*torch.sin(6*X)
model = torch.nn.Sequential(torch.nn.Linear(1, 12), torch.nn.ReLU(), torch.nn.Linear(12, 1))
optimizer = torch.optim.Adam(model.parameters(), lr=0.03)
loss_fn = torch.nn.MSELoss()
for _ in range(300):
    optimizer.zero_grad(); loss = loss_fn(model(X), y); loss.backward(); optimizer.step()
print(round(float(loss), 5))
```

Verified output

```
0.01343
```

Interpretation. The network learns a nonlinear approximation on a small CPU-friendly problem. The exact final loss may vary slightly by software and hardware.

49.6.2 Demonstration 30.2: Separate training from evaluation

```
model.eval()
with torch.no_grad():
    predictions = model(torch.tensor([[-0.5], [0.0], [0.5]]))
print(predictions.flatten().round(decimals=3).tolist())
```

Verified output

```
[-1.024999976158142, 0.0, 1.0119999647140503]
```

Interpretation. Evaluation mode and no_grad disable training-specific behaviour and gradient storage during prediction.

49.7 Visual evidence

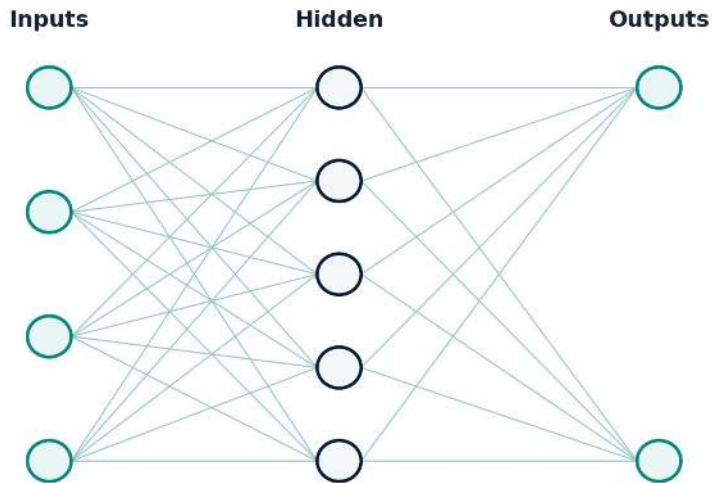


Figure 56. A feed-forward neural network transforms inputs through learned weighted connections.

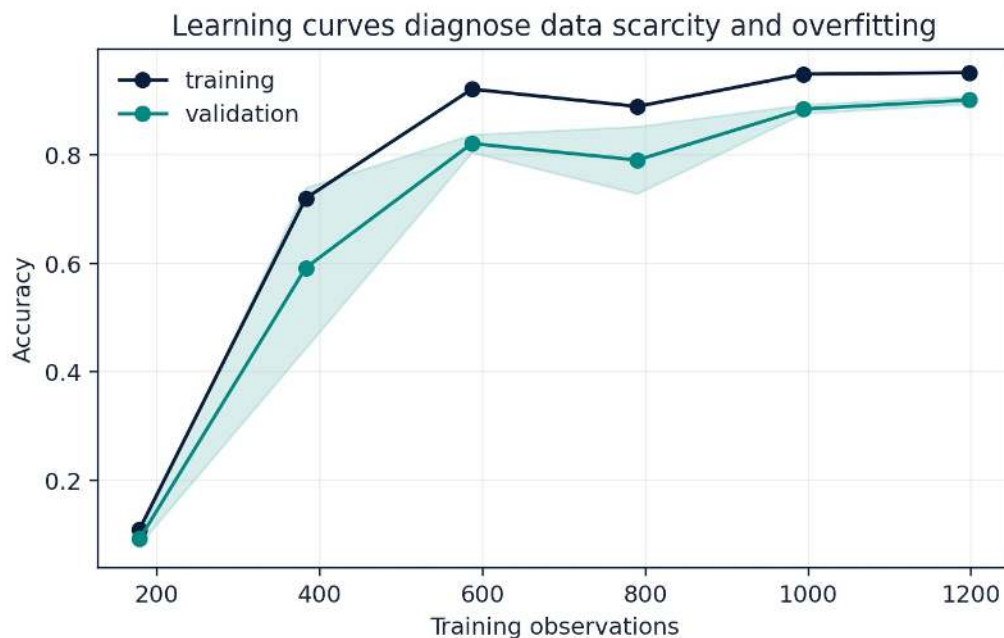


Figure 57. Training and cross-validation learning curves for a small neural classifier.

49.8 Reference table

Table 30. Chapter reference.

Component	Purpose	Typical choice
hidden activation	nonlinearity	ReLU
output activation	match target support	identity, sigmoid, softmax
loss	training objective	MSE or cross-entropy
optimizer	parameter update	SGD or Adam
regularization	reduce overfitting	weight decay/dropout
validation	generalization signal	held-out loss/metric

Why This Matters

Neural networks are flexible function approximators whose usefulness depends on data, validation, computation, and governance.

Common Mistake

Choosing a large network before establishing a linear or tree-based baseline.

Ceteris LAB Tip

Log training and validation metrics every epoch, but select the checkpoint using validation performance rather than training loss.

R-to-Python / Source Bridge

The chapter adapts IntroAI session 7, including neurons, activations, cross-entropy, backpropagation, and PyTorch, while keeping required examples small enough for CPU execution (Sharifi-Zarchi and contributors 2026).

49.9 Guided practice

1. Re-run Demonstration 30.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

49.10 Exercises

1. Build a one-hidden-layer regression network.
2. Compare ReLU and tanh activations.
3. Plot training and validation loss.
4. Add weight decay and discuss the change.

49.11 Chapter summary

- Neural networks compose linear transformations and nonlinear activations.
- Backpropagation computes gradients through the computation graph.
- Validation and regularization are essential for generalization.

49.12 Key Python commands

```
import torch
torch.manual_seed(30)
X = torch.linspace(-1, 1, 200).reshape(-1, 1)
y = 2*X + 0.3*torch.sin(6*X)
model = torch.nn.Sequential(torch.nn.Linear(1, 12), torch.nn.ReLU(), torch.nn.Linear(12, 1))
optimizer = torch.optim.Adam(model.parameters(), lr=0.03)
```

50 Computer Vision and Spatial Economic Measurement

Opening question: **How does a model transform pixels into useful local patterns without being told the exact edge or texture to search for?**

50.1 Learning objectives

- Represent images as arrays.
- Explain convolution, stride, padding, and channels.
- Describe CNNs, augmentation, and transfer learning.
- Evaluate errors, bias, and privacy.

Prerequisites: Chapters 8 and 30.

Key terms: pixel, channel, convolution, kernel, padding, transfer learning.

50.2 Images are structured tensors

A grayscale image is a two-dimensional array; a colour image commonly adds a channel axis. Pixel scale, channel order, and spatial dimensions must be documented. Resizing changes information, and normalization affects optimization. Plotting a few images and labels is a basic data audit that can reveal corrupted files, wrong orientation, or class leakage.

50.3 Convolution searches locally with shared weights

A convolution slides a small kernel across the image and computes local weighted sums. Weight sharing lets one pattern detector operate in many positions, reducing parameters relative to a fully connected layer. Stride controls movement; padding controls border treatment; multiple filters produce feature maps. Deeper layers combine edges and textures into task-specific representations.

50.4 Transfer learning is borrowed representation, not borrowed truth

A pretrained network can supply useful visual features when the new dataset is small. Fine-tuning adapts some or all weights. Performance depends on similarity between source and target images, class balance, image quality, and deployment conditions. Augmentation should represent plausible variation rather than alter the label. Vision systems can encode demographic bias and expose sensitive images, so privacy and subgroup evaluation are essential.

50.5 Core equations

Discrete convolution

$$Y_{i,j} = \sum_u \sum_v K_{u,v} X_{i+u,j+v}$$

A kernel K aggregates a local neighbourhood of image X into an output feature Y .

50.6 Python demonstrations

50.6.1 Demonstration 31.1: Apply a simple edge kernel

```
import numpy as np
from scipy.signal import convolve2d

image = np.zeros((8, 8)); image[:, 4:] = 1
kernel = np.array([[ -1, 0, 1], [ -1, 0, 1], [ -1, 0, 1]])
feature = convolve2d(image, kernel, mode="same", boundary="symm")
print(float(np.abs(feature).max()), feature.shape)
```

Verified output

```
3.0 (8, 8)
```

Interpretation. The vertical edge creates a strong response where intensity changes across columns.

50.6.2 Demonstration 31.2: Inspect a tensor shape

```
import torch

batch = torch.zeros((16, 3, 64, 64))
conv = torch.nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3, padding=1)
out = conv(batch)
print(tuple(out.shape))
```

Verified output

```
(16, 8, 64, 64)
```

Interpretation. Padding preserves height and width, while eight learned filters replace the three input channels.

50.7 Visual evidence

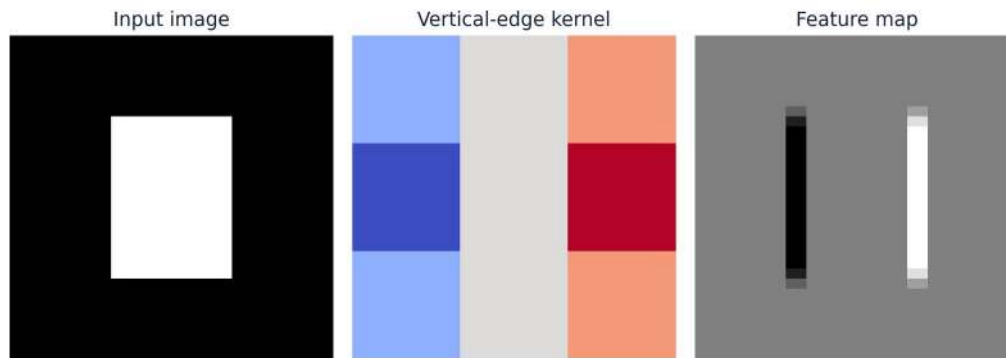


Figure 58. A small convolution kernel highlights vertical edges in an image.

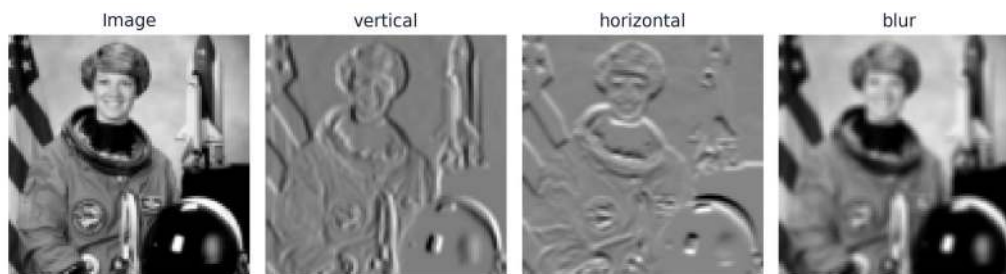


Figure 59. Different convolution filters extract different local image structures.

50.8 Reference table

Table 31. Chapter reference.

Choice	Effect	Question
kernel size	local receptive field	What pattern scale matters?
stride	downsampling	Will detail be lost?
padding	output size/border handling	How should edges be treated?
augmentation	synthetic variation	Does it preserve the label?
pretraining	initial representation	Is the source domain relevant?

Why This Matters

Computer vision combines spatial structure with learned representations, but deployment validity depends on the image-generating context.

Common Mistake

Applying augmentation that changes the label, such as flipping a directional medical image without checking domain meaning.

Ceteris LAB Tip

Inspect misclassified images by subgroup and acquisition source, not only the aggregate accuracy.

R-to-Python / Source Bridge

The chapter adapts IntroAI session 8 on convolution, padding, channels, augmentation, CNNs, and transfer learning, using new original diagrams instead of source screenshots (Sharifi-Zarchi and contributors 2026).

50.9 Guided practice

1. Re-run Demonstration 31.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

50.10 Exercises

1. Convolve an image with blur and edge kernels.
2. Compare output shapes under two padding choices.
3. Design three label-preserving augmentations.
4. Write a transfer-learning evaluation plan.

50.11 Chapter summary

- Images are tensors with spatial and channel structure.
- Convolution uses local shared filters.
- Transfer learning and augmentation require domain-valid assumptions.

50.12 Key Python commands

```
import numpy as np
from scipy.signal import convolve2d
image = np.zeros((8, 8)); image[:, 4:] = 1
kernel = np.array([[ -1, 0, 1], [ -1, 0, 1], [ -1, 0, 1]])
feature = convolve2d(image, kernel, mode="same", boundary="symm")
print(float(np.abs(feature).max()), feature.shape)
```

51 Natural-Language Processing and Transformers for Economics

Opening question: **How can text be converted into numerical representations while preserving enough context to classify or retrieve meaning?**

51.1 Learning objectives

- Tokenize and vectorize text.
- Use TF-IDF and word embeddings.
- Explain cosine similarity, attention, and transformers.
- Evaluate text classifiers and language limitations.

Prerequisites: Chapters 28 and 30.

Key terms: tokenization, TF-IDF, embedding, cosine similarity, attention, transformer.

51.2 Tokenization chooses the model's alphabet

Text must be divided into tokens before a model can process it. Tokens may be words, subwords, characters, or byte-level units. The choice affects vocabulary size, rare words, multilingual coverage, and sequence length. Cleaning decisions such as lowercasing or punctuation removal can discard useful information. A reproducible text pipeline stores both preprocessing and model parameters.

51.3 Representations move from counts to geometry

Bag-of-words and TF-IDF represent documents by weighted token counts. They are strong, interpretable baselines for many classification and retrieval tasks. Embeddings place tokens or documents in a continuous vector space where geometric proximity can encode usage similarity. Cosine similarity compares direction rather than magnitude, but similarity does not guarantee factual equivalence or social neutrality.

51.4 Attention connects tokens conditionally

Attention computes how strongly each token should use information from other tokens. Queries, keys, and values produce weighted combinations, and multi-head attention learns several relation patterns. Transformers combine attention with feed-forward layers, residual connections, and normalization (Vaswani et al. 2017). They can model long dependencies more directly than recurrent networks, but computation grows with sequence length and outputs inherit training-data limitations.

51.5 Core equations

Cosine similarity

$$\cos(a, b) = \frac{a^\top b}{\|a\| \|b\|}$$

The measure compares vector direction and ranges from -1 to 1 for nonzero real vectors.

Scaled attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Token values are combined using similarity-based weights.

51.6 Python demonstrations

51.6.1 Demonstration 32.1: TF-IDF document similarity

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

docs = ["inflation rose after energy prices increased", "energy costs pushed inflation
higher", "the football match ended in a draw"]
X = TfidfVectorizer().fit_transform(docs)
print(cosine_similarity(X[0], X).round(3).tolist()[0])
```

Verified output

```
[1.0, 0.25, 0.0]
```

Interpretation. The two inflation documents share weighted vocabulary, while the unrelated sports sentence has zero overlap in this tiny corpus.

51.6.2 Demonstration 32.2: A text-classification baseline

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

texts = ["flood warning issued", "sunny picnic today", "earthquake reported", "great concert
tonight"]
labels = [1, 0, 1, 0]
pipe = Pipeline([("tfidf", TfidfVectorizer()), ("model", LogisticRegression())]).fit(texts, labels)
print(pipe.predict(["storm warning tonight"]).tolist())
```

Verified output

```
[0]
```

Interpretation. The example demonstrates workflow, not reliable disaster detection. Four training texts are far too few for deployment.

51.7 Visual evidence



Keep train/test boundaries intact when learning vocabulary and feature weights.

Figure 60. A text-classification workflow from raw language to evaluated predictions.

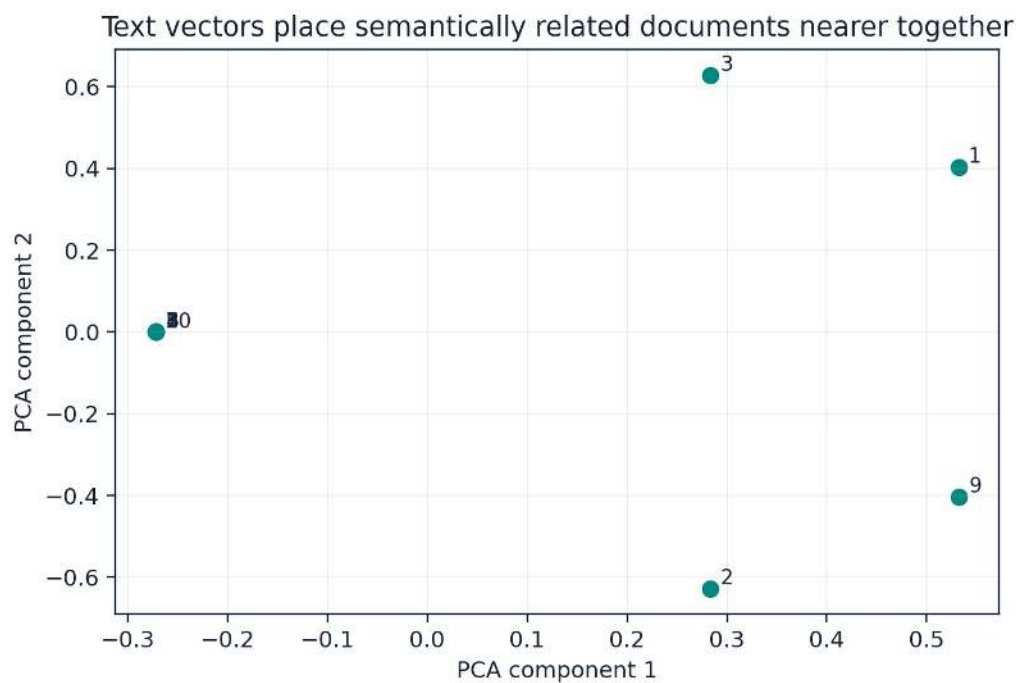


Figure 61. A two-dimensional projection of TF-IDF document vectors.

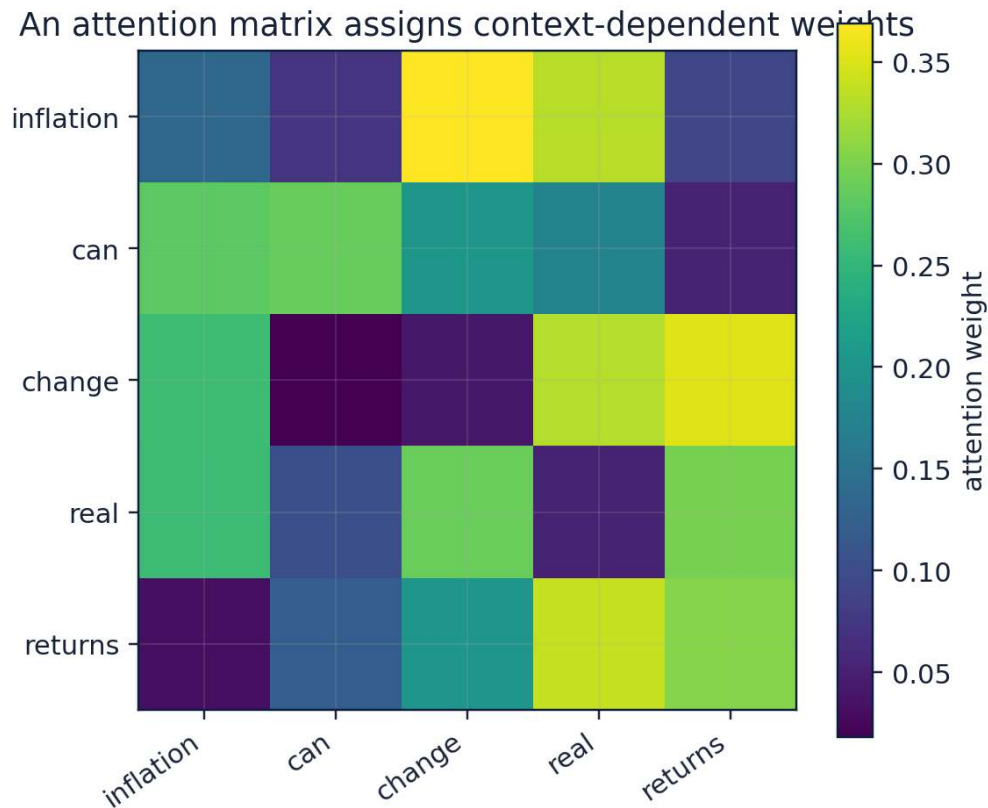


Figure 62. A synthetic attention-weight matrix for a five-token sentence.

51.8 Reference table

Table 32. Chapter reference.

Representation	Strength	Limitation
bag of words	simple and interpretable	ignores order
TF-IDF	strong sparse baseline	limited semantics
static embedding	compact semantic geometry	one vector per word
contextual embedding	context-sensitive	computational and opaque
attention map	relation visualization	not a complete explanation

Why This Matters

NLP representations shape what textual evidence a model can use and what distinctions it can miss.

Common Mistake

Treating attention weights as a definitive causal explanation of a model's decision.

Ceteris LAB Tip

Establish a TF-IDF baseline before adding a transformer. It reveals whether complexity produces real improvement.

R-to-Python / Source Bridge

The chapter adapts IntroAI session 9 on tokenization, embeddings, cosine similarity, attention, transformers, and disaster-tweet classification, with explicit baseline and evaluation guidance (Sharifi-Zarchi and contributors 2026).

51.9 Guided practice

1. Re-run Demonstration 32.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

51.10 Exercises

1. Compare word and character TF-IDF.
2. Calculate cosine similarities for five documents.
3. Build a train-test text classifier.
4. Explain how subword tokenization helps rare words.

51.11 Chapter summary

- Text must be tokenized and represented numerically.
- Sparse TF-IDF is a strong baseline; embeddings add geometric representation.
- Transformers use attention to construct contextual features.

51.12 Key Python commands

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
X = TfidfVectorizer().fit_transform(docs)
print(cosine_similarity(X[0], X).round(3).tolist()[0])
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

52 Language Models, Retrieval, and AI Agents for Econometric Research

Opening question: **How can a language model answer from a controlled evidence collection, use tools, and still remain subject to verification?**

52.1 Learning objectives

- Explain next-token prediction and context windows.
- Build a simple retrieval-augmented workflow.
- Describe agent loops, tools, memory, and stopping conditions.
- Evaluate grounding, privacy, cost, and safety.

Prerequisites: Chapters 27, 28, and 32.

Key terms: language model, context window, RAG, retrieval, agent, tool use.

52.2 A language model predicts continuations

A language model estimates a probability distribution over next tokens given prior context. Repeated sampling produces text. The model can encode extensive patterns without storing a searchable database of exact facts. Context windows limit how much prompt and retrieved material can be considered at once. Generated probability does not provide source attribution or truth verification.

52.3 Retrieval creates an evidence channel

Retrieval-augmented generation separates document search from response generation. Documents are split into chunks, represented for retrieval, ranked against a query, and inserted into the model context. The response can then cite the retrieved evidence. Failure can occur at ingestion, chunking, indexing, retrieval, context selection, or generation. A grounded answer requires both relevant evidence and faithful use of that evidence.

52.4 Agents are loops with authority

An AI agent typically observes a goal and state, chooses an action, calls a tool, receives a result, updates memory, and decides whether to continue. Tool permissions, input validation, rate limits, budgets, and stopping conditions are safety controls. Memory may improve continuity while increasing privacy and contamination risks. An agent should not receive more authority than the task requires, and consequential actions need confirmation and audit logs.

52.5 Python demonstrations

52.5.1 Demonstration 33.1: A local retrieval demonstration

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

chunks = [
    "Value at Risk is a loss quantile at a stated horizon and confidence level.",
    "Expected Shortfall averages losses beyond the VaR threshold.",
    "An AR model relates a series to its own lags.",
]
query = "What summarizes losses worse than VaR?"
vec = TfidfVectorizer().fit(chunks + [query])
X = vec.transform(chunks + [query])
scores = cosine_similarity(X[:-1], X[-1]).ravel()
print(scores.argmax(), chunks[scores.argmax()])
```

Verified output

```
1 Expected Shortfall averages losses beyond the VaR threshold.
```

Interpretation. The retrieval step is local, inspectable, and requires no paid API. A larger system needs better embeddings and evaluation.

52.5.2 Demonstration 33.2: A bounded tool loop

```
state = {"step": 0, "done": False}
while not state["done"] and state["step"] < 3:
    state["step"] += 1
    if state["step"] == 2:
        state["done"] = True
print(state)
```

Verified output

```
{'step': 2, 'done': True}
```

Interpretation. A maximum-step guard prevents an unbounded loop. Real agents also need tool-specific permissions, validation, and human confirmation.

52.6 Visual evidence

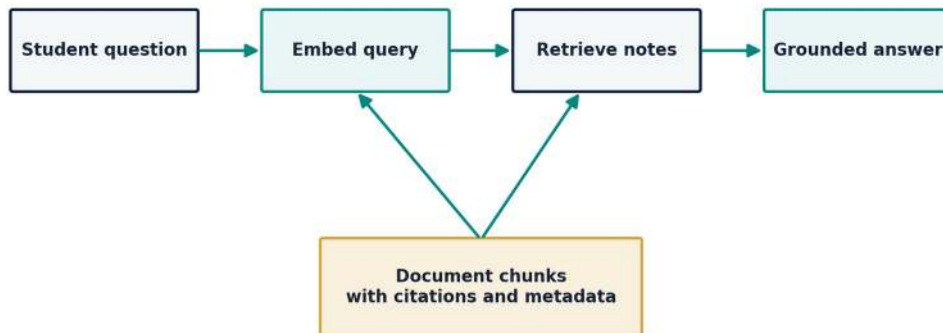


Figure 63. Retrieval-augmented generation grounds a response in selected source passages.

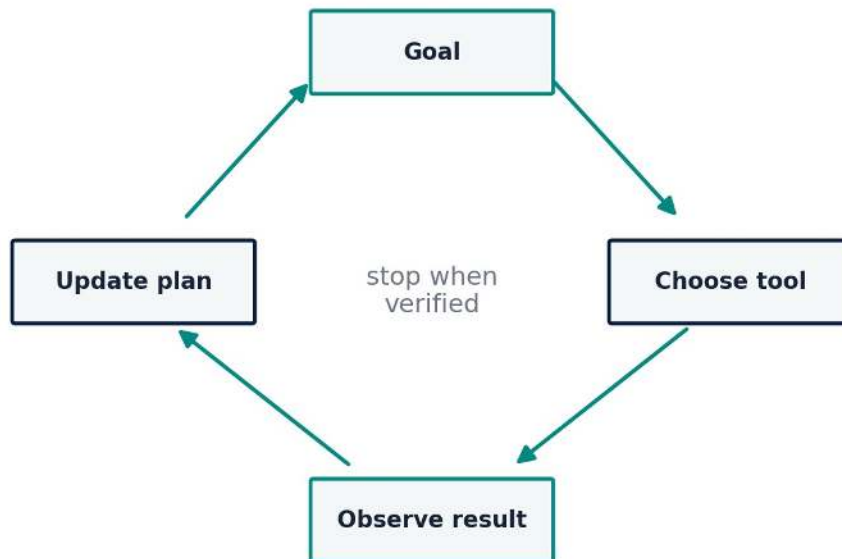


Figure 64. A tool-using agent alternates planning, action, observation, and verification.

52.7 Reference table

Table 33. Chapter reference.

Component	Function	Failure mode
chunker	divide documents	cuts crucial context
retriever	rank evidence	misses relevant source
generator	compose answer	ignores or distorts evidence

Component	Function	Failure mode
tool	perform action or query	unsafe/invalid invocation
memory	retain state	privacy or stale assumptions
evaluator	check answer and process	weak metric or circular grading

Why This Matters

Retrieval and tools can improve grounding and capability, but they add new failure surfaces that must be observed and tested.

Common Mistake

Assuming that adding documents to a vector store guarantees that the model will retrieve and cite the correct evidence.

Ceteris LAB Tip

Evaluate retrieval separately from generation: first ask whether the right chunk was found, then whether the answer used it faithfully.

R-to-Python / Source Bridge

The chapter adapts IntroAI session 10 on small language models, chatbots, retrieval, memory, agents, and tool use. The required lab remains local and no-API, with optional extensions clearly separated (Sharifi-Zarchi and contributors 2026).

52.8 Guided practice

1. Re-run Demonstration 33.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

52.9 Exercises

1. Create a TF-IDF retriever for five course notes.
2. Design three chunking strategies and compare them.
3. Specify tools and permissions for a safe data-analysis agent.
4. Write a stopping rule and budget for an agent loop.

52.10 Chapter summary

- Language models generate token continuations rather than verified facts.
- RAG adds a retrievable evidence path.
- Agents require bounded tools, memory controls, and stopping conditions.

52.11 Key Python commands

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
vec = TfidfVectorizer().fit(chunks + [query])
X = vec.transform(chunks + [query])
```

```
scores = cosine_similarity(X[-1], X[:-1]).ravel()
print(scores.argmax(), chunks[scores.argmax()])
```

53 Capstone Projects and Student Portfolio

Opening question: **How can a student transform code fragments into a defensible, reproducible analytical product for the Ceteris LAB website or a professional portfolio?**

53.1 Learning objectives

- Plan a complete project from question to communication.
- Use authoritative data and a data dictionary.
- Compare baseline and advanced models out of sample.
- Package code, figures, limitations, and reproducibility files.

Prerequisites: All previous chapters.

Key terms: capstone, data dictionary, benchmark, model comparison, reproducible report, portfolio.

53.2 Capstone A: Canadian exchange-rate forecasting

Use the Bank of Canada FXUSDCAD series or another official Canadian series. State a forecast question and horizon, preserve the frozen raw extract, document units and frequency, transform the series, compare a naive benchmark with at least two models, and evaluate on a later period. Discuss revisions, market closures, short samples, and the difference between statistical forecast accuracy and economic value.

53.3 Capstone B: inflation and interest-rate dashboard

Build a reproducible dashboard from official Canadian CPI and policy-rate sources. Distinguish headline, core, monthly, and year-over-year measures. Add source notes, retrieval dates, and an explanation of revisions. Use charts to communicate trends, not to imply that co-movement proves a policy effect. Include a plain-language summary and a technical appendix.

53.4 Capstones C and D: risk and AI

A financial-risk project can compare historical, parametric, and GARCH-based VaR and ES with an expanding-window backtest. An AI project can compare baseline and advanced models on a clearly licensed dataset, preserve a final test set, inspect errors, and discuss fairness, privacy, and deployment shift. Every portfolio project should include a README, environment file, notebook, script, data instructions, figures, and an honest limitations section.

53.5 Python demonstrations

53.5.1 Demonstration 34.1: A simple model-comparison table

```
import pandas as pd

results = pd.DataFrame({
    "model": ["naive", "ARIMA", "random_forest"],
    "MAE": [0.0124, 0.0118, 0.0121],
    "RMSE": [0.0168, 0.0160, 0.0164],
})
results["MAE_rank"] = results["MAE"].rank(method="min").astype(int)
print(results.sort_values("MAE"))
```

Verified output

	model	MAE	RMSE	MAE_rank
1	ARIMA	0.0118	0.0160	1
2	random_forest	0.0121	0.0164	2
0	naive	0.0124	0.0168	3

Interpretation. The advanced model improves only modestly over the naive benchmark, so uncertainty and practical relevance should accompany the ranking.

53.5.2 Demonstration 34.2: A reproducibility manifest

```
from pathlib import Path

required = ["README.md", "requirements.txt", "data", "notebooks", "scripts", "figures"]
root = Path.cwd()
manifest = {item: (root/item).exists() for item in required}
print(manifest)
```

Verified output

```
{'README.md': True, 'requirements.txt': True, 'data': True, 'notebooks': True, 'scripts': True, 'figures': True}
```

Interpretation. A manifest is not proof of quality, but it catches missing package components before publication.

53.6 Visual evidence

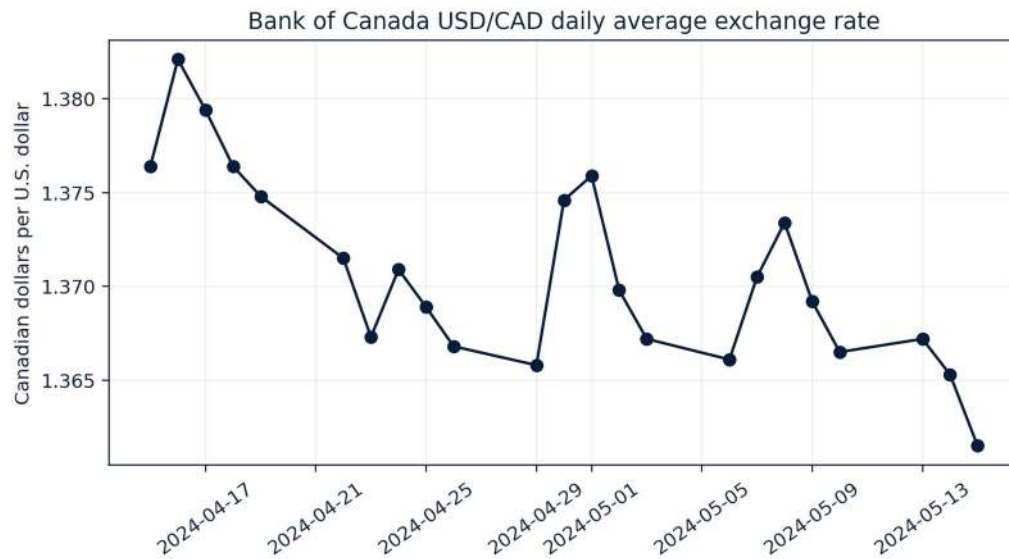


Figure 65. Official Bank of Canada FXUSDCAD observations from 15 April to 15 May 2024.

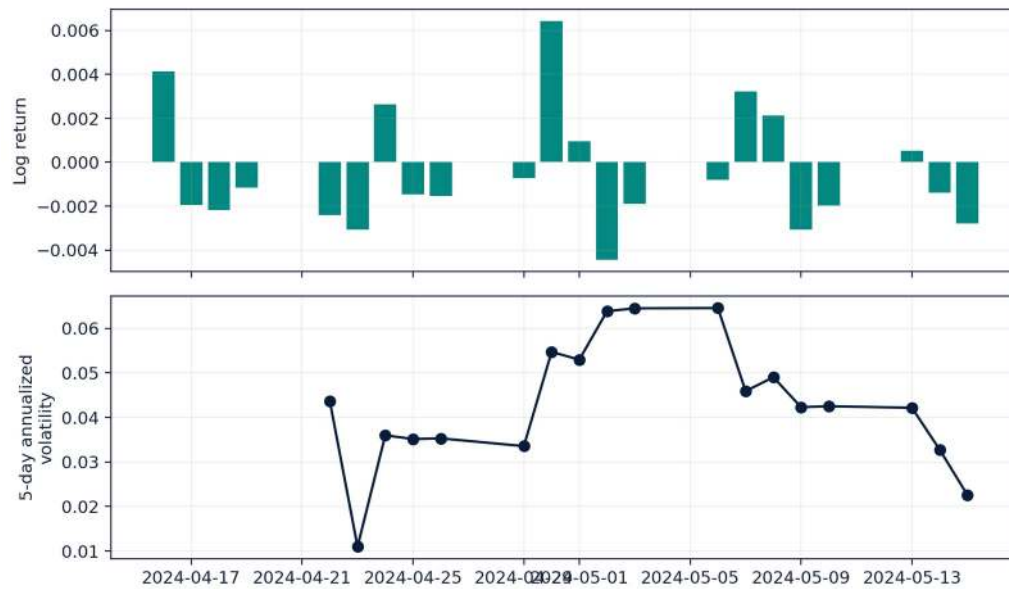


Figure 66. USD/CAD log changes and a short-window annualized volatility estimate.

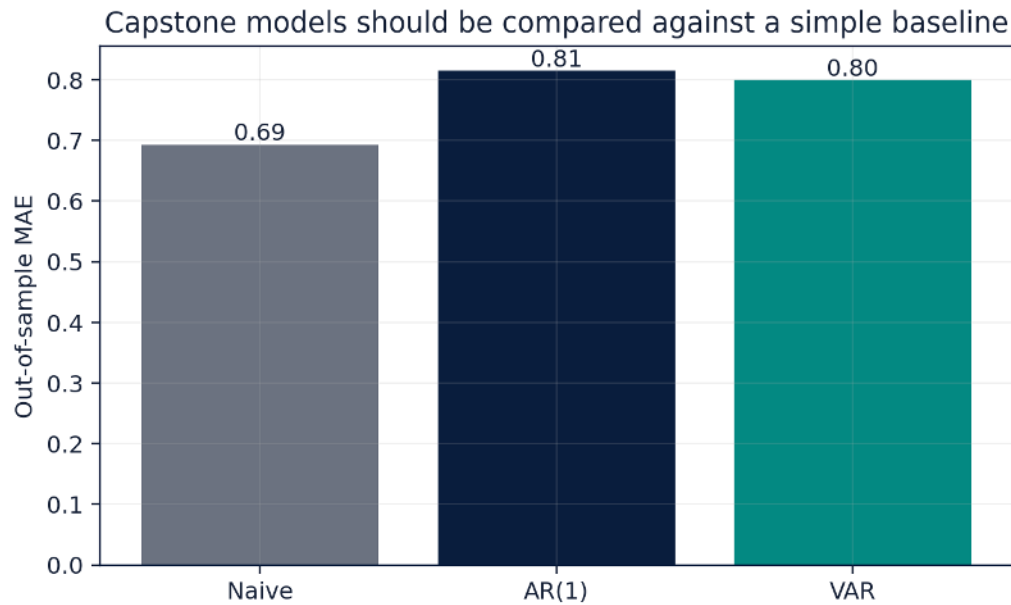


Figure 67. Out-of-sample mean absolute error for three illustrative GDP-growth forecasts.

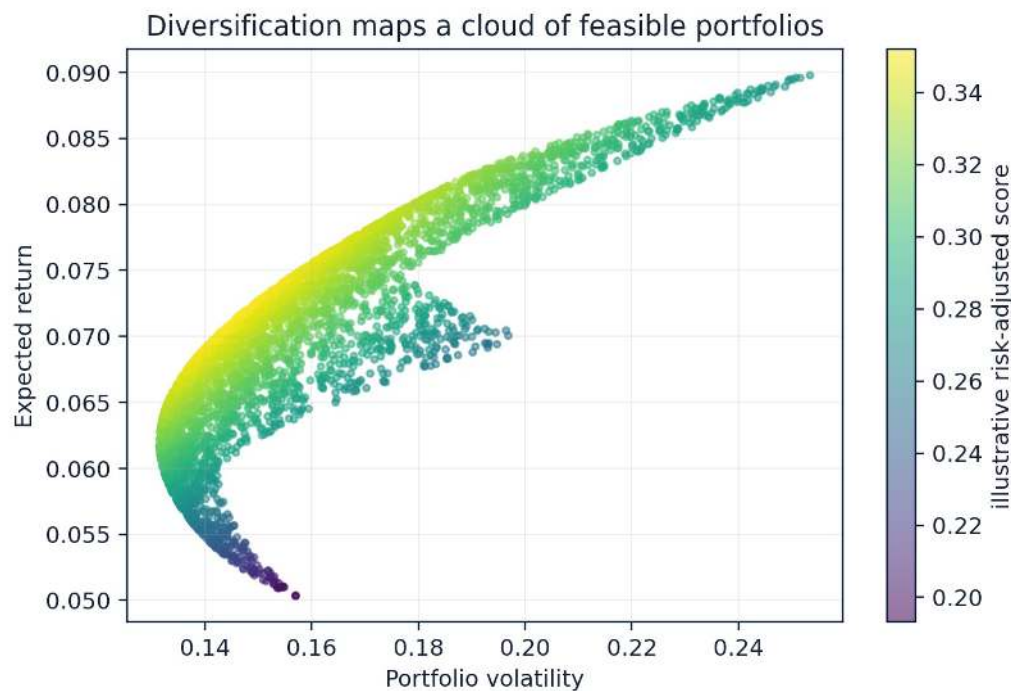


Figure 68. A simulated three-asset opportunity set generated from random portfolio weights.

53.7 Reference table

Table 34. Chapter reference.

Deliverable	Minimum content	Acceptance question
research question	population, outcome, horizon	Is it answerable with available data?
data package	source, licence, dictionary, retrieval	Can another student obtain it?

Deliverable	Minimum content	Acceptance question
analysis	baseline, model, diagnostics	Does code run top to bottom?
evaluation	held-out design and metrics	Does it mimic deployment?
communication	figures, interpretation, limits	Are claims proportional to evidence?
reproducibility	README and environment	Can the result be rebuilt?

Why This Matters

A capstone becomes professional when the data, code, evaluation, communication, and limitations form one auditable argument.

Common Mistake

Choosing a project because a complex model is available rather than because the question matters and the data can support it.

Ceteris LAB Tip

Begin the README on the first day. It will expose missing decisions while they are still easy to correct.

R-to-Python / Source Bridge

The capstones combine the strongest elements of the financial-time-series lectures, the IntroAI projects, and new Canadian applications into website-ready student work.

53.8 Guided practice

1. Re-run Demonstration 34.1 and change one input while keeping the analytical question fixed.
2. Explain in two sentences how the output supports, or fails to support, the chapter opening question.
3. Add one validation check that would prevent a plausible error.

53.9 Exercises

1. Draft a one-paragraph project question and scope.
2. Create a data dictionary before modeling.
3. Define a naive benchmark and final test design.
4. Build a publication checklist covering data rights, code execution, figures, and claims.

53.10 Chapter summary

- Capstones integrate question, data, model, evaluation, and communication.
- Benchmarks and held-out tests keep complexity honest.
- A portfolio artifact includes reproducibility and limitations, not only attractive charts.

53.11 Key Python commands

```
import pandas as pd
results = pd.DataFrame({
results["MAE_rank"] = results["MAE"].rank(method="min").astype(int)
print(results.sort_values("MAE"))
from pathlib import Path
root = Path.cwd()
```

35 Appendix A. Python Quick Reference

Task	Example
Assignment	<code>rate = 0.03</code>
Type check	<code>type(rate)</code>
List slice	<code>values[1:4]</code>
Dictionary lookup	<code>units["CPI"]</code>
Loop	<code>for value in values:</code>
Function	<code>def transform(x): return ...</code>
Project path	<code>Path("data") / "file.csv"</code>
NumPy array	<code>np.array(values)</code>
pandas import	<code>pd.read_csv(path)</code>
Filter	<code>df.loc[df["x"] > 0]</code>
Group	<code>df.groupby("group").agg(...)</code>
Plot	<code>fig, ax = plt.subplots()</code>
Regression	<code>smf.ols("y ~ x", data=df).fit()</code>
ARIMA	<code>ARIMA(y, order=(p,d,q)).fit()</code>
VAR	<code>VAR(df).fit(lags)</code>
Pipeline	<code>Pipeline([...]).fit(X, y)</code>

36 Appendix B. Installation and Troubleshooting

1. Install Python 3.13 or 3.14 from the official Python distribution, or use Google Colab.
2. From the project root, create a virtual environment: `python -m venv .venv`.
3. Activate it. Windows PowerShell: `.venv\Scripts\Activate.ps1`. macOS/Linux: `source .venv/bin/activate`.
4. Install the tested dependencies: `python -m pip install -r requirements.txt`.
5. Verify the environment: `python scripts/verify_environment.py`.
6. Start JupyterLab: `jupyter lab`.
7. Open notebooks in numerical order and use **Restart Kernel and Run All**.

Troubleshooting sequence: print `sys.executable`; confirm the active kernel; verify the project root; inspect the missing package name; reinstall from the requirements file; avoid installing packages into random notebook cells.

37 Appendix C. R-to-Python Crosswalk

The complete conversion ledger contains every substantive detected R/code segment and its disposition. The following table summarizes the most common bridges.

37.0.1 `read.table()` / `read.csv()` to `pandas.read_csv()`

Specify separators, dates, missing markers, and dtypes explicitly.

37.0.2 `head()` / `tail()` to `.head()` / `.tail()`

Methods belong to Series/DataFrame objects.

37.0.3 `dim()` to `.shape`

Attribute, not function call.

37.0.4 R vector to list, NumPy array, or pandas Series

Choose structure according to labels and numerical operations.

37.0.5 `ts()` to pandas Series with DatetimeIndex/PeriodIndex

Frequency and calendar metadata are explicit.

37.0.6 `diff()` to `.diff()` or `numpy.diff()`

pandas preserves index and inserts missing first value.

37.0.7 `acf()` / `pacf()` to `statsmodels.tsa.stattools.acf/pacf`

Default estimators and confidence intervals can differ.

37.0.8 `Box.test(..., type="Ljung")` to `acorr_ljungbox()`

Account for fitted-model degrees of freedom.

37.0.9 `lm()` to `statsmodels` OLS or `scikit-learn` regression

Choose inference or prediction objective.

37.0.10 `arima()` to `statsmodels.tsa.arima.model.ARIMA`

Intercept, trend, and MA-sign conventions require checking.

37.0.11 `arma.sim()` to `ArmaProcess.generate_sample()`

AR/MA polynomial signs follow statsmodels convention.

37.0.12 `garchFit()` to arch package or transparent MLE

Return scaling and standardized Student-t definitions differ.

37.0.13 `polr(..., method="probit")` to `OrderedModel(..., distr="probit")`

Threshold parameterization and exogenous constants differ.

37.0.14 `qnorm()` / `qt()` to `scipy.stats.norm.ppf()` / `t.ppf()`

Standardized Student-t requires an explicit variance adjustment.

37.0.15 `rnorm()` to `numpy.random.Generator.normal()`

Local generator objects avoid hidden global state.

37.0.16 Base R plot to Matplotlib figure/axes

Object-oriented figure control supports reproducible export.

37.0.17 Custom VAR scripts to `statsmodels.tsa.api.VAR`

Identification and deterministic terms must be specified.

37.0.18 urca Johansen tools to `statsmodels.tsa.vector_ar.vecm`

Normalization and deterministic-term conventions differ.

38 Appendix D. Mathematical Notation

Symbol	Meaning
t	time index
i	observation index
X_t	scalar time-series observation
$\mathbf{X}_t$	vector time-series observation
$E(\cdot)$	expectation
$Var(\cdot)$	variance
a_t or ε_t	innovation or model error
σ_t^2	conditional variance
B	lag operator, $BX_t = X_{t-1}$
p, d, q	AR order, differencing order, MA order
s	seasonal period
Φ	standard normal cumulative distribution function
L	loss variable
θ	generic model parameter vector
L	loss function or likelihood, according to context

39 Appendix E. Glossary

ACF. Correlation between a series and its lags under a specified sample estimator.

API. A documented interface through which software requests data or services.

ARCH. A conditional-variance model driven by lagged squared shocks.

ARIMA. Autoregressive integrated moving-average model.

Attention. A mechanism that forms weighted combinations of values using query-key similarities.

Backtesting. Evaluation of forecasts or risk thresholds using information available before each outcome.

Bias. Systematic deviation that may arise from data, labels, sampling, model assumptions, or deployment.

Brownian motion. Continuous stochastic process with independent normal increments.

Cointegration. Stationary linear combination among nonstationary series.

Conditional variance. Variance given an information set.

Confusion matrix. Counts of predicted and actual classification outcomes.

Cross-validation. Repeated training-validation procedure for estimating generalization.

Data leakage. Use of information during training that would not be available at prediction time.

DataFrame. Two-dimensional labelled pandas data structure.

Deep learning. Machine learning using multi-layer neural networks.

Embedding. Continuous vector representation of an item such as a token or document.

Expected Shortfall. Mean loss beyond a specified VaR quantile under the adopted definition.

Feature. Input variable supplied to a predictive model.

Forecast origin. Latest time point whose information is available when a forecast is made.

GARCH. Conditional-variance model using lagged shocks and lagged variance.

Gradient descent. Iterative parameter update opposite a loss gradient.

Granger predictability. Incremental predictive content of one variable's lags for another within a model.

Hallucination. Generated content that is unsupported or incorrect despite fluent presentation.

Index. Labels identifying pandas rows or array positions, depending on context.

Inference. Reasoning from sample evidence to uncertainty about model quantities under assumptions.

Kernel. Small matrix applied locally in a convolution.

Latent variable. Unobserved quantity represented indirectly through observed outcomes.

Log return. Difference of logarithmic prices over a period.

MAE. Mean absolute error.

Monte Carlo. Numerical approximation using repeated random simulation.

PACF. Correlation at a lag after removing linear effects of shorter lags.

Pipeline. Ordered preprocessing and modeling operations fitted as one workflow.

Precision. True positives divided by predicted positives.

RAG. Retrieval-augmented generation.

RMSE. Square root of mean squared prediction error.

Recall. True positives divided by actual positives.

Reproducibility. Ability to rebuild results from documented code, data, environment, and assumptions.

Residual. Observed outcome minus fitted value.

Seasonality. Dependence or pattern recurring at a calendar frequency.

Stationarity. Time-invariant mean and covariance structure under the adopted weak definition.

Tokenization. Division of text into units processed by a model.

Transformer. Neural architecture built around attention and feed-forward blocks.

Unit root. Root associated with persistent stochastic trend.

VAR. Vector autoregression.

Value at Risk. Specified quantile of a loss distribution.

Vectorization. Array-level expression of repeated numerical operations.

White noise. Uncorrelated, zero-mean, constant-variance sequence under the weak definition.

40 Appendix F. Selected Exercise Guidance

40.0.1 Chapter 1: Welcome to Python and Ceteris LAB

Selected guidance. Run the miniature analysis twice, first unchanged and then with one value altered. Confirm that the mean and chart change consistently, and explain why a reproducible notebook should preserve both the input and the resulting output.

40.0.2 Chapter 2: Installing and Running Python

Selected guidance. For an installation problem, report `sys.executable`, the selected Jupyter kernel, and the result of importing the missing package. A strong answer diagnoses the layer that failed instead of proposing a blind reinstall.

40.0.3 Chapter 3: Variables, Values, Types, Strings, and Dates

Selected guidance. Create variables for a policy rate, country name, observation count, release date, and missing value. Check each type explicitly, then explain why formatting a percentage does not change the stored numerical value.

40.0.4 Chapter 4: Collections, Indexing, and Slicing

Selected guidance. Use a list for an ordered series, a tuple for an immutable coordinate, a dictionary for metadata, and a set for unique identifiers. Justify each choice by the operations the object must support.

40.0.5 Chapter 5: Decisions, Loops, and Iteration

Selected guidance. Trace the loop by hand for the first three observations before running it. Then write a vectorized or comprehension-based alternative and compare readability, not only output equality.

40.0.6 Chapter 6: Functions, Modules, Errors, and Testing

Selected guidance. Test a function with a normal input, a boundary input, and an invalid input. The solution should distinguish an assertion used during development from an exception that communicates a recoverable user error.

40.0.7 Chapter 7: Files, Paths, Projects, and Reproducibility

Selected guidance. Move the project folder to a new location and rerun the notebook. A successful solution uses relative pathlib paths, preserves raw data, writes processed output separately, and records data provenance beside the file.

40.0.8 Chapter 8: NumPy and Numerical Computing

Selected guidance. Compute portfolio returns once with a Python loop and once with a NumPy dot product. Verify equality with `np.allclose`, inspect array shapes, and state which unintended broadcast could produce a plausible but incorrect result.

40.0.9 Chapter 9: pandas Fundamentals

Selected guidance. Build a DataFrame with labels, dates, and one missing observation. Filter it with `.loc`, correct the dtype, and explain why the index is part of the analytical structure rather than decorative row numbering.

40.0.10 Chapter 10: Importing, Cleaning, Validating, and Exporting Data

Selected guidance. Write a cleaning pipeline that parses dates, standardizes missing markers, removes exact duplicates, and flags an invalid range. Report row counts before and after every step so that no observation disappears silently.

40.0.11 Chapter 11: Manipulating, Grouping, Reshaping, and Joining Data

Selected guidance. Before merging, verify the key uniqueness required by the intended relationship. Use `validate=` and `indicator=True`, then investigate every unmatched row instead of discarding it automatically.

40.0.12 Chapter 12: Data Visualization with Python

Selected guidance. Choose a chart for a time trend, a distribution, and a relationship between two variables. Label units and sources, avoid a truncated axis that exaggerates change, and describe one accessibility feature beyond colour choice.

40.0.13 Chapter 13: Descriptive Statistics and Exploratory Data Analysis

Selected guidance. Compare mean and median under a deliberately inserted outlier. Report skewness using the stated convention, and explain why a single summary statistic cannot describe location, spread, and tail shape simultaneously.

40.0.14 Chapter 14: Probability, Simulation, and Statistical Inference

Selected guidance. Simulate repeated sample means with a documented random seed. Compare the empirical standard deviation with the theoretical standard error, then repeat with a larger sample size and explain the square-root relationship.

40.0.15 Chapter 15: Regression and Econometrics with Python

Selected guidance. Fit the same regression with conventional and heteroskedasticity-robust standard errors. Keep coefficients fixed, compare uncertainty, inspect residuals, and avoid turning an association into a causal claim without an identification argument.

40.0.16 Chapter 16: Obtaining Economic and Financial Data

Selected guidance. Retrieve or load the Bank of Canada series, retain its series identifier and units, and compare it with the frozen fallback. The solution should fail gracefully when the network is unavailable and should never invent replacement observations.

40.0.17 Chapter 17: Introduction to Time-Series Data

Selected guidance. Transform a price level into simple and log returns. Check missing trading dates and aggregation rules, then explain why summing log returns and compounding simple returns answer the same multi-period question differently.

40.0.18 Chapter 18: Dependence, Stationarity, and White Noise

Selected guidance. Simulate white noise and a persistent AR process. Compare their ACF and Ljung-Box results, then show that absence of linear autocorrelation does not by itself establish independence.

40.0.19 Chapter 19: AR, MA, ARMA, ARIMA, and Forecasting

Selected guidance. Estimate at least two ARIMA candidates on the same training sample. Use residual diagnostics and rolling out-of-sample errors, not in-sample fit alone, and report prediction intervals with the assumptions that support them.

40.0.20 Chapter 20: Unit Roots, Seasonality, and Dynamic Regression

Selected guidance. Plot the level, regular difference, and seasonal difference before fitting SARIMA. Explain which transformation targets trend and which targets calendar repetition, then verify that residual seasonal autocorrelation has been reduced.

40.0.21 Chapter 21: ARCH and GARCH Models

Selected guidance. Fit or reproduce a GARCH(1,1) model on scaled returns. Calculate persistence and unconditional variance when defined, inspect standardized residuals and their squares, and document the return scale used by the optimizer.

40.0.22 Chapter 22: Asymmetric, Advanced, and Realized Volatility

Selected guidance. Compare rolling, EWMA, GARCH, and at least one OHLC volatility estimate on the same period. Explain how window choice, overnight gaps, and microstructure noise alter the interpretation of apparent precision.

40.0.23 Chapter 23: Nonlinear Models, Regimes, Market Microstructure, and Ordered Outcomes

Selected guidance. Simulate a threshold process and estimate an ordered probit example. For each, describe what changes across regimes or categories, and report predicted probabilities rather than interpreting latent-index coefficients as direct outcome changes.

40.0.24 Chapter 24: Stochastic Processes and Option Pricing

Selected guidance. Simulate geometric Brownian motion and price a European option analytically and by Monte Carlo. Report discretization and simulation error, check put-call parity, and state which Black-Scholes assumptions the exercise imposes.

40.0.25 Chapter 25: Financial Risk Management

Selected guidance. Use one loss series to calculate historical VaR and Expected Shortfall with a consistent sign convention. Backtest exceedances, compare a heavy-tailed alternative, and explain what VaR does not reveal about losses beyond the threshold.

40.0.26 Chapter 26: Multiple Time Series

Selected guidance. Fit a small VAR after checking integration order and lag selection. Distinguish predictive Granger evidence from structural causation, and state the ordering or identification assumptions behind any impulse response.

40.0.27 Chapter 27: What Artificial Intelligence and Machine Learning Actually Do

Selected guidance. Classify several examples as rule-based software, predictive machine learning, or generative AI. For each AI output, identify a verification step, a plausible source of bias, and information that should not be placed in a public prompt.

40.0.28 Chapter 28: The Machine-Learning Workflow and Gradient Descent

Selected guidance. Create a training, validation, and test split before preprocessing. Fit the transformer only on training data, compare with a naive baseline, and use the learning curve to diagnose underfitting or overfitting.

40.0.29 Chapter 29: Applied Regression and Classification Projects

Selected guidance. For the housing data, compare a baseline with a regression model using MAE and residual plots. For Titanic, use a pipeline, confusion matrix, precision, recall, and cross-validation, then discuss why predictive performance does not settle fairness questions.

40.0.30 Chapter 30: Neural Networks and Deep Learning

Selected guidance. Train the small neural network with fixed seeds and plot training and validation loss. Change one architecture or regularization choice, report the effect, and avoid claiming that one stochastic run proves superiority.

40.0.31 Chapter 31: Computer Vision

Selected guidance. Apply the supplied convolution kernel to a simple image and explain the resulting feature map. Then identify one augmentation that preserves the label and one transformation that could invalidate it.

40.0.32 Chapter 32: Natural-Language Processing and Transformers

Selected guidance. Build a TF-IDF baseline before discussing embeddings or transformers. Inspect false positives and false negatives, calculate cosine similarity carefully, and note how tokenization and domain shift can change results.

40.0.33 Chapter 33: Language Models, Retrieval, and AI Agents

Selected guidance. Construct a tiny retrieval pipeline that returns source passages before generating an answer. Record chunking and ranking choices, require citations to retrieved evidence, and define a stopping rule for the agent loop.

40.0.34 Chapter 34: Capstone Projects and Student Portfolio

Selected guidance. For each capstone, state the question, provenance, benchmark, validation design, principal figure, limitation, and reproducibility steps. A portfolio-ready result communicates what was learned without hiding failed models or uncertain evidence.

41 Appendix G. Data Sources, Licences, and Attribution

Asset	Provider/source	Status in public package	Licence/rights note
FXUSDCAD snapshot	Bank of Canada Valet API	Included frozen CSV	Official data; attribution and retrieval metadata supplied.
Tehran house prices	IntroAI repository	Included	Reused under repository CC BY 4.0; creator credited.
Titanic data	IntroAI repository	Included	Reused under repository CC BY 4.0; upstream provenance noted.
Tiny Shakespeare text	IntroAI repository	Included for optional local NLP	Archive licence retained; external source history noted.
Financial lecture PDFs	Ruey S. Tsay course notes	Not redistributed	Used as conceptual source and cited.
Source figures from PDFs	Ruey S. Tsay course notes	Not reused	All visuals recreated with Python.
GloVe subsets/images in IntroAI	Third-party assets	Excluded	Rights or redistribution conditions handled conservatively.

For the complete file-level decision record, see `audits/SOURCE_AND_LICENSE_AUDIT.md` and `audits/SOURCE_INVENTORY.csv` in the source package.

42 Appendix H. Software and Reproducibility

The tested build records exact versions in `environment.yml` and in the code-execution QA report stored in the `audits` folder. The principal stack is Python, NumPy, pandas, SciPy, Matplotlib, statsmodels, scikit-learn, PyTorch, Jupyter, Pandoc, and XeLaTeX. The separate `requirements.txt` uses compatible version ranges for students, while the environment file preserves the authoring environment.

43 Appendix I. Accessibility Statement

The publication uses high-contrast navy and teal branding, readable body and code fonts, descriptive captions, selectable text, non-colour cues, wrapped code, and print-safe figures. The HTML edition includes responsive images, heading navigation, and descriptive link text. The PDF is generated from structured source; tagging support depends on the LaTeX engine and PDF toolchain.

44 Appendix J. Acknowledgments

Ceteris LAB acknowledges Ruey S. Tsay for the financial-time-series concepts in the supplied 2013 lecture notes and Ali Sharifi-Zarchi and contributors for the CC BY 4.0 IntroAI educational materials. The new organization, explanations, Python implementations, figures, exercises, Canadian applications, audits, and publication design were prepared for Ceteris LAB by Mohammad Safavi, Ph.D.

References

- Artzner, Philippe, Freddy Delbaen, Jean-Marc Eber, and David Heath. 1999. “Coherent Measures of Risk.” *Mathematical Finance* 9 (3): 203–28. <https://doi.org/10.1111/1467-9965.00068>.
- Bank of Canada. 2026. “Valet API Documentation.” <https://www.bankofcanada.ca/valet/docs>.
- Black, Fischer, and Myron Scholes. 1973. “The Pricing of Options and Corporate Liabilities.” *Journal of Political Economy* 81 (3): 637–54. <https://doi.org/10.1086/260062>.
- Bollerslev, Tim. 1986. “Generalized Autoregressive Conditional Heteroskedasticity.” *Journal of Econometrics* 31 (3): 307–27. [https://doi.org/10.1016/0304-4076\(86\)90063-1](https://doi.org/10.1016/0304-4076(86)90063-1).
- Engle, Robert F. 1982. “Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation.” *Econometrica* 50 (4): 987–1007. <https://doi.org/10.2307/1912773>.
- Federal Reserve Bank of St. Louis. 2026. “FRED API Documentation.” <https://fred.stlouisfed.org/docs/api/fred/>.
- Garman, Mark B., and Michael J. Klass. 1980. “On the Estimation of Security Price Volatilities from Historical Data.” *Journal of Business* 53 (1): 67–78.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
- Johansen, Søren. 1988. “Statistical Analysis of Cointegration Vectors.” *Journal of Economic Dynamics and Control* 12 (2-3): 231–54. [https://doi.org/10.1016/0165-1889\(88\)90041-3](https://doi.org/10.1016/0165-1889(88)90041-3).
- Matplotlib Development Team. 2026. “Matplotlib Documentation.” <https://matplotlib.org/stable/>.
- Merton, Robert C. 1973. “Theory of Rational Option Pricing.” *Bell Journal of Economics and Management Science* 4 (1): 141–83. <https://doi.org/10.2307/3003143>.
- Nelson, Daniel B. 1991. “Conditional Heteroskedasticity in Asset Returns: A New Approach.” *Econometrica* 59 (2): 347–70. <https://doi.org/10.2307/2938260>.
- NumPy Developers. 2026. “NumPy Documentation.” <https://numpy.org/doc/>.
- pandas Development Team. 2026. “Pandas Documentation.” <https://pandas.pydata.org/docs/>.
- Python Software Foundation. 2026. “Python 3.14 Documentation.” <https://docs.python.org/3.14/>.
- PyTorch Contributors. 2026. “PyTorch Documentation.” <https://docs.pytorch.org/docs/stable/>.

scikit-learn Developers. 2026. “Scikit-Learn User Guide.”

https://scikit-learn.org/stable/user_guide.html.

Sharifi-Zarchi, Ali, and contributors. 2026. “Introduction to the World of Artificial Intelligence.” <https://github.com/SharifiZarchi/IntroAI>.

Statistics Canada. 2026. “Web Data Service.” <https://www.statcan.gc.ca/en/developers/wds>.

statsmodels Developers. 2026. “Statsmodels Documentation.”

<https://www.statsmodels.org/stable/>.

Tsay, Ruey S. 2010. *Analysis of Financial Time Series*. 3rd ed. Wiley.

———. 2013. “Analysis of Financial Time Series: Course Lecture Notes.”

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. “Attention Is All You Need.” *Advances in Neural Information Processing Systems* 30.

World Bank. 2026. “World Bank Indicators API.”

<https://datahelpdesk.worldbank.org/knowledgebase/articles/889392>.

Yang, Dennis, and Qiang Zhang. 2000. “Drift-Independent Volatility Estimation Based on High, Low, Open, and Close Prices.” *Journal of Business* 73 (3): 477–91.