

Turn data into insight with practical Python workflows

Practical Data Analysis and Visualization with Python provides a hands-on guide to modern data workflows, taking readers from raw data to actionable insights through practical, integrated techniques.

What You'll Learn

- Clean and prepare real-world datasets
- Perform effective exploratory data analysis (EDA)
- Build compelling visualizations
- Create interactive dashboards
- Work with scalable data tools (Pandas, Polars, PySpark)
- Handle large datasets with Parquet and Apache Arrow
- Use DuckDB for analytical queries

About the Author:



Dr. Shouke Wei is a researcher and scientist specializing in data analysis, wavelet-based signal processing, and AI-driven systems. He earned his Ph.D. in Germany and has held research and academic positions in Switzerland, Canada, and China.

<https://press.deepsim.ca/>



Practical Data Analysis
and Visualization

Shouke
Wei



DEEPSIM
PRESS

Practical Data Analysis and Visualization with Python

Data Exploration, Visualization, and
Scalable Data Processing

- Data Cleaning & EDA
- Visualization with Matplotlib & Seaborn
- Scalable Processing (Pandas, Polars, PySpark)
- Interactive Dashboards (Streamlit)

Shouke Wei

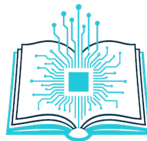
Practical Data Analysis and Visualization with Python

Data Exploration, Visualization, and Scalable Data
Processing

Practical Data Analysis and Visualization with Python

Data Exploration, Visualization, and Scalable Data
Processing

Shouke Wei



DEEPSIM
PRESS

Copyright © 2026 Shouke Wei
All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author, except in the case of brief quotations used in reviews, academic citations, or other non-commercial uses permitted by copyright law.

First Edition, 2026

For permission requests, contact the publisher:
Email: shouke.wei@deepsim.ca

ISBN 978-1-0675592-1-2 (Hardcover)
ISBN 978-1-0675592-2-9 (Paperback)
ISBN 978-1-0675592-0-5 (eBook)
DOI [10.5281/zenodo.19388650](https://doi.org/10.5281/zenodo.19388650)

Published by

Deepsim Press

An independent imprint of Deepsim Intelligence Technology Inc.
Abbotsford, British Columbia, Canada
<https://press.deepsim.ca>

This book is intended for educational and professional readers.

For resources, updates, and companion code, visit:
<https://press.deepsim.ca/data-analysis>



DEEPSIM
PRESS

About the Author

Shouke Wei, Ph.D., is a researcher, scientist, and entrepreneur specializing in intelligent IoT systems, robotics, big data analytics, modeling and forecasting, early-warning systems, and edge computing. With academic and industry experience across Europe, North America, and Asia, Dr. Wei is recognized for bridging advanced theory with real-world, production-ready systems.

Dr. Wei earned his Ph.D. in Environmental and Resource Management from the Department of Environmental Informatics at Brandenburg University of Technology Cottbus–Senftenberg (Germany). He conducted postdoctoral research at the Swiss Federal Institute of Aquatic Science and Technology (Eawag) and held research positions at the University of British Columbia (Canada). He has also served as a distinguished and adjunct professor at multiple institutions in China.

Dr. Wei currently serves as CEO and Chief Scientist of Deepsim Intelligent Technology Inc. (Canada), Chief Scientist at Canadian Sincerity Enterprises Inc., and Chief Scientist of Shandong Deepsim Intelligent Technology Co., Ltd. He is also a Postdoctoral Co-Supervisor at the Shandong Postdoctoral Innovation Practice Base and Director of Qilu Artificial Intelligence and Digital Manufacturing Innovation at Shandong Deepsim Intelligent Technology Co., Ltd., China.

Dr. Wei has led or contributed to 19 major international research projects and the development of numerous intelligent systems, including autonomous water-quality monitoring vessels, AI-based environmental early-warning platforms, medical image diagnosis systems, precision agriculture robots, and autonomous service robots.

His scholarly contributions include over 40 peer-reviewed publications, 5 books on practical wavelet transform applications, more than 500 technical tutorial articles, six patents, and over 30 software copyrights. His work focuses on making advanced computational methods—particularly data analysis, modeling, and wavelet-based signal processing—accessible, practical, and impactful for researchers and practitioners worldwide.

For more information, visit: <https://press.deepsim.ca/shouke/>

Contents

Preface	XIX
Acknowledgments	XXIII
I Foundations of Data Analysis	1
1 The Data Analysis Workflow	3
1.1 Chapter Overview	4
1.2 The Role of Data Analysis	4
1.3 Types of Data Problems	5
1.4 The Data Science Lifecycle	5
1.5 Reproducible Analysis with Python	6
1.6 Structuring Data Science Projects	6
1.7 Setting Up the Python Environment	7
1.7.1 Method 1: uv (Recommended)	7
1.7.2 Method 2: conda / Miniconda	9
1.7.3 Verifying the Installation	11
1.7.4 Basic PySpark Setup	12
1.7.5 Comparing the Two Methods	13
1.7.6 Best Practices	14
1.7.7 Summary	14
1.8 Companion Resources	14
1.9 Exercises	15
1.10 Quiz	15
2 Numerical Computing with NumPy	19
2.1 Chapter Overview	20
2.2 The Dataset: UCI Wine Quality	20
2.3 Arrays and Vectorized Operations	22
2.3.1 What is an ndarray?	22
2.3.2 Array creation	23
2.3.3 Indexing and slicing	23
2.3.4 Vectorized arithmetic	24
2.3.5 Aggregations	25

2.4	Efficient Numerical Computation	26
2.4.1	Broadcasting	26
2.4.2	Boolean masks and filtering	27
2.4.3	Performance: vectorization vs. loops	28
2.4.4	Memory layout: views vs. copies	29
2.5	Matrix Operations	30
2.5.1	The dot product	30
2.5.2	Matrix multiplication	30
2.5.3	Transpose	31
2.5.4	Solving a linear system	31
2.5.5	Eigenvalues and PCA by hand	32
2.5.6	Singular Value Decomposition (SVD)	33
2.5.7	Summary of key <code>linalg</code> functions	34
2.6	Exercises	34
2.7	Quiz	35
3	Data Analysis with Pandas	37
3.1	Chapter Overview	37
3.2	The Dataset: UCI Wine Quality	38
3.3	DataFrames and Series	40
3.3.1	What is a DataFrame?	40
3.3.2	What is a Series?	42
3.3.3	Key DataFrame attributes	42
3.4	Data Import and Export	43
3.4.1	Reading CSV files	43
3.4.2	Inspecting a freshly loaded DataFrame	44
3.4.3	Writing to common output formats	45
3.4.4	Reading other formats	45
3.5	Filtering and Transformation	46
3.5.1	Selecting columns	46
3.5.2	Label-based and position-based indexing	46
3.5.3	Boolean filtering	47
3.5.4	<code>.query()</code> syntax	48
3.5.5	Adding and transforming columns	48
3.5.6	Applying functions with <code>.apply()</code>	49
3.5.7	Renaming and dropping columns	49
3.6	Aggregation and Group Operations	50
3.6.1	Descriptive statistics on a DataFrame	50
3.6.2	<code>groupby</code> operations	51
3.6.3	Multiple aggregations with <code>.agg()</code>	52
3.6.4	Pivot tables	53
3.6.5	Value counts and frequency tables	53

3.7	Working with Time-Indexed Data	54
3.7.1	Building a DatetimeIndex	54
3.7.2	Slicing by date	55
3.7.3	Resampling	56
3.7.4	Rolling windows	56
3.7.5	Expanding windows and cumulative statistics	57
3.7.6	Shifting and lagged features	58
3.7.7	Summary of key time-series methods	58
3.8	Exercises	59
3.9	Quiz	60

II Data Preparation and Exploration 63

4 Data Cleaning and Feature Preparation 65

4.1	Chapter Overview	66
4.2	The Dataset: Titanic Passenger Records	66
4.3	Handling Missing Data	69
4.3.1	Detecting missing values	69
4.3.2	Types of missingness	70
4.3.3	Dropping rows and columns	70
4.3.4	Mean and median imputation	71
4.3.5	Forward fill and backward fill	72
4.3.6	Missing value indicators	72
4.4	Outlier Detection	73
4.4.1	What is an outlier?	73
4.4.2	The IQR method	74
4.4.3	Z-score method	74
4.4.4	Capping (Winsorisation)	75
4.4.5	Outlier summary	76
4.5	Data Transformation	76
4.5.1	Why transform features?	76
4.5.2	Min-Max scaling	76
4.5.3	Standardisation (Z-score scaling)	77
4.5.4	Robust scaling	78
4.5.5	Log transformation	78
4.5.6	Scaling method comparison	79
4.6	Encoding Categorical Variables	79
4.6.1	Why encoding is necessary	79
4.6.2	Label encoding	80
4.6.3	One-hot encoding	80
4.6.4	Ordinal encoding	81
4.6.5	Frequency encoding	82

4.6.6	Encoding method summary	82
4.7	Feature Engineering	83
4.7.1	Why engineer features?	83
4.7.2	Extracting titles from names	83
4.7.3	Family size	84
4.7.4	Binning continuous variables	85
4.7.5	Interaction features	86
4.7.6	Is the passenger travelling alone?	87
4.7.7	Fare per person	87
4.7.8	Summary of engineered features	88
4.8	Exercises	88
4.9	Quiz	89
5	Exploratory Data Analysis	91
5.1	Chapter Overview	92
5.2	The Dataset: Ames Housing	92
5.3	Load Dataset	93
5.4	Column Name Standardisation	95
5.5	Descriptive Statistics	96
5.5.1	Numeric summaries	96
5.5.2	Measures of central tendency	96
5.5.3	Measures of spread	97
5.5.4	Skewness and kurtosis	98
5.5.5	Summarising categorical variables	98
5.5.6	Descriptive statistics summary table	100
5.6	Distribution Analysis	100
5.6.1	Histograms and bin width	100
5.6.2	Kernel density estimation	101
5.6.3	Comparing distributions across groups	103
5.6.4	Quantile–Quantile plots	104
5.6.5	Normality tests	105
5.6.6	Box plots for distributional comparison	106
5.7	Correlation Analysis	108
5.7.1	Pearson correlation	108
5.7.2	Spearman rank correlation	109
5.7.3	The correlation matrix	109
5.7.4	Visualising the correlation matrix as a heatmap	110
5.7.5	Identifying multicollinearity	113
5.8	Detecting Patterns in Data	114
5.8.1	Scatter plots and linear trends	114
5.8.2	Group differences with box plots and violin plots	115
5.8.3	Cross-tabulation of categorical variables	117

5.8.4	Pair plots for multivariate patterns	119
5.8.5	Groupby-driven pattern discovery	121
5.8.6	Pivot tables for two-way pattern analysis	121
5.9	Exercises	123
5.10	Quiz	124

III Advanced Data Visualization 127

6 Data Visualization 129

6.1	Chapter Overview	130
6.2	The Datasets	130
6.2.1	Datasets from previous chapters	130
6.2.2	The new dataset: Gapminder	131
6.3	Principles of Visual Analytics	133
6.3.1	The grammar of graphics	133
6.3.2	Choosing the right chart type	134
6.3.3	Tufte’s data-ink ratio	134
6.3.4	Colour in visualization	135
6.4	Matplotlib Fundamentals	136
6.4.1	The Figure and Axes object model	136
6.4.2	Subplots and layouts	138
6.4.3	Line charts and time series	138
6.4.4	Bar charts	141
6.4.5	Scatter plots and annotations	143
6.4.6	Saving figures	143
6.5	Statistical Visualization with Seaborn	145
6.5.1	Seaborn’s design philosophy	145
6.5.2	Distribution plots	145
6.5.3	Categorical plots	146
6.5.4	Regression plots	147
6.5.5	Faceted grids with FacetGrid	148
6.5.6	Heatmaps	150
6.6	Visualizing Multivariate Data	151
6.6.1	Pair plots	151
6.6.2	Bubble charts	152
6.6.3	Parallel coordinates	156
6.6.4	Grouped bar charts and stacked bar charts	157
6.7	Interactive Visualization	159
6.7.1	Why interactivity matters	159
6.7.2	Scatter plots with hover tooltips	159
6.7.3	Interactive bar charts	160
6.7.4	Animated bubble charts	161

6.7.5	Interactive line charts	163
6.7.6	Saving interactive figures	164
6.7.7	Summary of visualization libraries	166
6.8	Exercises	166
6.9	Quiz	168
7	Advanced Plotting Libraries	171
7.1	Chapter Overview	172
7.2	The Dataset: NYC Yellow Taxi Trips — January 2023	173
7.3	Overview of Advanced Visualisation Tools	175
7.3.1	Limitations of basic plotting libraries	175
7.3.2	When to use advanced visualisation tools	175
7.3.3	Integration with the Python data ecosystem	176
7.4	Advanced Pandas Visualisation	176
7.4.1	Enhancing Pandas built-in plots	176
7.4.2	Custom styling and theming	177
7.4.3	Combining multiple plots	179
7.4.4	Handling large datasets efficiently	182
7.5	Interactive Visualisation with hvPlot	183
7.5.1	Introduction to hvPlot	183
7.5.2	hvPlot with Pandas DataFrames	185
7.5.3	hvPlot with Dask for large data	189
7.5.4	Linked and interactive plots	191
7.5.5	Customisation and styling	193
7.5.6	Performance considerations	194
7.6	Grammar of Graphics with Lets-Plot	195
7.6.1	Introduction to Lets-Plot	195
7.6.2	Core concepts: ggplot, aes, and geoms	196
7.6.3	Layers, aesthetics, and geom types	198
7.6.4	Visualising hierarchical data with Lets-Plot	200
7.6.5	Tree structures and relationships	203
7.6.6	Network graph visualisation	206
7.6.7	Faceting, theming, and export	209
7.6.8	Practical use cases	212
7.7	Categorical Visualisation with PyWaffle	213
7.7.1	Introduction to PyWaffle	213
7.7.2	Creating waffle charts	214
7.7.3	Customising layouts and colours	215
7.7.4	Comparing categories effectively	219
7.7.5	Use cases in dashboards	220
7.8	Comparing Advanced Visualisation Libraries	222
7.8.1	hvPlot vs Matplotlib vs Seaborn vs Lets-Plot	222

7.8.2	When to use each tool	223
7.8.3	Performance and scalability	224
7.9	Case Studies	226
7.9.1	Case study 1 — Interactive hourly analysis	226
7.9.2	Case study 2 — Category distribution visualisation	228
7.9.3	Case study 3 — Hierarchical data exploration with Lets-Plot	229
7.10	Exercises	232
7.11	Quiz	234
8	Data Dashboards with Streamlit and Bokeh	237
8.1	Chapter Overview	238
8.2	Introduction to Data Dashboards	239
8.2.1	What is a data dashboard?	239
8.2.2	Analytical vs operational dashboards	239
8.2.3	When dashboards are needed	240
8.2.4	From Notebook to Web App	240
8.3	Getting Started with Streamlit	241
8.3.1	Why Streamlit?	241
8.3.2	Installation and setup	241
8.3.3	The Streamlit execution model	242
8.3.4	Your first Streamlit app	243
8.3.5	Core display functions	244
8.4	Building Interactive Dashboards	245
8.4.1	Widgets	245
8.4.2	Handling user input and filtering data	247
8.4.3	Layout: columns, tabs, and expanders	249
8.5	Data Visualisation in Streamlit	250
8.5.1	Using Matplotlib and Seaborn	250
8.5.2	Using Plotly	252
8.5.3	Using hvPlot	254
8.6	Case Study — NYC Taxi Dashboard	255
8.6.1	Dataset overview and KPI design	255
8.6.2	The complete dashboard script	256
8.6.3	Key patterns in the dashboard	262
8.7	Real-Time Dashboards	264
8.7.1	What is real-time monitoring?	264
8.7.2	Timer-based refresh with <code>streamlit_autorefresh</code>	264
8.7.3	Event-driven refresh with <code>watchdog</code>	267
8.7.4	Choosing between timer and event-driven refresh	270
8.8	Deployment and Sharing	270
8.8.1	Running locally	270

8.8.2	Streamlit Community Cloud	272
8.8.3	Other deployment options	272
8.9	Other Dashboard Tools	273
8.9.1	Panel	273
8.9.2	Bokeh — Complete Dashboard Example	275
8.9.3	Dash	281
8.10	Best Practices for Dashboard Design	281
8.10.1	Focus on KPIs first	281
8.10.2	Avoid clutter	282
8.10.3	Layout hierarchy	282
8.10.4	Performance optimisation	282
8.10.5	Error handling and user feedback	283
8.11	Exercises	284
8.12	Quiz	285

IV Scalable Data Processing 289

9 High-Performance DataFrames with Polars 291

9.1	Chapter Overview	292
9.2	The Dataset: BTS On-Time Performance	292
9.3	Why Polars	296
9.3.1	The problem with Pandas at scale	296
9.3.2	What makes Polars fast	296
9.3.3	Benchmark: Pandas vs Polars on real flight data	297
9.3.4	When to use Polars	299
9.4	Polars vs Pandas	299
9.4.1	Data model differences	299
9.4.2	Side-by-side syntax comparison	300
9.4.3	Loading and inspecting the flights dataset	300
9.4.4	Filtering and column creation	302
9.4.5	Key Polars data types	304
9.5	Lazy Execution	305
9.5.1	What is a LazyFrame?	305
9.5.2	Inspecting the query plan	306
9.5.3	Optimisations applied automatically	307
9.5.4	Building a lazy pipeline	307
9.5.5	Converting between LazyFrame and DataFrame	309
9.6	Fast Aggregations and Joins	311
9.6.1	Group-by aggregations	311
9.6.2	Common aggregation expressions	312
9.6.3	Joins	313
9.6.4	Join types	315

9.6.5	Expressions inside aggregations	315
9.7	Processing Large Datasets	317
9.7.1	The memory wall	317
9.7.2	Scanning multiple CSV files at once	317
9.7.3	Streaming execution	319
9.7.4	Scanning Parquet files	320
9.7.5	Converting to and from Pandas	322
9.7.6	Large-dataset strategy summary	323
9.8	Visualization of Polars DataFrames	323
9.9	Exercises	328
9.10	Quiz	329
10	Distributed Data Processing with PySpark	333
10.1	Chapter Overview	334
10.2	The Dataset: NYC Taxi Trips	335
10.3	When Data Becomes Too Large	340
10.3.1	The single-machine ceiling	340
10.3.2	What changes with distributed computing	341
10.3.3	When to move to PySpark	341
10.4	Spark Architecture	342
10.4.1	The driver-executor model	342
10.4.2	The cluster manager	342
10.4.3	RDDs and DataFrames	344
10.4.4	Lazy evaluation in Spark	345
10.5	Spark DataFrames	346
10.5.1	Loading data into a Spark DataFrame	346
10.5.2	Inspecting the schema and shape	347
10.5.3	Selecting and filtering	349
10.5.4	Adding and transforming columns	350
10.5.5	Key PySpark DataFrame operations	352
10.5.6	Writing output	352
10.6	Pandas API on Spark	354
10.6.1	What is the Pandas API on Spark?	354
10.6.2	Creating a pandas-on-Spark DataFrame	355
10.6.3	Familiar operations on distributed data	357
10.6.4	Adding columns and applying functions	358
10.6.5	Group-by and aggregation	359
10.6.6	Joining two pandas-on-Spark DataFrames	360
10.6.7	Key differences from standard Pandas	362
10.6.8	When to use the Pandas API on Spark vs the native DataFrame API	364

10.7	Spark SQL	366
10.7.1	What is Spark SQL?	366
10.7.2	Registering a temporary view	366
10.7.3	Querying with spark.sql()	368
10.7.4	Aggregations in Spark SQL	369
10.7.5	Window functions in Spark SQL	370
10.7.6	Common table expressions (CTEs)	371
10.7.7	Spark SQL vs the DataFrame API	372
10.8	Distributed Data Processing	373
10.8.1	Partitions: the unit of parallelism	373
10.8.2	Shuffles and their cost	374
10.8.3	Group-by aggregations at scale	374
10.8.4	Joins between DataFrames	376
10.8.5	Processing multiple months at scale	378
10.8.6	Performance best practices	379
10.9	Native DataFrame plotting	381
10.10	Other Scalable Data Processing Tools	385
10.10.1	Dask for Parallel and Distributed DataFrames	386
10.10.2	Vaex for Out-of-Core Data Processing	386
10.10.3	Comparison of Scalable Data Tools	387
10.10.4	Choosing the Right Tool	388
10.11	Summary	389
10.12	Exercises	389
10.13	Quiz	391
11	Modern Data Storage and Query Engines	395
11.1	Chapter Overview	396
11.2	The Dataset: NYC Yellow Taxi Trips 2023	397
11.3	CSV Limitations	399
11.3.1	What CSV was designed for	399
11.3.2	The read performance problem	400
11.3.3	The type safety problem	401
11.3.4	CSV limitations summary	402
11.4	Parquet and Columnar Storage	402
11.4.1	Row-oriented vs columnar storage	402
11.4.2	Parquet internals	403
11.4.3	Reading and writing Parquet with Pandas and Polars	405
11.4.4	Parquet compression codecs	406
11.4.5	Column-selective reads and predicate pushdown	407
11.5	Apache Arrow	408
11.5.1	What is Apache Arrow?	408
11.5.2	The Arrow ecosystem	409

11.5.3	Working with PyArrow directly	409
11.5.4	Zero-copy conversion between libraries	410
11.5.5	Building Arrow arrays and tables directly	412
11.5.6	Key PyArrow operations	413
11.6	Partitioned Data	413
11.6.1	What is dataset partitioning?	413
11.6.2	Writing a partitioned Parquet dataset	414
11.6.3	Reading a partitioned dataset	416
11.6.4	Choosing partition columns	417
11.6.5	Verifying partition pruning	418
11.7	Analytical Query Engines: DuckDB	419
11.7.1	What is DuckDB?	419
11.7.2	Installing and starting DuckDB	419
11.7.3	Querying Parquet files directly	420
11.7.4	Performance comparison: DuckDB vs Pandas vs Polars	421
11.7.5	Group-by aggregations	423
11.7.6	Window functions	424
11.7.7	Joining Parquet files with a lookup table	425
11.7.8	Persisting results to Parquet	427
11.7.9	Persisting the full year to a DuckDB database file	428
11.7.10	Tool selection summary	433
11.8	Exercises	433
11.9	Quiz	435

V End-to-End Data Science Workflows 437

12 Integrating Analysis and Visualisation 439

12.1	Chapter Overview	440
12.2	Putting EDA and Visualisation Together	441
12.2.1	The five-stage analytical pipeline	441
12.2.2	How visualisation informs each stage	442
12.2.3	Common anti-patterns	443
12.3	Building Reusable Pipelines	444
12.3.1	Structuring analysis as functions	444
12.3.2	Using configuration objects	453
12.3.3	Example usage with the retail configuration:	455
12.3.4	Parameterised visualisation functions	455
12.3.5	Pipeline orchestration	456
12.4	Case Study 1 — Business: Retail Sales Analytics	457
12.4.1	Dataset overview	457
12.4.2	Stage 1 — Ingestion and audit	458
12.4.3	Stage 2 — Cleaning	459

12.4.4	Stage 3 — Feature engineering and EDA	460
12.4.5	Stage 4 — Monthly revenue and RFM analysis	464
12.4.6	Stage 5 — Communication: Streamlit retail dashboard	469
12.5	Case Study 2 — Finance: Equity Price Analysis	472
12.5.1	Dataset overview	472
12.5.2	Stage 1 — Download and validate	472
12.5.3	Stage 2 — Feature engineering: returns and rolling statistics	473
12.5.4	Stage 3 — EDA: price chart with rolling averages	474
12.5.5	Stage 3 — EDA: return distributions and correlation	475
12.5.6	Stage 4 — Aggregation: risk–return summary	477
12.5.7	Stage 5 — Communication: interactive finance dashboard	480
12.6	Case Study 3 — IoT: Sensor Stream Monitoring	482
12.6.1	Dataset overview	482
12.6.2	Stage 1 — Ingestion and audit	484
12.6.3	Stage 2 — Cleaning and datetime parsing	486
12.6.4	Stage 3 — EDA: temporal patterns	487
12.6.5	Stage 4 — Anomaly detection with rolling z-scores	489
12.6.6	Stage 5 — Communication: real-time IoT dashboard	491
12.7	Exercises	496
12.8	Quiz	499

References **503**

Index **511**

Preface

Data analysis has become a foundational skill across science, engineering, business, and technology. In an era where data is continuously generated at unprecedented scale, the ability to transform raw data into meaningful insights is no longer optional—it is essential.

This book, *Practical Data Analysis and Visualization with Python*, is designed to provide a clear, structured, and hands-on introduction to modern data analysis workflows. Rather than focusing on isolated tools or abstract theory, the book emphasizes how different components—data preparation, exploration, visualization, and scalable processing—fit together into a coherent and practical pipeline.

Why This Book

Many existing resources on data analysis fall into one of two categories: highly theoretical treatments that are difficult to apply, or tool-specific tutorials that lack a unifying perspective. This book aims to bridge that gap.

The goal is to help readers:

- Understand the **full data analysis workflow**, from raw data to insight
- Develop strong intuition for **data exploration and visualization**
- Learn how to work efficiently with **modern Python tools**
- Scale workflows from small datasets to **large and distributed systems**
- Build reproducible and structured data analysis projects

The focus is practical. Every concept is grounded in real-world usage, with an emphasis on clarity, usability, and performance.

What This Book Covers

The book is organized into six parts:

- **Part I — Foundations of Data Analysis**
Introduces the data analysis workflow, project structure, and the Python environment, along with core tools such as NumPy and Pandas.
- **Part II — Data Preparation and Exploration**
Covers data cleaning, transformation, and exploratory data analysis (EDA), forming the backbone of any data-driven project.
- **Part III — Advanced Data Visualization**
Explores the principles of visual analytics alongside practical visualization techniques. Covers Matplotlib and Seaborn, as well as modern interactive libraries such as hvPlot and Lets-Plot. Also includes categorical visualization with PyWaffle and the development of modern, interactive dashboards.
- **Part IV — Scalable Data Processing**
Introduces high-performance tools for large-scale data processing, including Polars and PySpark. Examines efficient data formats such as Parquet and columnar storage, and explores technologies like Apache Arrow, partitioned data workflows, and DuckDB, along with an overview of additional scalable frameworks.
- **Part V — End-to-End Data Science Workflows**
Demonstrates how to integrate analysis and visualization into complete, reusable workflows through practical case studies.

What This Book Does Not Cover

This book does not aim to provide an in-depth treatment of statistical modeling, time series modeling, or machine learning. While these topics are an essential component of the data science workflow, including them here would dilute the focus on data analysis and significantly increase the size of the book.

Instead, they are addressed in a companion volume, where they can be explored in greater depth, including modern machine learning techniques and time series modeling.

Who This Book Is For

This book is intended for:

- Students and practitioners beginning their journey in data analysis
- Developers who want to incorporate data-driven workflows into their work
- Analysts seeking to modernize their Python-based data pipelines
- Anyone interested in building practical, scalable data analysis systems

A basic familiarity with Python is recommended, but the book is structured to guide readers progressively from foundational concepts to more advanced topics.

How to Use This Book

The chapters are designed to be read sequentially, especially for readers new to data analysis. However, more experienced readers may choose to focus on specific parts, such as visualization or scalable data processing.

Each chapter includes:

- Clear explanations of key concepts
- Practical examples using real-world workflows
- Exercises to reinforce understanding
- A concise quiz to test comprehension

Readers are encouraged to experiment with the code, adapt examples to their own datasets, and build projects along the way.

A Practical Perspective

Throughout the book, the emphasis is on making decisions:

- Which tool should be used in a given situation?
- How should a dataset be structured and processed?
- When should workflows be optimized or scaled?

By focusing on these practical questions, the book aims to develop not just technical skills, but also sound judgment in data analysis.

Looking Ahead

The workflows and tools introduced in this book provide the foundation for more advanced topics, including statistical modeling, machine learning, and time series analysis. These are explored in the next volume, where the focus shifts from understanding data to building predictive and analytical models.

Data analysis is not just about tools—it is about thinking clearly, asking the right questions, and communicating insights effectively. This book is an invitation to develop those skills in a practical and structured way.

Shouke Wei, PhD

Deepsim Intelligent Technology Inc.

Deepsim Academy

Abbotsford, Canada

March 28, 2026

Acknowledgments

This book reflects a continuous journey of teaching, learning, and refining ideas through academic work and practical applications. It would not have been possible without the experience gained from developing and delivering a series of data-focused courses.

In particular, the course *Master Python Data Analysis and Modelling Essentials*—offered on Udemy and through Deepsim Academy—has played an important role in shaping this book. Through it, I engaged with learners from diverse backgrounds, whose questions, feedback, and real-world challenges helped clarify what truly matters in practical data analysis.

I am grateful to all students and participants who contributed, directly or indirectly, by sharing their perspectives and encouraging clearer explanations and more effective approaches. Their engagement has significantly influenced the examples, workflows, and practical focus of this book.

I also acknowledge the broader open-source community. The tools presented in this book—across data analysis, visualization, and scalable computing—reflect the collective efforts of countless developers and contributors, making modern data analysis more accessible, powerful, and efficient.

Finally, I would like to express my deepest gratitude to my family—my parents for their support and encouragement, and my wife for her patience and unwavering support throughout the writing of this book. Their support has been invaluable.

To all readers, thank you for being part of this journey.

Part I

Foundations of Data Analysis

1 The Data Analysis Workflow

The explosion of digital data over the past two decades has fundamentally transformed how organizations make decisions, conduct research, and develop products. From healthcare and finance to logistics and social science, the ability to extract actionable knowledge from data has become one of the most valued skills of the modern era (Manyika et al. 2011; Provost and Fawcett 2013).

Data analysis sits at the heart of this transformation. It is the discipline concerned with inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, drawing conclusions, and supporting decision-making (Tukey et al. 1977). Unlike traditional statistics, which often begins with a hypothesis, modern data analysis is exploratory, iterative, and deeply intertwined with computation.

The rise of open-source tools - particularly the Python ecosystem - has democratized data analysis, making sophisticated techniques accessible to practitioners across disciplines (VanderPlas 2016). Libraries such as NumPy, Pandas, and scikit-learn have become the lingua franca of data science, while environments like JupyterLab provide an interactive, reproducible workspace that bridges code, narrative, and visualization (Kluyver et al. 2016).

Yet, raw technical skill alone is insufficient. Effective data analysis requires a principled workflow: a repeatable sequence of steps from problem formulation through data collection, cleaning, exploration, modeling, and communication. Without such a workflow, analyses become brittle, results become hard to reproduce, and insights fail to translate into action (Wilson et al. 2017).

This chapter lays the conceptual and practical groundwork for the rest of the book. We introduce the core vocabulary of data analysis, survey the types of

problems data analysts encounter, and walk through the standard data science lifecycle. We also establish a robust Python environment that will be used throughout all subsequent chapters.

1.1 Chapter Overview

This chapter covers the following topics:

- **The Role of Data Analysis**— What data analysis is and why it matters in modern decision-making
- **Types of Data Problems** — A taxonomy of analytical problem types: descriptive, diagnostic, predictive, and prescriptive
- **The Data Science Lifecycle** — The iterative, end-to-end workflow from problem definition to deployment
- **Reproducible Analysis with Python** — Principles and practices for making your work reproducible and trustworthy
- **Structuring Data Science Projects** — How to organize files and folders for maintainability and collaboration
- **Setting Up the Python Environment** — Step-by-step instructions for configuring a Python environment with all required libraries
- **Exercises & Quiz** — Hands-on exercises and a short quiz to reinforce key concepts

By the end of this chapter, you will be able to:

- Explain the role and value of data analysis in real-world contexts.
- Distinguish between the four major types of data problems.
- Describe the stages of the data science lifecycle.
- Apply best practices for reproducibility and project organization.
- Set up a fully functional Python data analysis environment using either `uv` or `conda`.

1.2 The Role of Data Analysis

Data analysis is the process of transforming raw data into meaningful insights, actionable knowledge, and informed decisions. It supports decision-making,

helps understand complex systems, and enables predictive modeling (Provost and Fawcett 2013).

Data analysis connects data to real-world actions by uncovering patterns, trends, and relationships.

1.3 Types of Data Problems

Data problems generally fall into several categories:

- **Descriptive Analysis:** Understanding what happened
- **Diagnostic Analysis:** Understanding why it happened
- **Predictive Analysis:** Forecasting what will happen
- **Prescriptive Analysis:** Recommending actions

Each type requires different tools and methods, but all rely on clean, well-structured data.

1.4 The Data Science Lifecycle

A typical data science workflow includes:

1. Problem Definition
2. Data Collection
3. Data Cleaning
4. Exploratory Data Analysis
5. Modeling
6. Evaluation
7. Deployment

This lifecycle is iterative and often requires revisiting earlier steps (Wilson et al. 2017).

1.5 Reproducible Analysis with Python

Reproducibility ensures that results can be recreated. Key practices include:

- Using notebooks (Jupyter) (Kluyver et al. 2016)
- Version control (Git)
- Fixed environments (requirements.txt or uv.lock)
- Clear documentation

Reproducibility is essential for collaboration and reliability.

1.6 Structuring Data Science Projects

A well-organized project improves productivity and maintainability.

Typical structure:

```
data-analysis/  
  .venv/  
  data/  
  notebooks/  
  src/  
  models/  
  outputs/  
  requirements.txt  
  README.md
```

Best practices:

- Separate data and code
- Use modular scripts
- Track experiments
- Document workflows

1.7 Setting Up the Python Environment

A well-configured Python environment is essential for efficient and reproducible data analysis. In this section, we set up a practical environment that supports the tools used throughout this book, including NumPy, Pandas, Polars, PySpark, and visualization libraries.

The goal is not to cover every possible configuration, but to provide a simple, reliable setup that works for most users. We present two recommended methods for setting up a Python environment: the modern `uv` tool (recommended for new projects due to its speed and simplicity) and the traditional `conda` / `miniconda` approach, which is widely used in scientific computing and data science.

You can choose either method based on your preference and workflow.

1.7.1 Method 1: `uv` (Recommended)

`uv` is a blazing-fast Python package and project manager written in Rust, developed by [Astral](#). It is designed as a drop-in replacement for `pip`, `pip-tools`, `virtualenv`, `pyenv`, and more - all in a single tool. For new projects and learners setting up fresh environments, `uv` is the recommended approach in this book.

Why `uv`?

- 10–100× faster than `pip` for dependency resolution and installation
- Built-in virtual environment management (no need for separate `venv` or `conda`)
- Automatic Python version management (replaces `pyenv`)
- Lockfile support for fully reproducible environments (`uv.lock`)
- Works seamlessly with standard `pyproject.toml` and `requirements.txt`

1.7.1.1 Installing `uv`

macOS / Linux:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Windows (PowerShell):

```
powershell -ExecutionPolicy ByPass -c "irm  
↪ https://astral.sh/uv/install.ps1 | iex"
```

Verify the installation:

```
uv --version
```

1. Create a Virtual Environment Using `uv`, it is straightforward to create a virtual environment and install a specific Python version within it. This book uses Python 3.12, which provides a strong balance between modern features and broad library compatibility.

```
uv venv --python 3.12
```

This command creates an isolated environment for the project, ensuring that dependencies are managed consistently and do not interfere with other Python installations.

2. Activate the environment

macOS / Linux:

```
source .venv/bin/activate
```

Windows:

```
.venv\Scripts\activate
```

1.7.1.2 Installing Core Libraries

To ensure a consistent environment, download the `requirements.txt` file (see Section 1.8) and install all required dependencies.

```
uv pip install -r requirements.txt
```

Or simply install the dependencies manually as needed:

```
uv pip install numpy matplotlib seaborn
```

1.7.1.3 Managing Dependencies with uv

Generate a pinned requirements file at the end of your project to ensure reproducibility:

```
uv pip freeze > requirements.txt
```

For full project management with lockfiles (recommended for team projects):

```
# Initialize a project
uv init my-project
cd my-project

# Add dependencies
uv add numpy pandas polars jupyterlab

# This creates uv.lock for fully reproducible installs
uv sync
```

1.7.2 Method 2: conda / Miniconda

The `conda` ecosystem is a mature, widely adopted approach in data science and is particularly useful when working with packages that have complex non-Python dependencies (e.g., CUDA libraries, R packages).

1.7.2.1 Choosing a Python Distribution

There are several ways to install Python via conda:

- **Official Python** (python.org) - lightweight, uses pip only
- **Anaconda** - full distribution with many preinstalled packages
- **Miniconda** - minimal version of Anaconda (recommended if using conda)

Recommendation: Use Miniconda if you prefer the conda ecosystem, as it is lightweight, flexible, and widely used in data science.

1.7.2.2 Installing Miniconda

1. Download Miniconda from the [official website](#)
2. Install it using default settings
3. Verify installation:

```
conda --version
```

1.7.2.3 Creating a Virtual Environment

Using a dedicated environment ensures that your project dependencies are isolated and reproducible.

Create a new environment:

```
conda create -n data-env python=3.12
```

Activate the environment:

```
conda activate data-env
```

1.7.2.4 Installing Core Libraries

Readers are encouraged to download the `requirements.txt` file (see Section 1.8) and install the required libraries to reproduce the examples in this book.

```
pip install -r requirements.txt
```

To install individual packages manually, you can use either `conda` or `pip`:

Using conda:

```
conda install numpy matplotlib seaborn
```

Using pip:

```
pip install numpy matplotlib seaborn
```

1.7.2.5 Managing Dependencies

Using pip:

```
pip freeze > requirements.txt
```

Using conda:

```
conda env export > environment.yml
```

1.7.3 Verifying the Installation

1.7.3.1 Running JupyterLab

JupyterLab is an interactive environment for writing and running Python code.

Start Jupyter Lab:

```
jupyter lab
```

1.7.3.2 Verifying the Installation

Create a new notebook and run:

```
import numpy as np
import pandas as pd
import polars as pl

print("Environment ready!")
```

1.7.4 Basic PySpark Setup

1.7.4.1 Windows-specific: Installing winutils

PySpark on Windows requires a small Hadoop utility called `winutils.exe` to write files to disk. Without it, any `.write.parquet()` or `.write.csv()` call raises a `HADOOP_HOME is unset` error.

1. Download winutils

Download `winutils.exe` and `hadoop.dll` from: <https://github.com/kontext-tech/winutils/tree/master/hadoop-3.4.0-win10-x64/bin>

Use the `hadoop-3.4/bin/` folder. Place both files at:

`C:\hadoop\bin\winutils.exe`

`C:\hadoop\bin\hadoop.dll`

i Note

Note: If the error persists, download the entire bin folder and replace the contents of `C:\hadoop\bin\` with it.

It also needs to upgrade your JDK to version 17 or 21. Older JDK versions can cause compatibility errors with PySpark that are unrelated to Hadoop configuration.

2. Pin Pandas to a compatible version

When configuring the development environment, it is important to ensure compatibility between core libraries. At the time of writing, the `pandas-on-Spark` module in PySpark 4.x (`pyspark.pandas`) may not be fully compatible with Pandas 3.x.

As the Python ecosystem evolves, this limitation may be resolved in future releases. Readers are encouraged to check the official PySpark documentation for the latest compatibility information.

To ensure a stable experience when following the examples in this book, it is recommended to use a Pandas 2.x release (e.g., `pandas>=2.0,<3.0`) unless newer versions are explicitly verified to work.

```
pip install "pandas==2.2.3"
```

macOS and Linux users can skip both steps above.

1.7.4.2 Test PySpark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("test").getOrCreate()
spark.range(5).show()
```

1.7.5 Comparing the Two Methods

Table 1.1: Comparing the two Python environment management methods

Feature	uv (Recommended)	conda / Miniconda
Speed	Extremely fast (Rust-based)	Moderate
Python version management	Built-in	Requires separate install

Feature	<code>uv</code> (Recommended)	<code>conda</code> / Miniconda
Non-Python dependencies	Limited	Excellent (e.g., CUDA)
Lockfile support	Yes (<code>uv.lock</code>)	Partial (<code>environment.yml</code>)
Standard compatibility	<code>pip/pyproject.toml</code>	Conda ecosystem
Best for	New projects, learners	Legacy projects, HPC, scientific computing

1.7.6 Best Practices

- Use a separate environment for each project
- Avoid installing packages globally
- Keep dependencies minimal
- Document your setup
- Use version control (e.g., Git)
- Commit your lockfile (`uv.lock` or `environment.yml`) for reproducibility

1.7.7 Summary

Setting up a proper Python environment is the first step toward effective data analysis. Whether you choose `uv` for its speed and modern design or `conda` for its mature ecosystem, the key is consistency: always use a virtual environment, pin your dependencies, and verify your setup before beginning any project. This creates a stable foundation for all subsequent work in this book.

1.8 Companion Resources

All datasets, notebooks, scripts, and `requirements.txt` files for this book are available at press.deepsim.ca/data-analysis.

Updates and supplementary material will be posted there as well.

1.9 Exercises

Exercise 1:

Explain the importance of data analysis in decision-making.

Exercise 2:

Classify a business problem (e.g., sales forecasting) into one of the data problem types.

Exercise 3:

List and explain the steps in the data science lifecycle.

Exercise 4:

Describe how you would ensure reproducibility in a data project.

Exercise 5:

Design a folder structure for a machine learning project.

Exercise 6:

Set up a Python environment using the `uv` method. Create a virtual environment, install the core libraries, and verify the installation by importing NumPy, Pandas, and Polars in a Jupyter notebook.

Exercise 7:

Compare the `uv` and `conda` methods. In what scenarios would you prefer one over the other? Write a short paragraph justifying your choice for a hypothetical team project.

1.10 Quiz

1. What is the main goal of data analysis?
 - A. Store data
 - B. Transform data into insights
 - C. Delete data
 - D. Only visualize data

2. Predictive analysis is used to:
 - A. Describe past events
 - B. Explain causes
 - C. Forecast future outcomes
 - D. Clean data
3. Which step comes first in the data science lifecycle?
 - A. Modeling
 - B. Data cleaning
 - C. Problem definition
 - D. Evaluation
4. What ensures reproducibility?
 - A. Random processes
 - B. Version control and documentation
 - C. Ignoring data
 - D. Only visualization
5. A good project structure helps:
 - A. Increase confusion
 - B. Improve organization and reproducibility
 - C. Reduce data
 - D. Avoid coding
6. Which tool is described as the recommended environment manager in this book?
 - A. pip
 - B. Anaconda
 - C. uv
 - D. Docker
7. What is the main advantage of uv over pip?
 - A. It only works on Windows
 - B. It is significantly faster and includes virtual environment management
 - C. It requires no internet connection
 - D. It installs R packages

Answers:

1. B
2. C
3. C
4. B

5. B
6. C
7. B

2 Numerical Computing with NumPy

At the heart of nearly every data science workflow in Python lies a single foundational library: NumPy. Short for *Numerical Python*, NumPy provides the `ndarray` — an n-dimensional array object — along with a comprehensive suite of mathematical functions that operate on these arrays with remarkable efficiency (Harris et al. 2020). Without NumPy, libraries such as Pandas, scikit-learn, SciPy, and even TensorFlow would not exist in their current form; they all build directly on NumPy’s array primitives.

The case for NumPy rests on a simple but profound insight: Python’s built-in lists are flexible but slow for numerical work, because each element is a full Python object with its own type information and memory overhead. NumPy arrays, by contrast, store homogeneous data in contiguous blocks of memory, enabling operations to be dispatched to pre-compiled C and Fortran routines (VanderPlas 2016). The result is code that is often 10 to 100 times faster than equivalent pure Python, without sacrificing readability.

Beyond raw speed, NumPy introduces *vectorization* — the ability to express element-wise operations over entire arrays without explicit loops. This paradigm, borrowed from languages like MATLAB and R, makes numerical code more concise, more expressive, and easier to reason about (McKinney 2022). Combined with *broadcasting* — NumPy’s rules for performing operations between arrays of different shapes — vectorization becomes a powerful tool for expressing complex numerical transformations in just a few lines.

This chapter uses the **UCI Wine Quality dataset** (Cortez et al. 2009), hosted on GitHub, as a running example throughout. The dataset contains physicochemical measurements of Portuguese red wines — including alcohol content, acidity, residual sugar, and pH — alongside a sensory quality rating.

It is a clean, real-world numerical dataset that maps naturally onto the array operations introduced here.

2.1 Chapter Overview

This chapter covers the following topics:

- **Arrays and Vectorized Operations** — How to create, inspect, index, and slice NumPy arrays, and how to replace loops with fast vectorized expressions
- **Efficient Numerical Computation** — Aggregations, broadcasting, masking, and performance considerations for large arrays
- **Matrix Operations** — Linear algebra with NumPy: dot products, matrix multiplication, decompositions, and a practical application to the wine dataset

By the end of this chapter, you will be able to:

- Load a real-world CSV dataset into a NumPy array and inspect its structure.
- Perform vectorized arithmetic, comparisons, and aggregations over arrays.
- Apply broadcasting to operate on arrays of different shapes.
- Execute matrix operations including dot products, transposes, and solving linear systems.

2.2 The Dataset: UCI Wine Quality

Throughout this chapter we use the **Wine Quality** dataset from the UCI Machine Learning Repository (Cortez et al. 2009). The dataset is publicly available on GitHub and contains physicochemical test results for 1,599 red wine samples from the *Vinho Verde* region of Portugal.

Dataset URL:

```
https://raw.githubusercontent.com/mlflow/mlflow/master/tests/datasets/winequality-red.csv
```

Each row represents one wine sample. The 12 columns are:

Table 2.1: Wine Quality dataset columns

Column	Description	Unit
fixed acidity	Non-volatile acids	g/dm ³
volatile acidity	Acetic acid (vinegar effect)	g/dm ³
citric acid	Freshness and flavour	g/dm ³
residual sugar	Sugar after fermentation	g/dm ³
chlorides	Salt content	g/dm ³
free sulfur dioxide	Free SO ₂ (antimicrobial)	mg/dm ³
total sulfur dioxide	Total SO ₂	mg/dm ³
density	Density of wine	g/cm ³
pH	Acidity scale	—
sulphates	Antimicrobial additive	g/dm ³
alcohol	Alcohol percentage	% vol
quality	Sensory score (0–10)	—

1. Load a Dataset Directly from a GitHub Repository

If the dataset is accessible via a raw URL, you can load it directly using NumPy.

If the dataset is accessible via a raw URL, you can load it directly using pandas.

```
import numpy as np

url = (
    "https://raw.githubusercontent.com/mlflow/mlflow/"
    "master/tests/datasets/winequality-red.csv"
)

# Load CSV - skip the header row, use semicolon delimiter
data = np.genfromtxt(url, delimiter=";", skip_header=1)

print(data.shape)    # (1599, 12)
print(data.dtype)    # float64
```

i Note

When loading data from GitHub, always use the raw file URL (e.g., from `raw.githubusercontent.com`), not the repository page link.

This approach works for standard text-based formats such as CSV and TSV. For other formats (e.g., JSON), use appropriate loaders like `pandas.read_json()`.

2. Load a Dataset Locally

If the dataset is already downloaded to your machine:

```
import numpy as np

path = "../data/winequality-red.csv"

# Load CSV - skip the header row, use semicolon delimiter
data = np.genfromtxt(path, delimiter=";", skip_header=1)

print(data.shape)    # (1599, 12)
print(data.dtype)    # float64
```

For the rest of this chapter, each section extracts specific columns from `data` to demonstrate a concept using real values.

2.3 Arrays and Vectorized Operations

2.3.1 What is an ndarray?

The core object in NumPy is the `ndarray` — an n-dimensional, homogeneous array stored in contiguous memory. Unlike a Python list, every element in an `ndarray` shares the same data type (`dtype`), which allows NumPy to delegate computation to optimised compiled code (Harris et al. 2020).

```
import numpy as np

# A simple 1-D array
arr = np.array([7.4, 7.8, 7.8, 11.2, 7.4])
```

```
print(arr.dtype)    # float64
print(arr.shape)   # (5,)
print(arr.ndim)    # 1
print(arr.size)    # 5
```

A 2-D array (matrix) is created from a list of lists:

```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])
print(matrix.shape) # (2, 3) - 2 rows, 3 columns
```

2.3.2 Array creation

NumPy provides several factory functions for creating arrays without manually specifying every value:

```
np.zeros((3, 4))      # 3x4 array of zeros
np.ones((2, 5))      # 2x5 array of ones
np.eye(4)            # 4x4 identity matrix\index{identity matrix}
np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
np.linspace(0, 1, 5) # [0.0, 0.25, 0.5, 0.75, 1.0]
np.random.seed(42)
np.random.randn(3, 3) # 3x3 standard normal random array
```

2.3.3 Indexing and slicing

Array indexing follows the same syntax as Python lists, but extends to multiple dimensions using comma notation:

```
# Extract the alcohol column (index 10) from the wine dataset
alcohol = data[:, 10]      # all rows, column 10
print(alcohol[:5])        # first 5 values: [9.4 9.8 9.8 9.8 9.4]

# Extract the first 5 rows and first 3 columns
subset = data[:5, :3]
print(subset)
```

Output:

```
[9.4 9.8 9.8 9.8 9.4]
[[ 7.4  0.7  0. ]
 [ 7.8  0.88 0. ]
 [ 7.8  0.76 0.04]
 [11.2  0.28 0.56]
 [ 7.4  0.7  0. ]]
```

Boolean indexing filters rows by a condition:

```
# Select wines with alcohol content above 12%
high_alcohol = data[data[:, 10] > 12]
print(high_alcohol.shape) # (141, 12)
```

Fancy indexing uses an array of indices to select non-contiguous columns:

```
# Extract alcohol (10), pH (8), and quality (11) columns
cols = [10, 8, 11]
selected = data[:, cols]
print(selected.shape) # (1599, 3)
```

2.3.4 Vectorized arithmetic

The defining feature of NumPy is that arithmetic operators apply element-wise across entire arrays, with no explicit loops required:

```
alcohol = data[:, 10] # alcohol % vol
density = data[:, 7] # density g/cm3

# Compute alcohol-to-density ratio for every wine sample at once
ratio = alcohol / density
print(ratio[:5])
```

Output:

```
[9.4207256  9.83146067 9.82948847 9.81963928 9.4207256 ]
```

This is dramatically faster than a Python loop for 1,599 samples. The same applies to all standard operators (+, -, *, /, **) and NumPy's universal functions (np.sqrt, np.log, np.exp, np.abs, etc.):

```
# Log-transform residual sugar (column 3) to reduce skewness
sugar = data[:, 3]
log_sugar = np.log1p(sugar) # log(1 + x) - safe for zero values

# Standardise alcohol: z = (x - mean) /
  ↪ std\index{standardisation}\index{z-score}
alcohol_z = (alcohol - alcohol.mean()) / alcohol.std()
print(alcohol_z[:5])
```

Output:

```
[-0.96024611 -0.58477711 -0.58477711 -0.58477711 -0.96024611]
```

2.3.5 Aggregations

NumPy provides fast aggregation functions that summarise array data:

```
quality = data[:, 11]

print(f"Mean quality: {quality.mean():.2f}")
print(f"Std quality: {quality.std():.2f}")
print(f"Min quality: {quality.min():.0f}")
print(f"Max quality: {quality.max():.0f}")
print(f"Median quality: {np.median(quality):.1f}")
```

Output:

```
Mean quality: 5.64
Std quality: 0.81
Min quality: 3
Max quality: 8
Median quality: 6.0
```

References

- Aggarwal, Charu C. 2017. *Outlier Analysis*. 2nd ed. Springer International Publishing. <https://doi.org/10.1007/978-3-319-47578-3>.
- Apache Software Foundation. 2024. *Apache Arrow: A Cross-Language Development Platform for in-Memory Analytics*. <https://arrow.apache.org>.
- Bureau of Transportation Statistics. 2024. *Airline on-Time Performance Data*. <https://transtats.bts.gov>.
- Buuren, Stef van. 2018. *Flexible Imputation of Missing Data*. 2nd ed. Chapman & Hall/CRC Interdisciplinary Statistics. CRC Press. <https://stefvanbuuren.name/fimd/>.
- Chambers, John M., William S. Cleveland, Beat Kleiner, and Paul A. Tukey. 1983. *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole.
- Chen, Daqing. 2012. *Online Retail II Dataset*. UCI Machine Learning Repository. <https://archive.ics.uci.edu/dataset/502>.
- Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. “Modeling Wine Preferences by Data Mining from Physicochemical Properties.” *Decision Support Systems* 47 (4): 547–53. <https://doi.org/10.1016/j.dss.2009.05.016>.
- Damji, Jules S., Brooke Wenig, Tathagata Das, and Denny Lee. 2020. *Learning Spark: Lightning-Fast Data Analytics*. 2nd ed. O’Reilly Media.
- De Cock, Dean. 2011. “Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project.” *Journal of Statistics Education*

19 (3): 1–14. <https://doi.org/10.1080/10691898.2011.11889627>.

De Vito, Saverio et al. 2008. *Air Quality Dataset*. UCI Machine Learning Repository. <https://archive.ics.uci.edu/dataset/360>.

DeCarlo, Lawrence T. 1997. “On the Meaning and Use of Kurtosis.” *Psychological Methods* 2 (3): 292–307. <https://doi.org/10.1037/1082-989X.2.3.292>.

Dremio. 2023. *Apache Parquet: The Definitive Guide*. <https://www.dremio.com/resources/guides/apache-parquet-the-definitive-guide/>.

Eaton, John P., and Charles A. Haas. 1994. *Titanic: Triumph and Tragedy*. Patrick Stephens Ltd.

Few, Stephen. 2006. *Information Dashboard Design: The Effective Visual Communication of Data*. O’Reilly Media.

Few, Stephen. 2012. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. 2nd ed. Analytics Press.

Gapminder Foundation. 2023. *Gapminder Tools*. <https://www.gapminder.org/tools/>.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. “Array Programming with NumPy.” *Nature* 585: 357–62. <https://doi.org/10.1038/s41586-020-2649-2>.

HoloViz Team. 2024. *hvPlot: A High-Level Plotting API for the PyData Ecosystem*. <https://hvplot.holoviz.org>.

Hunter, John D. 2007. “Matplotlib: A 2D Graphics Environment.” *Computing in Science & Engineering* 9 (3): 90–95. <https://doi.org/10.1109/MCSE.2007.55>.

Iglewicz, Boris, and David C. Hoaglin. 1993. *How to Detect and Handle*

- Outliers*. Vol. 16. The ASQC Basic References in Quality Control: Statistical Techniques. ASQC Quality Press.
- JetBrains. 2024a. “Allowed Data Types in Lets-Plot.” JetBrains. https://lets-plot.org/examples/cookbook/allowed_data_types.html.
- JetBrains. 2024b. *Lets-Plot: A Multiplatform Plotting Library Based on the Grammar of Graphics*. <https://lets-plot.org>.
- Kleppmann, Martin. 2017. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O’Reilly Media.
- Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, et al. 2016. “Jupyter Notebooks—a Publishing Format for Reproducible Computational Workflows.” *Positioning and Power in Academic Publishing: Players, Agents and Agendas: Proceedings of the 20th International Conference on Electronic Publishing*, 87.
- Kuhn, Max, and Kjell Johnson. 2019. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. Chapman & Hall/CRC Data Science Series. CRC Press. <http://www.feats-engineering/>.
- Li, Guangyang. 2024. *PyWaffle: Make Waffle Charts in Python*. <https://py-waffle.readthedocs.io>.
- Manyika, James, Michael Chui, Brad Brown, et al. 2011. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*.
- McKinney, Wes. 2022. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyter*. 3rd ed. O’Reilly Media.
- Meng, Xinrong, and Ruifeng Zheng. 2025. “PySpark Native Plotting.” Databricks, June. <https://www.databricks.com/blog/pyspark-native-plotting>.
- New York City Taxi and Limousine Commission. 2024. *TLC Trip Record Data*. <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>.

- Pearson, Karl. 1895. “Note on Regression and Inheritance in the Case of Two Parents.” *Proceedings of the Royal Society of London* 58 (347-352): 240–42. <https://doi.org/10.1098/rspl.1895.0041>.
- Plotly Technologies Inc. 2015. *Collaborative Data Science*. Plotly Technologies Inc. <https://plot.ly>.
- Polars Contributors. 2024. “Visualization.” Polars. <https://docs.pola.rs/user-guide/misc/visualization/>.
- Provost, Foster, and Tom Fawcett. 2013. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. ” O’Reilly Media, Inc.”.
- Raasveldt, Mark, and Hannes Mühleisen. 2019. “DuckDB: An Embeddable Analytical Database.” *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*, 1981–84. <https://doi.org/10.1145/3299869.3320212>.
- Scott, David W. 1979. “On Optimal and Data-Based Histograms.” *Biometrika* 66 (3): 605–10. <https://doi.org/10.1093/biomet/66.3.605>.
- Shapiro, Samuel Sanford, and Martin B. Wilk. 1965. “An Analysis of Variance Test for Normality (Complete Samples).” *Biometrika* 52 (3-4): 591–611. <https://doi.org/10.1093/biomet/52.3-4.591>.
- Shneiderman, Ben. 1996. “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations.” *Proceedings of the IEEE Symposium on Visual Languages*, 336–43.
- Silverman, Bernard W. 1986. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman & Hall.
- Spearman, Charles. 1904. “The Proof and Measurement of Association Between Two Things.” *The American Journal of Psychology* 15 (1): 72–101. <https://doi.org/10.2307/1412159>.

- Tufte, Edward R. 1983. *The Visual Display of Quantitative Information*. Graphics Press.
- Tukey, John Wilder et al. 1977. *Exploratory Data Analysis*. Vol. 2. Springer.
- VanderPlas, Jake. 2016. *Python Data Science Handbook: Essential Tools for Working with Data*. ” O’Reilly Media, Inc.”.
- Vink, Ritchie. 2023. *Polars User Guide*. <https://docs.pola.rs>.
- Ware, Colin. 2012. *Information Visualization: Perception for Design*. 3rd ed. Interactive Technologies. Morgan Kaufmann Publishers.
- Waskom, Michael L. 2021. “Seaborn: Statistical Data Visualization.” *Journal of Open Source Software* 6 (60): 3021. <https://doi.org/10.21105/joss.03021>.
- Wei, Shouke. 2022a. “Convenient Methods to Encode Categorical Variables in Python.” Deepsim Blog. <https://blog.deepsim.ca/encode-categorical-variables-in-python/>.
- Wei, Shouke. 2022b. “Different Methods to Impute Missing Values of Datasets Using Python Pandas.” Deepsim Blog. <https://blog.deepsim.ca/impute-missing-values-of-datasets-python-pandas/>.
- Wei, Shouke. 2023a. “5 Practical Methods to Reshape DataFrame in Pandas.” Deepsim Blog. <https://blog.deepsim.ca/practical-methods-reshape-dataframe-pandas//>.
- Wei, Shouke. 2023b. “Data Visualization with hvPlot (II): Most Widely Used Basic Plots.” Deepsim Blog. <https://blog.deepsim.ca/data-visualization-hvplot-basic-plots/>.
- Wei, Shouke. 2025a. *Build Beautiful Web Apps in Minutes: The Complete Streamlit Guide*. <https://medium.com/@shouke.wei>.
- Wei, Shouke. 2025b. *Building Interactive Python Dashboards with Bokeh: A Complete Tutorial*. <https://medium.com/@shouke.wei>.

- Wei, Shouke. 2025c. *Consolidate Multiple CSV Datasets into a Single DuckDB Database (II): A Real-World Data Example*. <https://medium.com/@shouke.wei/consolidate-multiple-csv-datasets-into-a-single-duckdb-database-ii-a-real-world-data-example-baffa65b2309>.
- Wei, Shouke. 2025d. *DuckDB for Data Analysis and Visualization*. <https://medium.com/@shouke.wei/duckdb-for-data-analysis-and-visualization-15cf88e00294>.
- Wei, Shouke. 2025e. *DuckDB Tutorial: The Fast, Embedded Analytics Database*. <https://medium.com/@shouke.wei/duckdb-tutorial-the-fast-embedded-analytics-database-7d5d742bb251>.
- Wei, Shouke. 2025f. *Lets-Plot (II): Data Visualization Mastery with Lets-Plot and Pandas*. <https://medium.com/@shouke.wei/lets-plot-ii-data-visualization-mastery-with-lets-plot-and-pandas-2fd18535fd1a>.
- Wei, Shouke. 2025g. “Lets-Plot (IV): Data Visualization with Lets-Plot and Polars: A Comprehensive Tutorial.” Medium, August. <https://medium.com/@shouke.wei/lets-plot-iv>.
- Wei, Shouke. 2025h. *PyWaffle (I): Visualizing Data with Waffle Charts in Python*. <https://medium.com>.
- Wei, Shouke. 2025i. *Real-Time Dashboards with Watchdog and Streamlit_au-torefresh*. <https://medium.com/@shouke.wei>.
- Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer. <https://doi.org/10.1007/978-0-387-98141-3>.
- Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (10): 1–23. <https://doi.org/10.18637/jss.v059.i10>.
- Wilkinson, Leland. 2005. *The Grammar of Graphics*. 2nd ed. Statistics and Computing. Springer-Verlag. <https://doi.org/10.1007/0-387-28695-0>.
- Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt,

and Tracy K Teal. 2017. “Good Enough Practices in Scientific Computing.” *PLoS Computational Biology* 13 (6): e1005510.

Zaharia, Matei, Reynold S. Xin, Patrick Wendell, et al. 2016. *Apache Spark: A Unified Engine for Big Data Processing*. In *Communications of the ACM*, vol. 59. <https://doi.org/10.1145/2934664>.

Index

A

- actions, [345](#)
- aesthetics (visualization), [133](#)
- agg method, [52](#)
- aggregation, [311](#)
 - distributed, [374](#)
- aggregation functions, [25](#)
- aggregations, [25](#)
- AirQuality dataset, [482](#)
- Ames Housing dataset, [91](#), [92](#), [129](#),
[131](#)
- Ames, Iowa, [92](#)
- anaconda, [10](#)
- analytical pipeline, [439](#)
 - anti-patterns, [443](#)
 - configuration, [453](#)
 - functions, [444](#)
 - orchestration, [456](#)
 - saving, [468](#)
 - stages, [441](#)
 - visualisation functions, [455](#)
- animated chart, [161](#)
- annotation, [143](#)
- anomaly detection, [489](#)
- Apache Arrow, [291](#), [296](#), [395](#),
[408](#)
- Apache Spark, [333](#)
- apply method, [49](#)
- array creation, [23](#)

- array shapes, [26](#)

Arrow

- arrays, [412](#)
- ecosystem, [409](#)
- tables, [412](#)
- zero-copy, [410](#)

Axes (Matplotlib), [136](#)

axis parameter, [26](#)

B

- backward fill, [72](#)
- bar chart, [141](#)
- binning, [85](#)
- bistro module, [200](#)
- Bokeh, [185](#), [237](#), [275](#)
 - callbacks, [275](#)
 - complete dashboard, [275](#)
- boolean filtering, [47](#)
- boolean indexing, [24](#)
- boolean mask, [27](#)
- boolean Series, [47](#)
- box plot, [106](#)
- bracket notation, [46](#)
- broadcast join, [376](#)
- broadcasting, [19](#), [26](#)
- BTS On-Time Performance
 - dataset, [291](#), [292](#)
- bubble chart, [152](#)
- Bureau of Transportation
 - Statistics, [292](#)

C

capping, [75](#)
case study
 financial analysis, [472](#)
 grammar of graphics, [229](#)
 interactive analysis, [226](#)
 IoT monitoring, [482](#)
 retail analytics, [457](#)
 Streamlit dashboard, [255](#)
 waffle chart, [228](#)
Catalyst optimizer, [342](#), [344](#)
categorical encoding, [65](#), [79](#)
 summary, [82](#)
categorical summary, [98](#)
central tendency, [96](#)
centring, [26](#)
chart selection, [134](#)
chartjunk, [134](#)
cluster manager, [342](#)
cohort analysis, [464](#)
col, [349](#)
collect, [305](#)
colour (visualization), [135](#)
colour palettes, [135](#)
column chunks, [403](#)
column pruning, [407](#)
column selection, [46](#)
column statistics, [403](#)
column transformation, [48](#)
columnar storage, [402](#)
conda, [9](#)
contiguous memory, [22](#)
contingency table, [117](#)
copy (NumPy), [29](#)
correlation analysis, [108](#)
correlation heatmap, [110](#)
correlation matrix, [51](#), [109](#)

covariance matrix, [30](#), [32](#)
createOrReplaceTempView, [366](#)
cross-product matrix, [30](#)
cross-tabulation, [53](#), [117](#)
CSV, [399](#)
 limitations, [399](#)
 read performance, [400](#)
 summary, [402](#)
 type safety, [401](#)
CSV export, [45](#)
CTE, [371](#)
cumulative statistics, [57](#)

D

D'Agostino-Pearson test, [105](#)
DAG (Directed Acyclic Graph),
 [345](#)
Dash, [281](#)
dashboard design
 best practices, [281](#)
 clutter, [282](#)
 error handling, [283](#)
 KPIs, [281](#)
 layout, [282](#)
 performance, [282](#)
Dask, [189](#)
data analysis, [3](#), [4](#)
 principled workflow, [3](#)
data analysis workflow, [3](#)
data cleaning, [65](#)
Data Dashboard, [237](#)
 analytical, [239](#)
 definition, [239](#)
 operational, [239](#)
 use cases, [240](#)
data export, [45](#)
data import

- formats, 45
- data inspection, 44
- data locality, 341
- data preprocessing, 65
- data problems, 5
 - types, 4
- data science lifecycle, 4, 5
 - data cleaning, 5
 - data collection, 5
 - deployment, 5
 - evaluation, 5
 - exploratory data analysis, 5
 - modeling, 5
 - problem definition, 5
- data science project
 - structure, 6
- data storage, 395
- data transformation, 65, 76
 - comparison, 79
- data visualization, 129
- data wrangling, 65
- data-ink ratio, 134
- DataFrame, 37, 40
 - attributes, 42
 - inspection, 44
- Datashader, 189
- date slicing, 55
- DatetimeIndex, 54, 55
- descriptive analysis, 5
- descriptive statistics, 50, 96
 - summary, 100
- df.describe, 96
- df.plot, 176
- diagnostic analysis, 5
- discretisation, 85
- distributed computing, 341
- distribution analysis, 100
 - distribution comparison, 103
 - diverging palette, 135
 - dot notation, 46
 - dot product, 30
 - downsampling, 182
 - driver, 342
 - dropping columns, 49
 - dtype, 22
 - DuckDB, 395, 419
 - aggregations, 423
 - database file, 428
 - installation, 419
 - joins, 425
 - performance, 421
 - persistent database, 428
 - querying Parquet, 420
 - WAL, 429
 - window functions, 424
 - writing Parquet, 427
 - dummy variable trap, 81

E

EDA

- audit, 458
- correlation matrix, 475
- data cleaning, 459
- feature engineering, 460
- financial data, 472
- integration, 439
- IoT cleaning, 486
- IoT data audit, 484
- return distribution, 475
- risk-return analysis, 477
- temporal patterns, 487
- time series, 474
- eigendecomposition, 32, 33
- eigenvalues, 31, 32

element-wise operations, [24](#)
ETL, [299](#), [323](#)
Euclidean norm, [28](#)
Excel export, [45](#)
executor, [342](#)
exercises, [59](#), [88](#), [123](#), [166](#), [232](#), [284](#),
[328](#), [389](#), [433](#), [496](#)
expanding window, [57](#)
explain, [306](#)
exploratory data analysis (EDA),
[91](#)

F

faceted grid, [148](#)
FacetGrid, [148](#)
factory functions, [23](#)
fancy indexing, [24](#)
fastparquet, [405](#)
feature engineering, [48](#), [65](#), [83](#),
[350](#)
 family size, [84](#)
 fare per person, [87](#)
 solo traveller, [87](#)
 summary, [88](#)
 title extraction, [83](#)
feature preparation, [65](#)
Figure (Matplotlib), [136](#)
file system monitoring, [267](#)
filtering, [27](#)
financial analysis, [439](#)
forward fill, [72](#)
frequency encoding, [82](#)
frequency table, [53](#)

G

Gapminder dataset, [129](#), [131](#)
ggplot2, [133](#)

GIL (Global Interpreter Lock),
[296](#)

GitHub API, [238](#)

glob patterns, [317](#), [378](#)

Grammar of Graphics, [176](#), [195](#)

grammar of graphics, [133](#)

group differences, [115](#)

group operations, [51](#)

group_by, [311](#)

groupby, [51](#)

 pattern discovery, [121](#)

grouped bar chart, [157](#)

grouped distribution, [103](#)

H

hash join, [313](#)

heatmap, [110](#), [150](#)

histogram, [100](#)

Hive partitioning, [413](#), [414](#)

HoloViews, [183](#)

 layout, [191](#)

HoloViz, [273](#)

HoloViz ecosystem, [176](#)

hover tooltip, [159](#)

hvPlot, [171](#), [183](#)

 Dask, [189](#)

 introduction, [183](#)

 linked plots, [191](#)

 Pandas, [185](#)

 performance, [194](#)

 styling, [193](#)

I

iloc, [46](#)

imputation, [65](#)

 mean, [71](#)

 median, [71](#)

index (Pandas), [40](#)

- indexing, [23](#)
- inner product, [30](#)
- interaction feature, [86](#)
- interaction features, [86](#)
- interactive visualization, [159](#)
- interquartile range, [74](#)
- IoT
 - monitoring, [439](#)
 - Streamlit dashboard, [491](#)
- IQR, [97](#)
- IQR method, [74](#)

- J**
- join, [313](#)
 - distributed, [376](#)
- join types, [315](#)
- JupyterLab, [3](#), [11](#)
- JVM, [342](#)

- K**
- k-nearest neighbours, [76](#)
- kernel density estimation (KDE), [101](#)
- Kolmogorov-Smirnov test, [106](#)
- kurtosis, [98](#)

- L**
- label alignment, [37](#), [42](#)
- label encoding, [80](#)
- lagged features, [58](#)
- large datasets, [317](#)
 - plotting, [182](#)
 - PySpark, [378](#)
 - strategy, [323](#)
- lazy evaluation (Spark), [345](#)
- lazy execution, [305](#)
- lazy scanning, [317](#)
- LazyFrame, [297](#), [305](#)
 - collect, [309](#)
 - lazy, [309](#)
 - pipeline, [307](#)
- least squares (OLS), [31](#)
- Lets-Plot, [171](#), [195](#)
 - aes, [196](#)
 - faceting, [209](#)
 - geoms, [196](#)
 - ggplot, [196](#)
 - hierarchical data, [200](#)
 - introduction, [195](#)
 - layers, [198](#)
 - network graph, [206](#)
 - themes, [209](#)
 - tree layout, [203](#)
 - treemap, [200](#)
 - use cases, [212](#)
- line chart, [138](#)
- linear algebra, [30](#)
- linear regression, [30](#), [76](#)
- linear system, [31](#)
- linear trend, [114](#)
- lingua franca, [3](#)
- loc, [46](#)
- log returns, [473](#)
- log transformation, [78](#)
- logistic regression, [76](#)

- M**
- map method, [49](#)
- MAR, [70](#)
- Matplotlib, [129](#), [431](#)
 - Axes, [136](#)
 - Figure, [136](#)
 - limitations, [175](#)
 - savefig, [143](#)
 - styling, [177](#)

- subplots, [138](#), [179](#)
- matrix decomposition, [31](#)
- matrix multiplication, [30](#)
- matrix operations, [30](#)
- MCAR, [70](#)
- mean, [96](#)
- mean imputation, [71](#)
- measurement error, [73](#)
- measures of spread, [97](#)
- median, [96](#)
- median imputation, [71](#)
- memory layout, [29](#)
- MemoryError, [340](#)
- Min-Max scaling, [76](#)
- miniconda, [10](#)
- missing data, [65](#), [69](#)
 - types, [70](#)
- missing value indicator, [72](#)
- missing values, [44](#), [65](#)
 - detection, [69](#)
 - dropping, [70](#)
- MNAR, [70](#)
- mode, [96](#)
- moving average, [56](#)
- multicollinearity, [81](#), [113](#)
- multiple aggregations, [52](#)

N

- name parsing, [83](#)
- NaN, [69](#)
- narrow transformations, [374](#)
- ndarray, [22](#)
- NetworkX, [206](#)
- nominal variable, [80](#)
- normalisation, [76](#)
- normality assessment, [104](#)
- normality test, [105](#)

- np.genfromtxt, [21](#)
- np.where, [28](#)
- numeric summary, [96](#)
- numerical stability, [34](#)
- NumPy, [3](#), [19](#)
- Numpy, [19](#)
- NumPy array, [19](#)
- numpy.linalg, [31](#)
 - summary, [34](#)
- NYC Taxi Trip dataset, [171](#), [173](#),
[237](#), [333](#), [335](#), [396](#), [397](#)
- NYC TLC, [335](#)

O

- object dtype, [296](#)
- OLAP, [419](#)
- one-hot encoding, [80](#)
- Online Retail dataset, [457](#)
- ordinal encoding, [81](#)
- ordinal variable, [80](#)
- ordinary least squares (OLS), [31](#)
- out-of-memory, [340](#)
- outlier, [73](#)
- outlier detection, [65](#), [73](#)
 - summary, [76](#)

P

- pages (Parquet), [403](#)
- pair plot, [119](#), [151](#)
- Pandas, [3](#), [19](#), [37](#)
 - plotting, [176](#)
 - reading CSV, [38](#)
- Pandas API on Spark, [354](#)
- Panel, [273](#)
- panel data, [37](#)
- parallel coordinates, [156](#)
- parallelism, [373](#)
- Parquet, [299](#), [320](#), [346](#), [395](#)

- compression, [406](#)
 - internals, [403](#)
 - reading, [405](#)
 - writing, [405](#)
- partial string indexing, [55](#)
- partition columns
 - choosing, [417](#)
- partition pruning, [414](#)
 - verification, [418](#)
- partitioned datasets, [395](#), [413](#)
 - reading, [416](#)
 - writing, [414](#)
- partitions, [373](#)
- PCA, [30](#), [32](#)
 - by hand, [32](#)
- pd.crosstab, [117](#)
- pd.cut, [85](#)
- pd.get_dummies, [80](#)
- pd.qcut, [85](#)
- pd.read_csv, [43](#)
- Pearson correlation, [51](#), [108](#)
- performance, [28](#)
- pivot table, [53](#)
 - EDA, [121](#)
- pl.col, [302](#)
- pl.Expr, [300](#)
- Plotly, [129](#), [159](#)
 - animation, [161](#)
 - bar chart, [160](#)
 - line chart, [163](#)
 - PySpark backend, [381](#)
 - saving figures, [164](#)
 - scatter plot, [159](#)
- Polars, [291](#)
 - aggregation expressions, [312](#)
 - architecture, [296](#)
 - benchmark, [297](#)
 - data types, [304](#)
 - expressions, [315](#)
 - filtering, [302](#)
 - group_by, [311](#)
 - inspection, [300](#)
 - interoperability, [322](#)
 - joins, [313](#)
 - motivation, [296](#)
 - syntax, [300](#)
 - use cases, [299](#)
 - vs Pandas, [299](#)
 - with_columns, [302](#)
- predicate pushdown, [307](#), [407](#)
- predictive analysis, [5](#)
- prescriptive analysis, [5](#)
- principal component analysis, [32](#)
- printSchema, [347](#)
- projection pushdown, [297](#), [307](#)
- px.bar, [160](#)
- px.line, [163](#)
- px.scatter, [159](#)
- PyArrow, [405](#), [409](#)
 - operations, [413](#)
- PySpark, [333](#)
 - native plotting, [381](#)
 - performance, [379](#)
 - when to use, [341](#)
- PySpark setup, [12](#)
- pyspark.pandas, [354](#)
 - column assignment, [358](#)
 - creation, [355](#)
 - differences from Pandas, [362](#)
 - groupby, [359](#)
 - merge, [360](#)
 - operations, [357](#)
 - overview, [354](#)
 - when to use, [364](#)

- python environment, [4](#), [7](#)
 - recommended methods, [7](#)
- Python list, [19](#)
- PyWaffle, [171](#), [213](#)
 - basic chart, [214](#)
 - customisation, [215](#)
 - dashboard use cases, [220](#)
 - introduction, [213](#)
 - multi-chart layout, [219](#)

Q

- Q-Q plot, [104](#)
- qualitative palette, [135](#)
- query engines, [395](#)
- query method, [48](#)
- query optimisation, [307](#)
- query plan, [306](#)
- quiz, [35](#), [60](#), [89](#), [124](#), [168](#), [234](#), [285](#),
[329](#), [391](#), [435](#), [499](#)

R

- rank correlation, [109](#)
- rcParams, [177](#)
- RDD, [344](#)
- read.parquet, [346](#)
- reading CSV, [43](#)
- real-time dashboard, [264](#)
 - comparison, [270](#)
 - definition, [264](#)
- renaming columns, [49](#)
- reproducible analysis, [4](#), [6](#)
- resampling, [56](#)
- Resilient Distributed Dataset,
[344](#)
- retail analytics, [439](#)
- RFM analysis, [464](#)
- right-skewed distribution, [97](#)
- RMS Titanic, [67](#)

- robust scaling, [78](#)
- rolling statistics, [473](#)
- rolling window, [56](#)
- rolling z-score, [489](#)
- row groups, [403](#)
- row index, [299](#)
- row-oriented storage, [402](#)
- Rust, [291](#), [297](#)

S

- savefig, [143](#)
- saving figures, [143](#)
- scalability, [340](#)
- scan_csv, [317](#)
- scan_parquet, [320](#)
- scatter matrix, [119](#)
- scatter plot, [114](#), [143](#)
- scikit-learn, [3](#), [19](#)
- Scipy, [19](#)
- Seaborn, [129](#), [145](#)
 - categorical plots, [146](#)
 - design philosophy, [145](#)
 - distribution plots, [145](#)
 - FacetGrid, [148](#)
 - heatmap, [150](#)
 - limitations, [175](#)
 - pairplot, [151](#)
 - regression plot, [147](#)
- select, [350](#)
- sequential palette, [135](#)
- Series, [37](#), [40](#), [42](#)
- Shapiro-Wilk test, [105](#)
- shift method, [58](#)
- shuffle, [341](#), [374](#)
- SIMD, [291](#)
- singular value decomposition
(SVD), [33](#)

- skewness, [79](#), [98](#)
- slicing, [23](#)
- sns.boxplot, [146](#)
- sns.displot, [145](#)
- sns.heatmap, [150](#)
- sns.histplot, [145](#)
- sns.kdeplot, [145](#)
- sns.lmplot, [147](#)
- sns.pairplot, [151](#)
- sns.regplot, [147](#)
- sns.stripplot, [146](#)
- sns.violinplot, [146](#)
- sort-merge join, [376](#)
- Spark
 - architecture, [342](#)
 - DataFrame, [344](#)
- Spark Connect, [339](#), [344](#)
- Spark DataFrame
 - filter, [349](#)
 - groupBy, [374](#)
 - join, [376](#)
 - loading, [346](#)
 - operations, [352](#)
 - schema, [347](#)
 - select, [349](#)
 - select with expressions, [350](#)
 - writing, [352](#)
- Spark SQL, [366](#)
 - aggregation, [369](#)
 - vs DataFrame API, [372](#)
 - window functions, [370](#)
- spark.read, [346](#)
- spark.sql, [366](#), [368](#)
- spark.sql.shuffle.partitions, [373](#)
- SparkContext, [342](#)
- SparkSession, [342](#)
- Spearman correlation, [109](#)
- split-apply-combine, [51](#)
- SQLite, [419](#)
- st.cache_data, [242](#), [282](#)
- st.session_state, [242](#)
- stacked bar chart, [157](#)
- standard deviation, [97](#)
- standard score, [74](#)
- standardisation, [27](#), [77](#)
- streaming execution, [317](#), [319](#)
- Streamlit, [237](#), [241](#)
 - advantages, [241](#)
 - columns, [249](#)
 - Community Cloud, [272](#)
 - complete dashboard, [255](#)
 - dashboard patterns, [262](#)
 - deployment, [270](#)
 - display functions, [244](#)
 - Docker, [272](#)
 - execution model, [242](#)
 - expander, [249](#)
 - filtering data, [247](#)
 - first app, [243](#)
 - Heroku, [272](#)
 - hvPlot, [254](#)
 - installation, [241](#)
 - KPI cards, [255](#)
 - local deployment, [270](#)
 - Matplotlib, [250](#)
 - Plotly, [252](#)
 - rerun, [242](#)
 - Seaborn, [250](#)
 - session state, [247](#)
 - tabs, [249](#)
 - widgets, [245](#)
- streamlit_autorefresh, [237](#), [264](#)
- subplot grid, [138](#)
- subplot layout, [179](#)

support vector machines, [76](#)
SVD, [33](#)

T

temporary view, [366](#)
TensorFlow, [19](#)
time series (Pandas), [54](#)
 summary, [58](#)
time series plot, [138](#)
Titanic dataset, [65](#), [66](#), [129](#), [130](#)
tool selection, [433](#)
traditional statistics, [3](#)
transformations, [345](#)
transpose, [31](#)
Tufte, Edward, [129](#), [134](#)
Tukey, John W., [91](#)

U

UCI Machine Learning Repository,
 [20](#), [38](#), [482](#)
universal functions (ufuncs), [25](#)
uv, [7](#)

V

value counts, [98](#)
value_counts, [53](#)
variance, [97](#)
vectorization, [19](#)
vectorization vs. loops, [28](#)
vectorized arithmetic, [24](#)
view (NumPy), [29](#)
Vinho Verde, [20](#), [38](#)
violin plot, [115](#)

visualization

 advanced, [171](#)
 EDA role, [442](#)
 integration, [439](#)
 library comparison, [222](#)
 performance, [224](#)
 tool selection, [175](#)
 when to use, [223](#)
visualization libraries
 summary, [166](#)

W

waffle chart, [213](#)
watchdog, [237](#), [267](#)
wide transformations, [374](#)
Wilkinson, Leland, [133](#)
window functions, [370](#)
wine quality dataset, [19](#), [37](#), [129](#),
 [130](#)
Winsorisation, [75](#)
WITH clause, [371](#)
write-ahead log, [429](#)

Y

Yahoo Finance, [472](#)
yfinance, [472](#)

Z

Z-score, [74](#)
Z-score method, [74](#)
Z-score scaling, [77](#)
zero-copy, [409](#), [410](#)