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/

9 781067 559205

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

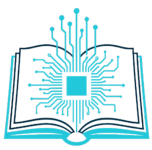
Data Exploration, Visualization, and Scalable Data Processing

Practical Data Analysis and Visualization with Python

Data Exploration, Visualization, and Scalable Data Processing

---

Shouke Wei



DEEPSIM PRESS

First Edition, 2026

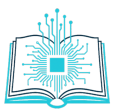
For permission requests, contact the publisher:
Email: shouke.wei@deepsim.ca

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

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

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
```

> **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	`uv` (Recommended)	`conda` / Miniconda
Non-Python dependencies	Limited	Excellent (e.g., CUDA)
Lockfile support	Yes (`uv.lock`)	Partial (`environment.yml`)
Standard compatibility	`pip`/`pyproject.toml`	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/dm³
`volatile acidity`	Acetic acid (vinegar effect)	g/dm³
`citric acid`	Freshness and flavour	g/dm³
`residual sugar`	Sugar after fermentation	g/dm³
`chlorides`	Salt content	g/dm³
`free sulfur dioxide`	Free SO (antimicrobial)	mg/dm³
`total sulfur dioxide`	Total SO	mg/dm³
`density`	Density of wine	g/cm³
`pH`	Acidity scale	—
`sulphates`	Antimicrobial additive	g/dm³
`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
```

> **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))          # 3×4 array of zeros
np.ones((2, 5))           # 2×5 array of ones
np.eye(4)                 # 4×4 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)     # 3×3 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.feat.engineering/.

Li, Guangyang. 2024. *PyWaffle: Make Waffle Charts in Python*. Https://pywaffle.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_autorefresh*. 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

Q

R

S