DATA SCIENCE NOTES: A ONE SEMESTER COURSE

Patrick McDowell and AI

JANUARY 8, 2026
SOUTHEASTERN LOUISIANA UNIVERSITY

Table of Contents

1.0 Course Outline: Introduction to Data Science 6

Course Description 6

Instructional Approach 6

Course Objectives 6

Prerequisites 7

Course Structure 7

Outcomes 9

Chapter 1: Python as a Computational Language for Data Science 13

Chapter Overview 13

1.1 Algorithms and Computational Thinking 13

1.2 Variables, Expressions, and Mathematical Notation 14

1.3 Control Flow: Decisions and Repetition 14

1.4 Functions as Reusable Algorithms 15

1.5 Representing Data in Python 16

1.6 Manual Implementation of Statistical Computations 18

1.7 From Algorithms to Libraries: Introducing NumPy 20

1.8 Abstraction, Efficiency, and Understanding 20

Chapter Summary 21

Chapter 1 Review Questions and Exercises 21

Chapter 2: Data and Data Visualization 25

2.1 What Is Data? 25

2.2 Types of Data 26

2.3 Data Quality and Common Issues 26

2.4 Exploratory Data Analysis (EDA) 27

2.5 Why Visualization Matters 27

2.6 Visualizing a Single Variable 28

2.7 Visualizing Relationships Between Variables 30

2.8 Visualizing Categorical Data 31

2.9 From Concepts to Implementation 34
2.10 Visualization as a Thinking Tool 34
2.11 Implementing Basic Visualizations in Python 35
2.12 Chapter Summary 39
Chapter 2 Exercises: Data and Data Visualization 39
Chapter 3: Statistics for Data Science 41
3.1 Why Statistics Matters in Data Science 41
3.2 Measures of Central Tendency 41
3.3 Measures of Variability 42
3.4 Visualizing Spread and Variability 44
3.5 Data Distributions 44
3.6 Z-Scores and Standardization 46
3.8 Sampling and Variability 46
3.9 From Concepts to Computation 46
3.10 Statistics as a Foundation for Learning Models 47
Chapter 3 Exercises: Describing and Understanding Data 47
Chapter 4: Linear Algebra for Data Science 52
4.1 Scalars, Vectors, and Matrices 52
4.2 Vectors as Data Points 53
4.3 Vector Operations and Interpretation 55
4.4 Geometry and Intuition 59
4.5 Matrices as Datasets 59
4.6 Matrix Operations 60
4.7 Systems of Linear Equations 63
4.8 Linear Transformations 70
4.9 Linear Algebra in Learning Models 71
4.10 From Concepts to Computation 71
Chapter Summary 72
Chapter 4 Exercises: Linear Algebra for Data Science 72

Chapter 5: Learning from Data 77
5.1 What Does It Mean to “Learn” from Data? 77
5.2 Linear Models 77
5.3 Linear Regression 78
5.4 Error and Loss 79
5.5 Learning as Optimization 83
5.6 Gradient Descent (Conceptual) 83
5.7 Adaptive Linear Elements (ADALINE) 85
5.8 ADALINE Learning Rule (Intuition) 86
5.9 Linear Algebra View of Learning 86
5.10 From Linear Models to Neural Networks 87
Chapter 5 Review Questions and Exercises: Learning from Data 88
Chapter 6: Implementing Linear Models 92
6.1 From Equations to Algorithms 92
6.2 A Simple Linear Regression Model (One Variable) 92
6.3 Manual Implementation: Prediction and Error 93
6.4 Average Loss over a Dataset 93
6.5 Manual Gradient Descent (One Variable) 94
6.6 Iterative Learning 95
6.7 Extending to Multiple Inputs (ADALINE) 97
6.8 Manual ADALINE Learning Rule 97
6.9 NumPy Implementation 99
6.10 A Brief Look at Library-Based Implementations (scikit-learn) 100
6.11 Comparing Manual and Library Approaches 103
Chapter Summary 104
Chapter 6 Exercises: Implementing Linear Models 104
Chapter 7: Course Projects 108
Project 1: Statistical Frequency Analysis and Message Decoding 108
Project 2: Noise Reduction and Outlier Detection in a Time Series 111

Project 3: Geometric Transformations Using a 2D Rotation Matrix 137
Appendix A: Python Quick Reference 144
Appendix B: NumPy Quick Reference 150
Appendix C – Differences in Syntax and Functionality of Python and Traditional Languages (Java/C/C++) 155
Appendix D: Pygame Template 157
References and Acknowledgements 159

1.0 Course Outline: Introduction to Data Science

Course Description

This course provides an introduction to data science through a practical, hands-on approach. Students will learn foundational programming skills in Python, explore core data science concepts such as data visualization and statistics, develop an applied understanding of linear algebra, and gain exposure to introductory machine learning and neural network models. Emphasis is placed on understanding data, extracting insights, and building intuition for modern data-driven methods.

Instructional Approach

This course emphasizes conceptual understanding before implementation. For each major topic, students will:

1. Develop intuition using diagrams, equations, and pseudocode
2. Implement core ideas directly using basic Python constructs
3. Introduce optimized library implementations (e.g., NumPy, Pandas)
4. Compare manual and library-based approaches for correctness, efficiency, and clarity

This approach is designed to help students understand not only *how* data science tools work, but *why* they work.

Course Objectives

By the end of this course, students will be able to:

- Use Python to load, manipulate, analyze, and visualize data
- Understand and apply basic statistical concepts to real datasets
- Perform and interpret matrix and vector operations relevant to data analysis
- Explain and implement simple learning models, including adaptive linear elements
- Describe the structure and operation of basic feedforward neural networks
- Connect mathematical concepts to practical data science applications

Prerequisites

- Basic algebra
- Comfort with high-school–level mathematics
  (No prior programming experience assumed)

Course Structure

The course is organized into three main components:

1. **Python Programming Foundations**
2. **Core Data Science Concepts**
3. **Introductory Machine Learning and Neural Networks**

The course will be taught in a series of 7 chapters. They are listed and outlined below:

Chapter 1: Python as a Computational Language for Data Science

Overview

This chapter introduces Python as a language for expressing algorithms, mathematical formulas, and data-driven procedures. Emphasis is placed on translating ideas—expressed through diagrams, equations, and pseudocode—into working Python programs. Students will implement core computations manually before transitioning to optimized library-based solutions.

Learning Objectives

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

- Translate pseudocode and mathematical formulas into Python code
- Implement basic algorithms using explicit, step-by-step logic
- Use Python data structures to represent datasets
- Understand the difference between manual and library-based implementations
- Read and write simple Python programs for data analysis tasks

Instructional Pattern (Used Throughout the Course)

For each new concept:

1. Conceptual explanation using diagrams and formulas
2. Pseudocode representation of the idea
3. Manual Python implementation (loops, conditionals, functions)
4. Introduction of optimized library tools (NumPy, Pandas)
5. Comparison of approaches (clarity, efficiency, scalability)

Topics

1. Python as Executable Pseudocode

- What is an algorithm?
- Relationship between pseudocode and Python
- Readability and clarity in code
- Python syntax as structured logic

2. Variables, Expressions, and Data Types

- Variables as named quantities
- Numeric types (integers, floats)
- Basic arithmetic operations
- Assignment and evaluation order
- Using variables to represent formula components

4. Functions as Reusable Algorithms

- Motivation for functions
- Input, output, and return values
- Writing functions that implement formulas
- Testing functions with simple datasets

5. Core Data Structures for Data Representation

- Lists as ordered collections of data

- Dictionaries for labeled data
- Indexing and iteration
- Representing datasets using basic Python structures

6. Manual Implementation of Data Computations

- Computing sums, averages, and counts
- Implementing variance and standard deviation from formulas
- Working with loops to process data
- Emphasis on correctness and transparency

7. Transition to Numerical Libraries (NumPy)

- Limitations of manual implementations
- Introduction to NumPy arrays
- Vectorized operations vs loops
- Rewriting earlier computations using NumPy
- Discussion of performance and abstraction

Assignments and Activities

- Translate given pseudocode into Python
- Implement statistical formulas manually
- Rewrite manual implementations using NumPy
- Compare outputs and performance
- Short reflection on abstraction vs understanding

Outcomes

Students complete this chapter with a clear understanding that Python is a tool for implementing ideas. They are prepared to engage with data science topics by first understanding the underlying concepts and then applying appropriate computational tools.

Chapter 2: Data and Data Visualization

Topics:

- What is data? (features, observations, labels)
- Types of data (numerical, categorical)
- Data quality and common issues (missing values, outliers)
- Exploratory Data Analysis (EDA)
- Data visualization principles
- Common plots:
    - Histograms
    - Scatter plots
    - Line plots
    - Bar charts

Learning Outcomes:

- Explore and summarize datasets
- Use visualization to identify trends and patterns
- Ask and answer questions using data

Chpater 3: Statistics for Data Science

Topics:

- Descriptive statistics:
    - Mean, median, mode
    - Variance and standard deviation
- Data distributions and histograms
- Normal distribution and intuition
- Z-scores and standardization
- Correlation and covariance

- Sampling and variability
- Introduction to statistical inference (conceptual)

Learning Outcomes:

- Compute and interpret basic statistics
- Understand uncertainty and variability in data
- Relate statistical measures to real-world datasets

Chapter 4: Linear Algebra for Data Science

Topics:

- Scalars, vectors, and matrices
- Vector operations and interpretation
- Matrix addition and multiplication
- Dot products and geometric intuition
- Systems of linear equations
- Representing datasets as matrices
- Linear transformations in data analysis

Learning Outcomes:

- Perform basic matrix and vector operations
- Interpret linear algebra concepts in a data context
- Understand the mathematical foundations of learning models

Chapter 5: Introduction to Machine Learning Concepts

Topics:

- What is machine learning?
- Supervised vs unsupervised learning
- Linear regression as a learning model

- Cost functions and error measures
- Gradient descent (conceptual and applied)

Learning Outcomes:

- Explain how machines learn from data
- Implement simple learning algorithms
- Interpret model performance and error

Chapter 6: Adaptive Linear Elements (ADALINE)

Topics:

- Perceptron vs ADALINE
- Linear neurons and weighted sums
- Learning rules
- Gradient descent for ADALINE
- Visualization of learning and convergence
- Relationship to linear regression

Learning Outcomes:

- Implement an ADALINE model
- Explain learning as an optimization process
- Connect statistics and linear algebra to learning systems

Assessment Methods (Flexible)

- Programming assignments
- Data analysis projects
- Visualization exercises
- Quizzes on core concepts
- Final project or capstone exercise

Chapter 1: Python as a Computational Language for Data Science

Chapter Overview

Data science is fundamentally about expressing ideas precisely and applying them to data. In this chapter, Python is introduced not as an end in itself, but as a language for expressing algorithms, mathematical formulas, and data-driven procedures. Throughout the chapter, concepts are developed using intuition, diagrams, formulas, and pseudocode before being implemented in Python. Optimized library-based solutions are introduced only after the underlying ideas are fully understood.

1.1 Algorithms and Computational Thinking

Concept

An algorithm is a finite sequence of steps used to solve a problem. In data science, algorithms transform raw data into useful information.

Algorithms can be expressed in:

- Natural language
- Diagrams and flowcharts
- Mathematical notation
- Pseudocode
- Executable code

Python serves as a bridge between human reasoning and computation.

Example (Pseudocode)

```
take two numbers
add them
store the result
display the result
```

Python Implementation

```
a = 3
b = 5
result = a + b
print(result)
```

Key Idea: Python closely mirrors well-written pseudocode.

In the above example, the symbols ‘a’ and ‘b’ are assigned the values, 3, and 5. When they summed together, the answer is placed into result, which is then printed.

1.2 Variables, Expressions, and Mathematical Notation

Variables represent quantities. Expressions represent computations on those quantities.

In mathematics the equation of line is:

$$y = mx + b$$

In Python we can calculate the value of y as follows:

```
m = 2
x = 4
b = 1
y = m * x + b
```

Python uses familiar arithmetic symbols, making it a natural language for implementing mathematical ideas.

1.3 Control Flow: Decisions and Repetition

Real-world algorithms require:

- Decisions
- Repeated computation

Decisions are represented as Conditional Logic using if statements

Pseudocode

```
if value is greater than threshold:
    take action
```

Python Implementation

```
value = 10
threshold = 20

if value > threshold:
    action()
```

Repetition using looping (Processing Data)

Pseudocode

```
set total to 0
for each value in data:
    add value to total
```

Python Implementation

```
data = [1, 2, 3, 4]
total = 0
for value in data:
    total += value
```

1.4 Functions as Reusable Algorithms

Functions package algorithms into reusable units.

Pseudocode

```
function average(data):
    compute sum
    divide by number of values
    return result
```

Python Implementation

```
def average(data):
    total = 0
    for value in data:
        total += value
    return total / len(data)
```

Functions allow us to separate *what* an algorithm does from *where* it is used. Another thing that functions provide is way to code something once and use it many times. In the example above, the average of groups of numbers may be needed several times throughout the code. The programmer could repeat the logic each time needed or create the "average" function and call it each time it is needed. By doing this, space in the code is saved and it creates a much more concise and easier to follow code space.

Note: By now you may have noticed that unlike other languages such as C++ or Java, the Python syntax does not use the semi-colon to end statements. In fact, there are several important differences between Python and more traditional languages. These differences are detailed in Appendix A.

1.5 Representing Data in Python

Data science works with collections of data.

Lists

```
data = [1, 2, 3, 4]
```

Lists represent ordered collections and allow iteration. Although it appears to be very similar to an array in the more traditional languages, lists have other properties that provide much more utility than just a simple array. The table below, generated using Google AI nicely summarizes the differences between Python lists and the more typical array structures of traditional languages.

Python's list is a dynamic, heterogeneous data structure, while C/C++/Java arrays are generally static, fixed-size, and homogeneous. Python lists are implemented as dynamic arrays under the hood, managing memory allocation automatically to allow for size changes.

Feature	*Python list*	*C/C++/Java Array*
Data Types	*Heterogeneous (can store mixed data types)*	*Homogeneous (stores a single, specific data type)*
Size Flexibility	*Dynamic (can grow or shrink in size automatically)*	*Static/Fixed (size must be specified at creation and cannot be changed)*
Memory Management	*Automatic (handled by Python's garbage collector)*	*Manual in C/C++ (via malloc/new and free/delete), automatic in Java (garbage collector, but array size is still fixed)*
Memory Usage	*Higher (stores references to objects plus type information)*	*Lower/more compact (stores values directly in a contiguous block)*
Performance	*Slower for numerical operations due to overhead*	*Faster for large numerical operations due to contiguous memory and type specificity*
Underlying Structure	*A dynamic array of pointers to objects*	*A contiguous block of memory storing the actual data elements*
Functionality	*Built-in methods for manipulation (append, remove, sort, etc.)*	*Requires external utility functions/libraries (e.g., Arrays.sort() in Java, C++ STL algorithms)*

In C++ and Java, dynamic array functionality similar to Python's list is available through standard library classes like C++ std::vector and Java ArrayList, which manage resizing and memory allocation automatically.

Dictionaries

```
record = {"height": 170, "weight": 65}
```

Dictionaries associate values with labels.

These structures form the foundation for more advanced data representations.

Here is a short example of a python dictionary taken from Google AI.

```
# Create a dictionary named 'car_info'
car_info = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}

# Print the entire dictionary
print(car_info)
# Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

# Access a value using its key
print(car_info["model"])
# Output: Mustang

# Add a new key-value pair
car_info["color"] = "red"
print(car_info)
# Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

1.6 Manual Implementation of Statistical Computations

Understanding statistics requires understanding how computations are performed.

Standard Deviation Formula

$$\sigma = \sqrt{\frac{1}{N}\sum(x_i - \mu)^2}$$

Before we go too far, we need to explain what Standard Deviation is a measure of. Essentially is a measure of how close the items in a set of data are to the mean of the data. In the formula above, the symbol σ is commonly used to represent the standard deviation. The x_i denotes the i^{th} value of the data set, while the μ symbol represents the mean or average value of the data set. The Σ symbol is a summation. In the statistics chapter we will delve into the details of this and other descriptive statistics.

Pseudocode

```
compute mean
for each value:
    subtract mean
    square result
average squared values
take square root
```

Basic Python Implementation

```
import math

def stdDev(data):
    # Sum the data.
    mySum = 0
    for x in data:
        mySum = mySum + x

    # Find the mean of the data.
    mean = mySum/len(data)

    # Sum the displacement of each value from the mean.
    total = 0
    for x in data:
        total = total + (x - mean)**2

    # Find the variance and the standard deviation.
    variance = total/len(data)
    myStdDev = math.sqrt(variance)

    return myStdDev
```

Python Implementation using functions *sum* and *len*

```
import math

def standard_deviation(data):
    mean = sum(data) / len(data)
    total = 0
    for x in data:
        total += (x - mean) ** 2
    return math.sqrt(total / len(data))
```

Looking at the two functions above, it can be noted that using the built in function sum essentially replaces the entire summing loop. There are many built in functions that can be used to create very powerful concise code in Python. Notice the use of the pound sign '#'

the basic implementation. It is used to indicate comments. Comments are used to make the code more clear.

1.7 From Algorithms to Libraries: Introducing NumPy

Manual implementations are clear but inefficient for large datasets.

NumPy provides optimized operations that implement the same mathematics.

```
import numpy as np

data = np.array([1, 2, 3, 4])
std = np.std(data)
```

Important: Libraries do not change the math — they change performance and convenience.

1.8 Abstraction, Efficiency, and Understanding

Manual code:

- Improves understanding
- Is easy to reason about
- Is slower for large data

Library-based code:

- Is efficient
- Is concise
- Relies on abstraction

A data scientist must understand both.

Chapter Summary

- Python is a language for expressing algorithms
- Understanding precedes optimization
- Libraries build on well-known computational patterns
- This workflow will be used throughout the book

Chapter 1 Review Questions and Exercises

Review Questions

Conceptual Questions

1. What is an algorithm?
   Give two examples of algorithms that do *not* involve programming.
2. Why is pseudocode useful when designing an algorithm?
3. In your own words, explain why Python can be viewed as "executable pseudocode."
4. What is the difference between:
   - A variable and an expression?
   - An algorithm and a function?
5. Why is it important to implement a computation manually before using a library function?
6. What is meant by *abstraction* in the context of numerical libraries such as NumPy?
7. Describe one advantage and one disadvantage of:
   - Manual implementations
   - Library-based implementations

Mathematical and Algorithmic Questions

8. Consider the formula

$$y = ax^2 + bx + c$$

Identify which parts of the formula would naturally correspond to:

   - Variables
   - Expressions
   - Algorithmic steps

9. Explain, step by step, how the mean of a dataset is computed.
   Do not use code.

10. What does standard deviation measure, conceptually?
    How does it differ from simply computing the range of the data?

11. In the standard deviation formula:

$$\sigma = \sqrt{\frac{1}{N}\sum(x_i - \mu)^2}$$

explain the meaning of each symbol.

Python and Data Representation

12. What is a Python list, and how does it differ from an array in a language such as C or Java?

13. Why are lists well suited for representing datasets in early data analysis?

14. When might a dictionary be more appropriate than a list for storing data?

Exercises

Section A: Pseudocode and Algorithm Design

1. Write pseudocode to compute the maximum value in a list of numbers.
2. Write pseudocode to count how many values in a dataset are greater than a given threshold.
3. Write pseudocode for an algorithm that computes the average of a dataset but ignores negative values.

Section B: Manual Python Implementations

4. Write a Python function that computes the **sum** of a list of numbers without using the built-in sum() function.
5. Write a function that computes the **mean** of a dataset using a loop.
6. Write a function that computes the **variance** of a dataset using the formula:

$$\text{Var} = \frac{1}{N}\sum(x_i - \mu)^2$$

7. Modify your variance function to compute the **standard deviation**.

Section C: Comparison with Library Implementations

8. Using NumPy, compute the mean and standard deviation of the same dataset used in Exercises 5–7.
9. Verify that your manual implementation and NumPy's results match (within reasonable numerical precision).
10. Rewrite one of your manual loop-based implementations using NumPy vectorized operations.

Section D: Reasoning and Reflection

11. Explain, in a short paragraph, what information is *lost* when using a library function instead of a manual implementation.
12. Describe a situation in which writing a manual implementation would be preferable to using a library.
13. Describe a situation in which using a library implementation is clearly the better choice.

Challenge Problems (Optional)

14. Write a function that computes the standard deviation **without** explicitly computing the mean first.
15. Write a function that returns both the mean and standard deviation of a dataset in a single pass through the data.
16. For a very large dataset, explain why NumPy's implementation is significantly faster than a Python loop, even though the algorithm is mathematically identical.

Chapter 1 Takeaway Exercise

17. Choose a simple mathematical formula of your choice.

- Explain it in words
- Write pseudocode
- Implement it manually in Python
- Implement it using a library (if applicable)
- Compare the two approaches

Chapter 2: Data and Data Visualization

Chapter Overview

Data science begins with data: collecting it, understanding it, and learning how to ask meaningful questions about it. Before applying statistical methods or learning models, it is essential to explore data and understand its structure, quality, and basic characteristics.

In this chapter, we introduce the concept of data, discuss common data types and representations, and develop tools for exploring datasets visually. Visualization is treated not as decoration, but as a fundamental analytical tool that helps reveal patterns, trends, and anomalies.

As in Chapter 1, we begin with ideas and reasoning, followed by algorithmic thinking, and then implementation using Python and standard data science libraries.

2.1 What Is Data?

At its core, **data** is a collection of observations about some system or phenomenon.

An observation may represent:

- A person
- A measurement
- A time step
- An experiment
- An event

Each observation is typically described by a set of **features** (also called variables or attributes).

Key Terms

- **Observation (row):** A single data point or record
- **Feature (column):** A measurable property or characteristic
- **Dataset:** A collection of observations and features

Thinking in terms of rows and columns provides a mental model that will be reused throughout the course.

An easy way to think about this is by using an example. Suppose we are collecting data about atmospheric conditions near the university. Suppose our data collection equipment collects temperature, humidity, wind direction at regular intervals, say once every 10 seconds. Each observation, taken every 10 seconds, is considered to be a row, or record. Each attribute of the observation (temperature, humidity, etc.) is a column.

Thinking in terms of rows and columns provides a mental model that will be reused throughout the course.

2.2 Types of Data

Not all data behaves the same way. Understanding data types is critical for choosing appropriate analysis and visualization methods.

Numerical Data

- **Continuous:** Can take on any value in an interval (e.g., height, weight, temperature). These are described by real numbers, most often kept in floating point format in the computer.
- **Discrete:** Takes on specific values (e.g., number of students, number of errors). These are described by integers.
- **Categorical Data**
  - **Nominal:** Categories with no inherent order (e.g., color, country)
  - **Ordinal:** Categories with an order (e.g., small, medium, large)

Different data types support different operations and visualizations.

2.3 Data Quality and Common Issues

Real-world data is often messy.

Common issues include:

- Missing values
- Outliers

- Inconsistent units
- Data entry errors
- Duplicate records

Before analysis, data must often be **cleaned**, which may involve:

- Removing or imputing missing values. **Imputing** missing values means **replacing missing data with a reasonable estimated value**, rather than deleting the observation.
- Identifying and investigating outliers
- Standardizing formats and units

Exploratory analysis helps reveal these issues early.

2.4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is the process of investigating data to understand its structure, patterns, and limitations before applying formal models.

EDA asks questions such as:

- What does a typical value look like?
- How spread out is the data?
- Are there unusual values?
- Are there visible relationships between variables?

Visualization plays a central role in EDA.

2.5 Why Visualization Matters

Visualization allows humans to detect patterns that are difficult to see in raw numbers.

Good visualizations:

- Reveal structure
- Highlight trends
- Expose anomalies

- Support reasoning and communication

Poor visualizations can mislead or obscure important information. The goal is insight, not decoration.

2.6 Visualizing a Single Variable

Histograms

A histogram shows how data values are distributed across intervals (bins).

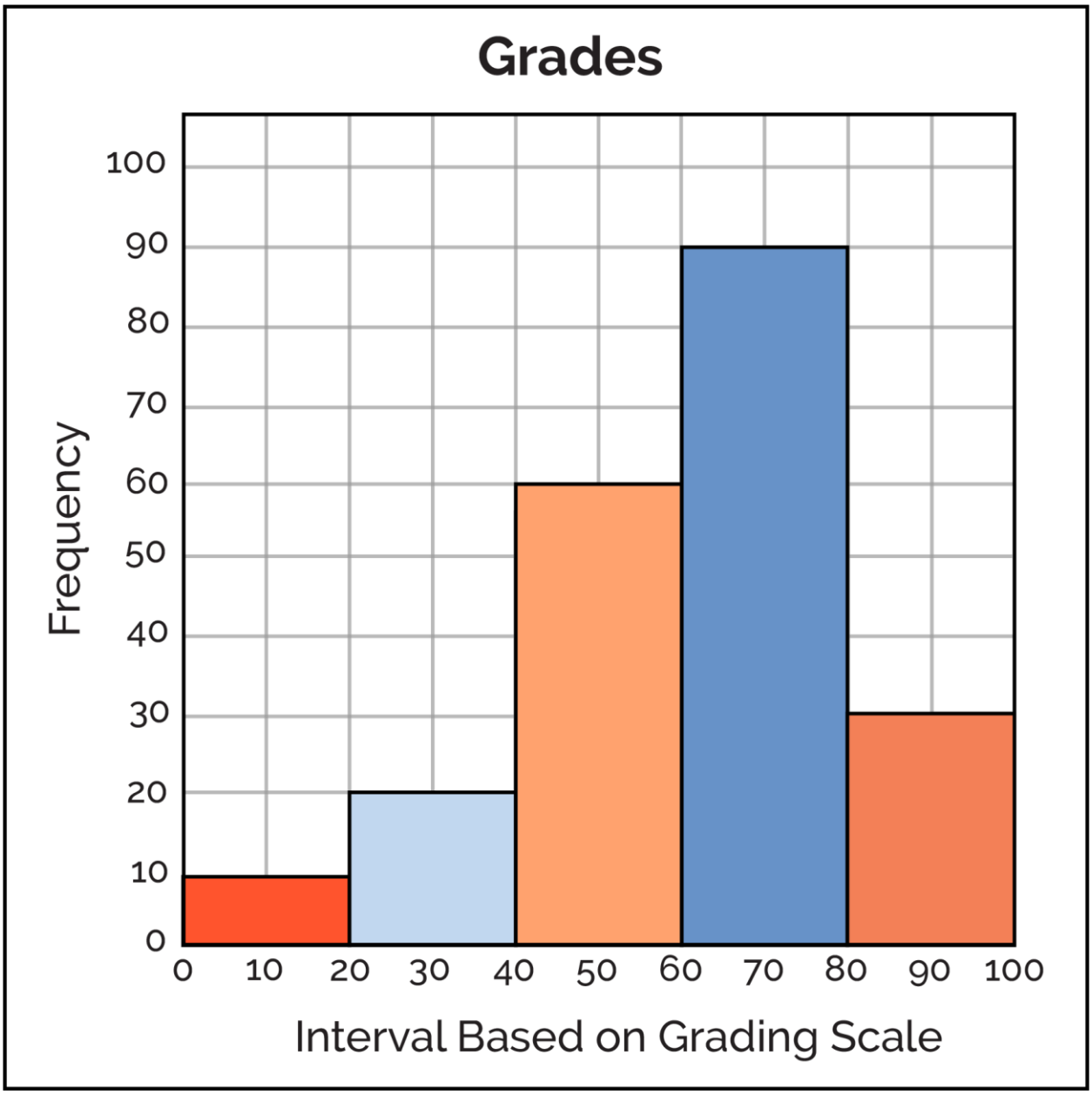
Conceptually, constructing a histogram involves:

1. Choosing bin ranges
2. Counting how many values fall into each bin
3. Displaying the counts graphically

Histograms help answer questions such as:

- Is the data symmetric?
- Is it skewed?
- Are there multiple clusters?

Histograms are closely related to probability distributions introduced later in the course. Below is an example of a histogram [1].



Summary Statistics vs Visualization

While statistics such as mean and standard deviation summarize data numerically, visualization provides a complementary perspective that can reveal features hidden by summary measures alone.

2.7 Visualizing Relationships Between Variables

Scatter Plots

Scatter plots are used to visualize relationships between two numerical variables.

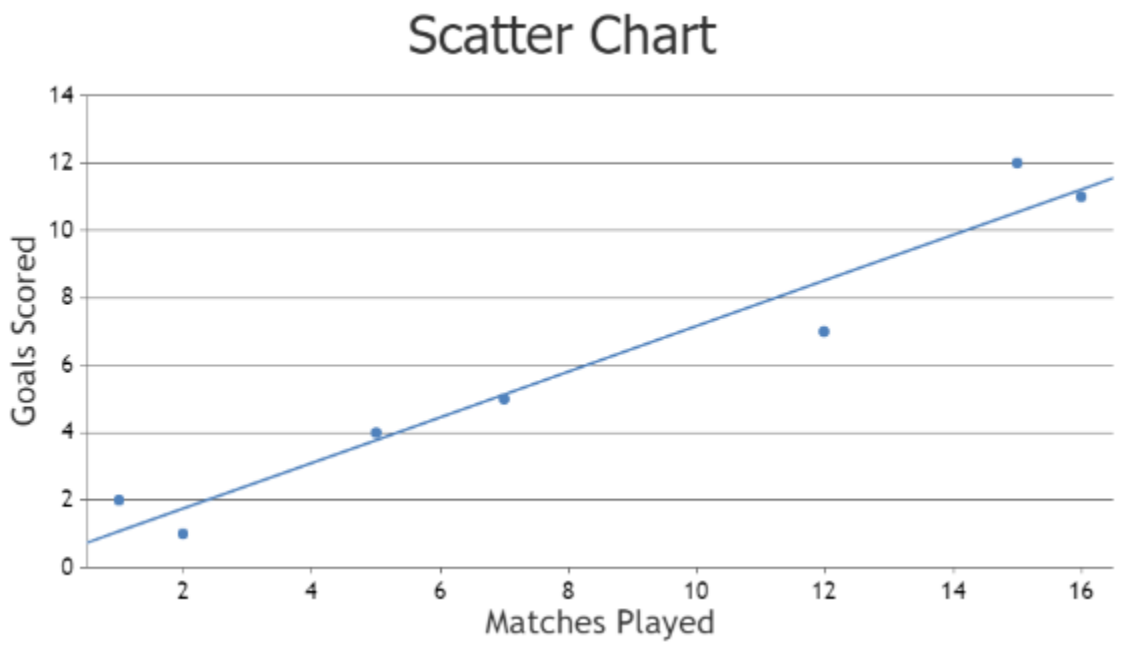
Each point represents:

- One observation
- One value on the horizontal axis
- One value on the vertical axis

Scatter plots help identify:

- Correlations
- Trends
- Clusters
- Outliers

They are foundational for understanding regression and learning models later in the course. Below is an example of a scatter plot [2].



Line Plots / Line Graph

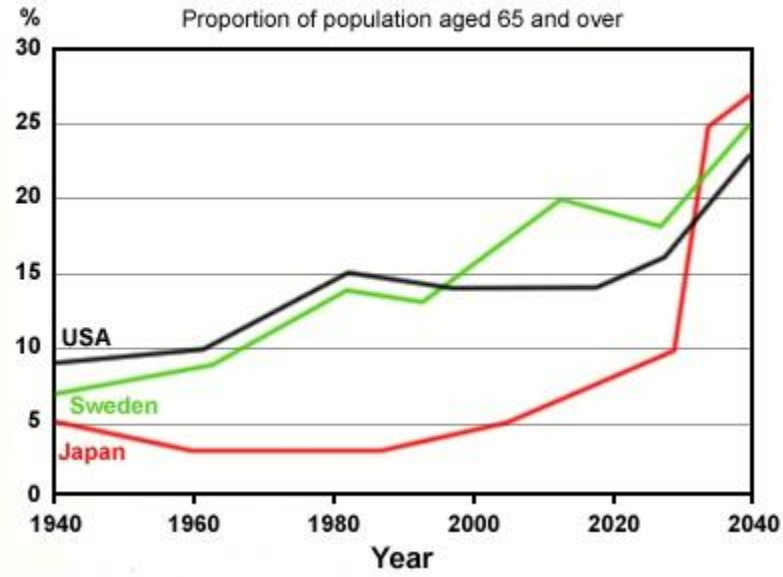
Line plots are useful when:

- One variable represents time
- Order matters

They are commonly used for:

- Time series data
- Sequential measurements
- Monitoring changes over time

An example of a Line Plot is below:



2.8 Visualizing Categorical Data

Bar Charts

Bar charts summarize categorical data by:

- Counting observations in each category
- Displaying counts or proportions

They are useful for:

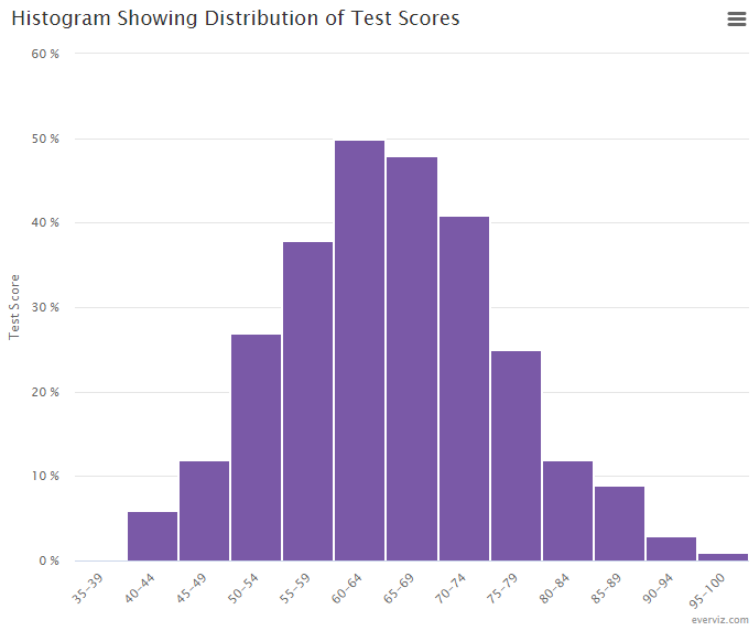
- Comparing groups
- Highlighting differences between categories

Care must be taken to ensure axes and labels are clear to avoid misleading interpretations.

Bar Charts vs. Histograms

Although bar charts and histograms may look similar, they serve different purposes and are used with different types of data. **Bar charts** are used for **categorical data**, where each bar represents a distinct category and the bars are separated by gaps to emphasize that the categories are discrete. The height of each bar indicates the count or proportion of observations in that category. **Histograms**, on the other hand, are used for **numerical data** and show how values are distributed across continuous intervals, or bins. In a histogram, the bars typically touch to indicate that the data values form a continuous range. In short, bar charts compare categories, while histograms describe the distribution of numerical values.

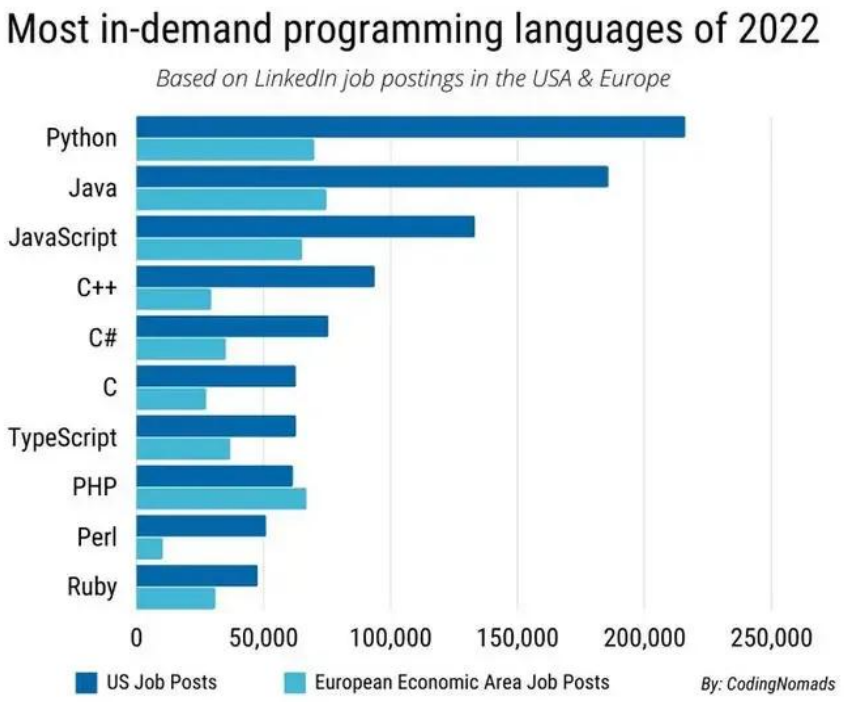
Example: Histogram



The figure above shows a **histogram of exam scores**. Each bar represents a *range of numerical values* (for example, scores between 70 and 80), and the height of the bar indicates how many observations fall within that range. The bars touch because the data is **continuous**—there are no gaps between possible values. This visualization helps answer

questions about the *shape* of the data, such as whether scores are clustered, skewed, or approximately normally distributed.

Example: Bar Chart



The **bar chart** above shows a **bar chart of favorite programming languages** collected from a survey. Each bar corresponds to a *distinct category* (such as Python, Java, or C++), and the height of the bar shows how many respondents selected that category. The bars are separated by gaps to emphasize that the categories are **discrete and unordered**. This visualization is used to compare counts or proportions across categories rather than to show a distribution of values.

Explanation of the Examples

These two figures illustrate the fundamental difference between histograms and bar charts. In the **histogram**, the horizontal axis represents numerical values grouped into intervals, and the visualization emphasizes how data is distributed across a continuous range. In the **bar chart**, the horizontal axis represents distinct categories, and the visualization emphasizes comparison between groups. Although both use rectangular bars, histograms are used to understand the *distribution* of numerical data, while bar

charts are used to *compare categorical quantities*. Confusing the two can lead to incorrect interpretations of data.

2.9 From Concepts to Implementation

As with previous chapters:

- Visualization ideas are introduced conceptually
- Algorithms for constructing plots are discussed informally
- Python libraries are used to implement these ideas efficiently

Behind every visualization function is a well-defined computational process:

- Counting
- Scaling
- Mapping values to visual elements

Understanding these processes helps students interpret plots correctly.

2.10 Visualization as a Thinking Tool

Visualization is not a final step — it is part of the reasoning process.

A typical workflow:

1. Visualize the data
2. Notice patterns or anomalies
3. Ask new questions
4. Refine analysis
5. Visualize again

This iterative process is central to effective data science.

2.11 Implementing Basic Visualizations in Python

In practice, exploratory data analysis involves creating visualizations early and often. While the previous sections described common plot types conceptually, this section demonstrates how those ideas are implemented in Python using standard visualization tools. The goal here is not to master plotting libraries, but to connect visualization concepts to concrete computational actions.

Throughout this section, visualizations are used as *thinking tools* rather than polished presentation graphics.

When it comes to selecting a Python programming environment, there are many to choose from. The standard one available at python.org is a good place to start, or you are interested in an integrated development environment (IDE), the Anaconda system with the Spyder IDE is easy to use and capable, as is PyCharm. There are online environments available at many sites, including the JDoodle site.

A Simple Example Dataset

To keep the focus on interpretation, we consider a small dataset of numerical values representing exam scores:

```
scores = [72, 85, 90, 78, 88, 95, 67, 82, 76, 89]
```

This dataset will be reused across multiple examples to emphasize how different plots answer different questions about the same data.

Histogram: Understanding Distributions

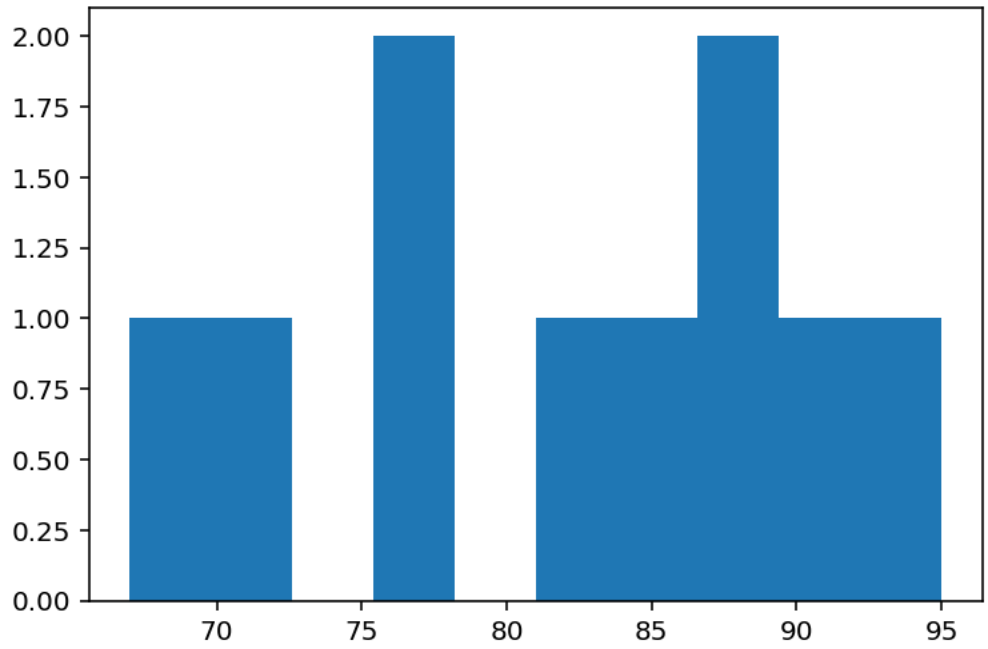
A histogram is used to visualize how numerical values are distributed across a range.

```
import matplotlib.pyplot as plt

scores = [72, 85, 90, 78, 88, 95, 67, 82, 76, 89]

plt.hist(scores)
plt.show()
```

This plot helps answer questions such as:

- Are most values clustered in a particular range?
- Is the distribution symmetric or skewed?
- Are there unusually low or high values?

At this stage, details such as bin size and styling are intentionally kept simple. The focus is on understanding *what the plot reveals*, not how it is customized. Below is the result of the example code.



A key feature of Python and most languages is that they have the ability to access different libraries in order to accomplish various tasks. As the reader may have noticed from earlier examples, the "import" keyword is what tells the Python interpreter to access the given library. In the example above, "import" is paired with "as" so that the programmer does not have to write out the entire program library name every time they want to access one of the plot functions.

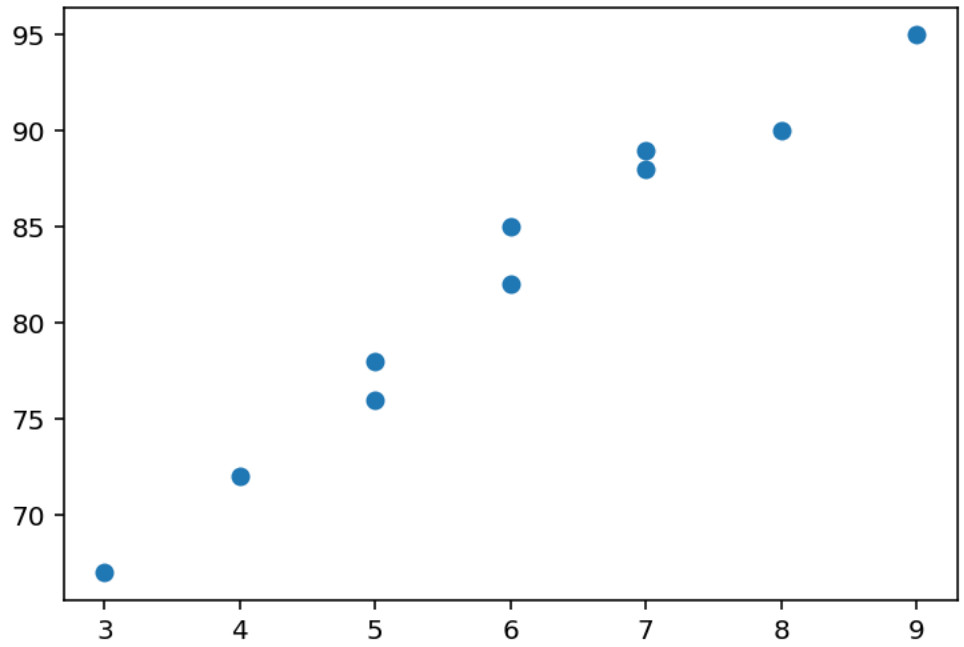
Scatter Plot: Exploring Relationships

Scatter plots are used to examine relationships between two numerical variables. For illustration, suppose we also have the number of hours each student studied:

```
import matplotlib.pyplot as plt
```

```
hours = [4, 6, 8, 5, 7, 9, 3, 6, 5, 7]

plt.scatter(hours, scores)
plt.show()
```

The code above produces the plot below:



This plot encourages questions such as:

- Do higher study hours tend to correspond to higher scores?
- Are there outliers that do not follow the general pattern?

Scatter plots are particularly important later in the course when discussing correlation and regression.

Line Plot: Observing Trends Over Order or Time

Line plots are useful when the order of observations matters.

```
plt.plot(scores)
plt.show()
```

The reader is encouraged to produce the remaining plots as a learning experience.

Bar Chart: Comparing Categories

Bar charts are used to compare categorical data. Suppose exam grades are grouped into letter categories:

```
grades = ["C", "B", "A", "C", "B", "A", "D", "B", "C", "A"]

plt.bar(["A", "B", "C", "D"], [3, 3, 3, 1])
plt.show()
```

This visualization makes it easy to compare categories and identify which outcomes are most common.

Interpreting Visualizations

Regardless of plot type, effective exploratory visualization involves asking:

- What patterns are visible?
- What information is emphasized?
- What information might be hidden?
- What follow-up questions does this raise?

Visualization is rarely a final step. Instead, it guides further analysis and deeper inquiry. In later chapters, visualization will be revisited with greater depth. Additional topics will include:

- Customizing plots
- Visualizing large datasets
- Combining visualizations with statistical analysis

For now, the emphasis remains on using visualizations to understand data and inform reasoning.

2.12 Chapter Summary

- Data consists of observations and features
- Different data types require different analysis techniques
- Data quality issues are common and must be addressed
- Visualization is a core analytical tool, not just presentation
- Exploratory analysis precedes formal modeling

Looking Ahead

In the next chapter, we build on exploratory analysis by introducing statistical tools that quantify patterns, uncertainty, and variability in data.

Chapter 2 Exercises: Data and Data Visualization

Conceptual Exercises

1. Consider the following types of data. For each, identify whether the data is **numerical** or **categorical**, and if numerical, whether it is **continuous** or **discrete**.
    - a) Age in years
    - b) Number of siblings
    - c) Eye color
    - d) Daily temperature
2. Explain why visualization is an important part of exploratory data analysis.
3. A dataset has a mean of 75 and a standard deviation of 5. What does this information tell you about the spread of the data? What does it *not* tell you?

Visualization Selection Exercises

4. For each situation below, choose the most appropriate plot type and briefly explain your choice.
    - a) Distribution of exam scores
    - b) Relationship between study time and exam score

- c) Number of students in each major
- d) Daily temperature over a month

Interpretation Exercises

5. Suppose a histogram of test scores is heavily skewed to the right. What does this indicate about the distribution of scores?
6. A scatter plot shows a strong upward trend but includes one point far from the rest. What is this point called, and why is it important to investigate?

Light Implementation Exercises

7. Given the following dataset of heights (in centimeters):

```
heights = [160, 165, 170, 172, 168, 175, 180, 158, 162, 169]
```

   a) Create a histogram of the data.

   b) Describe the shape of the distribution.

8. Suppose you also have the following weights (in kilograms):

```
weights = [55, 60, 65, 68, 62, 70, 75, 50, 58, 63]
```

   a) Create a scatter plot of height versus weight.

   b) Describe any visible relationship.

Reflection Exercise

9. In a short paragraph, explain how visualization helps guide further data analysis. Include at least one example from the exercises above.

Chapter 3: Statistics for Data Science

Chapter Overview

Statistics provides the language and tools used to describe, summarize, and reason about data. While data visualization offers intuition and qualitative insight, statistics allow us to quantify patterns, variability, and uncertainty.

In this chapter, we introduce core statistical concepts used throughout data science. The emphasis is on interpretation and reasoning rather than formal proofs. Whenever possible, statistical ideas are motivated using visual intuition and concrete examples. Computational implementation follows conceptual understanding, continuing the workflow established in earlier chapters.

3.1 Why Statistics Matters in Data Science

Data science is not only about collecting data, but about making **reliable conclusions** from it.

Statistics helps answer questions such as:

- What is a typical value?
- How much do values vary?
- How unusual is a particular observation?
- How confident can we be in a conclusion?

Without statistics, data analysis risks becoming anecdotal or misleading.

3.2 Measures of Central Tendency

Measures of central tendency describe where data is “centered.”

Mean

The mean (average) is computed by summing all values and dividing by the number of observations.

Conceptually, the mean represents a **balance point** of the data.

Median

The median is the middle value when data is ordered.

The median is:

- Less sensitive to extreme values
- Often a better representation of a "typical" value for skewed data

---

Mode

The mode is the most frequently occurring value.

Modes are particularly useful for categorical data.

3.3 Measures of Variability

While central tendency describes *where* data is centered, variability describes *how spread out* the data is.

Two datasets can share the same mean while behaving very differently.

Variability can be understood visually by analogy. Imagine repeatedly throwing darts at a target. In one case, the darts land tightly clustered around the center; in another, they are widely scattered, even if the average position is the same. Both scenarios may share the same mean, but the second exhibits much greater variability. Variance and standard deviation quantify this spread numerically, capturing how far observations typically lie from the center.

Range

The range is the difference between the maximum and minimum values.

While simple, it is highly sensitive to outliers and provides limited information.

Variance

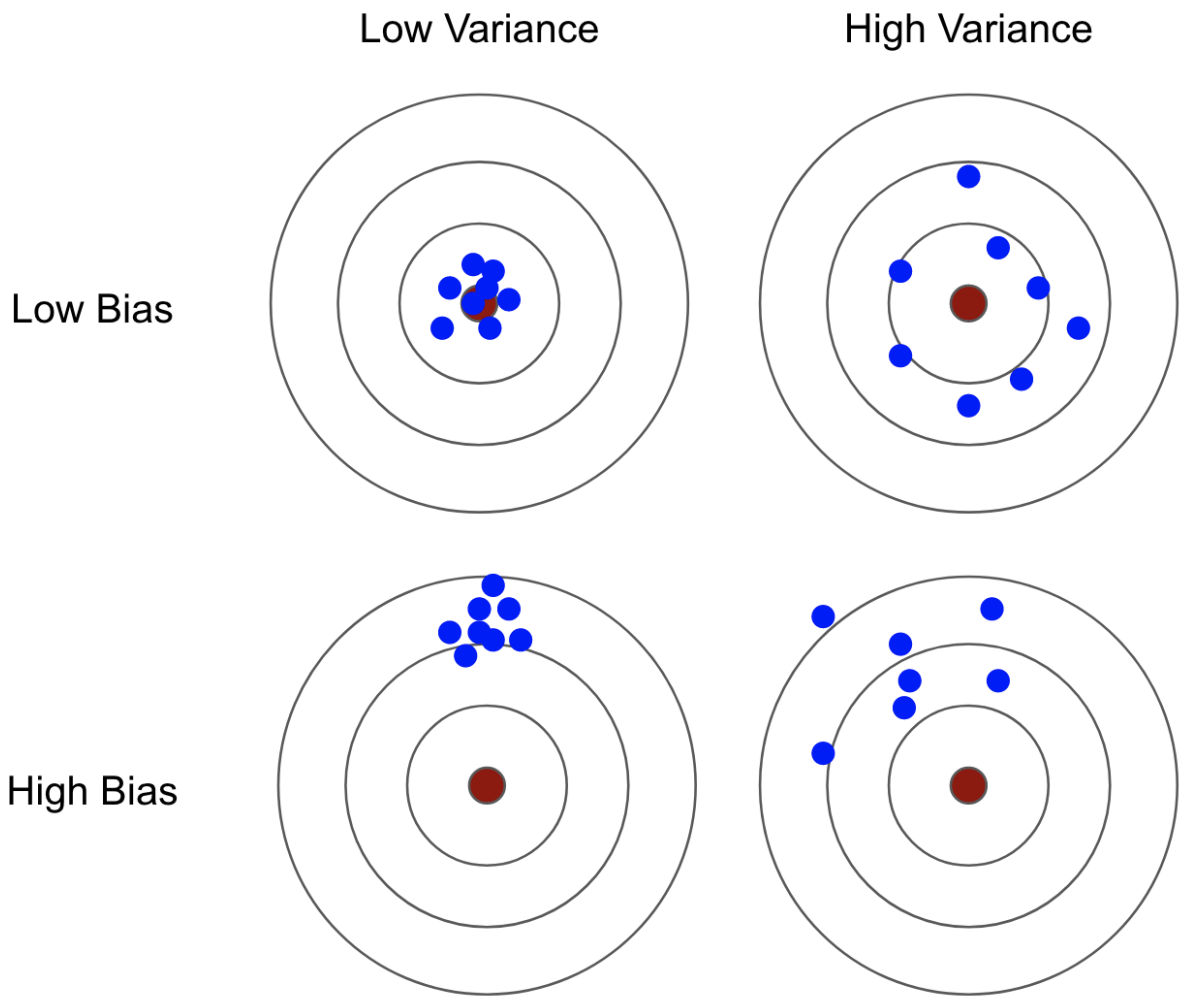
Variance measures the average squared distance of each data point from the mean.

$$\text{Var} = \frac{1}{N}\sum(x_i - \mu)^2$$

Key ideas:

- Each value's deviation from the mean is measured
- Squaring ensures all deviations contribute positively
- Larger deviations contribute more strongly

Variance quantifies **spread**, not direction. Looking at the figure below, the shots in the two upper targets have the about the same average shot location, however, the upper left target has the shots closely centered about the bullseye, while the shot grouping in the upper right target has much more distance from the bullseye. This is an example of low variance vs high variance with the same average. The lower two targets are similar except that the average shot placement is on the high side of the bullseye.



Standard Deviation

The standard deviation is the square root of the variance:

$$\sigma = \sqrt{\frac{1}{N}\sum(x_i - \mu)^2}$$

Taking the square root returns the measure to the **same units as the original data**, making interpretation easier.

Standard deviation answers the question:

On average, how far do values lie from the mean?

3.4 Visualizing Spread and Variability

Statistics and visualization work best together.

Consider two datasets:

- Both have the same mean
- One is tightly clustered
- The other is widely spread

Histograms and scatter plots make these differences visible, while variance and standard deviation quantify them. A good example of this is the previously discussed figure with the four targets.

Visualization helps explain *why* a statistic takes on a particular value.

3.5 Data Distributions

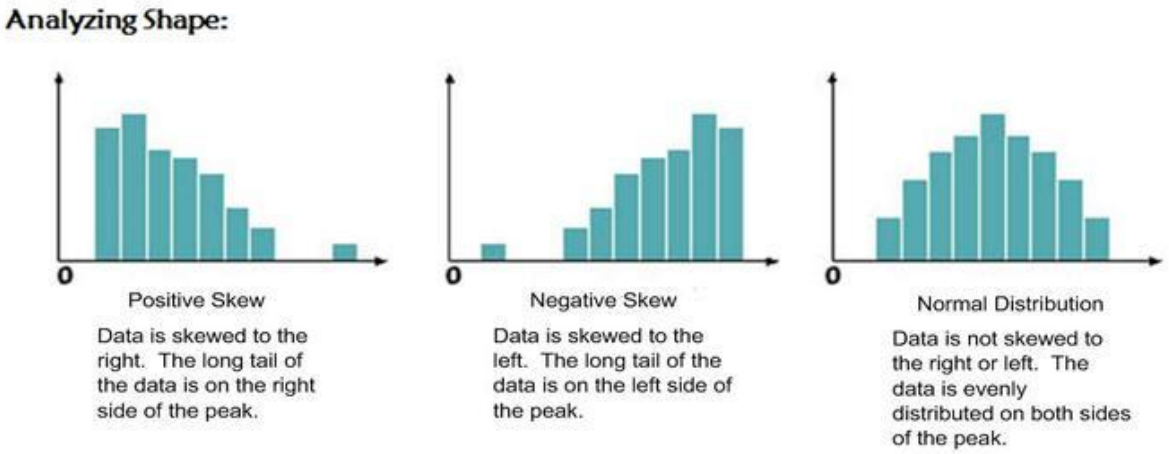
A **distribution** describes how values are spread across possible outcomes.

Understanding distributions helps answer:

- What values are common?
- What values are rare?

- How extreme is an observation?

Histograms provide a visual approximation of distributions. See the figure below [3].



Interpreting the Histogram [4]

With simple measures of central tendency, the shape of the histogram for a set of observations can also be easily interpreted.

If Mean > Median = Right skewed distribution

If Mean < Median = Left skewed distribution

The Normal Distribution

The normal (Gaussian) distribution appears frequently in real-world data.

Characteristics:

- Symmetric about the mean
- Bell-shaped
- Mean, median, and mode coincide

Many natural processes cluster around an average with decreasing frequency away from the center.

3.6 Z-Scores and Standardization

A **z-score** measures how far a value lies from the mean in units of standard deviation:

$$z = \frac{x - \mu}{\sigma}$$

Interpretation:

- $z = 0$: value equals the mean
- $z = 1$: one standard deviation above the mean
- $z = -1$: one standard deviation below the mean

Z-scores allow comparison across datasets with different units or scales. ***This is a very important point.***

3.8 Sampling and Variability

In practice, we often analyze samples rather than entire populations. Imagine if we wanted to know the average height of the students at the University. It would be nearly impossible to get each and every student to come into our lab and measure their height. Instead, of trying to do that we would attempt to get a *representative* sample, and then infer the average height from that sample.

Important ideas:

- Different samples yield different statistics
- Variability arises naturally from sampling
- Larger samples tend to produce more stable estimates

Understanding sampling variability is essential for interpreting results responsibly.

3.9 From Concepts to Computation

As with earlier chapters:

- Statistical ideas are introduced conceptually
- Algorithms are expressed explicitly

- Manual implementations reinforce understanding
- Library functions provide efficiency

Computational tools do not replace statistical reasoning — they support it.

3.10 Statistics as a Foundation for Learning Models

Many learning algorithms rely directly on statistical ideas:

- Means and variances
- Error measures
- Optimization of average loss
- Standardization of inputs

Statistics provides the bridge between raw data and predictive models.

Chapter Summary

- Statistics quantifies patterns observed in data
- Measures of central tendency describe typical values
- Measures of variability describe spread
- Visualization and statistics reinforce each other
- Distributions describe how data values are arranged
- Z-scores allow standardized comparison
- Statistical thinking underpins modern learning algorithms

Chapter 3 Exercises: Describing and Understanding Data

These exercises are designed to reinforce the core ideas of this chapter: central tendency, variability, distributions, and the relationship between numerical summaries and visual representations. Students are encouraged to reason about the data conceptually before relying on computational tools.

Unless otherwise stated, all computations should be implemented **manually using loops** before using library functions for verification.

Dataset A: One-Dimensional Numerical Data

The following dataset represents 100 numerical observations (for example, test scores or measurements):

```
data_1d = [
    52, 55, 57, 58, 60, 61, 62, 63, 64, 65,
    66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
    76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
    86, 87, 88, 89, 90, 91, 92, 93, 94, 95,
    96, 97, 98, 99, 100,
    70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
    80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
    90, 91, 92, 93, 94, 95, 96, 97, 98, 110
]
```

Exercise 3.1: Manual Computation of Basic Statistics

Using data_1d, write Python functions that compute the following **without using NumPy or built-in statistical functions**:

1. Mean
2. Median
3. Mode
4. Variance
5. Standard deviation

Each function should:

- Use explicit loops
- Be clear and readable
- Match the mathematical definitions discussed in the chapter

Exercise 3.2: Interpreting Central Tendency and Spread

Using your computed statistics:

1. Describe what a “typical” value in the dataset looks like.

2. Explain what the variance and standard deviation say about the spread of the data.
3. Identify whether the dataset appears to be skewed:
    - Compare the mean and median
    - State whether the skew is left, right, or approximately symmetric
    - Explain your reasoning

Exercise 3.3: Visualization and Confirmation

Create a histogram of data_1d.

1. Describe the overall shape of the distribution.
2. Identify any visible outliers.
3. Explain how the histogram supports or contradicts your conclusions about skewness from Exercise 3.2.

Exercise 3.4: Library Verification

Using NumPy:

1. Compute the mean, median, variance, and standard deviation.
2. Compare these values to your manual results.
3. Briefly explain why small numerical differences might occur.

Dataset B: Target (Arrow Impact) Data

```
target_hits = [
    (1, 2), (-1, -2), (2, 1), (-2, -1), (0, 1),
    (1, 0), (-1, 0), (0, -1), (2, 2), (-2, -2),
    (3, 1), (-3, -1), (1, 3), (-1, -3), (0, 2),
    (2, 0), (-2, 0), (0, -2), (3, 3), (-3, -3),
    (4, 1), (-4, -1), (1, 4), (-1, -4), (0, 3),
    (3, 0), (-3, 0), (0, -3), (4, 4), (-4, -4),
    (5, 2), (-5, -2), (2, 5), (-2, -5), (0, 4),
    (4, 0), (-4, 0), (0, -4), (5, 5), (-5, -5),
    (6, 3), (-6, -3), (3, 6), (-3, -6), (0, 5),
    (5, 0), (-5, 0), (6, 6), (-6, -6)
]
```

Exercise 3.5: Distance from the Center

1. For each arrow hit, compute the Euclidean distance from the center of the target.
2. Store these distances in a one-dimensional list.
3. Explain what these distances represent conceptually.

Exercise 3.6: Variability as Accuracy

Using the distance data:

1. Compute the mean distance from the center.
2. Compute the variance and standard deviation of the distances.
3. Explain how these quantities relate to:
    - Accuracy
    - Consistency
    - Spread of impacts

Exercise 3.7: Visualization of Target Data

1. Create a scatter plot of the arrow impacts.
2. Create a histogram of the distances from the center.
3. Explain how each visualization conveys different information about performance.

Exercise 3.8: Conceptual Reflection

In a short paragraph:

- Explain how two shooters could have the **same average distance from the center** but very different variances.
- Relate this explanation to the target diagrams discussed in the chapter.

Chapter 3 Takeaway

These exercises demonstrate how:

- Statistical measures summarize data numerically
- Visualization provides essential context
- Manual computation deepens understanding
- Libraries provide efficiency, not insight
- Variability has both numerical and geometric meaning

Chapter 4: Linear Algebra for Data Science

Chapter Overview

Linear algebra provides a compact and powerful language for working with data, transformations, and learning models. While its notation may initially appear abstract, linear algebra simplifies complex relationships by expressing them in a structured and scalable way.

In data science, linear algebra is not studied for its own sake. Instead, it is used to:

- Represent datasets efficiently
- Express relationships between variables
- Describe transformations of data
- Formulate learning models such as regression and neural networks

In this chapter, we introduce the essential ideas of linear algebra using geometric intuition, diagrams, and concrete examples. Computation follows understanding, and abstraction is introduced only when it reduces complexity rather than increasing it.

4.1 Scalars, Vectors, and Matrices

Linear algebra begins with three fundamental objects.

Scalars

A scalar is a single numerical value.

Examples:

- A temperature
- A weight
- A coefficient in an equation

Scalars scale other quantities.

Vectors

A vector is an ordered collection of numbers.

Examples:

- A list of measurements
- A point in space
- A row or column of data

A vector can represent:

- A single observation with multiple features
- A direction and magnitude
- A sequence of related values

Matrices

A matrix is a two-dimensional collection of numbers arranged in rows and columns.

Matrices are used to represent:

- Entire datasets
- Systems of equations
- Transformations applied to data

In data science, datasets are naturally expressed as matrices.

4.2 Vectors as Data Points

Consider a dataset where each observation has multiple features.

For example:

- Height
- Weight
- Age

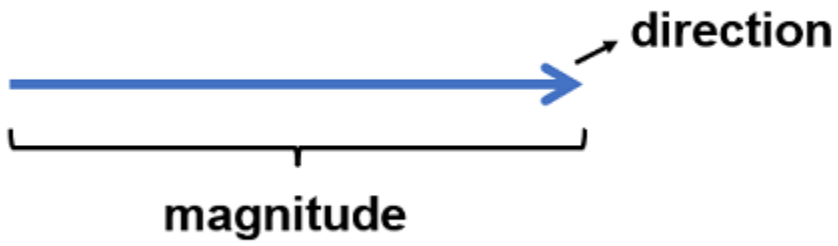
Each observation can be represented as a vector:

$$\mathbf{x} = [x_1, x_2, x_3]$$

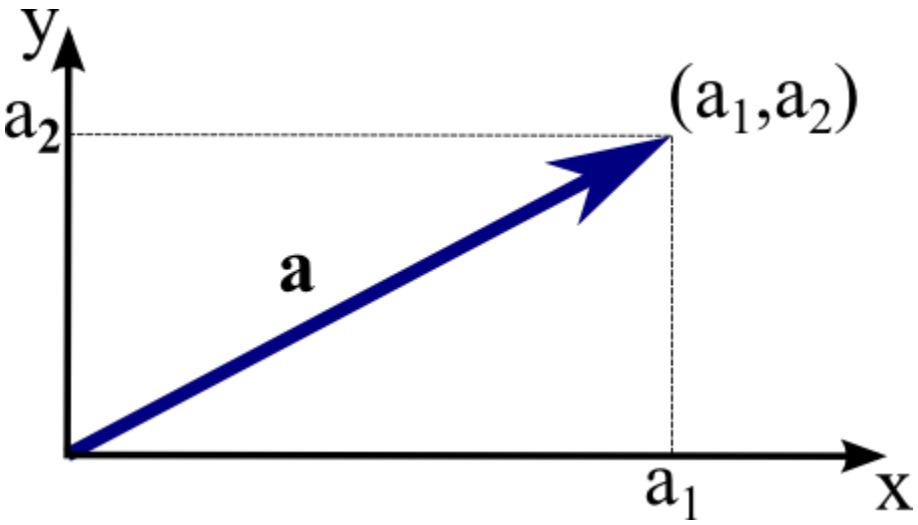
This representation allows:

- Mathematical manipulation
- Geometric interpretation
- Efficient computation

Thinking of data points as vectors is foundational for learning models. The figures below illustrate vectors in 2 dimensions.



In the above figure, we the blue line is the vector, it has a magnitude (its length) and a direction (where it is pointing).



In the figure above the vector 'a' has 2 components, a_1 and a_2.

4.3 Vector Operations and Interpretation

Vector Addition

Adding two vectors combines corresponding components.

Conceptually:

- Combines effects
- Aggregates changes

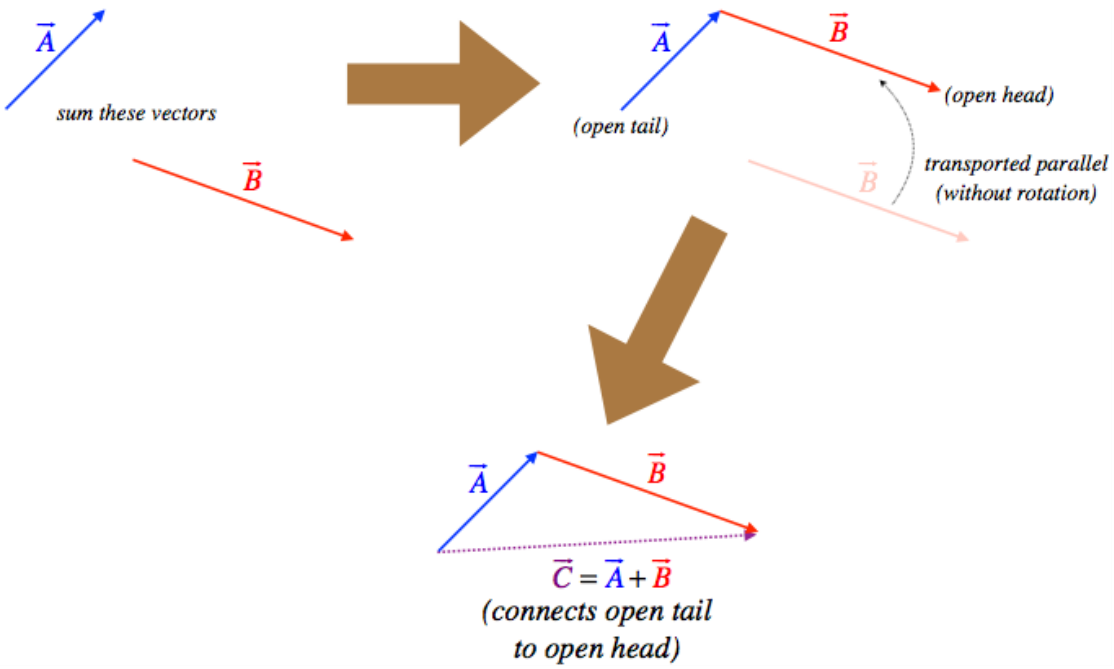
Example: Vector Addition (Numerical)

Consider the two vectors

$$\mathbf{u} = (2, 2) \text{ and } \mathbf{v} = (4, -1)$$

Vector addition is performed **component-wise**:

$$\mathbf{u} + \mathbf{v} = (2 + 4,\ 2 + -1) = (6,\ 1)$$



The figure above illustrates vector addition. Two vectors, A and B, with different magnitudes and directions are summed to result in vector C.

Interpretation

The resulting vector (6, 1) represents the combined effect of the two vectors.

Geometrically:

- Start at the origin
- Move according to u
- From the tip of u, move according to v
- The final position is $\mathrm{u} + \mathrm{v}$

This matches the tip-to-tail diagram you're using.

Key Idea

Vector addition combines corresponding components and represents the cumulative effect of multiple directions or influences.

Scalar Multiplication

Multiplying a vector by a scalar scales its magnitude.

Conceptually:

- Stretching or shrinking
- Adjusting influence or weight

Example: Scalar Multiplication of a Vector

Consider the vector

$$\mathbf{v} = (3, 2)$$

This vector can be interpreted geometrically as an arrow starting at the origin and ending at the point $(3, 2)$.

Now consider multiplying this vector by the scalar 2:

$$2\mathbf{v} = 2(3, 2)$$

Scalar multiplication is performed component-wise:

$$2\mathbf{v} = (2 \cdot 3,\ 2 \cdot 2) = (6,\ 4)$$

Interpretation

Multiplying a vector by a scalar scales its magnitude without changing its direction.

In this example:

- The original vector $(3, 2)$points in a particular direction
- The scaled vector $(6, 4)$points in the *same direction*
- The length of the vector has doubled

Geometrically, scalar multiplication stretches or shrinks a vector while preserving its orientation.

Key Idea

Scalar multiplication changes *how far* a vector reaches, not *where it points*.

Dot Product

The dot product combines two vectors into a single number.

It measures:

- Similarity
- Alignment
- Contribution of one vector along another

In data science, dot products appear everywhere:

- Linear regression
- Neural networks
- Projections
- Distance calculations

Example: Dot Product of Two Vectors (Aligned Directions)

Consider the two two-dimensional vectors

$$\mathrm{u} = (2, 2) \text{ and } \mathrm{v} = (3, 2)$$

Numerical Computation

The dot product is computed by multiplying corresponding components and summing the results:

$$\mathrm{u} \cdot \mathrm{v} = (2)(3) + (2)(2) = 6 + 4 = 10$$

So,

$$\mathrm{u} \cdot \mathrm{v} = 10$$

Interpretation of the Result

Because the dot product is positive and relatively large, the two vectors point in similar directions.

- Both vectors point to the right
- Both vectors point upward
- Their angle is acute (less than $90°$)

The dot product is large when:

- Vectors are long
- Vectors are well aligned

This reflects strong directional agreement.

- points diagonally upward and to the right
- $\mathbf{v} = (3,2)$points in a very similar direction
- The small angle between them explains the positive dot product

Key Idea

The dot product measures **how much one vector points in the direction of another**.

When vectors point in similar directions, the dot product is positive and large. When they are perpendicular, it is zero. When they point in opposite directions, it is negative.

4.4 Geometry and Intuition

Linear algebra is deeply geometric.

Vectors can be visualized as:

- Arrows
- Points in space
- Directions

Key ideas:

- Length (magnitude)
- Angle (relationship)
- Projection (influence)

Geometric intuition makes abstract formulas meaningful.

4.5 Matrices as Datasets

A dataset with:

- $n$observations
- $m$features

can be represented as an $n \times m$matrix.

Each:

- Row → observation
- Column → feature

This representation allows:

- Batch processing
- Compact notation
- Efficient computation across all data points

4.6 Matrix Operations

Matrix Addition

Combines datasets of the same shape.

Used less frequently in data science, but conceptually straightforward.

Example: Addition of Two 2×2 Matrices

Consider the two matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix}$$

Matrix addition is performed **element by element**, meaning that corresponding entries are added together.

Computation

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = \begin{bmatrix} 1+5 & 3+1 \\ 2+0 & 4+2 \end{bmatrix} = \begin{bmatrix} 6 & 4 \\ 2 & 6 \end{bmatrix}$$

Interpretation

Matrix addition combines two matrices of the same shape by adding corresponding entries.

Key points:

- Matrices must have the **same dimensions**
- Addition is **component-wise**

- The result is another matrix of the same size

Key Idea

Matrix addition aggregates corresponding values, much like vector addition extends scalar addition to higher dimensions.

Matrix Multiplication

Matrix multiplication is the most important operation in this chapter.

It:

- Applies transformations
- Combines multiple linear relationships
- Evaluates many equations simultaneously

While the mechanics involve nested loops, the result is a **compressed representation of computation**.

Matrix multiplication is the engine behind:

- Linear regression
- Neural networks
- Dimensionality reduction

Example: Multiplication of Two 2×2Matrices

Let

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \text{and} \mathbf{B} = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix}$$

We compute the matrix product:

$$\mathbf{C} = \mathbf{AB}$$

Matrix Multiplication Rule (Reminder)

Each entry of the product matrix is computed as a **row–column dot product**:

- Take a row from $\mathbf{A}$
- Take a column from $\mathbf{B}$
- Multiply corresponding entries and sum the results

Step-by-Step Computation

Entry C_{11}(row 1 of $\mathbf{A}$, column 1 of $\mathbf{B}$):

$$(1)(5) + (3)(0) = 5$$

Entry C_{12}(row 1 of $\mathbf{A}$, column 2 of $\mathbf{B}$):

$$(1)(1) + (3)(2) = 1 + 6 = 7$$

Entry C_{21}(row 2 of $\mathbf{A}$, column 1 of $\mathbf{B}$):

$$(2)(5) + (4)(0) = 10$$

Entry C_{22}(row 2 of $\mathbf{A}$, column 2 of $\mathbf{B}$):

$$(2)(1) + (4)(2) = 2 + 8 = 10$$

Final Result

$$\mathbf{C} = \begin{bmatrix} 5 & 7 \\ 10 & 10 \end{bmatrix}$$

Interpretation

Unlike matrix addition, matrix multiplication:

- Is **not** performed element-by-element

- Combines rows of the first matrix with columns of the second
- Represents a **linear transformation**

Each entry in the result summarizes multiple interactions between values.

Key Ideas

- Matrix multiplication is built from **dot products**
- The operation encodes many equations at once
- Order matters: $\mathbf{AB} \neq \mathbf{BA}$in general

4.7 Systems of Linear Equations

Many data problems can be expressed as systems of linear equations.

Example:

- Predicting an outcome from multiple features
- Solving for unknown coefficients

Matrix notation allows these systems to be written compactly and solved efficiently.

Example: A System of Linear Equations (Story Problem)

Problem

A small café sells only two items for breakfast:

- Coffee
- Bagels

One morning:

- A customer buys **2 coffees and 1 bagel** for a total of **$7**
- Another customer buys **1 coffee and 3 bagels** for a total of **$11**

Assuming each item has a fixed price, determine the price of one coffee and one bagel.

Step 1: Define Variables

Let:

- x= price of one coffee (in dollars)
- y= price of one bagel (in dollars)

Step 2: Translate the Story into Equations

From the first purchase:

$$2x + y = 7$$

From the second purchase:

$$x + 3y = 11$$

Together, these form a **system of linear equations**:

$$\begin{cases} 2x + y = 7 \\ x + 3y = 11 \end{cases}$$

Step 3: Solve the System

We can solve this system using substitution or elimination. Here we'll use substitution.

From the first equation:

$$y = 7 - 2x$$

Substitute this into the second equation:

$$x + 3(7 - 2x) = 11$$

Simplify:

$$\begin{aligned} x + 21 - 6x &= 11 \\ -5x &= -10 \\ x &= 2 \end{aligned}$$

Now substitute back to find y:

$$y = 7 - 2(2) = 3$$

Step 4: Interpret the Solution

- Price of one coffee: **$2**
- Price of one bagel: **$3**

Why This Is a Linear Algebra Problem

This situation illustrates several key ideas:

- Each equation represents a linear relationship
- The system represents multiple constraints acting together
- The solution satisfies *both equations simultaneously*

Later in the chapter, this same system can be written compactly using matrices:

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 11 \end{bmatrix}$$

This matrix form makes it easier to solve larger systems and connects directly to data science and learning models.

From Substitution to the Matrix Method

The substitution method provides a clear and intuitive way to solve small systems of linear equations. By solving one equation for a variable and substituting it into another, we eliminate unknowns step by step until a solution is found. However, this approach becomes increasingly cumbersome as the number of variables and equations grows.

Linear algebra offers a more systematic and scalable approach through the use of matrices and row operations. Although the notation may look different, row operations perform the same algebraic eliminations carried out in substitution—just in a structured, mechanical form that is well suited for computation. In the next example, we solve the same system using matrix row operations to highlight the connection between these two methods.

Solving the System Using Row Operations

Recall the system:

$$\begin{cases} 2x + y = 7 \\ x + 3y = 11 \end{cases}$$

Step 1: Write the Augmented Matrix

We write the coefficients and constants as an **augmented matrix**:

$$\begin{bmatrix} 2 & 1 & 7 \\ 1 & 3 & 11 \end{bmatrix}$$

Each row corresponds to one equation.

Step 2: Use Row Operations

Our goal is to transform the matrix into a form where the solution is obvious.

Swap rows (optional, but convenient)

Swap Row 1 and Row 2 to get a leading 1:

$$\begin{bmatrix} 1 & 3 & 11 \\ 2 & 1 & 7 \end{bmatrix}$$

Eliminate the x-term in Row 2

Subtract $2 \times$Row 1 from Row 2:

$$R_2 \leftarrow R_2 - 2R_1$$
$$\begin{bmatrix} 1 & 3 & 11 \\ 0 & -5 & -15 \end{bmatrix}$$

Simplify Row 2

Divide Row 2 by -5:

$$R_2 \leftarrow \frac{1}{-5} R_2$$
$$\begin{bmatrix} 1 & 3 & 11 \\ 0 & 1 & 3 \end{bmatrix}$$

Eliminate the y-term in Row 1

Subtract $3 \times$Row 2 from Row 1:

$$R_1 \leftarrow R_1 - 3R_2$$
$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{bmatrix}$$

Step 3: Read the Solution

This final matrix corresponds to:

$$\begin{cases} x = 2 \\ y = 3 \end{cases}$$

Interpretation

Row operations do **not change the solution** of the system — they only rewrite the equations into simpler, equivalent forms.

This process:

- Makes the structure of the solution explicit
- Scales naturally to larger systems
- Is the foundation for how computers solve linear systems

Solving the System Computationally: Manual Code and NumPy

The row operations shown above illustrate how a system of linear equations can be solved systematically by eliminating variables. Computers perform this same process algorithmically. In this section, we first implement a simple manual approach in Python to mirror the row operations, and then use NumPy to perform the same task efficiently.

Manual Python Implementation (Row Operations)

We begin by representing the augmented matrix as a list of lists:

$$\begin{bmatrix} 2 & 1 & 7 \\ 1 & 3 & 11 \end{bmatrix}$$

Python Code

```
# Augmented matrix
M = [
    [2, 1, 7],
    [1, 3, 11]
]

# Swap rows to get a leading 1
M[0], M[1] = M[1], M[0]

# Eliminate x from the second row, by subtracting 2*row 1 from it.
factor = M[1][0]
for j in range(len(M[0])):
    M[1][j] -= factor * M[0][j]

# Normalize the second row.
scale = M[1][1]
for j in range(len(M[1])):
    M[1][j] /= scale

# Eliminate y from the first row
factor = M[0][1]
for j in range(len(M[0])):
    M[0][j] -= factor * M[1][j]

print(M)
```

After these operations, the matrix becomes:

```
[
  [1, 0, 2],
  [0, 1, 3]
]
```

This corresponds to the solution:

$$x = 2, y = 3$$

Interpretation

This manual implementation mirrors the row operations performed algebraically:

- Rows represent equations
- Row operations preserve solutions
- The final matrix encodes the solution directly

While this approach is instructive, it does not scale well to large systems.

NumPy Implementation

NumPy provides optimized routines for solving linear systems.

We first express the system in matrix form:

$$\mathbf{Ax} = \mathbf{b}$$

```
import numpy as np


A = np.array([[2, 1],
              [1, 3]])

b = np.array([7, 11])

x = np.linalg.solve(A, b)
print(x)
```

This produces:

```
[2. 3.]
```

Comparison

- The manual approach makes the elimination process explicit and transparent.
- The NumPy approach performs the same logic internally but is optimized for speed and numerical stability.
- Both methods rely on the same linear algebra principles.

Key Idea

Libraries do not replace understanding. They automate procedures that are already well-defined mathematically.

Why This Matters for Data Science

- Linear regression solves *large* systems this way (implicitly)
- Matrix methods generalize beyond two variables
- Libraries automate row operations, but the logic is the same

---

Key Takeaway

A system of linear equations arises naturally when multiple unknown quantities must satisfy multiple constraints. Linear algebra provides tools to represent and solve these systems efficiently.

4.8 Linear Transformations

A matrix can be viewed as a transformation that:

- Rotates data
- Scales data
- Translates data
- Projects data into new spaces

Understanding transformations helps explain:

- Feature weighting
- Dimensionality reduction
- Learning model behavior

One of the common uses of using matrices in this manner is the transformation matrix in computer graphics. It offers a simple way to combine multiple graphics operations at once during image rendering.

4.9 Linear Algebra in Learning Models

Learning models rely heavily on linear algebra.

Examples:

- Linear regression: weighted sums of features
- ADALINE: optimization of linear combinations
- Neural networks: layered matrix multiplications

Linear algebra allows models to scale from:

- One data point → many
- One feature → many
- One neuron → entire networks

4.10 From Concepts to Computation

As in earlier chapters:

- Linear algebra concepts are introduced intuitively
- Algorithms are built step by step
- Manual implementations reinforce understanding
- Libraries provide efficient execution

Understanding the structure of linear algebra operations is more important than memorizing formulas.

Chapter Summary

- Scalars, vectors, and matrices are the building blocks of linear algebra
- Vectors represent data points and directions
- Matrices represent datasets and transformations
- Dot products measure alignment and contribution
- Matrix multiplication enables scalable computation
- Linear algebra underlies modern learning models

Chapter 4 Exercises: Linear Algebra for Data Science

These exercises reinforce the core ideas of linear algebra introduced in this chapter. Emphasis is placed on understanding vectors, matrices, and systems of linear equations both conceptually and computationally. Unless otherwise stated, each computational task should be completed **first using manual Python code**, followed by a **NumPy-based implementation**.

Exercise 4.1: Vectors as Numerical Objects

Consider the vectors:

$$\mathbf{u} = (2, 2), \mathbf{v} = (3, 2)$$

1. Compute $\mathbf{u} + \mathbf{v}$.
2. Compute $2\mathbf{u}$.
3. Compute $\mathbf{u} \cdot \mathbf{v}$.
4. Explain how the numerical result of the dot product relates to the geometric interpretation of these vectors.

Exercise 4.2: Manual Vector Operations in Python

Using Python lists to represent vectors:

```
u = [2, 2]
v = [3, 2]
```

1. Write code using loops to compute:
    - Vector addition
    - Scalar multiplication
    - Dot product
2. Verify your results manually.

Exercise 4.3: Vector Operations Using NumPy

Repeat Exercise 4.2 using NumPy arrays.

1. Represent the vectors using NumPy.
2. Compute:
    - Vector addition
    - Scalar multiplication
    - Dot product
3. Compare your NumPy results with your manual implementations.

Exercise 4.4: Matrix Addition

Let:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix}$$

1. Compute $\mathbf{A} + \mathbf{B}$by hand.
2. Write Python code using nested loops to compute the sum.

3. Verify the result using NumPy.

Exercise 4.5: Matrix Multiplication

Using the same matrices $\mathbf{A}$and $\mathbf{B}$:

1. Compute $\mathbf{AB}$by hand.
2. Write Python code using nested loops to compute the product.
3. Verify the result using NumPy.
4. Explain why matrix multiplication is fundamentally different from matrix addition.

Exercise 4.6: Systems of Linear Equations (Manual Reasoning)

Solve the following system using **substitution**:

$$\begin{cases} 2x + y = 7 \\ x + 3y = 11 \end{cases}$$

1. Show each step clearly.
2. State the final solution.

Exercise 4.7: Systems of Linear Equations (Row Operations)

Solve the same system using **row operations**.

1. Write the augmented matrix.
2. Perform row operations step by step.
3. Identify the solution from the final matrix.
4. Explain how row operations correspond to algebraic elimination.

Exercise 4.8: Solving Linear Systems in Python

Using the system from Exercises 4.6 and 4.7:

(a) Manual Python Implementation

1. Represent the augmented matrix as a list of lists.
2. Perform row operations using loops to solve the system.
3. Verify that your result matches the algebraic solution.

(b) NumPy Implementation

1. Represent the system in matrix form $\mathbf{Ax} = \mathbf{b}$.
2. Use NumPy to solve for $\mathbf{x}$.
3. Compare the result to your manual solution.

Exercise 4.9: Interpretation and Reflection

In a short paragraph, answer the following:

1. Why is linear algebra a natural language for representing datasets?
2. Why are matrix methods preferred over substitution for large systems?
3. What advantages do numerical libraries provide, and what do they *not* replace?

Optional Challenge Exercises

10. Modify one of the systems above to include a third variable.
11. Solve a 3×3system using row operations.
12. Investigate what happens when a system has:
    - No solution
    - Infinitely many solutions

Chapter 4 Takeaway

These exercises demonstrate how:

- Vectors and matrices encode data and relationships
- Linear algebra organizes multiple equations efficiently
- Manual computation builds understanding
- Libraries automate, but do not replace, reasoning

Chapter 5: Learning from Data

Chapter Overview

Up to this point, we have focused on understanding data: how it is structured, how it can be visualized, how it varies, and how it can be represented mathematically using vectors and matrices. In this chapter, we take the next step: **using data to learn relationships and make predictions**.

Learning models do not begin with neural networks or complex algorithms. At their core, they begin with simple ideas:

- Combining inputs with weights
- Measuring error
- Adjusting parameters to reduce that error

In this chapter, we introduce learning from data through **linear models**, beginning with linear regression and extending naturally to adaptive linear elements (ADALINE). The emphasis is on intuition, geometry, and computation rather than formal proofs.

5.1 What Does It Mean to "Learn" from Data?

In data science, a model is said to *learn* when it improves its performance on a task by adjusting internal parameters based on data.

Learning typically involves:

- Inputs (features)
- Outputs (targets)
- A model that connects inputs to outputs
- A measure of error
- A method for reducing that error

This process is iterative and data-driven.

5.2 Linear Models

The simplest learning models are **linear models**.

A linear model predicts an output as a weighted sum of inputs:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Here:

- x_i are input features
- w_i are weights
- b is a bias (intercept)
- $\hat{y}$ is the predicted output

Linear models are simple, interpretable, and surprisingly powerful.

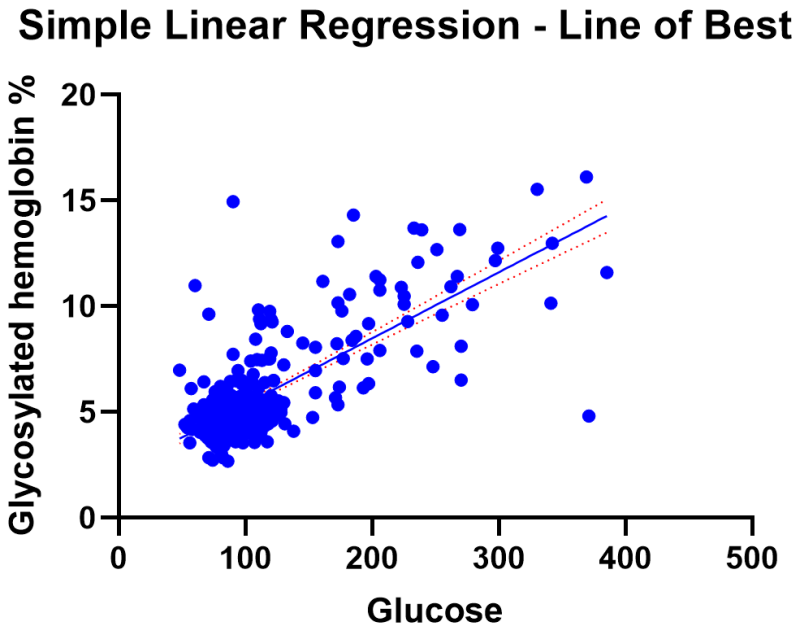
5.3 Linear Regression

Linear regression is a method for finding the best-fitting line (or hyperplane) through data.

For a single input variable:

$$\hat{y} = mx + b$$

The goal of linear regression is to choose m and b so that the predicted values are as close as possible to the observed data. The figure below shows a simple linear regression.



5.4 Error and Loss

To learn from data, we must quantify how wrong a model is.

A common error measure is the **squared error**:

$$\text{error} = (y - \hat{y})^2$$

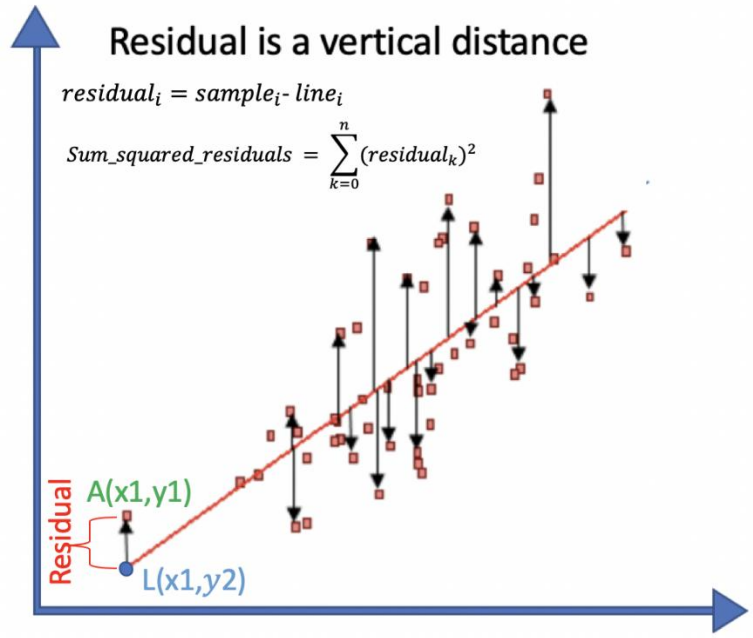
- In the above example, y is the desired value, and $\hat{y}$ is the output of the model.

For an entire dataset, we typically measure **average error**, often called a **loss function**.

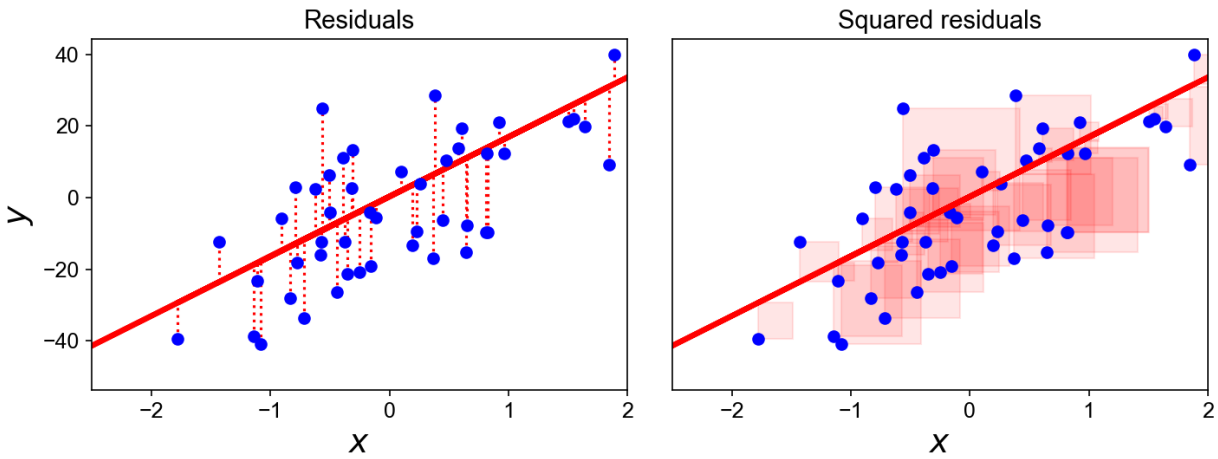
Squared error is used because:

- It penalizes large errors more strongly
- It is mathematically convenient
- It connects naturally to variance and standard deviation

Often times the error is referred to as a residual. We square the error so that negative error does not cancel out positive. As stated earlier, in the variance formula discussion, we could take the absolute value and end up with something a bit more intuitive, however, often times squaring is more computationally efficient, and mathematically useful. The figure below illustrates residuals on a regression line.



The next figure illustrates the squaring of the residuals. This is a geometric explanation of what is happening; by squaring the residuals, they become an area, always a positive quantity.



Numerical Example: Computing a Regression Line

Suppose we collect the following data, which appears to follow a roughly linear relationship:

```
x:  0   1   2   3   4   5   6   7   8   9
y:  3.1 3.6 4.0 4.4 5.0 5.4 6.0 6.5 7.1 7.4
```

A scatter plot of this data suggests a line of the form:

$$y \approx \frac{1}{2}x + 3$$

Our goal is to **compute** the best-fitting line:

$$\hat{y} = mx + b$$

Step 1: What Does "Best-Fitting" Mean?

In linear regression, the best-fitting line is defined as the one that **minimizes the total squared error**:

$$\text{Loss}(m, b) = \sum_{i=1}^{N} (y_i - (mx_i + b))^2$$

This loss function measures how far the predicted values are from the observed data.

The formulas used below are obtained by taking partial derivatives of this loss function with respect to $m$and b, setting them equal to zero, and solving the resulting equations.

We will **use the result** of this process without performing the calculus.

Step 2: Regression Formulas

The slope $m$and intercept $b$are given by:

$$m = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{N\sum x_i^2 - (\sum x_i)^2}$$
$$b = \bar{y} - m\bar{x}$$

These formulas guarantee the minimum squared error.

Step 3: Compute Required Quantities

From the data:

- $N = 10$
- $\sum x = 45$
- $\sum y = 51.5$
- $\sum x^2 = 285$
- $\sum xy = 303.5$

Step 4: Compute the Slope

$$m = \frac{10(303.5) - (45)(51.5)}{10(285) - 45^2}$$
$$m = \frac{3035 - 2317.5}{2850 - 2025}$$

$$m = \frac{717.5}{825} \approx 0.87$$

(Note: small datasets and noise affect estimates; this will be refined later.)

Step 5: Compute the Intercept

$$\bar{x} = 4.5, \bar{y} = 5.15$$
$$b = 5.15 - (0.87)(4.5) \approx 1.24$$

Step 6: Final Regression Line

$$\hat{y} = 0.87x + 1.24$$

Interpretation

- The slope indicates how much $y$changes per unit increase in x
- The intercept represents the predicted value when $x = 0$
- Noise and limited data affect parameter estimates
- With more data, estimates approach the true relationship

Key Takeaways

- Linear regression finds parameters by **minimizing squared error**
- The formulas for $m$and $b$come from calculus, even if calculus is not shown
- This method connects directly to:
    - Gradient descent
    - ADALINE
    - Neural network learning

5.5 Learning as Optimization

Learning can be viewed as an optimization problem:

- Parameters are adjusted
- Error is reduced
- A minimum is sought

For linear regression, the error surface is smooth and well-behaved, making it an ideal starting point for understanding learning algorithms.

5.6 Gradient Descent (Conceptual)

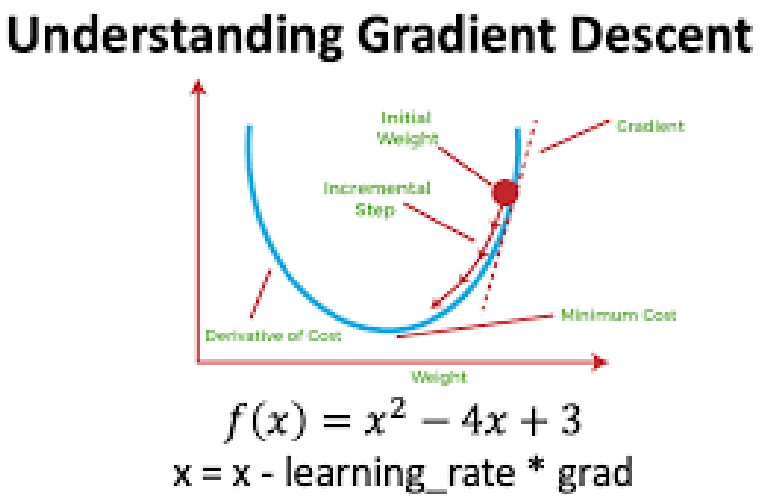
Gradient descent is a method for minimizing error by taking small steps in the direction that most reduces the loss.

Conceptually:

1. Make a guess for the parameters
2. Measure the error
3. Adjust parameters to reduce error
4. Repeat

The direction of adjustment is determined by the **gradient**, which indicates how sensitive the error is to changes in each parameter.

The figure below illustrates the process of gradient decent.



Gradient Descent Analogy

Imagine standing somewhere on a smooth hillside at night, trying to reach the lowest point of a valley. You cannot see the entire landscape, but you *can* feel which direction slopes downward beneath your feet.

Gradient descent works the same way.

- Your **current position** represents a particular choice of model parameters (such as weights and bias).
- The **height of the land** represents how much error the model is making.
- The **bottom of the valley** represents the smallest possible error.

At each step:

1. You check which direction slopes downward the most.
2. You take a small step in that direction.
3. You repeat the process until you can no longer go downhill.

Each step reduces error slightly. Over time, these small improvements add up, and the model settles near the lowest point.

Why the Steps Are Small

If the steps are too large:

- You might overshoot the bottom
- You might bounce back and forth

- Learning becomes unstable

If the steps are too small:

- Learning is very slow
- It takes many iterations to improve

The **learning rate** controls step size and balances speed with stability.

Key Ideas

Gradient descent is not guessing.
It is **systematic improvement**.

The model:

- Makes a prediction
- Measures how wrong it is
- Adjusts itself to be *slightly less wrong*

And then repeats.

5.7 Adaptive Linear Elements (ADALINE)

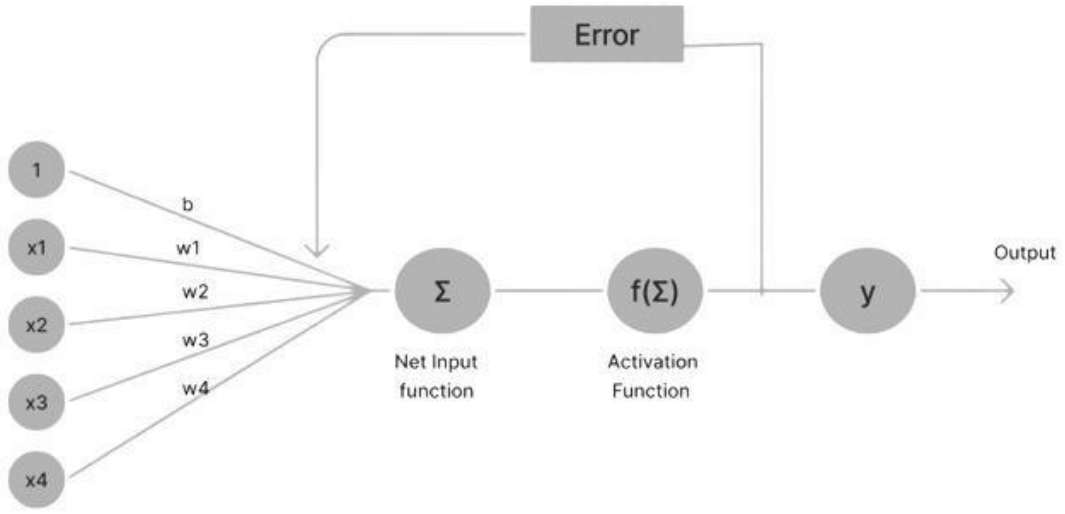
An **Adaptive Linear Element (ADALINE)** is a linear neuron that learns using gradient descent on squared error.

It consists of:

- Inputs
- Weights
- A linear combination
- An error-driven update rule

ADALINE differs from earlier models (such as the perceptron) in that it minimizes a continuous error function rather than making discrete decisions.

Below is a diagram of an ADALINE.



5.8 ADALINE Learning Rule (Intuition)

The ADALINE learning rule updates weights according to:

- The size of the error
- The size of the input
- A learning rate that controls step size

Intuitively:

- Large errors lead to larger adjustments
- Inputs with larger values influence the update more strongly
- Small learning rates lead to gradual, stable learning

This ties directly back to:

- Dot products
- Vector updates
- Matrix operations

5.9 Linear Algebra View of Learning

Using vector notation, a linear model can be written compactly as:

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$$

Learning becomes:

- Adjusting $\mathbf{w}$
- Reducing average squared error
- Repeating over the dataset

This perspective shows why linear algebra is the natural language of learning models.

5.10 From Linear Models to Neural Networks

A neural network can be viewed as:

- Many linear models
- Stacked together
- With nonlinear functions applied between layers

Understanding linear regression and ADALINE provides:

- Conceptual grounding
- Mathematical clarity
- Confidence moving forward

Neural networks extend these ideas; they do not replace them.

Chapter Summary

- Learning from data involves adjusting parameters to reduce error
- Linear models form the foundation of many learning algorithms
- Linear regression finds relationships by minimizing squared error
- Gradient descent provides a general learning mechanism
- ADALINE is a simple but powerful adaptive model
- Linear algebra unifies learning models under a common framework

---

Looking Ahead

In the next chapter, we will implement linear regression and ADALINE step by step, beginning with manual computations and progressing to efficient numerical implementations.

Chapter 5 Review Questions and Exercises: Learning from Data

These questions and exercises are designed to reinforce the core ideas of learning from data, linear models, error, and optimization. Emphasis is placed on interpretation and reasoning rather than detailed computation.

Review Questions

1. In your own words, what does it mean for a model to "learn" from data?
2. What are the key components of a learning system?
   (Hint: inputs, outputs, parameters, error, adjustment.)
3. Why are linear models a natural starting point for learning from data?
4. Explain the role of the weights and the bias in a linear model.
5. What does the predicted value $\hat{y}$ represent?

Conceptual Questions

6. Why is squared error commonly used as a loss function instead of absolute error?
7. How is the idea of squared error related to variance and standard deviation from earlier chapters?
8. Explain why learning can be viewed as an optimization problem.
9. What does the gradient tell us about a loss function?
10. Why is gradient descent an *iterative* process?

Visualization and Interpretation

11. In a linear regression plot:

- What do the data points represent?
- What does the fitted line represent?
- What do the vertical distances between points and the line represent?

12. In the gradient descent "bowl" diagram:
    - What does the horizontal position represent?
    - What does the height represent?
    - What does moving downhill correspond to?

ADALINE and Linear Neurons

13. What is an ADALINE, and why is it described as a *linear* learning model?
14. Why is ADALINE said to minimize a *continuous* error function?
15. How does the dot product appear naturally in the operation of a linear neuron?

Short Exercises

16. Consider the linear model:

$$\hat{y} = 2x + 1$$

For the input values $x = 0,1,2$:

- Compute the predicted values $\hat{y}$
- Suppose the true values are $y = 1,2,2$.
  Compute the squared error for each point.

17. Suppose a model's predictions are consistently too large.
    - Should the parameters be increased or decreased?
    - How does gradient descent help determine the direction of change?

18. Residuals and Squared Error

Consider the following dataset of 10 points:

```
x:  0   1   2   3   4   5   6   7   8   9
y:  3.2 3.6 4.1 4.5 5.0 5.4 6.1 6.4 7.0 7.5
```

Suppose we use the linear model:

$$\hat{y} = 0.5x + 3$$

Part A: Compute Predictions

1. For each value of x, compute the predicted value $\hat{y}$.
2. Create a table showing:
    - x
    - y
    - $\hat{y}$

Part B: Compute Residuals

3. For each data point, compute the residual:

$$\text{residual} = y - \hat{y}$$

4. Indicate whether each residual is:
    - Positive (model underestimates)
    - Negative (model overestimates)

Part C: Squared Error

5. Compute the squared residual for each data point.
6. Compute the **sum of squared residuals**.

Part D: Interpretation

7. Based on the residuals:

- Does the model tend to overestimate or underestimate the data?
- Are the residuals roughly balanced around zero?

8. Explain why a different choice of $m$and $b$might reduce the total squared error.

Reflection (Short Answer)

9. How do residuals connect the geometric idea of "distance from the line" to the statistical idea of minimizing squared error?

Reflection Questions

18. Why is it useful to understand learning models conceptually before implementing them in code?
19. How does linear algebra help simplify the description of learning models?
20. In a short paragraph, explain how Chapters 3, 4, and 5 fit together.

Chapter 5 Takeaway

These questions reinforce that:

- Learning is about reducing error
- Linear models form the foundation of more complex systems
- Optimization connects statistics, geometry, and computation
- Understanding precedes implementation

Chapter 6: Implementing Linear Models

Chapter Overview

In the previous chapter, we introduced learning from data conceptually, focusing on linear models, error, and optimization. In this chapter, we turn those ideas into working algorithms. We begin by implementing linear regression and adaptive linear elements (ADALINE) using explicit, step-by-step Python code. Only after the underlying logic is clear do we introduce library-based implementations.

The goal of this chapter is not speed or sophistication, but **understanding**. By the end of the chapter, students should be able to explain exactly what each line of code is doing and how it relates to the mathematical and conceptual ideas introduced earlier.

6.1 From Equations to Algorithms

Recall the linear model:

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$$

To implement this model, we must:

1. Compute a weighted sum (dot product)
2. Add a bias
3. Compare the prediction to the true value
4. Measure the error
5. Adjust the parameters

Each of these steps can be translated directly into code.

6.2 A Simple Linear Regression Model (One Variable)

We begin with the simplest case: one input variable.

$$\hat{y} = mx + b$$

Here:

- m is the slope
- b is the intercept

6.3 Manual Implementation: Prediction and Error

Consider a small dataset:

```
x_data = [1, 2, 3, 4]
y_data = [2, 4, 5, 7]
```

Prediction Function

```
def predict(x, m, b):
    return m * x + b
```

Squared Error

```
def squared_error(y, y_hat):
    return (y - y_hat) ** 2
```

These functions implement the core components of a learning model.

6.4 Average Loss over a Dataset

Learning depends on performance over *all* data points, not just one.

```
def mean_squared_error(x_data, y_data, m, b):
    total_error = 0
    for x, y in zip(x_data, y_data):
        y_hat = predict(x, m, b)
        total_error += squared_error(y, y_hat)
    return total_error / len(x_data)
```

This function directly reflects the loss functions discussed in Chapter 5.

Note the use of the “zip” function. This function is very handy as it acts to pair up the x_data and y_data arrays. Without it some sort of loop function with a counter would be needed. Two examples below illustrate the utility of the zip function.

```
for x, y in zip(x_data, y_data):
    print(x, y)
```

verses:

```
for j in range(len(x_data)):
    print(x_data[j], y_data[j])
```

6.5 Manual Gradient Descent (One Variable)

To learn, we adjust parameters to reduce error.

Gradient Update (Conceptual)

- If predictions are too large, reduce parameters
- If predictions are too small, increase parameters

A simple numerical gradient approximation can be used to illustrate the idea without calculus.

```
def gradient_descent_step(x_data, y_data, m, b, learning_rate):
    # Initialize values.
    m_grad = 0
    b_grad = 0

    # For each value in the data find y_hat and the error associated with it.
    for x, y in zip(x_data, y_data):
        # y_hat
        y_hat = predict(x, m, b)
        # error
        error = y_hat - y
        # Sum error direction, scaled by input for slope correction.
        m_grad += error * x
        # Sum error for intercept.
        b_grad += error

    # Calculate new values of m,b, scaled by learning rate and data set size.
    m -= learning_rate * (m_grad / len(x_data))
    b -= learning_rate * (b_grad / len(x_data))

    return m, b
```

This above code is a bit over-commented but it mirrors the ADALINE learning rule in its simplest form.

6.6 Iterative Learning

Learning occurs over many small updates.

```
m, b = 0, 0
learning_rate = 0.01

for epoch in range(100):
    m, b = gradient_descent_step(x_data, y_data, m, b, learning_rate)

print("m = ", m, " b = ", b)
```

Each iteration slightly reduces the overall error.

Putting it all together, here is the code:

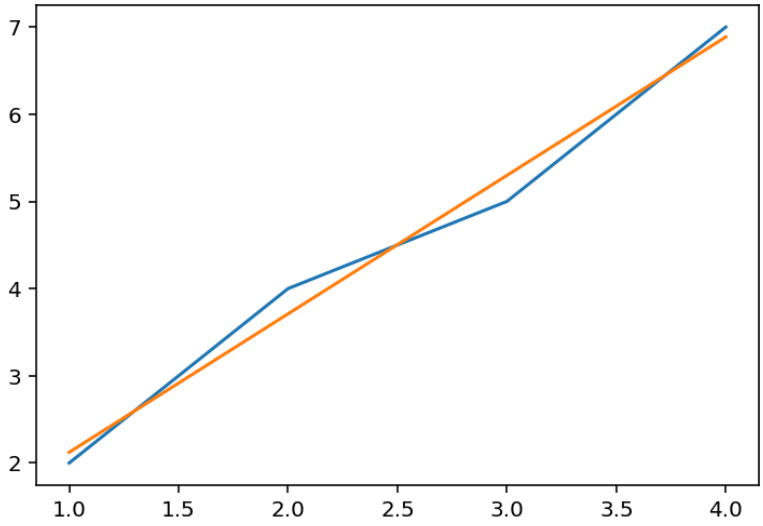
```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a simple Adeline test.
"""

def predict(x, m, b):
    return m * x + b

def squared_error(y, y_hat):
    return (y - y_hat) ** 2

def mean_squared_error(x_data, y_data, m, b):
    total_error = 0
    for x, y in zip(x_data, y_data):
        y_hat = predict(x, m, b)
        total_error += squared_error(y, y_hat)
    return total_error / len(x_data)


def gradient_descent_step(x_data, y_data, m, b, learning_rate):
    # Initialize values.
    m_grad = 0
    b_grad = 0

    # For each value in the data set find y_hat and the error associated with
it.
    for x, y in zip(x_data, y_data):
        # y_hat
        y_hat = predict(x, m, b)
        # error
        error = y_hat - y
        # Sum error direction, scaled by input for slope correction.
        m_grad += error * x
        # Sum error for intercept.
```

```
        b_grad += error

    # Calculate new values of m and b, scaled by learning rate and data set
size.
    m -= learning_rate * (m_grad / len(x_data))
    b -= learning_rate * (b_grad / len(x_data))

    return m, b


x_data = [1, 2, 3, 4]
y_data = [2, 4, 5, 7]

m, b = 0, 0
learning_rate = 0.01

for epoch in range(100):
    m, b = gradient_descent_step(x_data, y_data, m, b, learning_rate)

print("m = ", m, " b = ", b)
```

The results are:

```
m =  1.5881421952398438  b =  0.5340133193805199
```

Checking the results and plotting results in the following:



In the above plot, the blue line is actual data, the orange line is the estimated data. The same/similar results could be found using regression.

6.7 Extending to Multiple Inputs (ADALINE)

With multiple inputs, we use vectors.

Prediction Using a Dot Product

```
def predict_vector(x, w, b):
    total = 0
    for xi, wi in zip(x, w):
        total += xi * wi
    return total + b
```

Here:

- x is a feature vector (inputs)
- w is a weight vector

This is the computational form of:

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$$

6.8 Manual ADALINE Learning Rule

Single input Adaline.

```
# Test basic Adeline routines.
x_data = [1, 2, 3, 4]
y_data = [2, 4, 5, 7]

w, b = 0, 0
learning_rate = 0.01

def adaline_step_1d(x_data, y_data, w, b, lr):
    # w is a single float for the single feature
    w_grad = 0.0
    b_grad = 0.0
    n = len(x_data)

    for i in range(n):
        x = x_data[i]
        y = y_data[i]

        # Calculate predicted y.
        y_hat =(w * x) + b
        error = y_hat - y

        # Calculate error.
        w_grad += error * x
        b_grad += error
```

```
    w -= lr * (w_grad / n)
    b -= lr * (b_grad / n)
    return w, b

for epoch in range(100):
    w, b = adaline_step_1d(x_data, y_data, w, b, learning_rate)

print("adeline_step_1d, w,b ", w, b)
```

This code is a direct computational expression of the ADALINE update rule.

Mulit-Dimension inputs:

```
def dot(x, w):
    total = 0.0
    for i in range(len(w)):
        total += x[i] * w[i]
    return total

def adaline_step(X, y_data, w, b, lr):
    n = len(X)
    m = len(w)
    w_grad = [0.0] * m
    b_grad = 0.0

    for k in range(n):
        x = X[k]
        y = y_data[k]
        y_hat = dot(x, w) + b
        error = y_hat - y

        for i in range(m):
            w_grad[i] += error * x[i]
        b_grad += error

    for i in range(m):
        w[i] -= lr * (w_grad[i] / n)
    b -= lr * (b_grad / n)

    return w, b

# Each input is a vector.
x_data = [[1], [2], [3], [4]]
y_data = [2, 4, 5, 7]

# The weights have to be a vector.
w = [0]
b = 0.0
learning_rate = 0.01
```

```
for epoch in range(100):
    w, b = adaline_step(x_data, y_data, w, b, learning_rate)

print("adeline_step (general multi input), w,b ", w, b)
```

6.9 NumPy Implementation

Once the logic is understood, NumPy simplifies and accelerates computation.

```
import numpy as np

X = np.array([[1],
              [2],
              [3],
              [4]])

w = np.array([0.0])

b = 0

num_epochs = 100

for epoch in range(num_epochs):
    y_hat = X @ w + b
    errors = y_hat - y_data

    w -= learning_rate * (X.T @ errors) / len(X)
    b -= learning_rate * errors.mean()

print("numpy way, w, b = ", w, b)
```

This code performs the same operations as the manual version, but in a compact and efficient form.

Why X @ w Works

This line:

```
y_hat = X @ w + b
```

is doing exactly this math:

$$\hat{y} = Xw + b$$

- X @ w → dot product of each row of X with w

- Result shape: (N,)
- Adding b broadcasts the bias to every prediction

So NumPy is performing **all dot products at once**, which is why it feels so clean.

Gradient Update (What the Math Is)

This line:

```
w -= lr * (X.T @ errors) / len(X)
```

corresponds to:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{N} X^T (\hat{\mathbf{y}} - \mathbf{y})$$

Which is the vector form of the ADALINE update rule.

6.10 A Brief Look at Library-Based Implementations (scikit-learn)

Up to this point, we have implemented linear learning models explicitly in Python in order to understand how predictions are made, how error is measured, and how parameters are updated through repeated iterations. While this manual approach is essential for learning, most real-world data science work relies on well-tested libraries that implement these same ideas efficiently.

In this section, we briefly introduce scikit-learn to show how the concepts developed in this chapter appear in a standard, widely used tool.

The goal is not to replace our manual implementations, but to recognize them.

Linear Regression Using scikit-learn

We begin with the same type of data used earlier in the chapter:

```
import numpy as np
```

```
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 4, 5, 7])
```

In scikit-learn, data is represented as:

- A data matrix X, where each row is one input vector
- A target vector y, containing the corresponding outputs

This is the same representation we used in our NumPy-based implementation.

Fitting the Model

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)
```

Calling fit() performs the learning process. Internally, scikit-learn:

- Computes predictions
- Measures error
- Adjusts parameters to minimize squared error

Although these steps are not shown explicitly, they are the same operations we implemented manually.

Inspecting the Learned Parameters

```
print("model.coef = ", model.coef_)
print("model.intercept_ = ", model.intercept_)
```

- coef_ corresponds to the weight vector
- intercept_ corresponds to the bias

These values play the same roles as the parameters mmm, w\mathbf{w}w, and bbb in our earlier implementations.

Making Predictions

```
y_hat = model.predict(X)
print(y_hat)
```

This produces predicted values using the learned model:

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$$

Again, the computation mirrors the prediction functions written earlier in the chapter.

Here is the code all together:

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4]])
y = np.array([2, 4, 5, 7])

model = LinearRegression()
model.fit(X, y)

print("model.coef = ", model.coef_)
print("model.intercept_ = ", model.intercept_)
```

Again, notice:

- fit() is doing gradient-based optimization internally
- coef_ corresponds to weights
- intercept_ corresponds to bias
- The math is the same as what students already implemented

Connecting Back to the Manual Implementation

The scikit-learn implementation compresses the learning process into a small number of method calls, but nothing fundamentally new has been introduced.

- The loop still exists internally
- Gradient-based optimization is still occurring

- Squared error is still being minimized

The difference is that these steps are handled automatically and efficiently.

Key Takeaway

scikit-learn does not change how linear models work.
It provides a reliable and optimized way to apply ideas you already understand.

Understanding the underlying algorithm makes it easier to:

- Use libraries correctly
- Interpret results responsibly
- Debug unexpected behavior

Looking Ahead

In the next chapter, we will extend these ideas by introducing simple neural network structures. While the models will become more complex, they will still rely on the same core principles introduced here: linear combinations, error measurement, and iterative learning.

6.11 Comparing Manual and Library Approaches

- Manual code exposes every step of the algorithm
- NumPy code emphasizes mathematical structure
- Both implement the same learning process
- Libraries do not replace understanding.
  They package it.

Understanding both is essential for effective data science.

Chapter Summary

- Linear models can be implemented directly from equations
- Prediction, error, and learning follow a clear computational pattern
- Gradient descent adjusts parameters iteratively
- ADALINE generalizes linear regression to multiple inputs
- Manual implementations build understanding
- NumPy implementations provide efficiency and clarity

Looking Ahead

In the next chapter, we extend these ideas by introducing **nonlinearity and simple feedforward neural networks**, building directly on the linear foundations established here.

Chapter 6 Exercises: Implementing Linear Models

These exercises reinforce the implementation of linear models and learning algorithms introduced in this chapter. Emphasis is placed on understanding the relationship between equations, algorithms, and library-based implementations.

Exercise 6.1: Manual Prediction and Error

Consider the dataset:

x: 1 2 3 4

y: 2 4 5 7

Suppose the model is:

$$\hat{y} = 1.5x + 0.5$$

1. Compute the predicted value y^ for each x.
2. Compute the residual for each data point.
3. Compute the squared error for each data point.

4. Compute the mean squared error.

Exercise 6.2: Manual Gradient Descent (One Variable)

Using the same dataset:

1. Write a Python function that computes the mean squared error for given values of m and b.
2. Starting with m=0 and b=0, perform **one** gradient descent update step manually.
3. Explain how the update moves the parameters closer to the data.

Exercise 6.3: Iterative Learning

Using your gradient descent function:

1. Run the learning process for 100 iterations.
2. Print the values of m and b every 20 iterations.
3. Describe how the parameters change over time.

Exercise 6.4: Manual ADALINE (Multiple Inputs)

Consider the dataset:

X = [[1, 2],

[2, 3],

[3, 4],

[4, 5]]

y = [5, 7, 9, 11]

1. Initialize the weight vector and bias to zero.
2. Implement the ADALINE update rule using explicit loops.
3. Run the learning process for several iterations.

4. Report the final weights and bias.

Exercise 6.5: NumPy Implementation

Repeat Exercise 6.4 using NumPy arrays.

1. Express the prediction as a matrix–vector operation.
2. Implement the weight and bias updates using NumPy.
3. Verify that the learned parameters are similar to the manual implementation.

Exercise 6.6: Comparing Manual and NumPy Approaches

Answer the following questions:

1. Which parts of the code became simpler when using NumPy?
2. Which ideas remained unchanged?
3. Why is NumPy preferred for large datasets?

Exercise 6.7: Library-Based Implementation (scikit-learn)

Using **scikit-learn**:

1. Fit a linear regression model to the dataset from Exercise 6.4.
2. Extract the learned coefficients and intercept.
3. Compare these values to the parameters obtained using your manual and NumPy implementations.
4. Explain any small differences.

Exercise 6.8: Interpretation and Reflection

In a short paragraph, answer the following:

1. Why is it important to understand how learning algorithms work before using libraries?

2. How does the learning loop appear differently in manual code, NumPy code, and library-based code?
3. What role does linear algebra play in simplifying learning algorithms?

Optional Challenge Exercises

9. Modify the learning rate and observe how it affects convergence.
10. Add noise to the dataset and observe how the learned parameters change.
11. Experiment with stopping the learning process early.

Chapter 6 Takeaway

These exercises demonstrate that:

- Learning algorithms can be implemented directly from equations
- Iterative parameter updates drive learning
- Libraries automate well-understood procedures
- Understanding the process leads to better use of tools

Chapter 7: Course Projects

The following projects are designed to reinforce core concepts from this course through applied problem-solving. Each project emphasizes understanding, reasoning, and clear implementation rather than the use of advanced libraries or black-box tools.

Unless otherwise stated, students are expected to:

- Write clear, well-documented Python code
- Use visualization where appropriate
- Explain their reasoning in plain language

Project 1: Statistical Frequency Analysis and Message Decoding

Objective

Use basic statistics to analyze letter frequency in English text and apply those statistics to decode a substitution-based cipher.

Background

Languages exhibit characteristic statistical patterns. In English, certain letters (such as *e*, *t*, and *a*) occur more frequently than others. These patterns can be used to break simple substitution ciphers by comparing letter frequencies in an encoded message to those in standard English text.

Task Description

You are given:

1. A plain (non-encoded) English paragraph
2. A paragraph encoded using a simple substitution cipher

Your task is to:

- Analyze the letter frequency of the plain English text
- Use those statistics to decode the encrypted message

Required Steps

1. **Preprocessing**
    - Convert text to a consistent case
    - Remove punctuation (or justify keeping it)
    - Decide how to handle spaces
2. **Statistical Analysis**
    - Count the frequency of each letter in the plain text
    - Compute relative frequencies
    - Display results using a bar chart
3. **Cipher Analysis**
    - Compute letter frequencies for the encoded text
    - Compare frequency distributions
    - Propose a letter substitution mapping
4. **Decoding**
    - Apply your mapping to the encoded message
    - Iteratively refine the mapping if needed
5. **Evaluation**
    - Present the decoded message
    - Explain which statistical assumptions worked and which did not

---

Deliverables

- Python source code
- Frequency plots
- The decoded message
- A short written explanation (1–2 pages)

Project 1 Sample Dataset: Frequency-Based Decoding

Reference Text (Plain English)

This text is used to establish English letter frequency statistics.

Data science combines statistics, programming, and reasoning.

By examining patterns in data, we can extract meaning,

make predictions, and identify structure in complex systems.

Clear thinking and careful analysis matter more than tools.

Students should:

- Convert to lowercase
- Remove punctuation (or justify keeping it)
- Count letter frequencies

Encoded Message (Student Input)

This paragraph has been encoded using a **simple substitution cipher**.

gsv xlwv gl wzh rmgvi rmgvi rmgvi uli ziv gsviv

z urmwv droo zyzrmhg gsv nzb

The encoded text has the following properties:

- The encoding is a **letter-for-letter substitution**
- Spaces are preserved
- No letter maps to itself

Concepts Reinforced

- Descriptive statistics
- Distributions and visualization
- Real-world application of frequency analysis
- Reasoning under uncertainty

Project 2: Noise Reduction and Outlier Detection in a Time Series

Objective

Analyze a noisy time series, reduce noise using a sliding average, and detect outliers using statistical measures.

Background

Real-world data often contains noise and anomalies. Before meaningful analysis can occur, data must be cleaned and understood. Simple statistical tools such as moving averages and standard deviation can be used to reveal underlying trends and detect unusual behavior.

Task Description

You are given (or generate):

- A one-dimensional time series
- Artificial noise added to the signal
- A small number of extreme outliers

Your task is to:

- Smooth the data
- Identify outliers
- Justify your approach statistically

Required Steps

1. **Data Visualization**
    - Plot the raw time series
    - Identify visible noise and anomalies
2. **Noise Reduction**
    - Implement a sliding (moving) average
    - Experiment with different window sizes
    - Plot the smoothed signal alongside the original
3. **Outlier Detection**
    - Compute point-to-point differences
    - Compute mean and standard deviation of differences
    - Identify points that exceed a chosen threshold
4. **Evaluation**
    - Plot detected outliers
    - Discuss false positives and false negatives

Deliverables

- Python source code
- Before/after plots
- Highlighted outliers
- A short written explanation (1–2 pages)

Concepts Reinforced

- Variability and standard deviation
- Data cleaning
- Visualization as a diagnostic tool

- Practical statistics

Data Set:

Sampling rate: 256 Hz
Duration: 1024 / 256 = 4 seconds
Columns: sample index n, time in seconds t_sec, signal value y

n	t_sec	y
0	0	0.030471707975443137
1	0.003906	0.38777795094726353
2	0.007813	0.5952144369106754
3	0.011719	0.5263803425530875
4	0.015625	0.5406952353458889
5	0.019531	1.0318064305838777
6	0.023438	1.1468613252438262
7	0.027344	0.7642996373897387
8	0.03125	0.7454227214642207
9	0.035156	0.8765014597641982
10	0.039063	0.9302004847255647
11	0.042969	0.387485
12	0.046875	-0.02928
13	0.050781	0.13622761568300076
14	0.054688	0.001663
15	0.058594	-0.62454
16	0.0625	-0.92023
17	0.066406	-0.97812
18	0.070313	-0.6239
19	0.074219	-0.93122
20	0.078125	-1.23024
21	0.082031	-1.08993
22	0.085938	-0.44289
23	0.089844	-0.45255
24	0.09375	-0.60229
25	0.097656	-0.3752
26	0.101563	0.2901406171397947
27	0.105469	0.5939145759157168
28	0.109375	0.5011726360879296
29	0.113281	0.5447981417226944
30	0.117188	1.1448586049394323
31	0.121094	1.1781859332241318
32	0.125	0.9487757270928476

33	0.128906	0.6398577987017076
34	0.132813	0.8947466261018705
35	0.136719	1.0930834385601012
36	0.140625	0.6398463453453881
37	0.144531	0.032394
38	0.148438	-0.12332
39	0.152344	0.11158141766779206
40	0.15625	-0.13158
41	0.160156	-0.69998
42	0.164063	-1.04743
43	0.167969	-0.76292
44	0.171875	-0.73815
45	0.175781	-1.0495
46	0.179688	-1.11499
47	0.183594	-0.81086
48	0.1875	-0.38922
49	0.191406	-0.48284
50	0.195313	-0.50657
51	0.199219	-0.05851
52	0.203125	0.2803446231583815
53	0.207031	0.513437
54	0.210938	0.37948861565873027
55	0.214844	0.5807214256153972
56	0.21875	1.0731420026852525
57	0.222656	1.3319232544289072
58	0.226563	0.7697090568479753
59	0.230469	0.8178915845550798
60	0.234375	0.7588534932322366
61	0.238281	0.8773055357814935
62	0.242188	0.43889946283246867
63	0.246094	0.052806
64	0.25	0.071123
65	0.253906	0.085151
66	0.257813	-0.4575
67	0.261719	-0.95703
68	0.265625	-0.84134
69	0.269531	-0.74019
70	0.273438	-0.98386
71	0.277344	-1.29576
72	0.28125	-1.1926
73	0.285156	-0.59489
74	0.289063	-0.41228
75	0.292969	-0.47636

76	0.296875	-0.46879
77	0.300781	0.13749281274870206
78	0.304688	0.5980400367520029
79	0.308594	0.45866380513881166
80	0.3125	0.5027843049422889
81	0.316406	0.7670317308112136
82	0.320313	1.1658312711765086
83	0.324219	1.033192836118958
84	0.328125	0.6302314327165061
85	0.332031	0.8348373569954999
86	0.335938	0.9339376224180999
87	0.339844	0.7555470374061143
88	0.34375	0.2539814029590466
89	0.347656	-0.00187
90	0.351563	0.10741392882261531
91	0.355469	-0.12626
92	0.359375	-0.69357
93	0.363281	-0.98816
94	0.367188	-1.00188
95	0.371094	-0.86595
96	0.375	-1.13227
97	0.378906	-1.31855
98	0.382813	-0.89072
99	0.386719	-0.59226
100	0.390625	-0.49772
101	0.394531	-0.42745
102	0.398438	-0.27254
103	0.402344	0.4137346446427507
104	0.40625	0.4660983596607393
105	0.410156	0.4165572276647635
106	0.414063	0.47014084479603996
107	0.417969	0.9879351697398856
108	0.421875	1.2957859772767932
109	0.425781	0.7534922445723238
110	0.429688	0.755186
111	0.433594	0.906006
112	0.4375	0.8976917856168696
113	0.441406	0.3940012438367705
114	0.445313	0.052301
115	0.449219	-0.07645
116	0.453125	0.059147
117	0.457031	-0.30752
118	0.460938	-0.68208

119	0.464844	-0.98574
120	0.46875	-0.84945
121	0.472656	-0.778
122	0.476563	-1.11208
123	0.480469	-1.02611
124	0.484375	-0.65229
125	0.488281	-0.39664
126	0.492188	-0.37384
127	0.496094	-0.61065
128	0.5	-0.06398
129	0.503906	0.39911876743076014
130	0.507813	0.48118833701445635
131	0.511719	0.29465525615833466
132	0.515625	0.7993138488927044
133	0.519531	1.1398021115602304
134	0.523438	0.9869966554768375
135	0.527344	0.6943659885033451
136	0.53125	0.7784542219596637
137	0.535156	1.0456185093109924
138	0.539063	1.0419337764084649
139	0.542969	0.6010921558245224
140	0.546875	0.005561
141	0.550781	-0.07545
142	0.554688	-0.25829
143	0.558594	-0.51184
144	0.5625	-1.0384
145	0.566406	-0.92377
146	0.570313	-0.77295
147	0.574219	-0.9403
148	0.578125	-1.10516
149	0.582031	-1.00613
150	0.585938	-0.58101
151	0.589844	-0.54067
152	0.59375	-0.72693
153	0.597656	-0.38861
154	0.601563	0.23153144350262417
155	0.605469	0.7341631608372986
156	0.609375	0.47292682707562356
157	0.613281	0.5999899925242117
158	0.617188	0.8807642849984616
159	0.621094	1.1003330582208906
160	0.625	0.9034883237767659
161	0.628906	0.6487124650729209

162	0.632813	1.045995657087838
163	0.636719	0.898048
164	0.640625	0.73509
165	0.644531	0.026116819496431506
166	0.648438	0.052282
167	0.652344	0.085017
168	0.65625	-0.22157
169	0.660156	-0.75837
170	0.664063	-1.04636
171	0.667969	-0.74153
172	0.671875	-0.79531
173	0.675781	-1.19393
174	0.679688	-1.32993
175	0.683594	-0.81597
176	0.6875	-0.2992
177	0.691406	-0.4736
178	0.695313	-0.54734
179	0.699219	-0.09306
180	0.703125	0.55666
181	0.707031	0.5673425277731755
182	0.710938	0.3854331580046374
183	0.714844	0.7552380814457142
184	0.71875	1.1435318716540768
185	0.722656	1.3360047225050655
186	0.726563	0.874616
187	0.730469	0.5986168459238772
188	0.734375	0.7903245504692397
189	0.738281	1.0758868320032002
190	0.742188	0.5949907284003026
191	0.746094	-0.02377
192	0.75	-0.03832
193	0.753906	0.15196043098900586
194	0.757813	-0.53333
195	0.761719	-1.00027
196	0.765625	-0.86281
197	0.769531	-0.76052
198	0.773438	-0.8568
199	0.777344	-1.19877
200	0.78125	-1.0669
201	0.785156	-0.50386
202	0.789063	-0.41747
203	0.792969	-0.48101
204	0.796875	-0.63108

205	0.800781	0.11676700344800892
206	0.804688	0.4511579703259861
207	0.808594	0.36771715306846997
208	0.8125	0.3692915444936692
209	0.816406	0.7998119808477897
210	0.820313	1.2937269100686144
211	0.824219	0.9387273537448357
212	0.828125	0.752879
213	0.832031	0.7377231655835764
214	0.835938	0.9481105470021755
215	0.839844	0.8545734080638111
216	0.84375	0.25971828076885656
217	0.847656	0.087218
218	0.851563	0.025424850000349268
219	0.855469	-0.186
220	0.859375	-0.67363
221	0.863281	-0.95594
222	0.867188	-0.81549
223	0.871094	-0.82967
224	0.875	-0.99095
225	0.878906	-1.196
226	0.882813	-0.67895
227	0.886719	-0.31403
228	0.890625	-0.54522
229	0.894531	-0.58611
230	0.898438	-0.38325
231	0.902344	0.28091238639968885
232	0.90625	0.5910206280207687
233	0.910156	0.5576863455403505
234	0.914063	0.49223466651273345
235	0.917969	0.9564238333231491
236	0.921875	0.997026
237	0.925781	0.9099576948370731
238	0.929688	0.6055025744454442
239	0.933594	0.8292889523460399
240	0.9375	0.8694207033697909
241	0.941406	0.5291807738503698
242	0.945313	-0.13068
243	0.949219	-0.17021
244	0.953125	0.24880427231452026
245	0.957031	-0.43845
246	0.960938	-0.95194
247	0.964844	-0.77811

248	0.96875	-0.4566
249	0.972656	-0.91308
250	0.976563	-1.1709
251	0.980469	-1.12787
252	0.984375	-0.56293
253	0.988281	-0.53101
254	0.992188	-0.5447
255	0.996094	-0.41404
256	1	0.043477
257	1.003906	0.45416075444829684
258	1.007813	0.5067870208784293
259	1.011719	0.294834
260	1.015625	0.7119813798137916
261	1.019531	1.135385632269206
262	1.023438	1.1572942738897276
263	1.027344	0.7403911747422022
264	1.03125	0.7942566894691576
265	1.035156	1.063077434303524
266	1.039063	0.8578036200061326
267	1.042969	0.3448815500637156
268	1.046875	-0.03056
269	1.050781	0.023512
270	1.054688	-0.11724
271	1.058594	-0.50696
272	1.0625	-0.96684
273	1.066406	-0.67292
274	1.070313	-0.55441
275	1.074219	-0.88764
276	1.078125	-1.28806
277	1.082031	-1.13308
278	1.085938	-0.44603
279	1.089844	-0.41083
280	1.09375	-0.51145
281	1.097656	-0.51444
282	1.101563	0.3296535467402686
283	1.105469	0.6028122033007512
284	1.109375	0.3488563064868846
285	1.113281	0.45456356067690395
286	1.117188	0.9570655652880263
287	1.121094	1.224074
288	1.125	0.970783
289	1.128906	0.7108862447199009
290	1.132813	0.8079509460236212

291	1.136719	0.9954424604982625
292	1.140625	0.7983902884101973
293	1.144531	-0.14026
294	1.148438	-0.06456
295	1.152344	0.064173
296	1.15625	-0.17631
297	1.160156	-0.79149
298	1.164063	-1.15655
299	1.167969	-0.75334
300	1.171875	-0.57708
301	1.175781	-1.22475
302	1.179688	-1.11575
303	1.183594	-0.86608
304	1.1875	-0.46324
305	1.191406	-0.59489
306	1.195313	-0.56893
307	1.199219	0.008366
308	1.203125	0.484326
309	1.207031	0.7186354381777943
310	1.210938	0.5442670699857827
311	1.214844	0.6885178783326782
312	1.21875	1.2750496804246878
313	1.222656	1.2263284391792983
314	1.226563	0.9390909860970111
315	1.230469	0.6914066537424542
316	1.234375	0.9337950519797225
317	1.238281	0.8410516564500499
318	1.242188	0.5215825497814708
319	1.246094	-0.12365
320	1.25	0.078235
321	1.253906	-0.01325
322	1.257813	-0.3055
323	1.261719	-0.83571
324	1.265625	-0.74508
325	1.269531	-0.64799
326	1.273438	-1.0135
327	1.277344	-1.18912
328	1.28125	-1.21786
329	1.285156	-0.69644
330	1.289063	-0.2754
331	1.292969	-0.48165
332	1.296875	-0.49595
333	1.300781	0.020267

334	1.304688	0.5387592066428925
335	1.308594	0.36794244417273037
336	1.3125	0.38999275344689033
337	1.316406	0.8644252720729894
338	1.320313	1.3176678594436813
339	1.324219	1.1322427727412412
340	1.328125	0.520686
341	1.332031	0.8165767818199531
342	1.335938	0.9880812137616273
343	1.339844	0.7956866540706251
344	1.34375	0.3675277052335896
345	1.347656	-0.25285
346	1.351563	-0.01823
347	1.355469	-0.05732
348	1.359375	-0.8094
349	1.363281	-0.83259
350	1.367188	-0.79631
351	1.371094	-0.63658
352	1.375	-1.05709
353	1.378906	-1.13745
354	1.382813	-0.82385
355	1.386719	-0.47843
356	1.390625	-0.43646
357	1.394531	-0.53034
358	1.398438	-0.32324
359	1.402344	0.3252302197719975
360	1.40625	0.5442078744165015
361	1.410156	0.4754403698868851
362	1.414063	0.6651254749781077
363	1.417969	0.9159848691320982
364	1.421875	1.1992542604990974
365	1.425781	1.0743698416574665
366	1.429688	0.6373851927501436
367	1.433594	0.8000078933536143
368	1.4375	0.9773374003696651
369	1.441406	0.6230455483241719
370	1.445313	0.046231
371	1.449219	0.10939256405749909
372	1.453125	0.12155895936834923
373	1.457031	-0.22552
374	1.460938	-0.78685
375	1.464844	-0.72904
376	1.46875	-0.76762

377	1.472656	-0.99628
378	1.476563	-0.97365
379	1.480469	-1.20779
380	1.484375	-0.72501
381	1.488281	-0.30137
382	1.492188	-0.6804
383	1.496094	-0.61694
384	1.5	-0.16013
385	1.503906	0.41236273256877093
386	1.507813	0.564133
387	1.511719	0.48474265519524395
388	1.515625	0.7634261721441079
389	1.519531	1.0207477928808595
390	1.523438	0.9030669412638543
391	1.527344	0.8013592551631536
392	1.53125	0.699925
393	1.535156	1.007744429693214
394	1.539063	0.9124560498614225
395	1.542969	0.3235300524296628
396	1.546875	0.040134
397	1.550781	0.046425
398	1.554688	0.007918
399	1.558594	-0.60907
400	1.5625	-0.97507
401	1.566406	-0.86256
402	1.570313	-0.62969
403	1.574219	-0.9656
404	1.578125	-1.15964
405	1.582031	-1.04842
406	1.585938	-0.5769
407	1.589844	-0.35415
408	1.59375	-0.75877
409	1.597656	-0.46963
410	1.601563	0.088691
411	1.605469	0.3240085574944792
412	1.609375	0.3920729309128377
413	1.613281	0.5766594313946711
414	1.617188	0.9022054382141216
415	1.621094	1.238606443056778
416	1.625	1.108921749671093
417	1.628906	0.8540036847532653
418	1.632813	0.8262348903716876
419	1.636719	1.1155447982449456

420	1.640625	0.6604537569543175
421	1.644531	0.032669714922395315
422	1.648438	-0.10032
423	1.652344	-0.10153
424	1.65625	-0.29472
425	1.660156	-0.7901
426	1.664063	-0.90052
427	1.667969	-0.61406
428	1.671875	-0.88803
429	1.675781	-1.03208
430	1.679688	-1.30619
431	1.683594	-0.78575
432	1.6875	-0.47022
433	1.691406	-0.67269
434	1.695313	-0.44265
435	1.699219	-0.18214
436	1.703125	0.37267018130658697
437	1.707031	0.4384290365078157
438	1.710938	0.36109755341925415
439	1.714844	0.6873982545092114
440	1.71875	1.0817317937142665
441	1.722656	1.2152953235337298
442	1.726563	0.8924843538102069
443	1.730469	0.8531299146487488
444	1.734375	0.8928618553073688
445	1.738281	0.7631082570955017
446	1.742188	0.5293463979791839
447	1.746094	-0.03896
448	1.75	0.11145924445771957
449	1.753906	0.044154
450	1.757813	-0.43574
451	1.761719	-0.87592
452	1.765625	-0.73204
453	1.769531	-0.51337
454	1.773438	-0.84935
455	1.777344	-1.16641
456	1.78125	-0.99303
457	1.785156	-0.72918
458	1.789063	-0.39322
459	1.792969	-0.54799
460	1.796875	-0.39467
461	1.800781	0.038302
462	1.804688	0.3765242480243761

463	1.808594	0.28230011551172635
464	1.8125	0.3453683698875818
465	1.816406	0.7873567964239362
466	1.820313	1.1728174971681387
467	1.824219	1.2650897417617544
468	1.828125	0.8604147342661468
469	1.832031	0.689931
470	1.835938	1.0156487016834594
471	1.839844	0.71359
472	1.84375	0.17747035326444774
473	1.847656	-0.02799
474	1.851563	0.10279252962287212
475	1.855469	-0.15034
476	1.859375	-0.54486
477	1.863281	-1.09438
478	1.867188	-0.83251
479	1.871094	-0.46147
480	1.875	-0.97769
481	1.878906	-1.07551
482	1.882813	-0.92154
483	1.886719	-0.44364
484	1.890625	-0.46558
485	1.894531	-0.57441
486	1.898438	-0.31486
487	1.902344	0.3830132236917855
488	1.90625	0.47430640874315977
489	1.910156	0.5036595070828763
490	1.914063	0.6736717507316778
491	1.917969	1.0584916180536412
492	1.921875	1.183130290175492
493	1.925781	0.9716208448224573
494	1.929688	0.6808767100767588
495	1.933594	0.9757876076291672
496	1.9375	0.7739661727650392
497	1.941406	0.5050462924637736
498	1.945313	-0.15399
499	1.949219	-0.17301
500	1.953125	0.1722657840930742
501	1.957031	-0.22019
502	1.960938	-0.91421
503	1.964844	-1.11206
504	1.96875	-1.04356
505	1.972656	-0.85027

506	1.976563	-0.89204
507	1.980469	-1.11854
508	1.984375	-0.79176
509	1.988281	-0.38582
510	1.992188	-0.67627
511	1.996094	-0.52151
512	2	0.009948
513	2.003906	0.4831662963891985
514	2.007813	0.5992499294990578
515	2.011719	0.46678839353758206
516	2.015625	0.8026313560220637
517	2.019531	1.0931871530470263
518	2.023438	1.2238588257681653
519	2.027344	0.9588175913865102
520	2.03125	0.6500878852494956
521	2.035156	0.8730362869500784
522	2.039063	0.9758391208713251
523	2.042969	0.2905715105542884
524	2.046875	0.10450257809496555
525	2.050781	-0.02075
526	2.054688	0.10041620047343275
527	2.058594	-0.52546
528	2.0625	-0.93128
529	2.066406	-0.72576
530	2.070313	-0.74792
531	2.074219	-1.02034
532	2.078125	-1.25661
533	2.082031	-0.97661
534	2.085938	-0.72172
535	2.089844	-0.37335
536	2.09375	-0.61334
537	2.097656	-0.2252
538	2.101563	-0.00252
539	2.105469	0.47871359196888086
540	2.109375	0.2912525598049419
541	2.113281	0.4190930947855084
542	2.117188	0.95546
543	2.121094	1.2009047723176205
544	2.125	0.9746622431052321
545	2.128906	0.7053165843884914
546	2.132813	0.8534875079063086
547	2.136719	0.8793326050937034
548	2.140625	0.7219260552008104

549	2.144531	0.182676
550	2.148438	-0.00237
551	2.152344	0.1021754831604215
552	2.15625	-0.17626
553	2.160156	-0.55079
554	2.164063	-0.98959
555	2.167969	-3.8633
556	2.171875	-0.82517
557	2.175781	-1.17459
558	2.179688	-1.32658
559	2.183594	-0.9221
560	2.1875	-0.46417
561	2.191406	-0.45617
562	2.195313	-0.53037
563	2.199219	-0.19819
564	2.203125	0.5160786615459371
565	2.207031	0.619346
566	2.210938	0.41056105034582346
567	2.214844	0.5793175959732504
568	2.21875	1.1554990198908899
569	2.222656	1.2012264820540854
570	2.226563	0.7114794424758113
571	2.230469	0.7142657234974853
572	2.234375	0.9533440516012236
573	2.238281	0.8208245601313007
574	2.242188	0.44160849560574394
575	2.246094	-0.1513
576	2.25	0.1336186100012019
577	2.253906	0.1306110002707308
578	2.257813	-0.44788
579	2.261719	-0.87445
580	2.265625	-1.16813
581	2.269531	-0.8367
582	2.273438	-0.88567
583	2.277344	-1.28964
584	2.28125	-1.02922
585	2.285156	-0.44488
586	2.289063	-0.54419
587	2.292969	-0.62915
588	2.296875	-0.40252
589	2.300781	0.28275045849097724
590	2.304688	0.413244
591	2.308594	0.5145020713415338

592	2.3125	0.6392366662678475
593	2.316406	-3.70281
594	2.320313	1.0740297351484296
595	2.324219	1.029005964872738
596	2.328125	0.6977565559898823
597	2.332031	0.8674002376922956
598	2.335938	1.0050438284204488
599	2.339844	0.5768014195691533
600	2.34375	0.25744777735855984
601	2.347656	-0.10428
602	2.351563	0.1683201396194487
603	2.355469	-0.17917
604	2.359375	-0.7149
605	2.363281	-0.92607
606	2.367188	-0.75686
607	2.371094	-0.67643
608	2.375	-1.16856
609	2.378906	-1.16502
610	2.382813	-1.03412
611	2.386719	-0.47819
612	2.390625	-0.60227
613	2.394531	-0.51274
614	2.398438	-0.31757
615	2.402344	0.21171962276995732
616	2.40625	0.6308421413099958
617	2.410156	0.4612654356192285
618	2.414063	0.503745
619	2.417969	1.1669563622320351
620	2.421875	1.1676899213740848
621	2.425781	0.929435
622	2.429688	0.7386353603992586
623	2.433594	0.8202663016697755
624	2.4375	1.0042204105265504
625	2.441406	0.4852617945591925
626	2.445313	0.003925
627	2.449219	0.1127607689941122
628	2.453125	-0.06818
629	2.457031	-0.55098
630	2.460938	-0.68117
631	2.464844	-0.70687
632	2.46875	-0.78763
633	2.472656	-0.98961
634	2.476563	-1.16513

635	2.480469	-1.19065
636	2.484375	-0.75479
637	2.488281	-0.54366
638	2.492188	-0.46221
639	2.496094	-0.43933
640	2.5	-0.14944
641	2.503906	0.5616960347407987
642	2.507813	0.7254378155246353
643	2.511719	0.4495199041576742
644	2.515625	0.7020662380043907
645	2.519531	1.1478240598293379
646	2.523438	1.195602961623982
647	2.527344	0.6228567360734029
648	2.53125	0.7635419075123764
649	2.535156	0.9227594575752566
650	2.539063	1.027043188534897
651	2.542969	0.29228863584331066
652	2.546875	0.13090920000024855
653	2.550781	-0.08687
654	2.554688	0.013637559526639073
655	2.558594	-0.50667
656	2.5625	-1.04401
657	2.566406	-0.86449
658	2.570313	-0.59049
659	2.574219	-0.9586
660	2.578125	-1.38095
661	2.582031	-1.02359
662	2.585938	-0.65539
663	2.589844	-0.47134
664	2.59375	-0.56762
665	2.597656	-0.51055
666	2.601563	0.075344
667	2.605469	0.605577
668	2.609375	0.40762750531138336
669	2.613281	0.24524160723982613
670	2.617188	1.0091782485131875
671	2.621094	1.2460644103900207
672	2.625	0.9286125175963762
673	2.628906	0.5895520504206736
674	2.632813	0.9167294734757259
675	2.636719	1.0151212715440474
676	2.640625	0.8895013193793259
677	2.644531	0.15842839717764262

678	2.648438	-0.00211
679	2.652344	0.029829
680	2.65625	-0.12423
681	2.660156	-0.69179
682	2.664063	-0.85571
683	2.667969	-0.83827
684	2.671875	-0.79336
685	2.675781	-1.11928
686	2.679688	-1.12306
687	2.683594	-0.68339
688	2.6875	-0.50299
689	2.691406	-0.53208
690	2.695313	-0.50407
691	2.699219	-0.14621
692	2.703125	0.5212655704579389
693	2.707031	0.32022654857490007
694	2.710938	0.34385537639633534
695	2.714844	0.5663675770435654
696	2.71875	0.8686206265693039
697	2.722656	1.0860652788686078
698	2.726563	0.7647765568327154
699	2.730469	0.7009532836440314
700	2.734375	1.0384370595598325
701	2.738281	0.8862879736610283
702	2.742188	0.31954278429489547
703	2.746094	-0.01151
704	2.75	0.10491736214409426
705	2.753906	-0.09178
706	2.757813	-0.51368
707	2.761719	-0.85494
708	2.765625	2.850058188360331
709	2.769531	-0.65632
710	2.773438	-0.85766
711	2.777344	-1.11226
712	2.78125	-1.20416
713	2.785156	-0.64582
714	2.789063	-0.44759
715	2.792969	-0.66699
716	2.796875	-0.58241
717	2.800781	0.19021374716849568
718	2.804688	0.5003857591334749
719	2.808594	0.3873704309461029
720	2.8125	0.44748888749420757

721	2.816406	0.9460262106527877
722	2.820313	0.9740628712958102
723	2.824219	0.9217027565284794
724	2.828125	0.6575267530859084
725	2.832031	0.9322579955889443
726	2.835938	1.009136554730296
727	2.839844	0.8310293495794991
728	2.84375	0.091891
729	2.847656	-0.15848
730	2.851563	0.085657
731	2.855469	-0.11058
732	2.859375	-0.59637
733	2.863281	-0.99895
734	2.867188	-0.80533
735	2.871094	-0.70542
736	2.875	-0.92222
737	2.878906	-1.13813
738	2.882813	-1.09268
739	2.886719	-0.72644
740	2.890625	-0.35972
741	2.894531	-0.4386
742	2.898438	-0.33897
743	2.902344	0.154
744	2.90625	0.5693636097624489
745	2.910156	0.5301848060866032
746	2.914063	0.7449025025305017
747	2.917969	1.073468234540084
748	2.921875	1.174583493872498
749	2.925781	0.8369112247330848
750	2.929688	0.7128891447984584
751	2.933594	0.8523064246941172
752	2.9375	0.8555999906136716
753	2.941406	0.7434825943619798
754	2.945313	0.22360519736088177
755	2.949219	0.090101
756	2.953125	-0.05621
757	2.957031	-0.2242
758	2.960938	-0.7783
759	2.964844	-0.91755
760	2.96875	-0.62214
761	2.972656	-0.73239
762	2.976563	-1.06008
763	2.980469	-1.09833

764	2.984375	-0.70172
765	2.988281	-0.61068
766	2.992188	-0.51181
767	2.996094	-0.5474
768	3	-0.12798
769	3.003906	0.6599580235013961
770	3.007813	0.6930688514186131
771	3.011719	0.5682459309435395
772	3.015625	0.7613200958968401
773	3.019531	1.2970868933653488
774	3.023438	1.1352826029873018
775	3.027344	4.616326516583425
776	3.03125	0.637756
777	3.035156	1.001504982971442
778	3.039063	0.8482992800817624
779	3.042969	0.17944069754034692
780	3.046875	-0.041
781	3.050781	0.015530538600490102
782	3.054688	0.13466775951082208
783	3.058594	-0.44919
784	3.0625	-0.95596
785	3.066406	-0.85735
786	3.070313	-0.70732
787	3.074219	-0.94652
788	3.078125	-1.32
789	3.082031	-1.03694
790	3.085938	-0.63975
791	3.089844	-0.56213
792	3.09375	-0.50834
793	3.097656	-0.30086
794	3.101563	0.058238947447608075
795	3.105469	0.5451017057298011
796	3.109375	0.5594695135183668
797	3.113281	0.6076384153115214
798	3.117188	1.033277529140896
799	3.121094	1.222718755552739
800	3.125	0.9154954672031006
801	3.128906	0.6128651887906923
802	3.132813	0.8676091743193338
803	3.136719	1.018114212724511
804	3.140625	0.7799742022714121
805	3.144531	0.22640750233934753
806	3.148438	-0.0541

807	3.152344	-0.0779
808	3.15625	-0.23782
809	3.160156	-0.73257
810	3.164063	-1.00109
811	3.167969	-0.84393
812	3.171875	-0.72453
813	3.175781	-1.12176
814	3.179688	-1.26494
815	3.183594	-0.80208
816	3.1875	-0.49731
817	3.191406	-0.46519
818	3.195313	-0.50816
819	3.199219	-0.23558
820	3.203125	0.377936
821	3.207031	0.6891819716284229
822	3.210938	0.31031792981458955
823	3.214844	0.4329437655426672
824	3.21875	0.9144717124836357
825	3.222656	1.1853401179631504
826	3.226563	0.8593838962206174
827	3.230469	0.7093026635534045
828	3.234375	1.0485594532866611
829	3.238281	0.6435105954697979
830	3.242188	0.4622175437650794
831	3.246094	0.15032781841558246
832	3.25	-0.11278
833	3.253906	-0.03216
834	3.257813	-0.49791
835	3.261719	-1.00023
836	3.265625	-0.95977
837	3.269531	-0.57832
838	3.273438	-0.67255
839	3.277344	-1.21602
840	3.28125	-0.91015
841	3.285156	-0.64105
842	3.289063	-0.25116
843	3.292969	-0.55473
844	3.296875	-0.41295
845	3.300781	0.15818629090527286
846	3.304688	0.853366
847	3.308594	0.5747270165953677
848	3.3125	0.38638032819139867
849	3.316406	0.9301241119454552

850	3.320313	1.1659534021245719
851	3.324219	1.022391607228683
852	3.328125	0.8406755332646418
853	3.332031	0.789249
854	3.335938	1.0087354478895345
855	3.339844	0.755696
856	3.34375	0.23956504972899886
857	3.347656	-0.00403
858	3.351563	-0.15282
859	3.355469	-0.04975
860	3.359375	-0.74944
861	3.363281	-1.12442
862	3.367188	-0.83899
863	3.371094	-0.71284
864	3.375	-1.06935
865	3.378906	-1.13572
866	3.382813	-1.06491
867	3.386719	-0.54241
868	3.390625	-0.51839
869	3.394531	-0.56203
870	3.398438	-0.20902
871	3.402344	0.2391937928336598
872	3.40625	0.6318883954073509
873	3.410156	0.4434012202850182
874	3.414063	0.37594842495588787
875	3.417969	0.8259793332901837
876	3.421875	1.2105232342381287
877	3.425781	0.9041275551129812
878	3.429688	0.7014020042739297
879	3.433594	0.8794346494296267
880	3.4375	0.9796627651375672
881	3.441406	0.6333719902604673
882	3.445313	-0.06602
883	3.449219	-0.1407
884	3.453125	-0.07345
885	3.457031	-0.28081
886	3.460938	-0.71777
887	3.464844	-1.00494
888	3.46875	-0.99727
889	3.472656	-0.96632
890	3.476563	-1.21738
891	3.480469	-1.2178
892	3.484375	-0.77664

893	3.488281	-0.42847
894	3.492188	-0.55134
895	3.496094	-0.38684
896	3.5	-0.0676
897	3.503906	0.40554456562373836
898	3.507813	0.568057
899	3.511719	0.27876
900	3.515625	0.7747679247432644
901	3.519531	1.1722796996427156
902	3.523438	1.1193330359352958
903	3.527344	0.9547515073390561
904	3.53125	0.6848820926851624
905	3.535156	1.167835669573925
906	3.539063	0.8197175423165013
907	3.542969	0.1820042647267831
908	3.546875	-0.02889
909	3.550781	-0.08412
910	3.554688	-0.12026
911	3.558594	-0.4989
912	3.5625	-0.90155
913	3.566406	-0.94445
914	3.570313	-0.613
915	3.574219	-0.81047
916	3.578125	-1.06812
917	3.582031	-0.9689
918	3.585938	-0.4288
919	3.589844	-0.62518
920	3.59375	-0.59125
921	3.597656	-0.42668
922	3.601563	0.24883228469716828
923	3.605469	0.5002252475596416
924	3.609375	0.44328359073339796
925	3.613281	0.6899335294423008
926	3.617188	0.913722
927	3.621094	1.260206666581615
928	3.625	0.9767730670076402
929	3.628906	0.7288064003221862
930	3.632813	0.8337503518879245
931	3.636719	1.0250186158853285
932	3.640625	0.7677718446656103
933	3.644531	0.28114893706548927
934	3.648438	-0.00991
935	3.652344	0.10547682748884682

936	3.65625	-0.32099
937	3.660156	-0.76309
938	3.664063	-0.88685
939	3.667969	-0.69954
940	3.671875	-0.72865
941	3.675781	-0.98273
942	3.679688	-1.15306
943	3.683594	-0.71319
944	3.6875	-0.42817
945	3.691406	-0.52517
946	3.695313	-0.5019
947	3.699219	-0.4147
948	3.703125	0.46434877878243436
949	3.707031	0.1805629951153459
950	3.710938	0.25417954034747614
951	3.714844	0.6897860705338121
952	3.71875	1.148409161534564
953	3.722656	1.0661862452914816
954	3.726563	0.7850819642294455
955	3.730469	0.858118
956	3.734375	5.628952683881519
957	3.738281	1.135086070020383
958	3.742188	0.42243217100440433
959	3.746094	0.034988
960	3.75	0.16168743984985443
961	3.753906	0.018918713384137087
962	3.757813	-0.52286
963	3.761719	-0.92177
964	3.765625	-0.9307
965	3.769531	-0.85754
966	3.773438	-0.88188
967	3.777344	-1.25665
968	3.78125	-1.00822
969	3.785156	-0.64115
970	3.789063	-0.45481
971	3.792969	-0.55602
972	3.796875	-0.40375
973	3.800781	0.18332026360995332
974	3.804688	0.4355097744883152
975	3.808594	0.38543969584875776
976	3.8125	0.5675747027268021
977	3.816406	0.7919906376973515
978	3.820313	1.0604524465733218

979	3.824219	1.1157478964571608
980	3.828125	0.7961523778786948
981	3.832031	0.6330685831912634
982	3.835938	1.0038260279847155
983	3.839844	0.8278559347762137
984	3.84375	0.24334538223913652
985	3.847656	0.016676009490817836
986	3.851563	-0.09955
987	3.855469	-0.08331
988	3.859375	-0.6815
989	3.863281	-1.02847
990	3.867188	-0.74609
991	3.871094	-0.57331
992	3.875	-0.82056
993	3.878906	-1.08735
994	3.882813	-0.94167
995	3.886719	-0.46644
996	3.890625	-0.38322
997	3.894531	-0.54525
998	3.898438	-0.22383
999	3.902344	0.4223584009836439
1000	3.90625	0.5535318630591742
1001	3.910156	0.36417228981418004
1002	3.914063	0.523696
1003	3.917969	1.0852295151032714
1004	3.921875	1.2120544926510533
1005	3.925781	0.9602452845784428
1006	3.929688	0.7787519398516203
1007	3.933594	0.8447487487928513
1008	3.9375	1.0327315975495257
1009	3.941406	0.5764912896215195
1010	3.945313	-0.07839
1011	3.949219	0.12072698413241142
1012	3.953125	-0.01419
1013	3.957031	-0.47522
1014	3.960938	-0.94677
1015	3.964844	-1.06391
1016	3.96875	-0.74189
1017	3.972656	-0.82331
1018	3.976563	-1.16776
1019	3.980469	-1.10005
1020	3.984375	-0.70181
1021	3.988281	-0.40072

1022	3.992188	-0.47919
1023	3.996094	-0.43016

Project 3: Geometric Transformations Using a 2D Rotation Matrix

Objective

Use linear algebra to rotate two-dimensional shapes and visualize the results using Pygame.

Background

Linear algebra provides a compact way to describe geometric transformations. Rotation matrices are a classic example of how matrix–vector multiplication produces visible, meaningful change.

Task Description

You will:

- Define one or more 2D shapes as sets of points
- Construct a 2D rotation matrix
- Apply the matrix to rotate the shape
- Visualize the transformation in real time

Required Steps

1. **Define Geometry**
    - Represent a shape as a list of 2D vectors
    - Plot or display the initial shape
2. **Rotation Matrix**
    - Construct the 2D rotation matrix
    - Explain how it works geometrically

3. **Transformation**
    - Apply the rotation matrix to all points
    - Update the display to show rotation
4. **Visualization**
    - Use Pygame to animate or display the rotated shape
    - Ensure smooth and correct motion

Deliverables

- Python source code
- Working Pygame visualization
- A short written explanation (1 page)

Concepts Reinforced

- Vectors and matrices
- Matrix–vector multiplication
- Linear transformations
- Visual intuition for linear algebra

2D Rotation in the Plane

One of the most useful geometric transformations in two dimensions is **rotation**. A rotation turns every point in the plane around the origin by a fixed angle, while preserving distances and shapes.

To describe rotation mathematically, we use a **rotation matrix**.

The 2D Rotation Matrix

A rotation by an angle θ (measured counterclockwise) is given by the matrix:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

This matrix encodes how the xxx- and yyy-coordinates of a point change when rotated.

Rotating a Point

Suppose we have a point in the plane represented as a vector:

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$$

To rotate this point by an angle θ, we multiply:

$$\mathbf{p}' = R(\theta)\mathbf{p}$$

Carrying out the multiplication gives:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

So the rotated coordinates are:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Key Geometric Idea

- The original point (x,y) and the rotated point (x',y') are the **same distance from the origin**
- Only the **direction** changes
- The shape formed by multiple points is preserved

This is why rotation matrices are said to represent **rigid transformations**.

Why This Works (Intuition)

The sine and cosine functions encode how much of a vector lies along the horizontal and vertical directions after rotation. The rotation matrix simply recombines those components in a systematic way.

No special cases are required — the same formula works for *any* angle.

Connection to the Project

In the rotation project:

- Each point of a shape is stored as a 2D vector
- The same rotation matrix is applied to every point
- Repeated application with a small angle produces smooth rotation

This is a direct application of matrix–vector multiplication, made visible.

c

Takeaway

A 2D rotation matrix is a compact mathematical object that:

- Encodes a geometric transformation
- Preserves shape and size
- Produces visible motion when applied repeatedly

Once implemented, it becomes one of the clearest examples of how linear algebra “does something.”

Why Sometimes We Use a 3×3 Matrix in 2D

When we write a 2D point as:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

a **2×2 matrix** is sufficient for **pure rotations and scalings** about the origin.

However, translations (moving a shape without rotating it) **cannot** be represented by a 2×2 matrix multiplication alone. To unify **rotation, scaling, and translation** under a single matrix multiplication, we use **homogeneous coordinates**.

Homogeneous Coordinates

In homogeneous coordinates, a 2D point (x,y) is represented as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The extra coordinate does not represent a physical dimension; it is a mathematical convenience that allows translations to be expressed as matrix multiplications.

The 3×3 Rotation Matrix (Homogeneous Form)

A rotation by angle θ\thetaθ about the origin can be written as:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying this matrix by a point gives:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Which yields:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

The result is the same as with the 2×2 rotation matrix.

Why Bother with the Extra Dimension?

The power shows up when we include **translation**:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Now:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Rotation, scaling, and translation can all be **combined into a single matrix** by multiplication.

Why This Is Common in Graphics (and Pygame)

Using 3×3 matrices allows you to:

- Rotate a shape
- Translate it
- Combine transformations
- Apply everything with **one matrix multiply per point**

This is why graphics pipelines (2D and 3D) almost always use homogeneous coordinates.

General Expectations (All Projects)

- Code must be readable and well-organized
- Clear variable names and comments are required
- Plots and visualizations must be labeled
- Written explanations should emphasize *reasoning*, not jargon

---

Final Note to Students

These projects are designed to show how relatively simple mathematical and statistical ideas can be used to solve meaningful problems. Sophisticated tools are not required—only careful thinking and clear implementation.

Appendix A: Python Quick Reference

This appendix provides a concise reference to core Python concepts and syntax used throughout this book. It is not intended to be a complete guide to Python, but rather a practical reminder of common patterns, constructs, and built-in functions.

A.1 Basic Program Structure and Logic

Variables and Assignment

```
x = 10
y = 3.5
name = "data science"
```

Python is dynamically typed; variables do not require explicit type declarations.

Conditional Statements

```
if x > 0:
    print("positive")
elif x == 0:
    print("zero")
else:
    print("negative")
```

Indentation defines code blocks.

Loops

For loop:

```
for i in range(5):
    print(i)
```

While loop:

```
count = 0
while count < 5:
    count += 1
```

Loops are commonly used for iteration, accumulation, and repeated computation.

A.2 Data Structures

Lists

```
data = [1, 2, 3, 4]
data.append(5)
```

- Ordered
- Mutable
- Can contain mixed types

Tuples

```
point = (2, 3)
```

- Ordered
- Immutable
- Often used for fixed collections (e.g., coordinates)

Dictionaries

```
person = {"name": "Alice", "age": 30}
```

- Key–value pairs
- Unordered (conceptually)
- Useful for labeled data

A.3 Functions

Defining a Function

```
def square(x):
    return x * x
```

Functions:

- Encapsulate logic
- Improve readability
- Enable reuse

Multiple Parameters and Return Values

```
def stats(x, y):
    return x + y, x - y
```

Returned values are grouped as a tuple.

A.4 Classes (Minimal Introduction)

Defining a Class

```
class LinearModel:
    def __init__(self, w, b):
        self.w = w
        self.b = b

    def predict(self, x):
        return self.w * x + self.b
```

Classes are used to bundle:

- Data (attributes)
- Behavior (methods)
- The “self” variable hold the class variables.

This book uses classes sparingly and only when they clarify structure.

A.5 Common Built-in Functions and Keywords

Frequently Used Built-ins

Name	Purpose
len	Length of a collection
sum	Sum of elements
min	Smallest value
max	Largest value
range	Generate integer sequences
abs	Absolute value
round	Rounding numbers
zip	Iterate over multiple sequences

Membership and Iteration

```
if x in data:
    print("found")

for a, b in zip(list1, list2):
    print(a, b)
```

A.6 Useful Python Idioms

List Comprehensions

```
squares = [x*x for x in range(5)]
```

Compact syntax for generating lists.

Enumerate

```
data = [10, 20, 30, 40]

for index, value in enumerate(data):
    print(index, value)
```

Provides index and value together.

Output:

0 10

1 20

2 30

3 40

Why enumerate Is Useful

Without enumerate, you might write:

```
for i in range(len(data)):
    print(i, data[i])
```

While this works, enumerate is:

- Clearer
- Less error-prone
- More idiomatic Python

Key Idea

enumerate pairs each element in a sequence with its index, making it ideal when you need both position and value during iteration.

Connection to This Book

enumerate is frequently used when:

- Updating values inside loops
- Tracking iterations

- Implementing learning algorithms that require indices

It keeps loops readable while preserving control.

A.7 Notes on Style and Readability

- Prefer clear variable names
- Avoid unnecessary cleverness
- Write code for humans first
- Clarity beats conciseness

Appendix Summary

This appendix serves as a quick reminder of Python syntax and patterns used throughout the book. Students are encouraged to return here whenever syntax details distract from the underlying concepts.

Appendix B: NumPy Quick Reference

This appendix provides a concise reference to the NumPy concepts and operations used throughout this book. It is not intended to be a complete guide to NumPy, but rather a focused summary of array creation, manipulation, and common numerical operations.

B.1 What NumPy Is

NumPy provides efficient data structures and operations for numerical computing in Python. Its core object is the **ndarray**, which represents a multidimensional array of numbers.

NumPy is used throughout this book to:

- Represent vectors and matrices
- Perform fast numerical computations
- Express mathematical operations clearly and compactly

B.2 Creating NumPy Arrays

From Python Lists

```
import numpy as np

a = np.array([1, 2, 3])
```

Creates a one-dimensional array (vector).

```
A = np.array([[1, 2],
              [3, 4]])
```

Creates a two-dimensional array (matrix).

Common Array Creation Functions

```
np.zeros(3)         # [0. 0. 0.]
np.ones(4)          # [1. 1. 1. 1.]
np.arange(5)        # [0 1 2 3 4]
np.linspace(0, 1, 5)
```

These functions are commonly used for initialization and testing. These functions tend to be very similar to those used in MatLab.

B.3 Array Shape and Dimensions

Shape

```
A.shape
```

Returns a tuple indicating the array's dimensions.

Examples:

- (5,) → vector with 5 elements
- (4, 2) → 4 rows, 2 columns

Reshaping Arrays

```
a = np.array([1, 2, 3, 4])
A = a.reshape(2, 2)
```

Reshaping changes the *view* of the data, not the data itself.

```
print(A)
```

```
[[1 2]
 [3 4]]
```

B.4 Indexing and Slicing

Indexing

```
a[0]      # first element
A[1, 0]   # row 1, column 0
```

Slicing

```
a[1:4]
A[:, 0]   # all rows, first column
A[0, :]   # first row, all columns
```

Slicing is frequently used to extract features or subsets of data.

B.5 Vectorized Operations

NumPy applies operations **element-wise by default**.

```
a = np.array([1, 2, 3])
a * 2
```

```
Result:
```

```
[2 4 6]
```

This replaces many explicit loops and improves performance and clarity.

B.6 Matrix and Vector Operations

Dot Product and Matrix Multiplication

```
np.dot(a, b)
```

```
or, using the @ operator:
```

```
A @ B
```

The @ operator is used throughout this book to represent matrix–vector and matrix–matrix multiplication.

Transpose

```
A.T
```

Commonly used in gradient calculations.

B.7 Broadcasting

Broadcasting allows NumPy to perform operations between arrays of different shapes.

Example:

```
X = np.array([[1, 2],
              [3, 4]])
b = 1

X + b
```

The scalar b is added to every element of X.

Broadcasting is heavily used when adding biases in learning models.

B.8 Common Statistical Functions

```
np.mean(a)
np.var(a)
np.std(a)
np.sum(a)
np.min(a)
np.max(a)
```

These functions operate efficiently on entire arrays.

B.9 Linear Algebra Utilities

```
np.linalg.solve(A, b)
np.linalg.inv(A)
np.linalg.norm(a)
```

Used for solving systems of equations and measuring vector lengths.

B.10 NumPy in Learning Algorithms

In this book, NumPy is used to:

- Compute predictions: X @ w + b
- Measure error: y_hat - y
- Update parameters efficiently
- Replace nested loops with clear mathematical expressions

NumPy does **not** replace understanding — it expresses it more compactly.

Appendix Summary

This appendix provides a quick reference to the NumPy features used throughout the book. Students are encouraged to return here whenever array syntax or behavior becomes a distraction from the underlying ideas.

Appendix C – Differences in Syntax and Functionality of Python and Traditional Languages (Java/C/C++)

Note: This is a very brief summary of the differences. This is an AI Overview by Google.

The primary differences in syntax between Python and C/C++/Java relate to code structure, typing, and verbosity. C/C++/Java share a very similar syntax, while Python's syntax is designed for simplicity and readability.

Key Syntactical Differences

Feature	Python	C/C++/Java
Code Blocks	Uses **indentation** and **colons** (:).	Uses **curly braces** ({}) to define code blocks.
Statement Termination	Newlines terminate statements; semicolons (;) are generally optional and rarely used.	Statements **must** be terminated by a semicolon (;).
Variable Declaration	**Dynamically typed**: types are not explicitly declared; the interpreter infers them at runtime (e.g., x = 10).	**Statically typed**: variable types must be explicitly declared before use (e.g., int x = 10;).
Object Creation	Does not use a special keyword; object creation is implicit (e.g., my_object = MyClass()).	Uses the new keyword to allocate objects (e.g., MyClass* my_object = new MyClass(); in C++ or MyClass my_object = new MyClass(); in Java).
Main Function	Code execution typically starts from the first line, without a strict main function requirement.	Requires a specific main() function as the entry point for program execution.
Parentheses in Control Flow	Generally does not require parentheses around conditional expressions (e.g., if x == 10:).	Requires parentheses around control flow conditions (e.g., if (x == 10) { ... }).

In summary, Python is designed to be more concise and closer to plain English, resulting in shorter programs compared to equivalent C/C++/Java code. C/C++/Java, with their C-

derived syntax, are more rigid and verbose, requiring explicit declarations and structural symbols like braces and semicolons

Appendix D: Pygame Template

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 27 18:17:34 2023

@author: patrick

Simple Template for pygame.

"""

import pygame as p

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
ORANGE = (255, 165, 0)
YELLOW = (255, 255, 0)
CYAN = (0, 255, 255)
MAGENTA = (255, 0, 255)

screenWidth = 700
screenHeight = 500

def pyGameTemplate():

    p.init()


    # Set the width and height of the screen [width, height]
    size = (screenWidth, screenHeight)
    screen = p.display.set_mode(size)

    p.display.set_caption("basic Python graphics window()")

    # Loop until the user clicks the close button.
    running = True

    # Used to manage how fast the screen updates
    clock = p.time.Clock()

    # -------- Main Program Loop -----------
    while running:
        # --- Main event loop
        for event in p.event.get():
            if event.type == p.QUIT:
                running = False

        """ Check for keyboard presses. """
        key = p.key.get_pressed()
```

```
        if (key[p.K_ESCAPE] == True):
            running = False
        if (key[p.K_UP] == True):
            pass
        if (key[p.K_DOWN] == True):
            pass
        if (key[p.K_LEFT] == True):
            pass
        if (key[p.K_RIGHT] == True):
            pass
        if (key[p.K_SPACE] == True):
            pass

        # --- Game logic should go here


        # --- Screen-clearing code goes here

        # Here, we clear the screen to black. Don't put other drawing
commands
        # above this, or they will be erased with this command.

        # If you want a background image, replace this clear with blit'ing
the
        # background image.
        screen.fill(BLACK)


        # --- Drawing code should go here
        p.draw.circle(screen, ORANGE, [350, 250], 50, 10)
        p.draw.line(screen, GREEN, [0, 0], [700, 500], 2)
        p.draw.line(screen, CYAN, [0, 500], [700, 0], 2)


        # --- Go ahead and update the screen with what we've drawn.
        p.display.flip()

        # --- Limit to 60 frames per second
        clock.tick(60)

    # Close the window and quit.
    p.quit()

    return

pyGameTemplate()
```

References and Acknowledgements

This set of notes was compiled by the author using Southeastern Louisiana University's course descriptions, the authors ideas and materials, and with great contributions from Open AI's ChatGPT and Google AI.

1. Histogram example; https://discovery.cs.illinois.edu/learn/Exploratory-Data-Analysis/Histograms/
2. Scatterplot example; https://www.geeksforgeeks.org/maths/scatter-plot/
3. Histogram Skewness; https://naperville203algebra1.weebly.com/guided-learning-a2.html
4. Interpreting Histograms; https://www.facebook.com/emfreseearch/posts/interpreting-the-histogramwith-simple-measures-of-central-tendency-the-shape-of-/802834248689662/