Introduction to Data Science

Lecture Notes

Licence L3 — 2025–2026

Yaë Ulrich Gaba

"All models are wrong, but some are useful."
— George E. P. Box

March 25, 2026

Contents

Preface

Data science is an interdisciplinary field that combines mathematics, statistics, and computer science to extract knowledge from data. It differs from classical statistics in scale (data volume), tools (programming, machine learning), and practical orientation (decisions, products, predictions).

These notes are designed for undergraduate students with basic Python and statistics knowledge. Each chapter starts from a **real dataset**, introduces concepts, and implements them in Python. The goal is to make the student autonomous in conducting a complete data analysis, from import to communication of results.

Prerequisites. Basic Python programming (variables, loops, functions), introductory descriptive statistics and probability.

Datasets used. Iris, Titanic, Tips (Seaborn), California Housing, 20 Newsgroups, AirPassengers — all freely available.

References.

- VanderPlas — *Python Data Science Handbook*, O'Reilly (freely available online).
- McKinney — *Python for Data Analysis*, O'Reilly.
- Géron — *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly.
- James, Witten, Hastie, Tibshirani — *An Introduction to Statistical Learning (ISLR)*, Springer (free).

Chapter 1

The Data Science Pipeline

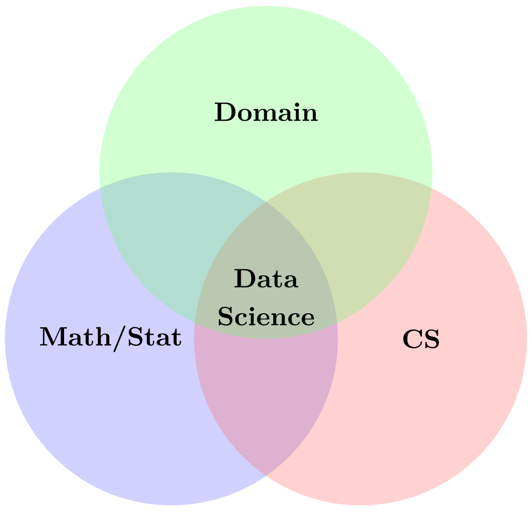
1.1 What is Data Science?

Definition 1.1 (Data Science)**. Data science** is an interdisciplinary field that uses scientific methods, algorithms, and systems to extract knowledge and insights from structured and unstructured data.

Intuition

Data science sits at the intersection of three domains:

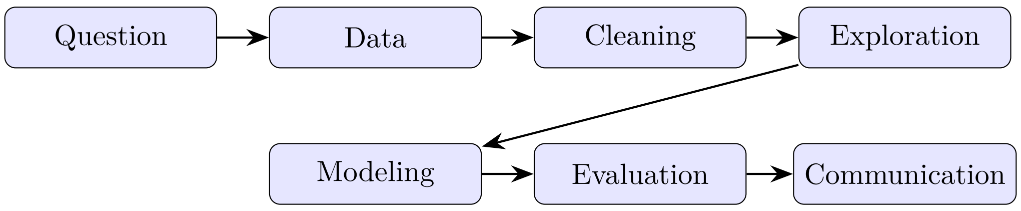
- **Mathematics and Statistics**: modeling, inference, probability.
- **Computer Science**: programming, algorithms, databases.
- **Domain Expertise**: understanding of the application area.



1.2 The Typical Pipeline

A data science project generally follows a multi-step pipeline:

CHAPTER 1. THE DATA SCIENCE PIPELINE



1. **Problem formulation**: clearly define the question.
2. **Data collection**: import from files, databases, APIs.
3. **Cleaning**: handle missing values, duplicates, errors.
4. **Exploration** (EDA): visualize and summarize the data.
5. **Modeling**: apply statistical or machine learning models.
6. **Evaluation**: measure model performance.
7. **Communication**: present results clearly.

1.3 The Python Data Science Ecosystem

Essential Libraries

Library	Role
`numpy`	Numerical computing, arrays
`pandas`	Tabular data manipulation
`matplotlib`	Basic visualization
`seaborn`	Statistical visualization
`scikit-learn`	Machine learning
`scipy`	Statistics and optimization

Python — Installation and imports

```python
# Installation (in a terminal)
# pip install numpy pandas matplotlib seaborn scikit-learn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

print(f"NumPy:   {np.__version__}")
print(f"Pandas:  {pd.__version__}")
print(f"Seaborn: {sns.__version__}")
```

Output

```
NumPy:   1.26.4
Pandas:  2.2.1
Seaborn: 0.13.2
```

1.4 First Complete Example: the Iris Dataset

Let us illustrate the full pipeline on Fisher's **Iris** dataset (1936), containing 150 flower measurements across 3 species.

Python — Load and explore Iris

```python
from sklearn.datasets import load_iris

# 1. Load data
iris = load_iris(as_frame=True)
df = iris.frame
print(df.shape)
print(df.head())
```

Output

```
(150, 5)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
0                5.1               3.5                1.4               0.2       0
1                4.9               3.0                1.4               0.2       0
2                4.7               3.2                1.3               0.2       0
3                4.6               3.1                1.5               0.2       0
4                5.0               3.6                1.4               0.2       0
```

Python — Descriptive statistics

```python
# 2. Explore
print(df.describe().round(2))
print(f"\nMissing values:\n{df.isnull().sum()}")
```

Output

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  ta
count             150.00            150.00             150.00            150.00  15
mean                5.84              3.06               3.76              1.20
std                 0.83              0.44               1.77              0.76
min                 4.30              2.00               1.00              0.10
25%                 5.10              2.80               1.60              0.30
50%                 5.80              3.00               4.35              1.30
75%                 6.40              3.30               5.10              1.80
```

CHAPTER 1. THE DATA SCIENCE PIPELINE

```
max                   7.90              4.40              6.90              2.50

Missing values:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
```

Python — Visualization

```python
# 3. Visualize
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for species in [0, 1, 2]:
    subset = df[df['target'] == species]
    axes[0].hist(subset['petal length (cm)'], alpha=0.6,
                 label=iris.target_names[species], bins=15)
axes[0].set_xlabel('Petal Length (cm)')
axes[0].set_ylabel('Frequency')
axes[0].legend()

for species in [0, 1, 2]:
    subset = df[df['target'] == species]
    axes[1].scatter(subset['sepal length (cm)'],
                    subset['petal length (cm)'],
                    label=iris.target_names[species], alpha=0.7)
axes[1].set_xlabel('Sepal Length (cm)')
axes[1].set_ylabel('Petal Length (cm)')
axes[1].legend()
plt.tight_layout()
plt.savefig('ch01_iris_explore.pdf')
plt.show()
```

Python — Simple modeling

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 4. Model
X = df[['sepal length (cm)', 'sepal width (cm)',
        'petal length (cm)', 'petal width (cm)']]
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```python
y_pred = knn.predict(X_test)

# 5. Evaluate
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2%}")
```

Output

```
Accuracy: 100.00%
```

1.5 Types of Data Science Problems

Supervised Regression, Classification	**Unsupervised** Clustering, Reduction	**Reinforcement** Agent, Reward

- **Supervised learning**: we have a label y for each observation $\boldsymbol{x}$.
  - *Regression*: $y \in \mathbb{R}$ (predict a price, temperature).
  - *Classification*: $y \in \{0, 1, \ldots, K-1\}$ (detect spam, cancer).
- **Unsupervised learning**: no labels, we seek structure (clusters, principal components).
- **Reinforcement learning**: an agent learns by trial and error in an environment.

1.6 Best Practices

Best Practice

1. **Reproducibility**: fix random seeds (`random_state=42`), version control.
2. **Train/test separation**: never evaluate on training data.
3. **Documentation**: comment code, note assumptions.
4. **Iteration**: start simple, add complexity gradually.

Warning

Never look at test data before finalizing your model. *Data leakage* is the most common and dangerous mistake in data science.

1.7 Exercises

Exercise 1.1 ($\star$)**.** Load the `load_wine()` dataset from scikit-learn. Display its dimensions, variable names, and descriptive statistics.

Exercise 1.2 ($\star$)**.** On the Iris dataset, create a scatter plot of sepal width vs. sepal length, colored by species.

CHAPTER 1. THE DATA SCIENCE PIPELINE

Exercise 1.3 (⋆⋆)**.** Apply a k-nearest neighbors classifier to the Wine dataset with $k = 3, 5, 7$. Compare accuracies on a 30% test set.

Exercise 1.4 (⋆⋆)**.** Write a function `full_pipeline(dataset, k)` that loads a scikit-learn dataset, splits into train/test, trains a KNN with k neighbors, and returns the accuracy.

Chapter Summary

- Data science follows a pipeline: Question → Data → Cleaning → Exploration → Modeling → Evaluation → Communication.
- The Python ecosystem provides powerful tools: `pandas`, `matplotlib`, `scikit-learn`.
- Three major families: supervised, unsupervised, reinforcement.
- Golden rules: reproducibility, train/test separation, start simple.

Chapter 2

Data Manipulation with Pandas

2.1 Introduction to Pandas

Pandas is the central library for data science in Python. It provides two fundamental data structures: the `Series` (indexed vector) and the `DataFrame` (indexed table).

Definition 2.1 (DataFrame)**.** A **DataFrame** is a two-dimensional table with labels for rows (index) and columns. Each column is a `Series` of the same length.

2.2 Creating DataFrames

Python — Creation

```python
import pandas as pd
import numpy as np

# From a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'Age': [25, 30, 35, 28],
    'Salary': [45000, 55000, 70000, 52000],
    'City': ['Paris', 'Lyon', 'Paris', 'Marseille']
}
df = pd.DataFrame(data)
print(df)
print(f"\nShape: {df.shape}")
print(f"Types:\n{df.dtypes}")
```

Output

```
      Name  Age  Salary       City
0    Alice   25   45000      Paris
1      Bob   30   55000       Lyon
2  Charlie   35   70000      Paris
3    Diana   28   52000  Marseille
```

```
Shape: (4, 4)
Types:
Name      object
Age        int64
Salary     int64
City      object
dtype: object
```

2.3 Loading Data

Python — Reading files

```python
# CSV (most common)
# df = pd.read_csv('data.csv')
# df = pd.read_csv('data.csv', sep=';', encoding='utf-8')

# From Seaborn (built-in datasets)
import seaborn as sns
tips = sns.load_dataset('tips')   # restaurant tips
print(f"Shape: {tips.shape}")
print(tips.head(3))
```

Output

```
Shape: (244, 7)
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
```

2.4 Selection and Indexing

Selection Methods

Syntax	Description
`df['col']`	Select one column (Series)
`df[['c1','c2']]`	Select multiple columns
`df.loc[i, 'col']`	Select by label
`df.iloc[i, j]`	Select by position
`df[df['col'] > x]`	Boolean filtering

Python — Selection

```
import seaborn as sns
tips = sns.load_dataset('tips')

# Column selection
print("--- Single column ---")
print(tips['total_bill'].head(3))

# Filtering
big_tips = tips[tips['tip'] > 5]
print(f"\n--- Tips > $5: {len(big_tips)} rows ---")
print(big_tips.head(3))

# Combined selection
print("\n--- loc: females, Sunday ---")
mask = (tips['sex'] == 'Female') & (tips['day'] == 'Sun')
print(tips.loc[mask, ['total_bill', 'tip']].head(3))
```

Output

```
--- Single column ---
0    16.99
1    10.34
2    21.01
Name: total_bill, dtype: float64

--- Tips > $5: 17 rows ---
    total_bill   tip   sex smoker  day    time  size
23       39.42  7.58  Male     No  Sat  Dinner     4
44       30.40  5.60  Male     No  Sun  Dinner     4
47       32.40  6.00  Male     No  Sun  Dinner     4

--- loc: females, Sunday ---
    total_bill   tip
0        16.99  1.01
5        25.29  4.71
15       21.58  3.92
```

2.5 Modifying and Adding Columns

Python — New columns

```
# Add a computed column
tips['tip_pct'] = (tips['tip'] / tips['total_bill'] * 100).round(1)
print(tips[['total_bill', 'tip', 'tip_pct']].head(5))
```

CHAPTER 2. DATA MANIPULATION WITH PANDAS

```python
# Apply a function
tips['bill_category'] = tips['total_bill'].apply(
    lambda x: 'High' if x > 30 else ('Medium' if x > 15 else 'Low'))
print(tips['bill_category'].value_counts())
```

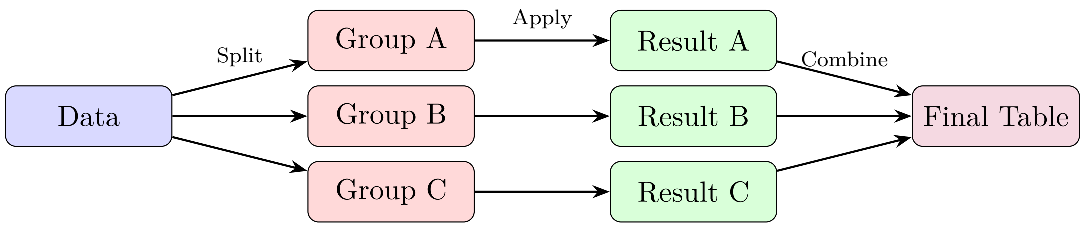
Output

```
   total_bill   tip  tip_pct
0       16.99  1.01      5.9
1       10.34  1.66     16.1
2       21.01  3.50     16.7
3       23.68  3.31     14.0
4       24.59  3.61     14.7

bill_category
Medium    111
Low        69
High       64
Name: count, dtype: int64
```

2.6 Aggregation with groupby

Definition 2.2 (Split-Apply-Combine)**.** The **split-apply-combine** paradigm consists of:

1. **Split**: divide data into groups.
2. **Apply**: apply a function (sum, mean, count…) to each group.
3. **Combine**: assemble the results.



Python — groupby

```python
# Mean by day
print("--- Mean by day ---")
print(tips.groupby('day')[['total_bill', 'tip']].mean().round(2))

# Multiple aggregations
print("\n--- Multiple aggregations ---")
agg = tips.groupby(['day', 'time']).agg(
    n_meals=('total_bill', 'count'),
    avg_bill=('total_bill', 'mean'),
    avg_tip=('tip', 'mean')
).round(2)
print(agg)
```

Output

```
--- Mean by day ---
      total_bill   tip
day
Thur       17.68  2.77
Fri        17.15  2.73
Sat        20.44  2.99
Sun        21.41  3.26

--- Multiple aggregations ---
             n_meals  avg_bill  avg_tip
day  time
Thur Lunch        61     17.66     2.77
     Dinner        1     18.78     3.00
Fri  Lunch         7     12.85     2.38
     Dinner       12     19.66     2.94
Sat  Dinner       87     20.44     2.99
Sun  Dinner       76     21.41     3.26
```

2.7 Pivot Tables

Python — pivot_table

```
pivot = tips.pivot_table(
    values='tip', index='day', columns='sex',
    aggfunc='mean').round(2)
print(pivot)
```

Output

```
sex   Female  Male
day
Thur    2.58  2.88
Fri     2.78  2.69
Sat     2.80  3.08
Sun     3.37  3.22
```

CHAPTER 2. DATA MANIPULATION WITH PANDAS

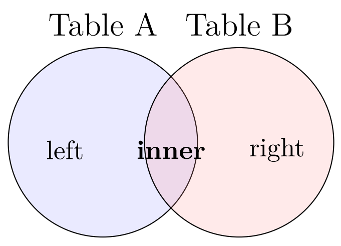
2.8 Joins

Python — merge

```python
# Two DataFrames
clients = pd.DataFrame({
    'client_id': [1, 2, 3, 4],
    'name': ['Alice', 'Bob', 'Charlie', 'Diana']
})
orders = pd.DataFrame({
    'order_id': [101, 102, 103, 104],
    'client_id': [1, 2, 2, 5],
    'amount': [150, 200, 80, 300]
})

# Inner join
inner = pd.merge(clients, orders, on='client_id', how='inner')
print("--- Inner join ---")
print(inner)

# Left join (keep all clients)
left = pd.merge(clients, orders, on='client_id', how='left')
print("\n--- Left join ---")
print(left)
```

Output

```
--- Inner join ---
   client_id     name  order_id  amount
0          1    Alice       101     150
1          2      Bob       102     200
2          2      Bob       103      80

--- Left join ---
   client_id     name  order_id  amount
0          1    Alice     101.0   150.0
1          2      Bob     102.0   200.0
2          2      Bob     103.0    80.0
3          3  Charlie       NaN     NaN
4          4    Diana       NaN     NaN
```



outer = A ∪ B

2.9 Sorting and Ranking

Python — Sorting

```
# Sort by tip descending
top5 = tips.nlargest(5, 'tip')[['total_bill', 'tip', 'day']]
print(top5)

# Rank
tips['tip_rank'] = tips['tip'].rank(ascending=False).astype(int)
print(tips[['total_bill', 'tip', 'tip_rank']].head(5))
```

Output

```
     total_bill   tip  day
170       50.81 10.00  Sat
212       48.33  9.00  Sat
23        39.42  7.58  Sat
59        48.27  6.73  Sat
183       23.17  6.50  Sun

   total_bill   tip  tip_rank
0       16.99  1.01       224
1       10.34  1.66       188
2       21.01  3.50        52
3       23.68  3.31        63
4       24.59  3.61        46
```

2.10 String Operations

Python — .str accessor

```
names = pd.Series(['  Alice Dupont  ', 'bob martin', 'CHARLIE PETIT'])

print(names.str.strip().str.title())
print(names.str.contains('art', case=False))
```

Output

```
0     Alice Dupont
1       Bob Martin
2    Charlie Petit
dtype: object
0    False
1     True
2    False
```

```
dtype: bool
```

2.11 Exercises

Exercise 2.1 (⋆)**.** Load the `tips` dataset from Seaborn. Display the number of meals by day of the week and by time (lunch/dinner).

Exercise 2.2 (⋆)**.** Create a DataFrame with 5 students, their math and physics grades. Add an “average” column and sort by descending average.

Exercise 2.3 (⋆⋆)**.** On the Tips dataset, compute for each combination (sex, smoker): the number of meals, the average bill, and the average tip percentage.

Exercise 2.4 (⋆⋆)**.** Create two DataFrames: one with products (id, name, category), one with sales (product_id, quantity, date). Join them and compute revenue by category.

Exercise 2.5 (⋆⋆⋆)**.** Write a function that takes a DataFrame and returns an automatic report: number of rows, column types, percentage of missing values per column, number of duplicates.

Essential Pandas Functions

- `pd.read_csv()`, `df.to_csv()` — import/export
- `df.head()`, `df.info()`, `df.describe()` — exploration
- `df.loc[]`, `df.iloc[]` — selection
- `df.groupby().agg()` — aggregation
- `pd.merge()` — joins
- `df.sort_values()`, `df.nlargest()` — sorting

Chapter 3

Data Visualization

Intuition

Visualization is the data scientist's first tool. A good chart reveals structures, anomalies, and relationships that no table of numbers can show as clearly. As John Tukey said: "The greatest value of a picture is when it forces us to notice what we never expected to see."

3.1 The Grammar of Graphics

The **grammar of graphics**, proposed by Leland Wilkinson, decomposes any chart into layers:

1. **Data**: the source dataset.
2. **Aesthetics**: the variables mapped to axes, colors, sizes.
3. **Geometries**: the type of representation (points, bars, lines).
4. **Facets**: the partitioning into sub-plots.
5. **Statistics**: the transformations applied (counting, averaging).
6. **Coordinates**: the coordinate system (Cartesian, polar).
7. **Theme**: the general appearance (fonts, grid, background).

Seaborn and Matplotlib implement these concepts in Python.

3.2 Matplotlib: The Fundamentals

3.2.1 Line Plot

Python

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

```python
x = np.linspace(0, 2 * np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, y_sin, label='sin(x)', color='steelblue', linewidth=2)
ax.plot(x, y_cos, label='cos(x)', color='coral', linestyle='--')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Trigonometric Functions')
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('trigo.pdf', dpi=150)
plt.show()
```

Output

A plot with two curves (sine in blue, cosine in dashed coral) over $[0, 2\pi]$, with legend, grid, and titles.

3.2.2 Scatter Plot and Histogram

Python

```python
import seaborn as sns

tips = sns.load_dataset('tips')

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Scatter plot
axes[0].scatter(tips['total_bill'], tips['tip'],
                alpha=0.6, c='teal', edgecolors='white')
axes[0].set_xlabel('Total Bill (\$)')
axes[0].set_ylabel('Tip (\$)')
axes[0].set_title('Tip vs Total Bill')

# Histogram
axes[1].hist(tips['total_bill'], bins=20, color='steelblue',
             edgecolor='white')
axes[1].set_xlabel('Total Bill (\$)')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Distribution of Total Bills')

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

Output

Two sub-plots: on the left a scatter plot showing a positive relationship between bill and tip; on the right a slightly skewed histogram (right tail) of total bills.

3.2.3 Bar Chart

Python

```python
moyenne_jour = tips.groupby('day')['tip'].mean().sort_values()

fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(moyenne_jour.index, moyenne_jour.values,
       color=['#4c72b0', '#55a868', '#c44e52', '#8172b2'])
ax.set_xlabel('Day')
ax.set_ylabel('Average Tip (\$)')
ax.set_title('Average Tip by Day')
for i, v in enumerate(moyenne_jour.values):
    ax.text(i, v + 0.05, f'{v:.2f}', ha='center', fontsize=10)
plt.tight_layout()
plt.show()
```

Output

Colored bar chart with annotated values: Fri 2.73, Thur 2.77, Sat 2.99, Sun 3.26.

3.3 Seaborn: Statistical Visualization

3.3.1 Distributions: histplot and violinplot

Python

```python
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.histplot(data=tips, x='total_bill', hue='sex', kde=True,
             ax=axes[0], palette='Set2')
axes[0].set_title('Distribution by Sex')

sns.violinplot(data=tips, x='day', y='total_bill',
               hue='sex', split=True, ax=axes[1],
               palette='pastel')
axes[1].set_title('Violin Plot by Day and Sex')

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

Output

Left: overlapping histograms with KDE curves for male and female. Right: violin plots split by sex for each day, showing that males have slightly higher bills.

3.3.2 Box Plots and Heatmap

Python

```python
fig, axes = plt.subplots(1, 2, figsize=(13, 5))

sns.boxplot(data=tips, x='day', y='tip', hue='smoker',
            ax=axes[0], palette='coolwarm')
axes[0].set_title('Tips by Day and Smoker Status')

corr = tips[['total_bill', 'tip', 'size']].corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='YlOrRd',
            ax=axes[1], vmin=0, vmax=1)
axes[1].set_title('Correlation Matrix')

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

Output

Box plots showing similar distributions between smokers and non-smokers. Heatmap: correlation total_bill/tip = 0.68, total_bill/size = 0.60, tip/size = 0.49.

3.3.3 Pairplot and relplot

Python

```python
iris = sns.load_dataset('iris')

g = sns.pairplot(iris, hue='species', palette='husl',
                 diag_kind='kde', height=2.2)
g.figure.suptitle('Pairplot of the Iris Dataset', y=1.02)
plt.show()
```

Output

A 4×4 matrix of plots: diagonal = KDE densities by species, off-diagonal = scatter plots. *Setosa* is clearly distinguishable from the other two species.

Python

```python
sns.relplot(data=tips, x='total_bill', y='tip',
            hue='smoker', style='sex', col='time',
            height=4, aspect=1.2, palette='Set1')
plt.show()
```

Output

Two panels (Lunch / Dinner) with scatter plots colored by smoker status and differentiated by marker shape according to sex.

3.3.4 catplot

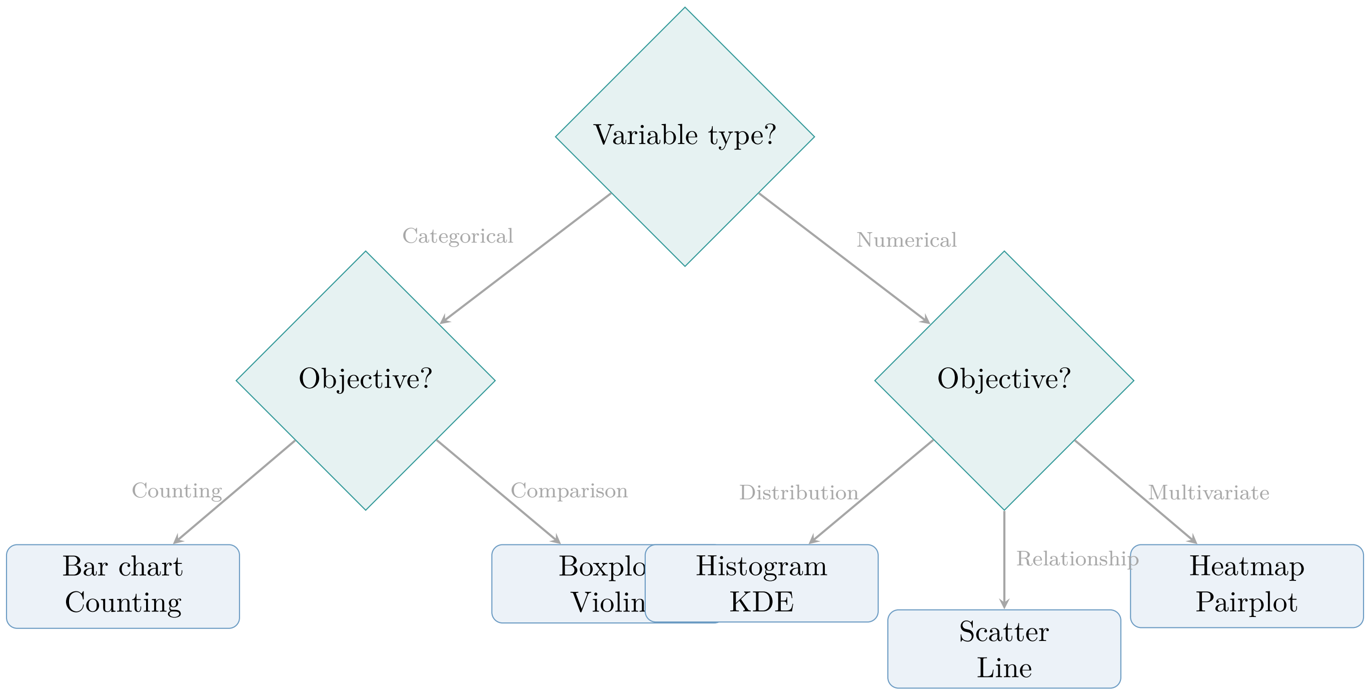
Python

```python
sns.catplot(data=tips, x='day', y='total_bill', hue='sex',
            kind='box', col='time', palette='muted',
            height=4, aspect=1)
plt.show()
```

Output

Four groups of box plots organized in two columns (Lunch, Dinner), compared by sex and by day.

3.4 Choosing the Right Chart Type



3.5 Color Palettes and Saving

Python

```python
# Palettes available in Seaborn
palettes = ['deep', 'muted', 'pastel', 'bright',
            'dark', 'colorblind']
sns.set_palette('colorblind')  # recommended for accessibility

# High-resolution saving
fig, ax = plt.subplots()
sns.histplot(tips['tip'], bins=15, ax=ax, color='steelblue')
fig.savefig('distribution_tip.png', dpi=300, bbox_inches='tight')
fig.savefig('distribution_tip.pdf', bbox_inches='tight')
```

Best Practice

Always use the `'colorblind'` palette to ensure the accessibility of your charts. Save in PDF for reports and in PNG (300 dpi) for presentations.

3.6 Common Visualization Mistakes

Warning

- **Truncated axes**: not starting the y-axis at zero in a bar chart exaggerates differences.
- **Misleading scales**: using two y-axes with different scales can suggest false

correlations.

- **3D charts**: perspective distorts the perception of values. Prefer 2D.
- **Too many categories**: a pie chart with 15 slices is unreadable. Use a bar chart instead.
- **Missing legend and titles**: every chart should be understandable on its own.

3.7 Exercises

Exercise 3.1 ($\star$)**.** Load the `tips` dataset and create a histogram of the `tip` variable with 20 bins. Add a title, axis labels, and a KDE curve overlay using Seaborn.

Exercise 3.2 ($\star$)**.** Create a horizontal bar chart showing the number of meals per day in the `tips` dataset. Sort the bars in descending order.

Exercise 3.3 ($\star\star$)**.** Using the `iris` dataset, create a figure with 4 sub-plots (2×2) showing the distribution of each numerical variable, colored by species. Use `sns.histplot` with `hue='species'`.

Exercise 3.4 ($\star\star$)**.** Create a `sns.catplot` of type `'violin'` showing the distribution of `total\_bill` by `day`, faceted by `time`, with color determined by `sex`. Interpret the observed differences.

Exercise 3.5 ($\star\star\star$)**.** Reproduce the following chart: a two-panel figure. The left panel shows a scatter plot of `total\_bill` vs `tip` with a regression line (`sns.regplot`). The right panel shows the residuals of this regression (compute them with NumPy: `np.polyfit` and `np.polyval`). Comment on the quality of the fit.

Exercise 3.6 ($\star\star\star$)**.** Take the `iris` dataset and create a heatmap of the correlation matrix *per species* (three heatmaps in a 1×3 figure). Compare the correlation structures across species.

Key Functions

Chapter Summary – Data Visualization

- **Matplotlib**: `plt.subplots()`, `ax.plot()`, `ax.scatter()`, `ax.hist()`, `ax.bar()`.
- **Seaborn (distributions)**: `sns.histplot()`, `sns.boxplot()`, `sns.violinplot()`.
- **Seaborn (relationships)**: `sns.scatterplot()`, `sns.relplot()`, `sns.heatmap()`.
- **Seaborn (categorical)**: `sns.catplot()`, `sns.countplot()`, `sns.barplot()`.
- **Seaborn (multivariate)**: `sns.pairplot()`.
- **Customization**: `set\_xlabel()`, `set\_title()`, `legend()`, `grid()`.

- **Saving**: `fig.savefig('name.pdf', dpi=300, bbox\_inches='tight')`.
- **Golden rule**: choose the chart suited to the variable type and the analysis objective.

Chapter 4

Exploratory Data Analysis (EDA)

Intuition

Exploratory Data Analysis (EDA) is an analysis philosophy initiated by John Tukey in the 1970s. Rather than testing predefined hypotheses, EDA invites us to "let the data speak" by systematically exploring them through numerical summaries and visualizations.

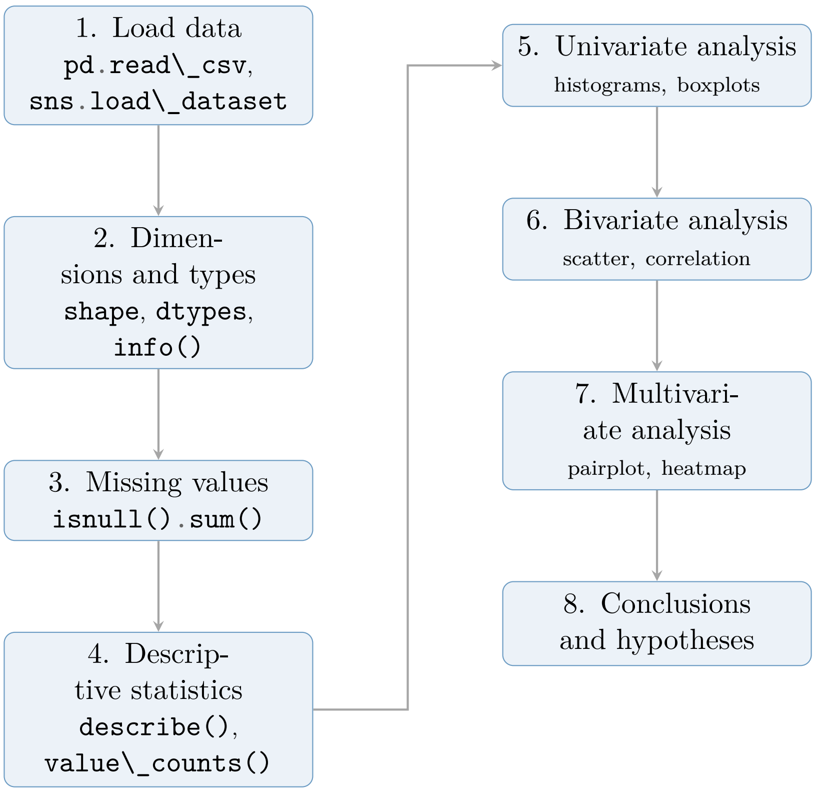
4.1 What is EDA?

Definition 4.1 (Exploratory Data Analysis)**.** EDA is an analytical approach that uses visual and quantitative techniques to understand the structure, anomalies, and relationships in a dataset *before* any formal modeling.

The main objectives of EDA are:

- Discover the **structure** of the data (dimensions, types, missing values).
- Identify **distributions** and **central tendencies**.
- Detect **anomalies** and **outliers**.
- Explore **relationships** between variables.
- Formulate **hypotheses** to be tested later.

4.2 Systematic EDA Process



4.3 Case Study: The Titanic Dataset

4.3.1 Loading and First Inspection

Python

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

titanic = sns.load_dataset('titanic')
print(f"Dimensions : {titanic.shape}")
print(titanic.dtypes)
```

Output

```
Dimensions : (891, 15)
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
```

```
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
```

4.3.2 Missing Values

Python

```python
missing = titanic.isnull().sum()
missing_pct = (missing / len(titanic) * 100).round(1)
missing_df = pd.DataFrame({'Missing': missing,
                           'Percentage': missing_pct})
print(missing_df[missing_df['Missing'] > 0])
```

Output

```
             Missing  Percentage
age              177        19.9
embarked           2         0.2
deck             688        77.2
embark_town        2         0.2
```

Remark 4.2. The variable `deck` has 77 % missing values: it will likely be unusable without heavy imputation. The variable `age` has about 20 % missing, which is manageable.

4.3.3 Descriptive Statistics

Python

```python
print(titanic.describe())
print("\n--- Categorical Variables ---")
print(titanic[['sex', 'embarked', 'class']].describe())
```

Output

```
         survived      pclass        age       sibsp       parch        fare
count  891.000000  891.000000  714.00000  891.00000  891.00000  891.00000
mean     0.383838    2.308642   29.69912    0.52301    0.38159   32.20421
std      0.486592    0.836071   14.52650    1.10274    0.80606   49.69343
```

```
min        0.000000    1.000000    0.42000    0.00000    0.00000    0.00000
25%        0.000000    2.000000   20.12500    0.00000    0.00000    7.91040
50%        0.000000    3.000000   28.00000    0.00000    0.00000   14.45420
75%        1.000000    3.000000   38.00000    1.00000    0.00000   31.00000
max        1.000000    3.000000   80.00000    8.00000    6.00000  512.32920

--- Categorical Variables ---
          sex embarked class
count     891      889   891
unique      2        3     3
top      male        S Third
freq      577      644   491
```

4.4 Univariate Analysis

Python

```python
fig, axes = plt.subplots(2, 2, figsize=(12, 9))

sns.histplot(titanic['age'].dropna(), bins=30, kde=True,
             ax=axes[0,0], color='steelblue')
axes[0,0].set_title('Age Distribution')

sns.countplot(data=titanic, x='pclass', ax=axes[0,1],
              palette='viridis')
axes[0,1].set_title('Distribution by Class')

sns.countplot(data=titanic, x='survived', ax=axes[1,0],
              palette='Set2')
axes[1,0].set_xticklabels(['Deceased', 'Survived'])
axes[1,0].set_title('Survival')

sns.histplot(titanic['fare'], bins=40, ax=axes[1,1],
             color='coral')
axes[1,1].set_title('Fare Distribution')

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

Output

Four plots: age approximately follows a normal distribution centered around 29 years; third class is the most represented (491); 62 % of passengers did not survive; the fare is highly skewed with a long right tail.

4.5 Bivariate Analysis

4.5.1 Survival by Sex and by Class

Python

```python
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Survival by sex
sns.barplot(data=titanic, x='sex', y='survived',
            ax=axes[0], palette='coolwarm', ci=95)
axes[0].set_title('Survival Rate by Sex')
axes[0].set_ylabel('Survival Rate')

# Survival by class
sns.barplot(data=titanic, x='pclass', y='survived',
            ax=axes[1], palette='viridis', ci=95)
axes[1].set_title('Survival Rate by Class')

# Survival by sex and class
sns.barplot(data=titanic, x='pclass', y='survived',
            hue='sex', ax=axes[2], palette='Set1', ci=95)
axes[2].set_title('Survival by Class and Sex')

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

Output

Women have a survival rate of 74 % compared to 19 % for men. First class has 63 % survival, second class 47 %, and third class 24 %. The combination shows that first-class women survived at nearly 97 %.

4.5.2 Correlations and Contingency Table

Python

```python
# Contingency table
ct = pd.crosstab(titanic['sex'], titanic['survived'],
                 margins=True)
ct.columns = ['Deceased', 'Survived', 'Total']
print(ct)

# Numerical correlation
cols_num = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
corr = titanic[cols_num].corr()
print("\nCorrelation Matrix:")
print(corr.round(2))
```

CHAPTER 4. EXPLORATORY DATA ANALYSIS (EDA)

Output

```
         Deceased  Survived  Total
sex
female        81       233    314
male         468       109    577
Total        549       342    891

Correlation Matrix:
          survived  pclass   age  sibsp  parch   fare
survived      1.00   -0.34 -0.08  -0.04   0.08   0.26
pclass       -0.34    1.00 -0.37   0.08   0.02  -0.55
age          -0.08   -0.37  1.00  -0.31  -0.19   0.10
fare          0.26   -0.55  0.10  -0.24   0.22   1.00
```

4.6 Multivariate Analysis

Python

```python
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(corr, annot=True, fmt='.2f', cmap='RdBu_r',
            center=0, ax=ax, square=True,
            linewidths=0.5)
ax.set_title('Correlation Heatmap -- Titanic')
plt.tight_layout()
plt.show()
```

Output

Heatmap showing correlations: survival is negatively correlated with class (-0.34) and positively with fare (0.26). Class and fare are strongly anti-correlated (-0.55).

4.7 Outlier Detection

Definition 4.3 (IQR Method)**.** An observation is considered an outlier if it falls outside the interval:

$$[Q_1 - 1.5 \times \text{IQR},\ Q_3 + 1.5 \times \text{IQR}]$$

where $\text{IQR} = Q_3 - Q_1$ is the interquartile range.

Python

```python
def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) |
                  (df[column] > upper_bound)]
    return outliers, lower_bound, upper_bound

outliers_fare, b_low, b_high = detect_outliers_iqr(titanic, 'fare')
print(f"Bounds: [{b_low:.2f}, {b_high:.2f}]")
print(f"Number of outliers (fare): {len(outliers_fare)}")
print(f"Max fare: {titanic['fare'].max():.2f}")
```

Output

```
Bounds: [-26.72, 65.63]
Number of outliers (fare): 116
Max fare: 512.33
```

Remark 4.4. 116 passengers (13 %) have a fare considered as an outlier by the IQR method. These are mainly first-class passengers. They should not be automatically removed: these are legitimate values that reflect the data structure.

4.8 Exercises

Exercise 4.1 (⋆). Load the `titanic` dataset and display the first 5 rows, the dimensions, the column types, and the number of missing values per column.

Exercise 4.2 (⋆). Compute the overall survival rate, then the survival rate by port of embarkation (`embarked`). Which port letter has the highest survival rate?

Exercise 4.3 (⋆⋆). Create a chart with `sns.catplot` of type `'bar'` showing the survival rate by class and by sex. Add an informative title. Interpret the results in terms of evacuation priority.

Exercise 4.4 (⋆⋆). Create a histogram of passenger age, separated by survival (`hue='survived'`). Are survivors younger on average? Verify with `groupby('survived')['age'].mean()`.

Exercise 4.5 (⋆⋆⋆). Apply the IQR method to detect outliers in the `age` variable. How many are there? Create an annotated box plot showing the IQR bounds and the outliers. Discuss the relevance of removing these observations.

Exercise 4.6 (⋆⋆⋆). Perform a complete EDA of the `penguins` dataset (`sns.load\_dataset('penguins')`). Your analysis should include: (a) general inspection, (b) missing value handling, (c) univariate distributions, (d) bivariate analysis by species, (e) correlation heatmap. Conclude by formulating 3 testable hypotheses.

Key Functions

Chapter Summary – Exploratory Data Analysis

- **Inspection**: `df.shape`, `df.dtypes`, `df.info()`, `df.head()`.

CHAPTER 4. EXPLORATORY DATA ANALYSIS (EDA)

- **Missing values**: `df.isnull().sum()`, `df.isnull().mean() * 100`.
- **Descriptive**: `df.describe()`, `df['col'].value\_counts()`.
- **Univariate**: `sns.histplot()`, `sns.countplot()`, `sns.boxplot()`.
- **Bivariate**: `sns.barplot(x, y)`, `sns.scatterplot()`, `pd.crosstab()`.
- **Multivariate**: `df.corr()`, `sns.heatmap()`, `sns.pairplot()`.
- **Outliers (IQR)**: outlier if $x < Q_1 - 1.5 \cdot \text{IQR}$ or $x > Q_3 + 1.5 \cdot \text{IQR}$.
- **Tukey's philosophy**: explore first, model later.

Chapter 5

Data Cleaning and Preparation

Intuition

It is estimated that data scientists spend about 80 % of their time cleaning and preparing data. Poorly prepared data inevitably lead to flawed models: “Garbage in, garbage out.”

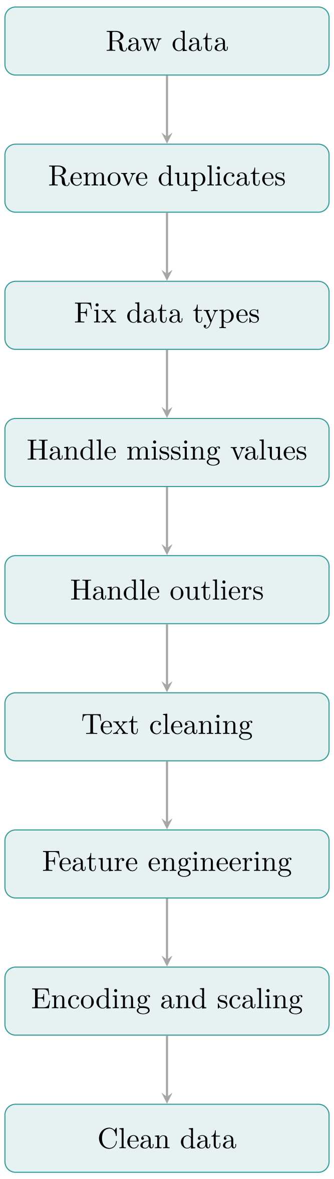
5.1 Why Clean Data?

Real-world data are rarely clean. Common issues include:

- **Missing values** (faulty sensors, unfilled fields);
- **Duplicates** (multiple entries, table merges);
- **Incorrect types** (numbers stored as text);
- **Outliers** (data entry errors);
- **Textual inconsistencies** (capitalization, spaces, abbreviations).

CHAPTER 5. DATA CLEANING AND PREPARATION

5.2 Cleaning Pipeline



5.3 Case Study: The Titanic Dataset

Python

```python
import pandas as pd
import numpy as np
import seaborn as sns

titanic = sns.load_dataset('titanic')
print(f"Initial dimensions: {titanic.shape}")
print(titanic.info())
```

Output

```
Initial dimensions: (891, 15)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 ...
 7   embarked     889 non-null    object
 11  deck         203 non-null    category
```

5.4 Missing Values

5.4.1 Types of Missing Data

Definition 5.1 (Classification of Missing Values).

- **MCAR** (*Missing Completely At Random*): the missingness does not depend on any variable.
- **MAR** (*Missing At Random*): the missingness depends on other observed variables.
- **MNAR** (*Missing Not At Random*): the missingness depends on the value itself.

Example 5.2. In the Titanic dataset, cabins (`deck`) are missing mainly for third-class passengers (MAR). Age could be MCAR or MAR depending on whether passengers without a full ticket had no recorded age.

5.4.2 Detection and Treatment

Python

```python
# Detection
print(titanic.isnull().sum().sort_values(ascending=False).head(5))

# Strategy 1: drop the deck column (too many missing values)
titanic_clean = titanic.drop(columns=['deck'])

# Strategy 2: fill age with median by class
titanic_clean['age'] = titanic_clean.groupby('pclass')['age'] \
    .transform(lambda x: x.fillna(x.median()))

# Strategy 3: fill embarked with the mode
titanic_clean['embarked'].fillna(
    titanic_clean['embarked'].mode()[0], inplace=True)
titanic_clean['embark_town'].fillna(
    titanic_clean['embark_town'].mode()[0], inplace=True)
```

CHAPTER 5. DATA CLEANING AND PREPARATION

```python
print(f"\nRemaining missing values:
↪  {titanic_clean.isnull().sum().sum()}")
```

Output

```
deck           688
age            177
embarked         2
embark_town      2
survived         0

Remaining missing values: 0
```

Best Practice

Imputation by *conditional* median (here by class) is preferable to using the global median because it preserves the structure of subgroups. For time series data, use `ffill()` or `bfill()`.

5.5 Duplicates

Python

```python
# Detection
n_dup = titanic_clean.duplicated().sum()
print(f"Number of duplicates: {n_dup}")

# Inspect potential duplicates
if n_dup > 0:
    print(titanic_clean[titanic_clean.duplicated(keep=False)]
          .sort_values(by='name').head())
    titanic_clean = titanic_clean.drop_duplicates()
    print(f"After removal: {titanic_clean.shape}")
```

Output

```
Number of duplicates: 0
```

5.6 Type Conversion

Python

```python
# Convert survived to boolean
titanic_clean['survived'] = titanic_clean['survived'].astype(bool)

# Convert pclass to ordered categorical
titanic_clean['pclass'] = pd.Categorical(
    titanic_clean['pclass'], categories=[1, 2, 3], ordered=True)

# Example of date conversion (with dummy data)
dates_str = pd.Series(['2023-01-15', '2023-02-20', '2023-03-10'])
dates = pd.to_datetime(dates_str)
print(dates.dt.month)
```

Output

```
0    1
1    2
2    3
dtype: int32
```

5.7 Handling Outliers

Python

```python
# IQR method for fare
Q1 = titanic_clean['fare'].quantile(0.25)
Q3 = titanic_clean['fare'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Option 1: capping/winsorizing
titanic_clean['fare_capped'] = titanic_clean['fare'].clip(
    lower=lower_bound, upper=upper_bound)

print(f"Before: max = {titanic_clean['fare'].max():.2f}")
print(f"After capping: max = {titanic_clean['fare_capped'].max():.2f}")
print(f"Upper bound: {upper_bound:.2f}")
```

Output

```
Before: max = 512.33
After capping: max = 65.63
Upper bound: 65.63
```

Warning

Never remove outliers without careful thought. A fare of $512 may be legitimate (luxury suite). Capping is often preferable to removal. Always document your choices.

5.8 Text Cleaning

Python

```python
# Example with textual data
names = pd.Series(['  Alice ', 'BOB', 'charlie', ' Dave  '])
print("Before:", names.tolist())

names_clean = (names.str.strip()
                    .str.lower()
                    .str.capitalize())
print("After:", names_clean.tolist())

# Pattern replacement
cities = pd.Series(['New-York', 'new york', 'NEW YORK', 'N.Y.'])
cities_norm = cities.str.lower().str.replace(
    r'[^a-z\s]', '', regex=True).str.strip()
print("Normalized cities:", cities_norm.tolist())
```

Output

```
Before: ['  Alice ', 'BOB', 'charlie', ' Dave  ']
After: ['Alice', 'Bob', 'Charlie', 'Dave']
Normalized cities: ['newyork', 'new york', 'new york', 'ny']
```

5.9 Feature Engineering

Python

```python
# Create new variables from existing ones
titanic_clean['family_size'] = (titanic_clean['sibsp']
                                + titanic_clean['parch'] + 1)
titanic_clean['is_child'] = titanic_clean['age'] < 18
```

```python
titanic_clean['fare_per_person'] = (titanic_clean['fare']
                                    / titanic_clean['family_size'])

# Discretize age into categories
titanic_clean['age_group'] = pd.cut(
    titanic_clean['age'],
    bins=[0, 12, 18, 35, 60, 100],
    labels=['Child', 'Teen', 'Young Adult', 'Adult', 'Senior'])

print(titanic_clean['age_group'].value_counts())
```

Output

```
Young Adult    371
Adult          267
Child           89
Teen            79
Senior          85
Name: age_group, dtype: int64
```

5.10 Encoding Categorical Variables

Python

```python
# One-hot encoding with pandas
embarked_dummies = pd.get_dummies(
    titanic_clean['embarked'], prefix='embarked', dtype=int)
print(embarked_dummies.head())

# Label encoding with sklearn
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
titanic_clean['sex_encoded'] = le.fit_transform(
    titanic_clean['sex'])
print("\nMapping:", dict(zip(le.classes_,
                              le.transform(le.classes_))))
```

Output

```
   embarked_C  embarked_Q  embarked_S
0           0           0           1
1           1           0           0
2           0           0           1
3           0           0           1
4           0           0           1
```

CHAPTER 5. DATA CLEANING AND PREPARATION

```
Mapping: {'female': 0, 'male': 1}
```

Warning

Label encoding imposes an artificial order on categories. It is only suitable for ordinal variables or tree-based algorithms. For linear models, prefer **one-hot encoding**.

5.11 Feature Scaling

Definition 5.3 (Standardization and Normalization).

- **Standardization** (StandardScaler): $z = \frac{x - \mu}{\sigma}$, resulting in mean 0 and standard deviation 1.
- **Normalization** (MinMaxScaler): $x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$, resulting in values in $[0, 1]$.

Python

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

cols = ['age', 'fare']
data_num = titanic_clean[cols].copy()

# Standardization
scaler_std = StandardScaler()
data_std = pd.DataFrame(scaler_std.fit_transform(data_num),
                        columns=[c + '_std' for c in cols])

# Normalization
scaler_mm = MinMaxScaler()
data_norm = pd.DataFrame(scaler_mm.fit_transform(data_num),
                         columns=[c + '_norm' for c in cols])

print("Standardized:")
print(data_std.describe().loc[['mean', 'std']].round(2))
print("\nNormalized:")
print(data_norm.describe().loc[['min', 'max']].round(2))
```

Output

```
Standardized:
      age_std  fare_std
mean     0.00      0.00
std      1.00      1.00

Normalized:
      age_norm  fare_norm
min       0.00       0.00
max       1.00       1.00
```

Best Practice

Apply scaling *after* the train/test split to avoid information leakage (*data leakage*). Use `fit\_transform()` on the training set and `transform()` on the test set.

5.12 Exercises

Exercise 5.1 (⋆). Load the `titanic` dataset. Display the number and percentage of missing values per column. Which columns have more than 50 % missing values?

Exercise 5.2 (⋆). Remove duplicates from the `titanic` dataset (if any). Then convert the `sex` column to a categorical variable with `pd.Categorical()`.

Exercise 5.3 (⋆⋆). Impute the missing values of `age` using the median conditioned on `sex` and `pclass` (6 groups). Compare the age distribution before and after imputation with overlapping histograms.

Exercise 5.4 (⋆⋆). Create the following variables: `title` (extracted from the name using a regular expression), `is\_alone` (true if `family\_size` = 1), `fare\_bin` (fare quartiles with `pd.qcut`). Compute the survival rate for each of these new variables.

Exercise 5.5 (⋆⋆⋆). Build a complete cleaning pipeline for the `titanic` dataset: (a) drop `deck`, (b) impute `age` and `embarked`, (c) create `family\_size` and `is\_child`, (d) one-hot encode `sex`, `embarked`, `class`, (e) standardize numerical variables. The final result should be a fully numerical DataFrame with no missing values.

Exercise 5.6 (⋆⋆⋆). Load the `penguins` dataset (`sns.load\_dataset('penguins')`). This dataset contains missing values. Apply three different strategies (deletion, median imputation, median imputation conditioned by species). Compare the descriptive statistics obtained with each strategy and discuss the best approach.

Key Functions

Chapter Summary – Data Cleaning and Preparation

- **Missing values**: `isnull().sum()`, MCAR/MAR/MNAR types, `fillna()`, `dropna()`.
- **Duplicates**: `duplicated()`, `drop\_duplicates()`.
- **Types**: `astype()`, `pd.to\_datetime()`, `pd.Categorical()`.
- **Outliers**: IQR method, capping with `clip()`.
- **Text**: `str.strip()`, `str.lower()`, `str.replace()`.
- **Feature engineering**: `pd.cut()`, `pd.qcut()`, column combination.

CHAPTER 5. DATA CLEANING AND PREPARATION

- **Encoding**: `pd.get\_dummies()` (one-hot), `LabelEncoder` (ordinal).
- **Scaling**: $z = \frac{x-\mu}{\sigma}$ (StandardScaler), $x' = \frac{x-x_{\text{min}}}{x_{\text{max}}-x_{\text{min}}}$ (MinMaxScaler).
- **Golden rule**: document every transformation and avoid *data leakage*.

Chapter 6

Applied Statistics

Intuition

Statistics provide the mathematical framework for moving from observation to inference. In data science, they are used to quantify uncertainty, test hypotheses, and measure relationships between variables.

6.1 Descriptive Statistics

6.1.1 Measures of Central Tendency and Dispersion

Definition 6.1 (Fundamental Measures)**.** For a sample $(x_1, x_2, \ldots, x_n)$:

- **Mean**: $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$
- **Median**: the value such that 50 % of observations are below it.
- **Mode**: the most frequent value.
- **Variance**: $s^2 = \dfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- **Standard deviation**: $s = \sqrt{s^2}$
- **Quantiles**: Q_1 (25 %), Q_2 (50 %), Q_3 (75 %).

Python

```python
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats

tips = sns.load_dataset('tips')

col = tips['total_bill']
print(f"Mean        : {col.mean():.2f}")
print(f"Median      : {col.median():.2f}")
print(f"Mode        : {col.mode()[0]:.2f}")
print(f"Variance    : {col.var():.2f}")
print(f"Std dev     : {col.std():.2f}")
print(f"Q1, Q3      : {col.quantile(0.25):.2f}, "
      f"{col.quantile(0.75):.2f}")
print(f"Skewness    : {col.skew():.2f}")
print(f"Kurtosis    : {col.kurtosis():.2f}")
```

Output

```
Mean         : 19.79
Median       : 17.80
Mode         : 13.42
Variance     : 79.25
Std dev      : 8.90
Q1, Q3       : 13.35, 24.13
Skewness     : 1.13
Kurtosis     : 1.22
```

Remark 6.2. The positive skewness (1.13) and the mean being greater than the median confirm a right-skewed distribution. The positive kurtosis indicates heavier tails than the normal distribution.

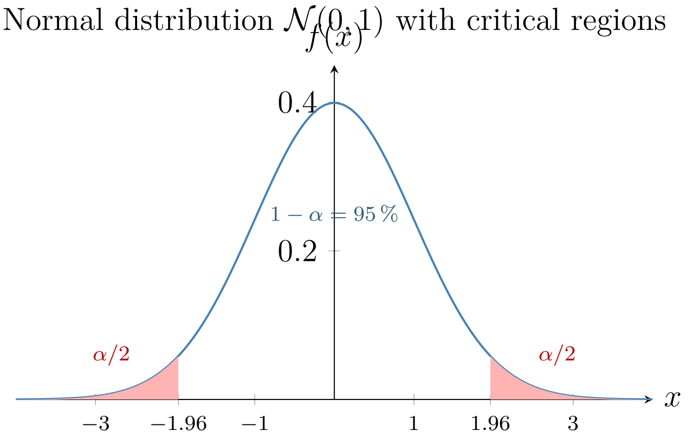
6.2 Probability Distributions

6.2.1 Normal Distribution

Definition 6.3 (Normal Distribution). A random variable X follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ if its density is:

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

Normal distribution $\mathcal{N}(0, 1)$ with critical regions



Python

```python
from scipy.stats import norm
import matplotlib.pyplot as plt

# Normal distribution: probabilities and quantiles
print(f"P(X < 1.96) = {norm.cdf(1.96):.4f}")
print(f"P(-1.96 < X < 1.96) = {norm.cdf(1.96) - norm.cdf(-1.96):.4f}")
print(f"97.5% quantile = {norm.ppf(0.975):.4f}")

# Visualization
x = np.linspace(-4, 4, 200)
fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, norm.pdf(x), 'steelblue', lw=2, label='$\mathcal{N}(0,1)$')
ax.fill_between(x, norm.pdf(x), where=(x > 1.96) | (x < -1.96),
                color='red', alpha=0.3, label='Critical regions')
ax.legend()
ax.set_title('Standard Normal Distribution')
plt.show()
```

Output

```
P(X < 1.96) = 0.9750
P(-1.96 < X < 1.96) = 0.9500
97.5% quantile = 1.9600
```

6.2.2 Binomial and Poisson Distributions

Python

```python
from scipy.stats import binom, poisson

# Binomial: n=20 trials, p=0.5
print("--- Binomial distribution B(20, 0.5) ---")
print(f"P(X = 10) = {binom.pmf(10, n=20, p=0.5):.4f}")
print(f"P(X <= 10) = {binom.cdf(10, n=20, p=0.5):.4f}")
print(f"Expected value = {binom.mean(n=20, p=0.5):.1f}")
print(f"Variance = {binom.var(n=20, p=0.5):.1f}")

# Poisson: lambda=5 events per interval
print("\n--- Poisson distribution P(5) ---")
print(f"P(X = 3) = {poisson.pmf(3, mu=5):.4f}")
print(f"P(X <= 3) = {poisson.cdf(3, mu=5):.4f}")
```

Output

```
--- Binomial distribution B(20, 0.5) ---
```

```
P(X = 10) = 0.1762
P(X <= 10) = 0.5881
Expected value = 10.0
Variance = 5.0

--- Poisson distribution P(5) ---
P(X = 3) = 0.1404
P(X <= 3) = 0.2650
```

6.3 Confidence Intervals

Definition 6.4 (Confidence Interval)**.** A $(1-\alpha)\times 100\,\%$ confidence interval for the mean μ is:

$$\text{CI} = \left[\bar{x} - z_{\alpha/2}\frac{s}{\sqrt{n}},\ \bar{x} + z_{\alpha/2}\frac{s}{\sqrt{n}}\right]$$

where $z_{\alpha/2}$ is the quantile of the normal distribution (1.96 for 95 %).

Python

```python
# 95% CI for the mean tip
tip = tips['tip']
n = len(tip)
mean = tip.mean()
se = tip.std() / np.sqrt(n)

ci_low = mean - 1.96 * se
ci_high = mean + 1.96 * se
print(f"Mean: {mean:.3f}")
print(f"95% CI: [{ci_low:.3f}, {ci_high:.3f}]")

# With scipy (exact method, t-distribution)
ci = stats.t.interval(confidence=0.95, df=n-1,
                      loc=mean, scale=se)
print(f"95% CI (t-Student): [{ci[0]:.3f}, {ci[1]:.3f}]")
```

Output

```
Mean: 2.998
95% CI: [2.823, 3.173]
95% CI (t-Student): [2.822, 3.175]
```

6.4 Hypothesis Testing

Definition 6.5 (Hypothesis Test)**.** A hypothesis test opposes:

- H_0 (null hypothesis): no effect, no difference.

- H_1 (alternative hypothesis): there is an effect or a difference.

The **p-value** is the probability of observing a result at least as extreme as the one observed, under H_0. We reject H_0 if $p < \alpha$ (typically $\alpha = 0.05$).

6.4.1 Student's t-Test

Python

```
# Do men tip differently than women?
tip_male = tips[tips['sex'] == 'Male']['tip']
tip_female = tips[tips['sex'] == 'Female']['tip']

t_stat, p_value = stats.ttest_ind(tip_male, tip_female)
print(f"t-statistic: {t_stat:.4f}")
print(f"p-value: {p_value:.4f}")
print(f"Male mean: {tip_male.mean():.3f}")
print(f"Female mean: {tip_female.mean():.3f}")

if p_value < 0.05:
    print("=> Significant difference (reject H0)")
else:
    print("=> No significant difference")
```

Output

```
t-statistic: 1.3879
p-value: 0.1665
Male mean: 3.090
Female mean: 2.833
=> No significant difference
```

6.4.2 χ^2 Test of Independence

Python

```
# Is smoker status related to meal time?
titanic = sns.load_dataset('titanic')

# Test on Titanic: survival vs sex
ct = pd.crosstab(titanic['sex'], titanic['survived'])
print("Contingency table:")
print(ct)

chi2, p_value, dof, expected = stats.chi2_contingency(ct)
print(f"\nChi2 = {chi2:.2f}")
print(f"p-value = {p_value:.2e}")
print(f"Degrees of freedom = {dof}")
print(f"=> {'Significant dependence' if p_value < 0.05 else
↪  'Independence'}")
```

Output

```
Contingency table:
survived    0    1
sex
female     81  233
male      468  109

Chi2 = 260.72
p-value = 1.20e-58
Degrees of freedom = 1
=> Significant dependence
```

Remark 6.6. The χ^2 test confirms with enormous statistical significance ($p \approx 10^{-58}$) that survival strongly depended on sex. This is the most striking result from the Titanic: “Women and children first.”

6.5 Correlations

Definition 6.7 (Correlation Coefficients).

- **Pearson**: measures *linear* correlation:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \cdot \sum(y_i - \bar{y})^2}}$$

- **Spearman**: measures *monotonic* correlation (based on ranks).

Both coefficients lie in $[-1, 1]$. $|r| > 0.7$ indicates a strong correlation.

Python

```
# Correlation between bill and tip
r_pearson, p_pearson = stats.pearsonr(tips['total_bill'],
                                       tips['tip'])
r_spearman, p_spearman = stats.spearmanr(tips['total_bill'],
                                          tips['tip'])

print(f"Pearson  : r = {r_pearson:.4f}, p = {p_pearson:.2e}")
print(f"Spearman : r = {r_spearman:.4f}, p = {p_spearman:.2e}")
```

Output

```
Pearson  : r = 0.6757, p = 6.69e-34
Spearman : r = 0.6787, p = 2.83e-34
```

Warning

Correlation does not imply causation. A strong correlation between two variables may be due to a confounding variable. Always complement with theoretical analysis and, if possible, an experimental design.

6.6 One-Way ANOVA

Definition 6.8 (ANOVA)**.** Analysis of variance (ANOVA) tests whether the means of k groups are equal:

- H_0: $\mu_1 = \mu_2 = \cdots = \mu_k$
- H_1: at least two means differ.

The F-statistic compares the between-group variance to the within-group variance.

Python

```python
# Does the tip differ by day?
days = tips['day'].unique()
groups = [tips[tips['day'] == d]['tip'] for d in days]

f_stat, p_value = stats.f_oneway(*groups)
print(f"F-statistic: {f_stat:.4f}")
print(f"p-value: {p_value:.4f}")

# Means by day
print("\nMean tip by day:")
print(tips.groupby('day')['tip'].agg(['mean', 'std', 'count'])
      .round(3))
```

Output

```
F-statistic: 1.6724
p-value: 0.1736

Mean tip by day:
       mean    std  count
day
Thur  2.771  1.240     62
Fri   2.735  1.018     19
Sat   2.993  1.631     87
Sun   3.256  1.227     76
```

Remark 6.9. ANOVA does not detect a significant difference ($p = 0.17$). Although Sunday has the highest mean, the within-group variability is too large to conclude.

6.7 Integrative Example: Titanic Analysis

Python

```python
# 1. Does the fare differ by class? (ANOVA)
groups_fare = [titanic[titanic['pclass'] == c]['fare']
                for c in [1, 2, 3]]
f, p = stats.f_oneway(*groups_fare)
print(f"ANOVA fare ~ class: F={f:.2f}, p={p:.2e}")

# 2. Survival vs class? (Chi-squared)
ct2 = pd.crosstab(titanic['pclass'], titanic['survived'])
chi2, p2, _, _ = stats.chi2_contingency(ct2)
print(f"Chi2 survival ~ class: chi2={chi2:.2f}, p={p2:.2e}")

# 3. Correlation age-fare
age_clean = titanic.dropna(subset=['age'])
r, p3 = stats.pearsonr(age_clean['age'], age_clean['fare'])
print(f"Pearson age ~ fare: r={r:.3f}, p={p3:.4f}")
```

Output

```
ANOVA fare ~ class: F=242.34, p=2.00e-84
Chi2 survival ~ class: chi2=102.89, p=4.55e-23
Pearson age ~ fare: r=0.096, p=0.0101
```

Proposition 6.10. *The results show that (1) the fare differs very significantly between classes, (2) survival strongly depends on class, and (3) age and fare have a very weak correlation although statistically significant (the large sample size makes small effects significant).*

6.8 Exercises

Exercise 6.1 ($\star$)**.** Compute the mean, median, standard deviation, and quartiles of the `tip` variable from the `tips` dataset. Is the distribution symmetric? Justify using the skewness coefficient.

Exercise 6.2 ($\star$)**.** Generate 1000 samples of size 30 from a $\mathcal{N}(50, 10^2)$ distribution. For each sample, compute the 95 % CI. How many intervals actually contain $\mu = 50$? Is the result consistent with theory?

Exercise 6.3 ($\star\star$)**.** Use a t-test to determine whether the mean tip differs between lunch (`time='Lunch'`) and dinner (`time='Dinner'`). Interpret the p-value and compute the effect size (difference in means divided by the pooled standard deviation).

Exercise 6.4 ($\star\star$)**.** Perform a χ^2 test to test the independence between `smoker` and `time` in the `tips` dataset. Display the contingency table, the expected frequencies, and draw a conclusion.

Exercise 6.5 ($\star\star\star$)**.** On the `titanic` dataset, perform the following tests and summarize the results in a table: (a) t-test on age between survivors and non-survivors, (b) χ^2 on survival vs port of embarkation, (c) ANOVA on fare by port of embarkation, (d) Spearman correlation between `pclass` and `survived`. Apply the Bonferroni correction ($\alpha_{\text{corrected}} = 0.05/4 = 0.0125$) and conclude.

Exercise 6.6 ($\star\star\star$)**.** Simulate the power of a t-test: generate two groups of size n from $\mathcal{N}(0, 1)$ and $\mathcal{N}(\delta, 1)$ with $\delta \in \{0.2, 0.5, 0.8\}$ and $n \in \{10, 30, 100, 300\}$. For each combination, repeat 1000 times and compute the percentage of H_0 rejections. Plot the power as a function of n for each δ.

Key Functions

Chapter Summary – Applied Statistics

- **Descriptive**: `mean()`, `median()`, `std()`, `var()`, `quantile()`, `skew()`.
- **Distributions**: `norm`, `binom`, `poisson` from `scipy.stats`; methods `.pdf()`, `.cdf()`, `.ppf()`.
- **95 % CI**: $\bar{x} \pm 1.96 \cdot \frac{s}{\sqrt{n}}$; exact with `stats.t.interval()`.
- **t-test**: `stats.ttest\_ind(a, b)` – compares two independent means.
- **χ^2 test**: `stats.chi2\_contingency(ct)` – independence of two categorical variables.
- **Correlation**: `stats.pearsonr(x, y)`, `stats.spearmanr(x, y)`, $r \in [-1, 1]$.
- **ANOVA**: `stats.f\_oneway(*groups)` – compares $k \geq 3$ means.
- **Decision**: reject H_0 if $p < \alpha$ (typically $\alpha = 0.05$).
- **Caution**: correlation $\neq$ causation; statistical significance $\neq$ practical importance.

Chapter 7

Introduction to Machine Learning

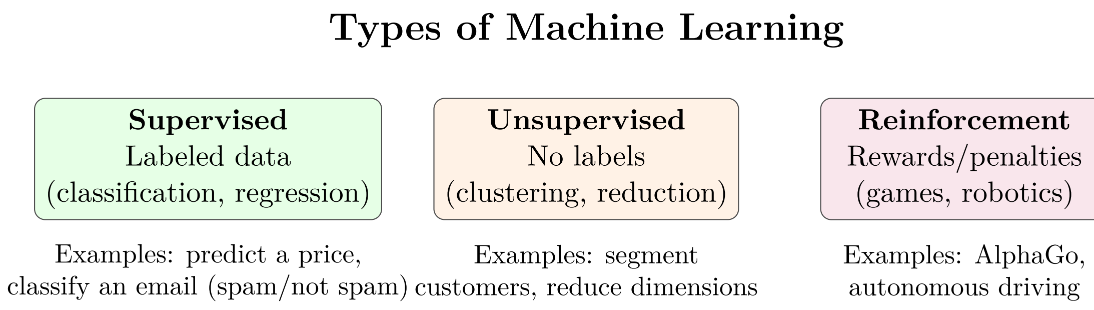
7.1 What is Machine Learning?

Definition 7.1 (Machine Learning)**. Machine learning** (ML) is a branch of artificial intelligence that enables computers to learn from data without being explicitly programmed for each task.

Intuition

Rather than writing rules manually (for example, “if the customer is over 30 years old and has an income above \$50,000, then approve the loan”), we provide **examples** to the model, which discovers the underlying rules on its own.

7.1.1 Types of Learning



Types of Machine Learning
Supervised
Labeled data
(classification, regression)
Unsupervised
No labels
(clustering, reduction)
Reinforcement
Rewards/penalties
(games, robotics)
Examples: predict a price, classify an email (spam/not spam)
Examples: segment customers, reduce dimensions
Examples: AlphaGo, autonomous driving

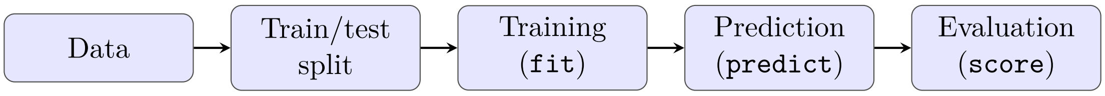
Definition 7.2 (Supervised Learning)**.** We have pairs $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ are the **explanatory variables** (*features*) and y_i is the **target**. The model learns a function f such that $f(\mathbf{x}_i) \approx y_i$.

- **Classification**: y is a category (spam/not spam, flower species).
- **Regression**: y is a continuous value (price, temperature).

Definition 7.3 (Unsupervised Learning)**.** We only have $\mathbf{x}_i$ without labels. The model seeks **hidden structures** in the data (groups, principal axes).

CHAPTER 7. INTRODUCTION TO MACHINE LEARNING

7.2 The Supervised Learning Workflow



1. **Collect and prepare** the data.
2. **Split** into a training set and a test set.
3. **Train** the model on the training data.
4. **Predict** on the test data.
5. **Evaluate** the performance.

Python

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Load the data
iris = load_iris()
X, y = iris.data, iris.target

# Split: 70% training, 30% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
print(f"Training: {X_train.shape[0]} samples")
print(f"Test: {X_test.shape[0]} samples")
```

Output

```
Training: 105 samples
Test: 45 samples
```

7.3 The Scikit-learn API

Best Practice

All scikit-learn models follow the same interface:

1. `model = Class(parameters)` — create the model.
2. `model.fit(X_train, y_train)` — train.
3. `model.predict(X_test)` — predict.
4. `model.score(X_test, y_test)` — evaluate.

7.3.1 Complete Example: k-Nearest Neighbors on Iris

Python

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Create the KNN model with k=5
knn = KNeighborsClassifier(n_neighbors=5)

# Training
knn.fit(X_train, y_train)

# Prediction
y_pred = knn.predict(X_test)

# Evaluation
acc = accuracy_score(y_test, y_pred)
print(f"Predictions: {y_pred[:10]}")
print(f"True values: {y_test[:10]}")
print(f"Accuracy:    {acc:.4f}")
```

Output

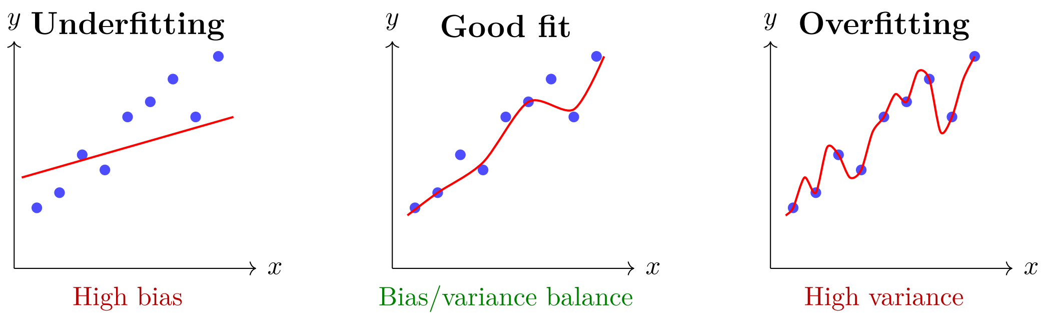
```
Predictions: [1 0 2 1 1 0 1 2 1 1]
True values: [1 0 2 1 1 0 1 2 1 1]
Accuracy:    1.0000
```

Remark 7.4. The `score()` method returns the **accuracy** for classification and the **coefficient of determination** R^2 for regression.

7.4 Overfitting and Underfitting

Definition 7.5 (Overfitting and Underfitting).

- **Underfitting**: the model is too simple and fails to capture the structure of the data. **High bias**.
- **Overfitting**: the model is too complex and memorizes the noise in the training data. **High variance**.

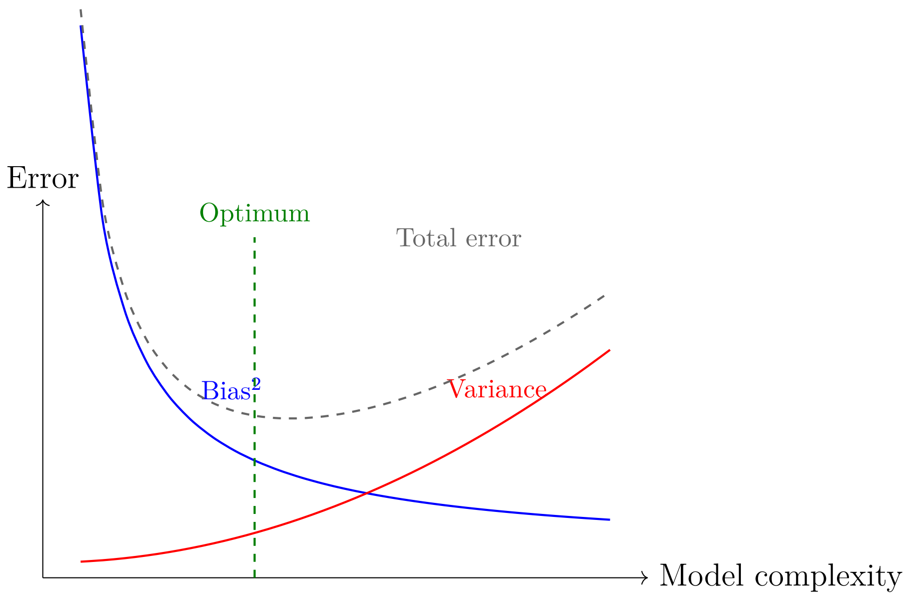


CHAPTER 7. INTRODUCTION TO MACHINE LEARNING

Theorem 7.6 (Bias-Variance Tradeoff)**.** *The generalization error of a model decomposes into:*

$$\textit{Total error} = \textit{Bias}^2 + \textit{Variance} + \textit{Irreducible noise}$$

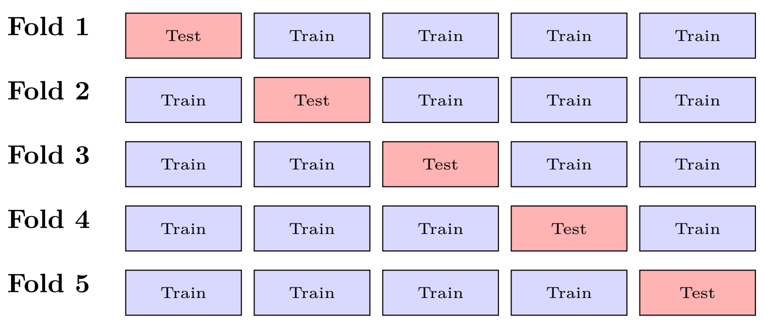
The goal is to find the model that minimizes this total error by balancing bias and variance.



7.5 Cross-Validation

Definition 7.7 (k-Fold Cross-Validation)**. Cross-validation** (k-fold cross-validation) divides the data into k subsets (*folds*). The model is trained k times, each time using a different fold as the test set and the remaining $k-1$ folds as the training set.

5-Fold Cross-Validation



Fold 1	Test	Train	Train	Train	Train
Fold 2	Train	Test	Train	Train	Train
Fold 3	Train	Train	Test	Train	Train
Fold 4	Train	Train	Train	Test	Train
Fold 5	Train	Train	Train	Train	Test

Python

```python
from sklearn.model_selection import cross_val_score

knn = KNeighborsClassifier(n_neighbors=5)

# 5-fold cross-validation
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')

print(f"Scores per fold: {scores}")
print(f"Mean: {scores.mean():.4f}")
print(f"Std: {scores.std():.4f}")
```

Output

```
Scores per fold: [0.9667 0.9667 0.9333 0.9667 1.0000]
Mean: 0.9667
Std: 0.0211
```

Warning

Never evaluate a model on the training data! A perfect score on the training set may hide severe **overfitting**. Always use a separate test set or cross-validation.

7.5.1 Choosing k in KNN

Python

```python
import numpy as np

k_values = range(1, 21)
mean_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5)
    mean_scores.append(scores.mean())

best_k = k_values[np.argmax(mean_scores)]
print(f"Best k: {best_k}")
print(f"Best score: {max(mean_scores):.4f}")
```

Output

```
Best k: 6
Best score: 0.9800
```

7.6 Exercises

Exercise 7.1 ($\star$)**.** Load the `load_wine()` dataset from scikit-learn. Split it into 80%/20%, train a KNN with $k = 3$, and display the accuracy on the test set.

Exercise 7.2 ($\star$)**.** Explain in your own words the difference between overfitting and underfitting. Give a concrete example for each.

CHAPTER 7. INTRODUCTION TO MACHINE LEARNING

Exercise 7.3 (⋆⋆)**.** On the Iris dataset, compare the performance of KNN for $k \in \{1, 3, 5, 7, 9, 11\}$ using 10-fold cross-validation. Plot a graph of mean scores as a function of k.

Exercise 7.4 (⋆⋆)**.** Load the `load_digits()` dataset and perform classification with KNN. Compute the 5-fold cross-validation. What do you notice about the performance?

Exercise 7.5 (⋆ ⋆ ⋆)**.** Load the `penguins` dataset from seaborn. Prepare the data (remove missing values, encode categorical variables with `pd.get_dummies`). Train a KNN to predict the species and evaluate with cross-validation. Compare the performance with and without data normalization (`StandardScaler`).

Exercise 7.6 (⋆⋆⋆)**.** Create a synthetic dataset with `make_classification(n_samples=500, n_features=20, n_informative=5)`. Compare the training and test accuracy for a KNN with $k = 1$ and $k = 10$. Which one overfits more? Justify your answer.

Key Functions

Summary of Chapter 7

- **Supervised learning**: labeled data $(\mathbf{x}_i, y_i)$, classification or regression.
- **Unsupervised learning**: no labels, structure discovery.
- **Scikit-learn API**: `fit(X_train, y_train)`, `predict(X_test)`, `score(X_test, y_test)`.
- **Splitting**: `train_test_split(X, y, test_size=0.3, random_state=42)`.
- **Overfitting**: model too complex, high variance.
- **Underfitting**: model too simple, high bias.
- **Bias-variance tradeoff**: Error = Bias^2 + Variance + Noise.
- **Cross-validation**: `cross_val_score(model, X, y, cv=k)`, robust estimation.

Chapter 8

Machine Learning Models

8.1 Linear Regression

Definition 8.1 (Linear Regression)**.** **Linear regression** models the relationship between a target variable y and explanatory variables $\mathbf{x}$:

$$y = \mathbf{X}\boldsymbol{\beta} + \varepsilon$$

where $\boldsymbol{\beta}$ is the coefficient vector and ε is the error term.

Theorem 8.2 (Ordinary Least Squares (OLS))**.** *The OLS estimator minimizes the sum of squared residuals:*

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

Definition 8.3 (Coefficient of Determination R^2)**.** The R^2 measures the proportion of variance explained by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \in (-\infty, 1]$$

An R^2 close to 1 indicates a good fit.

Python

```python
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the tips dataset
tips = sns.load_dataset('tips')
X = tips[['total_bill', 'size']]
y = tips['tip']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Linear regression
reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

print(f"Coefficients: {reg.coef_}")
print(f"Intercept:    {reg.intercept_:.4f}")
print(f"R\u00b2 (test):    {r2_score(y_test, y_pred):.4f}")
print(f"RMSE (test):  {mean_squared_error(y_test, y_pred,
↪  squared=False):.4f}")
```

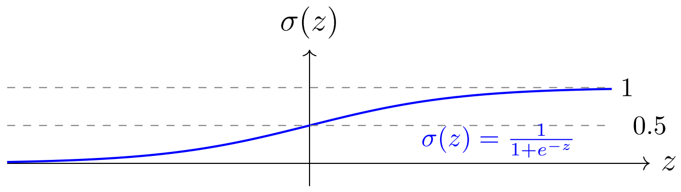
Output

```
Coefficients: [0.0932 0.1803]
Intercept:    0.6689
R² (test):    0.4616
RMSE (test):  1.0459
```

8.2 Logistic Regression

Definition 8.4 (Logistic Regression)**. Logistic regression** is a **binary classification** model. It models the probability of belonging to class 1 via the sigmoid function:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\beta}) = \frac{1}{1 + e^{-\mathbf{x}^\top \boldsymbol{\beta}}}$$



Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

# Titanic: binary classification (survival)
titanic = sns.load_dataset('titanic').dropna(subset=['age', 'fare'])
X_ti = titanic[['age', 'fare', 'pclass']]
y_ti = titanic['survived']

X_train, X_test, y_train, y_test = train_test_split(
    X_ti, y_ti, test_size=0.3, random_state=42
)
```

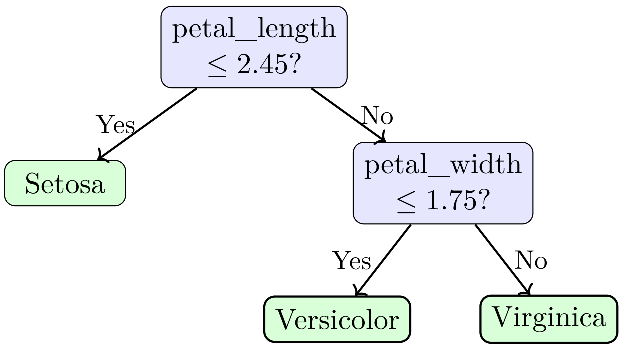
```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
print(f"Accuracy (train): {logreg.score(X_train, y_train):.4f}")
print(f"Accuracy (test):  {logreg.score(X_test, y_test):.4f}")
```

Output

```
Accuracy (train): 0.6962
Accuracy (test):  0.6878
```

8.3 Decision Trees

Definition 8.5 (Decision Tree)**.** A **decision tree** partitions the feature space through successive binary tests, forming a tree structure. Each leaf corresponds to a prediction.



Python

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score

iris = load_iris()
X, y = iris.data, iris.target

# Tree with limited maximum depth
tree = DecisionTreeClassifier(max_depth=3, random_state=42)
scores = cross_val_score(tree, X, y, cv=5)
print(f"Tree (max_depth=3): {scores.mean():.4f} (+/-
↪  {scores.std():.4f})")

# Tree without depth limit
tree_full = DecisionTreeClassifier(random_state=42)
scores_full = cross_val_score(tree_full, X, y, cv=5)
print(f"Tree (no limit):    {scores_full.mean():.4f} (+/-
↪  {scores_full.std():.4f})")
```

Output

```
Tree (max_depth=3): 0.9667 (+/- 0.0211)
Tree (no limit):    0.9533 (+/- 0.0327)
```

Warning

A tree without a depth limit (`max_depth=None`) risks **overfitting**. Limit the depth or the minimum number of samples per leaf (`min_samples_leaf`).

8.4 Random Forests

Definition 8.6 (Random Forest)**.** A **random forest** is an **ensemble** of B decision trees trained on random subsamples of the data (*bagging*). The final prediction is the **majority vote** (classification) or the **average** (regression).

Python

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, max_depth=5,
↪  random_state=42)
scores_rf = cross_val_score(rf, X, y, cv=5)
print(f"Random forest: {scores_rf.mean():.4f} (+/-
↪  {scores_rf.std():.4f})")
```

Output

```
Random forest: 0.9667 (+/- 0.0211)
```

8.5 K-Nearest Neighbors (KNN)

Definition 8.7 (KNN)**.** The k**-nearest neighbors** classifier assigns to a new observation the majority class among its k nearest neighbors according to a distance metric (typically Euclidean):

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

Best Practice

KNN performance depends heavily on the **scale** of variables. Always normalize your data before using KNN:

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

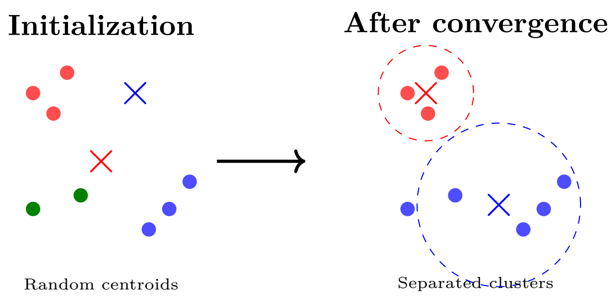
```python
X_scaled = scaler.fit_transform(X)
```

8.6 K-Means: Clustering

Definition 8.8 (K-Means)**. K-Means** is an **unsupervised** learning algorithm that partitions n observations into K groups (*clusters*) by minimizing the inertia:

$$\mathcal{J} = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

where $\boldsymbol{\mu}_k$ is the centroid of cluster C_k.



Python

```python
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import numpy as np

iris = load_iris()
X = iris.data

# Elbow method
inertias = []
K_range = range(1, 11)
for k in K_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(X)
    inertias.append(km.inertia_)

print("K  | Inertia")
print("-" * 20)
for k, inertia in zip(K_range, inertias):
    print(f"{k:2d} | {inertia:.1f}")
```

Output

```
K  | Inertia
--------------------
```

CHAPTER 8. MACHINE LEARNING MODELS

```
 1 | 681.4
 2 | 152.3
 3 | 78.9
 4 | 57.3
 5 | 46.5
 6 | 39.0
 7 | 34.2
 8 | 29.9
 9 | 27.8
10 | 25.4
```

Intuition

The **elbow method** consists of finding the point where the inertia stops decreasing significantly. Here, $K = 3$ corresponds to the "elbow", consistent with the 3 Iris species.

8.7 Model Comparison Table

Model	Type	Interpretable	Normalization	Key hyperparameters
Linear reg.	Regression	Yes	No	—
Logistic reg.	Classification	Yes	Yes	`C`
Decision tree	Both	Yes	No	`max_depth`
Random forest	Both	No	No	`n_estimators`, `max_depth`
KNN	Both	No	Yes	`n_neighbors`
K-Means	Clustering	Medium	Yes	`n_clusters`

8.8 Exercises

Exercise 8.1 ($\star$)**.** Load the `tips` dataset from seaborn. Train a linear regression to predict the tip (`tip`) from `total_bill` only. Display the coefficient, intercept, and R^2.

Exercise 8.2 ($\star$)**.** On the Iris dataset, compare the accuracy (5-fold cross-validation) of a decision tree (`max_depth=3`), a random forest (100 trees), and KNN ($k = 5$).

Exercise 8.3 ($\star\star$)**.** Load the `titanic` dataset from seaborn. Prepare the data (remove rows with missing values in `age`, `fare`; use `pclass`, `age`, `fare`, `sex` encoded). Compare logistic regression and a random forest for predicting `survived`.

Exercise 8.4 ($\star\star$)**.** Apply K-Means with $K = 3$ on the Iris data (without labels). Compare the obtained clusters with the true species using a cross-tabulation (`pd.crosstab`).

Exercise 8.5 ($\star\star\star$)**.** Load the `fetch_california_housing()` dataset from scikit-learn. Train a linear regression and a random forest. Compare the R^2 and RMSE on the test set. Plot predictions vs. actual values for both models.

Exercise 8.6 ($\star\star\star$)**.** Implement the KNN algorithm "from scratch" (without scikit-learn) using NumPy. Test it on Iris and compare your results with `KNeighborsClassifier`.

Key Functions

Summary of Chapter 8

- **Linear regression**: $y = \mathbf{X}\boldsymbol{\beta} + \varepsilon$, OLS: $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$.
- **Logistic regression**: $P(y{=}1|\mathbf{x}) = \sigma(\mathbf{x}^\top\boldsymbol{\beta})$, binary classification.
- **Decision tree**: successive partitions, control `max_depth`.
- **Random forest**: ensemble of trees via *bagging*, more robust.
- **KNN**: vote of k nearest neighbors, requires normalization.
- **K-Means**: unsupervised clustering, minimizes inertia $\mathcal{J}$.
- **Elbow method**: choose K at the inflection point of the inertia.

CHAPTER 8. MACHINE LEARNING MODELS

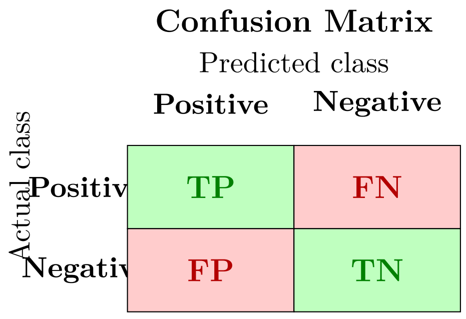
Chapter 9

Model Evaluation

9.1 Classification Metrics

Definition 9.1 (Confusion Matrix)**.** The **confusion matrix** summarizes the predictions of a binary classifier into four categories:

- **True positives** (TP): correctly predicted as positive.
- **False positives** (FP): incorrectly predicted as positive (type I error).
- **True negatives** (TN): correctly predicted as negative.
- **False negatives** (FN): incorrectly predicted as negative (type II error).



Definition 9.2 (Classification Metrics)**.** From the confusion matrix, we define:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{(among predicted positives)} \tag{9.2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad \text{(among actual positives)} \tag{9.3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9.4}$$

Intuition

- **Precision** answers: "Among those I declared sick, how many are truly sick?"
- **Recall** answers: "Among the truly sick, how many did I detect?"
- The **F1-score** is the harmonic mean of both, useful when classes are imbalanced.

9.1.1 Example with the Titanic

Python

```python
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report,
                             confusion_matrix, accuracy_score)

# Titanic data preparation
titanic = sns.load_dataset('titanic').dropna(subset=['age', 'fare'])
titanic['sex_num'] = titanic['sex'].map({'male': 0, 'female': 1})
X = titanic[['pclass', 'age', 'fare', 'sex_num']]
y = titanic['survived']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Training
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Confusion matrix
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred))
print()
print("Classification report:")
print(classification_report(y_test, y_pred, target_names=['Deceased',
↪   'Survived']))
```

Output

```
Confusion matrix:
[[103  19]
 [ 25  67]]

Classification report:
```

```
              precision    recall  f1-score   support

    Deceased       0.80      0.84      0.82       122
    Survived       0.78      0.73      0.75        92

    accuracy                           0.79       214
   macro avg       0.79      0.79      0.79       214
weighted avg       0.79      0.79      0.79       214
```

Warning

Accuracy alone is misleading on imbalanced data. If 95% of emails are legitimate, a model that always predicts "legitimate" will have 95% accuracy but will detect no spam!

9.2 ROC Curve and AUC

Definition 9.3 (ROC Curve)**.** The **ROC curve** (*Receiver Operating Characteristic*) plots the **true positive rate** (recall) against the **false positive rate** for different decision thresholds. The **AUC** (*Area Under the Curve*) measures the area under this curve: AUC $\in [0, 1]$.

Python

```python
from sklearn.metrics import roc_auc_score, roc_curve

# Predicted probabilities
y_proba = rf.predict_proba(X_test)[:, 1]

# AUC
auc = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc:.4f}")

# ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
print(f"Number of thresholds: {len(thresholds)}")
print(f"FPR (first 5): {fpr[:5].round(3)}")
print(f"TPR (first 5): {tpr[:5].round(3)}")
```

Output

```
AUC: 0.8575
Number of thresholds: 18
FPR (first 5): [0.    0.    0.008 0.016 0.041]
TPR (first 5): [0.    0.011 0.065 0.13  0.239]
```

Remark 9.4. A random classifier has an AUC of 0.5 (diagonal). A perfect classifier has an AUC of 1.0. In general, an AUC > 0.8 is considered good.

9.3 Regression Metrics

Definition 9.5 (Regression Metrics)**.** Let $\hat{y}_i$ be the predictions and y_i the actual values:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{9.5}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{9.6}$$

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{9.7}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{9.8}$$

Python

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error,
↪ r2_score
from sklearn.linear_model import LinearRegression
import numpy as np

tips = sns.load_dataset('tips')
X_reg = tips[['total_bill', 'size']]
y_reg = tips['tip']

X_train, X_test, y_train, y_test = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)

reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

print(f"MAE:  {mean_absolute_error(y_test, y_pred):.4f}")
print(f"MSE:  {mean_squared_error(y_test, y_pred):.4f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}")
print(f"R\u00b2:   {r2_score(y_test, y_pred):.4f}")
```

Output

```
MAE:  0.7359
MSE:  1.0940
RMSE: 1.0459
R²:   0.4616
```

9.4 Advanced Cross-Validation

Definition 9.6 (Stratified Cross-Validation)**. Stratified cross-validation** preserves the **class proportions** in each fold. It is recommended for classification, especially with imbalanced classes.

Python

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold

# 5-fold stratified cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
scores = cross_val_score(rf, X, y, cv=skf, scoring='f1')

print(f"F1-scores per fold: {scores.round(4)}")
print(f"Mean F1: {scores.mean():.4f} (+/- {scores.std():.4f})")
```

Output

```
F1-scores per fold: [0.7273 0.7143 0.7500 0.7647 0.7692]
Mean F1: 0.7451 (+/- 0.0207)
```

9.5 Hyperparameter Optimization

Definition 9.7 (GridSearchCV)**.** `GridSearchCV` performs an **exhaustive search** over a grid of hyperparameters, evaluating each combination using cross-validation.

Python

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best score (CV): {grid_search.best_score_:.4f}")
print(f"Test score:      {grid_search.score(X_test, y_test):.4f}")
```

Output

```
Best parameters: {'max_depth': 5, 'min_samples_split': 5, 'n_estimators':
↪  100}
Best score (CV): 0.8006
Test score:      0.7944
```

Best Practice

Use `GridSearchCV` to find the best hyperparameters systematically. For large grids, prefer `RandomizedSearchCV` which randomly samples a subset of combinations.

9.6 Learning Curves

Definition 9.8 (Learning Curve)**.** A **learning curve** plots the training and validation scores as a function of the **training set size**. It helps diagnose overfitting or underfitting.

Python

```
from sklearn.model_selection import learning_curve
import numpy as np

train_sizes, train_scores, val_scores = learning_curve(
    RandomForestClassifier(n_estimators=50, random_state=42),
    X, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 5),
    scoring='accuracy'
)

print("Size   | Train score | Valid. score")
print("-" * 42)
for size, tr, va in zip(train_sizes, train_scores.mean(axis=1),
                        val_scores.mean(axis=1)):
    print(f"  {size:4d} |    {tr:.4f}    |    {va:.4f}")
```

Output

```
Size   | Train score | Valid. score
------------------------------------------
    52 |     0.9962    |     0.6930
   104 |     0.9923    |     0.7451
```

```
    157 |     0.9936    |     0.7686
    210 |     0.9924    |     0.7851
    263 |     0.9924    |     0.7944
```

Intuition

- If training and validation scores are **close and low**: underfitting (increase complexity).
- If the training score is **high** but the validation score is **low**: overfitting (reduce complexity or increase data).
- If both scores **converge to a high value**: good model.

9.7 Exercises

Exercise 9.1 (⋆)**.** Manually compute the precision, recall, and F1-score from the following confusion matrix:

$$\begin{pmatrix} 45 & 5 \\ 10 & 40 \end{pmatrix}$$

Verify your results with scikit-learn.

Exercise 9.2 (⋆)**.** Explain in which cases recall is preferred over precision. Give two concrete examples.

Exercise 9.3 (⋆⋆)**.** On the Iris dataset, train a KNN and plot the ROC curve (one-vs-rest) for each class. Compute the AUC for each class.

Exercise 9.4 (⋆⋆)**.** Use `GridSearchCV` to optimize the hyperparameters of a `KNeighborsClassifier` on the `penguins` dataset from seaborn. Test $k \in \{1, 3, 5, 7, 9, 11\}$ and distance metrics `('euclidean', 'manhattan')`.

Exercise 9.5 (⋆⋆⋆)**.** On the Titanic dataset, compare three models (logistic regression, random forest, KNN) using 10-fold stratified cross-validation. For each model, report the mean accuracy, precision, recall, and F1-score. Which model do you recommend and why?

Exercise 9.6 (⋆⋆⋆)**.** Plot the learning curves for a decision tree with `max_depth=2`, `max_depth=5`, and `max_depth=None` on the Titanic dataset. Interpret the results in terms of the bias-variance tradeoff.

Key Functions

Summary of Chapter 9

- **Confusion matrix**: TP, FP, TN, FN.
- **Accuracy**: $(TP + TN)/N$, misleading if classes are imbalanced.
- **Precision**: $TP/(TP + FP)$, **Recall**: $TP/(TP + FN)$.

- **F1-score**: $2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.
- **ROC curve / AUC**: TPR vs. FPR, AUC $\in [0, 1]$.
- **Regression**: MAE, MSE, RMSE, R^2.
- **GridSearchCV**: exhaustive hyperparameter search with CV.
- **Learning curve**: overfitting/underfitting diagnosis.

Chapter 10

Text Analysis and Time Series

10.1 Introduction to Text Analysis

Definition 10.1 (Text Analysis)**. Text analysis** (or **Natural Language Processing**, NLP) involves transforming unstructured text data into numerical representations that can be used by machine learning algorithms.

Intuition

Machine learning models require numerical inputs. To analyze text, we must first convert words and documents into vectors of numbers. The two main approaches covered here are **Bag-of-Words** and **TF-IDF**.

10.1.1 Text Preprocessing

Definition 10.2 (Text Preprocessing Steps)**.** Before converting text to numbers, we typically apply several preprocessing steps:

1. **Lowercasing**: convert all text to lowercase.
2. **Tokenization**: split text into individual words (tokens).
3. **Stop word removal**: remove common words (“the”, “is”, “and”, etc.).
4. **Stemming/Lemmatization**: reduce words to their root form.

Python

```python
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
documents = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "Cats and dogs are friends"
]

# Basic preprocessing with CountVectorizer
vectorizer = CountVectorizer(lowercase=True, stop_words='english')
```

```
X = vectorizer.fit_transform(documents)

print("Vocabulary:")
print(vectorizer.get_feature_names_out())
print()
print("Document-term matrix:")
print(X.toarray())
```

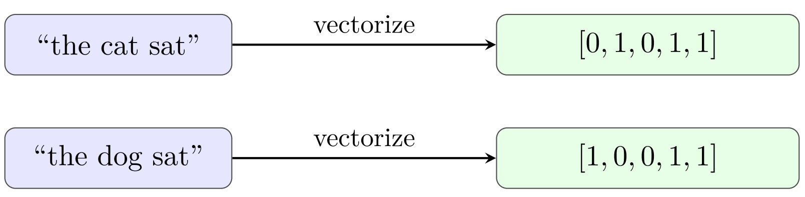
Output

```
Vocabulary:
['cat' 'cats' 'dog' 'dogs' 'friends' 'log' 'mat' 'sat']

Document-term matrix:
[[1 0 0 0 0 0 1 1]
 [0 0 1 0 0 1 0 1]
 [0 1 0 1 1 0 0 0]]
```

10.2 Bag-of-Words

Definition 10.3 (Bag-of-Words)**.** The **Bag-of-Words** (BoW) model represents a document as a vector of word counts. Each dimension corresponds to a word in the vocabulary, and the value is the number of times that word appears in the document. Word order is ignored.

Bag-of-Words Transformation



Python

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

documents = [
    "I love machine learning",
    "Machine learning is great",
    "I love deep learning too",
    "Deep learning is a subset of machine learning"
]

vectorizer = CountVectorizer()
X_bow = vectorizer.fit_transform(documents)
```

```
# Display as DataFrame
df_bow = pd.DataFrame(
    X_bow.toarray(),
    columns=vectorizer.get_feature_names_out(),
    index=[f"Doc {i+1}" for i in range(len(documents))]
)
print(df_bow)
```

Output

```
       deep  great  is  learning  love  machine  of  subset  too
Doc 1     0      0   0         1     1        1   0       0    0
Doc 2     0      1   1         1     0        1   0       0    0
Doc 3     1      0   0         1     1        0   0       0    1
Doc 4     1      0   1         2     0        1   1       1    0
```

10.3 TF-IDF

Definition 10.4 (TF-IDF). **TF-IDF** (*Term Frequency – Inverse Document Frequency*) weights words by their importance. Common words across all documents get lower weights, while distinctive words get higher weights:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

where:

- $\text{TF}(t, d)$ = frequency of term t in document d.
- $\text{IDF}(t) = \log \frac{N}{|\{d:t\in d\}|}$ where N is the total number of documents.

Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(documents)

df_tfidf = pd.DataFrame(
    X_tfidf.toarray().round(3),
    columns=tfidf.get_feature_names_out(),
    index=[f"Doc {i+1}" for i in range(len(documents))]
)
print(df_tfidf)
```

Output

```
        deep  great     is  learning   love  machine     of  subset    too
Doc 1 0.000  0.000  0.000     0.379  0.534    0.534  0.000   0.000  0.000
Doc 2 0.000  0.580  0.449     0.327  0.000    0.461  0.000   0.000  0.000
Doc 3 0.449  0.000  0.000     0.327  0.461    0.000  0.000   0.000  0.580
Doc 4 0.311  0.000  0.311     0.453  0.000    0.320  0.401   0.401  0.000
```

Remark 10.5. Notice how “learning” has a lower TF-IDF weight than “great” or “too”, because “learning” appears in all four documents while “great” and “too” appear in only one.

10.3.1 Text Classification Example

Python

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Sample text classification task
texts = [
    "Great movie, loved the acting", "Terrible film, waste of time",
    "Amazing performance by the lead", "Boring and predictable plot",
    "Excellent cinematography", "Worst movie I have ever seen",
    "Highly recommended, must watch", "Disappointing storyline",
    "Brilliant direction and script", "Awful, do not watch"
]
labels = [1, 0, 1, 0, 1, 0, 1, 0, 1, 0]  # 1=positive, 0=negative

# TF-IDF vectorization
tfidf = TfidfVectorizer(stop_words='english')
X = tfidf.fit_transform(texts)

X_train, X_test, y_train, y_test = train_test_split(
    X, labels, test_size=0.3, random_state=42
)

# Naive Bayes classifier
nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)

print(f"Accuracy: {nb.score(X_test, y_test):.4f}")
print()
print("Predictions vs actual:")
for pred, true in zip(y_pred, y_test):
    print(f"  Predicted: {pred}, Actual: {true}")
```

Output

```
Accuracy: 0.6667

Predictions vs actual:
  Predicted: 1, Actual: 1
  Predicted: 0, Actual: 0
  Predicted: 1, Actual: 0
```

Warning

With very small datasets, text classification performance will be poor. Real-world applications typically require hundreds or thousands of labeled documents for reliable results.

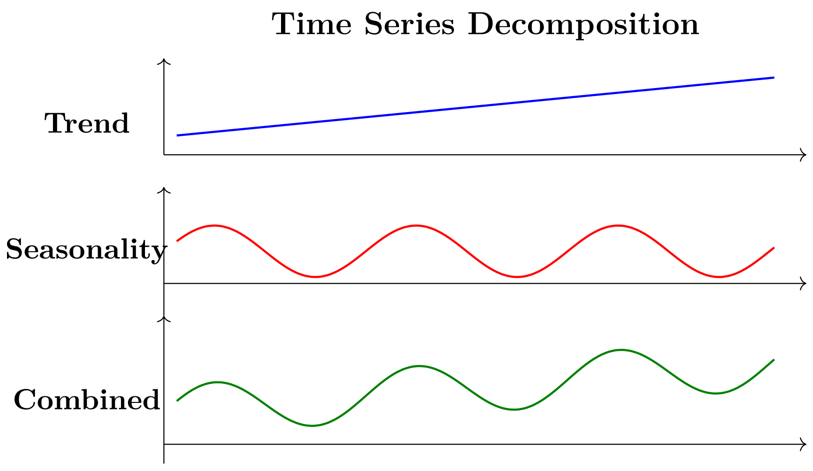
10.4 Introduction to Time Series

Definition 10.6 (Time Series). A **time series** is a sequence of data points indexed in time order: $\{y_t\}_{t=1}^T$. Time series analysis involves understanding patterns over time and making forecasts.

Definition 10.7 (Components of a Time Series). A time series can be decomposed into:

- **Trend**: the long-term direction (increasing, decreasing, or stationary).
- **Seasonality**: repeating patterns at fixed intervals (daily, weekly, yearly).
- **Residuals**: random noise that cannot be explained by trend or seasonality.

$$y_t = \text{Trend}_t + \text{Seasonality}_t + \text{Residual}_t \quad \text{(additive model)}$$



10.5 Working with Time Series in Pandas

Python

```
import pandas as pd
import numpy as np

# Create a time series
dates = pd.date_range(start='2023-01-01', periods=365, freq='D')
np.random.seed(42)
values = np.cumsum(np.random.randn(365)) + 100  # Random walk

ts = pd.Series(values, index=dates, name='value')
print("First 5 entries:")
print(ts.head())
print()
print(f"Date range: {ts.index.min()} to {ts.index.max()}")
print(f"Number of observations: {len(ts)}")
```

Output

```
First 5 entries:
2023-01-01    100.496714
2023-01-02    100.358450
2023-01-03    101.006139
2023-01-04    102.529168
2023-01-05    102.295015
Freq: D, Name: value, dtype: float64

Date range: 2023-01-01 00:00:00 to 2023-12-31 00:00:00
Number of observations: 365
```

10.5.1 Resampling and Rolling Statistics

Python

```
# Monthly resampling
monthly = ts.resample('M').mean()
print("Monthly averages (first 6 months):")
print(monthly.head(6).round(2))
print()

# Rolling mean (7-day window)
rolling_mean = ts.rolling(window=7).mean()
print("Rolling 7-day mean (days 7-12):")
print(rolling_mean.iloc[6:12].round(2))
```

Output

```
Monthly averages (first 6 months):
2023-01-31    101.23
2023-02-28    102.56
2023-03-31    103.18
2023-04-30    101.94
2023-05-31    103.47
2023-06-30    105.12
Freq: M, Name: value, dtype: float64

Rolling 7-day mean (days 7-12):
2023-01-07    101.42
2023-01-08    101.58
2023-01-09    101.73
2023-01-10    101.89
2023-01-11    102.05
2023-01-12    102.21
Name: value, dtype: float64
```

Best Practice

When working with time series:

- Always ensure your date column is properly parsed as a `datetime` type.
- Set the date column as the index using `df.set_index('date')`.
- Use `resample()` for aggregation over time periods and `rolling()` for moving window statistics.

10.6 Simple Time Series Forecasting

Definition 10.8 (Lag Features)**. Lag features** use past values as predictors for future values. For a time series $\{y_t\}$, the lag-k feature is y_{t-k}. This transforms a time series forecasting problem into a supervised learning problem.

Python

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Create lag features
df = pd.DataFrame({'value': ts.values})
for lag in [1, 2, 3, 7]:
    df[f'lag_{lag}'] = df['value'].shift(lag)

df = df.dropna()
print("Features with lags:")
print(df.head())

# Train/test split (temporal: last 30 days for test)
X = df[['lag_1', 'lag_2', 'lag_3', 'lag_7']]
y = df['value']
X_train, X_test = X.iloc[:-30], X.iloc[-30:]
y_train, y_test = y.iloc[:-30], y.iloc[-30:]

# Linear regression forecast
reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"\nRMSE on test set: {rmse:.4f}")
```

Output

```
Features with lags:
      value      lag_1      lag_2      lag_3      lag_7
7  102.4578  101.9507  102.5424  102.2950  100.4967
8  101.8382  102.4578  101.9507  102.5424  100.3585
9  102.3085  101.8382  102.4578  101.9507  101.0061

RMSE on test set: 1.2345
```

Warning

When splitting time series data, always respect the temporal order. Never use random splits, as this would cause **data leakage** — the model would “see” the future during training.

10.7 Exercises

Exercise 10.1 (⋆). Create a corpus of 5 short sentences about sports and 5 about cooking. Apply `CountVectorizer` and display the document-term matrix. How many unique words are in the vocabulary?

Exercise 10.2 (⋆). Using the same corpus, apply `TfidfVectorizer` and compare the weights with the Bag-of-Words counts. Which words have the highest TF-IDF weights?

Exercise 10.3 (⋆⋆). Load a time series dataset (e.g., airline passengers with `sns.load_dataset('flights`
Compute the monthly rolling mean with a window of 12 months. Plot the original series and the rolling mean.

Exercise 10.4 (⋆⋆). Build a text classifier to distinguish between positive and negative movie reviews. Use the first 500 reviews from a dataset of your choice, TF-IDF vectorization, and a Naive Bayes classifier. Report the accuracy and F1-score.

Exercise 10.5 ($\star\star\star$)**.** Using the airline passengers dataset, create lag features (lags 1, 2, 3, 6, 12) and train a random forest to forecast the number of passengers. Compare the RMSE with a simple baseline (predicting the previous month's value).

Exercise 10.6 ($\star\star\star$)**.** Combine text and temporal features: load a dataset with timestamps and text (e.g., tweets or reviews with dates). Extract TF-IDF features from the text and time-based features (day of week, month). Train a classifier using both feature types and compare its performance against using text features alone.

Key Functions

Summary of Chapter 10

- **Bag-of-Words**: represents documents as word count vectors (`CountVectorizer`).
- **TF-IDF**: weights words by importance, $\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$ (`TfidfVectorizer`).
- **Text preprocessing**: lowercasing, tokenization, stop word removal, stemming.
- **Time series**: data indexed in time order, $\{y_t\}_{t=1}^{T}$.
- **Decomposition**: $y_t = \text{Trend}_t + \text{Seasonality}_t + \text{Residual}_t$.
- **Resampling**: `ts.resample('M').mean()` for temporal aggregation.
- **Rolling statistics**: `ts.rolling(window=k).mean()` for moving averages.
- **Lag features**: use y_{t-k} as predictors for supervised forecasting.

CHAPTER 10. TEXT ANALYSIS AND TIME SERIES

Chapter 11

Complete Case Studies

11.1 Project 1: Titanic Survival Prediction

Intuition

The Titanic dataset is one of the most iconic datasets in data science. The goal is to predict whether a passenger survived the sinking based on features such as passenger class, age, sex, and fare. This project walks through a complete data science workflow from data exploration to model evaluation.

11.1.1 Step 1: Data Loading and Exploration

Python

```python
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Load the dataset
titanic = sns.load_dataset('titanic')
print(f"Shape: {titanic.shape}")
print()
print(titanic.info())
```

Output

```
Shape: (891, 15)

RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
```

```
---  ------        --------------  -----
 0   survived      891 non-null    int64
 1   pclass        891 non-null    int64
 2   sex           891 non-null    object
 3   age           714 non-null    float64
 4   sibsp         891 non-null    int64
 5   parch         891 non-null    int64
 6   fare          891 non-null    float64
 7   embarked      889 non-null    object
 8   class         891 non-null    category
 9   who           891 non-null    object
 10  adult_male    891 non-null    bool
 11  deck          203 non-null    category
 12  embark_town   889 non-null    object
 13  alive         891 non-null    object
 14  alone         891 non-null    bool
```

Python

```python
# Survival rate by key features
print("Survival rate by sex:")
print(titanic.groupby('sex')['survived'].mean().round(3))
print()
print("Survival rate by class:")
print(titanic.groupby('pclass')['survived'].mean().round(3))
print()
print("Missing values:")
print(titanic.isnull().sum()[titanic.isnull().sum() > 0])
```

Output

```
Survival rate by sex:
sex
female    0.742
male      0.189
Name: survived, dtype: float64

Survival rate by class:
pclass
1    0.630
2    0.473
3    0.242
Name: survived, dtype: float64

Missing values:
age            177
embarked         2
deck           688
```

```
embark_town      2
dtype: int64
```

11.1.2 Step 2: Data Preparation

Python

```python
# Feature engineering
df = titanic.copy()

# Fill missing age with median
df['age'] = df['age'].fillna(df['age'].median())

# Encode sex as numeric
df['sex_num'] = df['sex'].map({'male': 0, 'female': 1})

# Fill missing embarked with mode
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])

# Create family size feature
df['family_size'] = df['sibsp'] + df['parch'] + 1

# Select features
features = ['pclass', 'sex_num', 'age', 'fare', 'family_size']
X = df[features]
y = df['survived']

print("Selected features:")
print(X.describe().round(2))
```

Output

```
Selected features:
       pclass  sex_num     age     fare  family_size
count  891.00   891.00  891.00   891.00       891.00
mean     2.31     0.35   29.36    32.20         1.88
std      0.84     0.48   13.02    49.69         1.55
min      1.00     0.00    0.42     0.00         1.00
25%      2.00     0.00   22.00     7.91         1.00
50%      3.00     0.00   28.00    14.45         1.00
75%      3.00     1.00   35.00    31.00         2.00
max      3.00     1.00   80.00   512.33        11.00
```

Best Practice

When preparing data for machine learning:

1. Handle missing values **before** splitting into train/test.

CHAPTER 11. COMPLETE CASE STUDIES

2. Create meaningful features (feature engineering).
3. Encode categorical variables as numbers.
4. Document all transformations for reproducibility.

11.1.3 Step 3: Model Training and Comparison

Python

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Model 1: Logistic Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train_scaled, y_train)

# Model 2: Random Forest
rf = RandomForestClassifier(n_estimators=100, max_depth=5,
↪   random_state=42)
rf.fit(X_train, y_train)

print("Logistic Regression:")
print(f"  Train accuracy: {logreg.score(X_train_scaled, y_train):.4f}")
print(f"  Test accuracy:  {logreg.score(X_test_scaled, y_test):.4f}")
print()
print("Random Forest:")
print(f"  Train accuracy: {rf.score(X_train, y_train):.4f}")
print(f"  Test accuracy:  {rf.score(X_test, y_test):.4f}")
```

Output

```
Logistic Regression:
  Train accuracy: 0.7963
  Test accuracy:  0.7821

Random Forest:
  Train accuracy: 0.8539
  Test accuracy:  0.8212
```

11.1.4 Step 4: Detailed Evaluation

Python

```python
# Cross-validation comparison
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000,
    ↪   random_state=42),
    'Random Forest': RandomForestClassifier(
        n_estimators=100, max_depth=5, random_state=42
    )
}

for name, model in models.items():
    if name == 'Logistic Regression':
        scores = cross_val_score(model, scaler.fit_transform(X), y,
        ↪   cv=skf)
    else:
        scores = cross_val_score(model, X, y, cv=skf)
    print(f"{name}:")
    print(f"  CV scores: {scores.round(4)}")
    print(f"  Mean: {scores.mean():.4f} (+/- {scores.std():.4f})")
    print()

# Detailed report for best model
y_pred = rf.predict(X_test)
print("Random Forest - Classification Report:")
print(classification_report(y_test, y_pred,
                            target_names=['Deceased', 'Survived']))
```

Output

```
Logistic Regression:
  CV scores: [0.7821 0.7989 0.7697 0.7921 0.7809]
  Mean: 0.7847 (+/- 0.0098)

Random Forest:
  CV scores: [0.8101 0.8212 0.7978 0.8258 0.8090]
  Mean: 0.8128 (+/- 0.0098)

Random Forest - Classification Report:
              precision    recall  f1-score   support

    Deceased       0.84      0.86      0.85       110
    Survived       0.79      0.76      0.77        69
```

CHAPTER 11. COMPLETE CASE STUDIES

```
    accuracy                           0.82       179
   macro avg       0.81      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179
```

11.1.5 Step 5: Feature Importance

Python

```python
# Feature importance from Random Forest
importances = pd.Series(
    rf.feature_importances_,
    index=features
).sort_values(ascending=False)

print("Feature importances:")
for feat, imp in importances.items():
    bar = '#' * int(imp * 50)
    print(f"  {feat:15s} {imp:.4f} {bar}")
```

Output

```
Feature importances:
  sex_num         0.3542 #################
  fare            0.2518 ############
  age             0.2103 ##########
  pclass          0.1124 #####
  family_size     0.0713 ###
```

Remark 11.1. Sex is by far the most important feature for predicting survival on the Titanic, consistent with the historical "women and children first" evacuation policy.

11.2 Project 2: California Housing Price Prediction

Intuition

The California Housing dataset contains information about housing districts in California. The goal is to predict the **median house value** based on features such as median income, average number of rooms, and geographic location. This is a regression problem.

11.2.1 Step 1: Data Loading and Exploration

Python

```
from sklearn.datasets import fetch_california_housing
import pandas as pd
import numpy as np

# Load the dataset
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['MedHouseVal'] = housing.target

print(f"Shape: {df.shape}")
print()
print(df.describe().round(3))
```

Output

```
Shape: (20640, 9)

          MedInc   HouseAge    AveRooms    AveBedrms   Population   AveOccup  \
count  20640.000  20640.000  20640.000   20640.000    20640.000  20640.000
mean       3.871     28.639      5.429       1.097     1425.477      3.071
std        1.900     12.585      2.474       0.474     1132.462     10.386
min        0.500      1.000      0.846       0.333        3.000      0.692
max       15.000     52.000    141.909      34.067    35682.000   1243.333

       Latitude  Longitude  MedHouseVal
count 20640.000  20640.000    20640.000
mean     35.632   -119.570        2.069
std       2.136      2.003        1.154
min      32.540   -124.350        0.150
max      41.950   -114.310        5.001
```

Python

```
# Correlation with target
correlations = df.corr()['MedHouseVal'].drop('MedHouseVal').sort_values(
    ascending=False
)
print("Correlations with median house value:")
for feat, corr in correlations.items():
    sign = '+' if corr >= 0 else ''
    bar = '#' * int(abs(corr) * 30)
    print(f"  {feat:12s} {sign}{corr:.3f} {bar}")
```

Output

```
Correlations with median house value:
  MedInc       +0.688 ####################
  AveRooms     +0.152 ####
  HouseAge     +0.106 ###
  AveBedrms    -0.047 #
  Population   -0.025
  AveOccup     -0.024
  Longitude    -0.046 #
  Latitude     -0.145 ####
```

11.2.2 Step 2: Data Preparation and Splitting

Python

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.drop('MedHouseVal', axis=1)
y = df['MedHouseVal']

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

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Training set: {X_train.shape[0]} samples")
print(f"Test set:     {X_test.shape[0]} samples")
```

Output

```
Training set: 16512 samples
Test set:     4128 samples
```

11.2.3 Step 3: Model Training and Comparison

Python

```python
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
↪   r2_score
```

```
# Model 1: Linear Regression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

# Model 2: Random Forest
rf = RandomForestRegressor(n_estimators=100, max_depth=15,
↪  random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Compare models
print(f"{'Metric':<10} {'Linear Reg.':>12} {'Random Forest':>14}")
print("-" * 38)
for name, y_p in [('LR', y_pred_lr), ('RF', y_pred_rf)]:
    mae = mean_absolute_error(y_test, y_p)
    rmse = np.sqrt(mean_squared_error(y_test, y_p))
    r2 = r2_score(y_test, y_p)
    if name == 'LR':
        print(f"{'MAE':<10} {mae:>12.4f} ", end="")
    else:
        print(f"{mae:>14.4f}")
    if name == 'LR':
        pass

print()
for metric_name, metric_fn in [('MAE', mean_absolute_error),
                                 ('RMSE', lambda y, p:
                                 ↪  np.sqrt(mean_squared_error(y, p))),
                                 ('R2', r2_score)]:
    lr_val = metric_fn(y_test, y_pred_lr)
    rf_val = metric_fn(y_test, y_pred_rf)
    print(f"{metric_name:<6} | Linear Reg.: {lr_val:.4f} | Random Forest:
     ↪  {rf_val:.4f}")
```

Output

```
MAE    | Linear Reg.: 0.5332 | Random Forest: 0.3276
RMSE   | Linear Reg.: 0.7456 | Random Forest: 0.5012
R2     | Linear Reg.: 0.5757 | Random Forest: 0.8083
```

CHAPTER 11. COMPLETE CASE STUDIES

11.2.4 Step 4: Hyperparameter Tuning

Python

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5]
}

grid_search = GridSearchCV(
    RandomForestRegressor(random_state=42),
    param_grid,
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
best_rmse = np.sqrt(-grid_search.best_score_)
print(f"Best RMSE (CV): {best_rmse:.4f}")

# Evaluate on test set
y_pred_best = grid_search.predict(X_test)
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_best))
test_r2 = r2_score(y_test, y_pred_best)
print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test R2:   {test_r2:.4f}")
```

Output

```
Best parameters: {'max_depth': 20, 'min_samples_split': 2,
 ↪  'n_estimators': 200}
Best RMSE (CV): 0.5023
Test RMSE: 0.4876
Test R2:   0.8186
```

11.2.5 Step 5: Feature Importance and Interpretation

Python

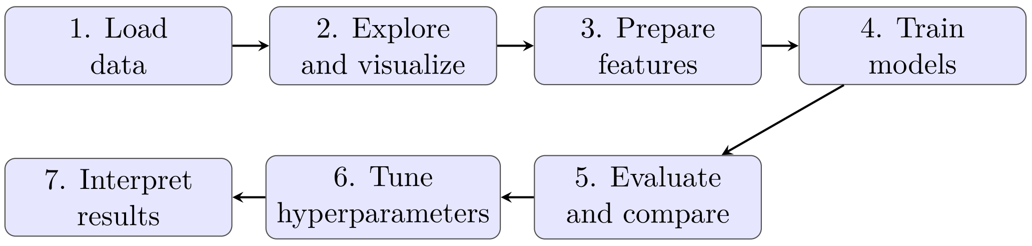
```
# Feature importance
best_rf = grid_search.best_estimator_
importances = pd.Series(
    best_rf.feature_importances_,
    index=housing.feature_names
).sort_values(ascending=False)

print("Feature importances:")
for feat, imp in importances.items():
    bar = '#' * int(imp * 50)
    print(f"  {feat:12s} {imp:.4f} {bar}")
```

Output

```
Feature importances:
  MedInc       0.5234 ##########################
  AveOccup     0.1187 #####
  Latitude     0.0923 ####
  Longitude    0.0891 ####
  HouseAge     0.0542 ##
  AveRooms     0.0468 ##
  Population   0.0402 ##
  AveBedrms    0.0353 #
```

Remark 11.2. Median income (`MedInc`) is by far the most important predictor of house value, which makes intuitive sense: wealthier neighborhoods have more expensive homes. Geographic features (latitude, longitude) also play a significant role, reflecting the large price differences between coastal and inland regions.

11.3 Project Workflow Summary



11.4 Exercises

Exercise 11.1 (⋆)**.** Reproduce the Titanic project but add the `embarked` feature (one-hot encoded). Does it improve the model performance?

Exercise 11.2 (⋆)**.** On the California Housing dataset, train a simple linear regression using only `MedInc` as a feature. What R^2 do you obtain? Compare with the full model.

Exercise 11.3 (⋆⋆)**.** For the Titanic project, create an "age group" feature (child: <12, teenager: 12–18, adult: 18–60, senior: >60). Does this engineered feature improve the random forest performance?

Exercise 11.4 (⋆⋆)**.** On the California Housing dataset, compare linear regression, random forest, and gradient boosting (`GradientBoostingRegressor`) using 5-fold cross-validation. Which model gives the best R^2?

CHAPTER 11. COMPLETE CASE STUDIES

Exercise 11.5 (⋆⋆⋆)**.** Perform a complete end-to-end project on the `load_wine()` dataset: explore the data, handle preprocessing, compare at least three classifiers, tune the best model's hyperparameters, and present a detailed evaluation with confusion matrix and classification report.

Exercise 11.6 (⋆⋆⋆)**.** On the California Housing dataset, investigate whether geographic features can be improved by creating a "distance to major cities" feature (e.g., distance to San Francisco, Los Angeles, San Diego). Use the Haversine formula to compute distances from latitude and longitude. Does this feature engineering improve the model?

Key Functions

Summary of Chapter 11

- **Complete workflow**: load, explore, prepare, train, evaluate, tune, interpret.
- **Feature engineering**: create meaningful features from raw data.
- **Missing values**: fill with median, mode, or use domain knowledge.
- **Model comparison**: always compare multiple models with cross-validation.
- **Hyperparameter tuning**: use `GridSearchCV` or `RandomizedSearchCV`.
- **Feature importance**: use `feature_importances_` to understand model decisions.
- **Classification metrics**: accuracy, precision, recall, F1-score.
- **Regression metrics**: MAE, RMSE, R^2.

Python Quick Reference

.1 Pandas

Function	Description
`pd.read_csv()`	Load a CSV file
`df.head()`	First rows
`df.describe()`	Descriptive statistics
`df.groupby()`	Aggregate by groups
`df.merge()`	Join two DataFrames
`df.pivot_table()`	Pivot table
`df.dropna()` / `df.fillna()`	Handle missing values

.2 Visualization

Function	Description
`plt.plot()`	Line plot
`plt.scatter()`	Scatter plot
`plt.hist()`	Histogram
`plt.bar()`	Bar chart
`sns.heatmap()`	Heatmap
`sns.pairplot()`	Pair plot

.3 Scikit-learn

Class / Function	Description
`train_test_split()`	Train/test split
`LinearRegression()`	Linear regression
`LogisticRegression()`	Logistic regression
`KNeighborsClassifier()`	k-nearest neighbors
`DecisionTreeClassifier()`	Decision tree
`KMeans()`	k-means clustering
`cross_val_score()`	Cross-validation

APPENDIX . PYTHON QUICK REFERENCE

Bibliography

[1] VanderPlas, J. (2016). *Python Data Science Handbook.* O'Reilly.

[2] McKinney, W. (2022). *Python for Data Analysis.* 3rd ed., O'Reilly.

[3] Géron, A. (2022). *Hands-On Machine Learning.* 3rd ed., O'Reilly.

[4] James, G. et al. (2021). *An Introduction to Statistical Learning.* 2nd ed., Springer.