Data Analysis with Python

for Health Specialists

A 30-hour practical course

Yaé Ulrich Gaba

AIRINA Labs

2026

Contents

Preface

This course is designed for health professionals — physicians, nurses, epidemiologists, public health researchers, and health policy analysts — who need to analyze data but have little or no programming experience. You do not need to become a software developer. You need to become someone who can load a dataset, clean it, explore it, test a hypothesis, build a simple predictive model, and communicate the results clearly.

We use Python because it is free, widely adopted in health research, and has an ecosystem of libraries specifically designed for the kinds of analyses health professionals need: survival analysis, epidemiological measures, geospatial mapping, and clinical prediction.

Every chapter follows the same rhythm: a brief explanation of the concept, a worked example on real health data, and exercises you complete in Jupyter notebooks. All datasets used in this course are freely available from the WHO, CDC, UCI Machine Learning Repository, and other open sources.

What this course is not. This is not a statistics course (though we use statistics). It is not a computer science course (though we write code). It is a *practical data analysis course* for people whose primary expertise is health, not programming.

Prerequisites. None beyond basic computer literacy. Familiarity with descriptive statistics (mean, median, standard deviation) is helpful but will be reviewed.

Software. All code runs in Jupyter notebooks, either locally (Anaconda) or in the cloud (Google Colab — free, no installation required).

Yaé Ulrich Gaba
Cotonou, 2026

Chapter 1

Python essentials for health professionals

> *"The goal is not to learn Python. The goal is to answer health questions with data. Python is just the tool."*

1.1 Why Python for health data?

Three reasons health professionals should learn Python rather than relying solely on Excel or SPSS:

1. **Reproducibility.** A Python script documents every step of your analysis. When a reviewer asks "how did you handle missing lab values?", you show the code — not a description of menu clicks.

2. **Scale.** Excel struggles with 100 000 patient records. Python handles millions without slowing down.

3. **Ecosystem.** Libraries like `lifelines` (survival analysis), `geopandas` (disease mapping), and `scikit-learn` (predictive models) give you tools that SPSS does not offer.

Clinical context

You will not need to memorize syntax. Health professionals use Python like clinicians use a stethoscope — you learn the parts you need, and you look up the rest. Every code block in this course can be copied into a Jupyter notebook and run immediately.

1.2 Setting up your environment

1.2.1 Google Colab (recommended)

No installation. Go to `https://colab.research.google.com`, sign in with a Google account, and create a new notebook. Every library we use is pre-installed.

1.2.2 Local installation with Anaconda

If you prefer working offline:

1. Download Anaconda: https://www.anaconda.com/download
2. Install with default settings.
3. Open **Jupyter Notebook** from Anaconda Navigator.

1.3 Your first notebook

A Jupyter notebook is a document that mixes text, code, and output. Each *cell* contains either:

- **Code** — Python instructions that produce a result.
- **Markdown** — formatted text (explanations, headings, notes).

Press `Shift + Enter` to run a cell.

```
# Your first Python cell
print("Hello, health data!")
```

1.4 Variables and types

A *variable* stores a value. In health data, you work with four types constantly:

```
patient_age = 45                # int (integer)
temperature = 37.2              # float (decimal number)
diagnosis = "Type 2 Diabetes" # str (string / text)
is_hospitalized = True          # bool (True or False)
```

Python tip

Variable names should be descriptive. Use `patient_age`, not `x`. Your future self (and your collaborators) will thank you.

1.5 Collections: lists and dictionaries

1.5.1 Lists

A list holds multiple values in order:

```
blood_pressures = [120, 135, 128, 142, 118]
print(f"Number of readings: {len(blood_pressures)}")
print(f"First reading: {blood_pressures[0]}")  # indexing starts at 0
print(f"Last reading: {blood_pressures[-1]}")
```

1.5.2 Dictionaries

A dictionary maps keys to values — like a patient chart:

```
patient = {
    "id": "PAT-0042",
    "age": 58,
    "sex": "F",
    "diagnosis": "Hypertension",
    "systolic_bp": 148,
    "medications": ["Amlodipine", "Hydrochlorothiazide"]
}
print(patient["diagnosis"])  # Hypertension
```

1.6 Control flow

1.6.1 Conditionals

```
systolic = 148

if systolic >= 140:
    category = "Stage 2 Hypertension"
elif systolic >= 130:
    category = "Stage 1 Hypertension"
elif systolic >= 120:
    category = "Elevated"
else:
    category = "Normal"

print(f"BP category: {category}")
```

1.6.2 Loops

```
patients = [
    {"id": "P001", "age": 34, "hba1c": 5.4},
    {"id": "P002", "age": 67, "hba1c": 7.8},
    {"id": "P003", "age": 52, "hba1c": 6.1},
]

for p in patients:
    if p["hba1c"] >= 6.5:
        print(f"{p['id']}: Diabetic (HbA1c = {p['hba1c']})")
    elif p["hba1c"] >= 5.7:
        print(f"{p['id']}: Pre-diabetic (HbA1c = {p['hba1c']})")
    else:
        print(f"{p['id']}: Normal (HbA1c = {p['hba1c']})")
```

1.7 Functions

A function encapsulates a reusable computation:

```
def bmi(weight_kg, height_m):
    """Calculate Body Mass Index."""
    return weight_kg / (height_m ** 2)

def bmi_category(bmi_value):
    """Classify BMI according to WHO categories."""
    if bmi_value < 18.5:
        return "Underweight"
    elif bmi_value < 25.0:
        return "Normal weight"
    elif bmi_value < 30.0:
        return "Overweight"
    else:
        return "Obese"

value = bmi(82, 1.75)
print(f"BMI: {value:.1f} ({bmi_category(value)})")
```

Exercise

1. Write a function `classify_bp(systolic, diastolic)` that returns a blood pressure category according to the AHA/ACC guidelines.
2. Create a list of 5 patient dictionaries with `weight_kg` and `height_m` fields. Loop through them, compute BMI, and print the category for each.
3. Write a function `egfr(creatinine, age, sex)` that estimates glomerular filtration rate using the CKD-EPI equation.

1.8 Importing libraries

Python's power comes from libraries. Here are the ones we will use throughout this course:

```
import pandas as pd          # data manipulation
import numpy as np           # numerical computing
import matplotlib.pyplot as plt  # plotting
import seaborn as sns        # statistical visualization
from scipy import stats      # statistical tests
```

Caution

If you get `ModuleNotFoundError`, install the missing library:

```
pip install pandas matplotlib seaborn scipy
```

In Google Colab, prefix with `!`: `!pip install lifelines`

1.9 Chapter summary

- Python is free, reproducible, and has health-specific libraries.
- Jupyter notebooks mix code, text, and output in one document.
- Variables store values; lists and dictionaries organize them.
- Functions make your analysis reusable and readable.
- `pandas`, `matplotlib`, `seaborn`, and `scipy` are the core libraries for health data analysis.

Chapter 2

Health data with Pandas

> *"80% of data analysis is getting the data into a shape where you can ask it questions."*

2.1 What is a DataFrame?

A DataFrame is a table — rows are observations (patients, countries, time points), columns are variables (age, diagnosis, lab values). If you have used Excel, you already understand the concept. Pandas gives you Excel's power with Python's reproducibility.

```
import pandas as pd

# Create a small clinical DataFrame
data = {
    "patient_id": ["P001", "P002", "P003", "P004", "P005"],
    "age": [45, 67, 34, 52, 71],
    "sex": ["M", "F", "F", "M", "F"],
    "systolic_bp": [128, 158, 112, 145, 162],
    "hba1c": [5.4, 7.8, 5.1, 6.3, 8.2],
    "diagnosis": ["None", "T2DM", "None", "Pre-DM", "T2DM"]
}
df = pd.DataFrame(data)
df
```

2.2 Loading real health data

2.2.1 CSV files

Most health datasets come as CSV (comma-separated values) files:

```
# WHO life expectancy data (Gapminder)
url =
↪  "https://raw.githubusercontent.com/datasets/gapminder/main/data/gapminder.csv"
gapminder = pd.read_csv(url)
print(f"Shape: {gapminder.shape}")  # (rows, columns)
```

```
gapminder.head()
```

2.2.2 Excel files

```
df = pd.read_excel("patient_records.xlsx", sheet_name="Lab Results")
```

2.2.3 From the WHO Global Health Observatory

```
# Maternal mortality ratio by country
url = "https://apps.who.int/gho/athena/api/GHO/MDG_0000000026?format=csv"
mmr = pd.read_csv(url)
mmr.head()
```

Clinical context

The WHO GHO API provides over 1 000 health indicators for 194 member states. You can browse indicators at https://www.who.int/data/gho/data/indicators. Each indicator has a code (e.g., `MDG_0000000026` for maternal mortality ratio) that you plug into the API URL.

2.3 Exploring your data

The first thing you do with any health dataset — before any analysis — is *look at it*.

```
# Basic exploration
print(gapminder.shape)           # dimensions
print(gapminder.columns.tolist())  # column names
print(gapminder.dtypes)          # data types
gapminder.describe()             # summary statistics
gapminder.info()                 # non-null counts and memory
```

2.3.1 Selecting columns

```
# Single column (returns a Series)
ages = gapminder["lifeExp"]

# Multiple columns (returns a DataFrame)
subset = gapminder[["country", "year", "lifeExp"]]
```

2.3.2 Filtering rows

```
# Patients with HbA1c >= 6.5 (diabetic threshold)
diabetic = df[df["hba1c"] >= 6.5]

# African countries in Gapminder
africa = gapminder[gapminder["continent"] == "Africa"]

# Multiple conditions: female patients over 60 with high BP
high_risk = df[(df["sex"] == "F") & (df["age"] > 60) & (df["systolic_bp"] >=
↪  140)]
```

> **Caution**
>
> Use `&` (and), `|` (or), `~` (not) with parentheses around each condition. Python's `and`/`or` keywords do not work with DataFrames.

2.4 Sorting and ranking

```
# Countries with highest life expectancy in 2007
recent = gapminder[gapminder["year"] == 2007]
top10 = recent.sort_values("lifeExp", ascending=False).head(10)
print(top10[["country", "lifeExp"]])
```

2.5 Grouping and aggregation

Grouping is how you answer questions like "what is the average life expectancy per continent?" or "how many patients in each diagnosis category?"

```
# Average life expectancy by continent (2007)
recent.groupby("continent")["lifeExp"].mean().sort_values(ascending=False)
```

```
# Multiple aggregations
recent.groupby("continent")["lifeExp"].agg(["mean", "median", "std", "count"])
```

```
# Patient counts by diagnosis
df.groupby("diagnosis")["patient_id"].count()
```

2.6 Creating new columns

```
# BMI from weight and height
```

```python
df["bmi"] = df["weight_kg"] / (df["height_m"] ** 2)

# Age group
df["age_group"] = pd.cut(df["age"],
                         bins=[0, 18, 40, 60, 100],
                         labels=["<18", "18-39", "40-59", "60+"])

# Binary flag
df["hypertensive"] = (df["systolic_bp"] >= 140).astype(int)
```

2.7 Saving your results

```python
# Save to CSV
df.to_csv("cleaned_patients.csv", index=False)

# Save to Excel
df.to_excel("cleaned_patients.xlsx", index=False, sheet_name="Patients")
```

2.8 Practical: WHO life expectancy analysis

Exercise

Using the Gapminder dataset:

1. Load the data and display basic statistics.
2. Filter to African countries only. How many unique countries?
3. Compute mean life expectancy by decade (1950s, 1960s, ..., 2000s) for Africa vs. Europe.
4. Find the 5 African countries with the lowest life expectancy in 2007.
5. Create a column `life_exp_category`: "Low" (< 55), "Medium" (55–70), "High" (> 70).
6. Group by continent and `life_exp_category`. How many countries in each?
7. Save the Africa-only data to a CSV file.

2.9 Chapter summary

- A DataFrame is a table. Rows = observations, columns = variables.
- `pd.read_csv()` loads data; `.head()`, `.describe()`, `.info()` explore it.
- Filter with boolean conditions: `df[df["col"] > value]`.

- `.groupby()` + `.agg()` answers "per group" questions.
- `pd.cut()` creates categorical variables from continuous ones.
- Always save cleaned data for reproducibility.

Chapter 3

Data cleaning in health contexts

> *"Garbage in, garbage out. In health data, garbage in can mean wrong treatments, wrong policies, and wrong conclusions about human lives."*

3.1 Why data cleaning matters in health

Every health dataset has problems. Lab values get entered with wrong units. Patients appear twice under different IDs. Dates are recorded as `03/04/2023` — is that March 4 or April 3? A blood glucose of 5000 mg/dL is clearly an error, but a glucose of 500 mg/dL might be a real diabetic emergency.

Data cleaning is not optional pre-processing. It is a clinical reasoning task. You must decide:

- Is this value *wrong* or just *unusual*?
- Should this record be *corrected*, *flagged*, or *removed*?
- Will my cleaning decisions *bias* the analysis?

Clinical context

In a 2019 study of electronic health records from a large US hospital system, researchers found that 18% of lab values had at least one quality issue — duplicate entries, physiologically impossible values, or inconsistent units. Cleaning decisions changed the estimated prevalence of chronic kidney disease by over 3 percentage points.

3.2 Missing data: types and mechanisms

3.2.1 The three mechanisms

Not all missing data is the same. The mechanism determines what you can do about it.

1. **MCAR — Missing Completely At Random.** The missingness has nothing to do with the data. Example: a lab tube is dropped and the sample is lost. The probability of losing the sample is unrelated to the patient's health.

2. **MAR — Missing At Random.** The missingness depends on *observed* variables but not the missing value itself. Example: younger patients are less likely to have cholesterol measured (because clinicians order it less often for young patients). Once you account for age, the missingness is random.

3. **MNAR — Missing Not At Random.** The missingness depends on the *unobserved* value itself. Example: very sick patients are too ill to attend follow-up appointments, so their outcome data is missing *because* they are sick. This is the hardest case.

> **⚠ Caution**
>
> MNAR is common in health data and dangerous. If you drop all patients with missing follow-up data, you are systematically excluding the sickest patients — your analysis will overestimate treatment success. There is no statistical fix for MNAR; you need domain knowledge to assess the likely direction and magnitude of bias.

3.2.2 Clinical examples of each mechanism

Type	Example	Consequence of dropping
MCAR	Lab sample hemolyzed due to transport vibration	No bias, but reduced sample size
MAR	HbA1c not ordered for non-diabetic patients	Bias if you study HbA1c without accounting for indication
MNAR	Severely depressed patients skip follow-up questionnaires	Underestimate depression severity in remaining data

3.3 Detecting missing values with pandas

3.3.1 Basic detection

```
import pandas as pd
import numpy as np

# Create a clinical dataset with missing values
data = {
    "patient_id": ["P001", "P002", "P003", "P004", "P005", "P006"],
    "age": [45, 67, 34, np.nan, 71, 52],
    "sex": ["M", "F", "F", "M", np.nan, "F"],
    "systolic_bp": [128, 158, np.nan, 145, 162, np.nan],
    "hba1c": [5.4, 7.8, 5.1, np.nan, 8.2, 6.3],
    "smoking": ["Never", np.nan, "Current", "Former", np.nan, "Never"]
}
df = pd.DataFrame(data)
```

```python
# Count missing values per column
print(df.isnull().sum())

# Percentage missing per column
print((df.isnull().sum() / len(df) * 100).round(1))

# Rows with any missing value
print(f"Rows with missing data: {df.isnull().any(axis=1).sum()} / {len(df)}")
```

Python tip

`.isnull()` and `.isna()` are identical in pandas. Use whichever reads better to you. Both detect `NaN`, `None`, and `NaT` (missing datetime).

3.3.2 Visualizing missingness with msno

The `missingno` library provides instant visual summaries of missing data patterns:

```python
import missingno as msno
import matplotlib.pyplot as plt

# Matrix plot: white bars = missing
msno.matrix(df)
plt.title("Missing data pattern")
plt.tight_layout()
plt.show()

# Bar chart: count of non-null values per column
msno.bar(df)
plt.tight_layout()
plt.show()

# Heatmap: correlations between missingness of different columns
# If two columns tend to be missing together, they will show high correlation
msno.heatmap(df)
plt.tight_layout()
plt.show()
```

Caution

If `missingno` is not installed, run `!pip install missingno` in your notebook. In Google Colab it is not pre-installed.

3.3.3 Patterns of co-missingness

When two variables are often missing together, it suggests a common cause. For example, if both `systolic_bp` and `diastolic_bp` are missing for the same patients, it likely means the BP measurement was not taken at all — not that each value was independently lost.

```python
# Check which columns are missing together
missing_pairs = df.isnull().corr()
print(missing_pairs)
```

3.4 Imputation strategies

3.4.1 When to impute vs. when to drop

- **Drop rows** if missingness is MCAR and the missing fraction is small ($< 5\%$).
- **Drop columns** if $> 50\%$ of values are missing and the variable is not critical.
- **Impute** when you need to retain sample size and you have a reasonable strategy.
- **Flag + sensitivity analysis** for MNAR data.

```python
# Drop rows with any missing value
df_complete = df.dropna()
print(f"Rows remaining: {len(df_complete)} / {len(df)}")

# Drop rows only if specific columns are missing
df_partial = df.dropna(subset=["age", "systolic_bp"])
```

3.4.2 Mean/median imputation for continuous variables

```python
# Median imputation for lab values (robust to outliers)
df["systolic_bp"] = df["systolic_bp"].fillna(df["systolic_bp"].median())
df["hba1c"] = df["hba1c"].fillna(df["hba1c"].median())

# Mean imputation for age (approximately symmetric distribution)
df["age"] = df["age"].fillna(df["age"].mean())
```

Clinical context

Use **median** rather than mean for lab values like creatinine, glucose, and triglycerides. These distributions are typically right-skewed — a few very high values pull the mean upward, making it a poor representative of the "typical" patient. The median is resistant to these outliers.

3.4.3 Mode imputation for categorical variables

```python
# Mode imputation for sex and smoking status
df["sex"] = df["sex"].fillna(df["sex"].mode()[0])
df["smoking"] = df["smoking"].fillna(df["smoking"].mode()[0])
```

3.4.4 Group-specific imputation

Sometimes the right imputation value depends on a group. Imputing systolic BP with the overall median ignores the fact that older patients have higher BP:

```
# Impute systolic_bp with the median for the patient's age group
df["age_group"] = pd.cut(df["age"], bins=[0, 40, 60, 100],
                         labels=["<40", "40-59", "60+"])
df["systolic_bp"] = df.groupby("age_group")["systolic_bp"].transform(
    lambda x: x.fillna(x.median())
)
```

3.4.5 KNN imputation

K-nearest neighbors imputation finds the k most similar patients (based on other variables) and uses their values to fill in the missing one:

```
from sklearn.impute import KNNImputer

# Select numeric columns for KNN imputation
numeric_cols = ["age", "systolic_bp", "hba1c"]
imputer = KNNImputer(n_neighbors=5)
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])
```

Python tip

KNN imputation respects relationships between variables. If a patient with high BMI, high age, and high HbA1c is missing systolic BP, KNN will impute from similar patients — who likely also have high BP. This is more realistic than using the overall median.

3.4.6 Creating a missingness indicator

For important variables, create a flag so you can later test whether missingness itself is associated with outcomes:

```
# Before imputing, create a flag
df["bp_was_missing"] = df["systolic_bp"].isnull().astype(int)

# Then impute
df["systolic_bp"] = df["systolic_bp"].fillna(df["systolic_bp"].median())
```

3.5 Handling duplicates

3.5.1 Detecting duplicates

```
# Load a dataset with deliberate duplicates
```

```
patients = pd.DataFrame({
    "patient_id": ["P001", "P002", "P003", "P002", "P004", "P003"],
    "visit_date": ["2023-01-15", "2023-01-16", "2023-01-17",
                   "2023-01-16", "2023-01-18", "2023-02-20"],
    "systolic_bp": [128, 158, 112, 158, 145, 118],
    "diagnosis": ["HTN", "T2DM", "None", "T2DM", "HTN", "None"]
})

# Exact duplicates (all columns identical)
print(f"Exact duplicates: {patients.duplicated().sum()}")
print(patients[patients.duplicated(keep=False)])  # show all copies

# Duplicates based on patient_id only
print(f"\nDuplicate patient IDs:
↪  {patients.duplicated(subset='patient_id').sum()}")
```

3.5.2 Deciding what counts as a duplicate

In health data, "duplicate" is not always straightforward:

Scenario	Appropriate action
Same patient, same visit, identical values	True duplicate — remove one copy
Same patient, same visit, different values	Data entry error — investigate which is correct
Same patient, different visit dates	Repeated measures — keep all (this is longitudinal data)
Different patient IDs, same demographics	Possible duplicate registration — flag for manual review

```
# Remove exact duplicates
patients_clean = patients.drop_duplicates()
print(f"After removing exact duplicates: {len(patients_clean)} rows")

# Keep only the first visit per patient
first_visits = patients.drop_duplicates(subset="patient_id", keep="first")
print(f"First visits only: {len(first_visits)} rows")

# Keep only the last visit per patient
last_visits = patients.drop_duplicates(subset="patient_id", keep="last")
```

Clinical context

In clinical registries, duplicate patient records are a major safety issue. A patient registered under two IDs may receive conflicting prescriptions. Deduplication in production systems uses probabilistic matching on name, date of birth, and address — far beyond what simple `drop_duplicates()` can do.

3.6 Inconsistent data

3.6.1 ICD code formats

The International Classification of Diseases uses codes like `E11.9` (Type 2 diabetes without complications). In messy data, you will find the same condition recorded as `E11.9`, `E119`, `e11.9`, or even `250.00` (the old ICD-9 code).

```
diagnoses = pd.Series(["E11.9", "e11.9", "E119", "E11.9 ", " e11.9"])

# Standardize: uppercase, strip whitespace, ensure dot format
cleaned = (diagnoses
           .str.strip()
           .str.upper()
           .str.replace(r"^([A-Z]\d{2})(\d+)$", r"\1.\2", regex=True))
print(cleaned)
```

3.6.2 Date formats

```
dates = pd.Series(["2023-01-15", "01/15/2023", "15-Jan-2023",
                   "Jan 15, 2023", "20230115"])

# pd.to_datetime handles multiple formats automatically
parsed = pd.to_datetime(dates, format="mixed", dayfirst=False)
print(parsed)

# Standardize to ISO format
print(parsed.dt.strftime("%Y-%m-%d"))
```

> **Caution**
>
> The date `03/04/2023` is March 4 in the US but April 3 in Europe. Always check the source. If your dataset mixes conventions, you will get silent errors — the date will parse without warning, but it will be *wrong*. Use `dayfirst=True` for European-format dates.

3.6.3 Unit conversions

Different labs report results in different units. The most common confusion:

Analyte	US unit	SI unit	Conversion
Glucose	mg/dL	mmol/L	divide by 18.018
Cholesterol	mg/dL	mmol/L	divide by 38.67
Creatinine	mg/dL	µmol/L	multiply by 88.42
Hemoglobin	g/dL	g/L	multiply by 10

```python
# Standardize glucose to mmol/L
# Assume values > 30 are in mg/dL (glucose in mmol/L is rarely > 30)
def standardize_glucose(value, unit=None):
    """Convert glucose to mmol/L. Infer unit if not provided."""
    if unit == "mg/dL" or (unit is None and value > 30):
        return value / 18.018
    return value  # already in mmol/L

df["glucose_mmol"] = df["glucose"].apply(
    lambda x: standardize_glucose(x)
)
```

Python tip

When a dataset has no unit column, use the value range to infer the unit. Fasting glucose in mg/dL is typically 70–300; in mmol/L it is 3.9–16.7. If you see a value of 126, it is almost certainly mg/dL. If you see 7.0, it is almost certainly mmol/L.

3.6.4 Standardizing text fields

```python
# Sex/gender field with inconsistent entries
sex_col = pd.Series(["Male", "male", "M", "m", "MALE", "Female",
                     "female", "F", "f", "FEMALE"])

# Map to standard values
sex_mapping = {"male": "M", "m": "M", "female": "F", "f": "F"}
sex_clean = sex_col.str.strip().str.lower().map(sex_mapping)
print(sex_clean)
```

3.7 Data validation

After cleaning, validate that your data makes clinical sense.

3.7.1 Range checks

```python
# Define clinically plausible ranges
ranges = {
    "age": (0, 120),
    "systolic_bp": (60, 300),
    "diastolic_bp": (30, 200),
    "heart_rate": (20, 300),
    "hba1c": (2.0, 20.0),
    "glucose_mgdl": (10, 1500),
    "bmi": (8, 80),
    "temperature_c": (25, 45),
}
```

```
def validate_range(df, column, low, high):
    """Flag values outside the plausible range."""
    out_of_range = df[(df[column] < low) | (df[column] > high)]
    if len(out_of_range) > 0:
        print(f"WARNING: {len(out_of_range)} values in '{column}' "
              f"outside [{low}, {high}]")
        print(out_of_range[["patient_id", column]])
    return out_of_range

# Apply range checks
for col, (low, high) in ranges.items():
    if col in df.columns:
        validate_range(df, col, low, high)
```

3.7.2 Cross-field validation

Some errors only become visible when you compare columns:

```
# Diastolic should be lower than systolic
bp_errors = df[df["diastolic_bp"] >= df["systolic_bp"]]
if len(bp_errors) > 0:
    print(f"WARNING: {len(bp_errors)} rows where diastolic >= systolic")

# Pregnancy flag should only be 1 for female patients
preg_errors = df[(df["pregnant"] == 1) & (df["sex"] == "M")]
if len(preg_errors) > 0:
    print(f"WARNING: {len(preg_errors)} male patients flagged as pregnant")

# Death date should be after admission date
date_errors = df[df["death_date"] < df["admission_date"]]
if len(date_errors) > 0:
    print(f"WARNING: {len(date_errors)} deaths recorded before admission")
```

Clinical context

Not every "impossible" value is an error. A heart rate of 250 bpm is physiologically extreme but can occur in supraventricular tachycardia. A glucose of 800 mg/dL is rare but real in diabetic ketoacidosis. Domain knowledge tells you whether to remove, flag, or keep.

3.7.3 Automated validation report

```
def data_quality_report(df):
    """Generate a summary of data quality issues."""
    report = []
    for col in df.columns:
        n_missing = df[col].isnull().sum()
```

```
        pct_missing = n_missing / len(df) * 100
        n_unique = df[col].nunique()

        row = {
            "column": col,
            "dtype": str(df[col].dtype),
            "n_missing": n_missing,
            "pct_missing": round(pct_missing, 1),
            "n_unique": n_unique,
        }

        if df[col].dtype in ["float64", "int64"]:
            row["min"] = df[col].min()
            row["max"] = df[col].max()
            row["mean"] = round(df[col].mean(), 2)

        report.append(row)

    return pd.DataFrame(report)

quality = data_quality_report(df)
print(quality.to_string(index=False))
```

3.8 A complete cleaning pipeline

Here is a realistic pipeline applied to NHANES-style data:

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer

# ------------------------------------------------------------
# Step 1: Load data
# ------------------------------------------------------------
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)
print(f"Raw data: {df.shape}")

# ------------------------------------------------------------
# Step 2: Standardize column names
# ------------------------------------------------------------
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")

# ------------------------------------------------------------
# Step 3: Remove exact duplicates
# ------------------------------------------------------------
n_before = len(df)
df = df.drop_duplicates()
print(f"Duplicates removed: {n_before - len(df)}")
```

```
# ------------------------------------------------------------
# Step 4: Assess missing data
# ------------------------------------------------------------
missing_pct = (df.isnull().sum() / len(df) * 100).round(1)
print("Missing percentages:")
print(missing_pct[missing_pct > 0])

# ------------------------------------------------------------
# Step 5: Validate ranges
# ------------------------------------------------------------
# Life expectancy should be between 20 and 90
outliers = df[(df["lifeexp"] < 20) | (df["lifeexp"] > 90)]
print(f"Life expectancy outliers: {len(outliers)}")

# ------------------------------------------------------------
# Step 6: Save cleaned data
# ------------------------------------------------------------
df.to_csv("gapminder_cleaned.csv", index=False)
print(f"Cleaned data saved: {df.shape}")
```

3.9 Practical: cleaning a messy clinical dataset

Exercise

Download or create a messy clinical dataset with the following deliberate errors, then clean it step by step:

```
import pandas as pd
import numpy as np

# Messy clinical data with deliberate errors
messy = pd.DataFrame({
    "patient_id": ["P001", "P002", "P003", "P004", "P002",
                   "P005", "P006", "P007", "P008", "P009"],
    "age": [45, 67, -3, 52, 67, 200, 38, np.nan, 71, 44],
    "sex": ["M", "F", "Female", "m", "F",
            "Male", np.nan, "F", "X", "M"],
    "systolic_bp": [128, 158, 112, 450, 158,
                    135, 122, np.nan, 162, 118],
    "diastolic_bp": [82, 160, 74, 92, 160,
                     88, 78, np.nan, 98, 76],
    "glucose_mgdl": [95, 7.2, 110, 180, 95,
                     88, 102, 6.8, 220, 5.5],
    "hba1c": [5.4, 7.8, 5.1, 6.3, 7.8,
              np.nan, 5.8, 6.1, 55, 5.2],
    "visit_date": ["2023-01-15", "01/16/2023", "2023-01-17",
                   "2023/01/18", "2023-01-16",
                   "15-Jan-2023", "2023-01-20", "2023-01-21",
```

```
                    "2023-01-22", "Jan 23, 2023"],
    "icd_code": ["I10", "E119", "e11.9", "I10 ", " i10",
                 "E11.9", "I10", "e119", "I10", "E11.9"]
})
```

Tasks:

1. Generate a data quality report: count missing values, check data types, compute summary statistics.
2. Identify and remove exact duplicate rows (P002 appears twice with identical data).
3. Fix the `sex` column: standardize to "M" and "F". Decide what to do with "X" and `NaN`.
4. Validate `age`: flag or correct the impossible values (-3 and 200). Set them to `NaN`.
5. Validate `systolic_bp`: a value of 450 is impossible. Set to `NaN`.
6. Check that diastolic $<$ systolic for all patients. Flag violations.
7. Fix `glucose_mgdl`: some values (7.2, 6.8, 5.5) are clearly in mmol/L. Convert them to mg/dL by multiplying by 18.018.
8. Fix `hba1c`: the value 55 is likely in mmol/mol (IFCC) rather than % (NGSP). Convert using: NGSP $= 0.0915 \times$ IFCC $+ 2.15$.
9. Standardize `visit_date` to ISO format (YYYY-MM-DD).
10. Standardize `icd_code`: uppercase, stripped, with dot separator.
11. Impute remaining missing values: median for continuous, mode for categorical.
12. Generate a final data quality report and compare with the initial one.
13. Save the cleaned dataset to `cleaned_clinical_data.csv`.

3.10 Chapter summary

- Missing data in health has three mechanisms: MCAR, MAR, and MNAR. The mechanism determines the appropriate strategy.
- Use `.isnull().sum()` for quick counts and `missingno` for visual patterns.
- Imputation options: mean/median for continuous data, mode for categorical, KNN for multivariate imputation.
- Duplicates in health data may be true duplicates, data entry errors, or repeated visits — each requires a different response.

- Inconsistent coding (ICD formats, date formats, units) is ubiquitous and must be standardized.
- Validate data against clinically plausible ranges *after* cleaning.
- Always create a data quality report before and after cleaning so you can document every decision.

CHAPTER 3. DATA CLEANING IN HEALTH CONTEXTS

Chapter 4

Descriptive statistics and epidemiological measures

> *"Before you fit a model, describe the data. Before you describe the data, look at the data. Epidemiology begins with counting."*

4.1 Central tendency: what is "typical"?

4.1.1 Mean, median, and when to use which

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

# Fasting blood glucose for 10 patients (mg/dL)
glucose = pd.Series([92, 98, 104, 95, 88, 110, 97, 340, 101, 93])

print(f"Mean:   {glucose.mean():.1f} mg/dL")
print(f"Median: {glucose.median():.1f} mg/dL")
```

The mean is 121.8 mg/dL. The median is 97.5 mg/dL. One patient with a glucose of 340 (diabetic ketoacidosis) pulls the mean up by nearly 25 mg/dL. The median is unaffected.

Clinical context

Rule of thumb for health data: use the **median** when the distribution is skewed (most lab values, length of stay, health expenditures). Use the **mean** when the distribution is approximately symmetric (height, diastolic blood pressure in healthy adults).

4.1.2 Mode

The mode is the most frequent value. It is the only measure of central tendency for categorical data:

```
diagnoses = pd.Series(["HTN", "T2DM", "HTN", "None", "HTN",
                       "T2DM", "None", "HTN", "COPD", "HTN"])
print(f"Most common diagnosis: {diagnoses.mode()[0]}")
print(f"Frequency: {diagnoses.value_counts().iloc[0]}/{len(diagnoses)}")
```

4.2 Spread: how variable is the data?

4.2.1 Standard deviation and variance

```
# Blood pressure readings from two clinics
clinic_a = pd.Series([120, 122, 118, 125, 121, 119, 123, 120])
clinic_b = pd.Series([98, 145, 110, 155, 102, 160, 115, 135])

print(f"Clinic A: mean={clinic_a.mean():.1f}, SD={clinic_a.std():.1f}")
print(f"Clinic B: mean={clinic_b.mean():.1f}, SD={clinic_b.std():.1f}")
```

Both clinics have similar means ($\approx$121), but Clinic B has much higher variability. A physician looking only at the mean would miss that Clinic B has patients with dangerously high *and* unusually low blood pressure.

4.2.2 IQR and percentiles

The interquartile range (IQR) is the range between the 25th and 75th percentiles. Like the median, it is robust to outliers.

```
# Length of hospital stay (days) --- typically right-skewed
los = pd.Series([2, 3, 3, 4, 4, 5, 5, 6, 7, 8, 12, 28, 45])

q25 = los.quantile(0.25)
q75 = los.quantile(0.75)
iqr = q75 - q25

print(f"Median LOS: {los.median():.0f} days")
print(f"IQR: {q25:.0f}--{q75:.0f} days (range: {iqr:.0f})")
print(f"Mean LOS: {los.mean():.1f} days")  # inflated by 28 and 45
```

Python tip

Pandas `.describe()` gives you count, mean, std, min, 25%, 50%, 75%, and max in one call. It is the fastest way to get a statistical snapshot of every numeric column.

4.2.3 Coefficient of variation

The coefficient of variation (CV) expresses the standard deviation as a percentage of the mean. It is useful for comparing variability between measurements with different units or scales:

```python
# Compare variability of glucose (mg/dL) vs hemoglobin (g/dL)
glucose = pd.Series([95, 102, 88, 110, 97, 105, 92, 98])
hemoglobin = pd.Series([13.2, 14.1, 12.8, 13.5, 14.0, 13.8, 12.5, 13.9])

cv_glucose = (glucose.std() / glucose.mean()) * 100
cv_hb = (hemoglobin.std() / hemoglobin.mean()) * 100

print(f"CV glucose: {cv_glucose:.1f}%")
print(f"CV hemoglobin: {cv_hb:.1f}%")
```

4.3 Frequency tables and cross-tabulations

4.3.1 One-way frequency tables

```python
# Load Gapminder data
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
gapminder = pd.read_csv(url)
recent = gapminder[gapminder["year"] == 2007]

# Create life expectancy categories
recent = recent.copy()
recent["le_cat"] = pd.cut(recent["lifeExp"],
                          bins=[0, 55, 70, 85],
                          labels=["Low (<55)", "Medium (55-70)",
                                  "High (>70)"])

# Frequency table
freq = recent["le_cat"].value_counts().sort_index()
print(freq)

# With proportions
freq_table = pd.DataFrame({
    "n": freq,
    "pct": (freq / freq.sum() * 100).round(1),
    "cum_pct": (freq.cumsum() / freq.sum() * 100).round(1)
})
print(freq_table)
```

4.3.2 Two-way cross-tabulations

```python
# Cross-tabulate life expectancy category by continent
ct = pd.crosstab(recent["continent"], recent["le_cat"], margins=True)
print(ct)

# With row percentages (percentage within each continent)
ct_pct = pd.crosstab(recent["continent"], recent["le_cat"],
```

```
                        normalize="index").round(3) * 100
print(ct_pct)
```

Clinical context

Cross-tabulations are the workhorse of descriptive epidemiology. Every "Table 1" in a clinical paper is essentially a cross-tabulation of patient characteristics by study group. In pandas, `pd.crosstab()` builds them in one line.

4.4 Epidemiological measures: counting disease

4.4.1 Prevalence

Prevalence is the *proportion* of a population that has a condition at a given point in time:

$$\text{Prevalence} = \frac{\text{Number of existing cases}}{\text{Total population}} \times 100$$

```
# Diabetes prevalence from a health survey
n_total = 5000
n_diabetic = 625

prevalence = n_diabetic / n_total * 100
print(f"Diabetes prevalence: {prevalence:.1f}%")

# With 95% confidence interval (normal approximation)
from scipy import stats

p = n_diabetic / n_total
se = np.sqrt(p * (1 - p) / n_total)
ci_low = (p - 1.96 * se) * 100
ci_high = (p + 1.96 * se) * 100
print(f"95% CI: [{ci_low:.1f}%, {ci_high:.1f}%]")
```

4.4.2 Incidence rate

Incidence measures *new* cases over a period of time. The incidence rate uses person-time in the denominator:

$$\text{Incidence rate} = \frac{\text{Number of new cases}}{\text{Total person-time at risk}}$$

```
# Tuberculosis cohort: 200 patients followed for varying durations
new_tb_cases = 18
total_person_years = 850  # sum of follow-up time for all patients

incidence_rate = new_tb_cases / total_person_years
print(f"TB incidence rate: {incidence_rate:.4f} per person-year")
print(f"                 = {incidence_rate * 1000:.1f} per 1,000 person-years")
```

```python
# 95% CI for incidence rate (Poisson approximation)
ir_se = np.sqrt(new_tb_cases) / total_person_years
ci_low = (incidence_rate - 1.96 * ir_se) * 1000
ci_high = (incidence_rate + 1.96 * ir_se) * 1000
print(f"95% CI: [{ci_low:.1f}, {ci_high:.1f}] per 1,000 person-years")
```

Caution

Prevalence and incidence answer different questions. Prevalence tells you the *burden* of disease (how many people are sick right now). Incidence tells you the *risk* of getting sick. A disease can have low incidence but high prevalence if patients survive a long time (e.g., Type 2 diabetes). A disease with high incidence and low prevalence kills quickly (e.g., Ebola).

4.4.3 Mortality rate

```python
# Crude mortality rate
deaths = 1250
population = 500_000
mortality_rate = deaths / population * 100_000
print(f"Crude mortality rate: {mortality_rate:.1f} per 100,000")

# Case fatality rate (CFR)
total_cases = 3200
deaths_among_cases = 128
cfr = deaths_among_cases / total_cases * 100
print(f"Case fatality rate: {cfr:.1f}%")
```

Clinical context

The case fatality rate (CFR) is often misinterpreted during epidemics. Early in an outbreak, the denominator (total cases) is underestimated because many mild cases are not tested. This makes the CFR appear higher than the true infection fatality rate (IFR). During early COVID-19, reported CFR was ~3–4%, but seroprevalence studies later estimated the IFR at ~0.5–1%.

4.5 Measures of association

4.5.1 Risk ratio (relative risk)

The risk ratio compares the probability of disease between exposed and unexposed groups:

$$\text{RR} = \frac{\text{Risk in exposed}}{\text{Risk in unexposed}} = \frac{a/(a+b)}{c/(c+d)}$$

```python
# Smoking and lung cancer (hypothetical cohort)
```

```
# Exposed (smokers):     a=80 cancer, b=920 no cancer
# Unexposed (non-smokers): c=10 cancer, d=990 no cancer
a, b, c, d = 80, 920, 10, 990

risk_exposed = a / (a + b)
risk_unexposed = c / (c + d)
rr = risk_exposed / risk_unexposed

print(f"Risk in smokers:     {risk_exposed:.3f} ({risk_exposed*100:.1f}%)")
print(f"Risk in non-smokers: {risk_unexposed:.3f} ({risk_unexposed*100:.1f}%)")
print(f"Risk Ratio: {rr:.2f}")

# 95% CI for log(RR)
import math
se_ln_rr = math.sqrt(1/a - 1/(a+b) + 1/c - 1/(c+d))
ln_rr = math.log(rr)
ci_low = math.exp(ln_rr - 1.96 * se_ln_rr)
ci_high = math.exp(ln_rr + 1.96 * se_ln_rr)
print(f"95% CI: [{ci_low:.2f}, {ci_high:.2f}]")
```

4.5.2 Odds ratio

The odds ratio is used in case-control studies where you cannot directly compute risk:

$$\text{OR} = \frac{a \times d}{b \times c}$$

```
# Case-control study: obesity and knee osteoarthritis
# Cases (OA):       a=120 obese, c=80 non-obese
# Controls (no OA): b=60 obese,  d=140 non-obese
a, b, c, d = 120, 60, 80, 140

odds_ratio = (a * d) / (b * c)
print(f"Odds Ratio: {odds_ratio:.2f}")

# 95% CI
se_ln_or = math.sqrt(1/a + 1/b + 1/c + 1/d)
ln_or = math.log(odds_ratio)
ci_low = math.exp(ln_or - 1.96 * se_ln_or)
ci_high = math.exp(ln_or + 1.96 * se_ln_or)
print(f"95% CI: [{ci_low:.2f}, {ci_high:.2f}]")
```

Clinical context

The odds ratio approximates the risk ratio when the disease is rare ($< 10\%$ prevalence in both groups). For common outcomes, the OR exaggerates the association. An OR of 2.0 for a condition with 40% baseline prevalence does not mean "twice the risk" — the actual RR would be lower.

4.5.3 Reusable functions

```python
def risk_ratio(a, b, c, d, alpha=0.05):
    """Compute risk ratio with confidence interval.

    a = exposed with disease
    b = exposed without disease
    c = unexposed with disease
    d = unexposed without disease
    """
    from scipy.stats import norm
    z = norm.ppf(1 - alpha / 2)

    rr = (a / (a + b)) / (c / (c + d))
    se = math.sqrt(1/a - 1/(a+b) + 1/c - 1/(c+d))
    ci = (math.exp(math.log(rr) - z * se),
          math.exp(math.log(rr) + z * se))

    return {"RR": round(rr, 3), "CI_low": round(ci[0], 3),
            "CI_high": round(ci[1], 3)}


def odds_ratio(a, b, c, d, alpha=0.05):
    """Compute odds ratio with confidence interval."""
    from scipy.stats import norm
    z = norm.ppf(1 - alpha / 2)

    o_r = (a * d) / (b * c)
    se = math.sqrt(1/a + 1/b + 1/c + 1/d)
    ci = (math.exp(math.log(o_r) - z * se),
          math.exp(math.log(o_r) + z * se))

    return {"OR": round(o_r, 3), "CI_low": round(ci[0], 3),
            "CI_high": round(ci[1], 3)}


# Example
print(risk_ratio(80, 920, 10, 990))
print(odds_ratio(120, 60, 80, 140))
```

4.6 Age-standardized rates

Crude rates can be misleading when comparing populations with different age structures. A country with an older population will have a higher crude mortality rate even if it is "healthier" at every age. Age standardization removes this confounding.

4.6.1 Direct standardization

```python
# Compare crude vs age-standardized mortality between two regions
```

```
# Region A (young population), Region B (old population)

data_a = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "population": [50000, 80000, 40000, 10000],
    "deaths": [50, 80, 200, 300]
})
data_a["rate"] = data_a["deaths"] / data_a["population"] * 100000

data_b = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "population": [15000, 40000, 50000, 45000],
    "deaths": [20, 50, 280, 1500]
})
data_b["rate"] = data_b["deaths"] / data_b["population"] * 100000

# Standard population (e.g., WHO World Standard)
standard_pop = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "std_weight": [0.26, 0.42, 0.22, 0.10]
})

# Crude rates
crude_a = data_a["deaths"].sum() / data_a["population"].sum() * 100000
crude_b = data_b["deaths"].sum() / data_b["population"].sum() * 100000
print(f"Crude rate Region A: {crude_a:.1f} per 100,000")
print(f"Crude rate Region B: {crude_b:.1f} per 100,000")

# Age-standardized rates
asr_a = (data_a["rate"] * standard_pop["std_weight"]).sum()
asr_b = (data_b["rate"] * standard_pop["std_weight"]).sum()
print(f"Age-standardized rate Region A: {asr_a:.1f} per 100,000")
print(f"Age-standardized rate Region B: {asr_b:.1f} per 100,000")
```

Caution

Always report which standard population you used (WHO World Standard, European Standard, US 2000 Standard). Different standards give different results. If you compare your rates with published rates, you must use the same standard population.

4.7 Practical: computing descriptive statistics with real data

4.7.1 Descriptive statistics on Gapminder data

```
import pandas as pd
import numpy as np
from scipy import stats
```

```python
# Load Gapminder
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)

# Summary statistics by continent (2007)
recent = df[df["year"] == 2007].copy()
summary = recent.groupby("continent")["lifeExp"].agg(
    ["count", "mean", "median", "std",
     lambda x: x.quantile(0.25),
     lambda x: x.quantile(0.75)]
)
summary.columns = ["n", "mean", "median", "sd", "Q1", "Q3"]
print(summary.round(1))
```

4.7.2 Computing malaria prevalence by region

```python
# Malaria estimated cases from Our World in Data
url = ("https://raw.githubusercontent.com/owid/"
       "owid-datasets/master/datasets/"
       "Malaria%20incidence%20-%20IHME/Malaria%20incidence%20-%20IHME.csv")
# Alternative: use WHO GHO data directly
# url = "https://apps.who.int/gho/athena/api/GHO/MALARIA_EST_CASES?format=csv"

# If the above URL does not work, use Our World in Data's GitHub
url2 = ("https://raw.githubusercontent.com/owid/etl/master/etl/steps/"
        "data/garden/who/2024-07-30/gho.py")  # metadata only

# For a reliable source, use the preprocessed OWID malaria dataset
owid_url = ("https://raw.githubusercontent.com/owid/"
            "owid-datasets/master/datasets/"
            "Estimated%20malaria%20incidence%20rate%20"
            "(per%201%2C000%20population%20at%20risk)%20-%20WHO/"
            "Estimated%20malaria%20incidence%20rate%20"
            "(per%201%2C000%20population%20at%20risk)%20-%20WHO.csv")

# Simulated malaria data for practice
np.random.seed(42)
countries = (["Nigeria", "DRC", "Mozambique", "Uganda", "Ghana"] * 3 +
             ["India", "Indonesia", "Myanmar", "Pakistan", "Ethiopia"] * 3)
regions = (["West Africa"] * 5 + ["Central Africa"] * 5 +
           ["East Africa"] * 5 +
           ["South Asia"] * 5 + ["Southeast Asia"] * 5 +
           ["East Africa"] * 5)

malaria = pd.DataFrame({
    "country": countries,
    "region": ["West Africa", "Central Africa", "East Africa",
               "East Africa", "West Africa"] * 6,
```

```
    "year": [2018]*5 + [2019]*5 + [2020]*5 +
            [2018]*5 + [2019]*5 + [2020]*5,
    "population_at_risk": np.random.randint(10_000_000, 200_000_000, 30),
    "estimated_cases": np.random.randint(500_000, 50_000_000, 30)
})

# Compute prevalence per 1,000 population at risk
malaria["prevalence_per_1000"] = (
    malaria["estimated_cases"] / malaria["population_at_risk"] * 1000
)

# Summary by region
region_summary = malaria.groupby("region").agg(
    total_cases=("estimated_cases", "sum"),
    total_pop=("population_at_risk", "sum"),
    n_countries=("country", "nunique")
)
region_summary["prevalence_per_1000"] = (
    region_summary["total_cases"] / region_summary["total_pop"] * 1000
)
print(region_summary.round(1))

# Trend by year
year_trend = malaria.groupby("year").agg(
    total_cases=("estimated_cases", "sum"),
    total_pop=("population_at_risk", "sum")
)
year_trend["prevalence_per_1000"] = (
    year_trend["total_cases"] / year_trend["total_pop"] * 1000
)
print(year_trend.round(1))
```

Exercise

Using the Gapminder dataset and the epidemiological functions defined above:

1. Compute summary statistics (mean, median, SD, IQR) for life expectancy, GDP per capita, and population for each continent in 2007.

2. Create a cross-tabulation of continent vs. life expectancy category (Low/Medium/High). Compute row percentages.

3. A cohort of 10 000 people was followed for 5 years. During this time, 450 developed Type 2 diabetes. Compute the cumulative incidence and the incidence rate (assuming an average of 4.5 years of follow-up per person).

4. In a case-control study of 200 cases and 200 controls, 140 cases and 80 controls were exposed to a risk factor. Compute the odds ratio and its 95% CI. Interpret the result.

5. Country A has a crude mortality rate of 350 per 100 000 and Country B has 580 per 100 000. Using the age-specific rates and standard population

provided above, compute age-standardized rates. Does the ranking change after standardization?

6. Using Our World in Data or WHO data, compute the malaria incidence rate for 5 African countries over 3 years. Which country has the highest burden? Is the trend increasing or decreasing?

7. Write a function `prevalence_ci(cases, total, confidence=0.95)` that returns the prevalence and its confidence interval. Test it with: 250 diabetic patients out of 2 000 screened.

4.8 Chapter summary

- Use the **median** for skewed distributions (lab values, costs, length of stay) and the **mean** for symmetric distributions.
- The **IQR** is a robust measure of spread; the **SD** is sensitive to outliers.
- **Prevalence** = existing cases / population. **Incidence rate** = new cases / person-time.
- The **risk ratio** compares risks in cohort studies; the **odds ratio** is used in case-control studies.
- The OR approximates the RR only when the outcome is rare ($< 10\%$).
- **Age standardization** removes confounding by age when comparing populations.
- Always report confidence intervals alongside point estimates.
- `pd.crosstab()` builds the two-way tables that form the basis of epidemiological analysis.

Chapter 5

Visualization for health data

> *"The greatest value of a picture is when it forces us to notice what we never expected to see." — John Tukey*

5.1 Principles of health data visualization

A good health figure answers a question. A bad one decorates a slide. Before writing any plotting code, ask yourself:

1. **What is the message?** "Malaria incidence is declining in Sub-Saharan Africa" — not "here is some data."
2. **Who is the audience?** Clinicians need different plots than policymakers.
3. **What comparison matters?** Between groups? Over time? Against a threshold?

Rules that apply to every health figure:

- Label axes with units ("Systolic BP (mmHg)", not "BP").
- Use colorblind-friendly palettes.
- Do not use 3D charts, pie charts, or dual y-axes unless you have a very good reason.
- Include sample sizes in titles or annotations.
- Start y-axes at zero for bar charts. For line plots, zero is optional if the range is narrow.

Clinical context

In clinical publications, figures are often the first (and sometimes the only) thing reviewers examine closely. A clear, well-labeled figure can carry a paper. A misleading one can sink it. The Lancet and BMJ both have specific figure guidelines — study them before submitting.

5.2 Setup: matplotlib and seaborn

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

# Set a clean style for all plots
sns.set_theme(style="whitegrid", font_scale=1.1)
plt.rcParams["figure.figsize"] = (8, 5)
plt.rcParams["figure.dpi"] = 100

# Colorblind-friendly palette
cb_palette = ["#0072B2", "#D55E00", "#009E73",
              "#CC79A7", "#F0E442", "#56B4E9"]
sns.set_palette(cb_palette)
```

Python tip

The six-color palette above is from Bang Wong's "Points of view: Color blindness" (Nature Methods, 2011). It is distinguishable by people with the most common forms of color vision deficiency. Use it as your default.

5.3 Histograms and density plots

Histograms show the *distribution* of a single variable. In health, this tells you whether a measurement follows a normal distribution, is skewed, or is bimodal.

```
# Load Gapminder data
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
gapminder = pd.read_csv(url)
recent = gapminder[gapminder["year"] == 2007]

# Histogram of life expectancy
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Left: histogram
axes[0].hist(recent["lifeExp"], bins=20, edgecolor="white", alpha=0.8)
axes[0].set_xlabel("Life expectancy (years)")
axes[0].set_ylabel("Number of countries")
axes[0].set_title("Distribution of life expectancy, 2007")
axes[0].axvline(recent["lifeExp"].median(), color="red",
                linestyle="--", label=f'Median:
                ↪  {recent["lifeExp"].median():.1f}')
axes[0].legend()

# Right: kernel density estimate (smooth version)
```

```
sns.kdeplot(data=recent, x="lifeExp", hue="continent",
            fill=True, alpha=0.3, ax=axes[1])
axes[1].set_xlabel("Life expectancy (years)")
axes[1].set_title("Life expectancy by continent, 2007")

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

```
# Clinical example: distribution of fasting glucose
np.random.seed(42)
normal_glucose = np.random.normal(95, 10, 800)
prediabetic = np.random.normal(115, 8, 150)
diabetic = np.random.normal(180, 40, 50)
all_glucose = np.concatenate([normal_glucose, prediabetic, diabetic])

fig, ax = plt.subplots(figsize=(10, 5))
ax.hist(all_glucose, bins=40, edgecolor="white", alpha=0.7)
ax.axvline(100, color="orange", linestyle="--", linewidth=2,
           label="Normal threshold (100 mg/dL)")
ax.axvline(126, color="red", linestyle="--", linewidth=2,
           label="Diabetic threshold (126 mg/dL)")
ax.set_xlabel("Fasting glucose (mg/dL)")
ax.set_ylabel("Number of patients")
ax.set_title("Distribution of fasting glucose (n=1,000)")
ax.legend()
plt.tight_layout()
plt.show()
```

5.4 Box plots

Box plots compare distributions across groups. The box shows the IQR (25th–75th percentile), the line is the median, and the whiskers extend to 1.5×IQR. Points beyond the whiskers are outliers.

```
# Blood pressure by continent
fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data=recent, x="continent", y="lifeExp", ax=ax)
ax.set_xlabel("Continent")
ax.set_ylabel("Life expectancy (years)")
ax.set_title("Life expectancy by continent, 2007 (n={})".format(len(recent)))
plt.tight_layout()
plt.show()
```

```
# Clinical example: BMI by sex and age group
np.random.seed(42)
n = 500
```

```
clinical = pd.DataFrame({
    "sex": np.random.choice(["Male", "Female"], n),
    "age_group": np.random.choice(["18-39", "40-59", "60+"], n,
                                  p=[0.3, 0.4, 0.3]),
    "bmi": np.random.normal(27, 5, n).clip(15, 50)
})

fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data=clinical, x="age_group", y="bmi", hue="sex",
            order=["18-39", "40-59", "60+"], ax=ax)
ax.axhline(25, color="orange", linestyle="--", alpha=0.7,
           label="Overweight threshold")
ax.axhline(30, color="red", linestyle="--", alpha=0.7,
           label="Obese threshold")
ax.set_xlabel("Age group")
ax.set_ylabel("BMI (kg/m²)")
ax.set_title(f"BMI by sex and age group (n={n})")
ax.legend()
plt.tight_layout()
plt.show()
```

Caution

Box plots hide the shape of the distribution. A bimodal distribution looks the same as a unimodal one in a box plot. Consider using a **violin plot** (`sns.violinplot`) or a **strip plot** (`sns.stripplot`) overlaid on the box plot for small samples.

5.5 Bar charts

Bar charts compare *counts* or *rates* across categories. They are the standard choice for disease prevalence comparisons.

```
# Life expectancy comparison: bottom 15 countries (2007)
bottom15 = recent.nsmallest(15, "lifeExp")

fig, ax = plt.subplots(figsize=(10, 7))
ax.barh(bottom15["country"], bottom15["lifeExp"], color="#D55E00")
ax.set_xlabel("Life expectancy (years)")
ax.set_title("15 countries with lowest life expectancy, 2007")
ax.invert_yaxis()  # highest at top
for i, v in enumerate(bottom15["lifeExp"]):
    ax.text(v + 0.5, i, f"{v:.1f}", va="center", fontsize=9)
plt.tight_layout()
plt.show()
```

```
# Disease prevalence comparison by country (simulated)
diseases = pd.DataFrame({
    "country": ["Nigeria", "DRC", "Tanzania", "Kenya", "Ghana",
```

```
                "South Africa", "Ethiopia", "Uganda"],
    "malaria_prev": [27.3, 31.2, 7.5, 5.8, 15.2, 0.5, 1.2, 9.7],
    "hiv_prev": [1.3, 0.7, 4.7, 4.9, 1.7, 19.0, 1.0, 5.4],
})

fig, ax = plt.subplots(figsize=(10, 6))
x = np.arange(len(diseases))
width = 0.35
ax.bar(x - width/2, diseases["malaria_prev"], width,
       label="Malaria", color="#0072B2")
ax.bar(x + width/2, diseases["hiv_prev"], width,
       label="HIV", color="#D55E00")
ax.set_xticks(x)
ax.set_xticklabels(diseases["country"], rotation=45, ha="right")
ax.set_ylabel("Prevalence (%)")
ax.set_title("Malaria vs HIV prevalence by country")
ax.legend()
plt.tight_layout()
plt.show()
```

5.6 Line plots: time trends and epidemic curves

Line plots show how a measurement changes over time. They are essential for tracking disease trends and epidemic progression.

5.6.1 Time trends

```
# Life expectancy trends by continent
continent_trend = gapminder.groupby(
    ["year", "continent"])["lifeExp"].mean().reset_index()

fig, ax = plt.subplots(figsize=(10, 6))
for continent in continent_trend["continent"].unique():
    subset = continent_trend[continent_trend["continent"] == continent]
    ax.plot(subset["year"], subset["lifeExp"],
            marker="o", markersize=4, label=continent)

ax.set_xlabel("Year")
ax.set_ylabel("Mean life expectancy (years)")
ax.set_title("Life expectancy trends by continent, 1952-2007")
ax.legend(title="Continent")
plt.tight_layout()
plt.show()
```

5.6.2 Epidemic curves

```
# Simulate an epidemic curve (SIR-like shape)
np.random.seed(42)
```

```
days = pd.date_range("2023-01-01", periods=120, freq="D")
# Simulate daily new cases with an SIR-like peak
t = np.arange(120)
peak = 45
cases = np.maximum(0, 200 * np.exp(-0.5 * ((t - peak) / 15) ** 2)
                   + np.random.normal(0, 10, 120)).astype(int)

epi = pd.DataFrame({"date": days, "new_cases": cases})
epi["rolling_7d"] = epi["new_cases"].rolling(7).mean()
epi["cumulative"] = epi["new_cases"].cumsum()

fig, axes = plt.subplots(2, 1, figsize=(12, 8), sharex=True)

# Top: daily cases + 7-day average
axes[0].bar(epi["date"], epi["new_cases"], alpha=0.4,
            color="#0072B2", label="Daily cases")
axes[0].plot(epi["date"], epi["rolling_7d"], color="#D55E00",
             linewidth=2, label="7-day average")
axes[0].set_ylabel("New cases per day")
axes[0].set_title("Epidemic curve: daily new cases")
axes[0].legend()

# Bottom: cumulative cases
axes[1].plot(epi["date"], epi["cumulative"], color="#009E73",
             linewidth=2)
axes[1].set_xlabel("Date")
axes[1].set_ylabel("Cumulative cases")
axes[1].set_title("Cumulative case count")

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

Clinical context

During outbreaks, the epidemic curve ("epi curve") is the single most important visualization. Its shape tells you the transmission pattern: a point-source outbreak has a sharp peak; a propagated outbreak has successive waves. The 7-day rolling average smooths out reporting delays and weekend effects.

5.7 Scatter plots

Scatter plots show the relationship between two continuous variables.

```
# GDP per capita vs life expectancy (the classic Gapminder plot)
fig, ax = plt.subplots(figsize=(10, 7))
for continent in recent["continent"].unique():
    subset = recent[recent["continent"] == continent]
    ax.scatter(subset["gdpPercap"], subset["lifeExp"],
               s=subset["pop"] / 1e6,  # bubble size = population
```

```
                alpha=0.6, label=continent)

ax.set_xscale("log")
ax.set_xlabel("GDP per capita (log scale, USD)")
ax.set_ylabel("Life expectancy (years)")
ax.set_title("Wealth vs Health, 2007 (bubble size = population)")
ax.legend(title="Continent")
plt.tight_layout()
plt.show()
```

```
# Clinical scatter: BMI vs systolic BP with regression line
np.random.seed(42)
n = 200
bmi = np.random.normal(28, 5, n).clip(18, 45)
sbp = 80 + 1.8 * bmi + np.random.normal(0, 12, n)

fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(bmi, sbp, alpha=0.5, s=30, color="#0072B2")

# Add regression line
from numpy.polynomial.polynomial import polyfit
b, m = polyfit(bmi, sbp, 1)
x_line = np.linspace(bmi.min(), bmi.max(), 100)
ax.plot(x_line, b + m * x_line, color="#D55E00", linewidth=2,
        label=f"y = {m:.1f}x + {b:.1f}")

# Correlation coefficient
from scipy.stats import pearsonr
r, p = pearsonr(bmi, sbp)
ax.set_xlabel("BMI (kg/m²)")
ax.set_ylabel("Systolic BP (mmHg)")
ax.set_title(f"BMI vs Systolic BP (r={r:.2f}, p<0.001, n={n})")
ax.legend()
plt.tight_layout()
plt.show()
```

Python tip

Seaborn's `sns.regplot()` and `sns.lmplot()` add regression lines and confidence bands automatically. Use `sns.lmplot()` when you want to split by a categorical variable (e.g., regression by sex).

5.8 Kaplan-Meier survival curves

Survival analysis tracks time until an event (death, relapse, infection). The Kaplan-Meier curve estimates the probability of surviving beyond each time point.

```
# Install lifelines if needed: !pip install lifelines
```

```python
from lifelines import KaplanMeierFitter

# Simulated survival data: two treatment groups
np.random.seed(42)
n_per_group = 100

# Treatment group: longer survival
treatment_time = np.random.exponential(24, n_per_group).clip(0, 60)
treatment_event = np.random.binomial(1, 0.7, n_per_group)

# Control group: shorter survival
control_time = np.random.exponential(15, n_per_group).clip(0, 60)
control_event = np.random.binomial(1, 0.8, n_per_group)

fig, ax = plt.subplots(figsize=(10, 6))

# Fit and plot treatment group
kmf_treatment = KaplanMeierFitter()
kmf_treatment.fit(treatment_time, event_observed=treatment_event,
                  label="Treatment (n=100)")
kmf_treatment.plot_survival_function(ax=ax, ci_show=True)

# Fit and plot control group
kmf_control = KaplanMeierFitter()
kmf_control.fit(control_time, event_observed=control_event,
                label="Control (n=100)")
kmf_control.plot_survival_function(ax=ax, ci_show=True)

ax.set_xlabel("Time (months)")
ax.set_ylabel("Survival probability")
ax.set_title("Kaplan-Meier survival curves by treatment group")
ax.set_ylim(0, 1.05)

# Add median survival lines
for kmf, color in [(kmf_treatment, "#0072B2"), (kmf_control, "#D55E00")]:
    median = kmf.median_survival_time_
    if not np.isinf(median):
        ax.axhline(0.5, color="gray", linestyle=":", alpha=0.5)
        ax.axvline(median, color=color, linestyle=":", alpha=0.5)

plt.tight_layout()
plt.show()

# Print median survival
print(f"Median survival (treatment): "
      f"{kmf_treatment.median_survival_time_:.1f} months")
print(f"Median survival (control): "
      f"{kmf_control.median_survival_time_:.1f} months")
```

Caution

If `lifelines` is not installed, run `!pip install lifelines` in your notebook. It is not pre-installed in Google Colab.

Clinical context

Kaplan-Meier curves handle **censored** observations — patients who are still alive at the end of the study or who were lost to follow-up. Without proper censoring, you would underestimate survival. The vertical tick marks on the curve show censored observations.

5.9 Heatmaps

Heatmaps display matrices of values using color intensity. Two common uses in health:

5.9.1 Correlation matrices

```
# Correlation between health indicators
np.random.seed(42)
n = 300
health = pd.DataFrame({
    "BMI": np.random.normal(27, 5, n),
    "Systolic_BP": np.random.normal(130, 18, n),
    "Diastolic_BP": np.random.normal(82, 10, n),
    "Glucose": np.random.normal(105, 25, n),
    "Cholesterol": np.random.normal(200, 40, n),
    "HbA1c": np.random.normal(5.8, 1.2, n),
    "Heart_Rate": np.random.normal(72, 12, n),
    "Age": np.random.normal(55, 15, n).clip(18, 90)
})

# Add realistic correlations
health["Systolic_BP"] += 0.8 * health["BMI"]
health["Glucose"] += 15 * health["HbA1c"]
health["Diastolic_BP"] += 0.4 * health["Systolic_BP"]
health["Cholesterol"] += 0.5 * health["BMI"]

corr = health.corr()

fig, ax = plt.subplots(figsize=(9, 8))
mask = np.triu(np.ones_like(corr, dtype=bool))  # hide upper triangle
sns.heatmap(corr, mask=mask, annot=True, fmt=".2f",
            cmap="RdBu_r", center=0, vmin=-1, vmax=1,
            square=True, ax=ax)
ax.set_title("Correlation matrix of health indicators")
plt.tight_layout()
plt.show()
```

5.9.2 Disease co-occurrence

```
# Co-occurrence of chronic conditions
conditions = ["Hypertension", "Diabetes", "COPD",
              "Heart Failure", "CKD", "Obesity"]
np.random.seed(42)
cooccurrence = np.random.randint(5, 60, (6, 6))
cooccurrence = (cooccurrence + cooccurrence.T) // 2  # make symmetric
np.fill_diagonal(cooccurrence, [320, 180, 95, 75, 110, 210])

co_df = pd.DataFrame(cooccurrence,
                     index=conditions, columns=conditions)

fig, ax = plt.subplots(figsize=(9, 8))
sns.heatmap(co_df, annot=True, fmt="d", cmap="YlOrRd",
            square=True, ax=ax)
ax.set_title("Disease co-occurrence matrix\n"
             "(diagonal = total cases, off-diagonal = co-occurring)")
plt.tight_layout()
plt.show()
```

5.10 Practical: build a 4-panel health dashboard

Combining multiple plots into a single figure is essential for reports and publications. Here is a complete 4-panel dashboard using real data:

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

sns.set_theme(style="whitegrid", font_scale=1.0)

# Load data
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)
recent = df[df["year"] == 2007].copy()

# Create the 4-panel figure
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle("Global Health Dashboard, 2007",
             fontsize=16, fontweight="bold", y=1.02)

# ------------------------------------------------------------
# Panel A: Life expectancy distribution by continent
# ------------------------------------------------------------
ax = axes[0, 0]
sns.boxplot(data=recent, x="continent", y="lifeExp", ax=ax,
```

```
            palette="Set2")
ax.set_xlabel("")
ax.set_ylabel("Life expectancy (years)")
ax.set_title("A. Life expectancy by continent")
ax.tick_params(axis="x", rotation=30)

# ---------------------------------------------------------
# Panel B: GDP vs Life expectancy scatter
# ---------------------------------------------------------
ax = axes[0, 1]
for continent in recent["continent"].unique():
    subset = recent[recent["continent"] == continent]
    ax.scatter(subset["gdpPercap"], subset["lifeExp"],
               s=subset["pop"] / 2e6, alpha=0.6,
               label=continent)
ax.set_xscale("log")
ax.set_xlabel("GDP per capita (USD, log scale)")
ax.set_ylabel("Life expectancy (years)")
ax.set_title("B. Wealth vs health")
ax.legend(fontsize=8, title="Continent", title_fontsize=8)

# ---------------------------------------------------------
# Panel C: Life expectancy trends over time
# ---------------------------------------------------------
ax = axes[1, 0]
trend = df.groupby(["year", "continent"])["lifeExp"].mean().reset_index()
for continent in trend["continent"].unique():
    subset = trend[trend["continent"] == continent]
    ax.plot(subset["year"], subset["lifeExp"],
            marker="o", markersize=3, label=continent)
ax.set_xlabel("Year")
ax.set_ylabel("Mean life expectancy (years)")
ax.set_title("C. Life expectancy trends, 1952-2007")
ax.legend(fontsize=8, title="Continent", title_fontsize=8)

# ---------------------------------------------------------
# Panel D: Bottom 10 countries bar chart
# ---------------------------------------------------------
ax = axes[1, 1]
bottom10 = recent.nsmallest(10, "lifeExp")
ax.barh(bottom10["country"], bottom10["lifeExp"], color="#D55E00")
ax.set_xlabel("Life expectancy (years)")
ax.set_title("D. 10 lowest life expectancy countries")
ax.invert_yaxis()
for i, v in enumerate(bottom10["lifeExp"]):
    ax.text(v + 0.3, i, f"{v:.1f}", va="center", fontsize=8)

plt.tight_layout()
plt.savefig("health_dashboard_2007.png", dpi=150, bbox_inches="tight")
plt.show()
print("Dashboard saved as health_dashboard_2007.png")
```

Exercise

1. Modify the dashboard to show 2002 data instead of 2007. Does the story change?
2. Add a 5th panel showing a histogram of GDP per capita (log-transformed) with a vertical line at the world median. Use `fig, axes = plt.subplots(2, 3)` or `plt.subplot2grid()`.
3. Create a clinical dashboard for a simulated cohort of 500 patients with the following panels:
   - Histogram of BMI with overweight/obese thresholds
   - Box plot of systolic BP by sex
   - Scatter plot of age vs. HbA1c with a diabetic threshold line at 6.5%
   - Bar chart of diagnosis frequency
4. Using Our World in Data (https://github.com/owid/owid-datasets), download a dataset on malaria or tuberculosis. Build an epidemic dashboard with:
   - Line plot of incidence over time for 5 countries
   - Bar chart of the most recent year's incidence
   - Scatter plot of incidence vs. GDP
   - Heatmap of correlation between health indicators
5. Create a Kaplan-Meier plot comparing survival between 3 treatment groups (drug A, drug B, placebo) using simulated data. Add median survival annotations.
6. Build a "pre/post intervention" visualization: show a health indicator (e.g., malaria cases) as a line plot with a vertical line marking the intervention date. Shade the pre-intervention and post-intervention periods in different colors.

5.11 Chapter summary

- Always label axes with units, include sample sizes, and use colorblind-friendly palettes.
- **Histograms** show distributions; add clinical threshold lines to give context.
- **Box plots** compare groups; consider violin plots for more detail.
- **Bar charts** compare counts or rates; use horizontal bars for many categories.
- **Line plots** show trends over time; add 7-day rolling averages for epidemic curves.
- **Scatter plots** reveal relationships; add regression lines and report r values.

- **Kaplan-Meier curves** are the standard for survival data; use the `lifelines` library.
- **Heatmaps** display correlation matrices and co-occurrence patterns.
- Multi-panel figures (`plt.subplots()`) combine related plots into a coherent story.
- Save publication-quality figures with `plt.savefig("file.png", dpi=300, bbox_inches="tigh`

Chapter 6

Hypothesis testing

> *"The purpose of a statistical test is not to prove something is true. It is to measure how surprised we should be if nothing is going on."*

6.1 The logic of hypothesis testing

Every clinical question can be framed as a test between two competing statements:

- **Null hypothesis (H_0):** There is *no* effect, no difference, no association. The drug does not work. The two groups have the same blood pressure.
- **Alternative hypothesis (H_1):** There *is* an effect, a difference, or an association. The drug lowers cholesterol. The treatment group has lower blood pressure than the control.

We collect data, compute a test statistic, and ask: *if H_0 were true, how likely is it that we would see data this extreme?* That probability is the **p-value**.

Clinical context

Clinical framing matters. A statistician writes $H_0 : \mu_1 = \mu_2$. A clinician writes "there is no difference in systolic blood pressure between the statin group and the placebo group." Both say the same thing — use whichever makes the research question clear.

6.1.1 The steps of a hypothesis test

1. **State the hypotheses** (H_0 and H_1).
2. **Choose the significance level** α (usually 0.05).
3. **Collect data** and compute the **test statistic**.
4. **Compute the p-value** — the probability of observing a result at least as extreme as yours, assuming H_0 is true.
5. **Decide:** if $p < \alpha$, reject H_0. Otherwise, fail to reject H_0.

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
```

6.2 p-values: what they mean and what they don't

6.2.1 What a p-value IS

A p-value is the probability of observing data as extreme as (or more extreme than) what you collected, *assuming* H_0 is true. A $p = 0.03$ means: if the drug truly has no effect, there is a 3% chance of seeing results this extreme by random chance alone.

6.2.2 What a p-value is NOT

- It is **not** the probability that H_0 is true.
- It is **not** the probability that the result is due to chance.
- It does **not** measure the size of the effect.
- $p = 0.049$ is not meaningfully different from $p = 0.051$.

Caution

A small p-value does not mean a *large* or *clinically important* effect. A study with 100 000 patients can find $p < 0.001$ for a blood pressure difference of 0.5 mmHg — statistically significant but clinically meaningless. Always report **effect size** alongside p-values.

6.2.3 Type I and Type II errors

- **Type I error (false positive):** You reject H_0 when it is actually true. You conclude the drug works when it does not. Controlled by α.
- **Type II error (false negative):** You fail to reject H_0 when H_1 is actually true. You miss a real effect. Related to **power** $(1 - \beta)$.

```
# Visualize the logic: sampling distribution under H0
np.random.seed(42)
null_distribution = np.random.normal(loc=0, scale=1, size=10000)

fig, ax = plt.subplots(figsize=(9, 4))
ax.hist(null_distribution, bins=60, density=True, alpha=0.7, color="steelblue")
ax.axvline(1.96, color="red", linestyle="--", label="Critical value
↪  (alpha=0.05)")
ax.axvline(-1.96, color="red", linestyle="--")
```

```
ax.fill_betweenx([0, 0.45], 1.96, 3.5, alpha=0.3, color="red", label="Rejection
↪  region")
ax.fill_betweenx([0, 0.45], -3.5, -1.96, alpha=0.3, color="red")
ax.set_xlabel("Test statistic (z)")
ax.set_ylabel("Density")
ax.set_title("Two-tailed test: rejection regions under H0")
ax.legend()
plt.tight_layout()
plt.show()
```

6.3 One-sample t-test

Question: Is the mean total cholesterol in our clinic patients different from the national average of 200 mg/dL?

```
# Simulated clinic cholesterol data (mg/dL)
np.random.seed(42)
cholesterol = np.random.normal(loc=212, scale=35, size=80)

# One-sample t-test: is the mean different from 200?
t_stat, p_value = stats.ttest_1samp(cholesterol, popmean=200)
print(f"Sample mean: {cholesterol.mean():.1f} mg/dL")
print(f"t-statistic: {t_stat:.3f}")
print(f"p-value: {p_value:.4f}")

if p_value < 0.05:
    print("Reject H0: mean cholesterol differs from 200 mg/dL")
else:
    print("Fail to reject H0: no evidence mean differs from 200 mg/dL")
```

Clinical context

The one-sample t-test compares a sample mean to a known reference value. In clinical practice, the reference might be a national average, a guideline threshold, or a pre-treatment baseline.

6.4 Two-sample t-test

Question: Is systolic blood pressure different between the treatment group and the control group?

```
# Simulated clinical trial data
np.random.seed(42)
treatment = np.random.normal(loc=128, scale=15, size=60)
control = np.random.normal(loc=138, scale=16, size=55)

# Independent two-sample t-test
```

```
t_stat, p_value = stats.ttest_ind(treatment, control)
print(f"Treatment mean: {treatment.mean():.1f} mmHg")
print(f"Control mean:   {control.mean():.1f} mmHg")
print(f"Difference:     {control.mean() - treatment.mean():.1f} mmHg")
print(f"t-statistic:    {t_stat:.3f}")
print(f"p-value:        {p_value:.4f}")
```

6.4.1 Checking assumptions

The t-test assumes approximate normality and equal variances. Check both:

```
# Normality: Shapiro-Wilk test (H0: data is normally distributed)
_, p_treat = stats.shapiro(treatment)
_, p_ctrl = stats.shapiro(control)
print(f"Shapiro-Wilk p (treatment): {p_treat:.4f}")
print(f"Shapiro-Wilk p (control):   {p_ctrl:.4f}")

# Equal variances: Levene's test
_, p_levene = stats.levene(treatment, control)
print(f"Levene's test p: {p_levene:.4f}")

# If variances are unequal, use Welch's t-test:
t_stat, p_value = stats.ttest_ind(treatment, control, equal_var=False)
print(f"Welch's t-test p-value: {p_value:.4f}")
```

Python tip

Set `equal_var=False` in `ttest_ind()` to use Welch's t-test. It does not assume equal variances and is often the safer default.

6.5 Paired t-test

Question: Does a 12-week exercise intervention reduce systolic blood pressure?

Each patient is measured *before* and *after*. The paired t-test uses the within-patient differences.

```
# Simulated before/after intervention data
np.random.seed(42)
n_patients = 45
bp_before = np.random.normal(loc=142, scale=12, size=n_patients)
# After intervention: mean reduction of ~8 mmHg with individual variation
bp_after = bp_before - np.random.normal(loc=8, scale=6, size=n_patients)

# Paired t-test
t_stat, p_value = stats.ttest_rel(bp_before, bp_after)
differences = bp_before - bp_after
print(f"Mean BP before:     {bp_before.mean():.1f} mmHg")
```

```
print(f"Mean BP after:       {bp_after.mean():.1f} mmHg")
print(f"Mean reduction:      {differences.mean():.1f} mmHg")
print(f"t-statistic:         {t_stat:.3f}")
print(f"p-value:             {p_value:.6f}")
```

```
# Visualize before/after
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Paired lines
for i in range(n_patients):
    axes[0].plot([0, 1], [bp_before[i], bp_after[i]],
                 color="gray", alpha=0.4)
axes[0].plot([0, 1], [bp_before.mean(), bp_after.mean()],
             color="red", linewidth=3, marker="o", markersize=8)
axes[0].set_xticks([0, 1])
axes[0].set_xticklabels(["Before", "After"])
axes[0].set_ylabel("Systolic BP (mmHg)")
axes[0].set_title("Individual patient trajectories")

# Distribution of differences
axes[1].hist(differences, bins=15, edgecolor="black", alpha=0.7,
↪  color="steelblue")
axes[1].axvline(0, color="red", linestyle="--", label="No change")
axes[1].axvline(differences.mean(), color="green", linestyle="--",
                label=f"Mean = {differences.mean():.1f}")
axes[1].set_xlabel("BP reduction (mmHg)")
axes[1].set_ylabel("Frequency")
axes[1].set_title("Distribution of BP changes")
axes[1].legend()

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

Caution

A common mistake is using an *independent* two-sample t-test on before/after data. This ignores the pairing and loses statistical power. If the same patients are measured twice, always use a **paired** t-test.

6.6 Chi-square test of independence

Question: Is diabetes prevalence associated with obesity category?

The chi-square test works with *categorical* variables. It compares observed counts to what we would expect if the variables were independent.

```
# Simulated cross-tabulation data
np.random.seed(42)
```

```
n = 500
bmi_cat = np.random.choice(["Normal", "Overweight", "Obese"],
                           size=n, p=[0.3, 0.4, 0.3])
# Diabetes probability increases with obesity
diabetes_prob = {"Normal": 0.08, "Overweight": 0.18, "Obese": 0.35}
diabetes = [np.random.binomial(1, diabetes_prob[b]) for b in bmi_cat]

df_chi = pd.DataFrame({"bmi_category": bmi_cat, "diabetes": diabetes})
df_chi["diabetes_label"] = df_chi["diabetes"].map({0: "No", 1: "Yes"})

# Contingency table
contingency = pd.crosstab(df_chi["bmi_category"], df_chi["diabetes_label"])
print(contingency)
print()

# Chi-square test
chi2, p_value, dof, expected = stats.chi2_contingency(contingency)
print(f"Chi-square statistic: {chi2:.2f}")
print(f"Degrees of freedom:   {dof}")
print(f"p-value:              {p_value:.4f}")
print(f"\nExpected frequencies (if independent):")
print(pd.DataFrame(expected, index=contingency.index,
                   columns=contingency.columns).round(1))
```

Clinical context

The chi-square test requires that expected cell counts are at least 5. If you have small cells (e.g., a rare disease), use Fisher's exact test instead: `stats.fisher_exact(table)` for 2×2 tables.

6.7 Mann-Whitney U test

When your data is not normally distributed — common with hospital length-of-stay, cost data, or small samples — use the Mann-Whitney U test. It compares the *ranks* of observations rather than the means.

```
# Hospital length of stay: surgical vs non-surgical (right-skewed data)
np.random.seed(42)
los_surgical = np.random.exponential(scale=7, size=40)
los_medical = np.random.exponential(scale=4.5, size=45)

# Mann-Whitney U test
u_stat, p_value = stats.mannwhitneyu(los_surgical, los_medical,
                                     alternative="two-sided")
print(f"Median LOS (surgical): {np.median(los_surgical):.1f} days")
print(f"Median LOS (medical):  {np.median(los_medical):.1f} days")
print(f"U statistic:           {u_stat:.0f}")
print(f"p-value:               {p_value:.4f}")
```

```python
# Visualize the skewed distributions
fig, ax = plt.subplots(figsize=(8, 4))
ax.hist(los_surgical, bins=15, alpha=0.6, label="Surgical", color="coral")
ax.hist(los_medical, bins=15, alpha=0.6, label="Medical", color="steelblue")
ax.set_xlabel("Length of stay (days)")
ax.set_ylabel("Frequency")
ax.set_title("Hospital length of stay by department")
ax.legend()
plt.tight_layout()
plt.show()
```

Python tip

Rule of thumb for choosing a test:

- Normal data, two groups → t-test
- Non-normal data, two groups → Mann-Whitney U
- Categorical data → chi-square
- Paired measurements → paired t-test (or Wilcoxon signed-rank if non-normal)

6.8 Multiple testing correction

6.8.1 The problem

If you test 20 hypotheses at $\alpha = 0.05$, you expect $20 \times 0.05 = 1$ false positive *even if nothing is going on*. In genomics, you might test 20 000 genes simultaneously. Without correction, you would get 1 000 spurious "discoveries."

```python
# Simulation: 1000 tests where H0 is true for all
np.random.seed(42)
n_tests = 1000
p_values = []
for _ in range(n_tests):
    # Two random samples from the SAME distribution (H0 is true)
    a = np.random.normal(0, 1, 30)
    b = np.random.normal(0, 1, 30)
    _, p = stats.ttest_ind(a, b)
    p_values.append(p)

p_values = np.array(p_values)
false_positives = (p_values < 0.05).sum()
print(f"Tests performed: {n_tests}")
print(f"False positives (p < 0.05): {false_positives}")
print(f"False positive rate: {false_positives/n_tests:.1%}")
```

6.8.2 Bonferroni correction

The simplest fix: divide α by the number of tests.

```
from statsmodels.stats.multitest import multipletests

# Bonferroni correction
rejected_bonf, pvals_corrected_bonf, _, _ = multipletests(
    p_values, alpha=0.05, method="bonferroni"
)
print(f"Bonferroni: {rejected_bonf.sum()} significant results")
```

6.8.3 Benjamini-Hochberg (FDR)

Less conservative: controls the *false discovery rate* (the proportion of rejected hypotheses that are false positives).

```
# Benjamini-Hochberg FDR correction
rejected_fdr, pvals_corrected_fdr, _, _ = multipletests(
    p_values, alpha=0.05, method="fdr_bh"
)
print(f"FDR (BH): {rejected_fdr.sum()} significant results")
```

Clinical context

In genomics and proteomics, FDR correction is standard. When you read a paper reporting "542 differentially expressed genes at FDR < 0.05," the authors used Benjamini-Hochberg (or a similar method) to control for the thousands of simultaneous tests.

Caution

Bonferroni is very conservative — it minimizes false positives but can miss real effects (high Type II error). FDR is a good middle ground for exploratory analyses. For confirmatory clinical trials, Bonferroni is often preferred because false positives are more costly.

6.9 Effect size: clinical vs statistical significance

6.9.1 Cohen's d

Cohen's d measures the size of the difference in *standard deviation units*:

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{pooled}}}$$

- $|d| < 0.2$: negligible effect

- $0.2 \leq |d| < 0.5$: small effect
- $0.5 \leq |d| < 0.8$: medium effect
- $|d| \geq 0.8$: large effect

```
def cohens_d(group1, group2):
    """Compute Cohen's d for two independent groups."""
    n1, n2 = len(group1), len(group2)
    var1, var2 = group1.var(ddof=1), group2.var(ddof=1)
    pooled_std = np.sqrt(((n1 - 1) * var1 + (n2 - 1) * var2) / (n1 + n2 - 2))
    return (group1.mean() - group2.mean()) / pooled_std

# Using our treatment vs control BP data
d = cohens_d(treatment, control)
print(f"Cohen's d: {d:.2f}")
print(f"Interpretation: {'large' if abs(d) >= 0.8 else 'medium' if abs(d) >=
↪  0.5 else 'small' if abs(d) >= 0.2 else 'negligible'} effect")
```

6.9.2 Why effect size matters

```
# Large sample + tiny effect = significant p-value
np.random.seed(42)
huge_group1 = np.random.normal(120.0, 15, size=50000)
huge_group2 = np.random.normal(120.5, 15, size=50000)  # only 0.5 mmHg
↪  difference

t, p = stats.ttest_ind(huge_group1, huge_group2)
d = cohens_d(huge_group1, huge_group2)
print(f"Mean difference: {huge_group2.mean() - huge_group1.mean():.2f} mmHg")
print(f"p-value: {p:.6f}")
print(f"Cohen's d: {d:.3f}")
print("Statistically significant? Yes. Clinically meaningful? No.")
```

Clinical context

In a clinical trial report, the FDA and EMA expect both p-values *and* confidence intervals for the treatment effect. A drug that lowers HbA1c by 0.01% with $p < 0.001$ will not get approved. The effect must be **clinically meaningful**.

6.10 Practical: hypothesis testing with Framingham-style data

```
# Load a Framingham-style cardiovascular dataset
url = ("https://raw.githubusercontent.com/dsrscientist/"
       "dataset-for-machine-learning/master/framingham.csv")
```

```
fram = pd.read_csv(url)
print(f"Shape: {fram.shape}")
print(f"Columns: {fram.columns.tolist()}")
fram.head()
```

```
# Quick exploration
fram.describe()
```

```
# Drop rows with missing values for simplicity
fram_clean = fram.dropna()
print(f"Clean dataset: {fram_clean.shape[0]} patients")
```

Exercise

Using the Framingham dataset:

1. **One-sample t-test.** Is the mean total cholesterol in this cohort different from the US national average of 200 mg/dL?
2. **Two-sample t-test.** Compare systolic blood pressure between patients who developed CHD (`TenYearCHD = 1`) and those who did not. Report the p-value, mean difference, and Cohen's d.
3. **Chi-square test.** Create a 2×2 contingency table of `currentSmoker` vs `TenYearCHD`. Is smoking associated with 10-year CHD risk?
4. **Mann-Whitney U.** Compare `cigsPerDay` between CHD and non-CHD groups. Why is Mann-Whitney more appropriate than a t-test here?
5. **Multiple testing.** Run t-tests comparing CHD vs non-CHD for all continuous variables (age, totChol, sysBP, diaBP, BMI, heartRate, glucose). Apply Bonferroni and FDR correction. Which variables remain significant after each correction?
6. **Clinical interpretation.** For the variable with the largest effect size (Cohen's d), discuss whether the difference is clinically meaningful.

6.11 Chapter summary

- A hypothesis test measures how surprising the data would be if H_0 were true.
- The p-value is *not* the probability that H_0 is true.
- One-sample t-test: compare a sample mean to a reference value.
- Two-sample t-test: compare means between two independent groups.
- Paired t-test: compare means from the same subjects measured twice.

- Chi-square test: test association between two categorical variables.
- Mann-Whitney U: non-parametric alternative when data is not normal.
- Multiple testing correction (Bonferroni, FDR) prevents false discoveries.
- Effect size (Cohen's d) distinguishes statistical from clinical significance.
- Always report both p-values and effect sizes in clinical research.

Chapter 7

Regression for health outcomes

> *"Correlation tells you that two things move together. Regression tells you by how much — and lets you adjust for confounders."*

7.1 From correlation to prediction

In Chapter 6 we asked "is there a difference?" Now we ask "can we predict one variable from another?" Regression is the workhorse of clinical research: it appears in nearly every epidemiological study, every clinical trial analysis, and every risk prediction model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

7.2 Simple linear regression

Question: How does systolic blood pressure change with age?

```
# Simulated clinical data
np.random.seed(42)
n = 200
age = np.random.uniform(25, 80, n)
# True relationship: SBP = 90 + 0.7 * age + noise
sbp = 90 + 0.7 * age + np.random.normal(0, 12, n)

df = pd.DataFrame({"age": age, "systolic_bp": sbp})

# Fit with scipy
slope, intercept, r_value, p_value, std_err = stats.linregress(df["age"],
↪  df["systolic_bp"])
print(f"Intercept: {intercept:.2f}")
print(f"Slope: {slope:.2f} mmHg per year of age")
print(f"R-squared: {r_value**2:.3f}")
```

```
print(f"p-value: {p_value:.2e}")
```

```
# Visualize
fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(df["age"], df["systolic_bp"], alpha=0.5, s=20, color="steelblue")
x_line = np.linspace(25, 80, 100)
ax.plot(x_line, intercept + slope * x_line, color="red", linewidth=2,
        label=f"SBP = {intercept:.1f} + {slope:.2f} x Age")
ax.set_xlabel("Age (years)")
ax.set_ylabel("Systolic BP (mmHg)")
ax.set_title("Simple linear regression: SBP vs Age")
ax.legend()
plt.tight_layout()
plt.show()
```

Clinical context

The slope of 0.7 means that, on average, systolic BP increases by 0.7 mmHg for each additional year of age. This does *not* mean aging *causes* higher BP — it means the two are linearly associated in this sample.

7.3 Multiple linear regression

Question: Predict systolic BP from age, BMI, and smoking status simultaneously.

```
import statsmodels.api as sm

# Expanded simulated data
np.random.seed(42)
n = 300
age = np.random.uniform(25, 80, n)
bmi = np.random.normal(27, 5, n)
smoking = np.random.binomial(1, 0.25, n)  # 25% smokers
# True model: SBP = 70 + 0.6*age + 0.8*BMI + 5*smoking + noise
sbp = 70 + 0.6 * age + 0.8 * bmi + 5 * smoking + np.random.normal(0, 10, n)

df = pd.DataFrame({
    "age": age, "bmi": bmi, "smoking": smoking, "systolic_bp": sbp
})

# Fit multiple regression with statsmodels
X = sm.add_constant(df[["age", "bmi", "smoking"]])
model = sm.OLS(df["systolic_bp"], X).fit()
print(model.summary())
```

7.3.1 Interpreting coefficients in clinical context

```
# Extract and display coefficients
coef_df = pd.DataFrame({
    "Coefficient": model.params,
    "95% CI lower": model.conf_int()[0],
    "95% CI upper": model.conf_int()[1],
    "p-value": model.pvalues
}).round(4)
print(coef_df)
```

Each coefficient answers: "holding all other variables constant, what is the expected change in SBP for a one-unit increase in this predictor?"

- **Age coefficient** ≈ 0.6: each additional year of age is associated with 0.6 mmHg higher SBP, *after adjusting for BMI and smoking.*
- **BMI coefficient** ≈ 0.8: each additional BMI unit is associated with 0.8 mmHg higher SBP, *after adjusting for age and smoking.*
- **Smoking coefficient** ≈ 5: smokers have, on average, 5 mmHg higher SBP than non-smokers, *after adjusting for age and BMI.*

Python tip

The phrase "after adjusting for" (or "controlling for") is the key difference between simple and multiple regression. It is why epidemiologists use multiple regression — to separate the effect of one variable from potential confounders.

7.4 Confounders

A confounder is a variable that is associated with both the exposure and the outcome, distorting the apparent relationship between them.

```
# Example: coffee drinking and heart disease
# Both are associated with smoking (the confounder)
np.random.seed(42)
n = 500
smoking = np.random.binomial(1, 0.3, n)
coffee = 1.5 + 2 * smoking + np.random.normal(0, 1.5, n)  # smokers drink more
↪  coffee
heart_disease_risk = 0.5 * smoking + np.random.normal(0, 0.3, n)  # smoking
↪  causes HD

# Unadjusted: coffee appears to predict heart disease
r_unadjusted, p_unadjusted = stats.pearsonr(coffee, heart_disease_risk)
print(f"Unadjusted correlation (coffee vs HD risk): r = {r_unadjusted:.3f}, p =
↪  {p_unadjusted:.4f}")
```

```
# Adjusted regression: coffee effect disappears
df_conf = pd.DataFrame({"coffee": coffee, "smoking": smoking, "hd_risk":
↪  heart_disease_risk})
X = sm.add_constant(df_conf[["coffee", "smoking"]])
model_adj = sm.OLS(df_conf["hd_risk"], X).fit()
print(f"\nAdjusted model:")
print(f"  Coffee coefficient: {model_adj.params['coffee']:.4f} (p =
↪  {model_adj.pvalues['coffee']:.4f})")
print(f"  Smoking coefficient: {model_adj.params['smoking']:.4f} (p =
↪  {model_adj.pvalues['smoking']:.4f})")
```

Caution

Without adjusting for smoking, you would wrongly conclude that coffee increases heart disease risk. This is one of the most common errors in observational studies. Always think about what confounders might be lurking in your data.

7.5 Checking regression assumptions

```
# Residual diagnostics for our SBP model
residuals = model.resid
fitted = model.fittedvalues

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Residuals vs fitted
axes[0].scatter(fitted, residuals, alpha=0.4, s=15)
axes[0].axhline(0, color="red", linestyle="--")
axes[0].set_xlabel("Fitted values")
axes[0].set_ylabel("Residuals")
axes[0].set_title("Residuals vs Fitted")

# Histogram of residuals
axes[1].hist(residuals, bins=25, edgecolor="black", alpha=0.7)
axes[1].set_xlabel("Residuals")
axes[1].set_title("Distribution of residuals")

# Q-Q plot
stats.probplot(residuals, dist="norm", plot=axes[2])
axes[2].set_title("Q-Q plot")

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

Clinical context

The four key assumptions of linear regression: (1) linearity, (2) independence of residuals, (3) homoscedasticity (constant variance), (4) normality of residuals. The plots above help you check assumptions 1, 3, and 4. Violation does not always invalidate results, but it should make you cautious.

7.6 Logistic regression

When the outcome is binary (disease: yes/no), linear regression is inappropriate. Logistic regression models the *probability* of the outcome.

Question: Predict whether a patient has diabetes (yes/no) from age, BMI, and glucose level.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Load the Pima Indians Diabetes dataset
url = ("https://raw.githubusercontent.com/jbrownlee/Datasets/"
       "master/pima-indians-diabetes.data.csv")
columns = ["pregnancies", "glucose", "blood_pressure", "skin_thickness",
           "insulin", "bmi", "diabetes_pedigree", "age", "outcome"]
pima = pd.read_csv(url, names=columns)
print(f"Shape: {pima.shape}")
print(f"Diabetes prevalence: {pima['outcome'].mean():.1%}")
pima.head()
```

```
# Fit logistic regression with statsmodels (for detailed output)
X = sm.add_constant(pima[["age", "bmi", "glucose"]])
y = pima["outcome"]

logit_model = sm.Logit(y, X).fit()
print(logit_model.summary())
```

7.7 Odds ratios from logistic regression

The coefficients of logistic regression are log-odds. Exponentiate them to get **odds ratios**:

```
# Odds ratios with 95% confidence intervals
odds_ratios = np.exp(logit_model.params)
ci = np.exp(logit_model.conf_int())

or_df = pd.DataFrame({
    "Odds Ratio": odds_ratios,
```

```
    "95% CI lower": ci[0],
    "95% CI upper": ci[1],
    "p-value": logit_model.pvalues
}).round(4)
print(or_df)
```

Clinical context

An odds ratio of 1.04 for glucose means: for each additional 1 mg/dL of glucose, the odds of diabetes increase by 4%, holding age and BMI constant. An OR > 1 means higher risk; OR < 1 means lower risk; OR $= 1$ means no association.

7.8 Model evaluation: confusion matrix and beyond

```
# Fit with scikit-learn for prediction
features = ["age", "bmi", "glucose"]
X = pima[features]
y = pima["outcome"]

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

clf = LogisticRegression(max_iter=1000, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print()

# Detailed metrics
print(classification_report(y_test, y_pred,
                            target_names=["No Diabetes", "Diabetes"]))
```

7.8.1 Sensitivity, specificity, PPV, NPV

```
tn, fp, fn, tp = cm.ravel()
sensitivity = tp / (tp + fn)     # true positive rate (recall)
specificity = tn / (tn + fp)     # true negative rate
ppv = tp / (tp + fp)             # positive predictive value (precision)
npv = tn / (tn + fn)             # negative predictive value

print(f"Sensitivity (recall): {sensitivity:.3f}")
```

```
print(f"  -> Of those WITH diabetes, {sensitivity:.1%} were correctly
↪  identified")
print(f"Specificity:          {specificity:.3f}")
print(f"  -> Of those WITHOUT diabetes, {specificity:.1%} were correctly
↪  identified")
print(f"PPV (precision):      {ppv:.3f}")
print(f"  -> Of those PREDICTED positive, {ppv:.1%} actually have diabetes")
print(f"NPV:                  {npv:.3f}")
print(f"  -> Of those PREDICTED negative, {npv:.1%} are truly disease-free")
```

```
# Visualize the confusion matrix
fig, ax = plt.subplots(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["No Diabetes", "Diabetes"],
            yticklabels=["No Diabetes", "Diabetes"], ax=ax)
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
ax.set_title("Confusion Matrix")
plt.tight_layout()
plt.show()
```

Caution

In clinical screening, missing a disease (false negative) is often more dangerous than a false alarm (false positive). A screening test for cancer should have **high sensitivity** even if specificity is moderate. A confirmatory test should have **high specificity**.

7.9 Introduction to survival analysis

Sometimes the outcome is not "yes/no" but "time until event" — time to death, time to relapse, time to hospital readmission. This is **survival analysis**.

7.9.1 Why not just use logistic regression?

Two reasons:

1. **Censoring.** Some patients leave the study before the event occurs. We know they survived *at least* that long, but not their true event time.
2. **Time matters.** A drug that delays relapse by 3 years is better than one that delays it by 3 months, even if both eventually fail.

7.9.2 Kaplan-Meier curves

```
# Install lifelines if needed: pip install lifelines
from lifelines import KaplanMeierFitter, CoxPHFitter
```

```
# Simulated survival data
np.random.seed(42)
n = 200
treatment = np.random.binomial(1, 0.5, n)
# Treatment group survives longer
time = np.where(treatment == 1,
                np.random.exponential(24, n),    # treatment: mean 24 months
                np.random.exponential(16, n))    # control: mean 16 months
event = np.random.binomial(1, 0.75, n)  # 25% censored

surv_df = pd.DataFrame({
    "time": time, "event": event, "treatment": treatment
})
```

```
# Kaplan-Meier curves
fig, ax = plt.subplots(figsize=(8, 5))

for label, group_df in surv_df.groupby("treatment"):
    kmf = KaplanMeierFitter()
    kmf.fit(group_df["time"], event_observed=group_df["event"],
            label="Treatment" if label == 1 else "Control")
    kmf.plot_survival_function(ax=ax)

ax.set_xlabel("Time (months)")
ax.set_ylabel("Survival probability")
ax.set_title("Kaplan-Meier survival curves")
plt.tight_layout()
plt.show()
```

7.9.3 Cox proportional hazards model

The Cox model is the "regression" of survival analysis. It estimates how covariates affect the hazard (risk of event at each time point):

```
# Cox proportional hazards
cph = CoxPHFitter()
cph.fit(surv_df, duration_col="time", event_col="event",
        formula="treatment")
cph.print_summary()
```

```
# Hazard ratio
hr = np.exp(cph.params_["treatment"])
print(f"Hazard ratio for treatment: {hr:.3f}")
if hr < 1:
    print(f"Treatment reduces hazard by {(1-hr)*100:.1f}%")
else:
    print(f"Treatment increases hazard by {(hr-1)*100:.1f}%")
```

Clinical context

A hazard ratio of 0.65 means the treatment group has 35% lower risk of the event at any given time point, compared to the control group. Hazard ratios are the standard way to report results in oncology trials, cardiovascular outcome studies, and any study with a time-to-event endpoint.

7.10 Practical: diabetes risk prediction

Exercise

Using the Pima Indians Diabetes dataset (`https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv`):

1. **Data preparation.** Load the dataset. Replace zero values in `glucose`, `blood_pressure`, `bmi`, `skin_thickness`, and `insulin` with `NaN` (these are biologically impossible zeros). Drop rows with missing values.

2. **Simple logistic regression.** Fit a model predicting diabetes from glucose alone. What is the odds ratio? Interpret it.

3. **Multiple logistic regression.** Fit a model using age, BMI, glucose, blood pressure, and diabetes pedigree function. Which variables are statistically significant? Report odds ratios with 95% CIs.

4. **Model comparison.** Split the data 70/30. Fit both models on the training set. Compare sensitivity, specificity, and accuracy on the test set. Which model is better for *screening*?

5. **Confounding.** Fit a model with glucose only, then add age. Does the glucose coefficient change? What does this tell you about confounding?

6. **Clinical thresholds.** Instead of the default 0.5 threshold, try 0.3 and 0.7. How do sensitivity and specificity change? When might you prefer a lower threshold?

7. **Bonus: survival analysis.** Using the simulated survival data above, add "age" and "smoking" as covariates to the Cox model. Which has the largest hazard ratio?

7.11 Chapter summary

- Simple linear regression models the relationship between one predictor and a continuous outcome.
- Multiple regression adjusts for confounders — the phrase "after controlling for" reflects this.
- Coefficients represent the expected change in the outcome for a one-unit change in the predictor, holding other variables constant.

- Logistic regression predicts binary outcomes (disease yes/no) and produces odds ratios.
- Model evaluation requires the confusion matrix: sensitivity, specificity, PPV, and NPV.
- High sensitivity is critical for screening; high specificity is critical for confirmation.
- Survival analysis (Kaplan-Meier, Cox model) handles time-to-event data with censoring.
- Always check regression assumptions and report confidence intervals, not just p-values.

Chapter 8

Machine learning for clinical prediction

"A model that works perfectly on your training data and fails on the next patient is not a model — it is a memory."

8.1 When to use ML vs traditional statistics

Machine learning and traditional statistics answer different questions:

- **Traditional statistics** (regression, t-tests): "Which risk factors are associated with heart disease, and by how much?" The goal is *inference* — understanding relationships.
- **Machine learning**: "Given this patient's data, what is their probability of heart disease?" The goal is *prediction* — accuracy on new patients.

Clinical context

Use logistic regression when you need to explain *why* (odds ratios, confidence intervals, p-values for each risk factor). Use ML when you need the best possible *prediction* and interpretability is secondary. In practice, many clinical studies use both: regression to understand the biology, ML to build the screening tool.

8.1.1 When NOT to use ML

- Small datasets (< 200 patients): ML overfits easily. Stick with logistic regression.
- When you need regulatory approval: the FDA requires interpretable models for most clinical decision support tools.
- When a simple rule works: if "glucose > 126" classifies 90% of diabetics correctly, you do not need a random forest.

```
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, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_curve, roc_auc_score, RocCurveDisplay)
```

8.2 Loading clinical data

We use the UCI Heart Disease dataset — a classic benchmark in clinical ML.

```
# UCI Heart Disease dataset
url = ("https://raw.githubusercontent.com/raphaelfontenelle/"
       "heart-disease-uci/main/heart.csv")
heart = pd.read_csv(url)
print(f"Shape: {heart.shape}")
print(f"Heart disease prevalence: {heart['target'].mean():.1%}")
heart.head()
```

```
# Column descriptions
column_info = {
    "age": "Age in years",
    "sex": "1 = male, 0 = female",
    "cp": "Chest pain type (0-3)",
    "trestbps": "Resting blood pressure (mmHg)",
    "chol": "Serum cholesterol (mg/dL)",
    "fbs": "Fasting blood sugar > 120 mg/dL (1 = true)",
    "restecg": "Resting ECG results (0-2)",
    "thalach": "Maximum heart rate achieved",
    "exang": "Exercise-induced angina (1 = yes)",
    "oldpeak": "ST depression induced by exercise",
    "slope": "Slope of peak exercise ST segment",
    "ca": "Number of major vessels colored by fluoroscopy (0-3)",
    "thal": "Thalassemia (1 = normal, 2 = fixed defect, 3 = reversible)",
    "target": "Heart disease (1 = yes, 0 = no)"
}
for col, desc in column_info.items():
    print(f"  {col:12s} : {desc}")
```

8.3 Train/test split and cross-validation

8.3.1 Why split the data?

If you train a model on all your data and then evaluate it on the same data, the accuracy is misleadingly high. The model has "seen the answers." To estimate real-world performance,

you must evaluate on data the model has *never seen*.

```
# Define features and target
feature_cols = ["age", "sex", "cp", "trestbps", "chol", "fbs",
                "restecg", "thalach", "exang", "oldpeak", "slope", "ca",
                ↪  "thal"]
X = heart[feature_cols]
y = heart["target"]

# 70/30 train/test split (stratified to preserve class balance)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
print(f"Training set: {X_train.shape[0]} patients")
print(f"Test set:     {X_test.shape[0]} patients")
print(f"Training prevalence: {y_train.mean():.1%}")
print(f"Test prevalence:     {y_test.mean():.1%}")
```

8.3.2 Cross-validation

A single train/test split can be lucky or unlucky. Cross-validation (CV) gives a more robust estimate by rotating the test set:

```
# 5-fold cross-validation with logistic regression
lr = LogisticRegression(max_iter=1000, random_state=42)
cv_scores = cross_val_score(lr, X, y, cv=5, scoring="accuracy")
print(f"CV accuracy: {cv_scores.mean():.3f} +/- {cv_scores.std():.3f}")
print(f"Per-fold: {cv_scores.round(3)}")
```

Caution

Never use test set results to choose your model or tune hyperparameters. The test set must only be used **once**, at the very end. If you peek at test results during model selection, you are effectively training on the test set.

8.4 Decision trees

A decision tree splits patients into groups using yes/no questions, forming a flowchart that clinicians can follow.

```
# Fit a decision tree
dt = DecisionTreeClassifier(max_depth=4, random_state=42)
dt.fit(X_train, y_train)

# Evaluate
y_pred_dt = dt.predict(X_test)
```

```
print("Decision Tree Results:")
print(classification_report(y_test, y_pred_dt,
                            target_names=["No Disease", "Disease"]))
```

```
# Visualize the tree
fig, ax = plt.subplots(figsize=(20, 10))
plot_tree(dt, feature_names=feature_cols,
          class_names=["No Disease", "Disease"],
          filled=True, rounded=True, fontsize=9, ax=ax)
plt.title("Decision tree for heart disease prediction")
plt.tight_layout()
plt.show()
```

Clinical context

Decision trees are popular in emergency medicine because they produce *clinical decision rules* — simple flowcharts that a physician can follow at the bedside without a computer. The HEART score, Wells criteria, and Ottawa ankle rules are all conceptually similar to decision trees.

8.4.1 The danger of deep trees

```
# Overfit tree (no depth limit)
dt_overfit = DecisionTreeClassifier(random_state=42)  # no max_depth
dt_overfit.fit(X_train, y_train)

print(f"Training accuracy (overfit): {dt_overfit.score(X_train, y_train):.3f}")
print(f"Test accuracy (overfit):     {dt_overfit.score(X_test, y_test):.3f}")
print(f"Training accuracy (depth=4): {dt.score(X_train, y_train):.3f}")
print(f"Test accuracy (depth=4):     {dt.score(X_test, y_test):.3f}")
```

Python tip

When training accuracy is much higher than test accuracy, your model is **overfitting** — it has memorized the training data instead of learning general patterns. Limit tree depth with `max_depth` or require a minimum number of samples per leaf with `min_samples_leaf`.

8.5 Random forests

A random forest combines hundreds of decision trees, each trained on a random subset of the data and features. The final prediction is the majority vote.

```
# Fit a random forest
rf = RandomForestClassifier(n_estimators=200, max_depth=6,
```

```
                              random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
print("Random Forest Results:")
print(classification_report(y_test, y_pred_rf,
                              target_names=["No Disease", "Disease"]))
```

```
# Cross-validation comparison
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree (depth=4)": DecisionTreeClassifier(max_depth=4,
    ↪  random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=6,
                                            random_state=42, n_jobs=-1)
}

print("5-fold CV accuracy:")
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
    print(f"  {name:30s}: {scores.mean():.3f} +/- {scores.std():.3f}")
```

8.6 ROC curves and AUC

The ROC (Receiver Operating Characteristic) curve plots sensitivity vs (1 – specificity) across all possible classification thresholds. The AUC (Area Under the Curve) summarizes discriminative ability in a single number.

- AUC = 0.5: no better than flipping a coin.
- AUC = 0.7–0.8: acceptable discrimination.
- AUC = 0.8–0.9: excellent discrimination.
- AUC > 0.9: outstanding (rare in clinical practice).

```
# Get predicted probabilities
lr.fit(X_train, y_train)
y_prob_lr = lr.predict_proba(X_test)[:, 1]
y_prob_rf = rf.predict_proba(X_test)[:, 1]

# Compute ROC curves
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
auc_lr = roc_auc_score(y_test, y_prob_lr)
auc_rf = roc_auc_score(y_test, y_prob_rf)

# Plot
fig, ax = plt.subplots(figsize=(7, 6))
```

```
ax.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (AUC = {auc_lr:.3f})",
        linewidth=2)
ax.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {auc_rf:.3f})",
        linewidth=2)
ax.plot([0, 1], [0, 1], "k--", alpha=0.5, label="Random (AUC = 0.500)")
ax.set_xlabel("False Positive Rate (1 - Specificity)")
ax.set_ylabel("True Positive Rate (Sensitivity)")
ax.set_title("ROC curves: Heart disease prediction")
ax.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

Clinical context

In clinical practice, ROC curves are used to compare diagnostic tests. When evaluating a new biomarker for sepsis, you plot its ROC curve against existing biomarkers (CRP, procalcitonin). The test with the highest AUC has the best overall discriminative ability.

8.7 Feature importance

Question: Which risk factors matter most for predicting heart disease?

```
# Random forest feature importance
importances = rf.feature_importances_
importance_df = pd.DataFrame({
    "Feature": feature_cols,
    "Importance": importances
}).sort_values("Importance", ascending=True)

fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(importance_df["Feature"], importance_df["Importance"],
↪   color="steelblue")
ax.set_xlabel("Feature importance (Gini)")
ax.set_title("Which risk factors predict heart disease?")
plt.tight_layout()
plt.show()
```

```
# Top 5 features
print("Top 5 most important features:")
for _, row in importance_df.tail(5).iloc[::-1].iterrows():
    print(f"  {row['Feature']:12s}: {row['Importance']:.3f}")
```

Caution

Feature importance from random forests tells you which variables are useful for *prediction*, not which variables *cause* the outcome. A variable can be important for prediction because it is a proxy for something else. Do not confuse predictive importance with causal importance.

8.8 Calibration

A model says "this patient has a 70% risk of heart disease." Does that mean that among all patients the model gives 70%, roughly 70% actually have heart disease? If yes, the model is **well-calibrated**.

```
from sklearn.calibration import calibration_curve

# Calibration plot for random forest
prob_true, prob_pred = calibration_curve(y_test, y_prob_rf, n_bins=8)

fig, ax = plt.subplots(figsize=(6, 6))
ax.plot(prob_pred, prob_true, "o-", linewidth=2, label="Random Forest")
ax.plot([0, 1], [0, 1], "k--", label="Perfectly calibrated")
ax.set_xlabel("Mean predicted probability")
ax.set_ylabel("Observed proportion")
ax.set_title("Calibration plot")
ax.legend()
plt.tight_layout()
plt.show()
```

```
# Brier score: lower is better (0 = perfect)
from sklearn.metrics import brier_score_loss

brier_lr = brier_score_loss(y_test, y_prob_lr)
brier_rf = brier_score_loss(y_test, y_prob_rf)
print(f"Brier score (Logistic Regression): {brier_lr:.4f}")
print(f"Brier score (Random Forest):       {brier_rf:.4f}")
```

Clinical context

Calibration is critical in clinical prediction models. If a diabetes risk calculator tells a patient "you have a 40% chance of developing diabetes in 10 years," that number must be accurate — it drives treatment decisions. The Framingham Risk Score and QRISK are regularly re-calibrated for different populations.

8.9 Overfitting: the danger of small clinical datasets

```
# Demonstrate overfitting with varying dataset sizes
```

```
train_sizes = [30, 50, 100, 150, 200, len(X_train)]
results = []

for size in train_sizes:
    if size > len(X_train):
        continue
    X_sub = X_train.iloc[:size]
    y_sub = y_train.iloc[:size]

    rf_temp = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_temp.fit(X_sub, y_sub)

    train_acc = rf_temp.score(X_sub, y_sub)
    test_acc = rf_temp.score(X_test, y_test)
    results.append({"n_train": size, "train_acc": train_acc, "test_acc":
    ↪  test_acc})

results_df = pd.DataFrame(results)

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(results_df["n_train"], results_df["train_acc"], "o-",
        label="Training accuracy", linewidth=2)
ax.plot(results_df["n_train"], results_df["test_acc"], "s-",
        label="Test accuracy", linewidth=2)
ax.set_xlabel("Number of training patients")
ax.set_ylabel("Accuracy")
ax.set_title("Learning curve: more data reduces overfitting")
ax.legend()
ax.set_ylim(0.5, 1.05)
plt.tight_layout()
plt.show()
```

Python tip

Signs of overfitting: (1) training accuracy is much higher than test accuracy, (2) performance varies wildly across cross-validation folds, (3) adding more features hurts test performance. The cure: more data, simpler models, regularization, or cross-validation for model selection.

8.10 Practical: heart disease prediction pipeline

```
# Complete pipeline from raw data to evaluated model
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Full pipeline
pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
```

```
    ("classifier", RandomForestClassifier(n_estimators=200, max_depth=6,
                                          random_state=42))
])

# Cross-validate the full pipeline
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring="roc_auc")
print(f"Pipeline CV AUC: {cv_scores.mean():.3f} +/- {cv_scores.std():.3f}")

# Final fit and evaluation
pipeline.fit(X_train, y_train)
y_prob_final = pipeline.predict_proba(X_test)[:, 1]
y_pred_final = pipeline.predict(X_test)

print(f"\nFinal test AUC: {roc_auc_score(y_test, y_prob_final):.3f}")
print(f"\n{classification_report(y_test, y_pred_final, target_names=['No
↪  Disease', 'Disease'])}")
```

8.11 Ethics: fairness in clinical prediction models

Caution

Clinical prediction models can perpetuate and amplify health disparities. A model trained predominantly on one population may perform poorly on another. Before deploying any clinical ML model, ask:

- **Who is in the training data?** If 90% of patients are white males, the model may not work for women or minority groups.
- **Does performance differ across subgroups?** Check AUC, sensitivity, and specificity separately by sex, race, and age group.
- **What are the consequences of errors?** A false negative for cardiac risk in a young woman is more dangerous than a false positive.
- **Is the model transparent?** Clinicians and patients deserve to know how a prediction was made.

```
# Check model performance by sex
for sex_val, sex_label in [(1, "Male"), (0, "Female")]:
    mask = X_test["sex"] == sex_val
    if mask.sum() < 10:
        print(f"{sex_label}: too few samples ({mask.sum()})")
        continue

    auc_sub = roc_auc_score(y_test[mask], y_prob_final[mask])
    sens = (y_pred_final[mask] == 1)[y_test[mask] == 1].mean()
    spec = (y_pred_final[mask] == 0)[y_test[mask] == 0].mean()

    print(f"{sex_label:8s}: AUC = {auc_sub:.3f}, "
```

```
    f"Sensitivity = {sens:.3f}, Specificity = {spec:.3f}, "
    f"n = {mask.sum()}")
```

Clinical context

In 2019, a widely-used commercial algorithm for allocating healthcare resources was found to be biased against Black patients. The algorithm used healthcare *costs* as a proxy for healthcare *needs*, but because Black patients had lower costs due to systemic barriers to access, the algorithm systematically underestimated their needs. This affected millions of patients. The lesson: always audit your model for disparities.

Exercise

Using the UCI Heart Disease dataset:

1. **Baseline.** Fit a logistic regression model using all features. Report CV accuracy and AUC.
2. **Decision tree.** Fit a decision tree with `max_depth=3`. Visualize the tree. Can a clinician follow this decision rule at the bedside?
3. **Random forest.** Fit a random forest with 200 trees. Compare its AUC to logistic regression and the decision tree. Is the improvement worth the loss of interpretability?
4. **Feature selection.** Train a random forest using only the top 5 most important features. How does performance compare to using all 13 features?
5. **ROC comparison.** Plot ROC curves for all three models on the same figure. Which model would you choose for a screening tool? For a confirmatory test?
6. **Threshold analysis.** For the random forest, plot sensitivity and specificity as a function of the classification threshold (from 0 to 1). At what threshold are they equal?
7. **Calibration.** Plot calibration curves for logistic regression and random forest. Which model is better calibrated?
8. **Fairness audit.** Compare AUC, sensitivity, and specificity between male and female patients. Are there concerning disparities? What might explain them?
9. **Clinical deployment.** Write a paragraph (in a Markdown cell) describing how you would validate this model before using it in a real hospital. Consider: external validation, regulatory requirements, clinician workflow, and patient consent.

8.12 Chapter summary

- Use traditional statistics (regression) for inference and ML for prediction.

- Always split data into training and test sets. Never evaluate on training data.
- Cross-validation gives more robust performance estimates than a single split.
- Decision trees are interpretable and clinician-friendly but prone to overfitting.
- Random forests improve accuracy by averaging many trees.
- ROC curves and AUC compare the discriminative ability of different models.
- Feature importance reveals which risk factors drive predictions (but not causation).
- Calibration checks whether predicted probabilities match observed outcomes.
- Overfitting is the central danger of ML on small clinical datasets — always validate.
- Fairness audits are essential: check model performance across demographic subgroups before deployment.

Chapter 9

Geospatial and temporal health data

"An epidemic is a story told in time and space. If you can plot both, you can see where it came from and where it is going."

9.1 Time series in health

A *time series* is a sequence of measurements recorded at regular intervals: daily case counts, weekly death tolls, monthly vaccination doses. Health time series have three components:

- **Trend.** The long-term direction — is incidence rising or falling?
- **Seasonality.** Regular cycles — malaria peaks during rainy seasons, influenza peaks in winter.
- **Noise.** Random day-to-day fluctuation that obscures the signal.

Clinical context

Epidemic curves ("epi curves") are the most important time series in public health. During an outbreak, the shape of the epi curve tells you whether the source is a point exposure (sharp peak), a continuous source (plateau), or person-to-person transmission (successive waves).

9.2 Plotting time series with pandas

9.2.1 Date parsing and indexing

Pandas handles dates natively. The key is to parse date columns and set them as the index:

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

# Example: daily malaria cases at a district hospital
dates = pd.date_range("2023-01-01", periods=365, freq="D")
```

```
import numpy as np
np.random.seed(42)
# Simulate seasonal pattern: peak during rainy season (June-September)
day_of_year = dates.dayofyear
seasonal = 50 + 40 * np.sin(2 * np.pi * (day_of_year - 90) / 365)
cases = np.maximum(0, seasonal + np.random.normal(0, 12, 365)).astype(int)

df = pd.DataFrame({"date": dates, "cases": cases})
df["date"] = pd.to_datetime(df["date"])
df = df.set_index("date")
df.head()
```

9.2.2 Rolling averages

Day-to-day fluctuation makes trends hard to see. A *rolling average* smooths the curve:

```
df["cases_7d"] = df["cases"].rolling(window=7).mean()
df["cases_14d"] = df["cases"].rolling(window=14).mean()

fig, ax = plt.subplots(figsize=(12, 5))
ax.bar(df.index, df["cases"], alpha=0.3, label="Daily cases",
↪  color="steelblue")
ax.plot(df.index, df["cases_7d"], color="red", linewidth=2, label="7-day
↪  average")
ax.plot(df.index, df["cases_14d"], color="darkgreen", linewidth=2,
↪  label="14-day average")
ax.set_xlabel("Date")
ax.set_ylabel("Malaria cases")
ax.set_title("Daily malaria cases with rolling averages")
ax.legend()
plt.tight_layout()
plt.show()
```

Python tip

The `window` parameter sets the number of days to average over. A 7-day window removes day-of-week effects; a 14-day window smooths more aggressively. Choose based on how noisy your data is.

9.2.3 Trend decomposition

The `statsmodels` library can decompose a time series into trend, seasonal, and residual components:

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df["cases"], model="additive", period=30)
```

```
fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)
result.observed.plot(ax=axes[0], title="Observed")
result.trend.plot(ax=axes[1], title="Trend")
result.seasonal.plot(ax=axes[2], title="Seasonal")
result.resid.plot(ax=axes[3], title="Residual")
plt.tight_layout()
plt.show()
```

> **Caution**
>
> The `period` parameter must match the expected cycle length. For daily data with monthly seasonality, use `period=30`. For weekly data with annual seasonality, use `period=52`. A wrong period will produce misleading decompositions.

9.3 COVID-19 case curves

9.3.1 Loading JHU data

The Johns Hopkins University CSSE COVID-19 dataset is one of the most cited public health datasets in history:

```
# JHU CSSE COVID-19 confirmed cases (global, time series)
url = ("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/"
       "master/csse_covid_19_data/csse_covid_19_time_series/"
       "time_series_covid19_confirmed_global.csv")
confirmed = pd.read_csv(url)
print(f"Shape: {confirmed.shape}")
confirmed.head()
```

9.3.2 Reshaping wide to long format

The JHU data is in wide format (one column per date). We need to reshape it:

```
# Focus on 5 African countries
countries = ["South Africa", "Nigeria", "Kenya", "Egypt", "Morocco"]
africa = confirmed[confirmed["Country/Region"].isin(countries)]

# Group by country (some countries have province-level rows)
africa = africa.groupby("Country/Region").sum(numeric_only=True)
africa = africa.drop(columns=["Lat", "Long"], errors="ignore")

# Reshape: wide -> long
africa_long = africa.T
africa_long.index = pd.to_datetime(africa_long.index)
africa_long.index.name = "date"
africa_long.head()
```

9.3.3 Computing daily cases and 7-day averages

```
# Cumulative -> daily new cases
daily = africa_long.diff().clip(lower=0)  # clip removes negative corrections

# 7-day rolling average
daily_7d = daily.rolling(7).mean()

# Plot
fig, ax = plt.subplots(figsize=(14, 6))
for country in countries:
    ax.plot(daily_7d.index, daily_7d[country], linewidth=2, label=country)
ax.set_xlabel("Date")
ax.set_ylabel("Daily new cases (7-day average)")
ax.set_title("COVID-19 daily cases in 5 African countries")
ax.legend()
plt.tight_layout()
plt.show()
```

Clinical context

The `.diff()` method converts cumulative counts to daily increments. Negative values occasionally appear because of data corrections (e.g., a country revises its total downward). Clipping at zero is the standard approach, though it slightly inflates subsequent days.

9.4 Introduction to geospatial data

Geospatial data links measurements to locations on Earth. Two common formats:

- **Shapefiles** (`.shp`) — the traditional GIS format. A shapefile is actually a set of files (`.shp`, `.shx`, `.dbf`, `.prj`) that together define polygon geometries and attributes.
- **GeoJSON** (`.geojson`) — a single JSON file. Easier to share, read, and use on the web.

Both formats store *geometries* (country borders, district boundaries, hospital point locations) and *attributes* (population, case count, mortality rate) in the same structure.

9.5 geopandas for disease mapping

`geopandas` extends pandas with spatial capabilities. A `GeoDataFrame` is a DataFrame with a special `geometry` column that holds shapes.

9.5.1 Installation

```
# In Google Colab or local environment
!pip install geopandas mapclassify
```

9.5.2 Loading a world map

```
import geopandas as gpd

# Natural Earth dataset (built into geopandas)
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
print(f"Columns: {world.columns.tolist()}")
print(f"CRS: {world.crs}")  # Coordinate Reference System
world.head()
```

9.5.3 Choropleth maps

A *choropleth* map colors regions according to a variable. This is the standard tool for mapping disease burden by country or district:

```
# Filter to Africa
africa_map = world[world["continent"] == "Africa"].copy()

# Plot population estimate
fig, ax = plt.subplots(figsize=(10, 10))
africa_map.plot(column="pop_est", cmap="YlOrRd", legend=True,
                legend_kwds={"label": "Population", "shrink": 0.6},
                edgecolor="black", linewidth=0.5, ax=ax)
ax.set_title("Population estimates in African countries")
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Python tip

Good colormaps for health data: `"YlOrRd"` (sequential, low-to-high severity), `"RdYlGn_r"` (diverging, red = bad, green = good), `"Blues"` (sequential, neutral). Avoid rainbow colormaps (`"jet"`) — they mislead the eye.

9.6 Mapping malaria prevalence by African country

Let us combine real health data with geospatial plotting. We will use the WHO Global Health Observatory malaria incidence data:

```
# WHO GHO: Estimated malaria incidence (per 1000 population at risk)
url = "https://apps.who.int/gho/athena/api/GHO/MALARIA_INCIDENCE?format=csv"
malaria = pd.read_csv(url)
print(malaria.columns.tolist())
malaria.head()
```

```python
# Filter to most recent year and country-level data
latest_year = malaria["YEAR (DISPLAY)"].max()
malaria_latest = malaria[malaria["YEAR (DISPLAY)"] == latest_year].copy()

# Keep relevant columns
malaria_latest = malaria_latest[["COUNTRY (DISPLAY)", "Numeric"]].copy()
malaria_latest.columns = ["country", "incidence_per_1000"]
malaria_latest.head()
```

```python
# Merge with Africa map
# Match country names (some need manual fixing)
name_map = {
    "United Republic of Tanzania": "Tanzania",
    "Democratic Republic of the Congo": "Dem. Rep. Congo",
    "Central African Republic": "Central African Rep.",
    "South Sudan": "S. Sudan",
    "Côte d'Ivoire": "Côte d'Ivoire",
    "Equatorial Guinea": "Eq. Guinea",
}
malaria_latest["country"] = malaria_latest["country"].replace(name_map)

africa_malaria = africa_map.merge(malaria_latest, left_on="name",
                                  right_on="country", how="left")
```

```python
fig, ax = plt.subplots(figsize=(10, 10))
africa_malaria.plot(column="incidence_per_1000", cmap="YlOrRd",
                    legend=True, missing_kwds={"color": "lightgrey"},
                    legend_kwds={"label": "Incidence per 1,000",
                                 "shrink": 0.6},
                    edgecolor="black", linewidth=0.5, ax=ax)
ax.set_title(f"Malaria incidence in Africa ({latest_year})")
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Clinical context

The malaria burden in sub-Saharan Africa accounts for approximately 95% of global malaria deaths. Choropleth maps immediately reveal the geographic concentration and help target intervention programs such as insecticide-treated nets (ITNs) and indoor residual spraying (IRS).

9.7 Combining temporal and spatial analysis

9.7.1 Small multiples: one map per time period

A practical alternative to animation is the *small multiples* approach — one map per time period side by side:

```
# Example: COVID-19 cumulative cases per country at 3 time points
dates_to_plot = ["2020-06-01", "2021-01-01", "2021-06-01"]

fig, axes = plt.subplots(1, 3, figsize=(18, 7))
for ax, date_str in zip(axes, dates_to_plot):
    date_col = pd.to_datetime(date_str).strftime("%-m/%-d/%y")
    if date_col in africa.columns:
        data_col = africa[date_col]
    else:
        # Find nearest date
        all_dates = pd.to_datetime(africa.columns, errors="coerce")
        nearest = all_dates[all_dates <= pd.to_datetime(date_str)][-1]
        data_col = africa[nearest.strftime("%-m/%-d/%y")]

    temp = africa_map.copy()
    temp = temp.merge(data_col.reset_index(), left_on="name",
                      right_on="Country/Region", how="left")
    temp.plot(column=data_col.name, cmap="Reds", legend=True,
              edgecolor="black", linewidth=0.5, ax=ax,
              missing_kwds={"color": "lightgrey"})
    ax.set_title(date_str)
    ax.set_axis_off()

plt.suptitle("COVID-19 cumulative cases in Africa", fontsize=14)
plt.tight_layout()
plt.show()
```

9.7.2 Animated maps (optional advanced)

For presentations, animated GIFs can show epidemic spread over time. The idea: create one frame per time step, then combine with `imageio` or `matplotlib.animation`:

```
import matplotlib.animation as animation

fig, ax = plt.subplots(figsize=(10, 10))

def update(frame_date):
    ax.clear()
    # Merge case data for this date with the map
    temp = africa_map.copy()
    # ... merge logic similar to above ...
    temp.plot(column="cases", cmap="Reds", ax=ax, edgecolor="black",
              linewidth=0.5, missing_kwds={"color": "lightgrey"},
```

```
            vmin=0, vmax=max_cases)
    ax.set_title(f"COVID-19 cases - {frame_date}")
    ax.set_axis_off()

# Create animation (one frame per month)
# ani = animation.FuncAnimation(fig, update, frames=monthly_dates,
#                               interval=500)
# ani.save("covid_africa.gif", writer="pillow")
```

Caution

Animated maps are visually impressive but can be hard to interpret scientifically. For publication, small multiples or static snapshots with a time slider (in a dashboard) are preferred. Use animations for presentations and communication with the public.

9.8 Practical: COVID-19 dashboard for 5 African countries

Exercise

Build a multi-panel COVID-19 dashboard for South Africa, Nigeria, Kenya, Egypt, and Morocco. Your dashboard should have four panels:
Panel 1 — Confirmed cases over time.

1. Load the JHU CSSE time series data for confirmed cases: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv
2. Filter to the 5 countries. Compute daily new cases and 7-day rolling averages.
3. Plot all 5 countries on one axis with a legend.

Panel 2 — Deaths over time.

1. Load the JHU deaths time series: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv
2. Compute daily deaths (7-day average) and plot.

Panel 3 — Vaccination progress.

1. Load the Our World in Data vaccination dataset: https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/vaccinations.csv
2. Filter to the 5 countries. Plot `people_fully_vaccinated_per_hundred` over time.

Panel 4 — Choropleth map.

1. Using `geopandas`, create a choropleth of total confirmed cases (or cases per million) across all African countries at the latest available date.
2. Highlight the 5 focus countries with thicker borders.

Layout:

```
fig, axes = plt.subplots(2, 2, figsize=(18, 14))
# axes[0, 0] -> Panel 1: Cases
# axes[0, 1] -> Panel 2: Deaths
# axes[1, 0] -> Panel 3: Vaccination
# axes[1, 1] -> Panel 4: Choropleth map
```

Bonus:

- Add a table below the figure showing summary statistics for each country: total cases, total deaths, case fatality rate (%), vaccination coverage (%).
- Normalize cases and deaths per 100 000 population to allow fair comparison.

9.9 Chapter summary

- Health time series have three components: trend, seasonality, and noise.
- Rolling averages (`.rolling()`) smooth daily fluctuations; `seasonal_decompose()` separates components formally.
- The JHU CSSE dataset provides global COVID-19 case, death, and recovery time series. Use `.diff()` to convert cumulative counts to daily new cases.
- Geospatial data comes in shapefiles (`.shp`) or GeoJSON (`.geojson`). `geopandas` reads both.
- Choropleth maps color regions by a health variable — use `.plot(column=...)` with an appropriate colormap.
- Merging health data with map data requires matching country names — always check for mismatches.
- Small multiples (one map per time period) are more scientifically rigorous than animations.

Chapter 10

Capstone project

> *"The best way to learn data analysis is to analyze data. Not toy data — real data, with real messiness, about real health questions."*

10.1 Project guidelines

The capstone project is your opportunity to apply every skill from this course to a real health question. You will work individually or in pairs.

10.1.1 Timeline

- **Week 1.** Choose a project, download the data, perform initial exploration.
- **Week 2.** Complete the analysis, generate figures and tables, draft the report.
- **Week 3.** Finalize the report and prepare a 5-minute oral presentation.

10.1.2 Expectations

Your project must include:

1. **A clear health question.** Not "explore the data" but "is diabetes prevalence associated with BMI after adjusting for age and sex?"
2. **Real public data.** No simulated datasets. All data sources must be cited.
3. **Reproducible code.** A Jupyter notebook that runs from top to bottom without errors.
4. **At least 3 visualizations.** Relevant, labeled, and interpretable.
5. **At least 1 statistical test or model.** With proper interpretation.
6. **A written report.** 5–8 pages following the template below.
7. **An oral presentation.** 5 minutes with slides.

Caution

Academic integrity: you may use AI tools (ChatGPT, GitHub Copilot) to help write code, but you must understand every line. During the oral presentation, you will be asked to explain your code and interpret your results. Copy-pasting code you do not understand will be obvious.

10.2 Suggested projects

Choose one of the five projects below, or propose your own (subject to instructor approval). Each project specifies the objective, dataset(s), required analyses, and expected deliverables.

10.2.1 Project 1: Diabetes risk factor analysis

Objective. Identify the strongest risk factors for type 2 diabetes and build a predictive model.

Datasets.

- **Pima Indians Diabetes Dataset** (UCI/Kaggle):
  `https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv`
  768 female patients, 8 features (pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, age), binary outcome.
- **CDC BRFSS** (Behavioral Risk Factor Surveillance System):
  `https://www.cdc.gov/brfss/annual_data/annual_data.htm`
  Annual survey of 400 000+ US adults. Download the most recent year's CSV.

Required analyses.

1. Load and clean the Pima dataset. Handle zeros in glucose, blood pressure, BMI (these are missing values coded as 0).
2. Exploratory analysis: distributions, correlations, box plots by diabetes status.
3. Logistic regression: which variables are significant predictors? Report odds ratios with 95% confidence intervals.
4. Compare logistic regression with a random forest classifier. Report accuracy, sensitivity, specificity, and AUC-ROC.
5. (Bonus) Repeat the risk factor analysis on BRFSS data and compare findings.

Deliverables.

- Correlation heatmap of all features.
- ROC curves comparing logistic regression and random forest.
- Table of odds ratios with confidence intervals.
- Written interpretation: which risk factors matter most clinically?

Clinical context

The Pima Indians dataset is historically important but has ethical considerations. The data was collected from the Akimel O'odham (Pima) community, which has one of the highest rates of type 2 diabetes in the world. When presenting, acknowledge the population and avoid generalizing findings to other groups.

10.2.2 Project 2: COVID-19 impact analysis across African countries

Objective. Analyze how COVID-19 affected African countries differently and explore correlations with health system capacity and socioeconomic factors.

Datasets.

- **JHU CSSE COVID-19 data**:
  https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series
  Global confirmed cases, deaths, and recovered (daily time series).
- **Our World in Data** (vaccination, testing):
  https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv
  Comprehensive dataset with cases, deaths, vaccinations, testing, hospital data, and demographic indicators.
- **World Bank Open Data** (GDP, health expenditure, hospital beds):
  https://data.worldbank.org/
  Use indicator codes: `SH.XPD.CHEX.PC.CD` (health expenditure per capita), `SH.MED.BEDS.ZS` (hospital beds per 1 000), `NY.GDP.PCAP.CD` (GDP per capita).
- **Africa CDC**:
  https://africacdc.org/covid-19/
  Continental-level dashboards and reports.

Required analyses.

1. Select 10–15 African countries across different subregions (West, East, Southern, North, Central Africa).
2. Plot epidemic curves (7-day rolling average) for each country. Identify wave patterns.
3. Compute key metrics: total cases per million, deaths per million, case fatality rate, vaccination rate.
4. Merge with World Bank indicators. Compute correlations between COVID-19 outcomes and health system capacity (health expenditure, hospital beds, physician density).
5. Build a choropleth map of case fatality rates across Africa.

6. (Bonus) Run a multiple regression: does health expenditure per capita predict case fatality rate after controlling for median age and GDP?

Deliverables.

- Epidemic curves for all selected countries (small multiples or grouped plot).
- Scatter plot of health expenditure vs. case fatality rate.
- Choropleth map of Africa showing COVID-19 burden.
- Summary table with key metrics for all countries.

10.2.3 Project 3: Maternal mortality determinants

Objective. Investigate which health system and socioeconomic factors are associated with maternal mortality across countries.

Datasets.

- **WHO Global Health Observatory** — Maternal mortality ratio (MMR): `https://apps.who.int/gho/athena/api/GHO/MDG_0000000026?format=csv`
- **WHO GHO** — Skilled birth attendance: `https://apps.who.int/gho/athena/api/GHO/MDG_0000000025?format=csv`
- **WHO GHO** — Antenatal care coverage (4+ visits): `https://apps.who.int/gho/athena/api/GHO/WHS4_154?format=csv`
- **World Bank** — Female literacy rate, GDP per capita, health expenditure: `https://data.worldbank.org/`
- **DHS Program** (Demographic and Health Surveys) — survey-level indicators: `https://www.statcompiler.com/`
  Provides subnational data on maternal health, contraceptive use, and antenatal care for African and Asian countries.

Required analyses.

1. Load MMR data and filter to the most recent year. Identify the 20 countries with the highest MMR.
2. Merge with skilled birth attendance, antenatal care coverage, and socioeconomic indicators.
3. Exploratory analysis: scatter plots of MMR vs. each predictor. Compute Pearson and Spearman correlations.
4. Multiple linear regression: which factors significantly predict MMR? Check model assumptions (residual plots, normality).
5. Create a choropleth map of MMR across Africa.
6. (Bonus) Compare MMR trends over time (2000–2020) for 5 countries that implemented specific maternal health programs.

Deliverables.

- Scatter plots with regression lines (MMR vs. skilled birth attendance, MMR vs. health expenditure).
- Regression table with coefficients, standard errors, p-values, and R^2.
- Choropleth map of maternal mortality in Africa.
- Discussion of policy implications: which interventions could reduce MMR?

Clinical context

Maternal mortality is one of the starkest health inequities in the world. Sub-Saharan Africa accounts for approximately 70% of global maternal deaths. Most of these deaths are preventable with skilled birth attendants, emergency obstetric care, and adequate antenatal visits.

10.2.4 Project 4: Heart disease prediction model comparison

Objective. Compare multiple machine learning models for predicting heart disease and identify the most important clinical predictors.

Dataset.

- **UCI Heart Disease Dataset** (Cleveland):
  https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data
  303 patients, 13 clinical features (age, sex, chest pain type, resting BP, cholesterol, fasting blood sugar, resting ECG, max heart rate, exercise-induced angina, ST depression, slope, number of major vessels, thalassemia), binary target (presence of heart disease).
- Column names: `age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal, target`.

Required analyses.

1. Load and clean the data. Handle missing values (coded as `?` in the original file).
2. Exploratory analysis: distributions of each feature by heart disease status.
3. Train-test split (80/20). Use stratified splitting to maintain class balance.
4. Train and evaluate 4 models:
   - Logistic regression
   - Decision tree
   - Random forest
   - K-nearest neighbors (KNN)
5. For each model: report accuracy, precision, recall, F1-score, and AUC-ROC.

6. Plot ROC curves for all 4 models on the same figure.
7. Feature importance: which clinical variables are the strongest predictors?

Deliverables.

- Comparison table of model performance metrics.
- ROC curves for all models on one plot.
- Feature importance bar chart (from random forest).
- Confusion matrix for the best-performing model.
- Clinical interpretation: do the top predictors align with cardiology guidelines?

Python tip

Use `sklearn.model_selection.cross_val_score` with 5-fold cross-validation instead of a single train-test split. This gives a more reliable estimate of model performance, especially with a small dataset like this one (303 patients).

10.2.5 Project 5: Malaria burden and intervention effectiveness

Objective. Analyze trends in malaria burden across African countries and evaluate whether key interventions (bed nets, indoor spraying, treatment) are associated with declining incidence.

Datasets.

- **WHO GHO** — Malaria incidence (per 1 000 population at risk):
  https://apps.who.int/gho/athena/api/GHO/MALARIA_INCIDENCE?format=csv
- **WHO GHO** — Malaria mortality rate:
  https://apps.who.int/gho/athena/api/GHO/MALARIA_DEATHS_PER100000?format=csv
- **WHO World Malaria Report data** — intervention coverage (ITN use, IRS coverage, ACT treatment):
  https://www.who.int/teams/global-malaria-programme/reports/world-malaria-report
  Annex data tables are available as Excel downloads.
- **The Malaria Atlas Project**:
  https://malariaatlas.org/
  Subnational prevalence estimates and raster maps.
- **DHS Program**:
  https://www.statcompiler.com/
  Survey-based data on ITN ownership and use, IRS coverage, and treatment-seeking behavior.

Required analyses.

1. Load malaria incidence and mortality data. Filter to sub-Saharan African countries.

2. Plot incidence trends (2000–2022) for 10 countries with the highest burden.

3. Merge with intervention coverage data (ITN use, IRS, ACT access).

4. Compute correlations between changes in intervention coverage and changes in incidence over time (i.e., does increasing ITN use predict decreasing incidence?).

5. Create a choropleth map of current malaria incidence across Africa.

6. Linear regression: predict change in malaria incidence from change in intervention coverage, controlling for GDP per capita and urbanization.

7. (Bonus) Create a before-and-after comparison for countries that scaled up ITN distribution after the Global Fund grants.

Deliverables.

- Time series plot of malaria incidence for 10 countries (2000–2022).
- Scatter plot of ITN use change vs. incidence change with regression line.
- Choropleth map of malaria incidence across Africa.
- Regression table with interpretation.
- Policy discussion: which interventions show the strongest association with declining malaria?

Clinical context

Between 2000 and 2015, malaria mortality in Africa declined by approximately 44%, largely driven by the scale-up of ITNs, IRS, and artemisinin-based combination therapies (ACTs). Since 2015, progress has stalled, making this analysis timely and policy-relevant.

10.3 Report template

Your written report should follow this structure (5–8 pages, excluding code appendix):

10.3.1 1. Introduction (0.5–1 page)

- Health context: why does this question matter?
- Specific research question or objective.
- Brief description of the dataset(s) used.

10.3.2 2. Data description (1 page)

- Source(s) and citation(s).
- Number of observations and variables.
- Key variables: name, type, unit, and clinical meaning.
- Summary statistics table (`.describe()`).
- Missing data: how much, which variables, how handled.

10.3.3 3. Methods (1 page)

- Data cleaning steps (handling missing values, recoding variables, outlier treatment).
- Statistical tests used and why (e.g., "we used the chi-squared test because both variables are categorical").
- Models used (logistic regression, random forest, etc.) with hyperparameters.
- Validation strategy (train-test split, cross-validation).

10.3.4 4. Results (1.5–2 pages)

- Figures and tables with captions.
- Report test statistics, p-values, confidence intervals.
- Model performance metrics.
- Do not interpret here — just present the numbers.

10.3.5 5. Discussion (1–1.5 pages)

- What do the results mean clinically?
- How do your findings compare with published literature?
- What surprised you?
- What are the practical implications?

10.3.6 6. Limitations (0.5 page)

- Data quality issues (missing data, measurement error, selection bias).
- Generalizability: does the dataset represent the population you care about?
- What analyses would you do with more time or data?

Caution

Every figure must have a title, labeled axes, and a caption explaining what the reader should see. Every table must have a header row and a caption. Figures without labels are the most common reason for lost marks.

10.4 Presentation guidelines

You will give a 5-minute oral presentation followed by 2–3 minutes of questions.

10.4.1 Structure

1. **Slide 1: Title.** Project title, your name, date.
2. **Slide 2: Motivation.** Why does this health question matter? One or two key facts.
3. **Slide 3: Data.** What data did you use? How many observations? What are the key variables?
4. **Slide 4–5: Key results.** Your 2–3 most important figures or tables. Walk the audience through each one.
5. **Slide 6: Conclusions.** One or two take-home messages. What should the audience remember?

10.4.2 Tips

- **5 minutes is short.** Practice with a timer. Cut anything that is not essential.
- **One message per slide.** Do not crowd slides with text.
- **Speak to the audience, not the screen.**
- **Anticipate questions.** Know your limitations and be ready to discuss them.
- **No code on slides.** Show results, not the code that produced them.

Python tip

Save your figures as high-resolution PNGs for slides:

```
fig.savefig("figure1.png", dpi=300, bbox_inches="tight")
```

This avoids blurry screenshots and ensures your plots look professional in a presentation.

10.5 Grading rubric

The capstone project is graded out of 100 points:

Component	Points	Criteria
Research question	10	Clear, specific, and health-relevant
Data handling	15	Proper loading, cleaning, missing value treatment, and documentation
Exploratory analysis	15	Appropriate summary statistics, distributions, and at least 3 well-labeled visualizations
Statistical analysis	20	Correct choice and application of tests or models; proper interpretation of p-values, confidence intervals, and effect sizes
Report quality	20	Follows the template; clear writing; figures have titles, labels, and captions; limitations honestly discussed
Code quality	10	Reproducible notebook that runs top-to-bottom; comments explain key steps; no unnecessary code
Oral presentation	10	Clear, within time limit, answers questions confidently
Total	**100**	

10.5.1 Grade boundaries

- **A (90–100):** Excellent analysis with insightful interpretation. Code is clean and well-documented. Goes beyond the minimum requirements.
- **B (75–89):** Solid analysis with correct methodology. Minor issues in presentation or interpretation.
- **C (60–74):** Meets basic requirements but has notable gaps (e.g., missing visualizations, superficial interpretation, code errors).
- **D/F (< 60):** Incomplete analysis, major methodological errors, or non-reproducible code.

10.6 Getting started: a checklist

Exercise

Before you leave today, complete this checklist:

1. Choose your project (1–5 or a custom proposal).
2. Download your primary dataset. Load it into a Jupyter notebook and run `.head()`, `.shape`, `.info()`, and `.describe()`.
3. Write your research question in one sentence.

4. List 3 visualizations you plan to create.
5. List 1 statistical test or model you plan to use.
6. Identify potential problems: missing data, messy column names, multiple files to merge.
7. Create a project folder with the structure:

```
capstone_project/
    data/
        raw/          # original downloaded files
        cleaned/      # processed data
    notebooks/
        01_exploration.ipynb
        02_analysis.ipynb
    figures/
    report.pdf
```

8. Submit your project choice and research question to the instructor by the end of the week.

10.7 Chapter summary

- The capstone project integrates all course skills: data loading, cleaning, exploration, visualization, statistical analysis, and interpretation.
- Five suggested projects cover diabetes, COVID-19, maternal mortality, heart disease, and malaria — all using real, publicly available datasets.
- Each project requires a clear health question, reproducible code, at least 3 visualizations, and at least 1 statistical test or model.
- The report follows a standard structure: Introduction, Data, Methods, Results, Discussion, Limitations.
- Presentations are 5 minutes — focus on motivation, key results, and conclusions.
- The grading rubric rewards clear thinking and honest interpretation over complex methodology.

Appendix A: Python installation guide

Option 1: Google Colab (recommended for beginners)

Go to https://colab.research.google.com. Sign in with a Google account. No installation needed. All libraries used in this course are pre-installed.

Option 2: Anaconda (local installation)

Download Anaconda from https://www.anaconda.com/download. Install with default settings. Open Jupyter Notebook from Anaconda Navigator.

Required libraries

```
pip install pandas matplotlib seaborn scipy scikit-learn lifelines geopandas
```

Appendix B: Health data sources

Source	Description	URL
WHO GHO	Global health indicators (190+ countries, 1000+ indicators)	who.int/data/gho
Gapminder	Health, income, population by country and year	gapminder.org/data
Our World in Data	COVID-19, mortality, disease burden, vaccination	ourworldindata.org
CDC NHANES	US national health survey (labs, questionnaires, exams)	cdc.gov/nchs/nhanes
UCI ML Repository	Curated ML datasets (heart disease, diabetes, cancer)	archive.ics.uci.edu
World Bank	Health expenditure, life expectancy, maternal mortality	data.worldbank.org
Africa CDC	African disease surveillance and outbreak data	africacdc.org
JHU COVID-19	Global COVID-19 cases, deaths, recoveries (time series)	github.com/CSSEGISandData
DHS Program	Demographic and Health Surveys (50+ African countries)	dhsprogram.com