

Study Delight Publications

From
Python
Basics to
Real Data
and AI

PYTHON

Simplified

Learn
Analyze
Build
Apply

PART 5

Python for Data + AI



NumPy
& Arrays



Pandas
& Data Analysis



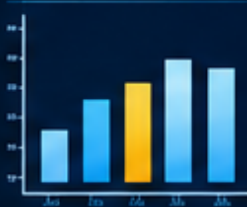
Data
Visualization



Working with
Real Data

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

data = pd.read_csv("data.csv")
print(data.head())
```



Machine
Learning
Basics



Automation
Projects



Real-World
Case Studies



Build
End-to-End
Projects

Turn Data into Insights
and Ideas into Intelligent Programs



Clear
Explanations



Practical
Examples



Real-World
Datasets



Mini
Projects



Career Ready
Skills

pythonforaiml.com

Python Simplified – Part 5

INDEX

1	Part 5 – Python for Data + AI	1
2	NumPy – Arrays	2
3	NumPy – Shape & Indexing	3
4	NumPy – Vectorized Operations	4
5	NumPy – Aggregations	5
6	Pandas – Series & DataFrame	6
7	Pandas – Load CSV	7
8	Pandas – Select & Filter	8
9	Pandas – Missing Values	9
10	Pandas – Sort & Group	10
11	Pandas – Merge Data	11
12	Matplotlib – Visualize	12
13	Data Analysis Mini Project	13
14	Automation with pathlib	14
15	Automation – Batch Rename	15
16	Automation – CSV Report	16
17	Regular Expressions	17
18	APIs – Getting Data	18
19	JSON – Working with API Data	19
20	Machine Learning – The Big Idea	20
21	Features & Labels	21
22	Train & Test Data	22
23	Scikit-learn Workflow	23
24	Linear Regression	24
25	Classification	25
26	Evaluation Metrics	26
27	ML Pipelines	27
28	AI / ML Project Structure	28
29	Final Capstone – Smart Sales Analyzer	29
30	Book Review – From Python to AI	30

PART 5 • PYTHON FOR DATA + AI

From Python basics to real data and intelligent programs

THEORY

This final part connects the Python skills from Parts 1–4 to the world of data analysis, automation and machine learning. The goal is not to memorize libraries. The goal is to understand the workflow: collect data, clean it, explore it, automate useful work, and finally build a simple model.

EXAMPLE

A practical data workflow: 1. Load data 2. Inspect it 3. Clean it 4. Analyze patterns 5. Visualize results 6. Automate repeated work 7. Build an ML model when prediction is useful

OUTPUT / RESULT

Python becomes powerful when small language features are combined into a repeatable workflow.

CONCEPT NOTE

Write one real-world problem you would like Python to solve.

PRACTICE

Start ☒ understand ☒ code ☒ test ☒ improve

STUDY LOOP: Understand ☒ code ☒ test ☒ explain

NUMPY • ARRAYS

Think in numbers, not individual variables

THEORY

NumPy is designed for numerical computing. Its central object is the array. An array can hold many numbers in an organized structure and lets us perform mathematical operations efficiently. Instead of writing a loop for every calculation, we can often operate on the whole array at once.

EXAMPLE

```
import numpy as np
scores = np.array([72, 81, 65, 90])
print(scores)
print(scores + 5)
print(scores.mean())
```

OUTPUT / RESULT

```
[72 81 65 90] [77 86 70 95] 77.0
```

CONCEPT NOTE

An array stores related values together and supports fast numerical operations.

PRACTICE

Create an array of five temperatures and calculate its average.

NUMPY • SHAPE & INDEXING

Understand the structure of your data

THEORY

An array has a shape. A one-dimensional array has one axis; a table-like array has rows and columns. Shape is important because many numerical operations depend on how data is arranged. Indexing starts at zero, just like Python lists.

EXAMPLE

```
import numpy as np marks = np.array([[70, 80, 90], [60, 75, 85]]) print(marks.shape) print(marks[0, 1]) print(marks[:, 0])
```

OUTPUT / RESULT

```
(2, 3) 80 [70 60]
```

CONCEPT NOTE

shape tells us the dimensions: here 2 rows × 3 columns.

PRACTICE

Make a 3×3 array and print its middle value.

STUDY LOOP: Shape → index → verify

NUMPY • VECTORIZED OPERATIONS

Let the array do the loop

THEORY

Vectorization means applying an operation to many values at once. This makes code shorter and often faster. It also makes the mathematical intention easier to read. Think of an array operation as saying what should happen to the data, rather than manually describing every iteration.

EXAMPLE

```
import numpy as np
prices = np.array([100, 200, 300])
discounted = prices * 0.90
print(discounted)
```

OUTPUT / RESULT

```
[ 90. 180. 270.]
```

CONCEPT NOTE

`prices * 0.90` applies the same calculation to every element.

PRACTICE

Calculate a 5% increase for an array of salaries.

NUMPY • AGGREGATIONS

Summaries turn many values into useful information

THEORY

Aggregation reduces many values into a summary. Common operations include sum, mean, minimum, maximum and standard deviation. These are the building blocks of descriptive statistics. Always ask what the summary means in the context of the problem.

EXAMPLE

```
import numpy as np
sales = np.array([120, 150, 90, 210, 180])
print("Total:", sales.sum())
print("Average:", sales.mean())
print("Highest:", sales.max())
```

OUTPUT / RESULT

Total: 750 Average: 150.0 Highest: 210

CONCEPT NOTE

Statistics are useful only when interpreted: an average of 150 means the typical value in this small sample is 150.

PRACTICE

Find total, average and minimum for your own five values.

PANDAS • SERIES & DATAFRAME

Python tables for practical data work

THEORY

Pandas is one of the most useful Python libraries for tabular data. A Series is similar to a labeled column. A DataFrame is a table made of rows and columns. DataFrames let us inspect, filter, transform and summarize datasets using readable operations.

EXAMPLE

```
import pandas as pd
df = pd.DataFrame({ "Name": ["Asha", "Ravi", "Neha"], "Marks": [82, 74, 91] })
print(df)
print(df["Marks"].mean())
```

OUTPUT / RESULT

```
Name Marks 0 Asha 82 1 Ravi 74 2 Neha 91 82.33333333333333
```

CONCEPT NOTE

A DataFrame is a structured table. Each column has a name and each row has an index.

PRACTICE

Create a DataFrame containing three products and their prices.

STUDY LOOP: Create ✎ inspect ✎ summarize

PANDAS • LOAD CSV

Bring real-world data into Python

THEORY

CSV files are common because they are simple, portable and easy to create. Pandas can load a CSV into a DataFrame. After loading, do not immediately analyze it. First inspect column names, row counts, data types and a few records.

EXAMPLE

```
import pandas as pd
df = pd.read_csv("sales.csv")
print(df.head())
print(df.shape)
print(df.columns)
```

OUTPUT / RESULT

head() shows the first rows. shape shows rows and columns. columns shows the available field names.

CONCEPT NOTE

Loading data is only the first step. Inspection prevents many later mistakes.

PRACTICE

Load any small CSV you have and inspect its first five rows.

STUDY LOOP: Load ✕ inspect ✕ understand

PANDAS • SELECT & FILTER

Ask precise questions of a table

THEORY

Filtering means keeping only rows that satisfy a condition. This is one of the most important data-analysis skills because real datasets often contain thousands of rows. The condition should express the business or analytical question you are asking.

EXAMPLE

```
high = df[df["Marks"] >= 80] print(high) selected =  
df[["Name", "Marks"]] print(selected)
```

OUTPUT / RESULT

The first expression keeps students with marks ≥ 80 . The second keeps only the requested columns.

CONCEPT NOTE

Filtering is not changing the original question; it creates a focused view of the data.

PRACTICE

Filter products whose price is greater than 500.

STUDY LOOP: Question $\rightarrow$ condition $\rightarrow$ result

PANDAS • MISSING VALUES

Dirty data is normal

THEORY

Real datasets frequently contain missing values. A blank cell might mean unknown, not applicable or simply a data-entry problem. Never blindly replace every missing value. First understand why values are missing, then choose a strategy such as dropping rows, filling with a statistic, or fixing the source.

EXAMPLE

```
print(df.isna().sum()) df["Marks"] = df["Marks"].fillna(df["Marks"].mean() )
```

OUTPUT / RESULT

isna().sum() counts missing values by column. fillna() replaces missing marks with the column average.

CONCEPT NOTE

The correct cleaning strategy depends on what the missing value means.

PRACTICE

Find missing values in a DataFrame and decide how you would handle them.

PANDAS • SORT & GROUP

Turn rows into patterns

THEORY

Sorting helps us inspect high and low values. Grouping is more powerful: it allows us to calculate summaries for categories. For example, instead of one total sales number, we can calculate total sales for each product or region.

EXAMPLE

```
print(df.sort_values("Marks", ascending=False))  
summary = df.groupby("Department")["Sales"].sum()  
print(summary)
```

OUTPUT / RESULT

sort_values() orders records. groupby() creates category-wise summaries.

CONCEPT NOTE

Grouping is often where raw rows become business insights.

PRACTICE

Group sales by product and calculate the total.

PANDAS • MERGE DATA

Combine related tables

THEORY

Often one dataset is not enough. Customer information may live in one table while orders live in another. A merge combines them using a shared key such as `customer_id`. The key must be meaningful and correctly typed, otherwise the result can be incomplete or misleading.

EXAMPLE

```
customers = pd.DataFrame({ "id": [1, 2], "name": ["Asha",  
"Ravi"] }) orders = pd.DataFrame({ "id": [1, 1, 2], "amount":  
[500, 300, 700] }) result = customers.merge(orders, on="id")  
print(result)
```

OUTPUT / RESULT

The customer name is attached to each matching order.

CONCEPT NOTE

A merge is similar to connecting two database tables using a common key.

PRACTICE

Create two small tables and merge them using an ID column.

MATPLOTLIB • VISUALIZE

A picture can reveal a pattern quickly

THEORY

Charts are useful when numbers become difficult to compare mentally. A line chart can show change over time; a bar chart compares categories; a scatter plot can show relationships. Visualization should answer a question, not merely decorate a report.

EXAMPLE

```
import matplotlib.pyplot as plt
months = ["Jan", "Feb", "Mar", "Apr"]
sales = [120, 150, 135, 190]
plt.plot(months, sales, marker="o")
plt.title("Monthly Sales")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show()
```

OUTPUT / RESULT

The chart shows how sales change from month to month.

CONCEPT NOTE

Choose the chart based on the question: trend, comparison, distribution or relationship.

PRACTICE

Plot five monthly values and identify the highest month.

DATA ANALYSIS MINI PROJECT

From raw CSV to a small report

THEORY

A useful beginner project is a sales report. Start with a CSV containing date, product, quantity and amount. Inspect the data, clean missing values, calculate totals, group by product, find the best-selling category and create one useful chart. The important lesson is the sequence, not the library syntax.

EXAMPLE

```
df = pd.read_csv("sales.csv") print(df.info())
print(df.isna().sum()) product_sales = (
df.groupby("Product")["Amount"] .sum()
.sort_values(ascending=False) ) print(product_sales.head())
```

OUTPUT / RESULT

You now have a repeatable analysis: load ☒ inspect ☒ clean ☒ group ☒ sort ☒ report.

CONCEPT NOTE

Extend the project with a chart and export the summary to a new CSV.

PRACTICE

Build ☒ check ☒ explain

STUDY LOOP: Understand ☒ code ☒ test ☒ explain

AUTOMATION WITH PATHLIB

Make repetitive file work automatic

THEORY

Automation means turning a repeated manual sequence into a program. pathlib gives Python a clean way to work with folders and files. Instead of opening files one by one, a script can discover matching files and process them consistently.

EXAMPLE

```
from pathlib import Path
folder = Path("reports")
for file in folder.glob("*.csv"):
    print(file.name)
```

OUTPUT / RESULT

The loop finds every CSV file in the reports folder.

CONCEPT NOTE

Automation is valuable when the same safe task happens repeatedly.

PRACTICE

List all .txt files in a folder using pathlib.

AUTOMATION • BATCH RENAME

Small scripts can save large amounts of time

THEORY

Suppose a folder contains files with inconsistent names. A batch-renaming script can apply a consistent rule. Before renaming many files, test the naming logic on a small set. Good automation is predictable, reversible when possible and easy to understand.

EXAMPLE

```
from pathlib import Path folder = Path("photos") for i, file
in enumerate(folder.glob("*.jpg"), 1): new_name = folder /
f"photo_{i}.jpg" print(file.name, "->", new_name.name) #
file.rename(new_name)
```

OUTPUT / RESULT

The rename operation is shown but commented out. This lets you test the plan before changing files.

CONCEPT NOTE

Always preview destructive file operations before enabling them.

PRACTICE

Design a safe renaming rule for a folder of documents.

AUTOMATION • CSV REPORT

Generate the same report every time

THEORY

A report script can load a CSV, calculate important metrics and save a clean summary. This is useful for recurring business tasks. Separate data processing from presentation so that the calculations can be tested independently.

EXAMPLE

```
import pandas as pd
df = pd.read_csv("sales.csv")
report = pd.DataFrame({ "total_sales": [df["Amount"].sum()],
    "average_sale": [df["Amount"].mean()], "orders": [len(df)] })
report.to_csv("summary.csv", index=False)
```

OUTPUT / RESULT

summary.csv contains a small machine-readable report.

CONCEPT NOTE

Once the script is reliable, it can become part of a scheduled workflow.

PRACTICE

Add minimum sale and maximum sale to the report.

REGULAR EXPRESSIONS

Find patterns inside text

THEORY

Regular expressions, or regex, describe text patterns. They are useful for tasks such as extracting numbers, validating simple formats and finding repeated structures. Regex can become complex, so begin with small patterns and test them against several examples.

EXAMPLE

```
import re
text = "Order 123 costs 450. Order 124 costs 700."
numbers = re.findall(r"\d+", text)
print(numbers)
```

OUTPUT / RESULT

```
['123', '450', '124', '700']
```

CONCEPT NOTE

The pattern `\d+` means one or more digits.

PRACTICE

Extract all numbers from a sentence and convert them to integers.

APIs • GETTING DATA

Programs can communicate with other programs

THEORY

An API is a structured way for software systems to exchange data. A typical HTTP API provides an endpoint and returns data, often as JSON. In real projects you should read the API documentation, understand authentication and respect rate limits and terms of use.

EXAMPLE

```
import requests
response = requests.get(
    "https://example.com/data" )
print(response.status_code)
print(response.json())
```

OUTPUT / RESULT

A successful response commonly uses a 2xx status code. The exact JSON structure depends on the API.

CONCEPT NOTE

Do not assume every endpoint returns the same fields. Inspect the documented response.

PRACTICE

Read the documentation for a public API and identify its endpoint and response fields.

STUDY LOOP: Request → response → validate

JSON • WORKING WITH API DATA

Convert structured responses into Python data

THEORY

JSON is common in web APIs because it represents objects, arrays, strings, numbers and booleans in a portable format. Python's json module converts JSON text into dictionaries and lists, making API results familiar to Python programmers.

EXAMPLE

```
import json
text = '{"name": "Asha", "score": 91}'
data = json.loads(text)
print(data["name"])
print(data["score"])
```

OUTPUT / RESULT

Asha 91

CONCEPT NOTE

Once JSON becomes a Python dictionary, normal indexing and looping techniques apply.

PRACTICE

Create a JSON string containing a product name and price, then load it.

MACHINE LEARNING • THE BIG IDEA

Prediction from examples

THEORY

Machine learning is a way of building systems that learn patterns from examples. Instead of writing every rule manually, we provide data and a learning algorithm. A model then uses learned relationships to make predictions on new data. Good ML still depends heavily on good data and clear problem definitions.

EXAMPLE

Example: Input features → ML model → prediction House size, bedrooms → model → estimated price The model learns from historical examples.

OUTPUT / RESULT

The key idea is generalization: learn from known examples, then perform well on unseen examples.

CONCEPT NOTE

Think of one problem where a rule-based program is difficult but historical examples are available.

PRACTICE

Problem ☒ data ☒ model ☒ prediction

STUDY LOOP: Understand ☒ code ☒ test ☒ explain

FEATURES & LABELS

Know what the model sees and predicts

THEORY

Features are the input information used by a model. The label, also called the target, is what we want to predict. For a house-price problem, size and bedrooms may be features while price is the label. Clearly separating these prevents confusion when building the training dataset.

EXAMPLE

```
X = df[["Size", "Bedrooms"]] y = df["Price"] # X = features #  
y = target / label
```

OUTPUT / RESULT

X contains the information given to the model. y contains the answer the model should learn to predict.

CONCEPT NOTE

Choose features that are available at prediction time. Do not accidentally include the answer itself.

PRACTICE

Define X and y for a student-score prediction problem.

STUDY LOOP: Inputs → target → check

TRAIN & TEST DATA

Measure performance on unseen examples

THEORY

If a model is tested on the same examples it learned from, the score can be misleading. A train/test split keeps some data aside for evaluation. The test set acts like new data. This simple idea is fundamental to trustworthy machine learning.

EXAMPLE

```
from sklearn.model_selection import train_test_split X_train,
X_test, y_train, y_test = train_test_split( X, y,
test_size=0.2, random_state=42 ) print(len(X_train),
len(X_test))
```

OUTPUT / RESULT

Approximately 80% of the examples are used for training and 20% are held out for testing.

CONCEPT NOTE

random_state makes the split reproducible.

PRACTICE

Explain why evaluating on training data can give an overly optimistic result.

SCIKIT-LEARN WORKFLOW

A standard recipe for many ML tasks

THEORY

Scikit-learn provides consistent interfaces for many classical machine-learning algorithms. A common workflow is: prepare X and y, split the data, create a model, fit it on training data, predict on test data and evaluate the predictions. Keeping these steps separate makes debugging easier.

EXAMPLE

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(predictions[:5])
```

OUTPUT / RESULT

`fit()` learns parameters from training data. `predict()` produces outputs for new feature rows.

CONCEPT NOTE

Different algorithms can often be used with a similar fit/predict workflow.

PRACTICE

Write the five main steps of an ML workflow from memory.

STUDY LOOP: Prepare → fit → predict → evaluate

LINEAR REGRESSION

Predict a numeric value

THEORY

Linear regression models a relationship between input features and a numeric target. A simple model tries to find a line that best represents the training examples. It is useful for learning the ML workflow even when real-world relationships are more complicated.

EXAMPLE

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
prediction = model.predict([[1500, 3]])
print(prediction)
```

OUTPUT / RESULT

The output is a predicted numeric value. It is an estimate, not a guaranteed truth.

CONCEPT NOTE

Predictions should be interpreted with the model's evaluation results and the limits of the training data.

PRACTICE

Imagine predicting rent from area and number of rooms. Identify the features and target.

STUDY LOOP: Fit → predict → interpret

CLASSIFICATION

Predict a category

THEORY

Classification predicts a class rather than a continuous number. Examples include spam/not spam, pass/fail or customer categories. The classes should be defined clearly, and evaluation should consider whether mistakes in one class are more costly than mistakes in another.

EXAMPLE

```
from sklearn.tree import DecisionTreeClassifier model =  
DecisionTreeClassifier(random_state=42) model.fit(X_train,  
y_train) pred = model.predict(X_test) print(pred[:5])
```

OUTPUT / RESULT

The predictions are class labels such as: 0, 1 or "pass", "fail".

CONCEPT NOTE

Classification answers: which category is this example most likely to belong to?

PRACTICE

Create a fictional binary classification problem and name its features and labels.

EVALUATION METRICS

A model is not useful just because it runs

THEORY

Evaluation tells us how well predictions match reality. For regression, metrics such as mean absolute error describe prediction error. For classification, accuracy, precision, recall and F1 score answer different questions. Choose metrics based on the cost of mistakes, not only the easiest number to calculate.

EXAMPLE

```
from sklearn.metrics import mean_absolute_error
error = mean_absolute_error(y_test, predictions)
print("MAE:", error)
```

OUTPUT / RESULT

MAE is the average absolute difference between actual and predicted numeric values.

CONCEPT NOTE

A lower MAE generally means predictions are closer to the actual values.

PRACTICE

Explain why accuracy alone can be misleading for a highly imbalanced classification problem.

STUDY LOOP: Predict ✕ compare ✕ interpret

ML PIPELINES

Make preprocessing and modeling reproducible

THEORY

Real datasets often need preprocessing before modeling: scaling numbers, encoding categories or filling missing values. A pipeline connects preprocessing steps with the model so the same transformations are applied consistently. This reduces manual mistakes and helps prevent data leakage.

EXAMPLE

```
from sklearn.pipeline import Pipeline from
sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LogisticRegression pipe =
Pipeline([ ("scale", StandardScaler()), ("model",
LogisticRegression()) ]) pipe.fit(X_train, y_train) pred =
pipe.predict(X_test)
```

OUTPUT / RESULT

The pipeline treats preprocessing and prediction as one reusable workflow.

CONCEPT NOTE

Think of a pipeline as an assembly line: each step receives data, transforms it and passes it forward.

PRACTICE

Draw the steps of a preprocessing + model pipeline for a dataset.

STUDY LOOP: Transform → model → predict

AI / ML PROJECT STRUCTURE

Turn experiments into maintainable projects

THEORY

A real AI/ML project needs more than a notebook. Keep raw data separate from processed data, put reusable code into modules, record dependencies and document how to run the project. Save model assumptions and evaluation results. A clean structure makes experiments repeatable and easier for another person to understand.

EXAMPLE

```
project/ ├── data/ ├── notebooks/ ├── src/ ├── models/ ├──  
reports/ ├── requirements.txt └── README.md
```

OUTPUT / RESULT

Typical flow: data → cleaning → features → training → evaluation → model → report

CONCEPT NOTE

A project structure is a communication tool: it tells future-you where each responsibility belongs.

PRACTICE

Design a folder structure for a customer-churn prediction project.

STUDY LOOP: Organize → document → reproduce

FINAL CAPSTONE • SMART SALES ANALYZER

Bring the whole book together

THEORY

Build a final project that loads sales data, cleans it, summarizes products, creates a chart and optionally predicts a numeric business measure. Use functions for reusable logic, pathlib for file handling, pandas for tables, matplotlib for visualization and scikit-learn only when prediction adds value. Add error handling and a short README.

EXAMPLE

Suggested pipeline: CSV files ↓ Load with pandas ↓ Clean + validate ↓ Analyze + group ↓ Visualize ↓ Optional ML model ↓ Save report The result should be a repeatable command, not a one-time manual analysis.

OUTPUT / RESULT

Final deliverables: • cleaned dataset • summary CSV • one useful chart • optional trained model • README explaining the workflow

CONCEPT NOTE

Do not try to make the project huge. Make one complete workflow reliable.

PRACTICE

Build the project in small milestones and test each stage before adding the next.

STUDY LOOP: Build ☒ test ☒ explain ☒ improve

BOOK REVIEW • FROM PYTHON TO AI

You now have a complete learning path

THEORY

Part 1 built the language foundations. Part 2 strengthened functions and collections. Part 3 introduced object-oriented design. Part 4 moved into practical Python: files, exceptions, generators, decorators, JSON, testing and automation. Part 5 connected those skills to data analysis, automation and machine learning.

EXAMPLE

The learning ladder: Python syntax ↓ Problem solving ↓ Reusable functions + OOP ↓ Files + automation ↓ Data analysis ↓ Machine learning ↓ Your own projects

OUTPUT / RESULT

The most important lesson: Do not measure progress by how many Python commands you remember. Measure it by whether you can take a problem, break it into steps, write code, test it, and explain the result.

CONCEPT NOTE

Choose one project and keep improving it. Your next level comes from building, debugging and explaining—not from collecting more syntax.

PRACTICE

pythonforaiml.com • Keep learning by building.

STUDY LOOP: Understand ☒ code ☒ test ☒ explain