

Prepare, Explore and Split Your Data

Lesson 1 - Python for data science with pandas, matplotlib and scikit-learn.

~75 MIN · Python · pandas · matplotlib · scikit-learn · PyCharm or Google Colab

Scenario. You are a junior data analyst for a youth coding-school programme. The team collected data about participants: their **age**, their **interest** in the programme (score from 0 to 100) and whether they were a **success** (1) or not (0). Before anyone can build a model to predict success, you must **clean**, **explore** and **split** the data.

OBJECTIVES

By the end of this lesson you will be able to:

- Load a CSV file with **pandas** and inspect it.
- Clean data: round values and handle **missing values**.
- Compute **descriptive statistics** and draw histograms.
- Create a new column, group data and compare groups with **bar charts**.
- Explore relationships with **scatter plots**.
- Split data into a **train set** and a **test set**, then save them.

BEFORE YOU START

- Python 3 with **PyCharm** (or a **Google Colab** notebook). Install once: **pip install pandas numpy matplotlib scikit-learn**.
- Dataset from your trainer: **classification.csv** (columns: age, interest, success). Put it in the same folder as your script (in Colab: upload it in the Files panel).

Ground rules: write the code yourself (no copy-paste). Work in order, one task at a time. **Print** every result you are asked for, add a short **comment** (#) above each block, and give every chart a **title** and **axis labels**. The file uses a semicolon (;) as separator.

YOUR TASKS

1 Set up and load the data

Where: Your Python file or notebook, first block.

1. Import **pandas** as **pd**, **numpy** as **np** and **matplotlib.pyplot** as **plt**.
2. Read **classification.csv** into a variable called **data** (mind the separator).
3. Display the first 30 rows, then print how many **rows** and how many **attributes** (columns) there are.

HINTS

```
pd.read_csv(..., sep=...) · data.head(n) · len(data) · data.columns
```

Checkpoint: You can state the number of rows and columns, and explain what each column means.

2 Clean the data: ages and missing values

Where: Your Python file or notebook.

1. Round the values of the **age** column to whole numbers and store them back in the same column.
2. For **age**, **interest** and **success**, print the number of **missing values** in each column.
3. Fill the missing values of **interest** with the **mean** of that column, then check that none is left.

HINTS

```
round(...) · isnull().sum() · fillna(...) · mean()
```

Checkpoint: Ages are whole numbers, you know which column had empty cells, and your final check prints 0.

3 Descriptive statistics and histograms

Where: Your Python file or notebook.

1. For **age** and for **interest**, print: number of elements, maximum, minimum, mean, median and **standard deviation**. Draw one **histogram** for each column.
2. For **success**, print only the number of 0s and the number of 1s, then draw its histogram.

HINTS

```
max() · min() · median() · std() · plt.hist(...) · plt.xlabel / ylabel / title
```

Tip: `std()` gives the standard deviation, not the variance. The variance is `var()` (the standard deviation squared).

Checkpoint: You can describe the shape of the age histogram in one sentence (young, old, evenly spread?).

4 Age groups and success

Where: Your Python file or notebook.

1. Create a new column **age_groups**: **1** = age up to 18, **2** = 19 to 35, **3** = over 35.
2. Count the people for each combination of **age_groups** and **success**.
3. Draw a **bar chart** of the **number** of successful people per group (kids, mid-ages, seniors).
4. Draw a second bar chart with the **ratio**: successful people divided by the total number of people in the group.

HINTS

```
for / if / elif · data['age_groups'] = ... · value_counts([...]) · plt.bar(...) · plt.ylim(0) · list.count(...)
```

Checkpoint: You can explain why the ratio chart is fairer than the count chart.

5 Scatter plots: age and interest

Where: Your Python file or notebook.

1. Sort the data by **age**, then draw a **scatter plot** with age on the X axis and interest on the Y axis.
2. Draw it again, this time **coloured by success**.

HINTS

```
sort_values('age') · plt.scatter(x, y, c=...)
```

Checkpoint: In one sentence: who tends to succeed - higher interest, a certain age, or both?

6 Train set, test set and saving

Where: Your Python file or notebook.

1. Import **train_test_split** from **sklearn.model_selection**.
2. Split **data** into a **train set** (70%) and a **test set** (30%). Use **random_state = 42** so everyone gets the same split. Print the size of each set.
3. Save both sets as **train_data_success.csv** and **test_data_success.csv**.

HINTS

```
train_test_split(data, test_size=..., random_state=...) · to_csv('file.csv')
```

Checkpoint: The two sizes add up to the total number of rows, and both CSV files exist in your folder.

FINAL CHECKLIST

- Your code runs **from top to bottom without errors**, with a comment above each block.
- Every chart has a **title**, labelled axes, and you can interpret it in one sentence.
- **Bonus:** repeat Task 2 filling with the **median** instead of the mean. Does any statistic change?

REFLECT

Why do we keep a test set aside before training any model? Which data-cleaning choice (for example filling missing values with the mean) could change your conclusions?