Lecture 17 | Matplotlib: exercises

Guided demonstration, two laboratories and three independent problems

ISEG Lisbon School of Economics & Management Universidade de Lisboa

Guided 01 | Loan repayment trajectories

- A credit analyst has already calculated the remaining loan balance and cumulative interest for months 0–8. The figures have been saved as `loan_history.csv`
- **Task**: draw both monetary quantities on the SAME plot, so a reader can follow their different evolution throughout the loan
- The aim is Matplotlib, not calculating the amortization schedule. The starter supplies file reading, the three Series and the Figure creation
- Produce `loan_trajectories.png`. Use a line for each sequence, visible markers, a title, axis labels with units, a light grid and a legend.

Month	Balance	Cumul. interest
0	1200	0
1	1050	18
2	900	34
…	…	…
8	0	88

Input: loan_history.csv | Output: loan_trajectories.png

Guided 01 | Every plotting step

- Identify coordinates: `Month` is x; `Balance_EUR` and `CumulativeInterest_EUR` are two alternative y sequences
- Call `ax.plot(months, balances, ...)` with `marker="o"`, `linewidth=2` and a descriptive label
- Call `ax.plot()` again ON THE SAME ax using interest, `marker="s"` and `linestyle="--"`. This overlays a second curve
- Add title, x/y labels and integer ticks with `ax.set_xticks(...)`. Then call `ax.grid(...)`, `ax.legend()` and `fig.savefig(...)`
- Check: the balance falls to zero, interest increases, and the legend distinguishes the two curves

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(months, balances,
        marker='o', label='...')
ax.plot(months, interest,
        linestyle='--', label='...')
ax.set_xlabel('Month')
ax.legend()
fig.savefig('loan_trajectories.png')
```

Guided 01 | Interpreting the result

- The CSV has one observation per month. There is no need to reshape or aggregate it before drawing.
- Each plotted line must contain nine points, at months 0, 1, ..., 8.
- At month 4 the two quantities are 600 EUR and 61 EUR. Their separation is expected: the curves have different meanings but the same unit.
- Two `ax.plot(...)` calls add content to one Axes. Creating two independent figures would not satisfy the requested single comparison.
- No `plt.show()` is needed in an automated script: saving the image is sufficient.

Month	Balance EUR	Interest EUR
0	1200	0
4	600	61
8	0	88

Lab 01 | Community centre budget

- The manager of a community centre records individual expenses, each with a category and amount. Several records belong to the same category
- Produce a ranked horizontal bar chart showing TOTAL euros spent in each category, rather than one bar for every original expense
- Input: `community_expenses.csv` with columns Category and `Amount_EUR`. The starter already loads and prints the data
- Deliver `category_spending.png` and `category_totals.csv`, with columns Category and `Total_EUR`.

Use .groupby(), .sum(), .sort_values(), .barh() and .to_csv()

Category	Amount EUR
Workshops	120
Utilities	180
Sports	65
Workshops	95
…	…

Lab 01 | From records to categories

1. First calculate the total expense for each category with

   ```
   .groupby('Category', as_index=False).agg(Total_EUR=('Amount_EUR', 'sum'))
   ```

   This produces a DataFrame with Category and `Total_EUR` columns

2. Sort by `Total_EUR` in ascending order using `.sort_values('Total_EUR')`; the largest horizontal bar will appear at the top

3. Pass the sorted Category and `Total_EUR` columns to `ax.barh()`. Add a label identifying euros on the numerical x-axis

4. Export the resulting two-column DataFrame with `.to_csv('category_totals.csv', index=False)`; no further reshaping is necessary

5. Check the totals: Workshops = 325, Utilities = 340, Sports = 208, Supplies = 106 EUR.

Lab 01 | Requirements and checks

- The chart must have exactly FOUR bars even though the source file has ten records
- Each category name should be displayed clearly beside its bar; the horizontal axis should state that values are EUR
- Use the two columns of the SAME sorted DataFrame for category labels and bar lengths so the correspondence remains correct
- Give the plot a title and a light grid on the numerical axis. Do not export the ten unaggregated source rows
- A correct output CSV has one row per category and totals summing to 979 EUR.

Category	Total EUR
Supplies	106
Sports	208
Workshops	325
Utilities	340

Lab 02 | Service waiting times

- A municipal counter records the **waiting time**, in minutes, for 30 customers. Management wants to understand the **frequency distribution** (how many times does a) rather than the order in which customers arrived
- Construct a histogram from **waiting_times.csv** and mark its average waiting time with a vertical reference line, using the simple `.axvline( )` method explained in the next slide
- Use fixed bin boundaries 0, 10, 20, 30, 40 and 50 minutes so the intervals have a clear interpretation
- Output: `waiting_distribution.png`. The starter supplies file reading, the average, Figure creation and saving.

Sample recorded times (min)
3, 7, 10, 11, 12, 13
15, 15, 16, 17, 18, 19
20, 20, 21, 22, 23, 23
... up to 49

Plotting focus: .hist(), bins=, .axvline(), label=, .legend()

Lab 02 | Histogram construction

- The values form ONE numeric Series. Do not calculate frequencies manually: `.hist()` counts observations in each bin.
- Pass `bins=[0, 10, 20, 30, 40, 50]`. Matplotlib assigns values to intervals [0, 10), [10, 20), etc.; the last includes the right endpoint.
- Use `edgecolor="white"` so neighboring histogram bars remain distinguishable.
- The method `.axvline(x, ...)` draws a vertical line at horizontal coordinate x. Use `x=average` to mark the mean; `color=` controls its color, `linestyle="--"` makes it dashed, and `label=` gives it a legend name
- Add the title, axis names and ticks at the bin boundaries; the vertical axis must indicate NUMBER of customers.

```
ax.hist(values, bins=[0,10,20,
                      30,40,50],
        edgecolor='white')

ax.axvline(average,
           linestyle='--',
           label='Average')

ax.legend()
```

Lab 02 | Checks before submitting

- The histogram must have five bins, even if one interval contains relatively few observations.
- A waiting time of exactly 20 minutes belongs to [20, 30), not [10, 20).
- The sum of the five bar heights must equal 30, the number of rows in the CSV.
- The reference line must be at the mean computed from the recorded times, not at a manually chosen position.
- Do not connect observations with a line: the goal is a distribution of waiting times.

The histogram shows frequencies; the reference line identifies a statistic on the same x scale

Independent 01 | Weather and electricity demand

- An energy analyst has paired observations of daily average temperature (°C) and electricity demand (MWh). The data are in `weather_demand.csv`.
- The analyst wants to inspect the relationship and distinguish high-demand days (demand ≥ 120 MWh) from all other days.
- Construct a SCATTER plot with temperature on the horizontal axis and demand on the vertical axis.
- Use distinct markers or colors for the two groups, draw a horizontal reference line at 120 MWh and include a legend.
- Save `temperature_demand.png` and print the number of high-demand observations.

Temperature C	Demand MWh
12	114
14	109
15	105
...	...
30	149

Independent 01 | Requirements and hints

- Build the two categories from the original records, using a for loop and an if statement rather than assuming that the high-demand points occur together.
- The classification needs to keep corresponding temperatures and demands together. One approach is to maintain two lists for the x/y coordinates of each group.
- Call `.scatter()` twice on the same Axes and add `.legend()`. To mark the threshold, `.axhline(y=120, linestyle="--", ...)` draws a horizontal line at y = 120; `color=` sets its color and `label=` names it in the legend
- Explain the plotted information through units, a title and a count printed to the terminal.
- Expected check: the high-demand group contains 3 observations; every source row must appear as exactly one point.

Independent tasks have no starter script; input data and a complete solution are provided separately

Independent 02 | Training attendance heatmap

- A training organization records the number of participants in each Course on each Day. Some course–day combinations appear more than once, because multiple sessions were offered
- Produce a single heatmap with courses as rows and weekdays as columns, summarizing the TOTAL participants for each combination
- Input: `training_attendance.csv` (Course, Day, Participants). Output: `attendance_heatmap.png` and `attendance_wide.csv`.
- The colors and annotations must correspond to the aggregated numeric table rather than to the original list of records.

Course	Day	Participants
A	Mon	9
B	Tue	7
B	Tue	9
C	Wed	9

Independent 02 | Exact reporting requirements

- Read the CSV and create a wide DataFrame using `.pivot_table(index="Course", columns="Day", values="Participants", aggfunc="sum")`
- Display the days in this order: `Mon, Tue, Wed`. The heatmap must show a numeric annotation in every cell and label both axes
- Use `sns.heatmap(..., annot=True, fmt=".0f", ...)` and configure the plot using the Matplotlib Axes
- Save the resulting wide DataFrame as CSV, preserving the Course index labels so that the saved rows can be interpreted
- Check: Course B, Tue equals 16 (7 + 9). Averages would be incorrect for this task.

Course	Mon	Tue	Wed
A	9	7	8
B	6	16	8
C	5	7	9

Independent 03 | Café dashboard

- A small café records **units sold by day and product** in sales.csv. The separate `products.csv` file contains the category and unit price of each product.
- Create a compact visual report that compares daily revenue with revenue by product category. Both measures must be calculated from the two input files.
- Produce `cafe_dashboard.png` containing TWO panels placed horizontally: a line chart for revenue by day and a bar chart for category revenue.
- Also export `sales_enriched.csv`, with the original sales plus product information and calculated Revenue_EUR.

sales.csv	products.csv
Day,Product,Quantity	Product,Category,UnitPrice_EUR
Mon,Tea,12	Tea,Drinks,2.0
Mon,Coffee,18	Coffee,Drinks,2.5
…	…

Independent 03 | Required processing

- Read both CSV files and combine the rows using the common Product key. Every sales row should receive its Category and `UnitPrice_EUR`.
- Define a simple function `calculate_revenue(quantity, unit_price)` and use it when creating the `Revenue_EUR` column.
- Calculate total revenue for each day and Category. For Mon → Thu, use an explicitly ordered list of day labels to select the corresponding Series values; do not rely on alphabetical sorting
- Use `fig, axes = plt.subplots(1, 2, figsize=(10, 4))`; the first Axes must contain the daily line and the second the category bars.
- Add different titles, correct axis labels with EUR units, and save the complete Figure. Export the enriched DataFrame to a CSV.

One input record is one product sold on one day; revenue = quantity × unit price

Independent 03 | Output and verification

- The two panels must use different representations of the SAME total revenue data: day totals on the left, category totals on the right.
- The four daily totals must be in chronological weekday order, not alphabetical order.
- The overall revenue summed by day must equal the overall revenue summed by category.
- Verify the output file contains 16 sales records and new columns `Category, UnitPrice_EUR` and `Revenue_EUR`.
- Keep the program focused: no interactive menu, advanced data-cleaning framework or extra plotting library is required.

Panel 1	Panel 2
Line: days Mon–Thu	Bars: Food vs Drinks
x = day	x = category
y = revenue EUR	y = revenue EUR

Quick reference | What each exercise practices

- Guided 01: `.plot()` twice on one Axes, labels, legend, saving a PNG
- Lab 01: `.groupby().sum()` + sorting → `.barh()` + summary CSV
- Lab 02: numeric observations → `.hist()` + reference line
- Independent 01: loops and conditionals → two `.scatter()` calls
- Independent 02: `.pivot_table()` → `sns.heatmap()` + wide CSV
- Independent 03: CSV input + function + `.merge()` + `.groupby()` → two subplots
- For all problems: the code must operate on the supplied data, not on manually copied output values.

Consult the Lecture 17 plotting examples and the course notes on pandas and Python basics