Data science

matplotlib

Chapter 4 – Matplotlib

2025-2026

Overview

1. Matplotlib object hierarchy
2. Line chart
3. Subplots
4. Bar chart
5. Histogram
6. Scatter plot
7. Pie chart

Open the corresponding notebook for more detailed information!

Matplotlib

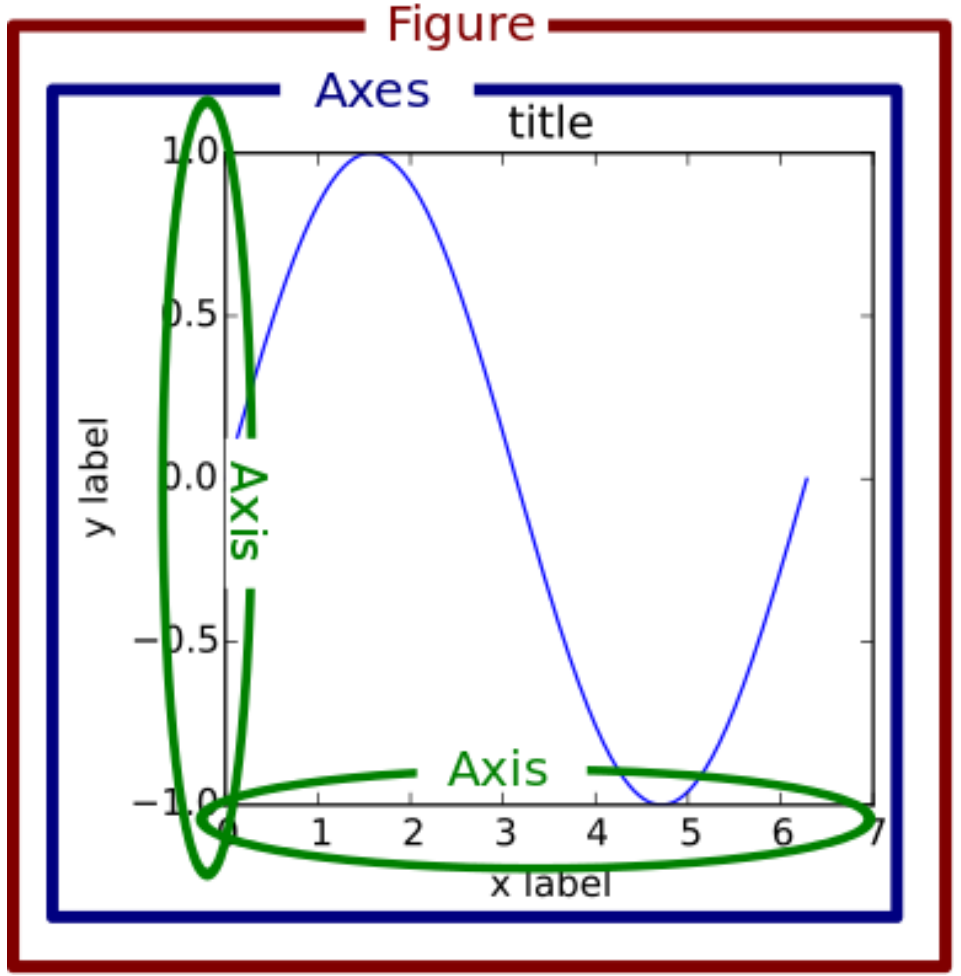
- Matplotlib is a Python library used for data visualization.
- The library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, scatter, pie, …
- In this course just basic plotting is explained. Best practices in the world of plotting are covered in the ‘Data visualization’ course in the 2nd term.
- Most Matplotlib utilities lie under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

- Install the package first if it is not available!

Matplotlib object hierarchy

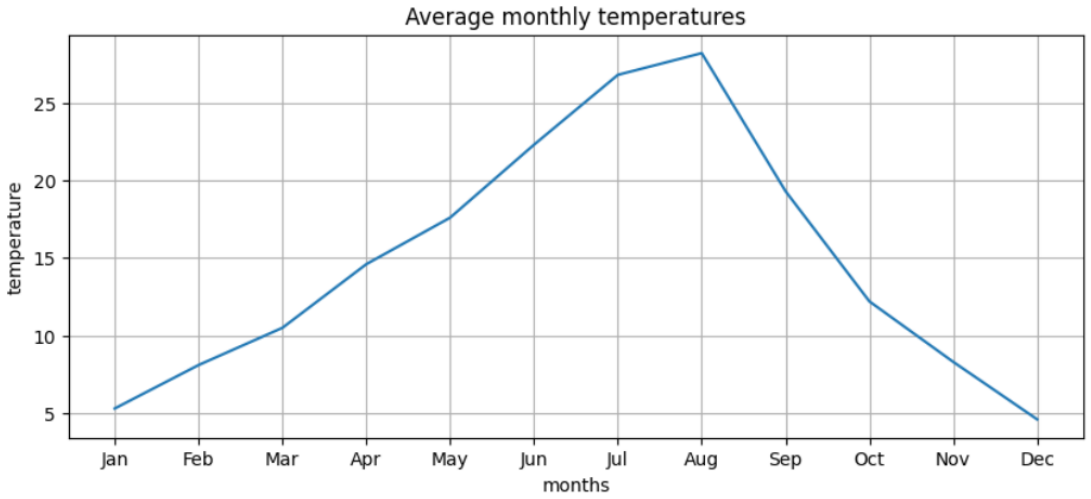
- Matplotlib's plotting area is called the Figure object.
- This Figure can be seen as a grid containing 1 or more Axes objects.
- Each Axes is an individual plot. (NOT the plural of "axis")
- Using Pyplot functions smaller objects such as tick marks, individual lines, legends, text boxes, … can be added to the current Axes in the current Figure.



plt.figure() & plt.plot()

```
average_monthly_temperatures_year1 = np.array([5.3, 8.1, 10.5, 14.6, …,4.6])
months=np.array(['Jan','Feb','Mar','Apr',…,'Dec'])
fig = plt.figure(figsize=(10,4)) #width=10, height=4
plt.plot(months,average_monthly_temperatures_year1)
plt.title("Average monthly temperatures")
plt.xlabel("months")
plt.ylabel("temperature")
plt.grid(True)
plt.show()
```

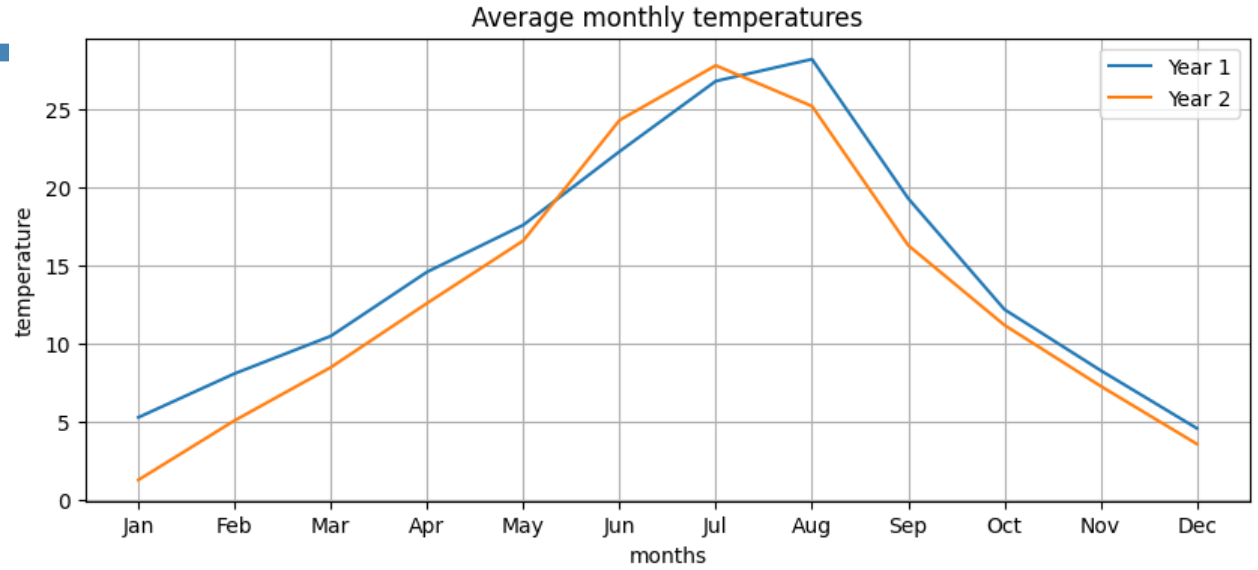
- figure() creates Figure
- plot() creates Axes (line chart) on current Figure or creates new Figure if none exists
- Pyplot functions that manipulate the current Axes on the current Figure



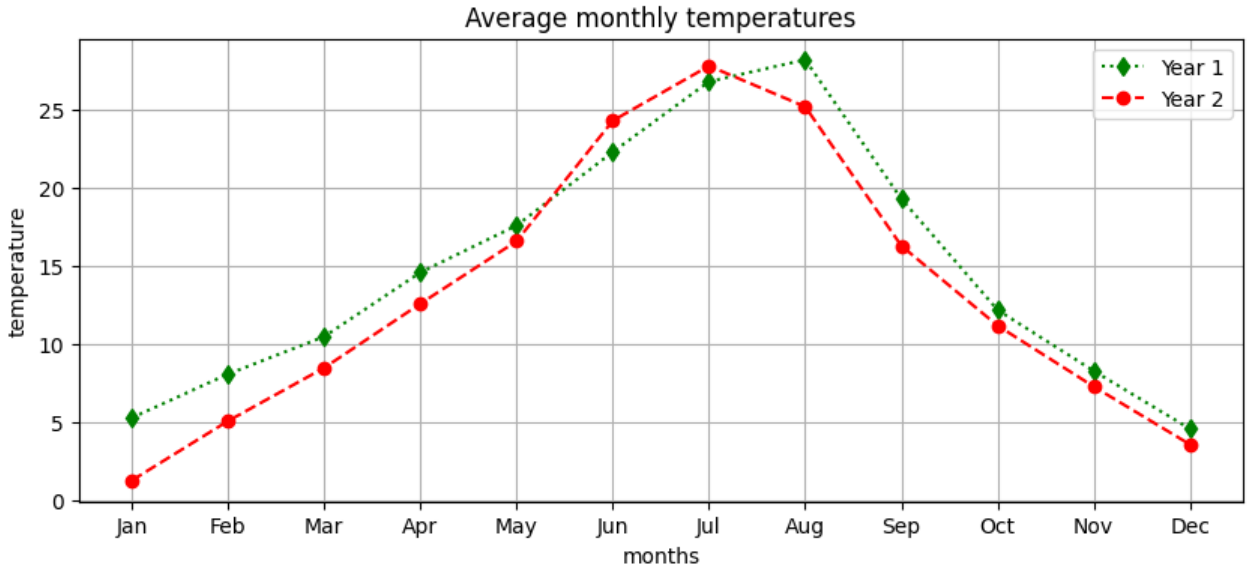
1 Figure object
1 Axes object
1 X-Axis, 1 Y-Axis
1 X-Axis label, 1 Y-axis, label
1 plot title

Plots with extra toppings

- Added a second set of data
- Added a legend



- Changed the line style



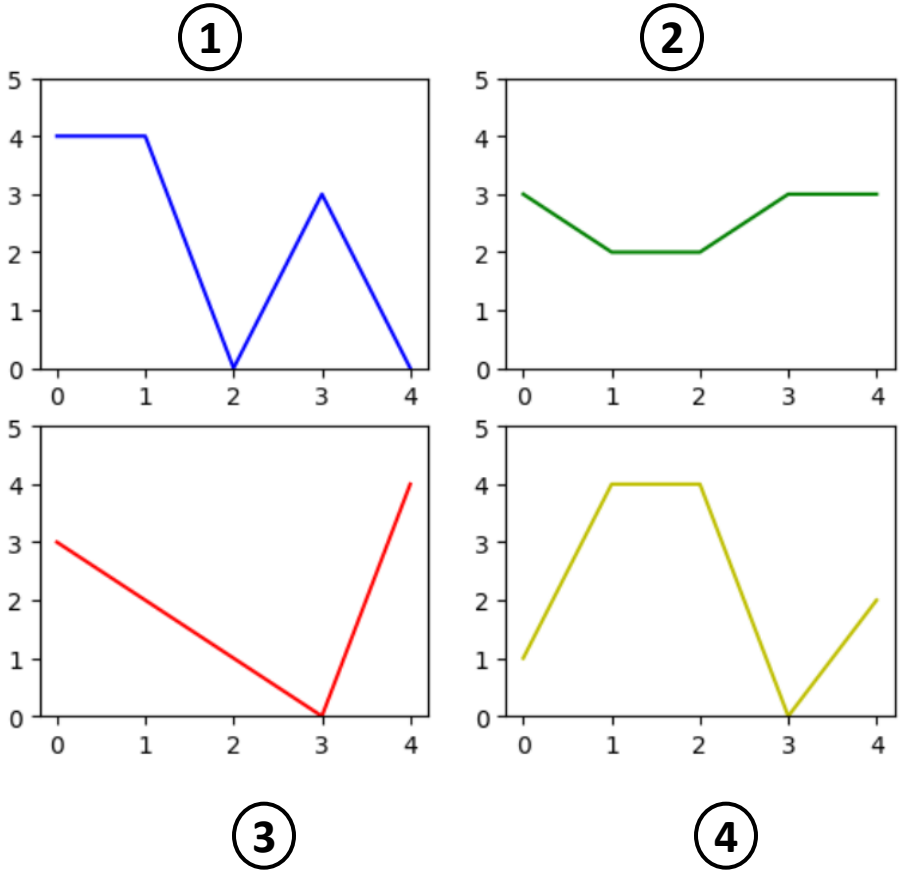
Line chart

- plot() draws a line chart as a series of points which are joined with straight lines
- Using a line chart implies continuous data. For every value of x, a corresponding value of y exists.
- Mostly line charts are used to display how data changes continuously over periods of time.

subplot()

- subplot() splits the Figure in a 2D grid.
- Each Axes gets its own number

```
plt.subplot(2,2,1)
plt.ylim(0,5)
plt.plot(a,b,'b')

plt.subplot(2,2,2)
plt.ylim(0,5)
plt.plot(a,d,'g')

plt.subplot(2,2,3)
plt.ylim(0,5)
plt.plot(a,c,'r')

plt.subplot(2,2,4)
plt.ylim(0,5)
plt.plot(a,e,'y')
```

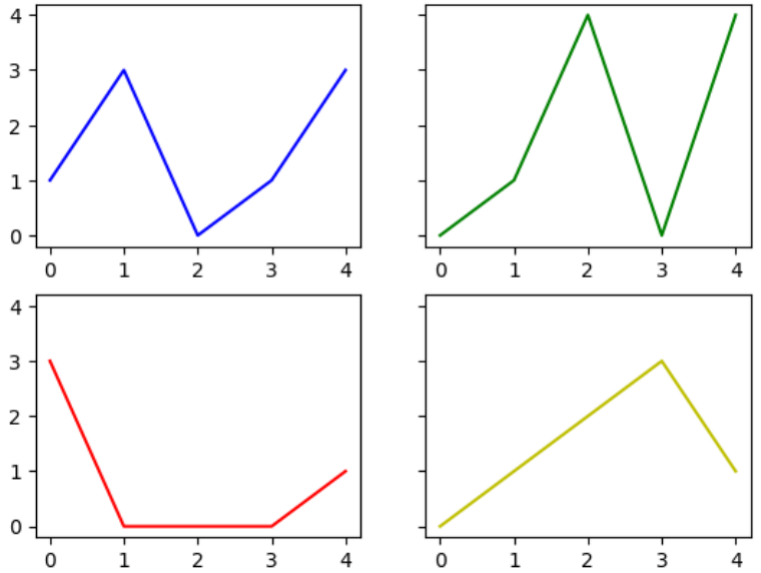


Best practice data visualization:
Y-axis of each plot has the same scaling!

subplots()

- A shortcut to create a grid for subplotting is to define the Figure as a NumPy object array which supports indexing.
- Note the attributes sharex and sharey to make sure that the plots in the grid use the same axis.

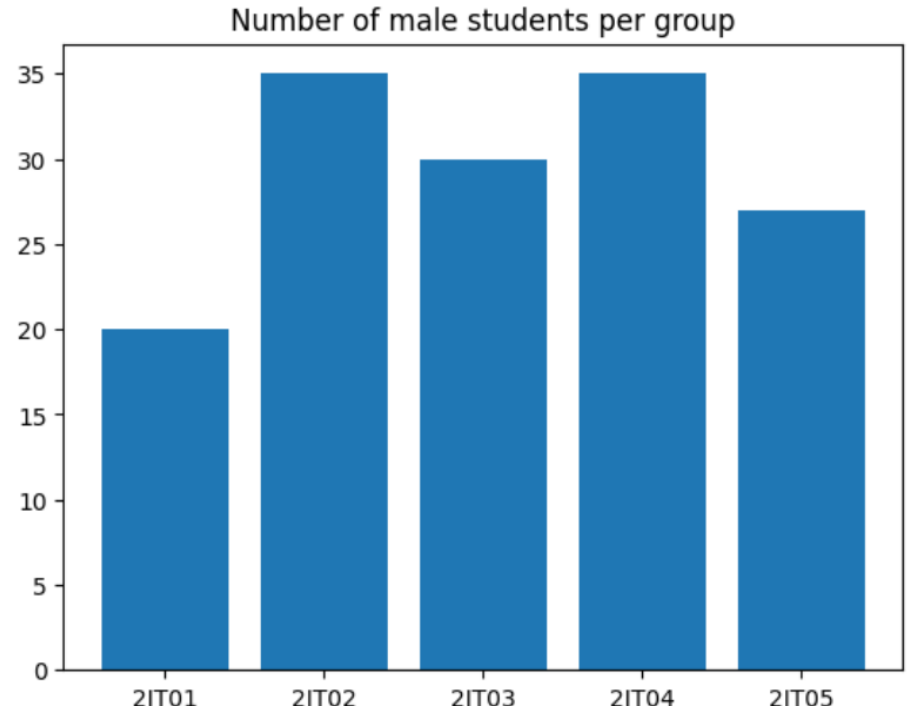
```
fig, axs = plt.subplots(2,2, sharey=True)
…

axs[0,0].plot(a,b,'b')
axs[0,1].plot(a,d,'g')
axs[1,0].plot(a,c,'r')
axs[1,1].plot(a,e,'y')
```



Bar chart

- Used for representing categorical, discrete data

```
groups = np.array((['2IT01', '2IT02', '2IT03', '2IT04', '2IT05']))
number_of_men = np.array ([20, 35, 30, 35, 27])
plt.bar(groups,number_of_men)
plt.title('Number of male students per group')
plt.show()
```

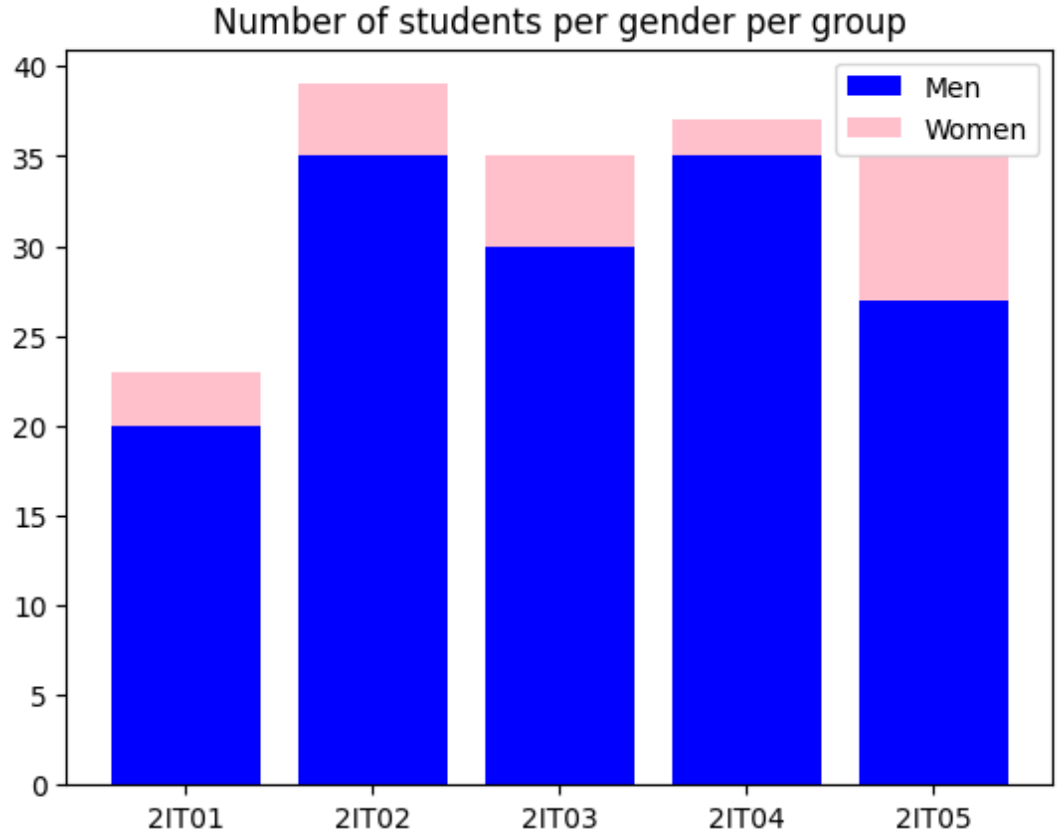


- A line chart cannot be used because a line suggests that an individual student can be between 2 class groups

Bar chart with toppings

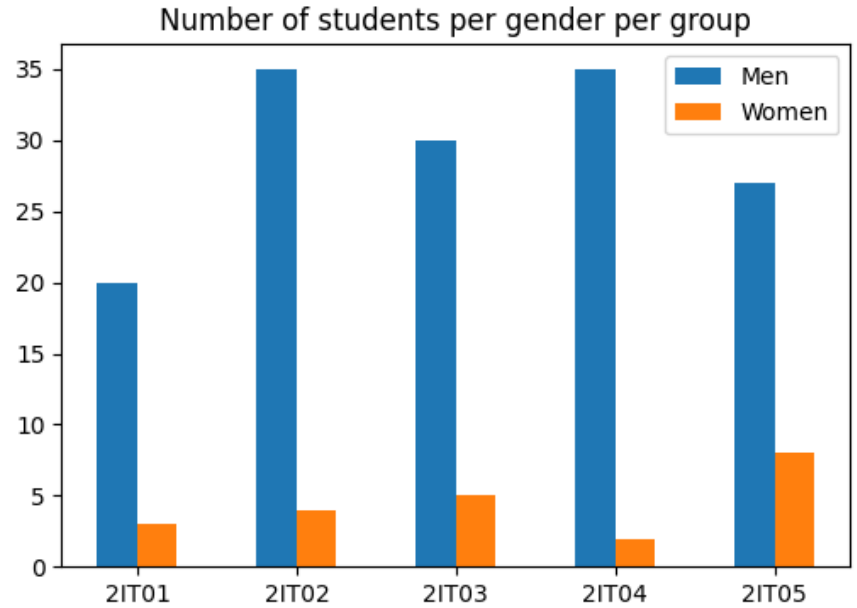
- Stacked bar chart: gender is added
- Each gender has it’s colour
- Legend is added

You can find the code in the notebook!



Bar chart with a lot of toppings

- Plotting the bars next to each other is more difficult.
  - You can find the code in the notebook
- In a practical setting you could consider seaborn in this case, a different plotting library that would make the code easier.

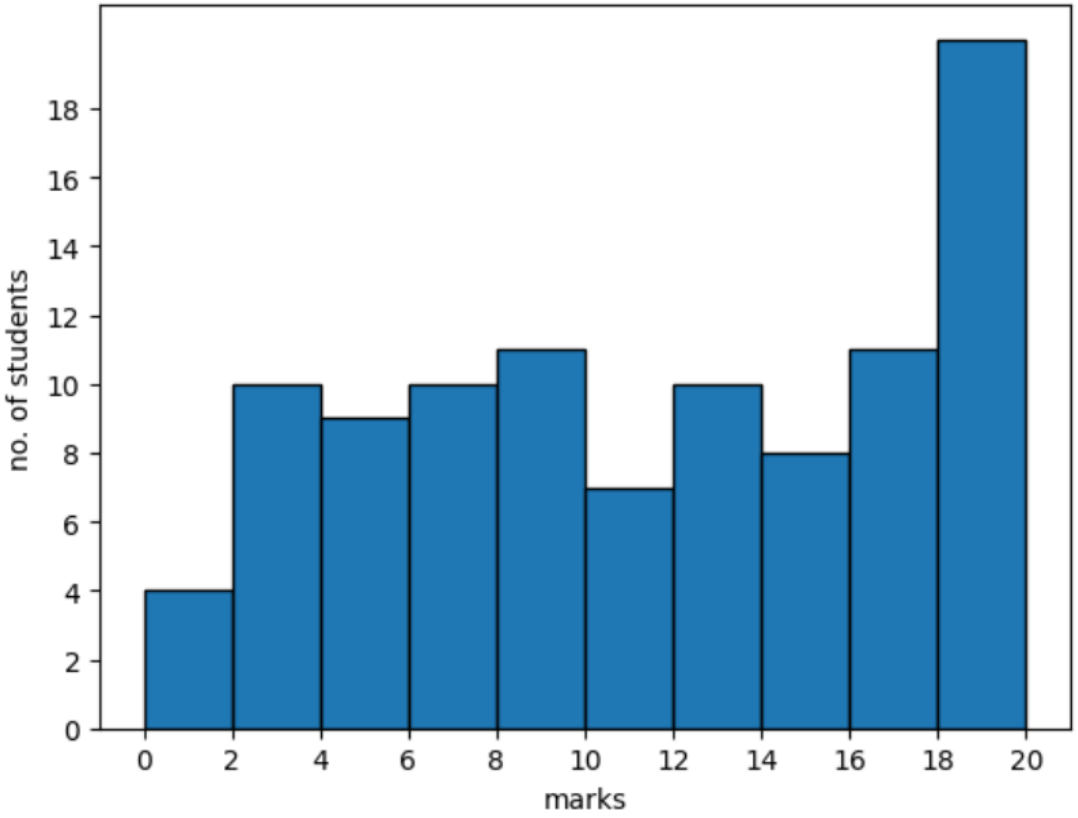


Histogram

- A representation of the distribution of continuous numerical data
- To construct a histogram, we divide the entire range of values into a series of intervals (bins).
- These intervals are plotted on the X-axis.
- Then we count how many values fall into each interval. This number is shown on the Y-axis.
- The result is a kind of bar graph that shows the number of times that the values occurred within each interval.

Histogram

```
#result of 100 students for exam data science
results = np.random.randint(1,21,(100,1))
#bins
intervals = [0,2,4,6,8,10,12,14,16,18,20]
#ec = edgecolor --> creates separate bars
plt.hist(results, bins=intervals, ec='black')
plt.xticks(intervals)
plt.yticks(np.arange(0,20,step=2))
plt.xlabel('marks')
plt.ylabel('no. of students')
plt.show()
```



Histograms show continuous data.
Every mark between 0 and 20 is possible.
There are no gaps between the bars.

Scatter plot

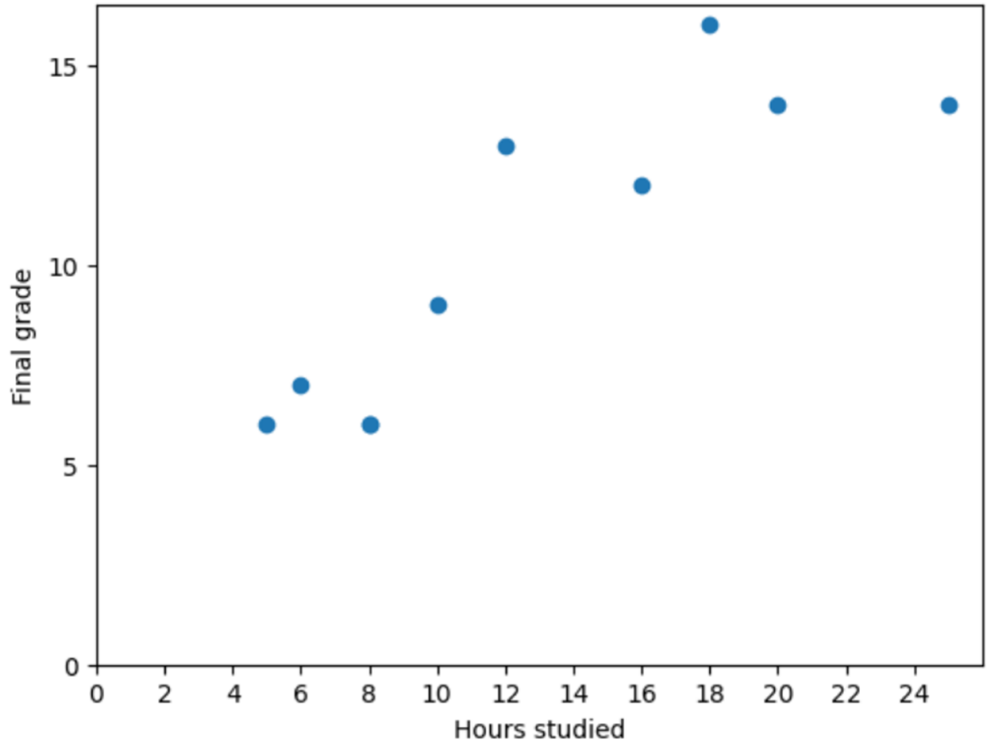
- Used to show whether or not there is a relationship or connection between 2 sets of data.
- The data is plotted on a graph such that one quantity is plotted on the x-axis and one quantity is plotted on the y-axis.
- Each observation (= each row in the data table) is represented by a marker.

Scatter plot

```
hours_studied = np.array([16, 6, 20, 8, 25, 12, 10, 8, 5, 18])
grades = np.array([12, 7, 14, 6, 14, 13, 9, 6, 6, 16])
plt.scatter ( x=hours_studied,y=grades)
plt.xticks(np.arange(0,26,step=2))
plt.yticks(np.arange(0,20,step=5))
plt.xlabel('Hours studied')
plt.ylabel('Final grade')
plt.show()
```

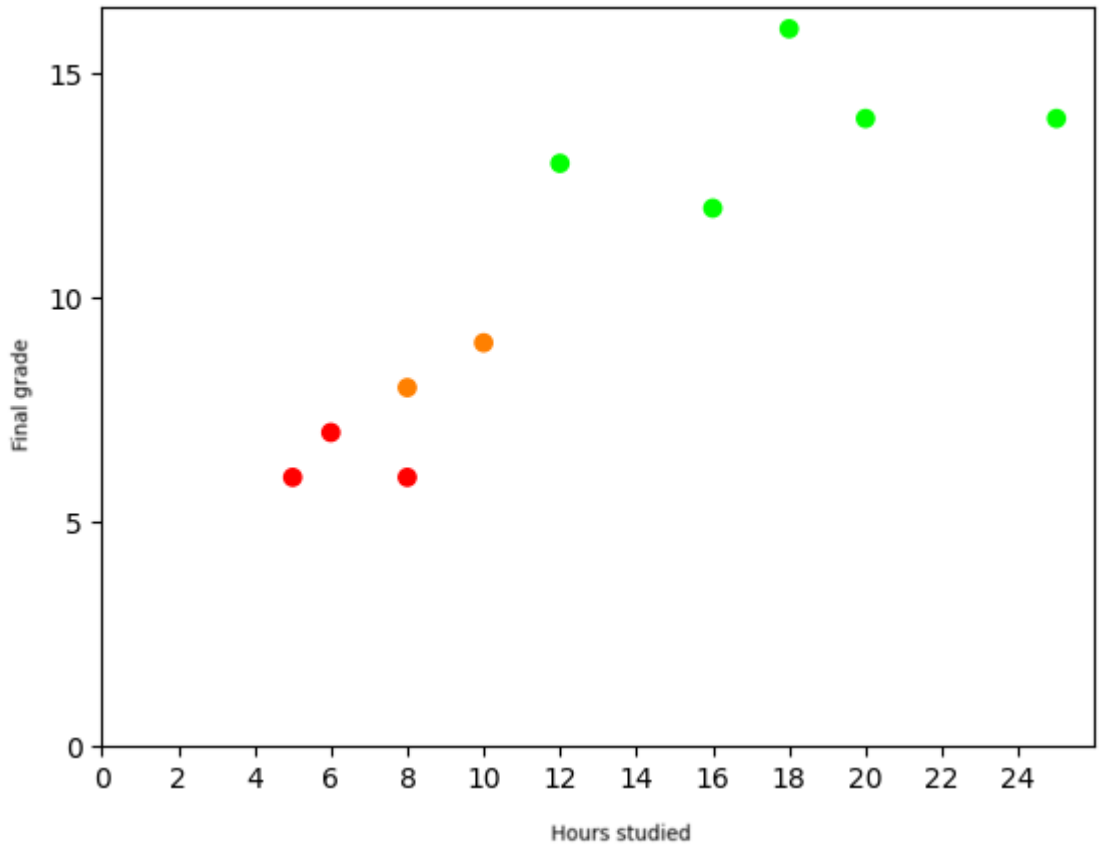
Each student is represented by a blue marker.

Relationship is clear:
The more you study, the higher your grades!



Scatter plot with toppings

- Colour of the dots is changed
- When you check the code you'll note we set the data for the colour manually, but you could have generated this easily
  - <=7: failed
  - 8-9: tolerated
  - >=10: passed

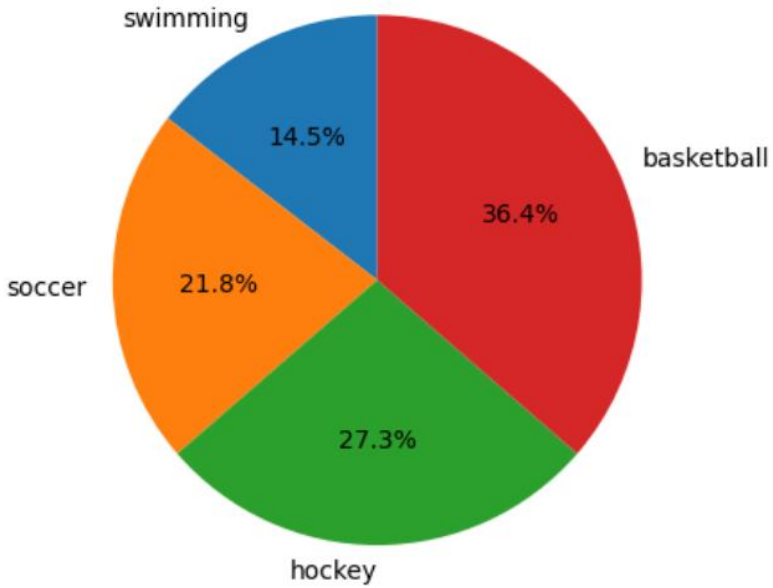


Pie chart

- Used for comparing the contribution of different categories to the whole.
- Can only be used when the distinct parts add up to a meaningful whole (100%).
- Best practice is to
  - plot the relative values (percentages) for each category
  - limit the number of categories to max 6
  - sort relative values clockwise descending

Pie chart

```
sports = np.array(['soccer', 'swimming', 'hockey', 'basketball'])
number_of_students = np.array([120, 80, 150, 200])

# sort
number_of_students,sports = zip(*sorted(list(zip(number_of_students, sports))))

plt.pie(number_of_students, labels=sports, autopct='%1.1f%%', startangle=90)
plt.show()
```



Summary

- These slides went over the different plots we expect you to be able to produce on the exam
- When using AI on an exam, consider the time you'll need for
  - Adding a legend manually if you know you'll need ".legend"
  - Prompt engineering the AI to generate the legend we are asking for
- Also bear in the mind the rules for the different graphs
  - Bar vs line
  - Bar vs histogram
  - Scatter
  - Pie chart