Lab 3: Matplotlib

The objective of this notebook is to learn about the Matplotlib library (official documentation). You can find a good guide at this link.

Outline

- 1. Drawing lines
- 2. Plot bars
- 3. Plot points and Multiple Charts

First, run the following cell to import some useful libraries to complete this Lab. If not already done, you must install them in your virtual environment

In [1]:

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

If the previous cell outputs one of the following errors: `ModuleNotFoundError: No module named 'numpy'` or `ModuleNotFoundError: No module named 'matplotlib'`, then, you have to install the numpy or the matplotlib packages. If you don't remember how to install a Python package, please retrieve the guide on Anaconda-Navigator.

To install **numpy** you can use one of the following commands from the terminal of your virtual environment:

```
conda install numpy
pip install numpy
```

To install **matplotlib** you can use one of the following commands from the terminal of your virtual environment:

```
conda install matplotlib
pip install matplotlib
```

1. Drawing lines

Exercise 1.1

Create a Numpy array `X` containing **100 samples evenly spaced** over the interval `[-10, 10]`. Then, define a variable `Y_squares` containing the squares of each element of `X`, and a variable `Y_lin`, where each element $y_i \in Y_{lin}$ is computed with the following linear equation: $y_i = x_i * 10 + 9$ for each $x_i \in X$.

▶ **Hints**

In [2]:

```
#### START CODE HERE (~3 lines) ####

X = np.linspace(-10, 10, 100)
Y_squares = X**2
Y_lin = X * 10 + 9

#### END CODE HERE ####

print(f"X shape: {X.shape}")
print(f"X min: {X.min()}")
print(f"X max: {X.max()}")
print(f"\nY_squares shape: {Y_squares.shape}")
print(f"Y_squares min: {Y_squares.min()}")
print(f"Y_squares max: {Y_squares.max()}")
print(f"\nY_lin shape: {Y_lin.shape}")
print(f"Y_lin min: {Y_lin.min()}")
print(f"Y_lin max: {Y_lin.max()}")
```

```
X shape: (100,)
X min: -10.0
X max: 10.0

Y_squares shape: (100,)
Y_squares min: 0.010203040506070672
Y_squares max: 100.0

Y_lin shape: (100,)
Y_lin min: -91.0
Y_lin max: 109.0
```

Expected output

```
X shape: (100,)
X min: -10.0
X max: 10.0
```

```
Y_squares shape: (100,)
Y_squares min: 0.010203040506070672
Y_squares max: 100.0

Y_lin shape: (100,)
Y_lin min: -91.0
Y_lin max: 109.0
```

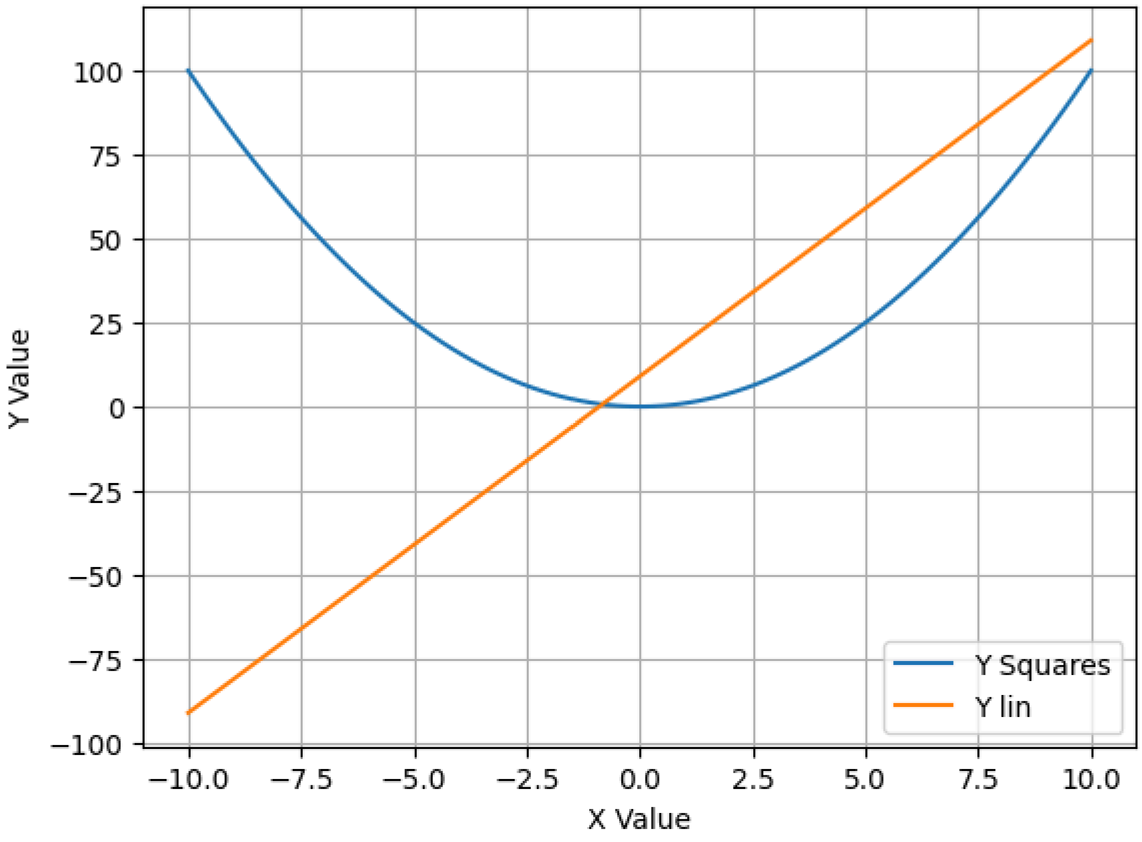
Exercise 1.2

Create a **single chart** containing the two **lines** `(X, Y_squares)` and `(X, Y_lin)`. You should set the **label** of the first line `(X, Y_squares)` as `Y squares` and the **label** of the second line `(X, Y_lin)` as `Y lin`. You should set the label of the **X axis** to `X Value` and of the **Y axis** to `Y Value`. You should also show the **legend** and the **grid** of the chart.

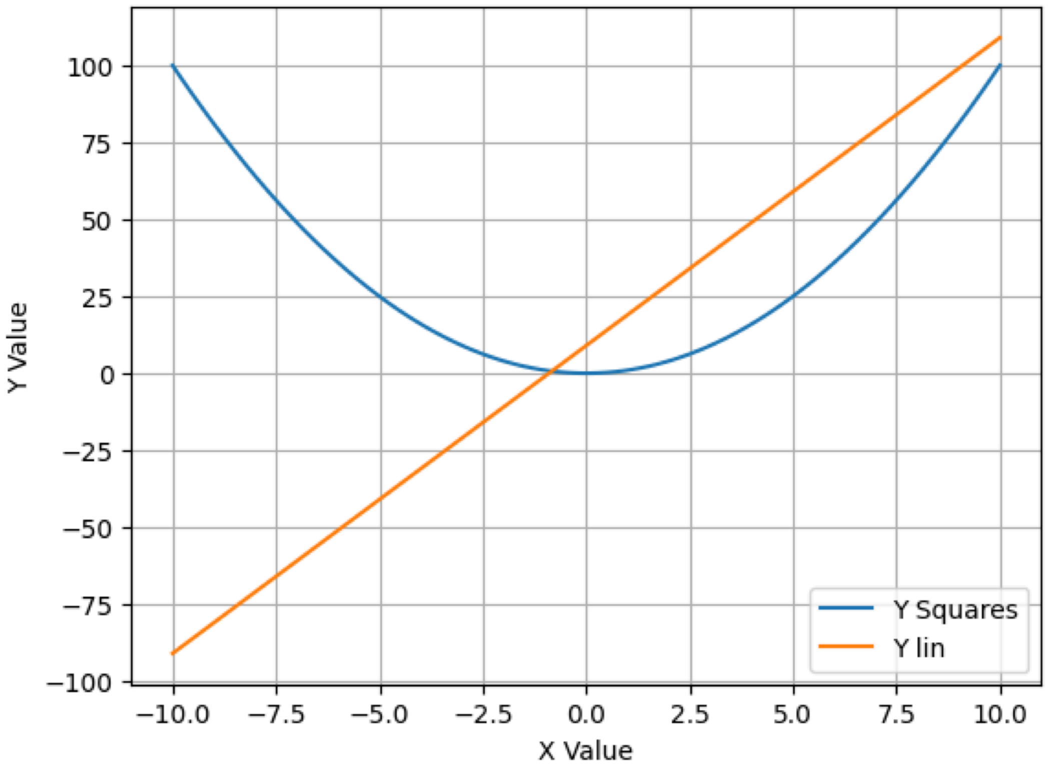
▶ **Hints**

In [3]:

```
#### START CODE HERE (~8 lines) ####

fig, ax = plt.subplots()

ax.plot(X, Y_squares, label='Y Squares')
ax.plot(X, Y_lin, label='Y lin')

ax.set_ylabel("Y Value")
ax.set_xlabel("X Value")

plt.legend()
plt.grid(True)
plt.show()

#### END CODE HERE ####
```



Expected output



2. Plot bars

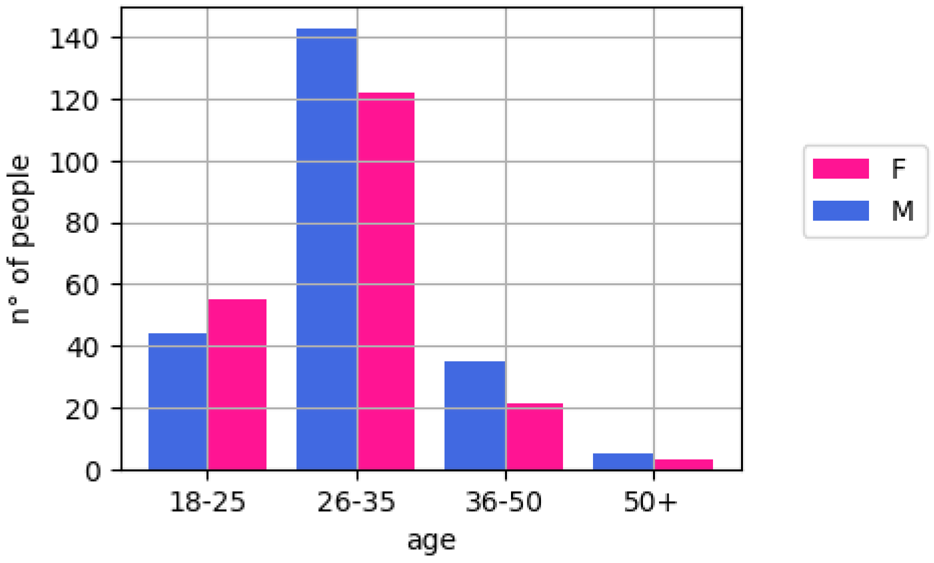
Exercise 2.1

Create a barplot chart starting from the two dictionaries, `males_dict` and `females_dict`. For each **age range** in the **keys** of the two dictionaries (i.e., `18-25`, `26-35`, `36-50`, and `50+`), you should plot the **two barplots** (male and then female) **side-by-side**. Put in the **xticks** the name of each age range (i.e., `18-25`, `26-35`, `36-50`, and `50+`). Then, set the name of the **x axis** to `age`, and the name of the **y axis** to `n° of people`. The bars for the **males** should be set to the color `"royalblue"`, and for the **females** to the color `"deeppink"`. You should also plot the **legend**. For the **males bars** in the legend you should put `M`, and for the **females** `F`. The legend should be **located on the right of the plot and in a center height** (you can set the location to the following values `(1.1, 0.5)`).

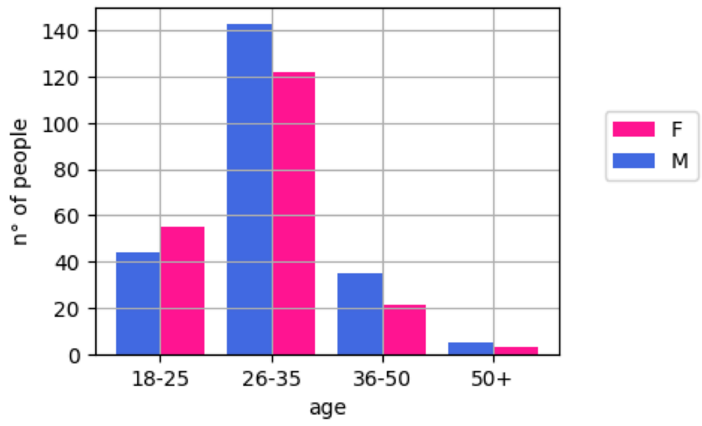
▶ **Hints**

In [4]:

```
females_dict = {"18-25": 55,"26-35":122, "36-50":21, "50+": 3 }
males_dict = {"18-25": 44,"26-35":143, "36-50":35, "50+": 5 }
bar_width = 0.4

#### START CODE HERE (~12 lines) ####

x = np.arange(len(males_dict.values()))

labels = list(males_dict.keys())

fig, ax = plt.subplots(figsize=(4, 3))

ax.bar(x+bar_width/2, females_dict.values(), color='deeppink', width=bar_width, label='F')
ax.bar(x-bar_width/2, males_dict.values(), color="royalblue", width=bar_width, label='M')

ax.set_xticks(x)              # setup positions of x ticks
ax.set_xticklabels(labels)    # set up labels of x ticks

ax.set_xlabel("age")
ax.set_ylabel("n° of people")

ax.legend(loc=(1.1, 0.5))     # x, y position, in percentage
plt.grid(True)
plt.show()

#### END CODE HERE ####
```



Expected output



3. Plot points and Multiple Charts

Please run the following cell containing useful functions already implemented for you to plot some charts.

In [5]:

```python
def is_above_fn(p, x, y):
    """ This functions returns True if the point p (as a tuple) is above the line defined by the points in x and y"""
    return np.cross(p-x, y-x) < 0

def find_above_and_below_points_fn(x, y, x_bound, y_bound):
    """ This funtion split a numpy array into two numpy arrays with all the points that stand
    above and below a line, respetively"""
    X_above = []
    X_below = []
    Y_above = []
    Y_below = []
    for xi, yi in zip(x, y):
        if is_above_fn((xi, yi), x_bound, y_bound):
            X_above.append(xi)
            Y_above.append(yi)
        else:
            X_below.append(xi)
            Y_below.append(yi)
    X_above = np.array(X_above)
    X_below = np.array(X_below)
    Y_above = np.array(Y_above)
    Y_below = np.array(Y_below)
    return X_above, Y_above, X_below, Y_below

def generate_gradient_colors_fn(x, y):
    """ This functin generates color values as a function of x and y"""
    return x + y
```

Exercise 3.1

In [6]:

```python
x = np.random.rand(20)
y = np.random.rand(20)

p1_bound = np.array([0.0, 1.0])
p2_bound = np.array([1.0, 0.0])

colors_1 = generate_gradient_colors_fn(x, y)
x_above, y_above, x_below, y_below = find_above_and_below_points_fn(x, y, p1_bound, p2_bound)
```

Plot **two charts side-by-side**.

In the **first chart**, you should draw a **green line** through the points `p1_bound` and `p2_bound`. Then, you should plot the **points** stored in the variables `x` and `y` with the **color list** stored in the variable `colors_1` (the colors are already computed as a gradient defined with a function of `x` and `y`) and the **colormap** `seismic`. Finally, you should set the name of the **X axis** to `X` and of the **Y axis** to `Y`.

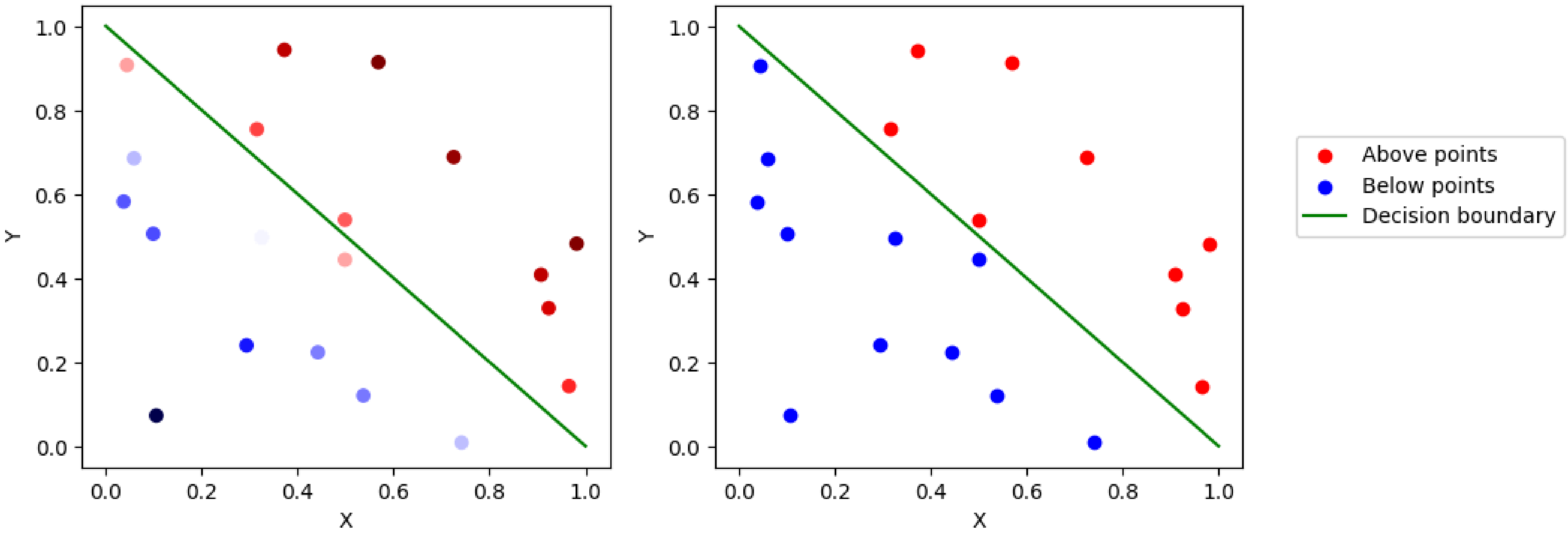
In the **second chart**, you should draw the same **green line** through the points `p1_bound` and `p2_bound`. This time, you should set the name of the line to `Decision boundary`. Then, you should plot all the points lying above the line, stored in the variable `X_above` (they are already computed for you) with the color `red` and the colormap `seismic`. Set the label for those points to `Above points`. Then, you should plot all the points lying below the line, stored in the variable `X_below` (they are already computed for you) with the color `blue` and the colormap `seismic`. Set the label for those points to `Below points`. Finally, you should set the name of the **X axis** to `X` and of the **Y axis** to `Y`, and show the legend of the second chart with the following location `loc=(1.1, 0.5)`.

▶ **Hints**

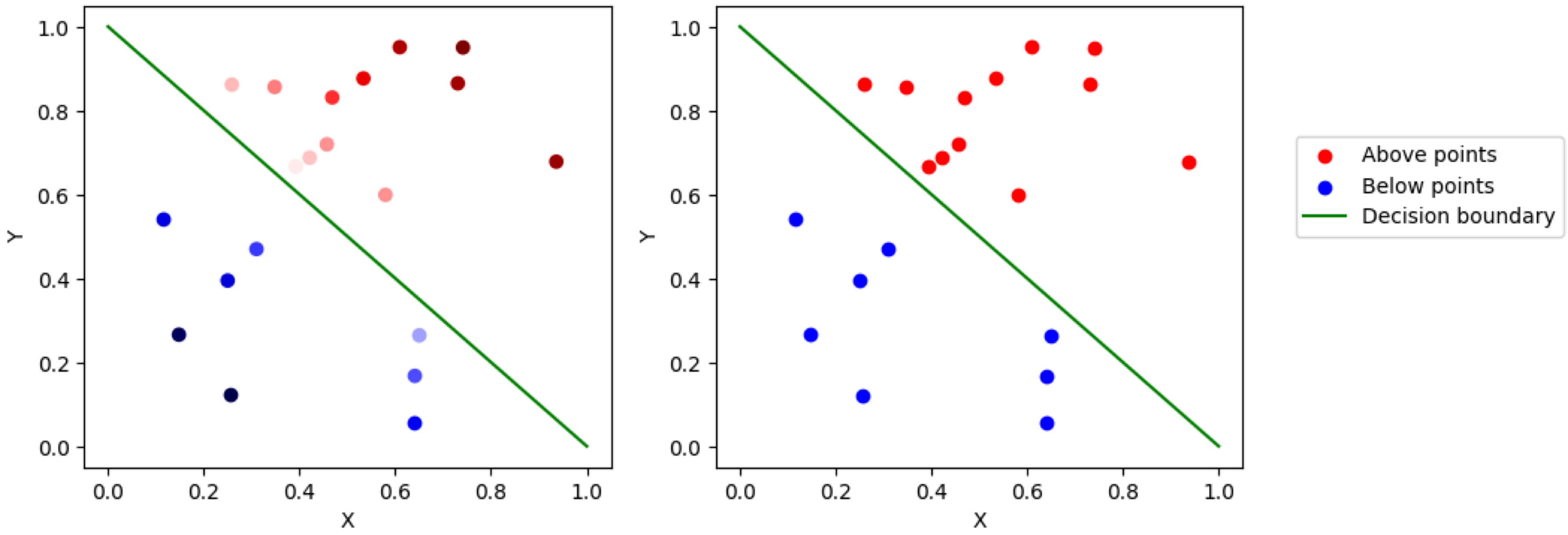
In [7]:

```python
#### START CODE HERE (~12 lines) ####

fig, ax = plt.subplots(1, 2, figsize=(10, 4))

ax[0].scatter(x, y, c=colors_1, cmap='seismic')
ax[0].plot(p1_bound, p2_bound, color='green')
ax[0].set_xlabel("X")
ax[0].set_ylabel("Y")

ax[1].scatter(x_above, y_above, c="red", cmap='seismic', label="Above points")
ax[1].scatter(x_below, y_below, c="blue", cmap='seismic', label="Below points")
ax[1].plot(p1_bound, p2_bound, color='green', label="Decision boundary")
ax[1].set_xlabel("X")
ax[1].set_ylabel("Y")
ax[1].legend(loc=(1.1, 0.5))

plt.show()

#### END CODE HERE ####
```

```
/var/folders/ck/5bn3d96976q9mdgwzsdcxtmw0000gn/T/ipykernel_5431/2808962858.py:10: UserWarning: No data for colormappi
ng provided via 'c'. Parameters 'cmap' will be ignored
  ax[1].scatter(x_above, y_above, c="red", cmap='seismic', label="Above points")
/var/folders/ck/5bn3d96976q9mdgwzsdcxtmw0000gn/T/ipykernel_5431/2808962858.py:11: UserWarning: No data for colormappi
ng provided via 'c'. Parameters 'cmap' will be ignored
  ax[1].scatter(x_below, y_below, c="blue", cmap='seismic', label="Below points")
```



Expected output


Y
X
Above points
Below points
Decision boundary