Data Science and Machine Learning for Engineering Applications

Lecture Notes 3: Matplotlib

March 22, 2023 - Politecnico di Torino

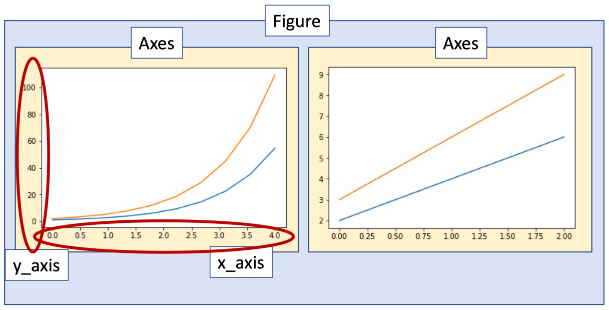
1 Matplotlib Introduction

Matplotlib [1] is a **graphical library** for creating static, animated, and interactive visualizations in Python. In this course, we will use the object-oriented interface of the library (stateless). You can find the official documentation, and a good guide here.

2 Figures and Axes

The **Figure** object contains all the plots. Indeed, you can have **multiple charts** (i.e., plots) inside the **Figure** object. When you plot your figure, if you are using Jupyter, it will show the plot as the output of the cell. You can also save the figure as an image file (e.g., jpeg, png, etc.).

You can have 1 or more **Axes** inside the figure object. Each **Axes** object corresponds to a **chart**. Each **Axes** has an **x_axis** and an **y_axis** that represent the data that you want to plot.

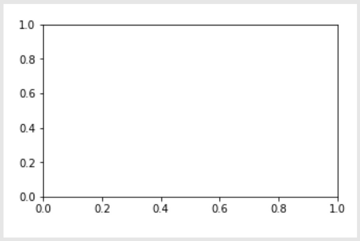


2.1 Create a new figure

To create a **Figure** you should use the `subplots()` function of the **Matplotlib** library. The `subplots()` returns a new **Figure** and its **Axes** object. In the creation, you can specify the figure size (**figsize**) as a tuple `(width, height)` in **inches**. As a default, **ax** is a single **Axes** object (i.e., a Figure with 1 single chart). However, you can also specify multiple charts. To create a **Figure** with a single **chart**, you can simply call the `subplots()` function without specifying the number of charts:

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 3)) # The width and the height are 5 and 3 inches
plt.show()
```

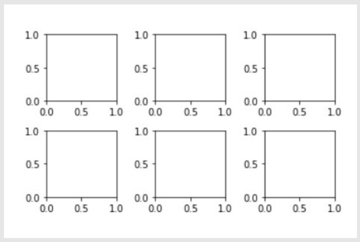
`Output:`



If you want to create **multiple charts**, you can specify it in the parameters. The **first two parameters** of `plt.subplots()` specify the number of charts as **rows** and **columns**. For example, if you specify `plt.subplots(2, 3)`, you will create a **Figure** object with 6 axes in two rows, and three columns. The `plt.subplots(2, 3)` will return the **Figure** object and the **ax** as a Numpy array of charts. You can access each chart with the *Numpy indexing* methods.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(2, 3, figsize=(5, 3)) # multiple charts with 2 rows and 3 cols
plt.tight_layout() # required to visualize correctly
plt.show()
```

Output:



The `plt.tight_layout()` fits all your charts in the figure correctly. It is necessary at the end to let the subplots fit the frame size without blank spaces at the borders

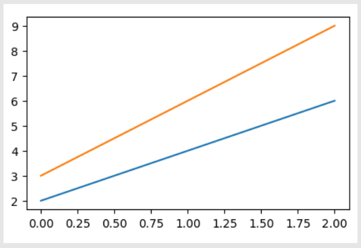
3 Plotting

3.1 Drawing a line plot into a single Axes object

To plot a line into a single **Axes** object, you can use the `plot()` method of the `ax` object (`ax.plot()`). The `ax.plot()` takes as input two lists or Numpy arrays: the first list contains the x of the points in the line, while the second the y coordinates of the points in the line (e.g., (x[0], y[0]) is the first point, (x[1], y[1]) is the second point, etc.). The default style of the `ax.plot()` method draws **segments** passing through the specified coordinates. If you perform subsequent calls of `ax.plot()`, it will add a new segment (i.e., line) to the same Figure. This example plots two lines in one chart:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 3))
ax.plot([0,1,2],[2,4,6]) # line through (x,y) -> (0,2), (1,4), (2,6)
ax.plot([0,1,2],[3,6,9]) # new line through (x,y) -> (0,3), (1,6), (2,9)
plt.show()
```

Output:

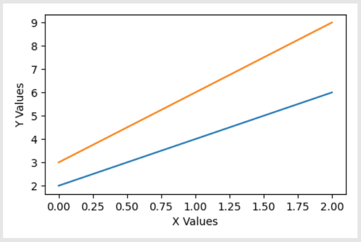


3.1.1 Specifying x and y axis labels

You can specify the labels (i.e., name) of the x and y axis with `.set_xlabel()` and `.set_ylabel()`, respectively.

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 3))
ax.plot([0,1,2],[2,4,6]) # line through (x,y) -> (0,2), (1,4), (2,6)
ax.plot([0,1,2],[3,6,9]) # new line through (x,y) -> (0,3), (1,6), (2,9)
ax.set_xlabel("X Values")
ax.set_ylabel("Y Values")
plt.show()
```

`Output:`

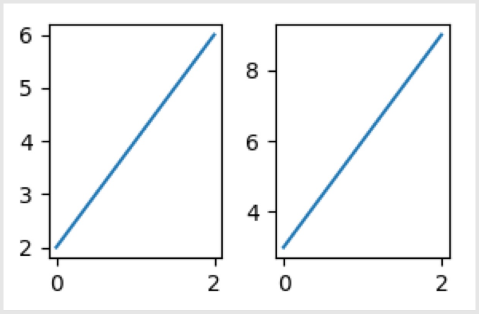


3.2 Drawing a line plot into one row and multiple columns Axes

To plot two charts in **1 row** and ***n* columns**, you should specify that you want 1 row and n columns in the construction of the subplot, for example, `plt.subplots(1,2)`. The returned `ax` object is a **Numpy array** with the created axes objects. If the figure has 1 row and n columns (it is a row vector), the `ax` object has **shape = (n,)**. You can access each **Axes** by indexing the `ax` object (i.e., `ax[0]` is the first chart, `ax[1] the second chart`).

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 2, figsize=(5, 3)) # create plot with 1 row and 2 cols
ax[0].plot([0,1,2],[2,4,6]) # line in the first chart
ax[1].plot([0,1,2],[3,6,9]) # line in the second chart
plt.tight_layout()
plt.show()
```

`Output:`

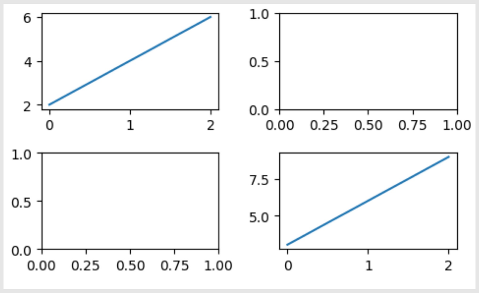


3.3 Drawing a line plot into multiple rows and multiple columns Axes

If you want to plot charts in **multiple rows** and **multiple columns**, you have to specify the number of rows and columns in the construction (`plt.subplots(n_rows, n_cols)`). It returns an `ax` object that is a Numpy 2-dimensional array with **shape= (n_rows, n_cols)**. You can access each plot in the same way you access Numpy arrays. However, you must be careful this time that **ax** is a two-dimensional array.

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(2, 2, figsize=(5, 3)) # create plot with 2 rows and 2 cols
ax[0, 0].plot([0,1,2],[2,4,6]) # chart in first row and first column
ax[1, 1].plot([0,1,2],[3,6,9]) # chart in the second row and second column
plt.tight_layout()
plt.show()
```

Output:

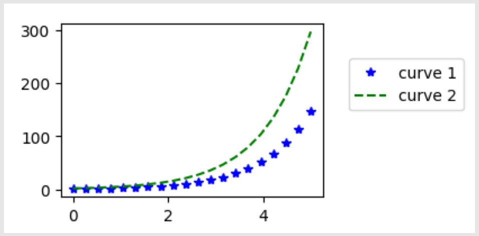


3.4 Drawing a sequence of point/segments

The `plt.plot()` function also allows displaying a sequence of points/segments that **share the same properties** (e.g., size, color, width). To do so, you have to specify the `linestyle` and/or the `marker` parameters. The `linestyle` specifies the type of line that connects the points (e.g., '-', '–', '.', ':'), and the `marker` specifies the type of points to be drawn (e.g., 'o', '*', '+'). There is also a parameter to specify the color `c` to be applied to markers and segments. You can specify i) the name of the color (e.g., 'red', 'green', blue); ii) the hexadecimal value (e.g., '#OFOF6B'); iii) the RGBA tuple values (e.g., 0.5, 1, 0.8, 0.8). You can read all the possible colors in the documentation.

```python
import matplotlib.pyplot as plt
x = np.linspace(0, 5, 20)
y = np.exp(x)
fig, ax = plt.subplots(figsize=(3, 2))
ax.plot(x, y, c="blue", linestyle="", marker='*', label='curve 1')
ax.plot(x, 2*y, c='green', linestyle='--', label='curve 2')
ax.legend(loc=(1.1, 0.5)) # specify the legend position with relative position
plt.show()
```

Output:

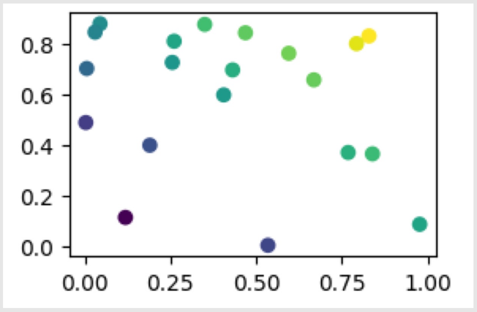


3.5 Scatter plot: displaying a set of points

If you want to show a **set of points** and assign them **custom properties** (e.g., color, size), you can use the `ax.scatter()` method. You have to specify the list of `x` and `y` coordinates of all your points as parameters (as a list or NumPy array). You can also specify a **list of numbers** as **colors** with the `c` parameter (one color for each point). In this example, points are plotted with a color that is a function of the position (x and y) of the points:

```python
import matplotlib.pyplot as plt
x = np.random.rand(20)
y = np.random.rand(20)
colors = x + y  # color as a function of the positions of the point (x and y)
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors)
plt.show()
```

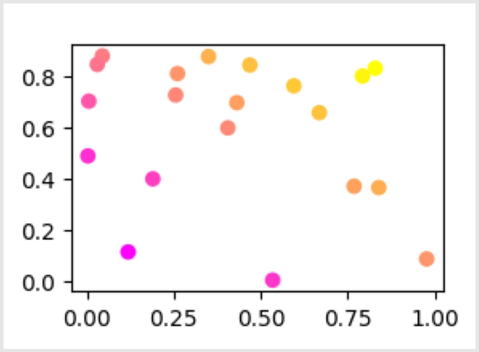
Output:



You can also specify a **colormap** with the `cmap` parameter (you can see all the colormaps in the documentation):

```python
import matplotlib.pyplot as plt
colors = x + y  # color as a function of x and y
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors, cmap='spring') # specify the colormap to 'spring'
plt.show()
```

Output:

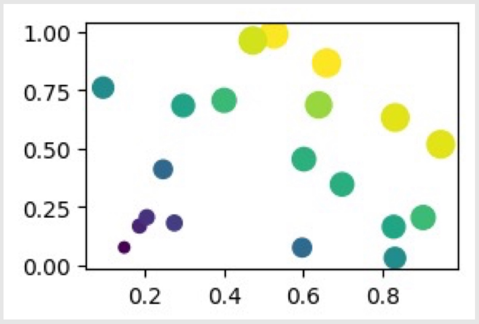


You can also specify the **size** of the points with the `s` parameter. The size is expressed as the **area** in **dpi**. This example shows how to change the size of the points:

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

x = np.random.rand(20)
y = np.random.rand(20)
colors = x + y     # color as a function of x and y
area = 100*(x+y)   # size as a function of x, y
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors, s=area)
plt.show()
```

Output:



3.6 Bar chart: displaying a sequence of numbers as bars

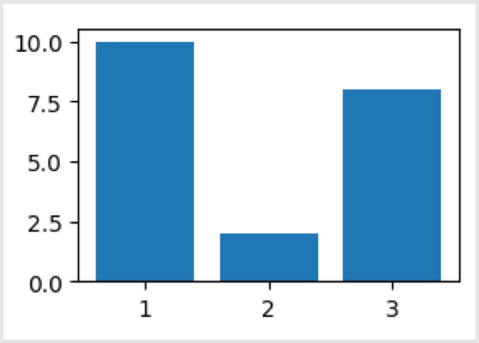
If you want to plot **vertical** or **horizontal bars**, you can use the `ax.bar()` method. It allows displaying a sequence of numbers as vertical or horizontal bars. You should specify the **position** of each bar on the *x* axis as a list, and the **height** of each bar as a list (with the same size.)

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

height = [10, 2, 8]
x = [1, 2, 3]     # position of the bars, x axis

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x, height)
plt.show()
```

Output:



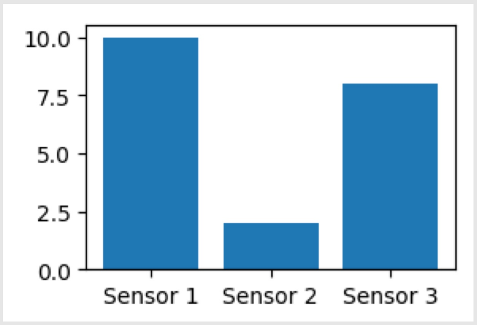
You can also assign a **text label** to the **ticks** on the horizontal axis:

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

height = [10, 2, 8]
x = [1, 2, 3] # position of the bars, x axis
labels = ['Sensor 1', 'Sensor 2', 'Sensor 3']

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x, height, tick_label=labels)
plt.show()
```

Output:



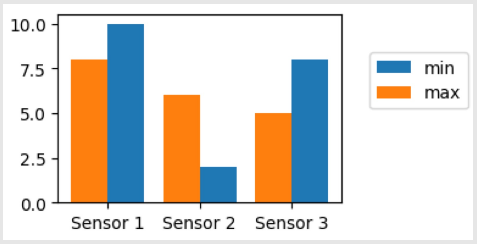
You can also **group multiple bars** side-by-side. You should put the position of the two bar plots as `x + width /2` and `x + width /2`, as shown in the following example:

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

height_min = [10, 2, 8]
height_max = [8, 6, 5]
x = np.arange(3)
width = 0.4
labels = ['Sensor 1', 'Sensor 2', 'Sensor 3']

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x+width/2, height_min, width=width, label='min')  # blue bars
ax.bar(x-width/2, height_max, width=width, label='max')  # orange bars
ax.set_xticks(x)              # setup positions of x ticks
ax.set_xticklabels(labels)    # set up labels of x ticks
ax.legend(loc=(1.1, 0.5))     # x, y position, in percentage
plt.show()
```

`Output:`



4 Save a plot to file

The generated figures can be **saved** to files with different formats (e.g., PNG, JPEG, PDF, EPS, etc.). You should use the `fig.savefig()`.

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(3, 2))
ax.plot([0,1,2],[2,4,6])
ax.plot([0,1,2],[3,6,9])
fig.savefig("./out/test.png") # or .jpg, .eps, .pdf
```

`Output:`

test.png file in the file system

References

[1] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.