Lab 7: Data Visualization using Matplotlib

Acknowledgement: This lab sheet is based on the online tutorial at

https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python

Section 1: What Does A Matplotlib Python Plot Look Like?

Matplotlib is a very powerful and flexible visualization library. In the simplest way, you only need to make the necessary imports, prepare some data, and you can start plotting with the help of the `plot()` function! When you're ready, don't forget to show your plot using the `show()` function.

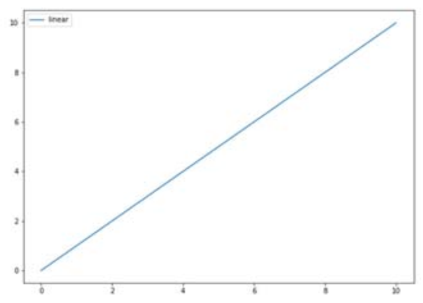
In Spyder's IPython console, we use the following commands to control where to display your figures:

%matplotlib inline – your figures will be shown inside the Console

%matplotlib qt – your figures will be shown in a separate window

Let's begin with the following example to see how easy it really is to plot a figure:

```
# Import the necessary packages and modules
import matplotlib.pyplot as plt
import numpy as np
# Prepare the data
x = np.linspace(0, 10, 100)
# Plot the data
plt.plot(x, x, label='linear')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

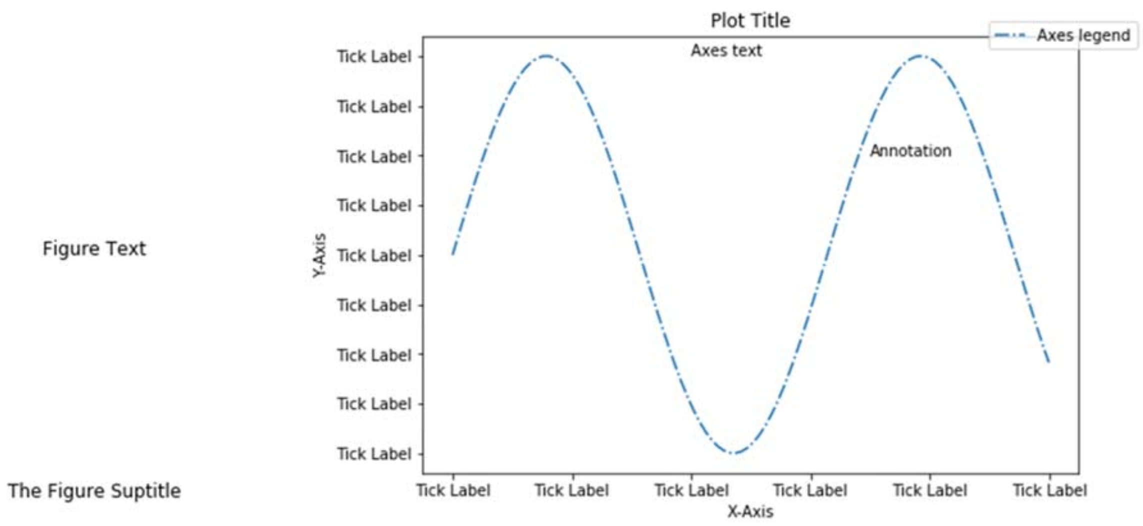


Note that you import the `pyplot` module of the `matplotlib` library under the alias `plt`.

Remark: the NumPy function **np.linspace(start, stop, num)** returns a one-dimensional ndarray with **num** evenly spaced samples over the interval [**start**, **stop**].

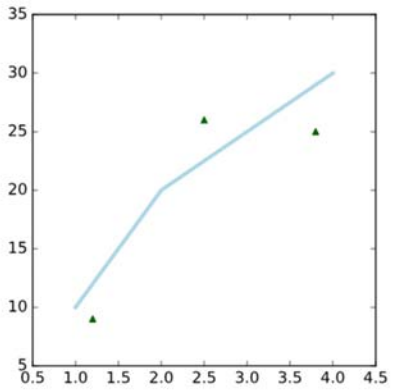
What you can't see on the surface is that you have made use of the built-in defaults that take care of the creation of the underlying components, such as the **Figure** and the **Axes**. In essence, there are two big components that you need to take into account:

- The **Figure** is the overall window or page that everything is drawn on. It's the top-level component of all the ones that you will consider in the following points. A Figure can have several other things in it, such as a **suptitle**, which is a centered title to the figure. You'll also find that you can add a **legend**, for example, to your Figure.
- To the figure you add **Axes**. The Axes is the area on which the data is plotted with functions such as `plot()` and `scatter()` and that can have **ticks**, **labels**, etc. associated with it. And, a Figure can contain multiple Axes.



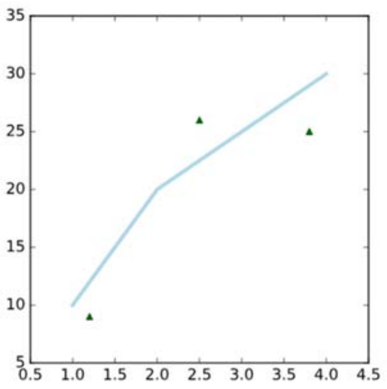
You'll see what "clean" means when you compare the following piece of code:

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue',
linewidth=3)
ax.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color
='darkgreen', marker='^')
ax.set_xlim(0.5, 4.5)
plt.show()
```



with the piece of code below:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue',
linewidth=3)
plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26],
color='darkgreen', marker='^')
plt.xlim(0.5, 4.5)
plt.show()
```



The second code chunk is definitely cleaner, isn'it it?

However, if you have multiple axes, it's still better to make use of the first code chunk because it's always better to prefer explicit above implicit code! In such cases, you want to make use of the Axes object `ax`.

Next to these two components, there are a couple more that you can keep in mind: each Axes has an **x-axis** and a **y-axis**, which contain **ticks** and **tick labels**.

There are also the **axis labels**, **title**, and **legend** to consider when you want to customize your axes, but also taking into account the **axis scales** and **gridlines** might come in handy.

How are Matplotlib and pyplot related?

First off, you'll already know Matplotlib by now. When you talk about "Matplotlib", you talk about the whole Python data visualization package. Secondly, `pyplot` is a module in the `matplotlib` package. That's why you often see `matplotlib.pyplot` in code. The module provides an interface that allows you to implicitly and automatically create figures and axes to achieve the desired plot.

This is especially handy when you want to quickly plot something without instantiating any Figures or Axes, as you saw in the example in the first section of this tutorial. You see, you haven't explicitly specified these components, yet you manage to output a plot that you have even customized! The defaults are initialized and any customizations that you do, will be done with the current Figure and Axes in mind.

Data For Matplotlib Plots

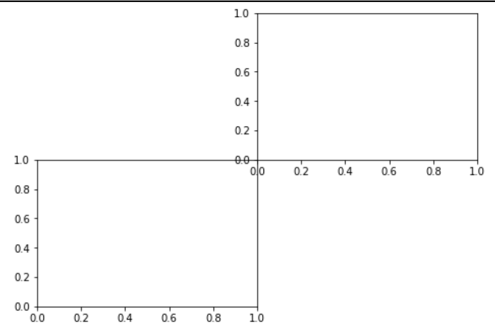
As you have read in one of the previous sections, Matplotlib is often used to visualize analyses or calculations. That's why the first step that you have to take in order to start plotting in Python yourself is to consider revising NumPy, the Python library for scientific computing.

Scientific computing might not really seem of much interest, but when you're doing data science you'll find yourself working a lot with data that is stored in arrays. You'll need to perform operations on them, inspect your arrays and manipulate them so that you're working with the (subset of the) data that is interesting for your analysis and that is in the right format, etc.

Section 2: Create Your Plot

Alright, you're off to create your first plot yourself with Python! As you have read in one of the previous sections, the Figure is the first step and the key to unlocking the power of this package. Next, you see that you initialize the axes of the Figure in the code chunk above with `fig.add_axes()`:

```
%matplotlib inline

# Import pyplot

import matplotlib.pyplot as plt

# Initialize a Figure

fig = plt.figure()

# Add two Axes to the Figure

ax1 = fig.add_axes([0,0,0.5,0.5])

ax2 = fig.add_axes([0.5,0.5,0.5,0.5])
```



The **plt.figure()** function generates a Figure object (say, fig), and then the **fig.add_axes(rect)** function adds an axes at position ***rect* ([*left*, *bottom*, *width*, *height*])** where all quantities are in fractions of figure width and height.

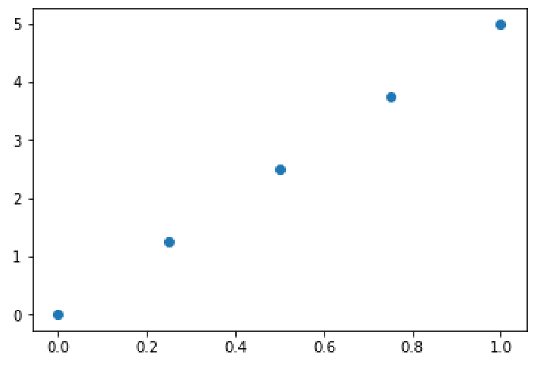
What Is a Subplot?

You have seen all components of a plot and you have initialized your first figure and Axes, but to make things a bit more complicated, you'll sometimes see **subplots** pop up in code.

You use subplots to set up and place your Axes on a regular grid. So that means that in most cases, **Axes and subplot are synonymous**, they will designate the same thing. When you do call subplot to add Axes to your figure, do so with the `add_subplots()` function. There is, however, a difference between the `add_axes()` and the `add_subplots()` function, but you'll learn more about this later on in the tutorial.

Consider the following example:

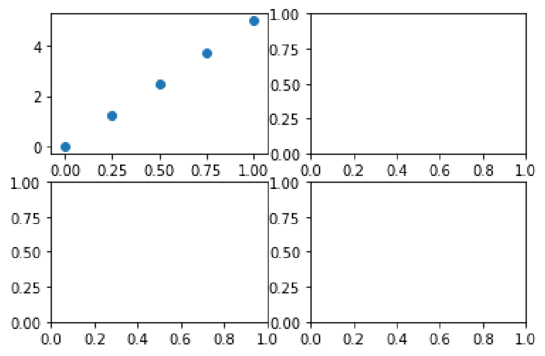
```
%matplotlib inline
# Import the necessary packages and modules
import matplotlib.pyplot as plt
import numpy as np
# Create a Figure
fig = plt.figure()
# Set up Axes
ax = fig.add_subplot(111)
# Scatter the data
ax.scatter(np.linspace(0, 1, 5), np.linspace(0, 5, 5))
# Show the plot
```



What does `add_subplot(111)` mean?

Well, `111` is equal to `1,1,1`, which means that you actually give three arguments to `add_subplot()`. The three arguments designate the number of rows (1), the number of columns (1) and the plot number (1). So you actually make one subplot. You can also try the following statements:

```
fig = plt.figure()
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)
ax1.scatter(np.linspace(0, 1, 5), np.linspace(0, 5, 5))
plt.show()
```



That's right, your Figure will have four axes in total, arranged in a structure that has two rows and two columns. With the line of code that you have considered, you say that the variable ax1 is the first of the four axes to which you want to start plotting. The "first" in this case means that it will be the first axes on the left of the 2x2 structure that you have initialized.

What Is The Difference Between `add_axes()` and `add_subplot()`?

The difference between `fig.add_axes()` and `fig.add_subplot()` doesn't lie in the result: they both return an Axes object. However, they do differ in the mechanism that is used to add the axes: you pass a list to `add_axes()` which is the lower left point, the width and the height. This means that the axes object is positioned in absolute coordinates.

In contrast, the `add_subplot()` function doesn't provide the option to put the axes at a certain position: it does, however, allow the axes to be situated according to **a subplot grid**, as you have seen in the section above.

In most cases, you'll use `add_subplot()` to create axes; Only in cases where the positioning matters, you'll resort to `add_axes()`. Alternatively, you can also use `subplots()` if you want to get one or more subplots at the same time.

How To Change The Size of Figures

Now that you have seen how to initialize a Figure and Axes from scratch, you will also want to know how you can change certain small details that the package sets up for you, such as the figure size.

Let's say you don't have the luxury to follow along with the defaults and you want to change this. How do you set the size of your figures manually?

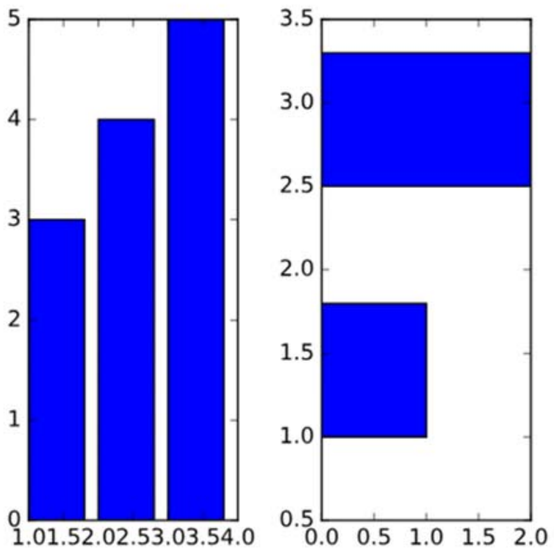
Like everything with this package, it's pretty easy, but you need to know first what to change.

Add an argument `figsize` to your `plt.figure()` function of the `pyplot` module; You just have to specify a tuple with the width and hight of your figure in inches, just like this `plt.figure(figsize=(3,4))`, for it to work.

Note that you can also pass `figsize` to the `plt.subplots()` function of the same module; The inner workings are the same as the `figure()` function that you've just seen.

See an example of how this would work here:

```
%matplotlib inline
# Import `pyplot` from `matplotlib`
import matplotlib.pyplot as plt
# Initialize the plot
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
# or replace the three lines of code above by the
following line:
#fig, (ax1, ax2) = plt.subplots(1,2, figsize=(20,10))
# Plot the data
ax1.bar([1,2,3],[3,4,5])
ax2.barh([0.5,1,2.5],[0,1,2])
# Show the plot
plt.show()
```



The **plt.subplots(nrows, ncols)** function returns a tuple of (Figure, array of axes). The argument (nrows, ncols) defines the Figure grid.

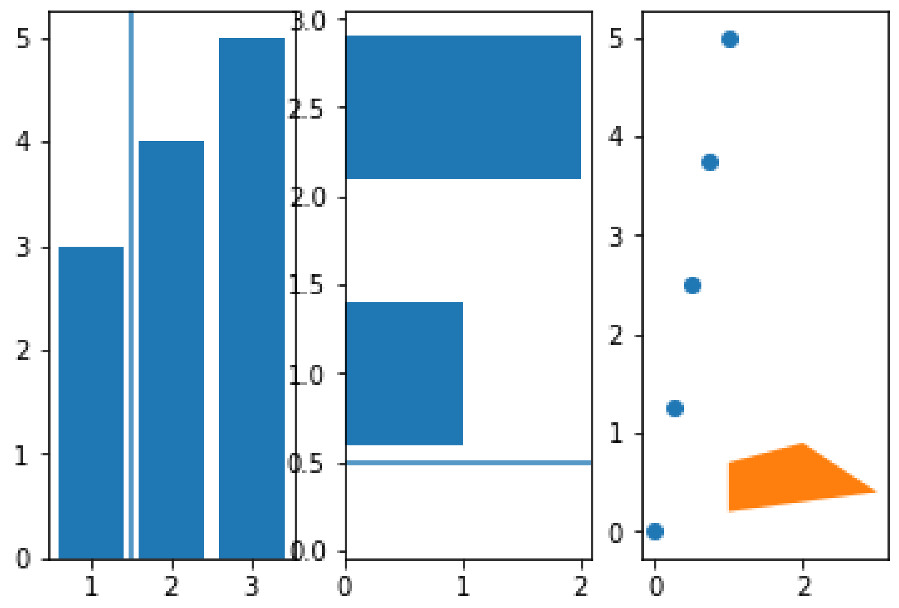
Section 3: Working With Pyplot: Plotting Routines

Now that all is set for you to start plotting your data, it's time to take a closer look at some plotting routines. You'll often come across functions like `plot()` and `scatter()`, which either draw points with lines or markers connecting them, or draw unconnected points, which are scaled or colored. These functions are only the bare basics. You will need some other functions to make sure your plots look awesome:

ax.bar()	Vertical rectangles
ax.barh()	Horizontal rectangles
ax.axhline(y)	Horizontal line across axes
ax.axvline(x)	Vertical line across axes
ax.fill_between(x, y1, y2)	Fill between two horizontal curves

If you're curious how you can use these functions to plot your data, consider the following example.

```
# Import `pyplot` from
`matplotlib`

import matplotlib.pyplot as
plt

# Initialize the plot

fig = plt.figure()

ax1 = fig.add_subplot(131)

ax2 = fig.add_subplot(132)

ax3 = fig.add_subplot(133)

# Plot the data

ax1.bar([1,2,3],[3,4,5])

ax2.barh([0.5,1,2.5],[0,1,2])

ax2.axhline(0.5)

ax1.axvline(1.5)

x = np.linspace(0, 1, 5)

y = np.linspace(0, 5, 5)

ax3.scatter(x,y)

ax3.fill_between([1, 2, 3],
[0.2, 0.3, 0.4], [0.7, 0.9, 0.4])

# Show the plot

plt.show()
```



Section 4: Customizing Your PyPlot

A lot of questions about this package come from the fact that there are a lot of things that you can do to **personalize your plots** and make sure that they are unique: besides adjusting the colors, you also have the option to change **markers**, **linestyles** and **linewidths**, add text, legend and annotations, and change the limits and layout of your plots.

It's exactly the fact that there is an endless range of possibilities when it comes to these plots that makes it difficult to set out some things that you need to know when you start working on this topic.

Great tips that you should keep in the back of your mind are not only the [gallery](), which contains many real-life examples that are already coded for you and which you can use, but also the [documentation](), which can tell you more about the arguments that you can pass to certain functions to adjust visual features.

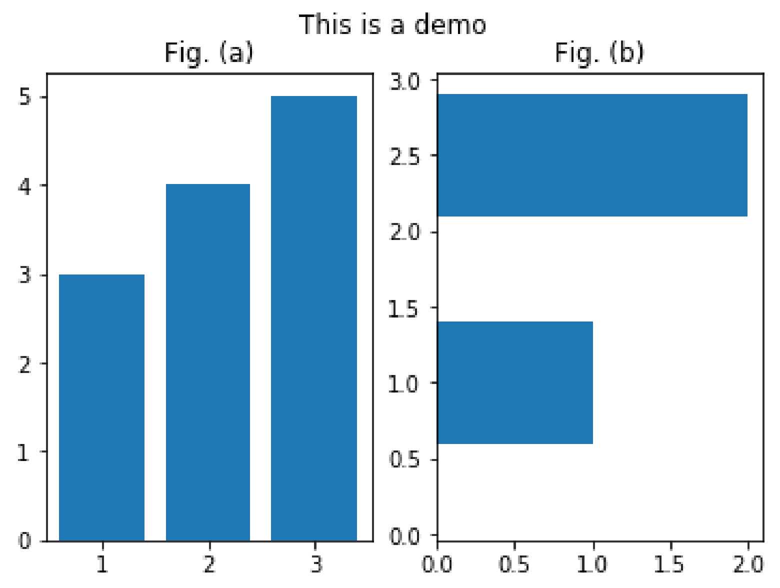
Also keep in mind that there are multiple solutions for one problem and that you learn most of this stuff when you're getting your hands dirty with the package itself and when you run into troubles.

How To Set Plot Title And Axes Labels

To change your plot title and axes labels, you can follow one of the following approaches, depending of which container of which you want to make use:

- The easiest way to set these things right is by using `ax.set(title="A title", xlabel="x", ylabel="y")` or `ax.set_xlim()`, `ax.set_ylim()` or `ax.set_title()`.
- If you want to work with the figure, you might also resort to `fig.suptitle()` to add a title to your plot.
- If you're making use of the default settings that the package has to offer, you might want to use `plt.title()`, `plt.xlabel()`, `plt.ylabel()`.

```
# Import `pyplot` from `matplotlib`
%matplotlib inline
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
# Plot the data
ax1.bar([1,2,3],[3,4,5])
ax2.barh([0.5,1,2.5],[0,1,2])
ax1.set(title="Fig. (a)")
ax2.set_title("Fig. (b)")
# Show the plot
fig.suptitle("This is a demo")
plt.show()
```


This is a demo
Fig. (a)
Fig. (b)