Visualization_with_Matplotlib_L1

February 8, 2021

1 Visualization with matplotlib I

1.1 Content:

1.2 Introduction

One of the most important modules in Python for 2-dimensional visualization is Matplotlib. It is a very comprehensive package with many display options for 2-dimensional data, whether xy plots, bar charts, pie charts, box plots, contours, vectors, scatters, and so on.

Matplotlib home page https://matplotlib.org/

Matplotlib provides many example scripts with a short description, see https://matplotlib.org/3.3.3/gallery/index.html For the beginner it is sometimes difficult to get started, but through the gallery with the many examples you can quickly find something that suits your needs.

Matplotlib's collection of style functions **pyplot** work very similar to MATLAB. For a detailed description see https://matplotlib.org/users/pyplot_tutorial.html.

1.3 Preliminary work

Jupyter notebook offers the possibility to display the plots within the notebook. Therefore we have to turn on the matplotlib mode.

```
[1]: %matplotlib inline
```

1.4 Import plot library

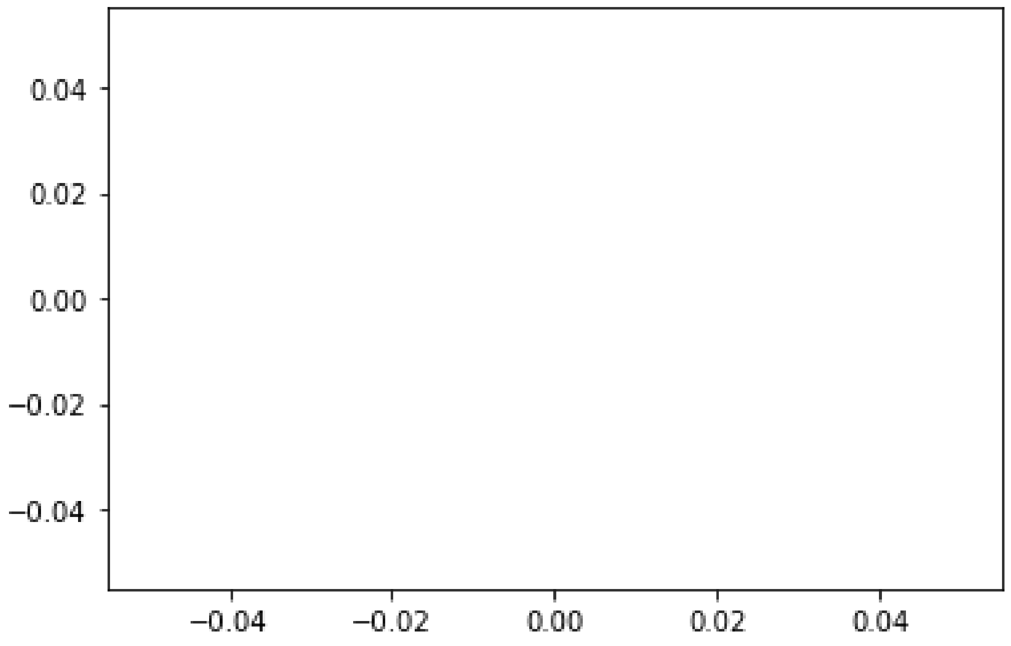
Import the function collection pyplot from matplotlib and use the shortcut plt for it.

```
[2]: import matplotlib.pyplot as plt
```

1.5 A simple plot

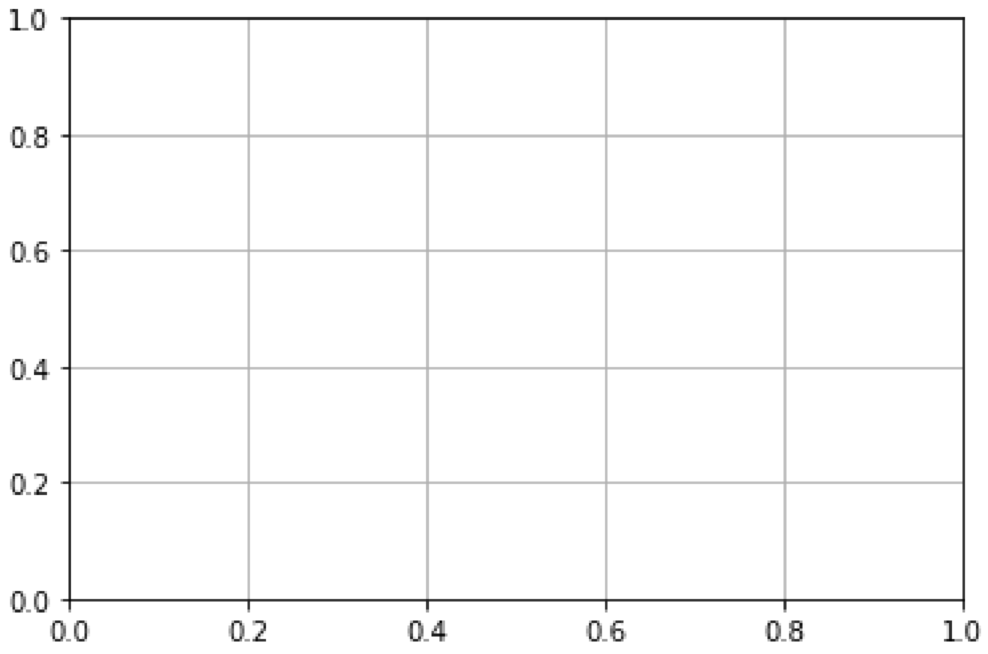
```
[3]: plt.plot()
```

```
[3]: []
```



1.6 Add grid lines

```
[4]: plt.grid()
```



1.7 Basic concepts

Matplotlib allows us to create a basic plot in an comfortable way. Only two lists or arrays are needed.

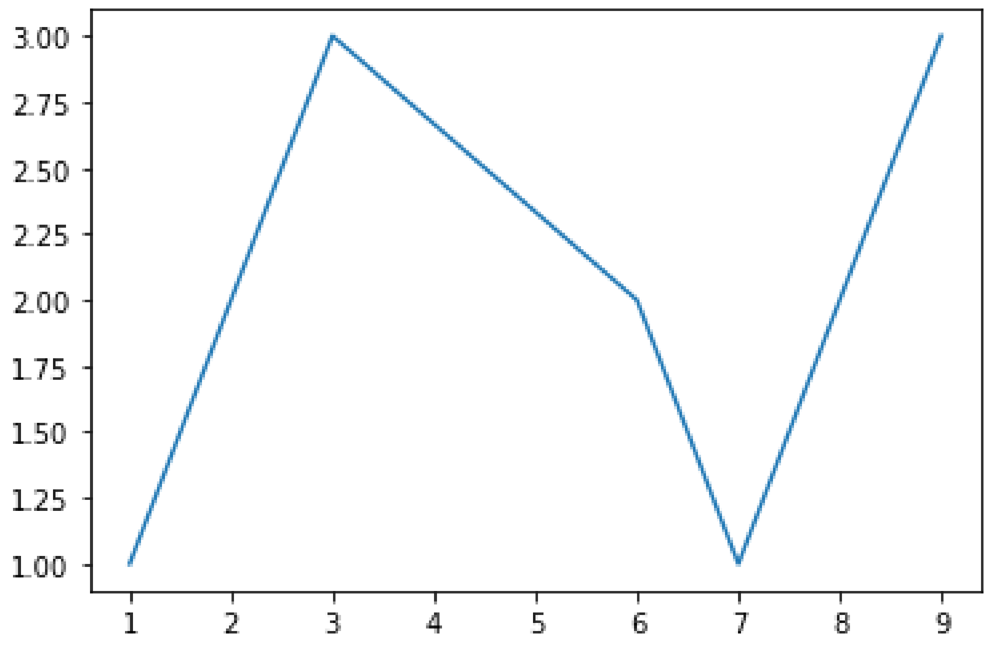
We define two short lists containing some integer values for x and y with the same size.

[5]:
```
x = [1,3,6,7,9]
y = [1,3,2,1,3]
```

Now we use the two lists to generate a line plot with Matplotslib's pyplot.plot function (we use the abbrieviation plt for pyplot). Running the following command line will display the line plot inside this notebook (if you have set `%matplotlib inline`).

[6]:
```
plt.plot(x, y)
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7f8182c81160>]
```



This plot is more than minimalistic and we will show you how to change the line style now.

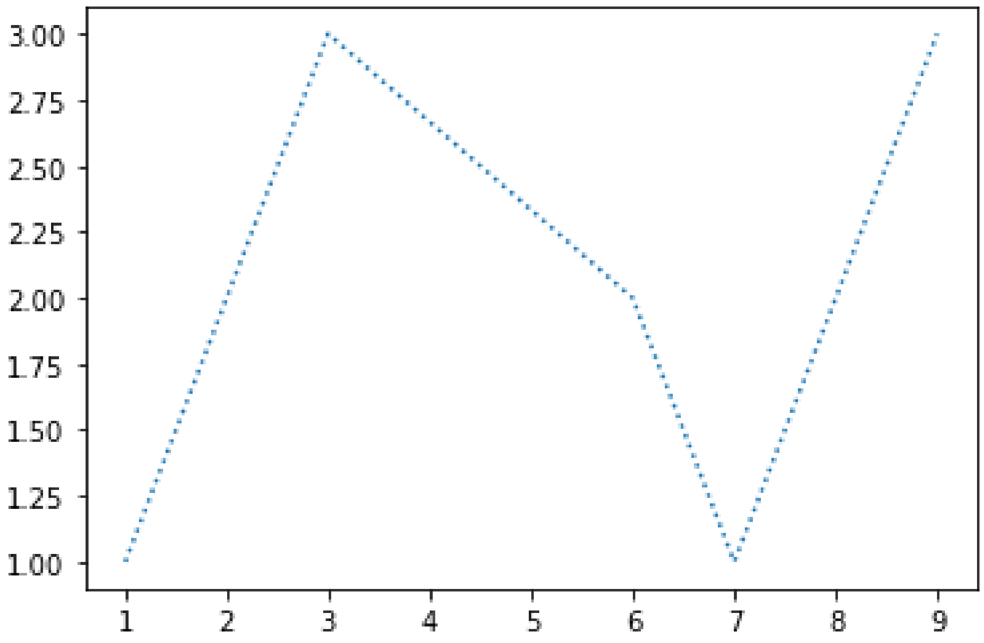
You can try some of the following line styles. The code string as well as the short code string can be used.

```
Line style      code       short code
-------------------------------------
Solid line     'solid'     '-'
Dashed line    'dashed'    '--'
Dotted line    'dotted'    ':'
Dash dot line  'dashdot'   '-.'
```

> Note: There are many more line styles available. See: https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/linestyles.html

[7]:
```
plt.plot(x, y, linestyle=':')
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f8182d5ac10>]
```



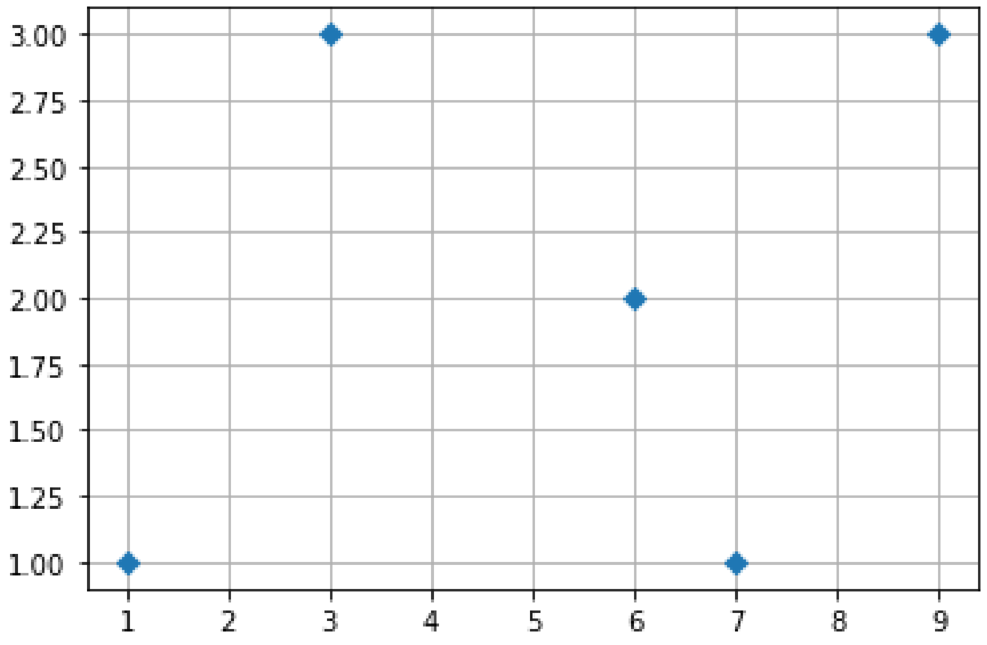
Let's switch from line to marker style.

```
Circle          'o'
Triangle up     '^'
Triangle down   'v'
Square          's'
Star            '*'
Plus            '+'
X               'x'
Diamond         'D'
Diamond thin    'd'
```

Note: There are many more markers available. See: https://matplotlib.org/3.3.3/api/markers_api.html

[8]:
```
plt.grid()

plt.plot(x, y, 'D')
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f8182dea070>]
```



That looks quite nice but we want to use another color, maybe red.

The following color abbreviations are available:

```
Color   code
--------------
black    'k'
white    'w'
red      'r'
green    'g'
blue     'b'
cyan     'c'
magenta  'm'
yellow   'y'
```

> Note: There are many more named colors available. See: https://matplotlib.org/3.1.0/gallery/color/named_colors.html

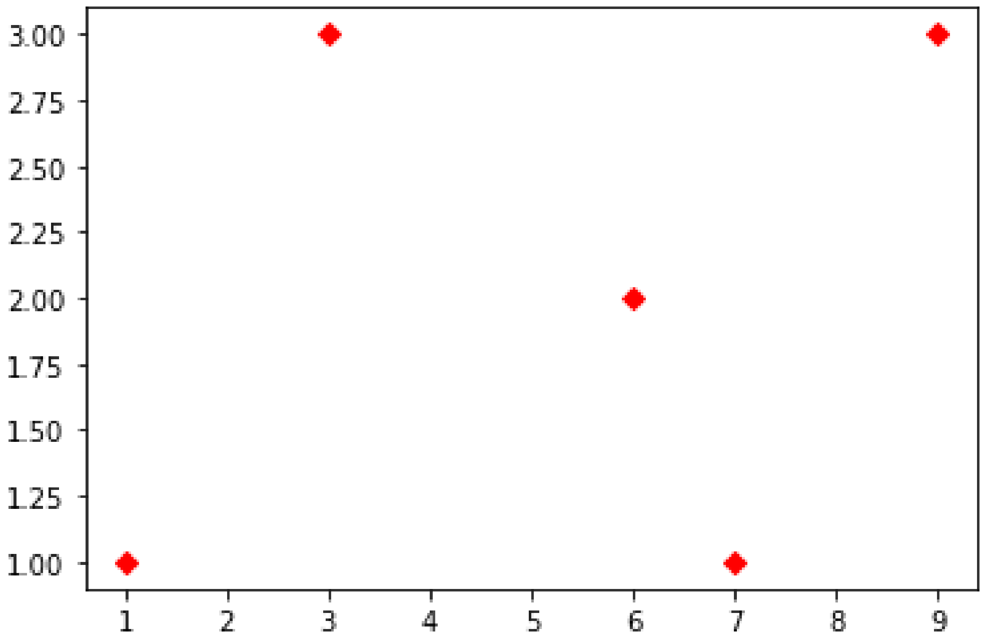
Matplotlib accepts combinations of multiple parameter abbreviations, e.g.

`'Dr'` is the same as `linestyle='none', marker='D', color='r'`

`plt.plot(x, y, 'Dr')` is the same as `plt.plot(x, y, linestyle='none', marker='D', color='r')`

[9]:
```
plt.plot(x, y, 'Dr')
#plt.plot(x, y, linestyle='none', marker='D', color='r')
```

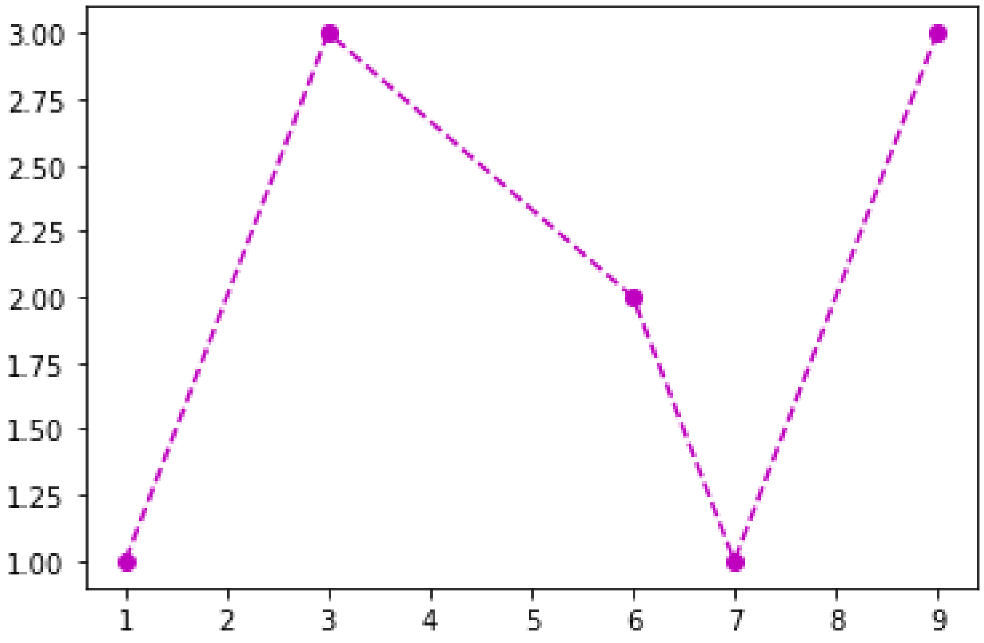
[9]:
```
[<matplotlib.lines.Line2D at 0x7f8182f1ddf0>]
```



Let's create a dashed line plot with circle markers in magenta.

```
[10]: plt.plot(x, y, '--om')
      #plt.plot(x, y, linestyle='--', marker='o', color='m')
```

```
[10]: [<matplotlib.lines.Line2D at 0x7f8182fa2490>]
```



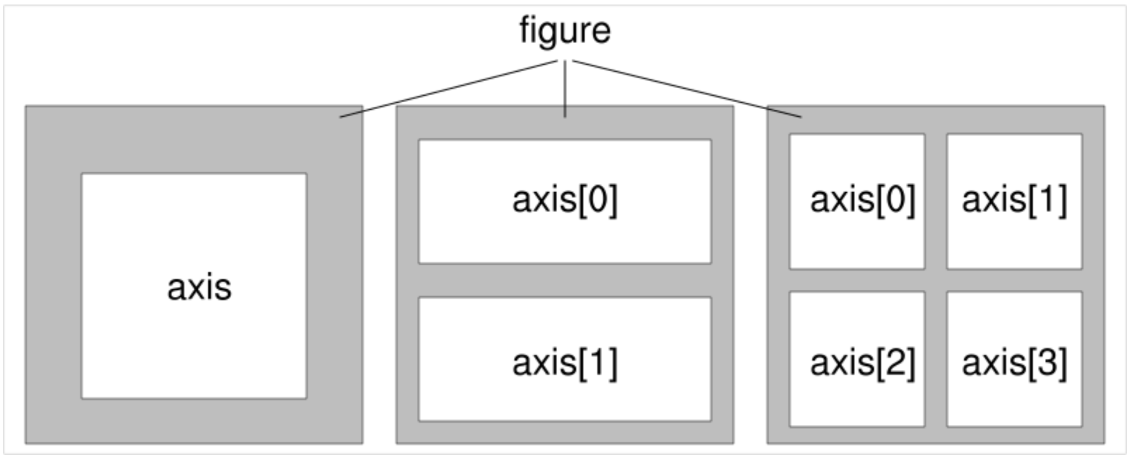
But of course you can still adjust the various settings manually, such as size, color, output resolution, line width, text settings, etc..

1.8 Figures and axes

To make the best use of Matplotlib, you should understand the base plot concept. Matplotlib provides two objects to control the plots

- figure object
- axis object

Figure can be considered as the workspace and axis as the actual plot. You can create a figure that contains one or more plots.



The left part of the image shows in gray a figure containing one plot (axis). In the middle we see in gray one figure containing two plots (axis with 2 elements) and on the right side we see in gray a figure containing four plots (axis with 4 elements). In the next section, the use of figure and axis will be shown in more detail.

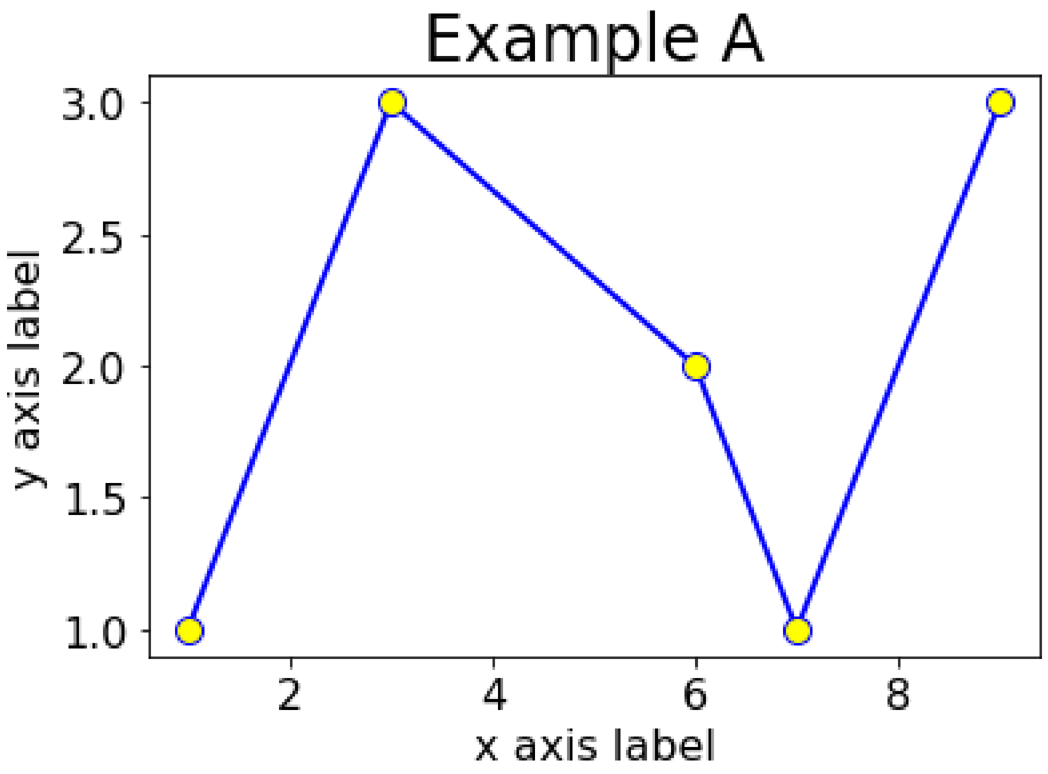
1.9 Single plot

It is very convenient to create figure and axis objects at once using the Matplotlib pyplot.subplots() function. Furthermore, further setting options for the axis are introduced, which are mostly self-explanatory.

The next example demonstrates the use of fig and axis for a plot where we also add a title and the x- and y-axis titles.

[11]:
```
fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Example A', fontsize=24)
ax.set_ylabel('y axis label', fontsize=16)
ax.set_xlabel('x axis label', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(x, y, linestyle="solid", linewidth=2, color="blue",
               marker="o", markersize="10", markerfacecolor="yellow")
```

```
[11]: [<matplotlib.lines.Line2D at 0x7f81830e9b50>]
```



1.10 Multiple plots

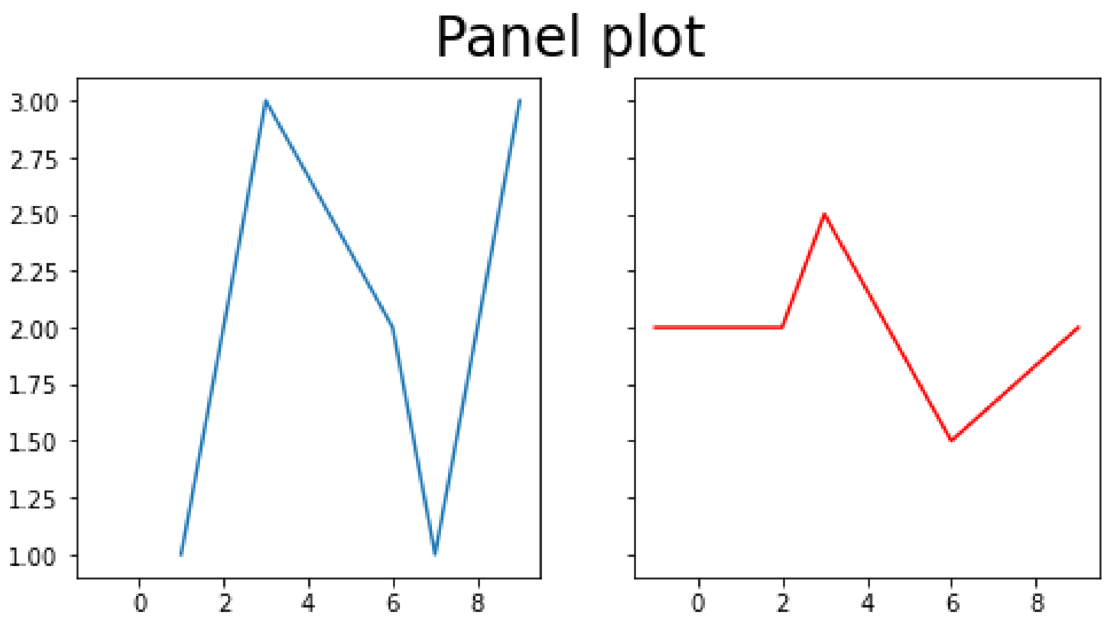
In the next section we explain how to use pyplot.subplots functionality to create two plots on one workspace.

1. create two plots in one row
2. use same x- and y-axis ranges for both plots
3. draw a common title

```
[12]: fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(8,4), sharex=True,␣
       ↪sharey=True)
      fig.suptitle('Panel plot', fontsize=24)
```

```
x2 = [-1, 2, 3,   6,   9]
y2 = [ 2, 2, 2.5, 1.5, 2]

axs[0].plot(x, y)
axs[1].plot(x2, y2, "r")
```

[12]:
```
[<matplotlib.lines.Line2D at 0x7f818320eeb0>]
```



1.11 Plot types

Line plots are a little bit boring that's why we change the plot type and see what we get.

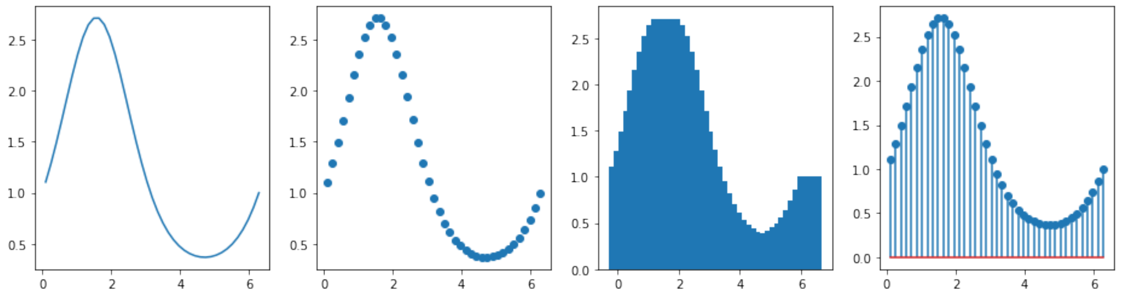
You can find the list of different pyplot functions at https://matplotlib.org/api/pyplot_summary.html

Next, we use arrays as input, which we create using the Numpy module (plots just looks nice ;)). Here are just a few examples of the different plot functions.

[13]:
```
import numpy as np

x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))

fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(16,4))

axs[0].plot(x, y)
axs[1].scatter(x, y)
```

```
axs[2].bar(x, y)
axs[3].stem(x, y)
```

```
[13]: <StemContainer object of 3 artists>
```



1.12 Text annotations

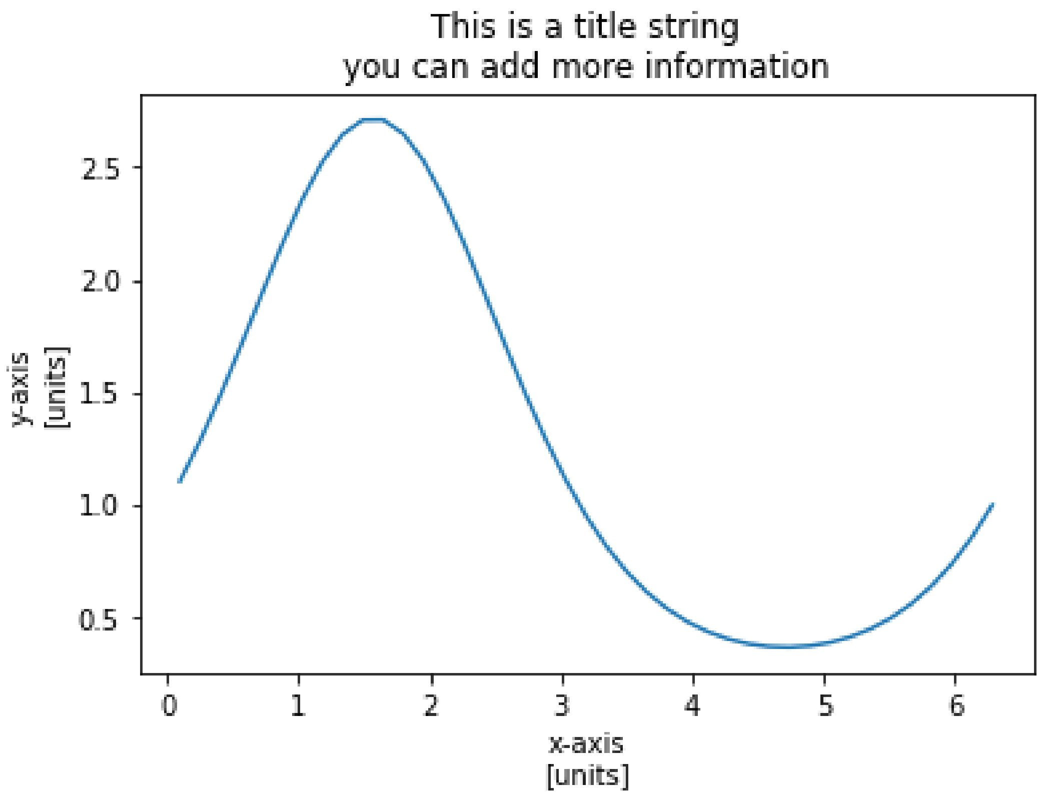
You can customize the titles and axis labels with the ax.set() method in one line.

The next example adds

1. a multiline title above the plot
2. multiline labels at x- and y-axis

```
[14]: fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

ax.set(title='This is a title string\nyou can add more information',␣
 ↪xlabel='x-axis\n[units]', ylabel='y-axis\n[units]')
```

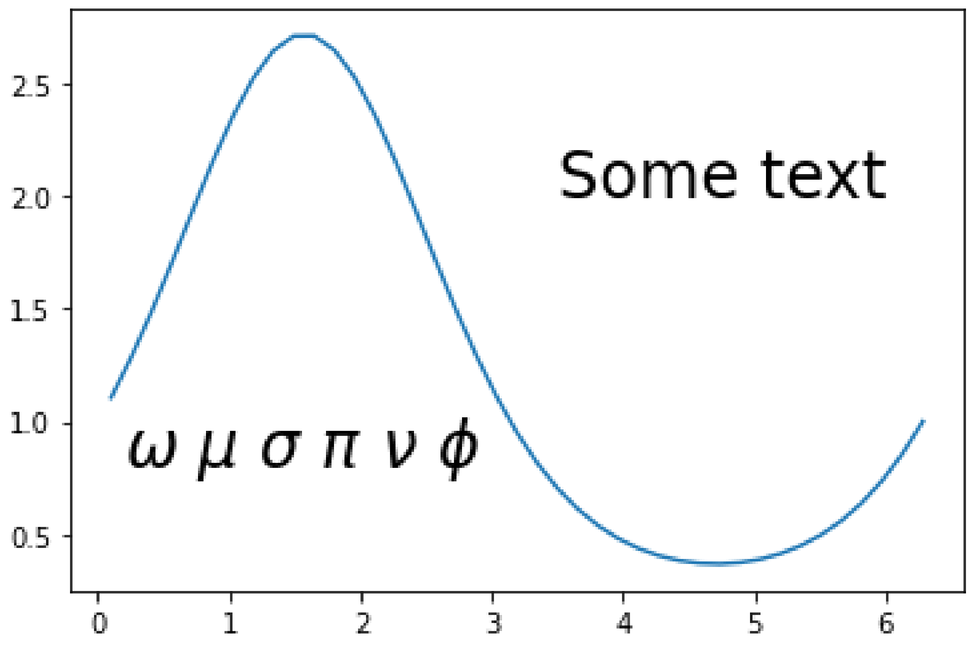
```
[14]: [Text(0.5, 1.0, 'This is a title string\nyou can add more information'),
       Text(0.5, 0, 'x-axis\n[units]'),
       Text(0, 0.5, 'y-axis\n[units]')]
```



You can also write additional text in a plot and of course change the font, e.g. Greek, and size.

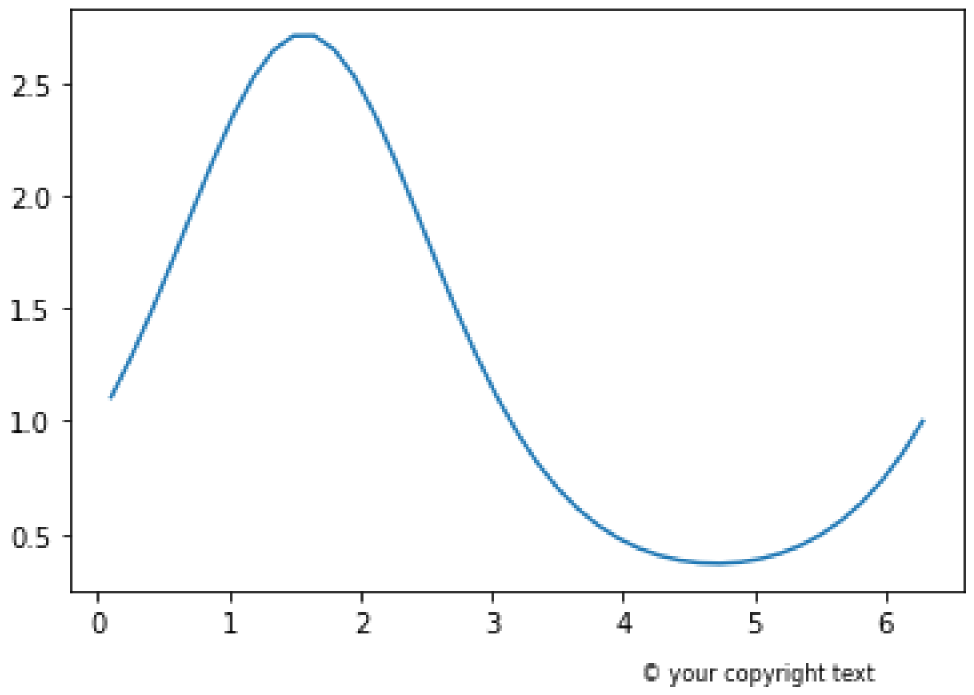
```
[15]: fig, ax = plt.subplots(figsize=(6,4))

      ax.plot(x, y)

      ax.text(3.5, 2.0, 'Some text',fontsize=24)
      ax.text(0.2, 0.8, r'$\omega\ \mu\ \sigma\ \pi\ \nu\ \phi$', fontsize=24)
```

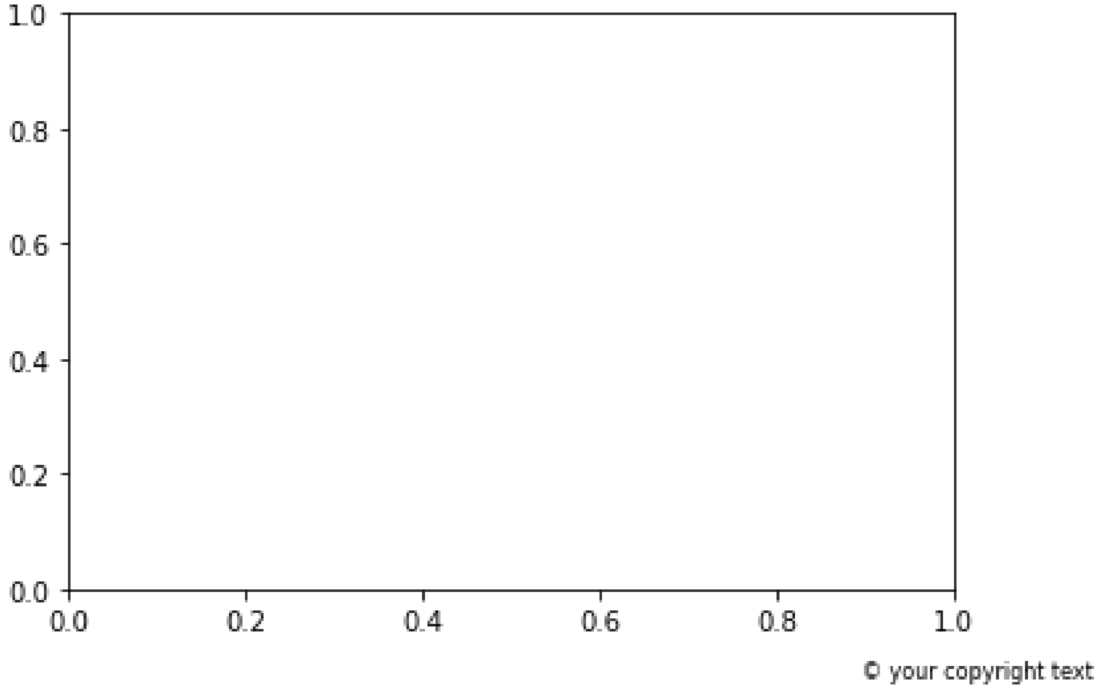
```
[15]: Text(0.2, 0.8, '$\\omega\\ \\mu\\ \\sigma\\ \\pi\\ \\nu\\ \\phi$')
```



We can also add text to the workspace outside the plot, e.g. a copyright text. The figure size will be extended when the text string won't fit and the plot size will be unchanged.

[16]:
```
fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y)

plt.text(0.62, 0.01, 'Âľ your copyright text', fontsize=8, transform=plt.gcf().
 ↪transFigure)
plt.show()
plt.text(0.82, 0.01, 'Âľ your copyright text', fontsize=8, transform=plt.gcf().
 ↪transFigure)
plt.show()
```


2.5
2.0
1.5
1.0
0.5
0
1
2
3
4
5
6
© your copyright text


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
© your copyright text

2 Plot settings

Next, as an example, we want to plot annual average temperature data from Germany (from DWD), coloring the data points according to their values, add s title string, and set the axis titles as well. To display the data as individual points we use the scatter function as shown above.

Used parameters of the scatter plot command:

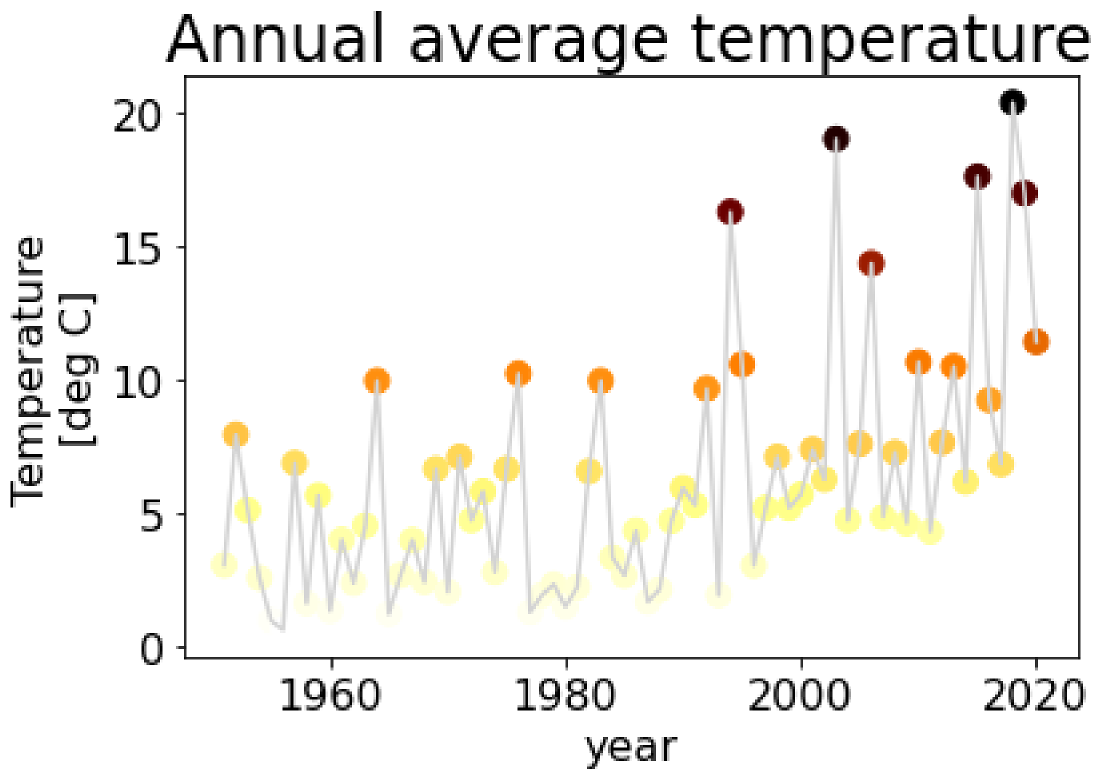
c=temp use variable temp for coloring the marker

cmap='afmhot_r' use colormap afmhot but reversed

s=80 increase the size of the markers

[17]:
```
years = np.arange(1951,2021)

temp = np.array([3.02, 7.91, 5.08, 2.53, 0.93, 0.58, 6.85, 1.58, 5.63, 1.30, 3.
 ↪97, 2.34, 4.50, 9.93, 1.18, \
                 2.64, 3.95, 2.38, 6.63, 2.01, 7.08, 4.70, 5.78, 2.75, 6.63, 10.
 ↪20, 1.25, 1.89, 2.29, 1.45, \
                 2.19, 6.54, 9.93, 3.30, 2.63, 4.31, 1.64, 2.10, 4.67, 5.91, 5.
 ↪27, 9.64, 1.89, 16.27, 10.54, \
                 3.00, 5.16, 7.08, 5.16, 5.67, 7.36, 6.23, 19.01, 4.70, 7.57, 14.
 ↪34, 4.82, 7.24, 4.58, 10.63, \
                 4.27, 7.62, 10.47, 6.14, 17.60, 9.21, 6.80, 20.37, 16.97, 11.39])

fig, ax = plt.subplots(figsize=(6,4))

ax.set_title('Annual average temperature', fontsize=24)
ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
ax.set_xlabel('year', fontsize=16)
ax.tick_params(labelsize=16)

ax.plot(years, temp, color='lightgray')
ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80)
```

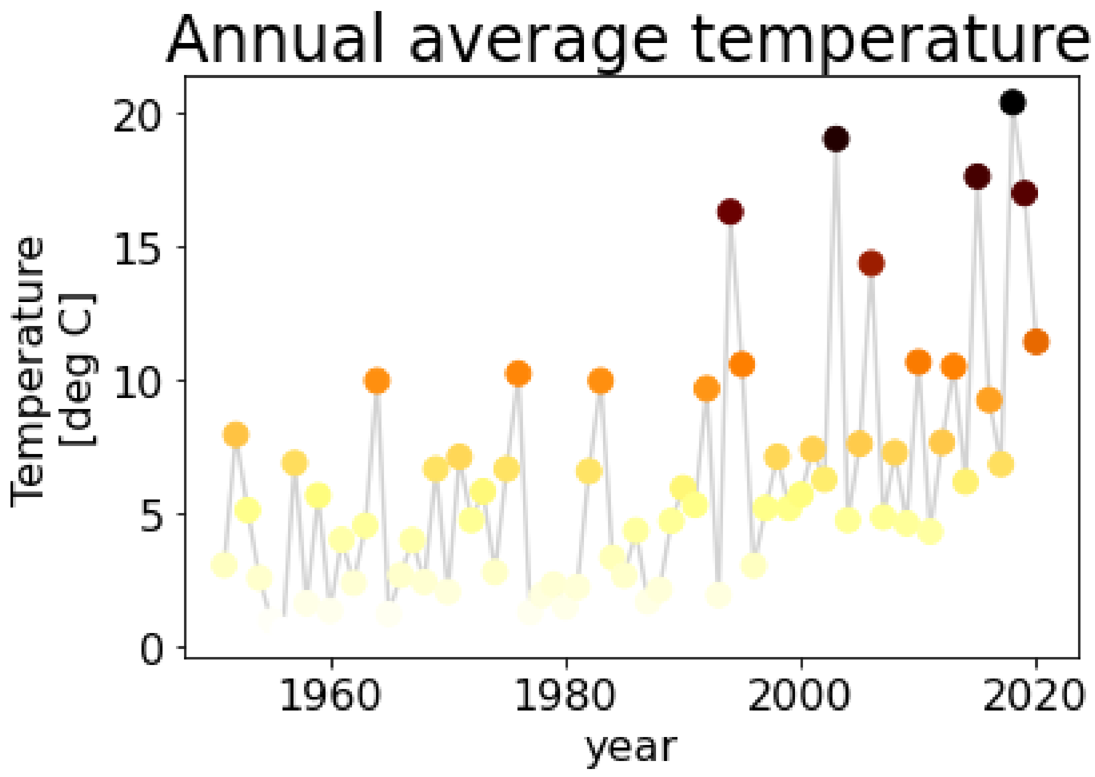
[17]:
```
<matplotlib.collections.PathCollection at 0x7f81843905e0>
```



Note: The draw order is first marker and than line, which doen't look quite nice. To change the draw order we can use the zorder parameter.

```
[18]: fig, ax = plt.subplots(figsize=(6,4))

      ax.set_title('Annual average temperature', fontsize=24)
      ax.set_ylabel('Temperature\n[deg C]', fontsize=16)
      ax.set_xlabel('year', fontsize=16)
      ax.tick_params(labelsize=16)

      ax.plot(years, temp, color='lightgray', zorder=1)
      ax.scatter(years, temp, c=temp, cmap='afmhot_r', s=80, zorder=2)
```

```
[18]: <matplotlib.collections.PathCollection at 0x7f81845a3940>
```



2.1 Save the plot to PNG

Since we do not only want to have the plots as an inline image, but also save them as a PNG file, we show here how to do that. Therefore, we add a line with pyplots function savefig call at the end of the script snippet.

```
[19]: plt.savefig("Germany_average_annual_temperature_1951-2020.png")
```

```
<Figure size 432x288 with 0 Axes>
```

```
[ ]:
```

```
[ ]:
```