TeleSoft

Intelligence everywhere...

MATPLOTLIB BASICS

Zephania Reuben

TeleSoft Ai

1.0 Introduction to Matplotlib

- **Matplotlib** is a Python package for 2D plotting that generates production-quality graphs. It supports interactive and non-interactive plotting, and can save images in several output formats (PNG, PS, and others).
- It provides a wide variety of plot types (lines, bars, pie charts, histograms, and many more). In addition to this, it is highly customizable, flexible, and easy to use.
- The dual nature of Matplotlib allows it to be used in both interactive and non-interactive scripts. It can be used in scripts without a graphical display, embedded in graphical applications, or on web pages. It can also be used interactively with the Python interpreter or IPython.

- **Matplotlib** was born in the scientific area of computing, where **gnuplot** and **MATLAB** were (and still are) used a lot.

- **Matplotlib** was modeled on **MATLAB**, because graphing was something that MATLAB did very well. The high degree of compatibility between them made many people move from **MATLAB** to **Matplotlib**.

But what are the points that built the success of Matplotlib?

- Let's look at some of them:
- **It uses Python:**
  - Python is a very interesting language for scientific purposes (it's interpreted, high-level, easy to learn, easily extensible, and has a powerful standard library) and is now used by major institutions such as NASA, JPL, Google, DreamWorks, Disney, and many more.
- **It's open source, so no license to pay:**
  - This makes it very appealing for professors and students, who often have a low budget.
- **It's very customizable and extensible:**
  - Matplotlib can fit every use case because it has a lot of graph types, features, and configuration options.
- **It's cross-platform and portable:**
  - Matplotlib can run on Linux, Windows, Mac OS X, and Sun Solaris (and Python can run on almost every architecture available).

TeleSoft i

Matplotlib dependencies

- Matplotlib has its origin in scientific fields, so it is commonly used to plot huge datasets. Python's native support for long lists becomes impractical for such sizes, so Matplotlib needs better support for arrays, so NumPy has to be available to use Matplotlib.
- **Build dependencies**
  - **Python:** Version (>=3.6).
  - **NumPy:** Version (>=1.11).
  - **libpng:** Version 1.2 or higher is needed to load or save PNG images (Windows users can skip this requirement).
  - **FreeType:** Version 2.3 or higher is needed for reading TrueType font files (Windows users can skip this requirement).

Installing Matplotlib

- **Installing Matplotlib on Linux**
  - In the following table, we will present some of the common Linux distributions package names for Matplotlib and the tools we can use to install the package:

Distribution	Package Name
Debian or Ubuntu (And other Debian derivatives)	sudo apt-get install python3-matplotlib
Fedora	sudo dnf install python3-matplotlib
Red Hat	sudo yum install python3-matplotlib
Arch	sudo pacman -S python-matplotlib

- **Installing Matplotlib on Windows**
- Open command line interface (CMD) use the next command to install **Matplotlib.**

```
python -m pip install matplotlib
```

Testing our installation

- To ensure we have correctly installed Matplotlib and its dependencies, a very simple test can be carried out in the following manner:

```
In[1]:  import matplotlib

In[2]:  print("Matplotlib version : ", matplotlib.__version__)

In[3]:  import numpy

In[4]:  print("Numpy version : ", numpy.__version__)
```

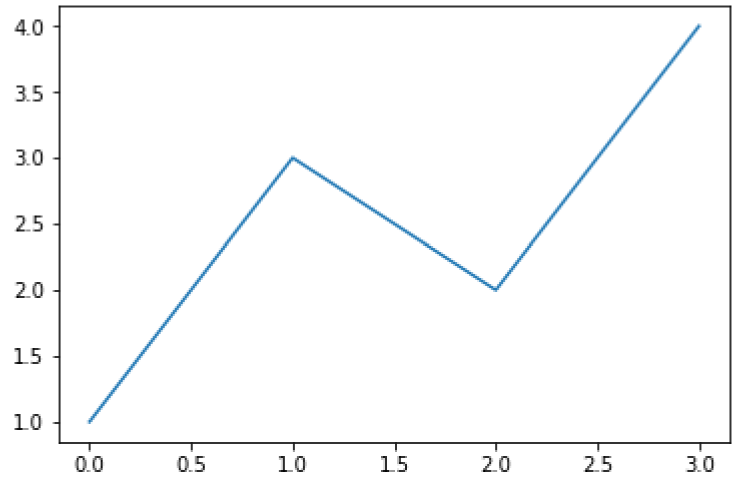
- If there is no errors then everything fine, we can dive in.

2.0 Getting started with Matplotlib

First plots with Matplotlib

```
In[1]:  import matplotlib.pyplot as plt

In[2]:  plt.plot([1, 3, 2, 4])

In[3]:  plt.show()
```

output:



TeleSoft Ai

- Let's look at each line of the previous code in detail:

```
In [1]: import matplotlib.pyplot  as plt
```

- This is the preferred format to import the main Matplotlib submodule for plotting, pyplot. It's the best practice and in order to avoid pollution of the global namespace, it's strongly encouraged to never import like:

```
In[1]:  from <module> import  *
```

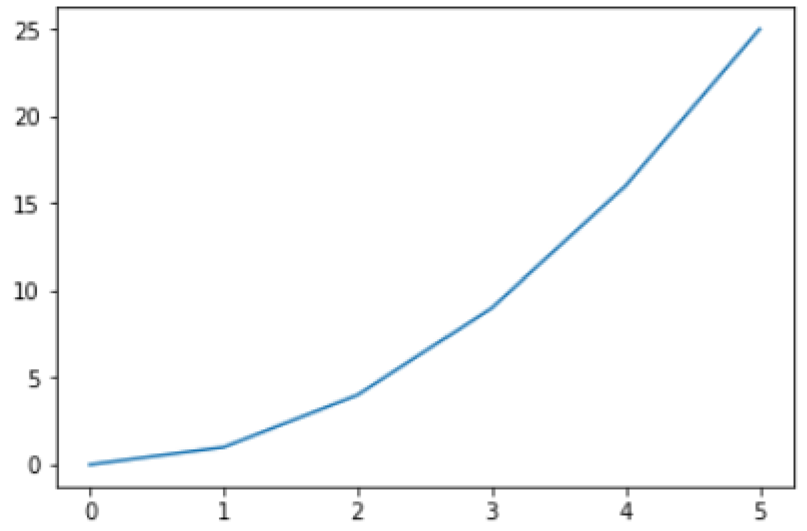
- This code line is the actual plotting command. We have specified only a list of values that represent the vertical coordinates of the points to be plotted. Matplotlib will use an implicit horizontal values list, from 0 (the first value) to N-1 (where N is the number of items in the list).

```
In[3] : plt.show()
```

- This command actually opens the window containing the plot image.
- Of course, we can also explicitly specify both the lists:

```
In[1]: import matplotlib.pyplot as plt

In[2]: x = range(6)

In[3]: plt.plot(x, [xi**2 for xi in x])

In[4]: plt.show()
```

output:



- We can start introducing the interaction with NumPy with one of its most used functions, arange(), and highlighting the difference with range():

i. range(i , j, k) is a Python built-in function that generates a sequence of integers from i to j with an increment of k (both, the initial value and the step are optional).

ii. arange(x ,y , z) is a part of NumPy, and it generates a sequence of elements (with data type determined by parameter types) from x to y with a spacing z (with the same optional parameters as that of the previous function).

- Let's use arange()

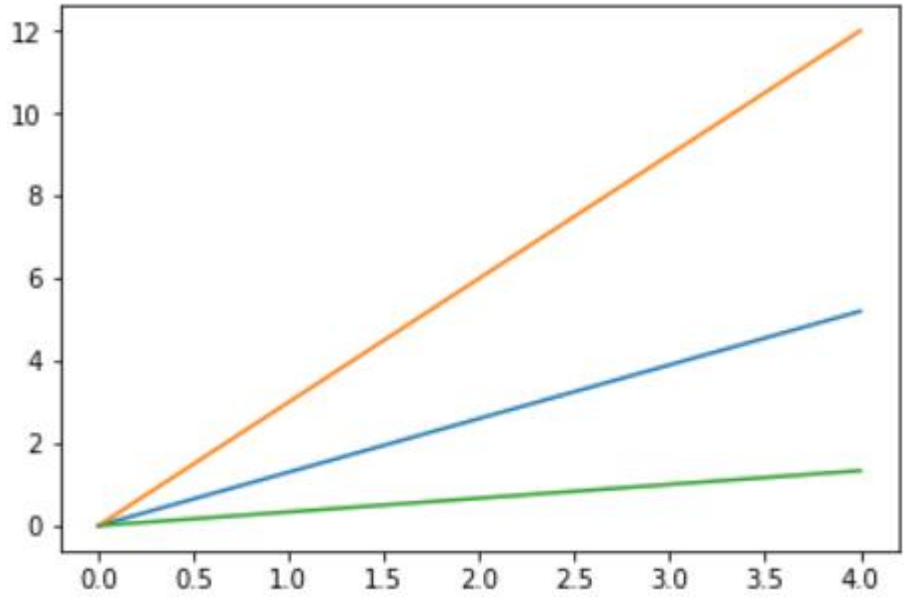
TeleSoft Ai

Multiline plots

- It's fun to plot a line, but it's even more fun when we can plot more than one line on the same figure. This is really easy with Matplotlib as we can simply plot all the lines that we want before calling show().

```
In[1]: import matplotlib.pyplot as plt
In[2]: import numpy as np
In[3]: x = np.arange(5)
In[4]: plt.plot(x,[y*1.3 for y in x])
In[5]: plt.plot(x,[y*3 for y in x])
In[6]: plt.plot(x,[y/3 for y in x])
 In[7]: plt.show()
```

output:



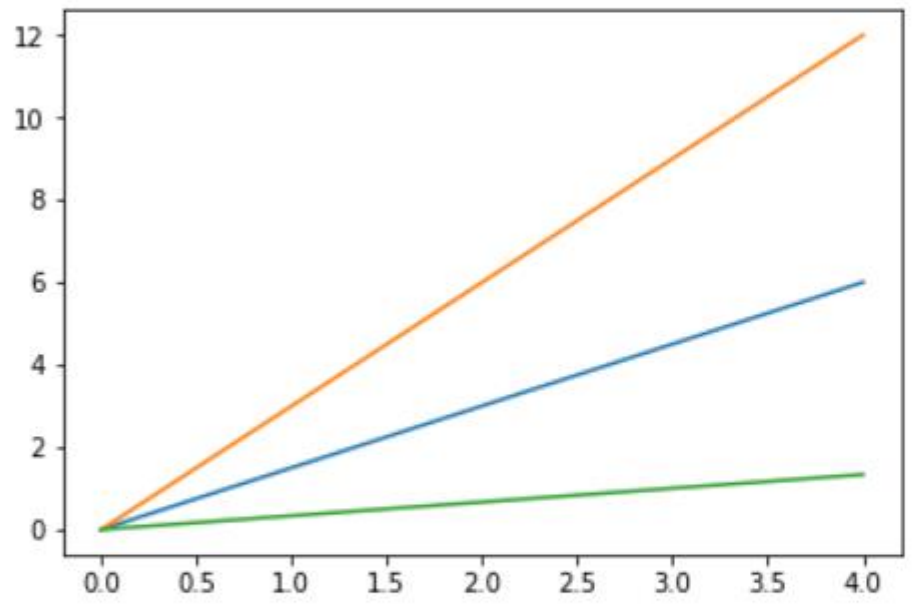
TeleSoft Ai

- **Note** how Matplotlib automatically chooses different colors for each line—green for the first line, blue for the second line, and red for the third one (from bottom to top).

plot() supports another syntax useful in this situation. We can plot multiline figures by passing the X and Y values list in a single plot() call.

```
In[1]: import matplotlib.pyplot as plt

In[2]: x = range(5)

In[3]: plt.plot(x, [xi*1.5 for xi in x], x, [xi*3.0 for xi in x], x,[xi/3.0 for xi in x])

In[4]: plt.show()
```

output:



Grid, axes, and labels

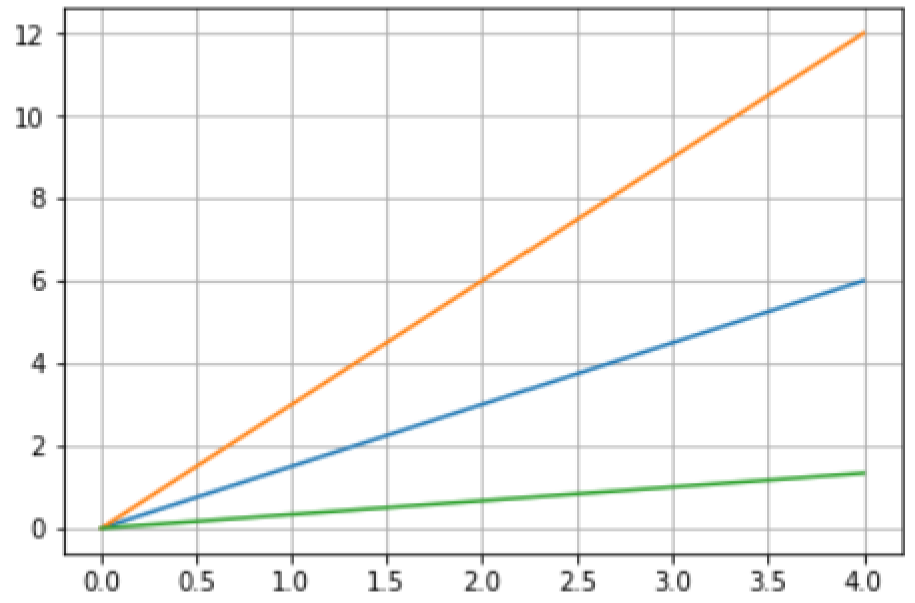
Adding grid

- In the previous images, we saw that the background of the figure was completely blank.

- While it might be nice for some plots, there are situations where having a reference system would improve the comprehension of the plot—for example with multiline plots. We can add a grid to the plot by calling the grid() function; -it takes one parameter, a Boolean value, to enable (if True) or disable (if False) the grid:

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: x = np.arange(5)

In[4]: plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)

In [5]: plt.grid(True)

In [6]: plt.show()
```

output:



TeleSoft Ai

Handling axes

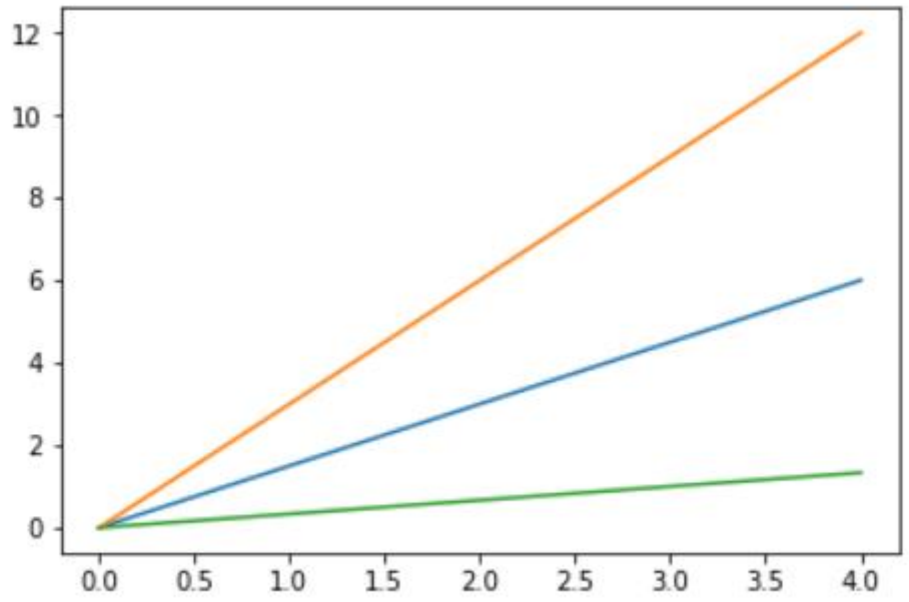
- You might have noticed that Matplotlib automatically sets the limits of the figure to precisely contain the plotted datasets.

- However, sometimes we want to set the axes limits ourselves (defining the scale of the chart). Let's take the first multiline plot.

- Wouldn't it be better to have more spaces between lines and borders? We can achieve this with the following code:

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: x = np.arange(5)

In[4]: plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)

In[5]: plt.axis() # shows the current axis limits values

Out[1]: (-0.2, 4.2, -0.6000000000000001, 12.6)
```

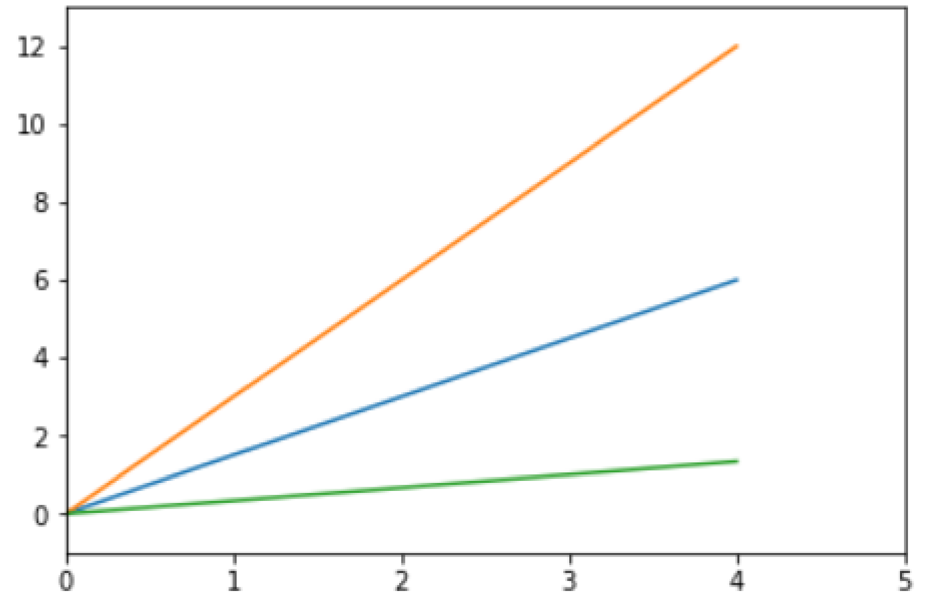
output:



TeleSoft Ai

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: x = np.arange(5)

In[4]: plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)

In[5]: plt.axis([0, 5, -1, 13]) # set new axes limits

In[6]: plt.show()
```

output:



TeleSoft Ai

- If we execute axis() without parameters, it returns the actual axis limits. There are two ways to pass parameters to axis(): by a list of four values or by keyword arguments.
- The list of values, that's the whole set of four values of keyword arguments [xmin, xmax, ymin, ymax], allows us to specify at the same time, the minimum and maximum limits respectively for the X-axis and the Y-axis. We can use the specific keyword arguments, for example:

```
plt.axis(xmin=NN, xmax=NN)
```

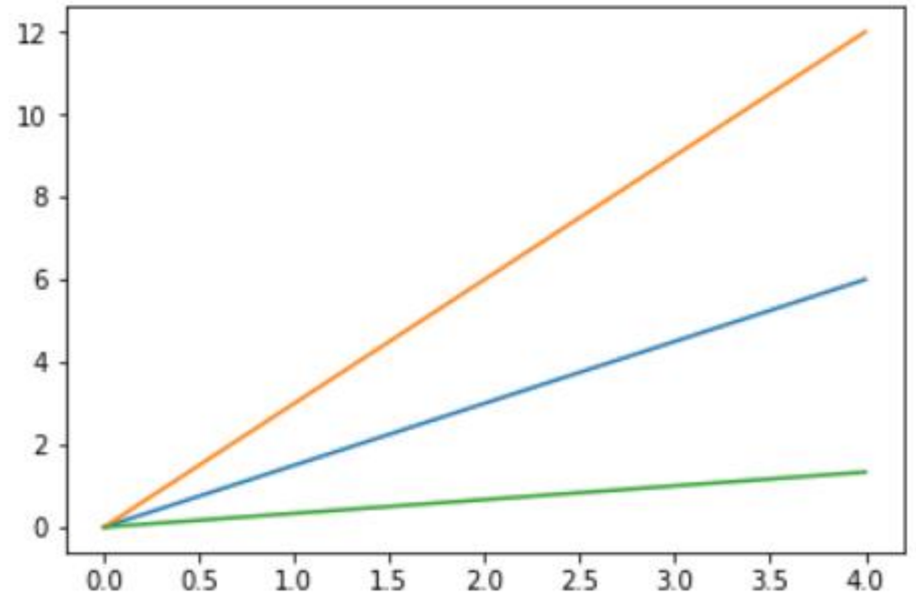
TeleSoftAi

- If we wish to set only some of these limits (in the previous line, we set only the minimum value for X-axis and the maximum value for Y-axis).

- We can also control the limits for each axis separately using xmin and xmax functions. Let's take the previous code before calling the axis() function, and change it in the following way:

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In][3]: x = np.arange(5)

In[4]: plt.plot(x, x*1.5, x, x*3.0, x, x/3.0)

In[5]: print(plt.xlim())

In[6]: print(plt.ylim())

Out[1]: (-0.2, 4.2)

Out[2]: (-0.6000000000000001, 12.6)
```

output:



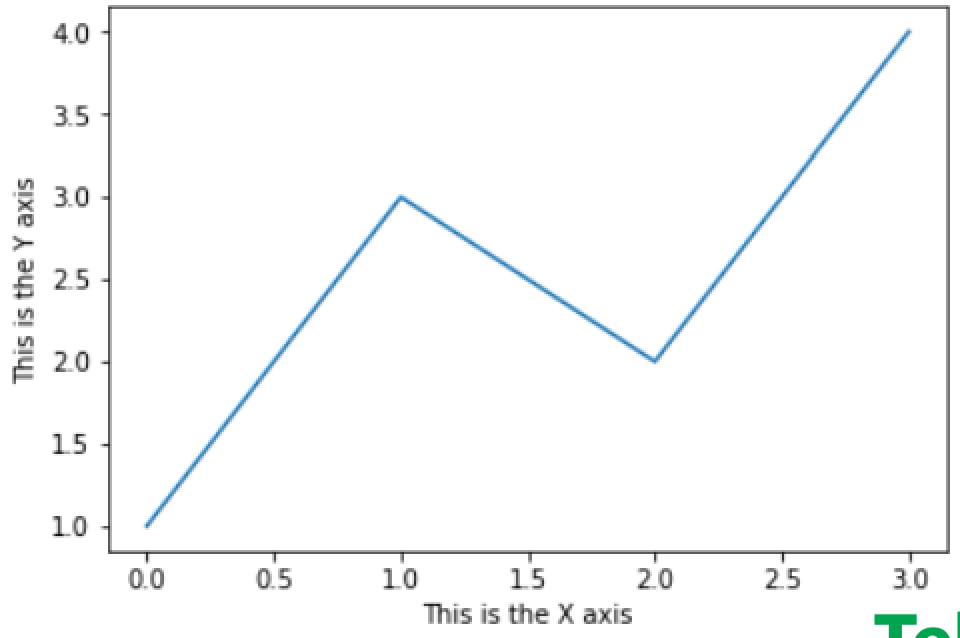
- Also for xlim() and ylim(), we can pass a list of two values (for example xlim([xmin,xmax]).

Adding labels

- Now that we know how to manage the axes dimensions, another important piece of information to add to a plot is the axes labels, since they usually specify what kind of data we are plotting.
- Then add the xlabel() and ylabel() functions:

```
In[1]: import matplotlib.pyplot as plt

In[2]: plt.plot([1, 3, 2, 4])

In[3]: plt.xlabel('This is the X axis')

In[4]: plt.ylabel('This is the Y axis')

In[5]: plt.show()
```

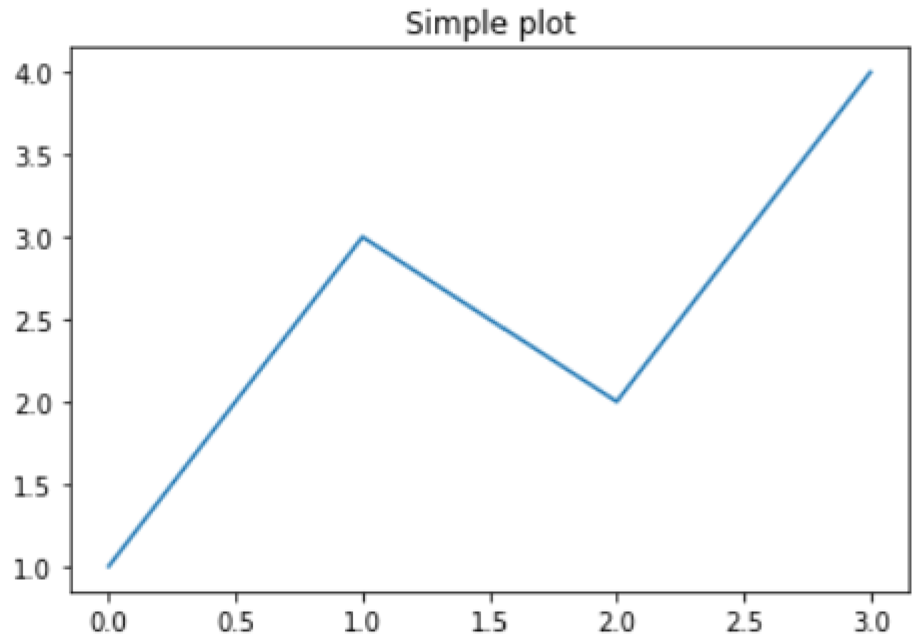
output



Titles and legends

- We are about to introduce two other important plot features , titles and legends.
- **Adding a title**
  - Just like in a book or a paper, the title of a graph describes what it is.
  - Matplotlib provides a simple function, to add a title to an image, as shown in the following code.

output:



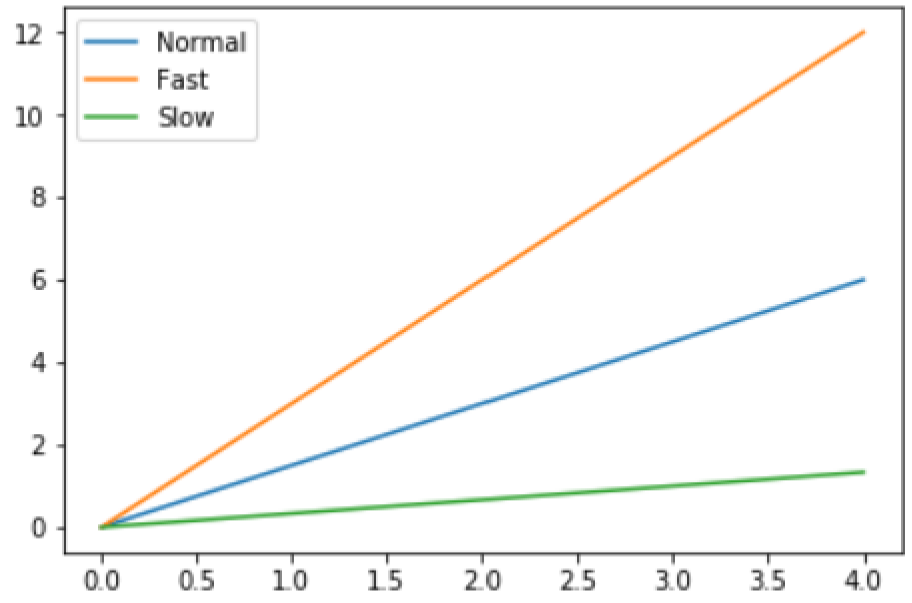
```
In[1]: import matplotlib.pyplot as plt

In[2]: plt.plot([1, 3, 2, 4])

In[3]: plt.title('Simple plot')

In[4]: plt.show()
```

Adding a legend

- The last thing we need to see to complete a basic plot is a legend.
- **Legends** are used to explain what each line means in the current figure. Let's take the multiline plot example again, and extend it with a legend:

```
In[1]: import matplotlib.pyplot as plt
In[2]: import numpy as np
In[3]: x = np.arange(5)
In[4]: plt.plot(x, x*1.5, label='Normal')
In[5]: plt.plot(x, x*3.0, label='Fast')
In[6]: plt.plot(x, x/3.0, label='Slow')
In[7]: plt.legend()
In[8]: plt.show()
```

output:



- The legend() function allows us to select several locations, which can be specified as an optional argument (or with the keyword argument, loc).
- The following table gives us the various positions at which the legend could be placed along with the equivalent codes for these positions.

String	Code	String	Code
best	0	center left	6
upper right	1	center right	7
upper left	2	Lower center	8
lower left	3	Upper center	9
lower right	3	center	10
right	5		

- An interesting functionality is the auto-legend positioning, setting loc=‘best’, Matplotlib automatically tries to find the optimal legend position.

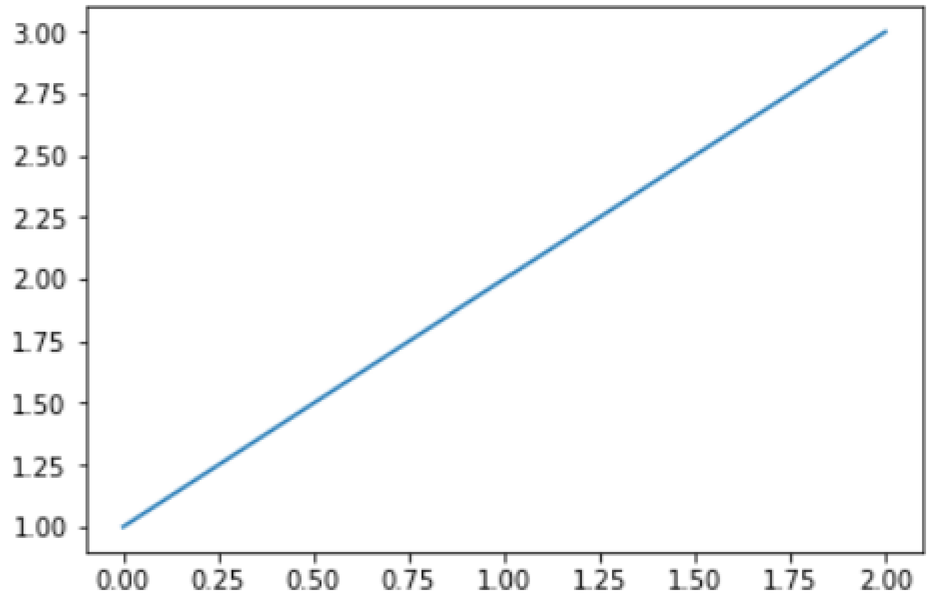
- Another nice argument we'd like to mention is ncol; this argument specifies how many columns to use to layout the legend items.

Saving plots to a file

- Saving a plot to a file is an easy task. The following example shows how:

```
In[1]: import matplotlib.pyplot as plt

In[2]: plt.plot([1, 2, 3])

In[3]: plt.savefig('plot123.png')
```

output:



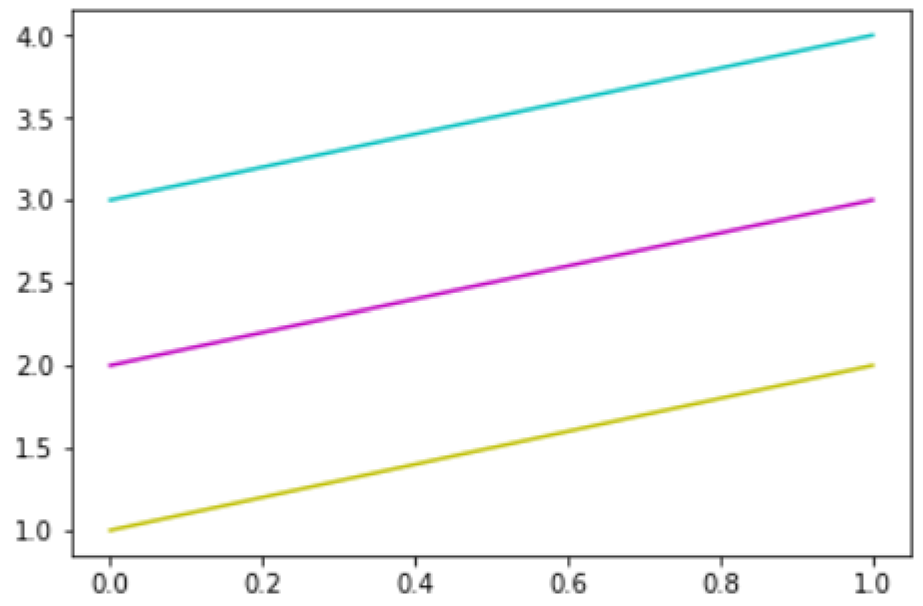
3.0 Decorate Graphs with Plot Styles and Types

Control colors

- We've already seen that in a multiline plot, Matplotlib automatically chooses different colors for different lines. We are also free to choose them by ourselves:

output:

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: y = np.arange(1, 3)

In[4]: plt.plot(y, 'y');

In[5]: plt.plot(y+1, 'm');

In[6]: plt.plot(y+2, 'c');

In[7]: plt.show()
```



TeleSoft Ai

- Here is a table of the abbreviations used to select colors:

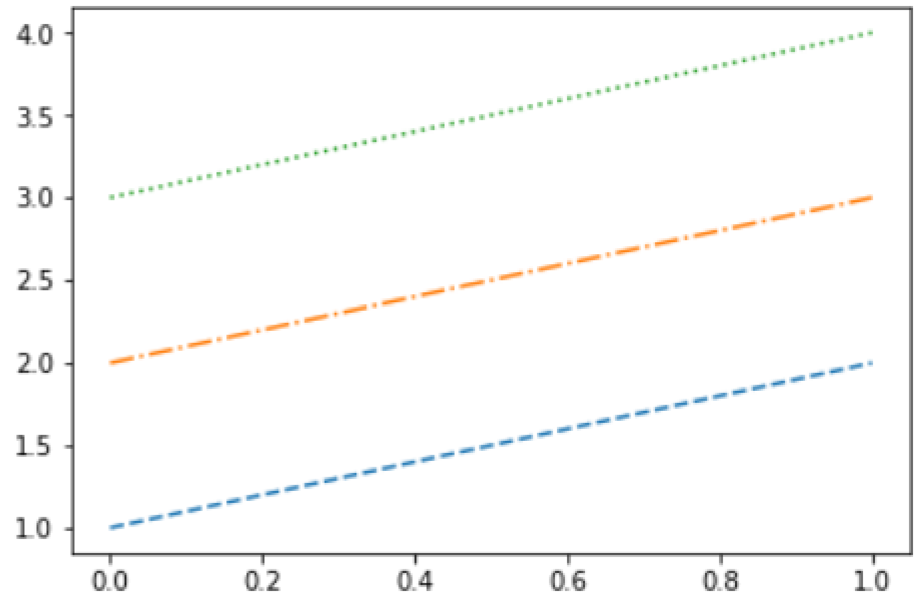
Color abbreviation	Color name
b	blue
c	cyan
g	green
r	red
m	magenta
y	yellow
k	black
w	white

Control line styles

- All the lines seen until now are proper ones without any dots or dashes. Matplotlib allows us to use different line styles, for example:

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: y = np.arange(1, 3)

In[4]: plt.plot(y,'--', y+1, '-.', y+2, ':');

In[5]: plt.show()
```

output:



TeleSoft Ai

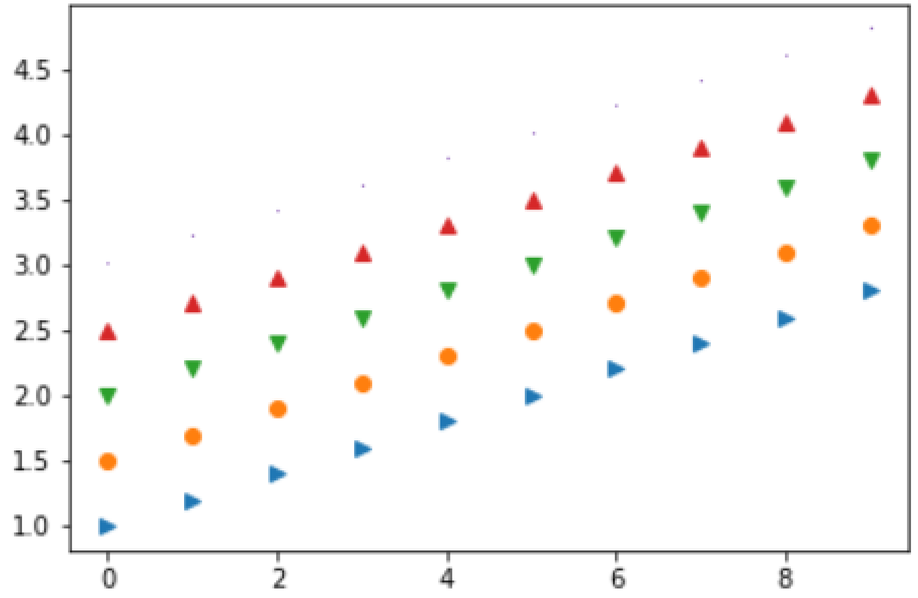
Control marker styles

- Markers are, by default, drawn as point markers. They are just a location on the figure where segments join.
- Matplotlib provides a lot of customization options for markers. The following table contains a list of the some styles:

Marker abbreviation	Marker style
.	Point marker
,	Pixel maker
o	Circle marker
v	Triangle down marker
^	Triangle up marker
>	Triangle right marker
<	Triangle left marker

- Let's look at some of them:

```
In[1]: import matplotlib.pyplot as plt                                    :

In[2]: import numpy as np

In[3]: y = np.arange(1, 3, 0.2)

In[4]: plt.plot(y, '>', y+0.5, 'o', y+1, 'v', y+1.5, '^', y+2, ',');

In[5]: plt.show()                  output:
```



TeleSoft△i

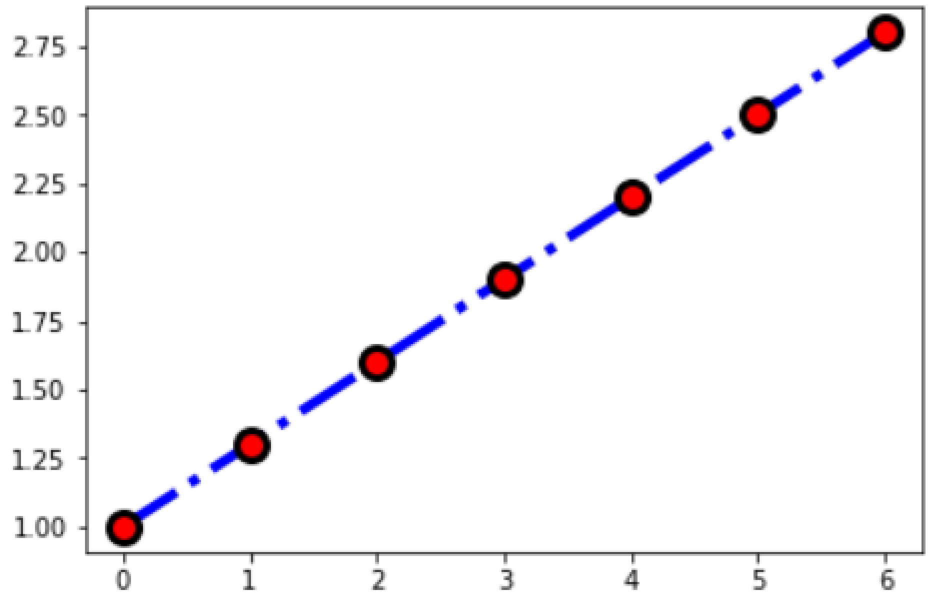
Control marker styles

- Format strings are really useful, but they have some drawbacks. For example, they don't allow us to specify different colors for lines and markers, as we saw in the previous example, plot() is a really rich function, and there are some keyword arguments to configure colors, markers, and line styles:

Keyword argument	Description
color or c	Sets the color of the line; accepts any Matplotlib color format.
linestyle	Sets the line style; accepts the line styles seen previously.
linewidth	Sets the line width; accepts a float value in points.
marker	Sets the line marker style.
markeredgecolor	Sets the marker edge color; accepts any Matplotlib color format.
markeredgewidth	Sets the marker edge width; accepts float value in points.
markerfacecolor	Sets the marker face color; accepts any Matplotlib color format.
marker	Sets the marker size in points; accepts float values.

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: y = np.arange(1, 3, 0.3)

In[4]: plt.plot(y, color='blue', linestyle='dashdot', linewidth=4,
marker='o', markerfacecolor='red', markeredgecolor='black',
markeredgewidth=3, markersize=12)

 In[5]: plt.show()
```

output:



TeleSoft

Handling X and Y ticks

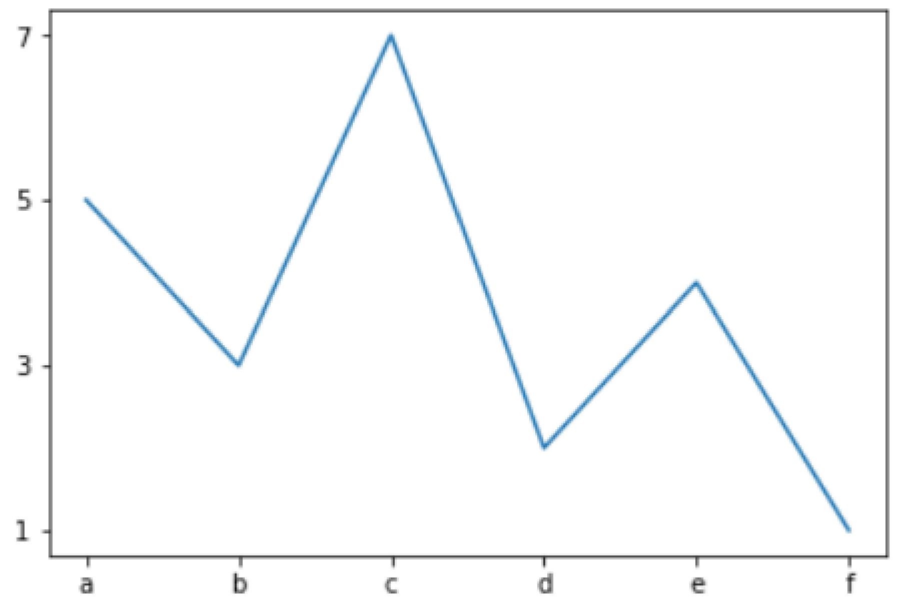
- Matplotlib provides two basic functions to manage them— xticks() and yticks(). They behave in the same way, so the description for one function will apply to the other too.
- Executing with no arguments, the tick function returns the current ticks' locations and the labels corresponding to each of them:

  locs, xlabel = xticks()

- The arguments (in the form of lists) that we can pass to the function are:
  - **Locations of the ticks**
  - **Labels to draw at these locations (if necessary)**
- Let's try to explain it with an example:

```
In[1]: import matplotlib.pyplot as plt

In[2]: x = [5, 3, 7, 2, 4, 1]

In[3]: plt.plot(x)

In[4]: plt.xticks(range(len(x)), ['a', 'b', 'c', 'd', 'e', 'f'])

 In[5]: plt.yticks(range(1, 8, 2))

 In[6]: plt.show()
```

output:



TeleSoft Ai

Plot types

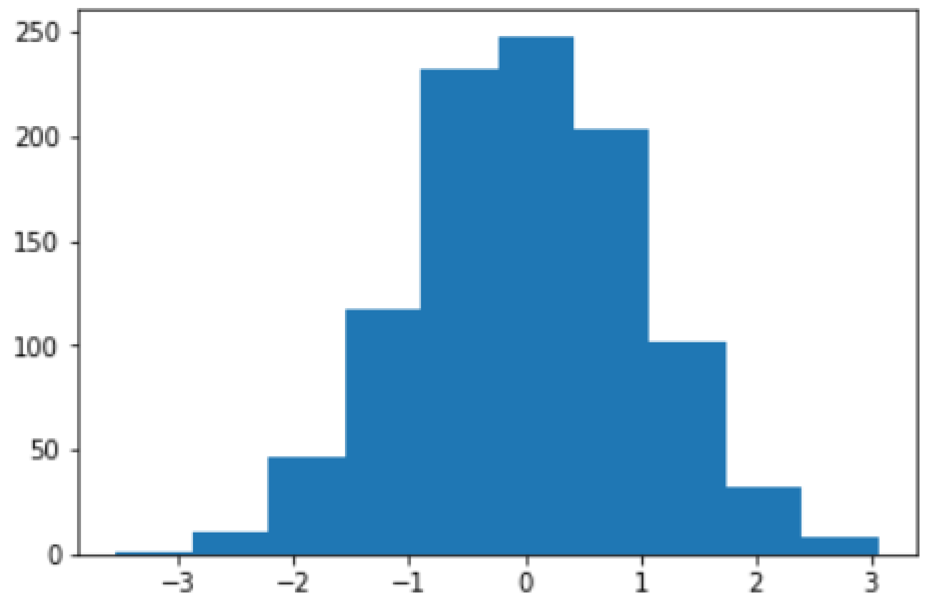
Histogram charts

- Histogram charts are a graphical display of frequencies, represented as bars.

- They show what portion of the dataset falls into each category, usually specified as non-overlapping intervals. Matplotlib calls those categories bins.

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]:  y = np.random.randn(1000)

In[4]: plt.hist(y)

In[5]: plt.show()
```

output:



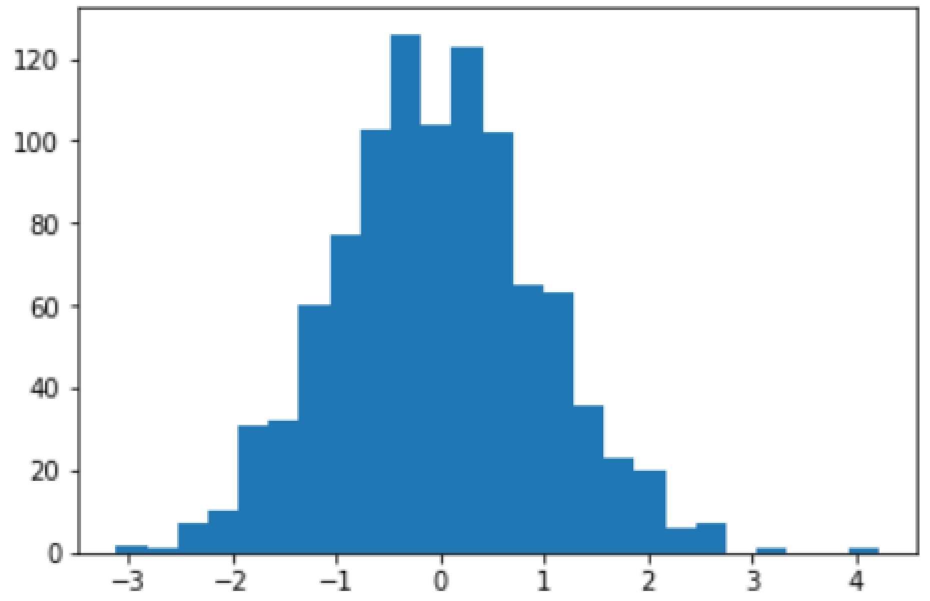
TeleSoft∆i

- Histogram plots group up values into bins of values. By default, hist() uses bin a value of 10 (so only ten categories, or bars, are computed), but we can customize it, either by passing an additional parameter, for example, in hist(y,) or hist(y,bins=)

- Replotting the previous dataset, but with bins=25.

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: y = np.random.randn(1000)

In[4]: plt.hist(y,bins=25)

In[5]: plt.show()
```

output:



TeleSoft Ai

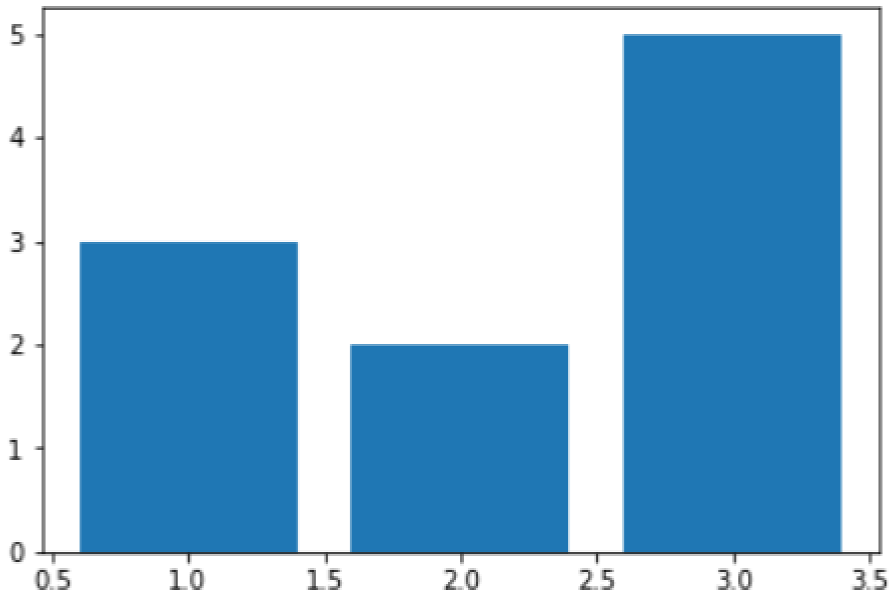
Bar charts

- Bar charts display rectangular bars (either vertical or horizontal) with their length proportional to the values they represent. They are commonly used to visually compare two or more values.

- The bar() function is used to generate bar charts in Matplotlib. The function expects two lists of values: the X coordinates that are the positions of the bar's left margin and the heights of the bars:

```
In[1]:import matplotlib.pyplot as plt

In[2]: plt.bar([1, 2, 3], [3, 2, 5])

In[3]: plt.show()
```

output:



Pie charts

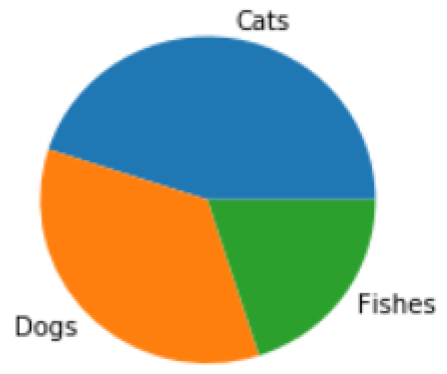
- Pie charts are circular representations, divided into sectors (also called wedges). The arc length of each sector is proportional to the quantity we're describing.
- It's an effective way to represent information when we are interested mainly in comparing the wedge against the whole pie, instead of wedges against each other.
- Matplotlib provides the pie() function to plot pie charts from an array X. Wedges are created proportionally, so that each value x of array X generates a wedge proportional to x/sum(X).

- Please note that if sum(X) is less than 1, then the pie is drawn using X values directly and no normalization is done, resulting in a pie with discontinuity.

- In the next example, we are going to plot a simple pie, using the legend keyword argument to give names to the wedges:

```
In[1]: import matplotlib.pyplot as plt

In[2]: plt.figure(figsize=(3,3))

In[3]: x = [45, 35, 20]

In[4]: labels = ['Cats', 'Dogs', 'Fishes']

In[5]: plt.pie(x, labels=labels);

In[6]: plt.show()
```

output:

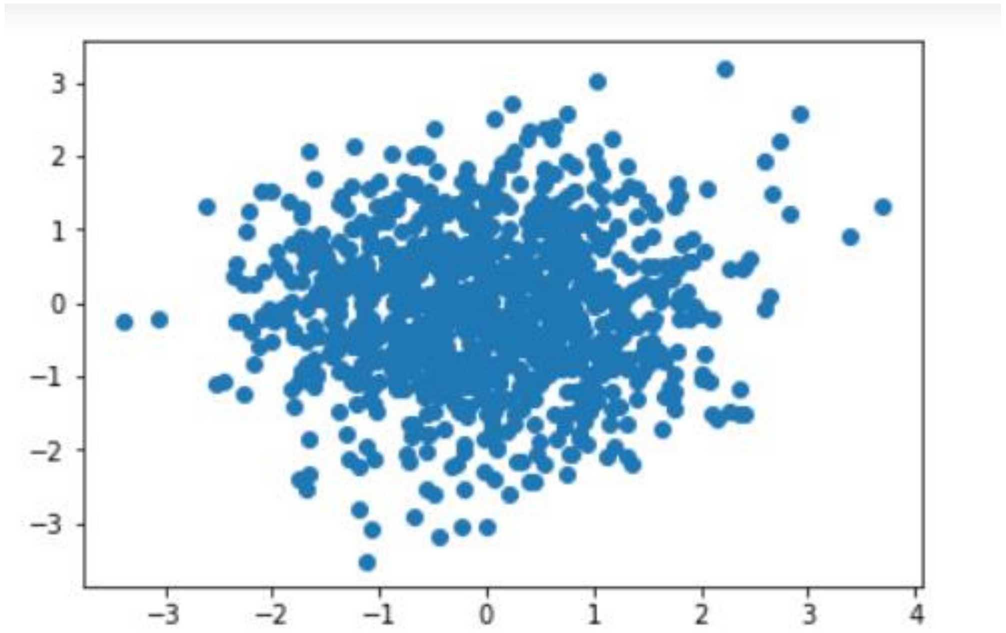


Scatter plots

- Scatter plots display values for two sets of data. The data visualization is done as a collection of points not connected by lines. Each of them has its coordinates determined by the value of the variables (one variable determines the X position, the other the Y position).
- A scatter plot is often used to identify potential association between two variables, and it's often drawn before working on a fitting regression function. It gives a good visual picture of the correlation, in particular for nonlinear relationships.
- Matplotlib provides the scatter() function to plot X versus Y unidimensional array of the same length as scatter plot.

```
In[1]: import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: x = np.random.randn(1000)

In[4]: y = np.random.randn(1000)

In[5]: plt.scatter(x, y);

In[6]: plt.show()
```

output:



TeleSoft Ai

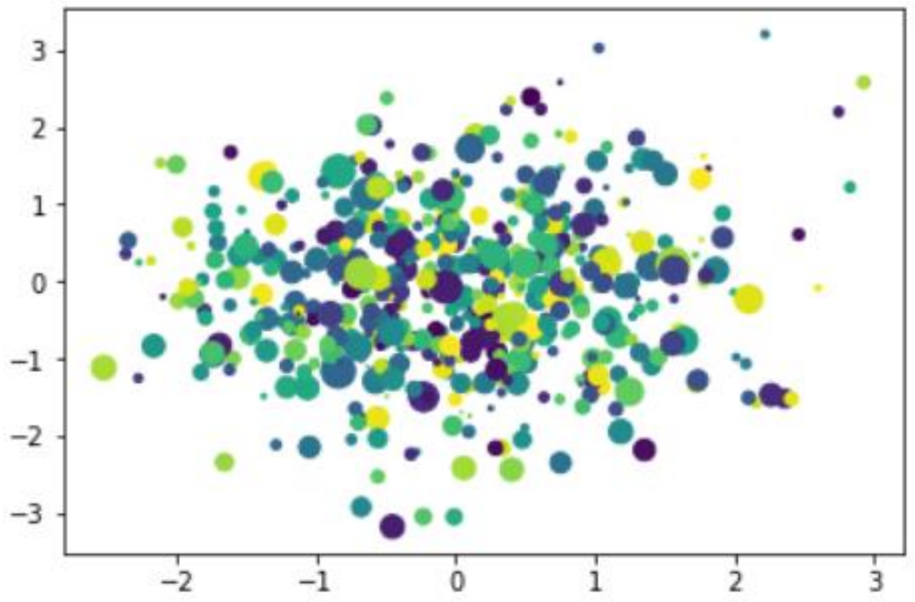
- We can decorate the chart by using some of the following keyword arguments:
- **s**: This stands for the size of the markers in pixel*pixel. It can be a single value (to be used for all the points) or an array of the same size of X and Y (so that each point will have its own size).
- **c**: This is the points color. It can be a single value or a list of colors (that will be cycled on the points plotted) eventually of the same size of X and Y. The values can be the Matplotlib color codes or even numbers mapped to colors using color maps.
- **marker**: This specifies the marker to use to plot the points;
- the available values are:

Marker value	Description
s	Square
o	Circle
^	Triangle up
v	Triangle down
>	Triangle right
<	Triangle left
d	Diamond
p	Pentagon
h	Hexagon
8	octagon
+	Plus
x	cross

- We can now apply some of them to the next example, where we specify a different size and color for each point of the previous dataset.

```
In[1]:import matplotlib.pyplot as plt

In[2]: import numpy as np

In[3]: size = 50*np.random.randn(1000)

In[4]: colors = np.random.rand(1000)

In[5]: plt.scatter(x, y, s=size, c=colors)

In[6]: plt.show()
```

output:



References

- Matplotlib Release 2.2.2, John Hunter, Darren Dale, Eric Firing, Michael Droettboom,March,2018.
- Matplotlib tutorial,Nicolas P. Rougier - Euroscipy 2012 – Prace,2013