University of Sulaimani
College of Science
Computer Department
4th Stage

Data Science Management

Visualization and Matplotlib in Python

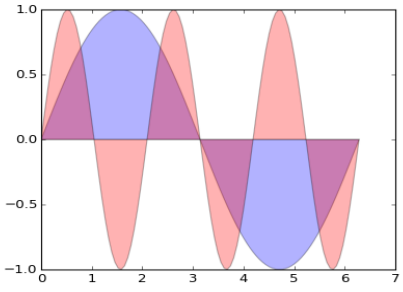
Class 6

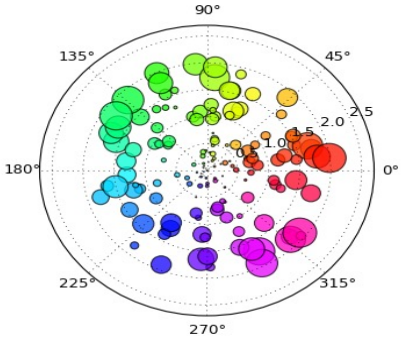
Theoretical and practical lectures

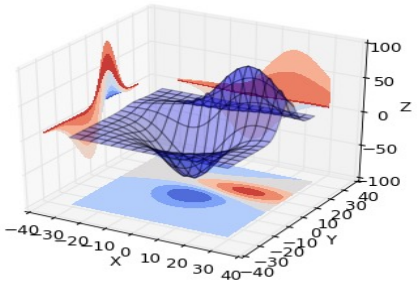
Assist. Prof. Dr. Miran Taha Abdullah
2025-2026

What is data visualization?

- Data visualization is the graphical representation of information and data.
  - Can be achieved using visual elements like figures, charts, graphs, maps, and more.
- Data visualization tools provide a way to present these figures and graphs.
- Often, it is essential to analyze massive amounts of information and make data-driven decisions.
  - converting complex data into an easy to understand representation.







Matplotlib

- **Matplotlib** is one of the most powerful tools for data visualization in Python.
- **Matplotlib** is an incredibly powerful (and beautiful!) 2-D plotting library.
  - It is easy to use and provides a huge number of examples for tackling unique problems
- In order to get **`matplotlib`** into your script,
  - first you need to import it, for example:

```
import matplotlib.pyplot as plt
```

- However, if it is not installed, you may need to install it:
  - Easiest way to install **matplotlib** is using **`pip`**.
  - Type the following command in the command prompt (cmd) or your Linux shell;
    - **`pip install matplotlib`**
    - *Note that you may need to run the above cmd as an administrator*

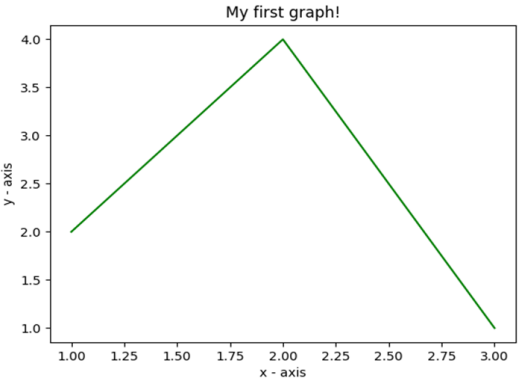
Matplotlib

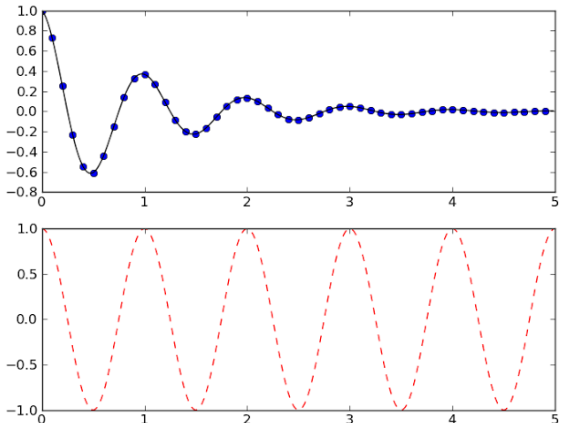
- Strives to emulate MATLAB
  - **`matplotlib.pyplot`** is a collection of command style functions that make **`matplotlib`** work like **MATLAB**.
- Each **`pyplot`** function makes some change to the figure:
  - e.g.,
    - creates a figure,
    - creates a plotting area in the figure,
    - plots some lines in the plotting area,
    - decorates the plot with labels, etc.
- **Note** that **various states** are preserved **across function calls**

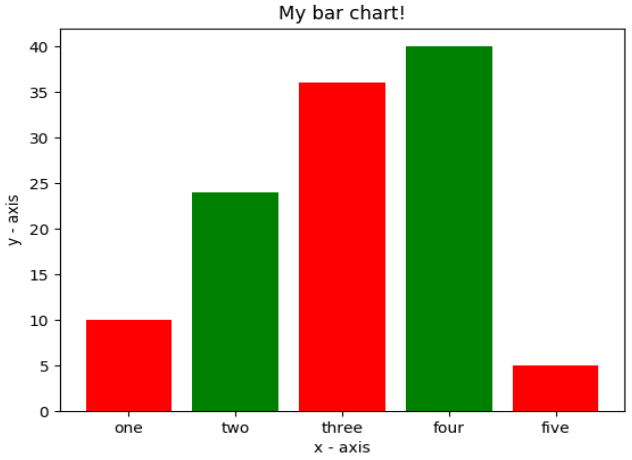
- Whenever you plot with **`matplotlib`**, the two main code lines should be considered:
  - Type of graph
    - this is where you define a bar chart, line chart, etc.
  - Show the graph
    - this is to display the graph

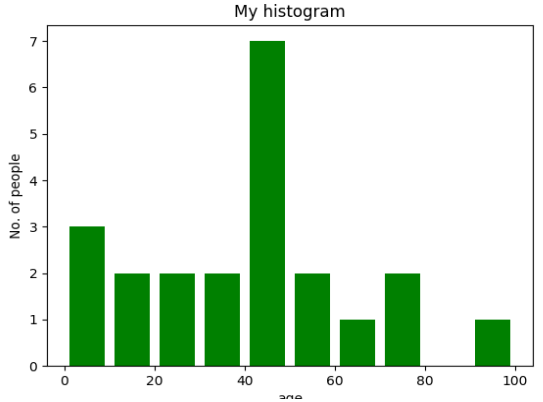
E.g. Matplotlib

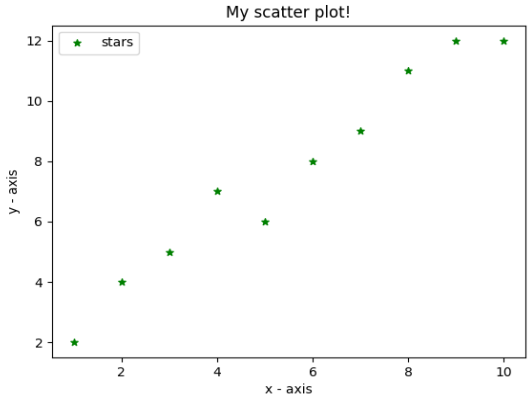
- **`Matplotlib`** allows you to make easy things
- You can generate **plots**, **histograms**, **power spectra**, **bar charts**, **errorcharts**, **scatterplots**, etc., with just a few lines of code.

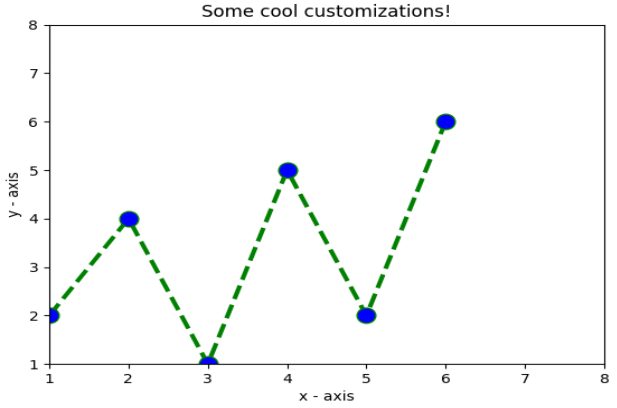


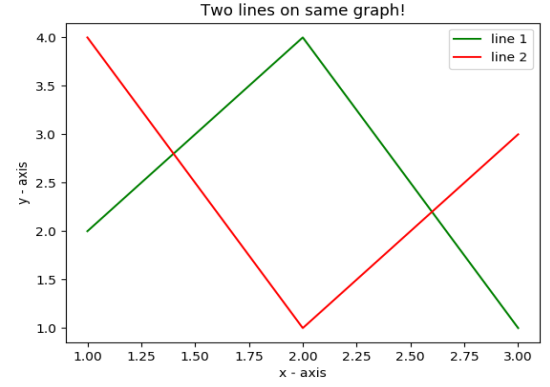


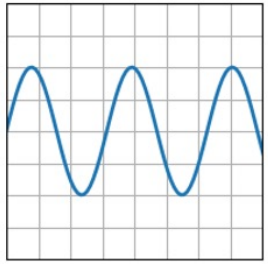




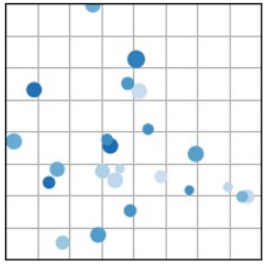




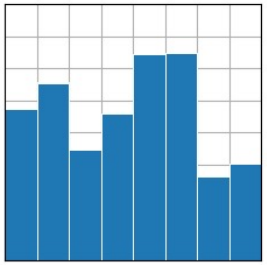




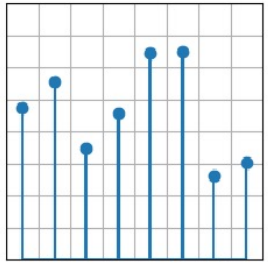
plot(x, y)



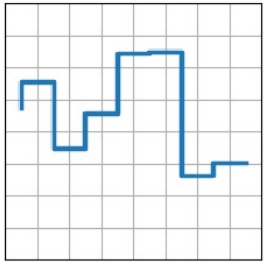
scatter(x, y)



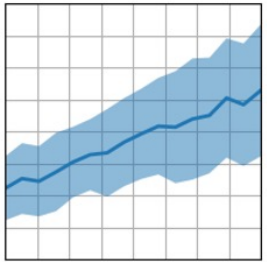
bar(x, height)



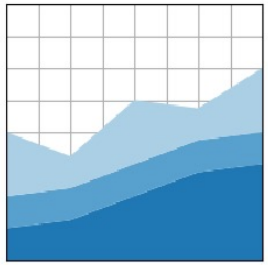
stem(x, y)



step(x, y)

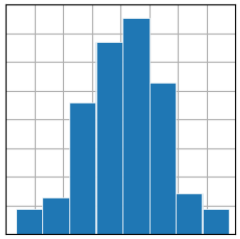


fill_between(x, y1, y2)

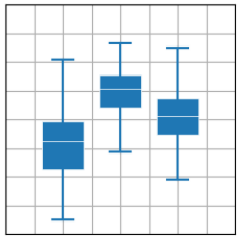


stackplot(x, y)

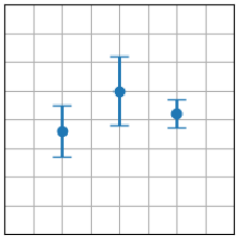
Basic Plots



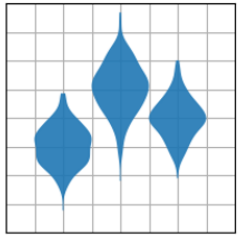
hist(x)



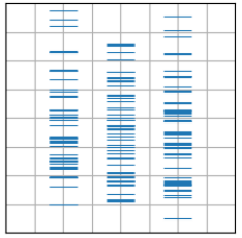
boxplot(X)



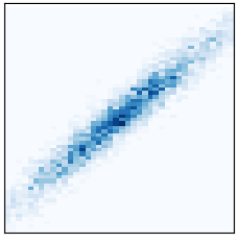
errorbar(x, y, yerr, xerr)



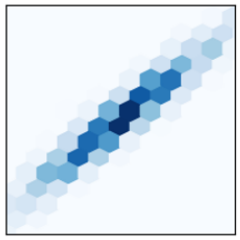
violinplot(D)



eventplot(D)



hist2d(x, y)



hexbin(x, y, C)

pie(x)

Statistics plots

Seaborn is a high-level **data visualization library** built on top of Matplotlib.

It provides **beautiful, modern, and statistical visualizations** with very few lines of code.

Key Features

Automatically applies attractive styles

Built-in statistical plots (histograms, boxplots, heatmaps, etc.)

Works directly with **Pandas DataFrames**

Great for exploring datasets

import seaborn as sns

Comparison of Seaborn and Matplotlib for Data Visualization in Python

Feature / Criteria	Seaborn	Matplotlib
Level of Abstraction	High-level (simpler, fewer lines of code)	Low-level (more control, more coding)
Ease of Use	Very easy, beginner-friendly	Requires more coding and setup
Default Appearance / Style	Modern, attractive by default	Basic and it needs customization
Integration with Pandas	Excellent, it works directly with DataFrames	Good, but manual setup needed
Statistical Plotting	Built-in (heatmaps, boxplots, pairplots)	Limited, it requires manual coding
Customization / Flexibility	Medium	Very high
Performance	Good for medium datasets	Good, it supports large datasets
Complex Layouts / Subplots	Limited	Excellent
Best Use Cases	Data analysis, machine learning, quick plots	Scientific figures, research, full control

Types of Plots

Function	Description
Bar	Make a bar plot.
Barh	Make a horizontal bar plot.
Boxplot	Make a box and whisker plot.
Hist	Plot a histogram.
hist2d	Make a 2D histogram plot.
Pie	Plot a pie chart.
Plot	Plot lines and/or markers to the Axes.
Scatter	Make a scatter plot of x vs y.
Polar	Make a polar plot.
Stackplot	Draws a stacked area plot.
Stem	Create a stem plot
Step	Make a step plot.
Quiver	Plot a 2-D field of arrows.

Axis Functions

Function	Description
Axes	Add axes to the figure.
Text	Add text to the axes.
Title	Set a title of the current axes.
Xlabel	Set the x axis label of the current axis.
Xlim	Get or set the x limits of the current axes.
Xscale	Set the scaling of the x-axis.
Xticks	Get or set the x-limits of the current tick locations and labels.
Ylabel	Set the y axis label of the current axis.
Ylim	Get or set the y-limits of the current axes.
Yscale	Set the scaling of the y-axis.
Yticks	Get or set the y-limits of the current tick locations and labels.
Axes	Add axes to the figure.
Text	Add text to the axes.

Figure Functions

The **figure() function** in pyplot module of matplotlib library is used to create a new figure.

Syntax: *matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=, clear=False, **kwargs)*

Function	Description
Figtext	Add text to figure.
Figure	Creates a new figure.
Show	Display a figure.
Savefig	Save the current figure.
Close	Close a figure window.

Plotting commands: List of some commands which corresponding to `Matplotlib`

`acorr`	Plot the autocorrelation of *x*.
`angle_spectrum`	Plot the angle spectrum.
`annotate`	Annotate the point *xy* with text *text*.
`arrow`	Add an arrow to the Axes.
`autoscale`	Autoscale the axis view to the data (toggle).
`axes`	Add an Axes to the current figure and make it the current Axes.
`axhline`	Add a horizontal line across the Axes.
`axhspan`	Add a horizontal span (rectangle) across the Axes.
`axis`	Convenience method to get or set some axis properties.
`axline`	Add an infinitely long straight line.
`axvline`	Add a vertical line across the Axes.
`axvspan`	Add a vertical span (rectangle) across the Axes.

`bar`	Make a bar plot.
`bar_label`	Label a bar plot.
`barbs`	Plot a 2D field of barbs.
`barh`	Make a horizontal bar plot.
`box`	Turn the axes box on or off on the current axes.
`boxplot`	Draw a box and whisker plot.
`broken_barh`	Plot a horizontal sequence of rectangles.

`cla`	Clear the current axes.
`clabel`	Label a contour plot.
`clf`	Clear the current figure.
`clim`	Set the color limits of the current image.
`close`	Close a figure window.

`xlabel`	Set the label for the x-axis.
`xlim`	Get or set the x limits of the current axes.
`xscale`	Set the xaxis' scale.
`xticks`	Get or set the current tick locations and labels of the x-axis.
`ylabel`	Set the label for the y-axis.
`ylim`	Get or set the y-limits of the current axes.
`yscale`	Set the yaxis' scale.
`yticks`	Get or set the current tick locations and labels of the y-axis.

`grid`	Configure the grid lines.
`hexbin`	Make a 2D hexagonal binning plot of points *x*, *y*.
`hist`	Compute and plot a histogram.
`hist2d`	Make a 2D histogram plot.
`hlines`	Plot horizontal lines at each *y* from *xmin* to *xmax*.
`imread`	Read an image from a file into an array.
`imsave`	Colormap and save an array as an image file.
`imshow`	Display data as an image, i.e., on a 2D regular raster.

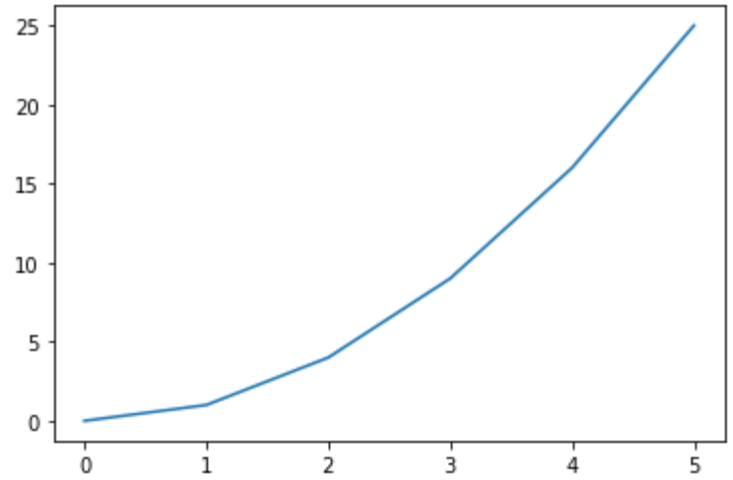
pyplot

- `text()` : adds text in an arbitrary location
- `xlabel()` : adds text to the x-axis
- `ylabel()` : adds text to the y-axis
- `title()` : adds title to the plot
- `clear()` : removes all plots from the axes.
- `savefig()` : saves your figure to a file
- `legend()` : shows a legend on the plot

All methods are available on `pyplot` and on the axes instance generally.

Line Graphs

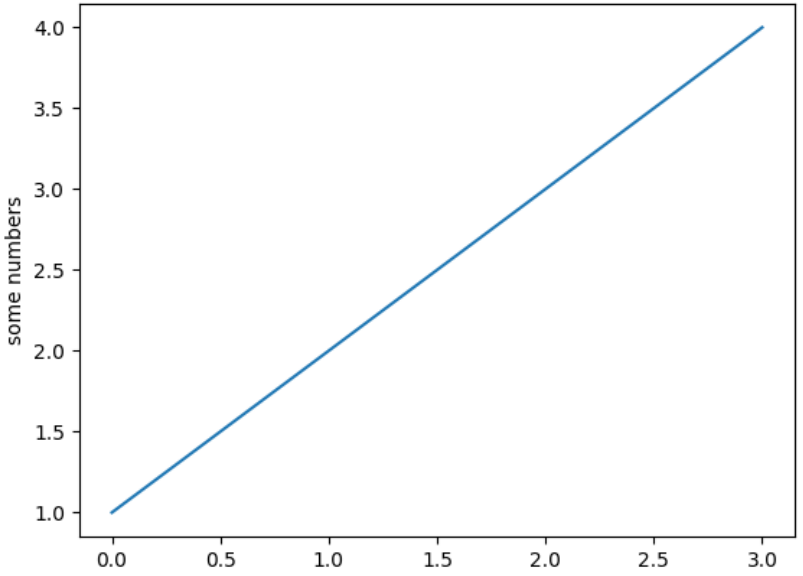
```
import matplotlib.pyplot as plt

#create data for plotting
x_values = [0, 1, 2, 3, 4, 5 ]
y_values = [0, 1, 4, 9, 16,25]

#the default graph style for plot is a line
plt.plot(x_values, y_values)

#display the graph
plt.show()
```



More on Line Graph

- Note: if you provide a single list or array to the `plot()` command,
  - then `matplotlib` assumes it is a sequence of `y values`, and
  - automatically generates the `x values` for you.
- Since python ranges start with `0`, the default `x` vector has the same length as `y` but starts with `0`.
  - Hence the `x` data are `[0, 1, 2, 3]`.

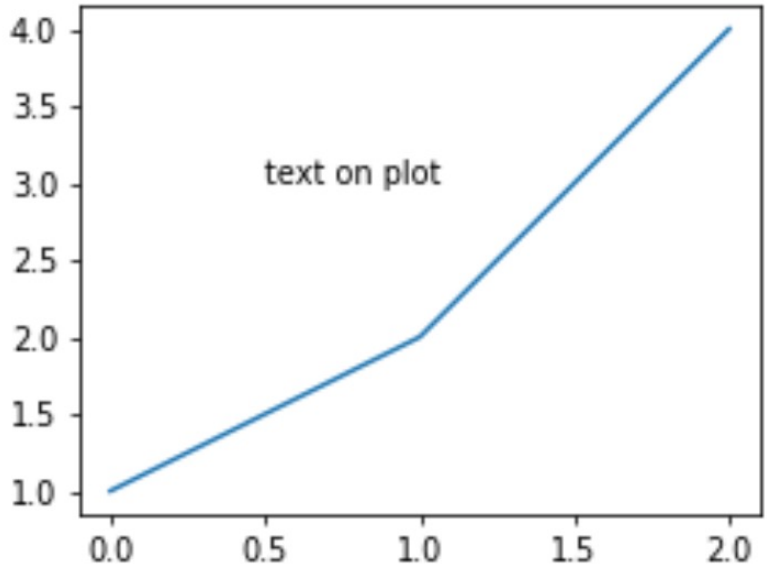
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



matplotlib.pyplot.text

Text() → `text(10, 20, "function", fontsize=12)`

```
import matplotlib.pyplot as plt


w = 4
h = 3
d = 70

plt.figure(figsize=(w, h), dpi=d)

x = [1, 2, 4]
x_pos = 0.5
y_pos = 3

plt.text(x_pos, y_pos, "text on plot")
plt.plot(x)
plt.savefig("out.png")
```

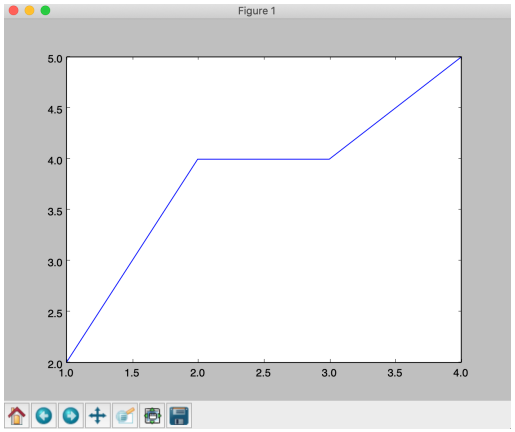


How To Clear A Plot In Python

clf()* | *class: matplotlib.pyplot.clf(). Used to clear the current Figure's state without closing it.

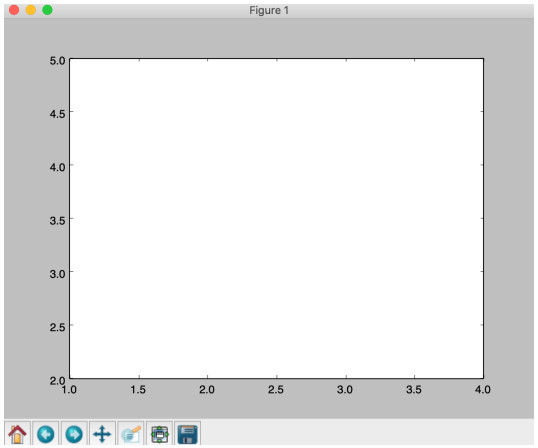
cla()* | *class: matplotlib.pyplot.cla(). Used to clear the current Axes state without closing it.

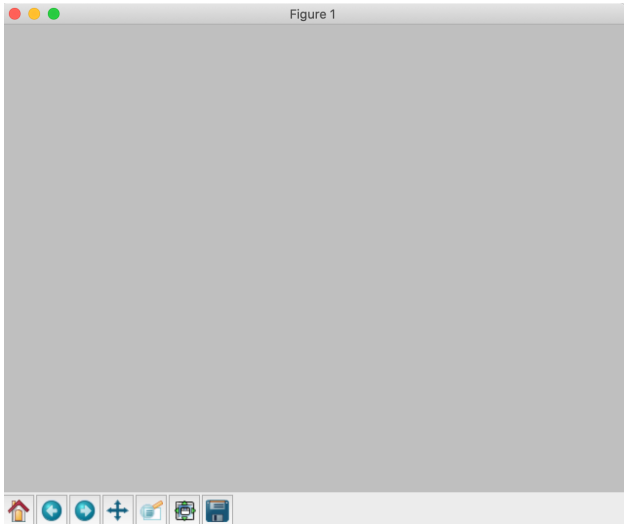
```
import matplotlib.pyplot as plt
f1 = plt.figure()
x = [1,2,3,4]
y = [2,4,4,5]
plt.plot(x,y)
plt.show()
```




```
import matplotlib.pyplot as plt
f1 = plt.figure()
x = [1,2,3,4]
y = [2,4,4,5]
plt.plot(x,y)
plt.clf()
plt.show()
```

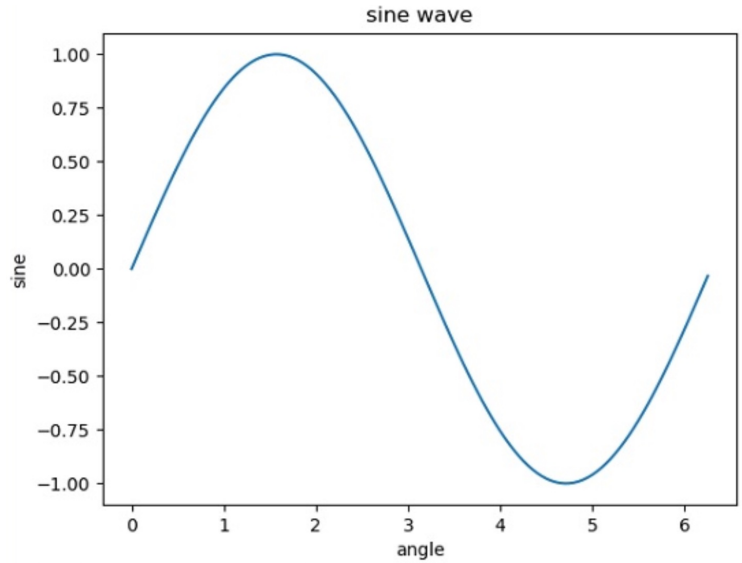
```
import matplotlib.pyplot as plt
f1 = plt.figure()
x = [1,2,3,4]
y = [2,4,4,5]
plt.plot(x,y)
plt.cla()
plt.show()
```



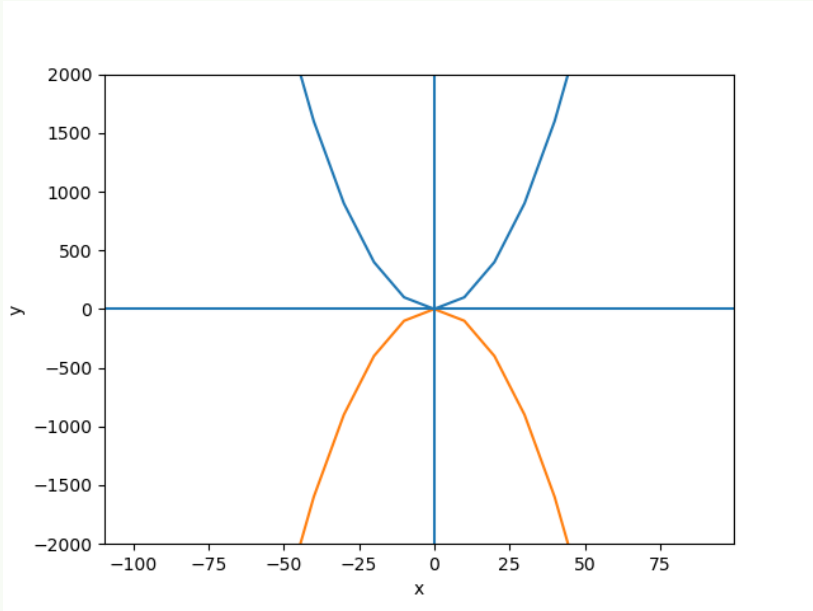


xlabel, ylabel example:

```
from matplotlib import pyplot as plt
import numpy as np
import math #needed for definition of pi
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```



```
import matplotlib.pyplot as plt
y1 =[]
y2 =[]
x = range(-100,100,10)
for i in x: y1.append(i**2)
for i in x: y2.append(-i**2)

plt.plot(x, y1)
plt.plot(x, y2)
plt.xlabel("x")
plt.ylabel("y")
plt.ylim(-2000, 2000)
plt.axhline(0) # horizontal line
plt.axvline(0) # vertical line
plt.savefig("quad.png")
plt.show()
```

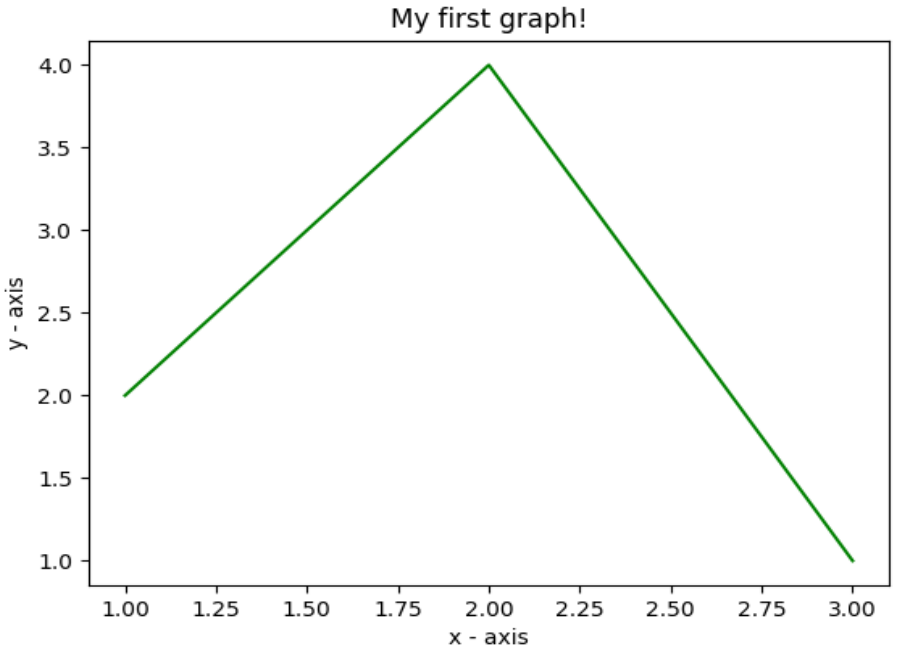


Save your figure to a file

Show it on the screen

Simple line

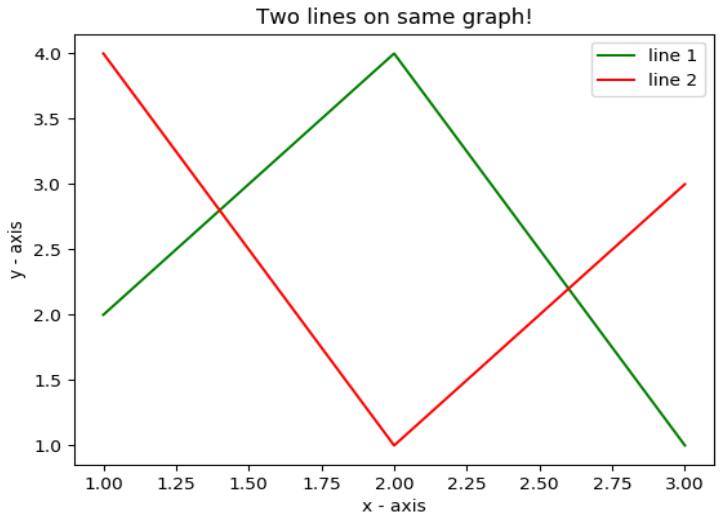
```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```



- Define the **`x-axis`** and corresponding **`y-axis`** values as lists.
- Plot them on canvas using **`.plot()`** function.
- Give a name to x-axis and y-axis using **`.xlabel()`** and **`.ylabel()`** functions.
- Give a title to your plot using **`.title()`** function.
- Finally, to view your plot, we use **`.show()`** function.

Simple 2 lines

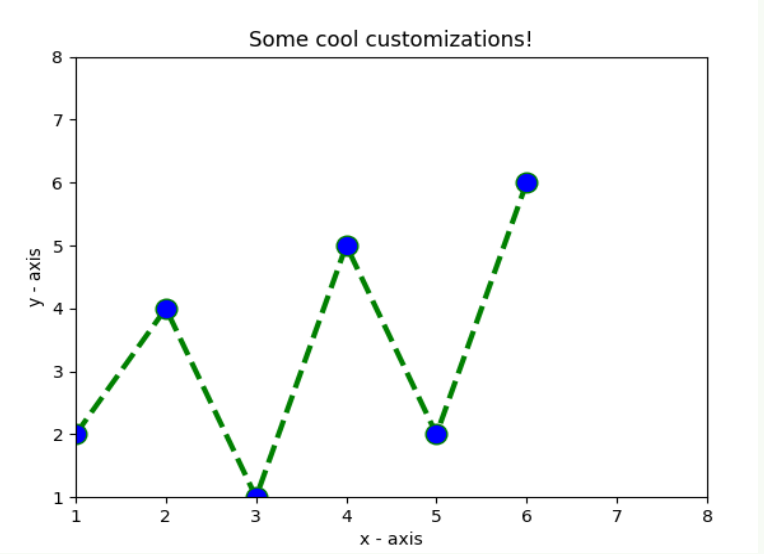
```
import matplotlib.pyplot as plt

# line 1 points
x1 = [1,2,3]
y1 = [2,4,1]
# plotting the line 1 points
plt.plot(x1, y1, label="line 1")

# line 2 points
x2 = [1,2,3]
y2 = [4,1,3]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('Two lines on same graph!')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```



- Here, we plot two lines on same graph. We differentiate between them by giving them a name(label) which is passed as an argument of `.plot()` function.
- The small rectangular box giving information about type of line and its color is called legend. We can add a legend to our plot using `.legend()` function.

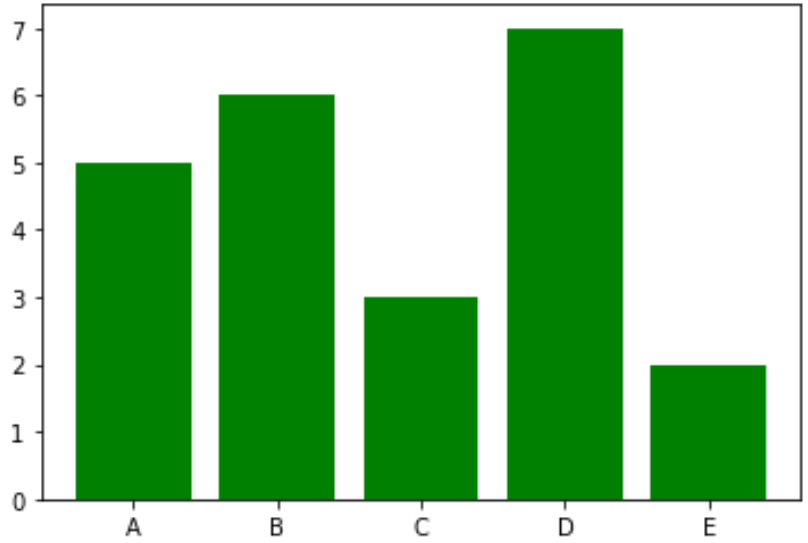
Customization of Plots

```
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3,4,5,6]
# corresponding y axis values
y = [2,4,1,5,2,6]

# plotting the points
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)

# setting x and y axis range
plt.ylim(1,8)
plt.xlim(1,8)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Some cool customizations!')

# function to show the plot
plt.show()
```



Bar graphs

```
import matplotlib.pyplot as plt

#Create data for plotting
values = [5, 6, 3, 7, 2]
names  = ["A", "B", "C", "D", "E"]

plt.bar(names, values, color="green")
plt.show()
```

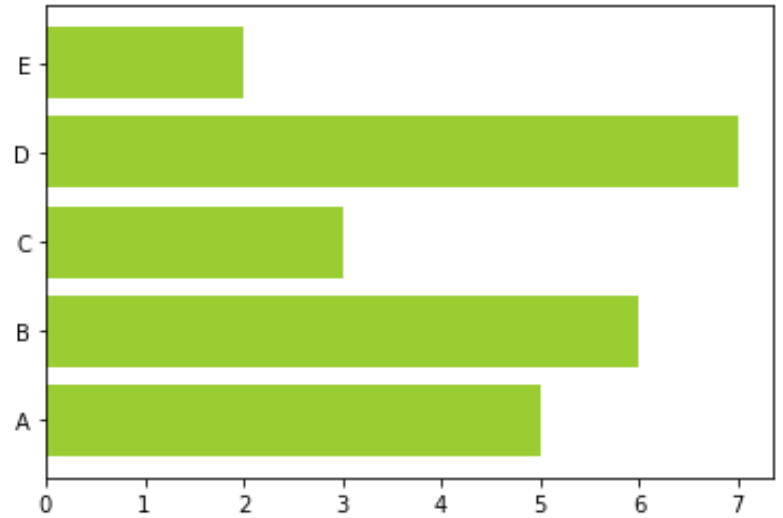


- When using a bar graph, the change in code will be from `plt.plot()` to `plt.bar()` changes it into a bar chart.

Bar graphs

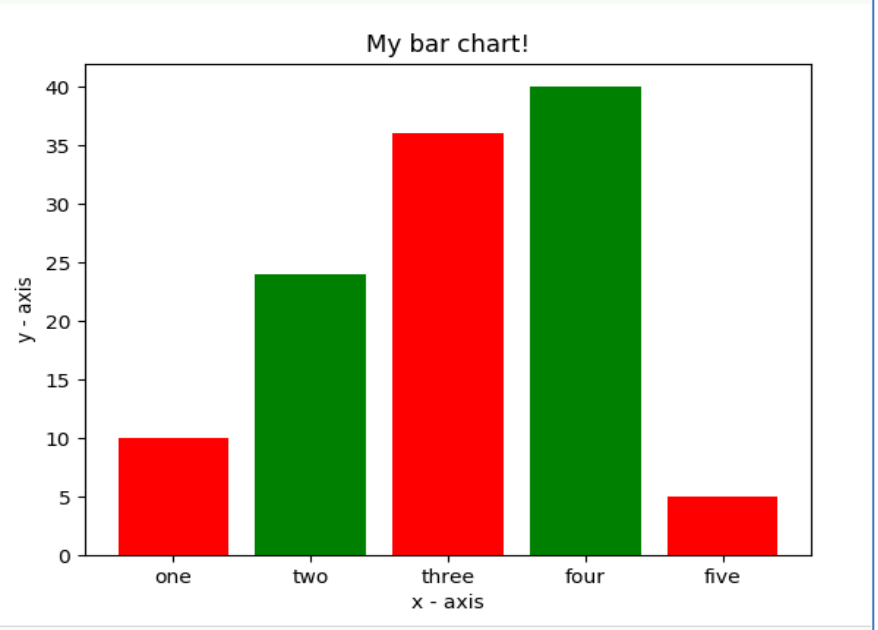
We can also flip the bar graph horizontally with the following

```
import matplotlib.pyplot as plt

#Create data for plotting
values = [5,6,3,7,2]
names  = ["A", "B", "C", "D", "E"]

# Adding an "h" after bar will flip the graph
plt.barh(names, values, color="yellowgreen")
plt.show()
```



Bar Chart

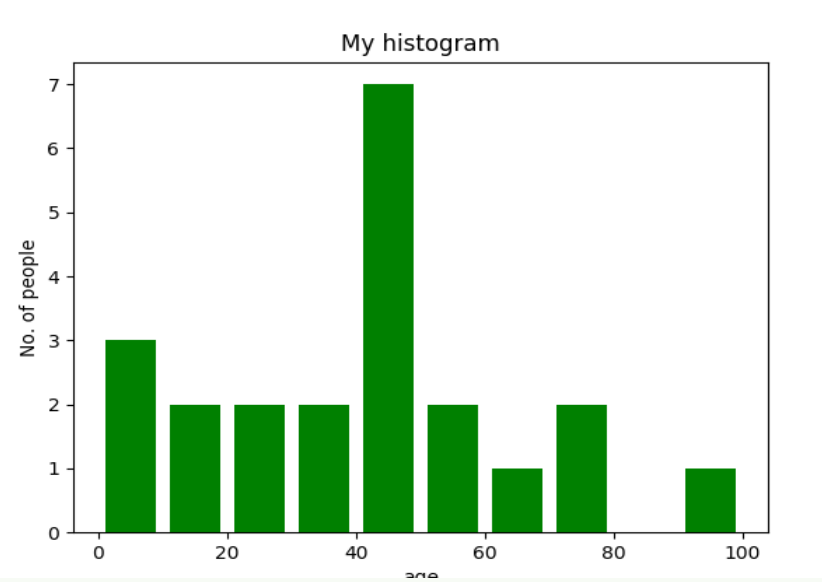
```
import matplotlib.pyplot as plt


# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
names = ['one','two','three','four','five']

# plotting a bar chart
c1 =['red', 'green']
c2 =['b', 'g'] # we can use this for color
plt.bar(left, height, width=0.8, color=c1)

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()
```



- Here, we use `plt.bar()` function to plot a bar chart.
- you can also give some name to x-axis coordinates by defining tick_labels

Histogram

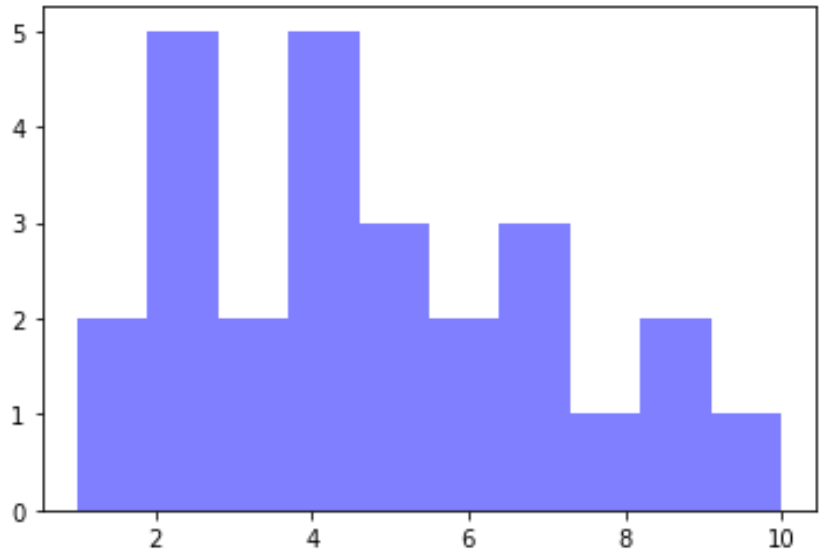
```
import matplotlib.pyplot as plt

# frequencies
ages=[2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]

# setting the ranges and no. of intervals
range = (0, 100)
bins = 10

# plotting a histogram
plt.hist(ages, bins, range, color='green',histtype='bar',rwidth=0.8)

# x-axis label
plt.xlabel('age')
# frequency label
plt.ylabel('No. of people')
# plot title
plt.title('My histogram')

# function to show the plot
plt.show()
```



Histograms

```
import matplotlib.pyplot as plt

#generate fake data
x = [2,1,6,4,2,4,8,9,4,2,4,10,6,4,5,7,7,3,2,7,5,3,5,9,2,1]

#plot for a histogram
plt.hist(x, bins = 10, color='blue', alpha=0.5)
plt.show()
```

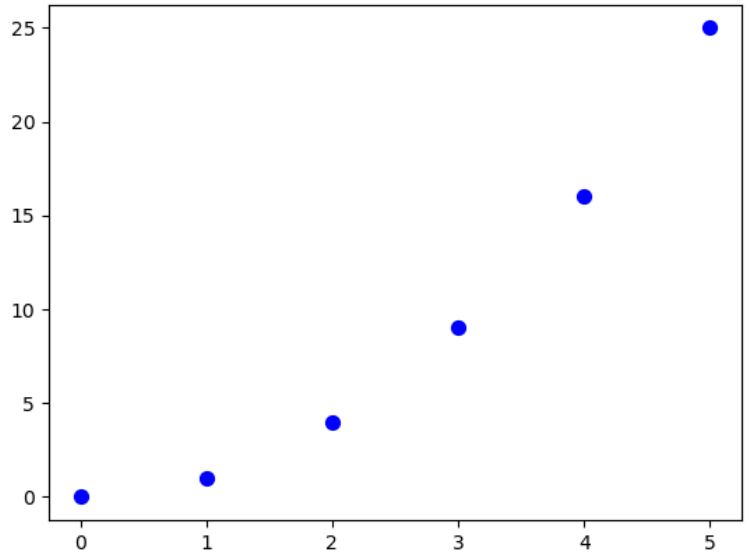
- Looking at the code snippet, I added two new arguments:
  - **Bins** — is an argument specific to a histogram and allows the user to customize how many bins they want.
  - **Alpha** — is an argument that displays the level of transparency of the data points.



Scatter Plots

```
import matplotlib.pyplot as plt

#create data for plotting

x_values = [0,1,2,3,4,5]
y_values = [0,1,4,9,16,25]

plt.scatter(x_values, y_values, s=30, color="blue")
plt.show()
```

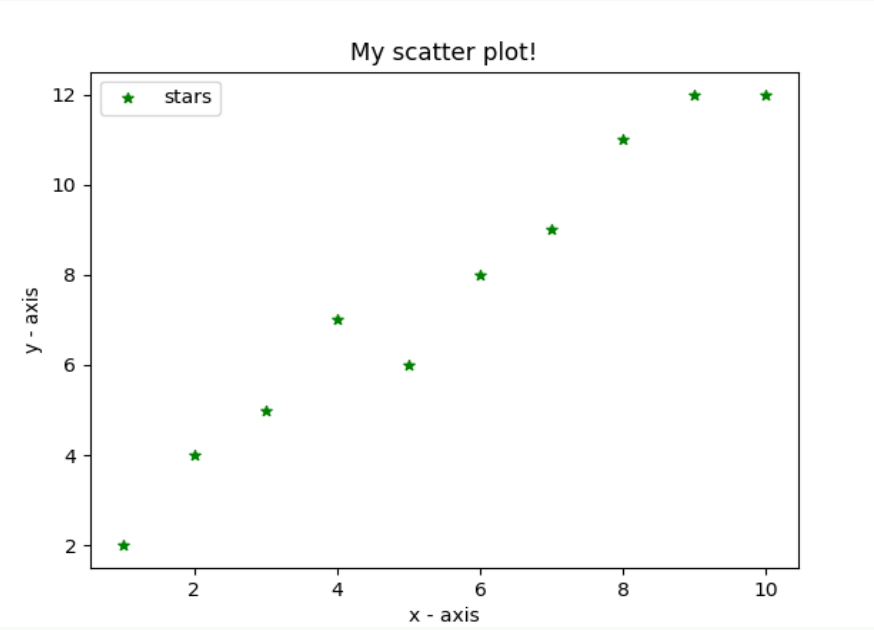
- Can you see the pattern? Now the code changed from `plt.bar()` to `plt.scatter()`.



Assignment # program a real time scatter plot | Dynamic Plotting

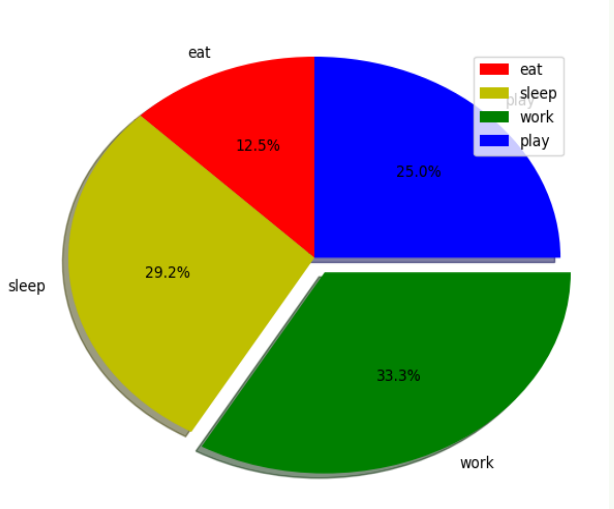
Scatter plot

```
import matplotlib.pyplot as plt

# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color="green", marker="*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```



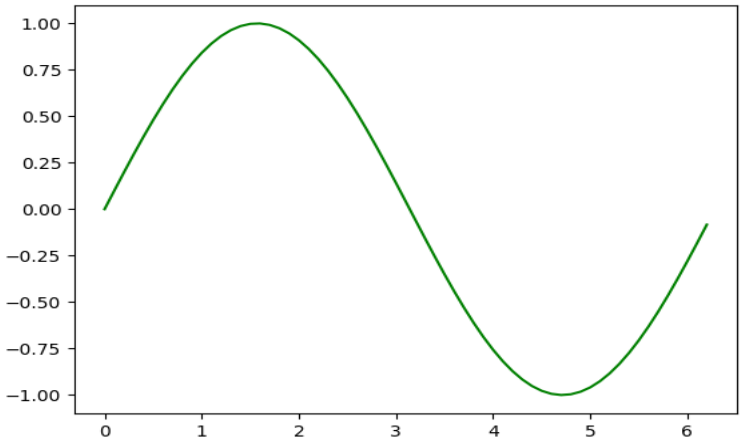
Pie-chart

```
import matplotlib.pyplot as plt

# defining labels
activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label
slices = [3, 7, 8, 6]

# color for each label
colors = ['r', 'y', 'g', 'b']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors,
        startangle=90, shadow = True, explode = (0, 0, 0.1, 0),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()

# showing the plot
plt.show()
```



Plotting curves of given equation

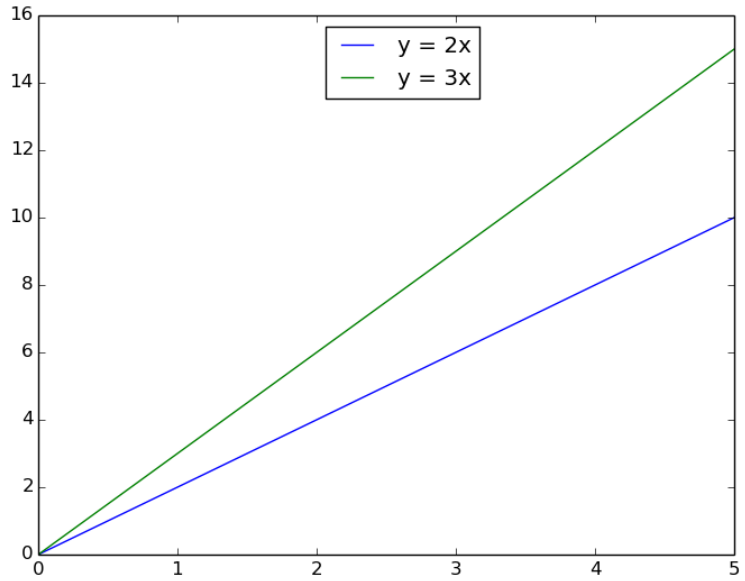
```
# importing the required modules
import matplotlib.pyplot as plt
import numpy as np

# setting the x - coordinates
x = np.arange(0, 2*(np.pi), 0.1)
# setting the corresponding y - coordinates
y = np.sin(x)

# potting the points
plt.plot(x, y)

# function to show the plot
plt.show()
```



Examples taken from:
Graph Plotting in Python | Set 1

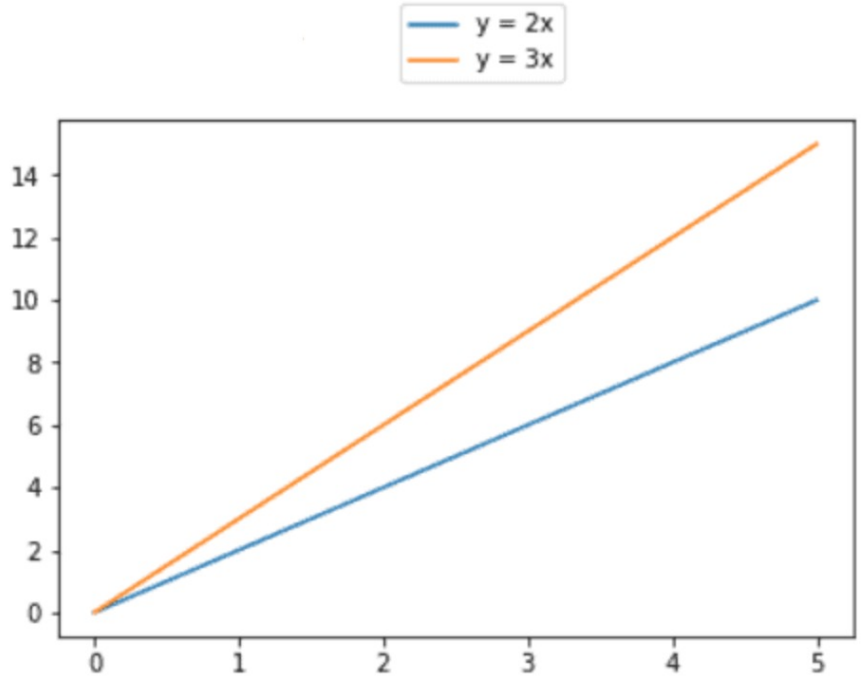
How to add the legend inside the plot

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

# Define Data
x = [0, 1, 2, 3, 4, 5]
y1 = [0, 2, 4, 6, 8, 10]
y2 = [0, 3, 6, 9, 12, 15]

# Plot graph
plt.plot(y1, label ="y = 2x")
plt.plot(y2, label ="y = 3x")

# Add legend
plt.legend(bbox_to_anchor =(0.65, 1))

# Show plot
plt.show()
```



How to add the legend outside the plot (basic method)

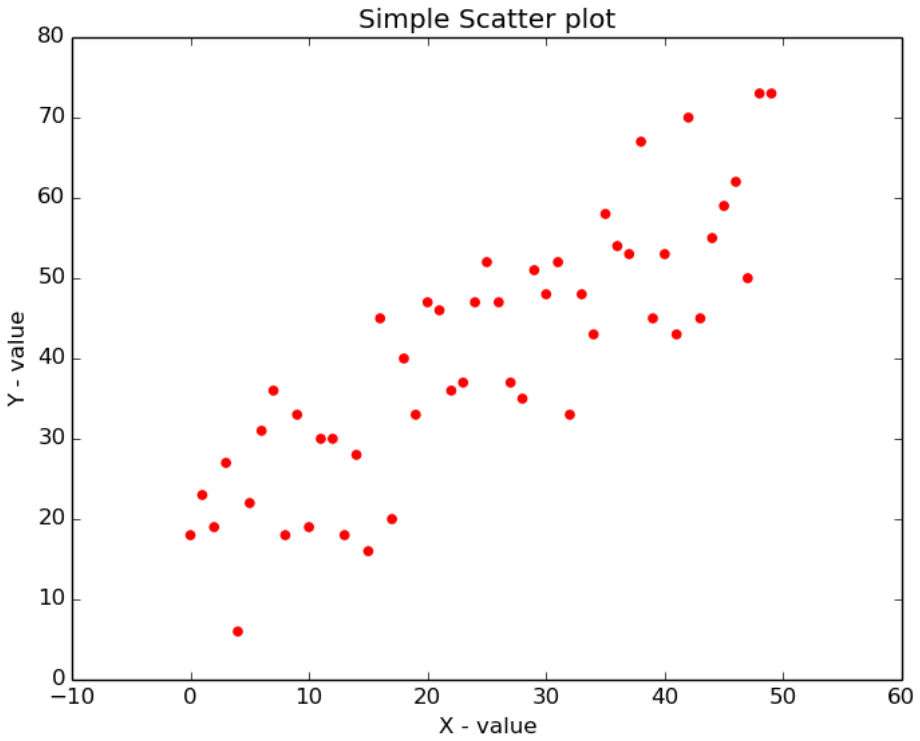
```
import numpy as np
import matplotlib.pyplot as plt

# Define Data
x = [0, 1, 2, 3, 4, 5]
y1 = [0, 2, 4, 6, 8, 10]
y2 = [0, 3, 6, 9, 12, 15]

# Plot graph
plt.plot(y1, label ="y = 2x")
plt.plot(y2, label ="y = 3x")

# Add legend
plt.legend(bbox_to_anchor =(0.65, 1,20))

# Show plot
plt.show()
```



Modify this value

Assignment

```
import matplotlib.pyplot as plt
import numpy as np
x = range(50)
y = range(50) + np.random.randint(0,30,50)
plt.scatter(x, y, color = 'red')
plt.title('Simple Scatter plot')
plt.xlabel('X - value')
plt.ylabel('Y - value')
plt.show()
```



Give green color to the scatter points when (x,y) values are greater than 35

References:

1- https://matplotlib.org/stable/api/pyplot_summary.html

2- https://jakevdp.github.io/PythonDataScienceHandbook/index.html

3- https://www.oreilly.com/library/view/python-data-science/9781491912126/ch04.html

End of Class 6