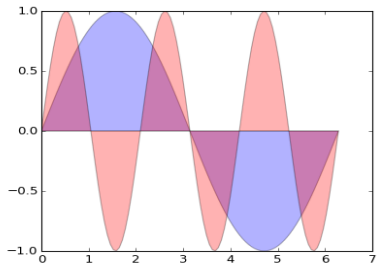
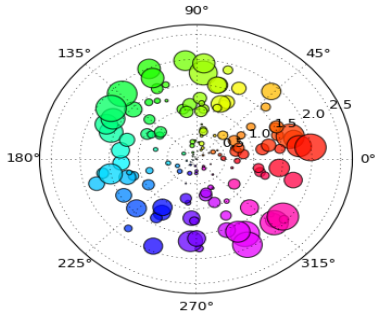
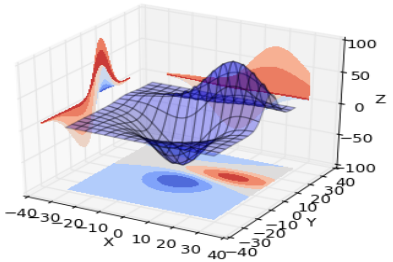
Data Visualization using matplotlib

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!)
  - It is easy to use and is a 2-D plotting library

- In order to get `matplotlib` into your script,
  - first you need to import it, for example:

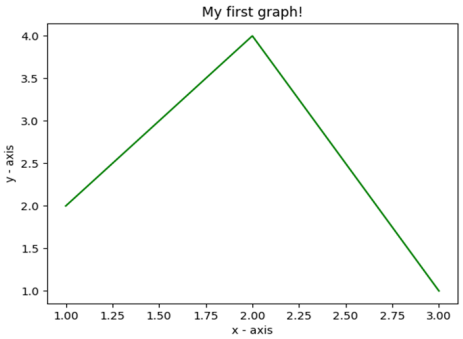
```
import matplotlib.pyplot as plt
```

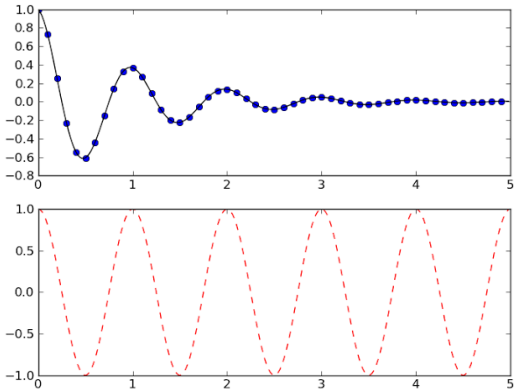
- However, if it is not installed, you may need to install it:
  - Easiest way to install **matplotlib** is using **`pip`**.
  - Type the following command in the command prompt (cmd) or your Linux shell;
    - **`pip install matplotlib`**
    - *Note that you may need to run the above cmd as an administrator*

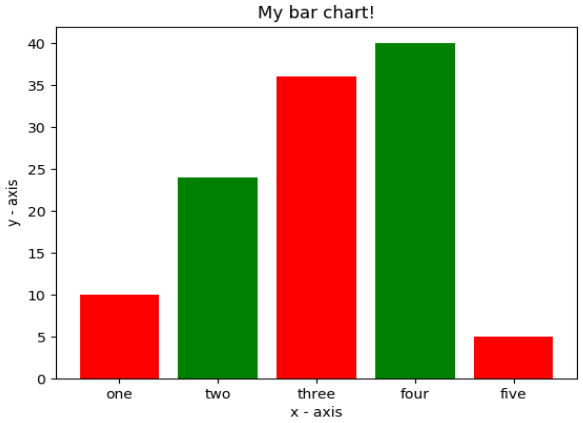
- Each `pyplot` function makes some change to the figure:
  - e.g.,
    - creates a figure,
    - creates a plotting area in the figure,
    - plots some lines in the plotting area,
    - decorates the plot with labels, etc.

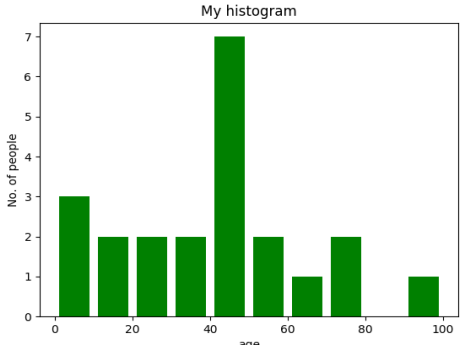
- Whenever we plot with **`matplotlib`**, the two main code lines should be considered:
  - Type of graph
    - this is where you define a bar chart, line chart, etc.
  - Show the graph
    - this is to display the graph

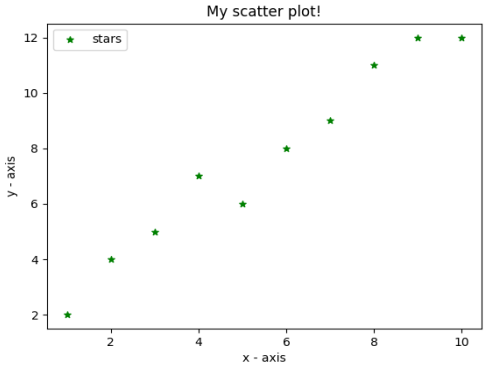
- **`Matplotlib`** allows you to make easy things
- You can generate **plots**, **histograms**, **power spectra**, **bar charts**, **scatterplots**, etc., with just a few lines of code.

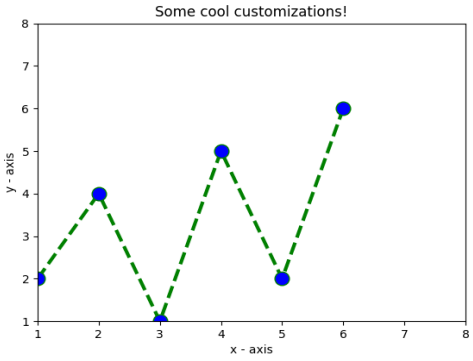


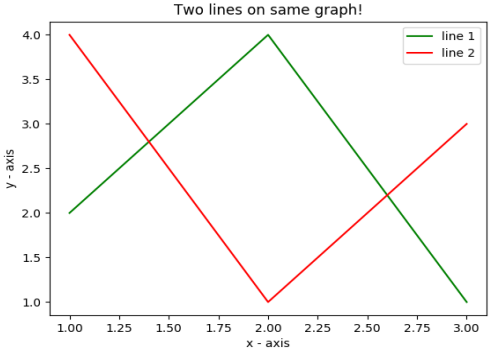












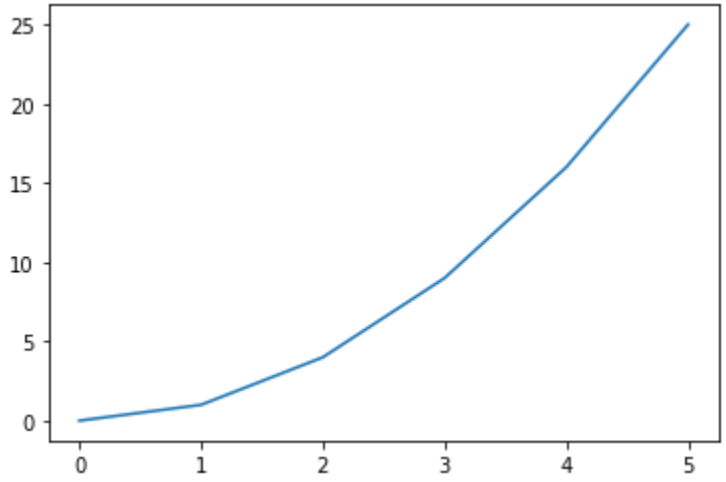
Line Graphs

```
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()
```



The `plot()` function is used to draw points (markers) in a diagram.

By default, the `plot()` function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the **x-axis**.

Parameter 2 is an array containing the points on the **y-axis**.

- Draw a line in a diagram from position (1, 4) to (3, 8) then to (8, 2) and finally to position (9, 10)

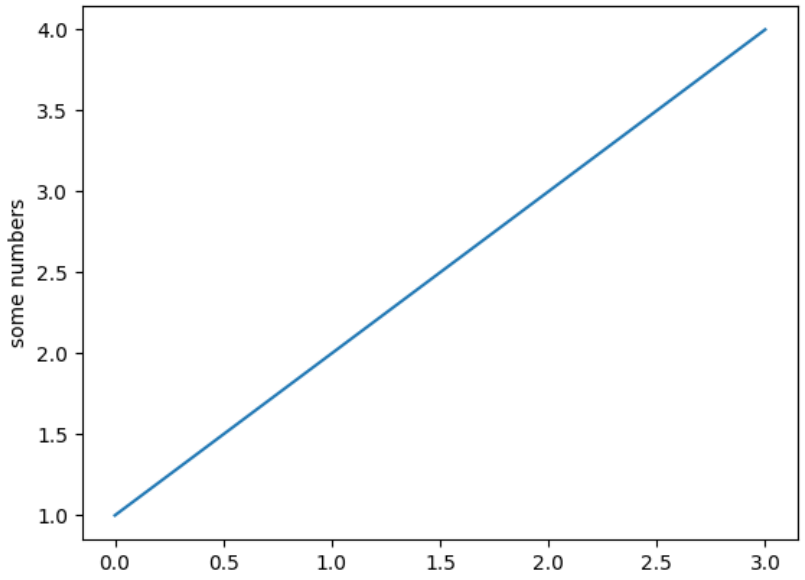
- ```
  import matplotlib.pyplot as plt
  import numpy as np

  xpoints = np.array([1, 3, 8, 9])
  ypoints = np.array([4, 8, 2, 10])

  plt.plot(xpoints, ypoints)
  plt.show()
  ```
  ```

- 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()
```



Default X-Points

- If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 (etc., depending on the length of the y-points.

- So, if we take the same example as above, and leave out the x-points,

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

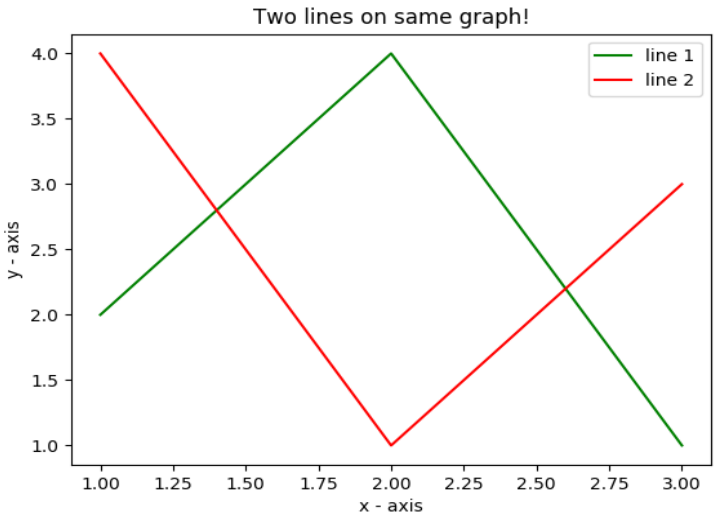
```
import matplotlib.pyplot as plt

# 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!')


# 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.

Marker Reference

Marker	Description	
'o'	Circle	
'*'	Star	
'.'	Point	
','	Pixel	
'x'	X	
'X'	X (filled)	
'+'	Plus	

Color Reference

Color Syntax	Description	
'r'	Red	
'g'	Green	
'b'	Blue	
'c'	Cyan	
'm'	Magenta	
'y'	Yellow	
'k'	Black	
'w'	White	

Line Reference

Line Syntax	Description	
'-'	Solid line	
':'	Dotted line	
'--'	Dashed line	
'-.'	Dashed/dotted line	

Marker Size

- You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms
= 20)
plt.show()
```

```
# 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, marker='o')

# 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()
```

Set Font Properties for Title and Labels

You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```

Add Grid Lines to a Plot

With Pyplot, you can use the `grid()` function to add grid lines to the plot

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

Set Line Properties for the Grid

We can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*)

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```

Creating Scatter Plots

With Pyplot, you can use the `scatter()` function to draw a scatter plot. The `scatter()` function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

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

x = np.array([50,70,80,70,20,170])
y = np.array([90,80,98,88,111,86])

plt.scatter(x, y)
plt.show()
```

Scatter plot

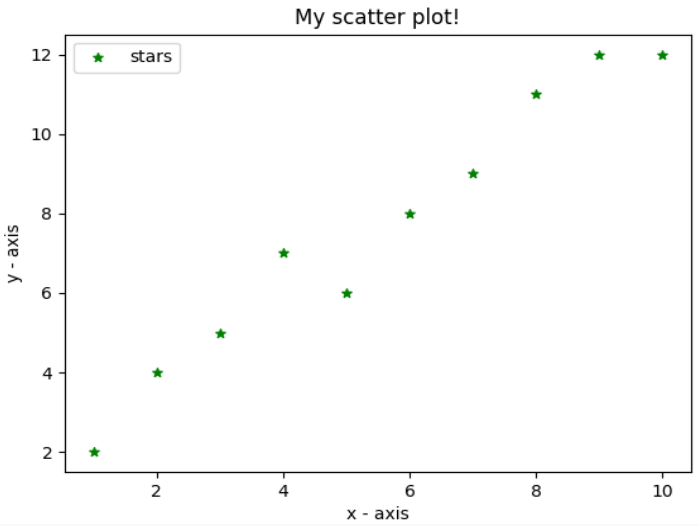
```
import matplotlib.pyplot as plt

# 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()
```



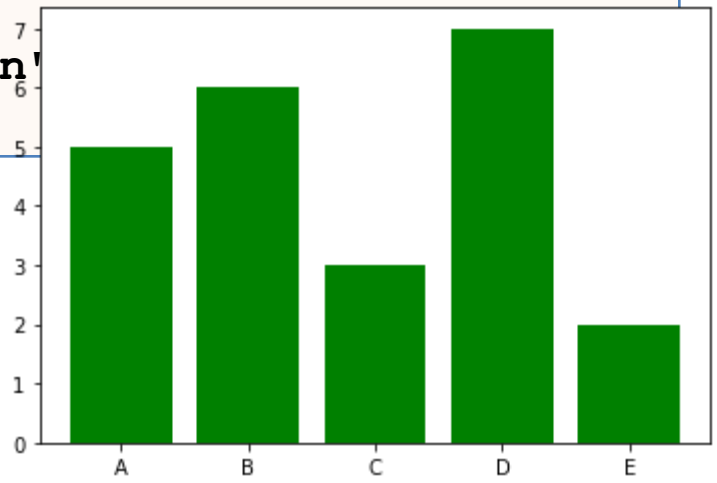
Bar graphs

With Pyplot, you can use the `bar()` function to draw 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.

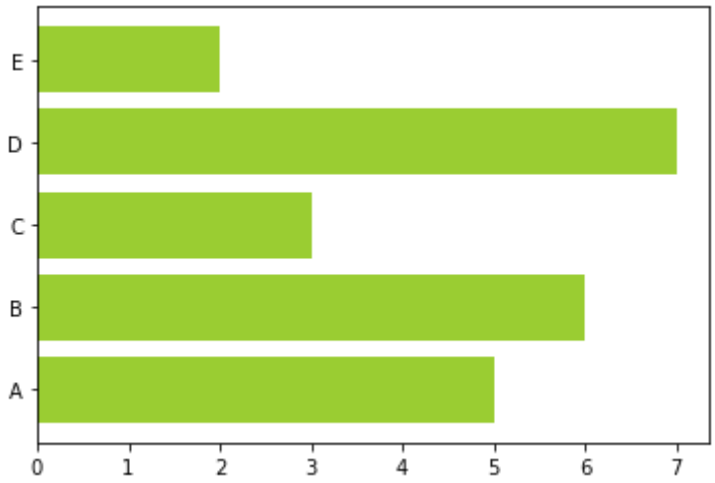
Bar graphs

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

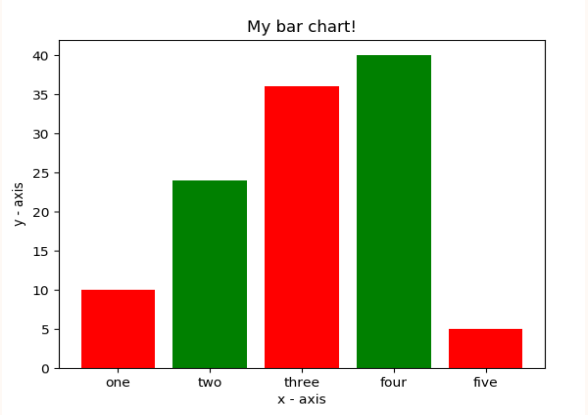
# heights of bars
height = [10, 24, 36, 40, 5]

# 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

To save plot image on harddisk

- plt.savefig('my_plot.png')

Histogram

- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- A histogram is basically used to represent data provided in a form of some groups.
- It is accurate method for the graphical representation of numerical data distributions.
- It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

- To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals.
- Bins are clearly identified as consecutive, non-overlapping intervals of variables.
- Range is the lower and upper range of the bins.

- To construct a histogram, follow these steps −

i. **Bin** the range of values.
ii. Divide the entire range of values into a series of intervals.
iii. Count how many values fall into each interval.

Histogram

```
import matplotlib.pyplot as plt

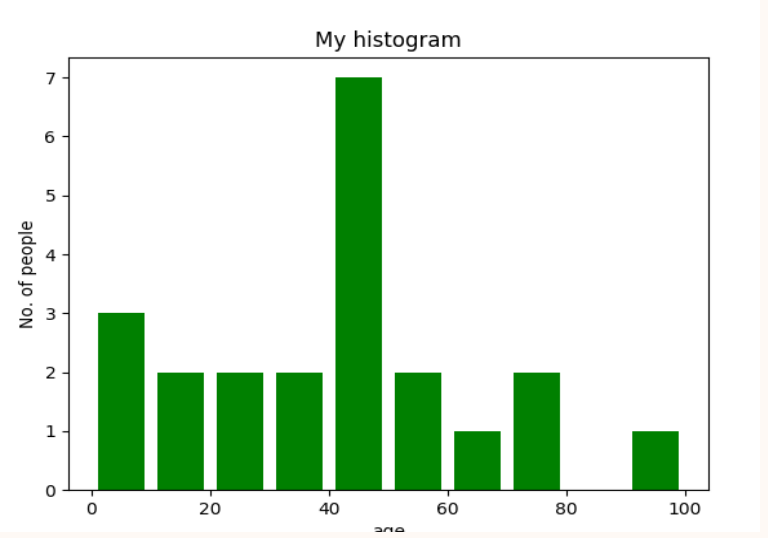
# frequencies
ages=[2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]

# 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()
```



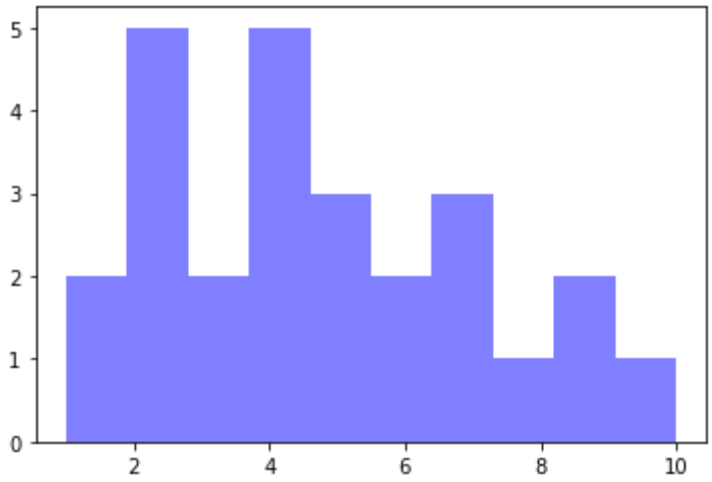
Histograms

```
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')
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.
  - .



Pie-chart

```
import matplotlib.pyplot as plt

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

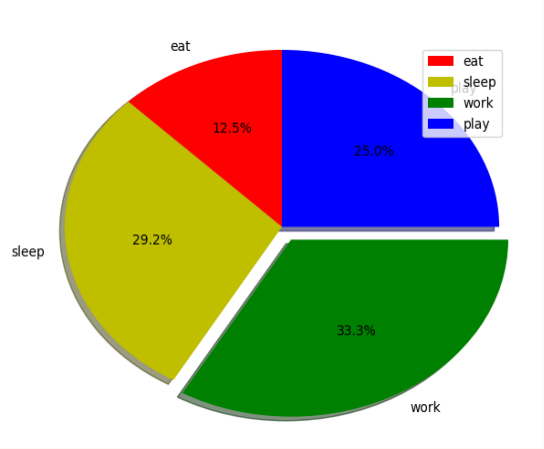
# portion covered by each label
slices = [3, 7, 8, 6]

# 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()
```



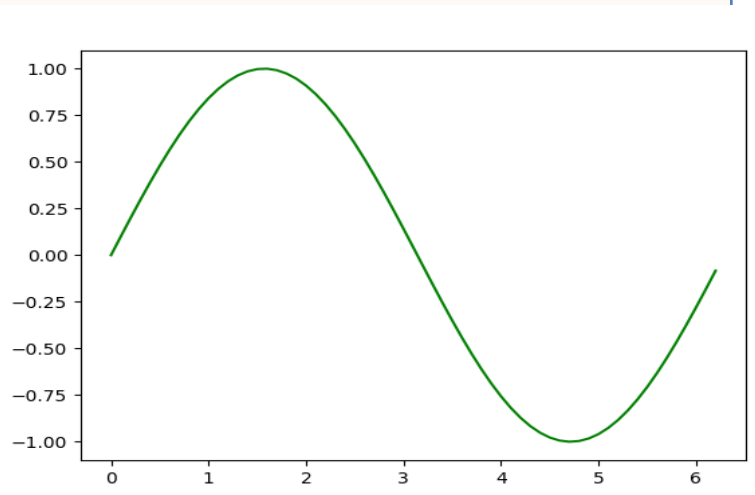
Plotting curves of given equation

```
# 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)
#Returns an array with evenly spaced elements as per the interval, with specified start and stop point
# setting the corresponding y - coordinates
y = np.sin(x)

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

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



Plotting from Pandas DataFrame

- import pandas as pd

```
car_sales = pd.read_csv("C:/Users/hp/Desktop/sanjay/Data_Analysis/car-sales.csv")
car_sales

# Remove price column symbols
car_sales["Price"] = car_sales["Price"].str.replace('[\$\,\.]', '')
car_sales
```

- # Remove last two zeros

```
car_sales["Price"] = car_sales["Price"].str[:-2]
car_sales

car_sales.plot(x="Odometer (KM)", y="Price",
kind="scatter")
```

- #loading heat desease data

heart_disease = pd.read_csv("C:/Users/hp/Desktop/sanjay/Data_Analysis/heart-disease.csv") heart_disease.head()

Perform data analysis on patients over 50

over_50 = heart_disease[heart_disease["age"] > 50]

over_50// This will show the patient details whose age is greater than 50