**1. (a) Installation, Basic Understanding and Working with Jupyter (IPython) Notebook.**
**(b) Installing python data analysis modules or libraries in device using PIP command.**

**Installing the classic Jupyter Notebook interface**

This section includes instructions on how to get started with **Jupyter Notebook**. But there are multiple Jupyter user interfaces one can use, based on their needs. Please check out the list and links below for additional information and instructions about how to get started with each of them.

This information explains how to install the Jupyter Notebook and the IPython kernel.

# Prerequisite: Python

While Jupyter runs code in many programming languages, **Python** is a requirement (Python 3.3 or greater, or Python 2.7) for installing the Jupyter Notebook.

We recommend using the Anaconda distribution to install Python and Jupyter. We'll go through its installation in the next section.

# Installing Jupyter using Anaconda and conda

For new users, we **highly recommend** installing Anaconda. Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

Use the following installation steps:

1. Download Anaconda. We recommend downloading Anaconda's latest Python 3 version (currently Python 3.9).
2. Install the version of Anaconda which you downloaded, following the instructions on the download page.
3. Congratulations, you have installed Jupyter Notebook. To run the notebook:

```
4. jupyter notebook
```

See Running the Notebook for more details.

# *Alternative for experienced Python users:* Installing Jupyter with pip

Jupyter installation requires Python 3.3 or greater, or Python 2.7. IPython 1.x, which included the parts that later became Jupyter, was the last version to support Python 3.2 and2.6.
As an existing Python user, you may wish to install Jupyter using Python's package manager, pip, instead of Anaconda.
First, ensure that you have the latest pip; older versions may have trouble with some dependencies:

```
pip3 install --upgrade pip
```

Then install the Jupyter Notebook using:

```
pip3 install jupyter
```

(Use pip if using legacy Python 2.)

# Basic Steps

1. Start the notebook server from the command line:

```
2. jupyter notebook
```

3. You should see the notebook open in your browser.

# Starting the Notebook Server

After you have installed the Jupyter Notebook on your computer, you are ready to run the notebook server. You can start the notebook server from the command line (using Terminal on Mac/Linux, Command Prompt on Windows) by running:

```
jupyter notebook
```

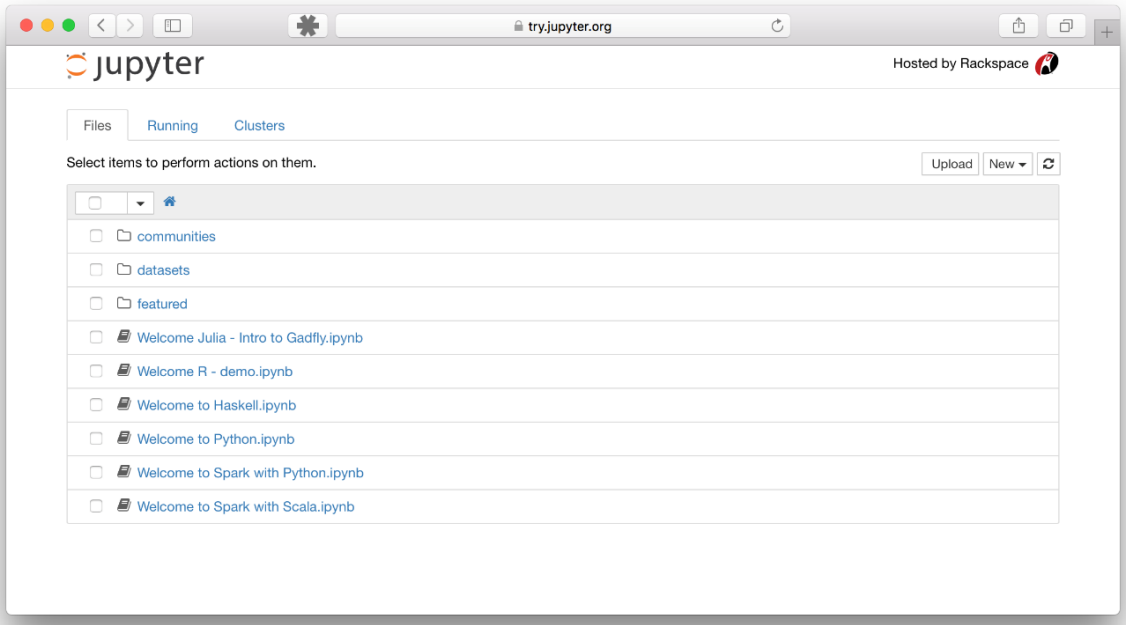
This will print some information about the notebook server in your terminal, including the URL of the web application (by default, `http://localhost:8888`):

```
$ jupyter notebook
[I 08:58:24.417 NotebookApp] Serving notebooks from local directory:
/Users/catherine
[I 08:58:24.417 NotebookApp] 0 active kernels
[I 08:58:24.417 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/
[I 08:58:24.417 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
```

It will then open your default web browser to this URL.

When the notebook opens in your browser, you will see the Notebook Dashboard, which will show a list of the notebooks, files, and subdirectories in the directory where the notebook server was started. Most of the time, you will wish to start a notebook server in the highest level directory containing notebooks. Often this will be your home directory.

**Notebook Dashboard**




# Introducing the Notebook Server's Command Line Options

## How do I open a specific Notebook?

The following code should open the given notebook in the currently running notebook server, starting one if necessary.

```
jupyter notebook notebook.ipynb
```

## How do I start the Notebook using a custom IP or port?

By default, the notebook server starts on port 8888. If port 8888 is unavailable or in use, the notebook server searches the next available port. You may also specify a port manually. In this example, we set the server's port to 9999:

```
jupyter notebook --port 9999
```

## How do I start the Notebook server without opening a browser?

Start notebook server without opening a web browser:

```
jupyter notebook --no-browser
```

## How do I get help about Notebook server options?

The notebook server provides help messages for other command line arguments using the `--help` flag:

```
jupyter notebook --help
```

**See also**

**Jupyter Installation, Configuration, and Usage**
Detailed information about command line arguments, configuration, and usage.

# Using a command-line interface

Notebooks can be executed from your terminal using the `run` subcommand. It expects notebook paths as input arguments and accepts optional flags to modify the default behavior.

Running a notebook is this easy.

```
jupyter run notebook.ipynb
```

You can pass more than one notebook as well.

```
jupyter run notebook.ipynb notebook2.ipynb
```

By default, notebook errors will be raised and printed into the terminal. You can suppress them by passing the `--allow-errors` flag.

```
jupyter run notebook.ipynb --allow-errors
```

## 2. Create a IPython notebook to work with following built-in data structures

(a) List (b) Tuple (c) Set (d) Dictionary

## List

In [1]:
```
numbers = [2, 3, 7, 8, 9]
numbers
```

Out[1]:
```
[2, 3, 7, 8, 9]
```

In [2]:
```
type(numbers)
```

Out[2]:
```
list
```

In [3]:
```
colors = ['red','green','blue']
colors
```

Out[3]:
```
['red', 'green', 'blue']
```

In [4]:
```
#List from range() generator
a_list = list(range(11,18))
a_list
```

Out[4]:
```
[11, 12, 13, 14, 15, 16, 17]
```

In [5]:
```
#Accessing elements
print(numbers[1])
print(numbers[-2])
```

```
3
8
```

In [6]:
```
#slicing
print(numbers[1:4])
print(numbers[:3])
print(numbers[2:])
print(numbers[::2]) # step size 2
print(numbers[::-1]) # reverse
```

```
[3, 7, 8]
[2, 3, 7]
[7, 8, 9]
[2, 7, 9]
[9, 8, 7, 3, 2]
```

### Adding and removing elements in list

In [7]:
```
colors
```

Out[7]:
```
['red', 'green', 'blue']
```

In [8]:
```
colors.append('yellow')
```

In [9]:
```
colors
```

Out[9]:
```
['red', 'green', 'blue', 'yellow']
```

In [10]:
```
colors.insert(2,'black')
```

In [11]:
```
colors
```

Out[11]:
```
['red', 'green', 'black', 'blue', 'yellow']
```

In [12]:
```
colors.remove('black')
```

In [13]:
```
colors
```

Out[13]:
```
['red', 'green', 'blue', 'yellow']
```

In [14]:
```
colors.pop()
```

Out[14]:
```
'yellow'
```

In [15]:
```
colors.pop(2)
```

Out[15]:
```
'blue'
```

In [16]:
```
colors
```

Out[16]:
```
['red', 'green']
```

## Concatenating and combining lists

In [17]:
```
a_list = [2, 4, 6, 8]
b_list = [1, 3, 5, 7]
```

In [18]:
```
# conctenation
a_list + b_list
```

Out[18]:
```
[2, 4, 6, 8, 1, 3, 5, 7]
```

In [19]:
```
a_list.extend(b_list)
```

In [20]:
```
a_list
```

Out[20]:
```
[2, 4, 6, 8, 1, 3, 5, 7]
```

## Sorting a list

In [21]:
```
a = [7, 2, 5, 1, 3]
```

In [22]:
```
a.sort()
```

In [23]:
```
a
```

Out[23]:
```
[1, 2, 3, 5, 7]
```

In [24]:
```
b = ['saw', 'small', 'He', 'foxes', 'six']
```

In [25]:
```
b.sort(key=len)
```

In [26]:
```
b
```

Out[26]:
```
['He', 'saw', 'six', 'small', 'foxes']
```

## Tuple

In [27]:
```
tup1 = (4,5,6, 8,9,6,7,4)
```

In [28]:
```
tup1.count(4)
```

Out[28]:
```
2
```

In [29]:
```
tup1.index(8)
```

Out[29]:
```
3
```

## Set

In [30]:
```
s1 = {3, 6, 8, 2}
s2 = {5, 4, 3, 2, 7}
```

In [31]:
```
s1.union(s2)
```

Out[31]:
```
{2, 3, 4, 5, 6, 7, 8}
```

In [32]:
```
s1.intersection(s2)
```

Out[32]:
```
{2, 3}
```

In [33]:
```
s1.difference(s2)
```

```
{6, 8}
```

Out[33]:

In [34]:
```
s1.symmetric_difference(s2)
```

Out[34]:
```
{4, 5, 6, 7, 8}
```

## Dictionary

In [35]:
```
myd = {'a':97, 'b':98, 'c':99, 'd':100}
```

In [36]:
```
myd.keys()
```

Out[36]:
```
dict_keys(['a', 'b', 'c', 'd'])
```

In [37]:
```
myd.values()
```

Out[37]:
```
dict_values([97, 98, 99, 100])
```

In [38]:
```
myd.items()
```

Out[38]:
```
dict_items([('a', 97), ('b', 98), ('c', 99), ('d', 100)])
```

In [39]:
```
myd['b']
```

Out[39]:
```
98
```

In [40]:
```
myd['e']= 101
```

In [41]:
```
myd
```

Out[41]:
```
{'a': 97, 'b': 98, 'c': 99, 'd': 100, 'e': 101}
```

In [42]:
```
myd.update({'f':102,'a':94})
```

In [43]:
```
myd
```

Out[43]:
```
{'a': 94, 'b': 98, 'c': 99, 'd': 100, 'e': 101, 'f': 102}
```

# 3. Create a IPython notebook to work with numpy module.

## (a) Creating 1d-array and nd-array.

In [1]:
```
import numpy as np
```

In [2]:
```
# creating 1D-Array from List
numbers = [10,20,30,40,50,60,70,80,90]
arr1d = np.array(numbers)
```

In [3]:
```
arr1d
```

Out[3]:
```
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [4]:
```
arr1d.size
```

Out[4]:
```
9
```

In [5]:
```
arr1d.ndim
```

Out[5]:
```
1
```

In [6]:
```
arr1d.shape
```

Out[6]:
```
(9,)
```

In [7]:
```
# creating 1D-Array using arange()
ar1d = np.arange(11,20)
```

In [8]:
```
ar1d
```

Out[8]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [9]:
```
#creationg 1D-array using randint, randn, random and rand methods of numpy random mo
from numpy import random
```

In [10]:
```
a1 = random.randint(20,60, size=6) # 6 random integers from given range
```

In [11]:
```
a1
```

Out[11]:
```
array([38, 51, 52, 52, 54, 30])
```

In [12]:
```
a2 = random.randn(6) # Normal distibution
```

In [13]:
```
a2
```

Out[13]:
```
array([-0.85468327, -0.49336356, -0.57592498, -0.08558379,  0.6429982 ,
        0.05108368])
```

In [14]:
```
a3 = random.random(size=6) # half-open interval [0.0, 1.0)
```

In [15]:
```
a3
```

Out[15]:
```
array([0.25250702, 0.71498189, 0.03565941, 0.22869174, 0.03450219,
       0.63118868])
```

In [16]:
```
a4 = random.rand(6) # uniform distribution
```

In [17]:
```
a4
```

Out[17]:
```
array([0.73377747, 0.05473498, 0.01553481, 0.07525761, 0.80726457,
       0.42393222])
```

In [18]:
```
#creating n-D(2-D) arrays from List
values = [[10,20,30], [40,50,60], [70,80,90]]
arr2d = np.array(values)
```

In [19]:
```
arr2d
```

Out[19]:
```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

In [20]:
```
arr2d.size
```

Out[20]:
```
9
```

In [21]:
```
arr2d.ndim
```

Out[21]:
```
2
```

In [22]:
```
arr2d.shape
```

Out[22]:
```
(3, 3)
```

In [ ]:
```
#creating n-D(2-D) arrays using randint, randn, random and rand methods of numpy ran
```

In [27]:
```
b1 = random.randint(10,99, size=(3,3))
```

In [28]:
```
b1
```

Out[28]:
```
array([[67, 88, 88],
       [20, 63, 20],
       [40, 10, 80]])
```

In [29]:

```
b2 = random.randn(3,3)
```

In [30]:
```
b2
```

Out[30]:
```
array([[-0.29893781, -0.8279161 ,  1.03824269],
       [-0.65656252,  0.12153522, -4.28206731],
       [ 0.45360064,  1.19931908,  0.41203586]])
```

In [31]:
```
b3 = random.random(size=(3,3))
```

In [32]:
```
b3
```

Out[32]:
```
array([[0.08129655, 0.81277675, 0.61978155],
       [0.82826038, 0.40079894, 0.24905223],
       [0.88487556, 0.59234584, 0.37885026]])
```

In [33]:
```
b4 = random.rand(3,3)
```

In [34]:
```
b4
```

Out[34]:
```
array([[0.41217495, 0.94717176, 0.85091539],
       [0.35556207, 0.90501464, 0.02638753],
       [0.407     , 0.21822397, 0.84202797]])
```

In [35]:
```
# creating identity matrix (array)
idarr = np.eye(3)
```

In [36]:
```
idarr
```

Out[36]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [38]:
```
#creating 1-D and 2-D array with all zeros
zero1d = np.zeros(5)
zero2d = np.zeros((4,4))
```

In [39]:
```
zero1d
```

Out[39]:
```
array([0., 0., 0., 0., 0.])
```

In [40]:
```
zero2d
```

Out[40]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [41]:
```
#creating 1-D and 2-D array with all ones
one1d = np.ones(5)
one2d = np.ones((4,4))
```

In [42]:
```
one1d
```

Out[42]:
```
array([1., 1., 1., 1., 1.])
```

In [43]:
```
one2d
```

Out[43]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

## (b) Arithmetic operations with arrays.

In [50]:
```
import numpy as np
from numpy import random
```

In [52]:
```
a1 = random.randint(10,20,size=5)
a2 = random.randint(10,80,size=(3,3))
b1 = np.eye(3)
b2 = random.randint(20,60,size=(3,3))
```

In [53]:
```
a1
```

Out[53]:
```
array([15, 13, 15, 17, 18])
```

In [54]:
```
a2
```

Out[54]:
```
array([[35, 46, 51],
       [37, 75, 30],
       [72, 55, 67]])
```

In [55]:
```
b1
```

Out[55]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [56]:
```
b2
```

Out[56]:
```
array([[35, 26, 51],
       [43, 40, 22],
       [43, 35, 35]])
```

### array and scalar operations

In [57]:
```
a1 + 20
```

Out[57]:
```
array([35, 33, 35, 37, 38])
```

In [58]:
```
a2 + 30
```

Out[58]:
```
array([[ 65,  76,  81],
       [ 67, 105,  60],
       [102,  85,  97]])
```

In [59]:
```
a1 - 10
```

Out[59]:
```
array([5, 3, 5, 7, 8])
```

In [60]:
```
a2 - 10
```

Out[60]:
```
array([[25, 36, 41],
       [27, 65, 20],
       [62, 45, 57]])
```

In [61]:
```
a1 * 10
```

Out[61]:
```
array([150, 130, 150, 170, 180])
```

In [62]:
```
a2 * 10
```

Out[62]:
```
array([[350, 460, 510],
       [370, 750, 300],
       [720, 550, 670]])
```

In [63]:
```
a1 / 2
```

Out[63]:
```
array([7.5, 6.5, 7.5, 8.5, 9. ])
```

In [64]:
```
a2 / 2
```

Out[64]:
```
array([[17.5, 23. , 25.5],
       [18.5, 37.5, 15. ],
       [36. , 27.5, 33.5]])
```

In [65]:
```
a1 // 2
```

Out[65]:
```
array([7, 6, 7, 8, 9], dtype=int32)
```

In [66]:
```
a2 // 2
```

Out[66]:
```
array([[17, 23, 25],
       [18, 37, 15],
       [36, 27, 33]], dtype=int32)
```

In [67]:
```
a1 % 2
```

Out[67]:
```
array([1, 1, 1, 1, 0], dtype=int32)
```

In [68]:
```
a2 % 2
```

Out[68]:
```
array([[1, 0, 1],
       [1, 1, 0],
       [0, 1, 1]], dtype=int32)
```

In [69]:
```
a1 ** 2
```

Out[69]:
```
array([225, 169, 225, 289, 324], dtype=int32)
```

In [70]:
```
a2 ** 2
```

Out[70]:
```
array([[1225, 2116, 2601],
       [1369, 5625,  900],
       [5184, 3025, 4489]], dtype=int32)
```

### array and array operations

In [71]:
```
a2 + b2
```

Out[71]:
```
array([[ 70,  72, 102],
       [ 80, 115,  52],
       [115,  90, 102]])
```

In [72]:
```
a2 - b2
```

Out[72]:
```
array([[ 0, 20,  0],
       [-6, 35,  8],
       [29, 20, 32]])
```

In [73]:
```
a2 * b1
```

Out[73]:
```
array([[35.,  0.,  0.],
       [ 0., 75.,  0.],
       [ 0.,  0., 67.]])
```

## (c) Basic Indexing, Slicing and Boolean Indexing.

In [76]:
```
from numpy import random
c1 = random.randint(10,90,size=9)
c2 = random.randint(10,90,size=(5,5))
```

In [77]:
```
c1
```

Out[77]:
```
array([81, 33, 11, 51, 68, 15, 33, 71, 41])
```

In [78]:
```
c2
```

Out[78]:
```
array([[84, 38, 12, 15, 19],
       [69, 62, 40, 21, 24],
       [61, 88, 65, 30, 17],
       [21, 12, 56, 68, 25],
       [52, 13, 51, 85, 49]])
```

### Basic indexing

In [79]:
```
c1[4]
```

Out[79]:
```
68
```

In [80]:
```
c1[7]
```

Out[80]:
```
71
```

In [81]:
```
c2[2]
```

Out[81]:
```
array([61, 88, 65, 30, 17])
```

In [82]:
```
c2[4]
```

Out[82]:
```
array([52, 13, 51, 85, 49])
```

In [83]:
```
c2[2][3]
```

Out[83]:
```
30
```

In [84]:
```
c2[0,4]
```

Out[84]:
```
19
```

### Slicing

In [85]:
```
c1[1:5]
```

Out[85]:
```
array([33, 11, 51, 68])
```

In [86]:
```
c1[:4]
```

Out[86]:
```
array([81, 33, 11, 51])
```

In [87]:
```
c1[2:]
```

Out[87]:
```
array([11, 51, 68, 15, 33, 71, 41])
```

In [89]:
```
c1[:]
```

Out[89]:
```
array([81, 33, 11, 51, 68, 15, 33, 71, 41])
```

In [91]:
```
c2[1:3,1:4] # row slice and then column slice like [rowslice, colslice]
```

Out[91]:
```
array([[62, 40, 21],
       [88, 65, 30]])
```

In [92]:
```
c2[:4,2:4]
```

Out[92]:
```
array([[12, 15],
       [40, 21],
       [65, 30],
       [56, 68]])
```

In [93]:
```
c2[:,1:3]
```

Out[93]:
```
array([[38, 12],
       [62, 40],
       [88, 65],
```

```
       [12, 56],
       [13, 51]])
```

In [94]:
```
c2[2:4,2:4]
```

Out[94]:
```
array([[65, 30],
       [56, 68]])
```

## Boolean indexing

In [113…
```
c1
```

Out[113…
```
array([81, 33, 11, 51, 68, 15, 33, 71, 41])
```

In [114…
```
c1 > 35
```

Out[114…
```
array([ True, False, False,  True,  True, False, False,  True,  True])
```

In [115…
```
c1[c1 > 35]
```

Out[115…
```
array([81, 51, 68, 71, 41])
```

In [116…
```
c2
```

Out[116…
```
array([[84, 38, 12, 15, 19],
       [69, 62, 40, 21, 24],
       [61, 88, 65, 30, 17],
       [21, 12, 56, 68, 25],
       [52, 13, 51, 85, 49]])
```

In [117…
```
c2 <= 40
```

Out[117…
```
array([[False,  True,  True,  True,  True],
       [False, False,  True,  True,  True],
       [False, False, False,  True,  True],
       [ True,  True, False, False,  True],
       [False,  True, False, False, False]])
```

In [118…
```
c2[c2 <= 40]
```

Out[118…
```
array([38, 12, 15, 19, 40, 21, 24, 30, 17, 21, 12, 25, 13])
```

## (d) Transposing arrays and Swapping Axes.

In [95]:
```
import numpy as np
from numpy import random
```

In [98]:
```
d1 = random.random(size=(4,4))
d2 = random.randint(35, 145, size=(4,4))
```

In [99]:
```
d1
```

```
array([[0.39428302, 0.79844338, 0.69201039, 0.95925536],
```

Out[99]:
```
       [0.09355752, 0.94961324, 0.03552949, 0.73881034],
       [0.93220203, 0.78779133, 0.36744209, 0.13814493],
       [0.56334118, 0.18013536, 0.92937392, 0.9100367 ]])
```

In [100…
```
d2
```

Out[100…
```
array([[119,  90,  97,  58],
       [ 91,  84,  56, 123],
       [ 58,  80, 128, 138],
       [ 92, 109,  65, 134]])
```

## Transpose

In [101…
```
d2.transpose()
```

Out[101…
```
array([[119,  91,  58,  92],
       [ 90,  84,  80, 109],
       [ 97,  56, 128,  65],
       [ 58, 123, 138, 134]])
```

In [102…
```
d2.T
```

Out[102…
```
array([[119,  91,  58,  92],
       [ 90,  84,  80, 109],
       [ 97,  56, 128,  65],
       [ 58, 123, 138, 134]])
```

In [104…
```
np.transpose(d1)
```

Out[104…
```
array([[0.39428302, 0.09355752, 0.93220203, 0.56334118],
       [0.79844338, 0.94961324, 0.78779133, 0.18013536],
       [0.69201039, 0.03552949, 0.36744209, 0.92937392],
       [0.95925536, 0.73881034, 0.13814493, 0.9100367 ]])
```

In [105…
```
np.transpose(d2)
```

Out[105…
```
array([[119,  91,  58,  92],
       [ 90,  84,  80, 109],
       [ 97,  56, 128,  65],
       [ 58, 123, 138, 134]])
```

## Swap axes

In [107…
```
d2
```

Out[107…
```
array([[119,  90,  97,  58],
       [ 91,  84,  56, 123],
       [ 58,  80, 128, 138],
       [ 92, 109,  65, 134]])
```

In [110…
```
d2.swapaxes(0,1) # 0 means row axes and 1 means col axes
```

Out[110…
```
array([[119,  91,  58,  92],
       [ 90,  84,  80, 109],
       [ 97,  56, 128,  65],
       [ 58, 123, 138, 134]])
```

In [111…
```
d1
```

Out[111…
```
array([[0.39428302, 0.79844338, 0.69201039, 0.95925536],
       [0.09355752, 0.94961324, 0.03552949, 0.73881034],
       [0.93220203, 0.78779133, 0.36744209, 0.13814493],
       [0.56334118, 0.18013536, 0.92937392, 0.9100367 ]])
```

In [112…
```
d1.swapaxes(0,1)
```

Out[112…
```
array([[0.39428302, 0.09355752, 0.93220203, 0.56334118],
       [0.79844338, 0.94961324, 0.78779133, 0.18013536],
       [0.69201039, 0.03552949, 0.36744209, 0.92937392],
       [0.95925536, 0.73881034, 0.13814493, 0.9100367 ]])
```

## Result: Numpy Array Creation, Arithmetic Operations, Indexing, Slicing, Boolean indexing, Transpose and Swap axes are done successfully.

In [ ]:

# 4. Create a IPython notebook to work with numpy Mathematical and Statistical Methods.

## Aim: To perform Mathematical and Statistical Methods on numpy arrays

### Importing modules

In [1]:
```
import numpy as np
from numpy import random
```

### Mathematical Methods

In [2]:
```
ar1 = random.randint(low=1,high=20, size=(4,4))
```

In [3]:
```
ar2 = random.randint(low=1, high = 20, size=(4,4))
```

In [4]:
```
ar1
```

Out[4]:
```
array([[17, 11, 14,  3],
       [ 9,  9, 12, 12],
       [10,  3, 13,  7],
       [ 7,  1,  4, 17]])
```

In [5]:
```
ar2
```

Out[5]:
```
array([[18,  3, 14,  5],
       [ 4,  6, 12, 13],
       [11, 18, 18, 12],
       [ 8, 19,  2,  9]])
```

### sum()

In [6]:
```
np.sum(ar1) # sum of all elements in array
```

Out[6]:
```
149
```

In [7]:
```
np.sum(ar2, axis=0) # axis 0 means coloumns(col wise sum)
```

Out[7]:
```
array([41, 46, 46, 39])
```

In [8]:
```
np.sum(ar2, axis=1) # axis 1 means rows (row wise sum)
```

Out[8]:
```
array([40, 35, 59, 38])
```

In [9]:
```
ar1
```

Out[9]:
```
array([[17, 11, 14,  3],
       [ 9,  9, 12, 12],
```

```
       [10,  3, 13,  7],
       [ 7,  1,  4, 17]])
```

In [10]:
```
np.sum(ar1, where = ar1 > 11)
```

Out[10]:
```
85
```

In [11]:
```
np.sum(ar1, axis=0, where= ar1 > 11) # col wise sum of only values greaterthan 11
```

Out[11]:
```
array([17,  0, 39, 29])
```

In [12]:
```
np.sum(ar1, axis=1, where= ar1 < 11) # row wise sum of only values lessthan than 11
```

Out[12]:
```
array([ 3, 18, 20, 12])
```

## min() & max()

In [13]:
```
np.min(ar1) # gives minimum value of array
```

Out[13]:
```
1
```

In [14]:
```
np.max(ar2) # gives maximum value of array
```

Out[14]:
```
19
```

## argmin() & argmax()

In [15]:
```
np.argmin(ar2) # gives index of minimum value of array
```

Out[15]:
```
14
```

In [16]:
```
np.argmax(ar2) # gives index of maximum value of array
```

Out[16]:
```
13
```

## cumsum()

In [17]:
```
ar1
```

Out[17]:
```
array([[17, 11, 14,  3],
       [ 9,  9, 12, 12],
       [10,  3, 13,  7],
       [ 7,  1,  4, 17]])
```

In [18]:
```
np.cumsum(ar1)
```

Out[18]:
```
array([ 17,  28,  42,  45,  54,  63,  75,  87,  97, 100, 113, 120, 127,
       128, 132, 149], dtype=int32)
```

In [19]:
```
np.cumsum(ar1, axis=0)
```

```
array([[17, 11, 14,  3],
```

Out[19]:
```
       [26, 20, 26, 15],
       [36, 23, 39, 22],
       [43, 24, 43, 39]], dtype=int32)
```

In [20]:
```
np.cumsum(ar1, axis=1)
```

Out[20]:
```
array([[17, 28, 42, 45],
       [ 9, 18, 30, 42],
       [10, 13, 26, 33],
       [ 7,  8, 12, 29]], dtype=int32)
```

## cumprod()

In [21]:
```
ar2
```

Out[21]:
```
array([[18,  3, 14,  5],
       [ 4,  6, 12, 13],
       [11, 18, 18, 12],
       [ 8, 19,  2,  9]])
```

In [22]:
```
np.cumprod(ar2)
```

Out[22]:
```
array([         18,          54,         756,        3780,       15120,
            90720,     1088640,    14152320,   155675520, -1492807936,
       -1100739072,  -323966976,  1703231488, -1998340096,   298287104,
       -1610383360], dtype=int32)
```

In [23]:
```
np.cumprod(ar2, axis=0)
```

Out[23]:
```
array([[  18,    3,   14,    5],
       [  72,   18,  168,   65],
       [ 792,  324, 3024,  780],
       [6336, 6156, 6048, 7020]], dtype=int32)
```

In [24]:
```
np.cumprod(ar2, axis=1)
```

Out[24]:
```
array([[   18,    54,   756,  3780],
       [    4,    24,   288,  3744],
       [   11,   198,  3564, 42768],
       [    8,   152,   304,  2736]], dtype=int32)
```

## mean()

In [25]:
```
normd = random.randn(3,3)
```

In [26]:
```
normd
```

Out[26]:
```
array([[ 0.45352732,  0.89511391,  1.71788602],
       [-0.27050059,  0.84132046,  0.93694378],
       [-1.3216317 , -0.38423519,  0.68706941]])
```

In [27]:
```
np.mean(normd)
```

Out[27]:
```
0.39505482412369297
```

In [28]:
```
np.mean(normd, axis=0)
```

Out[28]:
```
array([-0.37953499,  0.45073306,  1.1139664 ])
```

In [29]:
```
np.mean(normd, axis=1)
```

Out[29]:
```
array([ 1.02217575,  0.50258788, -0.33959916])
```

### std()

In [30]:
```
np.std(normd)
```

Out[30]:
```
0.8549658358249043
```

In [31]:
```
np.std(normd, axis=0)
```

Out[31]:
```
array([0.72879524, 0.59082001, 0.43905085])
```

In [32]:
```
np.std(normd, axis=1)
```

Out[32]:
```
array([0.52393337, 0.54804823, 0.82065596])
```

### var()

In [33]:
```
np.var(normd)
```

Out[33]:
```
0.7309665804277771
```

In [34]:
```
np.var(normd, axis=0)
```

Out[34]:
```
array([0.5311425 , 0.34906828, 0.19276565])
```

In [35]:
```
np.var(normd, axis=1)
```

Out[35]:
```
array([0.27450618, 0.30035687, 0.67347621])
```

### Result: Mathematical and Statistical methods on numpy arrays performed and executed successfully.

# 5. Create a IPython notebook to work with pandas: Series and DataFrame.

## (a) Series

### Importing libraries

In [1]:
```
import pandas as pd
import numpy as np
```

### Creating Series

In [2]:
```
ser1 = pd.Series(data = [20, -15, 34, -42, 67])
```

In [3]:
```
ser1
```

Out[3]:
```
0    20
1   -15
2    34
3   -42
4    67
dtype: int64
```

In [4]:
```
ser1.dtype
```

Out[4]:
```
dtype('int64')
```

In [5]:
```
ser1.values
```

Out[5]:
```
array([ 20, -15,  34, -42,  67], dtype=int64)
```

In [6]:
```
ser1.index
```

Out[6]:
```
RangeIndex(start=0, stop=5, step=1)
```

In [7]:
```
# Series with customized index
ser2 = pd.Series(data = [30, -25, 44, -52, 63],
                 index = ['A', 'B', 'C', 'D', 'E'])
```

In [8]:
```
ser2
```

Out[8]:
```
A    30
B   -25
C    44
D   -52
E    63
dtype: int64
```

In [9]:
```
ser2.values
```

Out[9]:
```
array([ 30, -25,  44, -52,  63], dtype=int64)
```

In [10]:
```
ser2.index
```

Out[10]:
```
Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

In [11]:
```
# Creating series without keyword arguments
ser3 = pd.Series([87, 65, -66, -78, 55],
                 ['a', 'b', 'c', 'd','e'])
```

In [12]:
```
ser3
```

Out[12]:
```
a    87
b    65
c   -66
d   -78
e    55
dtype: int64
```

In [13]:
```
#Creating Series with dictionary
ser4 = pd.Series({'A':65, 'B':66, 'C':67, 'D':68})
```

In [14]:
```
ser4
```

Out[14]:
```
A    65
B    66
C    67
D    68
dtype: int64
```

## Indexing and Slicing

In [15]:
```
ser1[2]
```

Out[15]:
```
34
```

In [16]:
```
ser2['B']
```

Out[16]:
```
-25
```

In [17]:
```
ser1[1:4]
```

Out[17]:
```
1   -15
2    34
3   -42
dtype: int64
```

In [18]:
```
ser2['A':'D']
```

Out[18]:
```
A    30
B   -25
C    44
D   -52
dtype: int64
```

In [19]:

```
# fancy indexing
ser1[[2,3,0]]
```

Out[19]:
```
2    34
3   -42
0    20
dtype: int64
```

In [20]:
```
# boolean Series
ser2 > 0
```

Out[20]:
```
A     True
B    False
C     True
D    False
E     True
dtype: bool
```

In [21]:
```
# boolean indexing
ser2[ser2 > 0]
```

Out[21]:
```
A    30
C    44
E    63
dtype: int64
```

In [22]:
```
ser2[['A','D','C']]
```

Out[22]:
```
A    30
D   -52
C    44
dtype: int64
```

## DataFrame

In [23]:
```
df1 = pd.DataFrame(data=np.random.randint(-20, 20, size=(4,4)),
                   columns=['W','X','Y','Z'])
```

In [24]:
```
df1
```

Out[24]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **0** | -10 | -20 | -6 | 11 |
| **1** | 1 | -16 | -9 | 17 |
| **2** | 4 | -11 | -1 | 9 |
| **3** | -2 | -8 | -2 | -10 |

In [25]:
```
df1.values
```

Out[25]:
```
array([[-10, -20,  -6,  11],
       [  1, -16,  -9,  17],
       [  4, -11,  -1,   9],
       [ -2,  -8,  -2, -10]])
```

In [26]:
```
df1.columns
```

Out[26]:
```
Index(['W', 'X', 'Y', 'Z'], dtype='object')
```

In [27]:
```
df1.index
```

Out[27]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [28]:
```
df2 = pd.DataFrame(data=np.random.randint(-20, 20, size=(4,4)),
                   columns=['W','X','Y','Z'],
                   index=['A','B','C','D'])
```

In [29]:
```
df2
```

Out[29]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |
| **C** | -11 | -8 | 14 | 19 |
| **D** | 6 | 19 | 14 | -19 |

In [30]:
```
df2.values
```

Out[30]:
```
array([[ -4,  14,  13, -16],
       [ 11,   2,   2, -10],
       [-11,  -8,  14,  19],
       [  6,  19,  14, -19]])
```

In [31]:
```
df2.index
```

Out[31]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [32]:
```
df2.columns
```

Out[32]:
```
Index(['W', 'X', 'Y', 'Z'], dtype='object')
```

## Indexing and Slicing

### column wise

In [33]:
```
df1['X']
```

Out[33]:
```
0   -20
1   -16
2   -11
3    -8
Name: X, dtype: int32
```

In [34]:
```
df1['X'][1]
```

Out[34]:
```
-16
```

In [35]:
```
df1['Z'][3]
```

Out[35]:
```
-10
```

In [36]:
```
df1[['W','Z']]
```

Out[36]:

| | W | Z |
|---|---|---|
| **0** | -10 | 11 |
| **1** | 1 | 17 |
| **2** | 4 | 9 |
| **3** | -2 | -10 |

In [37]:
```
df2['W']
```

Out[37]:
```
A    -4
B    11
C   -11
D     6
Name: W, dtype: int32
```

In [38]:
```
df2['W']['A']
```

Out[38]:
```
-4
```

In [39]:
```
df2[['W','Y','X']]
```

Out[39]:

| | W | Y | X |
|---|---|---|---|
| **A** | -4 | 13 | 14 |
| **B** | 11 | 2 | 2 |
| **C** | -11 | 14 | -8 |
| **D** | 6 | 14 | 19 |

## Row wise

In [40]:
```
df2
```

Out[40]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |
| **C** | -11 | -8 | 14 | 19 |
| **D** | 6 | 19 | 14 | -19 |

In [41]:
```
df2.loc['B']
```

Out[41]:
```
W    11
```

```
X     2
Y     2
Z   -10
Name: B, dtype: int32
```

In [42]:
```
df2.loc['C']
```

Out[42]:
```
W   -11
X    -8
Y    14
Z    19
Name: C, dtype: int32
```

In [43]:
```
df2.loc['C']['Y']
```

Out[43]:
```
14
```

In [44]:
```
df1.loc[1]['W']
```

Out[44]:
```
1
```

In [45]:
```
df2.loc['A':'C']
```

Out[45]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |
| **C** | -11 | -8 | 14 | 19 |

In [46]:
```
df2.loc[['A','C']]
```

Out[46]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **C** | -11 | -8 | 14 | 19 |

In [47]:
```
df2.loc['A':'C']
```

Out[47]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |
| **C** | -11 | -8 | 14 | 19 |

In [48]:
```
df2.iloc[0]
```

Out[48]:
```
W    -4
X    14
Y    13
Z   -16
Name: A, dtype: int32
```

In [49]:
```
df2.iloc[2]
```

Out[49]:
```
W   -11
X    -8
Y    14
Z    19
Name: C, dtype: int32
```

In [50]:
```
df2.iloc[2]['X']
```

Out[50]:
```
-8
```

In [51]:
```
df2.iloc[0]['Y']
```

Out[51]:
```
13
```

## Rows selection on top of Columns

In [52]:
```
df2
```

Out[52]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |
| **C** | -11 | -8 | 14 | 19 |
| **D** | 6 | 19 | 14 | -19 |

In [53]:
```
df2[['W','Y','Z']]['A':'B']
```

Out[53]:

| | W | Y | Z |
|---|---|---|---|
| **A** | -4 | 13 | -16 |
| **B** | 11 | 2 | -10 |

In [54]:
```
df2[['W','Y','Z']].loc[['A','B']]
```

Out[54]:

| | W | Y | Z |
|---|---|---|---|
| **A** | -4 | 13 | -16 |
| **B** | 11 | 2 | -10 |

## Columns selection on top of Rows

In [55]:
```
df2
```

Out[55]:

| | W | X | Y | Z |
|---|---|---|---|---|
| **A** | -4 | 14 | 13 | -16 |
| **B** | 11 | 2 | 2 | -10 |

| | W | X | Y | Z |
|---|---|---|---|---|
| **C** | -11 | -8 | 14 | 19 |
| **D** | 6 | 19 | 14 | -19 |

In [56]:
```
df2.loc[['A','C']][['W','Y']]
```

Out[56]:

| | W | Y |
|---|---|---|
| **A** | -4 | 13 |
| **C** | -11 | 14 |

In [57]:
```
df2.loc['A':'C'][['W','Y']]
```

Out[57]:

| | W | Y |
|---|---|---|
| **A** | -4 | 13 |
| **B** | 11 | 2 |
| **C** | -11 | 14 |

In [58]:
```
df2.iloc[[0,2]][['W','Y']]
```

Out[58]:

| | W | Y |
|---|---|---|
| **A** | -4 | 13 |
| **C** | -11 | 14 |

In [59]:
```
df2.iloc[0:2][['W','Y']]
```

Out[59]:

| | W | Y |
|---|---|---|
| **A** | -4 | 13 |
| **B** | 11 | 2 |

# 6. Create a IPython notebook to read and write following types of data using pandas.

(a) CSV (b) Excel (c) XML (d) JSON

## Import libraries

In [1]:
```
import pandas as pd
import numpy as np
```

## (a) CSV

### csv reading:

In [2]:
```
movies_df1 = pd.read_csv('movies.csv')
```

## csv writing:

In [3]:
```
movies_df1.to_csv('movies_copy1.csv', index= False)
```

## Exploratory Analysis:

In [4]:
```
movies_df1.head()
```

Out[4]:

| | movie_name | category | release_date | budget | gross | imdb_rating |
|---|---|---|---|---|---|---|
| **0** | RRR | U/A | 24.03.2022 | 550 | 1100.0 | 8.2 |
| **1** | KGF2 | A | 14.05.2022 | 200 | 1180.0 | 8.7 |
| **2** | Pushpa | U/A | 17.12.2021 | 175 | 322.6 | 7.6 |
| **3** | Akhanda | A | 01.12.2021 | 70 | 133.2 | 8.9 |
| **4** | SVP | U/A | 12.05.2022 | 60 | 175.0 | 6.3 |

In [5]:
```
movies_df1.columns
```

Out[5]:
```
Index(['movie_name', 'category', 'release_date', 'budget', 'gross',
       'imdb_rating'],
      dtype='object')
```

In [6]:
```
movies_df1.shape
```

Out[6]:
```
(7, 6)
```

In [7]:
```
movies_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   movie_name    7 non-null      object
```

```
 1   category      7 non-null      object
 2   release_date  7 non-null      object
 3   budget        7 non-null      int64
 4   gross         7 non-null      float64
 5   imdb_rating   7 non-null      float64
dtypes: float64(2), int64(1), object(3)
memory usage: 464.0+ bytes
```

In [8]:
```
movies_df1.describe()
```

Out[8]:

| | budget | gross | imdb_rating |
|---|---|---|---|
| **count** | 7.000000 | 7.000000 | 7.000000 |
| **mean** | 203.571429 | 456.700000 | 6.985714 |
| **std** | 176.274272 | 471.879437 | 1.896865 |
| **min** | 60.000000 | 131.200000 | 3.800000 |
| **25%** | 70.000000 | 144.050000 | 5.850000 |
| **50%** | 175.000000 | 175.000000 | 7.600000 |
| **75%** | 250.000000 | 711.300000 | 8.450000 |
| **max** | 550.000000 | 1180.000000 | 8.900000 |

In [9]:
```
movies_df1.isnull()
```

Out[9]:

| | movie_name | category | release_date | budget | gross | imdb_rating |
|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False |
| **1** | False | False | False | False | False | False |
| **2** | False | False | False | False | False | False |
| **3** | False | False | False | False | False | False |
| **4** | False | False | False | False | False | False |
| **5** | False | False | False | False | False | False |
| **6** | False | False | False | False | False | False |

In [10]:
```
movies_df1.isnull().count()
```

Out[10]:
```
movie_name      7
category        7
release_date    7
budget          7
gross           7
imdb_rating     7
dtype: int64
```

## (b) Excel

## Excel reading

In [11]:
```
movies_df2 = pd.read_excel('movies.xlsx')
```

## Excel writing

In [12]:
```
movies_df2.to_excel('movies_copy2.xlsx', index=False)
```

## EDA

In [13]:
```
movies_df2.head()
```

Out[13]:

| | movie_name | category | release_date | budget | gross | imdb_rating |
|---|---|---|---|---|---|---|
| **0** | RRR | U/A | 24.03.2022 | 550 | 1100.0 | 8.2 |
| **1** | KGF2 | A | 14.05.2022 | 200 | 1180.0 | 8.7 |
| **2** | Pushpa | U/A | 17.12.2021 | 175 | 322.6 | 7.6 |
| **3** | Akhanda | A | 01.12.2021 | 70 | 133.2 | 8.9 |
| **4** | SVP | U/A | 12.05.2022 | 60 | 175.0 | 6.3 |

In [14]:
```
movies_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   movie_name    7 non-null      object 
 1   category      7 non-null      object 
 2   release_date  7 non-null      object 
 3   budget        7 non-null      int64  
 4   gross         7 non-null      float64
 5   imdb_rating   7 non-null      float64
dtypes: float64(2), int64(1), object(3)
memory usage: 464.0+ bytes
```

## (c) XML : (E)Xtented Markup Language

## XML reading

In [15]:
```
books_df = pd.read_xml('books.xml')
```

## XML writing

In [16]:
```
books_df.to_xml('books_copy.xml', index=False)
```

In [17]:
```
books_df.head()
```

Out[17]:

| | id | author | title | genre | price | publish_date | description |
|---|---|---|---|---|---|---|---|
| **0** | bk101 | Gambardella, Matthew | XML Developer's Guide | Computer | 44.95 | 2000-10-01 | An in-depth look at creating applications \n ... |
| **1** | bk102 | Ralls, Kim | Midnight Rain | Fantasy | 5.95 | 2000-12-16 | A former architect battles corporate zombies, ... |

| | id | author | title | genre | price | publish_date | description |
|---|---|---|---|---|---|---|---|
| **2** | bk103 | Corets, Eva | Maeve Ascendant | Fantasy | 5.95 | 2000-11-17 | After the collapse of a nanotechnology \n ... |
| **3** | bk104 | Corets, Eva | Oberon's Legacy | Fantasy | 5.95 | 2001-03-10 | In post-apocalypse England, the mysterious \n ... |
| **4** | bk105 | Corets, Eva | The Sundered Grail | Fantasy | 5.95 | 2001-09-10 | The two daughters of Maeve, half-sisters, \n ... |

## (d) JSON

## JSON reading

In [18]:
```
sample_df = pd.read_json('sample.json')
```

## JSON writing

In [19]:
```
sample_df.to_json('sample_copy.json')
```

In [20]:
```
sample_df.head()
```

Out[20]:

| | feeds | totalFeed |
|---|---|---|
| **0** | {'id': 2140, 'title': 'gj', 'description': 'gh... | 125 |
| **1** | {'id': 2139, 'title': 'dfg', 'description': 'd... | 125 |
| **2** | {'id': 2138, 'title': 'TEst Video', 'descripti... | 125 |
| **3** | {'id': 2137, 'title': 'QQ', 'description': 'qq... | 125 |
| **4** | {'id': 2136, 'title': 'gj', 'description': 'gh... | 125 |

In [21]:
```
sample_df.shape
```

Out[21]:
```
(100, 2)
```

In [22]:
```
sample_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   feeds      100 non-null    object
 1   totalFeed  100 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 1.7+ KB
```

## Result:

Reading and Writing with different file formats(csv, excel, xml and json) performed successfully.

## 7. Create ona IPyth notebook to perform Data Cleaning on given dataset by Handling and Filling in missing data using pandas.

### import librareies

In [1]:

```
import pandas as pd
import numpy as np
```

### Create a data frame

In [2]:

```
df = pd.DataFrame([[1.5, 2.56, 3.45],
                   [2.36, np.nan, np.nan],
                  [np.nan, 3.65, 4.26],
                  [4.56, 5.67, 3.98]])
```

In [3]:

```
df
```

Out[3]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.50 | 2.56 | 3.45 |
| 1 | 2.36 | NaN | NaN |
| 2 | NaN | 3.65 | 4.26 |
| 3 | 4.56 | 5.67 | 3.98 |

### Identifieng null values

In [4]:

```
df.isnull()
```

Out[4]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | False | False | False |
| 1 | False | True | True |
| 2 | True | False | False |
| 3 | False | False | False |

In [5]:

```
df.isna()
```

Out[5]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | False | False | False |
| 1 | False | True | True |
| 2 | True | False | False |
| 3 | False | False | False |

### Identifieng not null values

In [6]:

```
df.notnull()
```

Out[6]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | True | True | True |
| 1 | True | False | False |
| 2 | False | True | True |
| 3 | True | True | True |

### Filling missing values

In [7]:

```
df.fillna(value=0.5)
```

Out[7]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.50 | 2.56 | 3.45 |
| 1 | 2.36 | 0.50 | 0.50 |
| 2 | 0.50 | 3.65 | 4.26 |
| 3 | 4.56 | 5.67 | 3.98 |

In [8]:

```
df.fillna(value=np.mean(df))
```

Out[8]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.500000 | 2.56 | 3.450000 |
| 1 | 2.360000 | 3.96 | 3.896667 |
| 2 | 2.806667 | 3.65 | 4.260000 |
| 3 | 4.560000 | 5.67 | 3.980000 |

**dropping missing values**

In [9]:

```
df.dropna()
```

Out[9]:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.50 | 2.56 | 3.45 |
| 3 | 4.56 | 5.67 | 3.98 |

**Result:**

Data cleaning is performed successfully by handling and filling missiong values in give data frame or data set

# 8. Create a IPython notebook to perform Data wrangling by Combining and Merging Datasets using pandas.

In [1]:
```
import numpy as np
import pandas as pd
```

## Merging

In [2]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
```

In [3]:
```
df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [4]:
```
df1
```

Out[4]:

| | key | data1 |
|---|---|---|
| **0** | b | 0 |
| **1** | b | 1 |
| **2** | a | 2 |
| **3** | c | 3 |
| **4** | a | 4 |
| **5** | a | 5 |
| **6** | b | 6 |

In [5]:
```
df2
```

Out[5]:

| | key | data2 |
|---|---|---|
| **0** | a | 0 |
| **1** | b | 1 |
| **2** | d | 2 |

In [6]:
```
pd.merge(df1, df2, on='key')
```

Out[6]:

| | key | data1 | data2 |
|---|---|---|---|
| **0** | b | 0 | 1 |
| **1** | b | 1 | 1 |
| **2** | b | 6 | 1 |
| **3** | a | 2 | 0 |
| **4** | a | 4 | 0 |

| | key | data1 | data2 |
|---|---|---|---|
| **5** | a | 5 | 0 |

In [7]:
```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
```

In [8]:
```
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [9]:
```
df3
```

Out[9]:

| | lkey | data1 |
|---|---|---|
| **0** | b | 0 |
| **1** | b | 1 |
| **2** | a | 2 |
| **3** | c | 3 |
| **4** | a | 4 |
| **5** | a | 5 |
| **6** | b | 6 |

In [10]:
```
df4
```

Out[10]:

| | rkey | data2 |
|---|---|---|
| **0** | a | 0 |
| **1** | b | 1 |
| **2** | d | 2 |

In [11]:
```
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[11]:

| | lkey | data1 | rkey | data2 |
|---|---|---|---|---|
| **0** | b | 0 | b | 1 |
| **1** | b | 1 | b | 1 |
| **2** | b | 6 | b | 1 |
| **3** | a | 2 | a | 0 |
| **4** | a | 4 | a | 0 |
| **5** | a | 5 | a | 0 |

## merge- outer join

In [12]:
```
pd.merge(df1, df2, how='outer')
```

Out[12]:

| | key | data1 | data2 |
|---|---|---|---|
| **0** | b | 0.0 | 1.0 |
| **1** | b | 1.0 | 1.0 |
| **2** | b | 6.0 | 1.0 |
| **3** | a | 2.0 | 0.0 |
| **4** | a | 4.0 | 0.0 |
| **5** | a | 5.0 | 0.0 |
| **6** | c | 3.0 | NaN |
| **7** | d | NaN | 2.0 |

## merge- inner join

In [13]:
```
pd.merge(df1, df2, how='inner') # default merge
```

Out[13]:

| | key | data1 | data2 |
|---|---|---|---|
| **0** | b | 0 | 1 |
| **1** | b | 1 | 1 |
| **2** | b | 6 | 1 |
| **3** | a | 2 | 0 |
| **4** | a | 4 | 0 |
| **5** | a | 5 | 0 |

## merge- left join

In [14]:
```
pd.merge(df1, df2, how='left')
```

Out[14]:

| | key | data1 | data2 |
|---|---|---|---|
| **0** | b | 0 | 1.0 |
| **1** | b | 1 | 1.0 |
| **2** | a | 2 | 0.0 |
| **3** | c | 3 | NaN |
| **4** | a | 4 | 0.0 |
| **5** | a | 5 | 0.0 |
| **6** | b | 6 | 1.0 |

## merge- right join

In [15]:
```
pd.merge(df1, df2, how='right')
```

Out[15]:

| | key | data1 | data2 |
|---|---|---|---|

| | key | data1 | data2 |
|---|---|---|---|
| **0** | a | 2.0 | 0 |
| **1** | a | 4.0 | 0 |
| **2** | a | 5.0 | 0 |
| **3** | b | 0.0 | 1 |
| **4** | b | 1.0 | 1 |
| **5** | b | 6.0 | 1 |
| **6** | d | NaN | 2 |

## Combining

In [16]:
```
df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
                   columns=['one', 'two'])
```

In [17]:
```
df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
                   columns=['three', 'four'])
```

In [18]:
```
df1
```

Out[18]:

| | one | two |
|---|---|---|
| **a** | 0 | 1 |
| **b** | 2 | 3 |
| **c** | 4 | 5 |

In [19]:
```
df2
```

Out[19]:

| | three | four |
|---|---|---|
| **a** | 5 | 6 |
| **c** | 7 | 8 |

In [20]:
```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

Out[20]:

| | level1 | | level2 | |
|---|---|---|---|---|
| | **one** | **two** | **three** | **four** |
| **a** | 0 | 1 | 5.0 | 6.0 |
| **b** | 2 | 3 | NaN | NaN |
| **c** | 4 | 5 | 7.0 | 8.0 |

In [21]:
```
pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

Out[21]:

| | level1 | | level2 | |
|---|---|---|---|---|
| | one | two | three | four |
| a | 0 | 1 | 5.0 | 6.0 |
| b | 2 | 3 | NaN | NaN |
| c | 4 | 5 | 7.0 | 8.0 |

In [22]:
```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
```

In [23]:
```
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
```

In [24]:
```
df1.combine_first(df2) #column by column
```

Out[24]:

| | a | b | c |
|---|---|---|---|
| 0 | 1.0 | NaN | 2.0 |
| 1 | 4.0 | 2.0 | 6.0 |
| 2 | 5.0 | 4.0 | 10.0 |
| 3 | 3.0 | 6.0 | 14.0 |
| 4 | 7.0 | 8.0 | NaN |

## Result:

Data wrangling by Combining and Merging Datasets using pandas performed successfully.

# 9. Create a IPython notebook to perform basic Data transformation techniques on given dataset using pandas.

In [1]:
```
import pandas as pd
import numpy as np
```

## Removing Duplicates

In [2]:
```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
```

In [3]:
```
data
```

Out[3]:

| | k1 | k2 |
|---|---|---|
| **0** | one | 1 |
| **1** | two | 1 |
| **2** | one | 2 |
| **3** | two | 3 |
| **4** | one | 3 |
| **5** | two | 4 |
| **6** | two | 4 |

In [4]:
```
data.duplicated()
```

Out[4]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [5]:
```
data.drop_duplicates()
```

Out[5]:

| | k1 | k2 |
|---|---|---|
| **0** | one | 1 |
| **1** | two | 1 |
| **2** | one | 2 |
| **3** | two | 3 |
| **4** | one | 3 |
| **5** | two | 4 |

## Replacing Values

In [6]:
```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
```

In [7]:
```
data
```

Out[7]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [8]:
```
data.replace(-999, np.nan)
```

Out[8]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [9]:
```
#replace multiple values at once
data.replace([-999, -1000], np.nan)
```

Out[9]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [10]:
```
#different replacement for each value
data.replace([-999, -1000], [np.nan, 0])
```

Out[10]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [11]:
```
#argument passed can also be a dict
data.replace({-999: np.nan, -1000: 0})
```

Out[11]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

## Renaming Axis Indexes

In [12]:
```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),
                    index=['Ohio', 'Colorado', 'New York'],
```

```
                    columns=['one', 'two', 'three', 'four'])
```

In [13]:
```
data
```

Out[13]:

| | one | two | three | four |
|---|---|---|---|---|
| **Ohio** | 0 | 1 | 2 | 3 |
| **Colorado** | 4 | 5 | 6 | 7 |
| **New York** | 8 | 9 | 10 | 11 |

In [14]:
```
transform = lambda x: x[:4].upper()
```

In [15]:
```
data.index = data.index.map(transform)
```

In [16]:
```
data
```

Out[16]:

| | one | two | three | four |
|---|---|---|---|---|
| **OHIO** | 0 | 1 | 2 | 3 |
| **COLO** | 4 | 5 | 6 | 7 |
| **NEW** | 8 | 9 | 10 | 11 |

In [17]:
```
data.rename(index=str.title, columns=str.upper)
```

Out[17]:

| | ONE | TWO | THREE | FOUR |
|---|---|---|---|---|
| **Ohio** | 0 | 1 | 2 | 3 |
| **Colo** | 4 | 5 | 6 | 7 |
| **New** | 8 | 9 | 10 | 11 |

In [18]:
```
data.rename(index={'OHIO': 'INDIANA'}, columns={'three': 'thirty'})
```

Out[18]:

| | one | two | thirty | four |
|---|---|---|---|---|
| **INDIANA** | 0 | 1 | 2 | 3 |
| **COLO** | 4 | 5 | 6 | 7 |
| **NEW** | 8 | 9 | 10 | 11 |

In [19]:
```
data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
```

In [20]:
```
data
```

Out[20]:

| | one | two | three | four |
|---|---|---|---|---|

| | one | two | three | four |
|---|---|---|---|---|
| **INDIANA** | 0 | 1 | 2 | 3 |
| **COLO** | 4 | 5 | 6 | 7 |
| **NEW** | 8 | 9 | 10 | 11 |

## Discretization and Binning

In [21]:
```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [22]:
```
bins = [18, 25, 35, 60, 100]
```

In [23]:
```
cats = pd.cut(ages, bins)
```

In [24]:
```
cats
```

Out[24]:
```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 6
0], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

In [25]:
```
cats.codes
```

Out[25]:
```
array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

In [26]:
```
cats.categories
```

Out[26]:
```
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int64, righ
t]')
```

In [27]:
```
pd.value_counts(cats)
```

Out[27]:
```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
dtype: int64
```

In [28]:
```
#giving labels for bins
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
pd.cut(ages, bins, labels=group_names)
```

Out[28]:
```
['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior', 'Mid
dleAged', 'MiddleAged', 'YoungAdult']
Length: 12
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

## Computing Indicator/Dummy Variables

In [29]:
```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                   'data1': range(6)})
```

In [30]:
```
df
```

Out[30]:

| | key | data1 |
|---|---|---|
| **0** | b | 0 |
| **1** | b | 1 |
| **2** | a | 2 |
| **3** | c | 3 |
| **4** | a | 4 |
| **5** | b | 5 |

In [31]:
```
pd.get_dummies(df['key'])
```

Out[31]:

| | a | b | c |
|---|---|---|---|
| **0** | 0 | 1 | 0 |
| **1** | 0 | 1 | 0 |
| **2** | 1 | 0 | 0 |
| **3** | 0 | 0 | 1 |
| **4** | 1 | 0 | 0 |
| **5** | 0 | 1 | 0 |

In [32]:
```
dummies = pd.get_dummies(df['key'])
```

In [33]:
```
df_with_dummy = df[['data1']].join(dummies)
```

In [34]:
```
df_with_dummy
```

Out[34]:

| | data1 | a | b | c |
|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 |
| **1** | 1 | 0 | 1 | 0 |
| **2** | 2 | 1 | 0 | 0 |
| **3** | 3 | 0 | 0 | 1 |
| **4** | 4 | 1 | 0 | 0 |
| **5** | 5 | 0 | 1 | 0 |

In [38]:
```
dummies1 = pd.get_dummies(df['key'], prefix='key_')
```

In [41]:
```
df_with_dummy1 = df[['data1']].join(dummies1)
```

In [42]:
```
df_with_dummy1
```

Out[42]:

| | data1 | key__a | key__b | key__c |
|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 |
| **1** | 1 | 0 | 1 | 0 |
| **2** | 2 | 1 | 0 | 0 |
| **3** | 3 | 0 | 0 | 1 |
| **4** | 4 | 1 | 0 | 0 |
| **5** | 5 | 0 | 1 | 0 |

## Result:

Data transformation techniques on given dataset using pandas performed successfully.

# 10. Create a IPython notebook to work with matplotlib (a) Plots (b) Subplots

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

In [2]:
```
%matplotlib inline
```

In [3]:
```
x = np.linspace(0,5,11)
y = x ** 2
```

In [4]:
```
x
```

Out[4]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

In [5]:
```
y
```

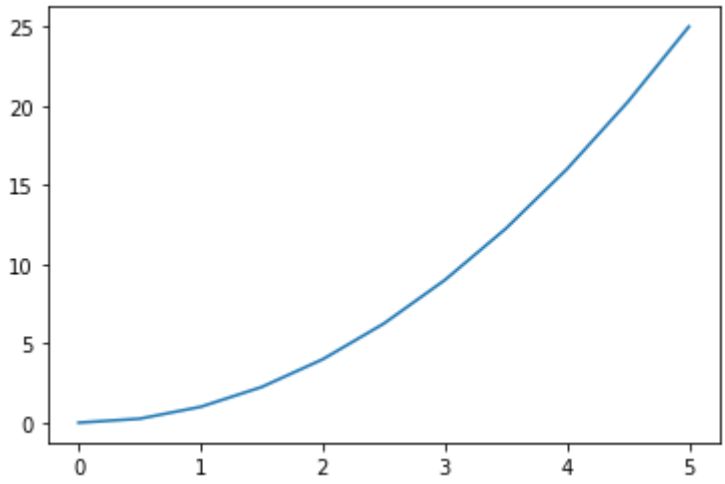
Out[5]:
```
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
       20.25, 25.  ])
```

## Functional Oriented Approach

### plots

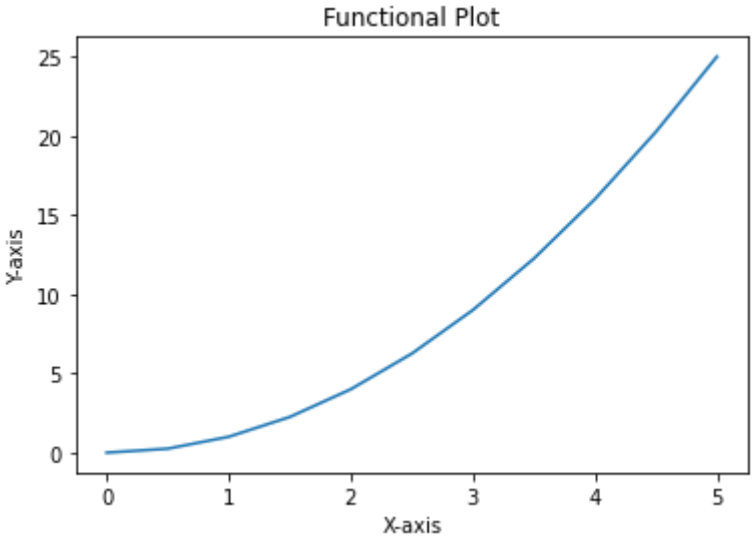
In [6]:
```
plt.plot(x,y)
#plt.show()
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x2273d541310>]
```




In [7]:
```
plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Functional Plot')
#plt.show()
```

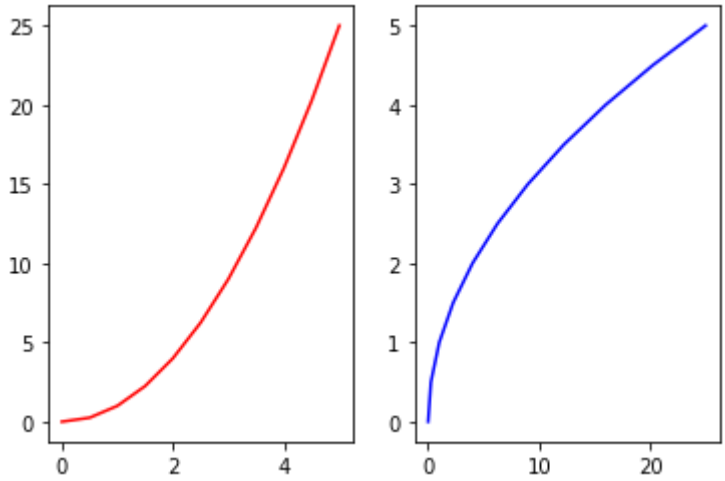
Out[7]:
```
Text(0.5, 1.0, 'Functional Plot')
```




## Subplots

In [8]:

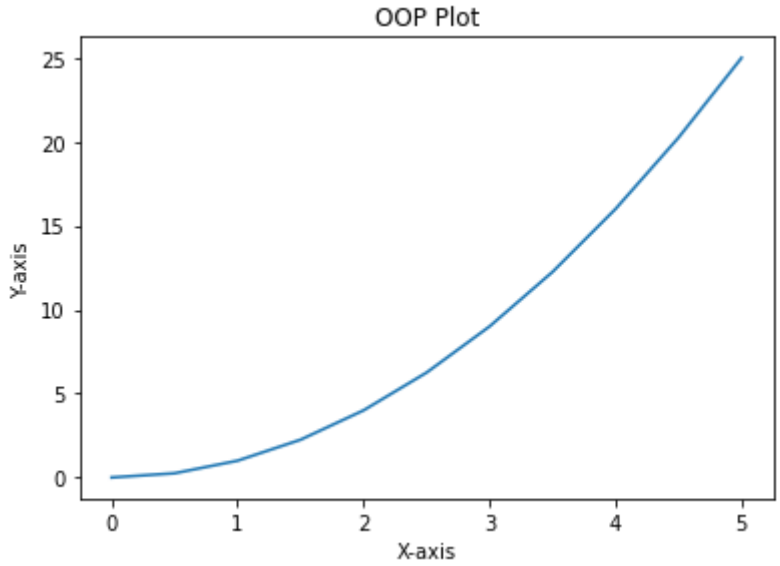
```
plt.subplot(1,2,1) # nrows, ncols, index
plt.plot(x,y,'r')

plt.subplot(1,2,2)
plt.plot(y,x,'b')
plt.show()
```




## Object Oriented Approach

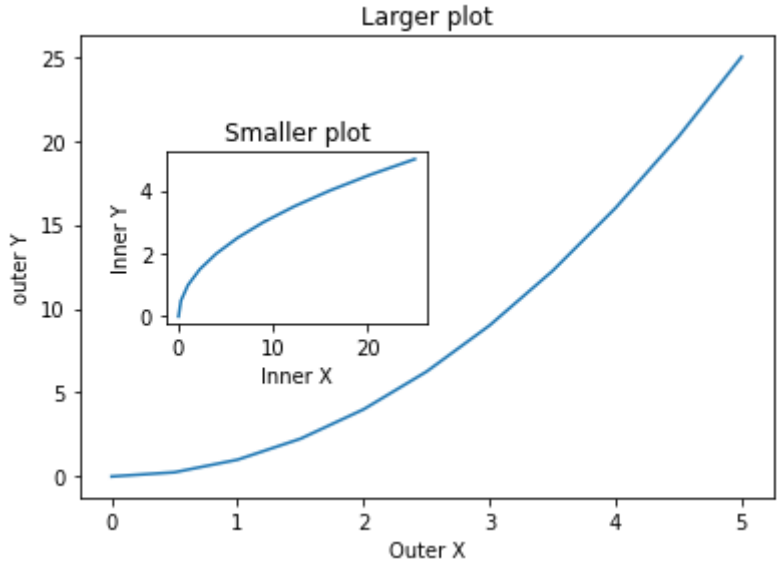
In [9]:

```
fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) #[left, bottom, width, height] range from
axes.plot(x,y)
axes.set_xlabel('X-axis')
axes.set_ylabel('Y-axis')
axes.set_title('OOP Plot')
plt.show()
```



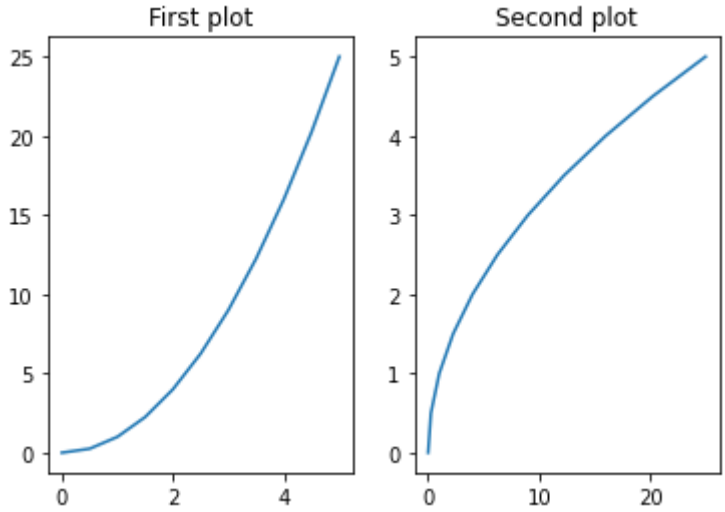

In [10]:

```
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes2 = fig.add_axes([0.2, 0.4, 0.3, 0.3])

axes1.plot(x,y)
axes1.set_title('Larger plot')
axes1.set_xlabel('Outer X')
axes1.set_ylabel('outer Y')

axes2.plot(y,x)
axes2.set_title('Smaller plot')
axes2.set_xlabel('Inner X')
axes2.set_ylabel('Inner Y')
plt.show()
```



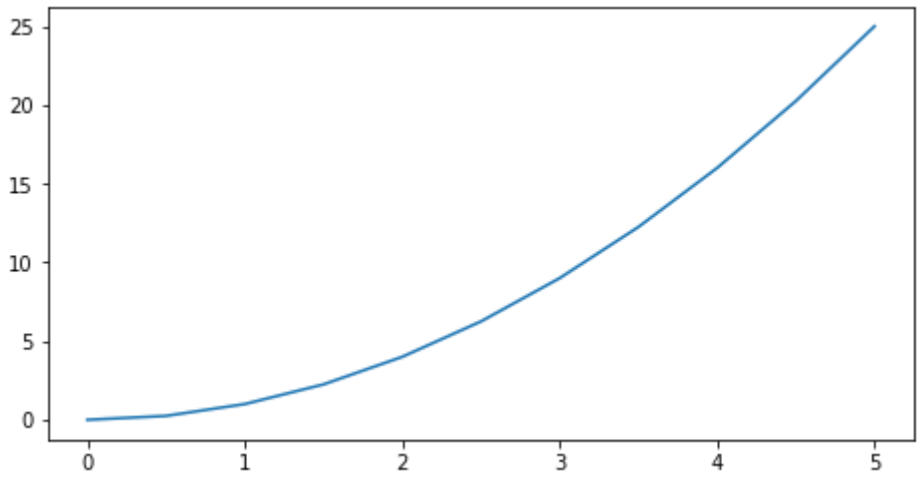

In [11]:

```
fig, axes = plt.subplots(nrows=1, ncols=2)

axes[0].plot(x,y)
axes[0].set_title('First plot')

axes[1].plot(y,x)
axes[1].set_title('Second plot')
plt.show()
```



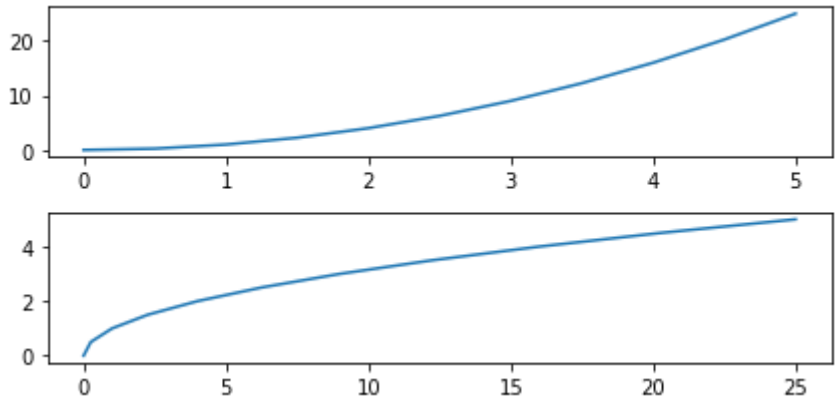

In [12]:

```
fig = plt.figure(figsize=(6,3))
axes = fig.add_axes([0,0,1,1])
axes.plot(x,y)
plt.show()
```



In [13]:

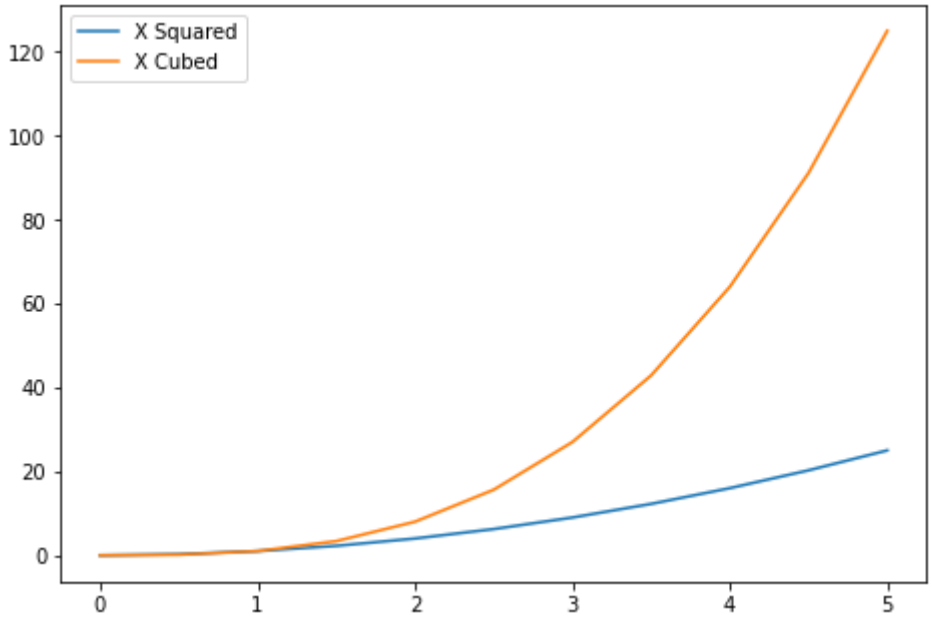
```
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(6,3))
axes[0].plot(x,y)
axes[1].plot(y,x)
plt.tight_layout()
```



In [14]:

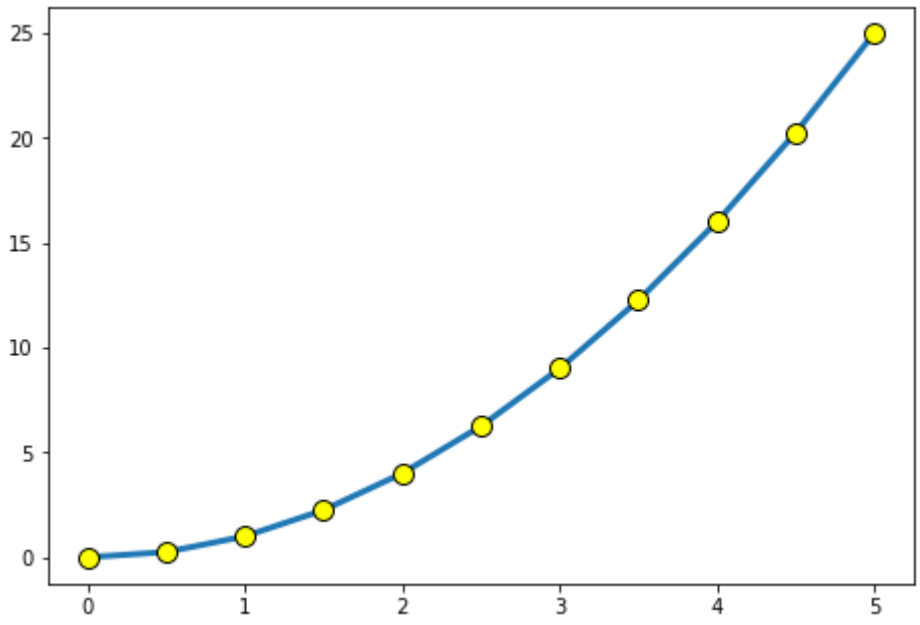
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,x**2, label='X Squared')
ax.plot(x,x**3, label='X Cubed')
```

```
plt.legend()
plt.show()
```



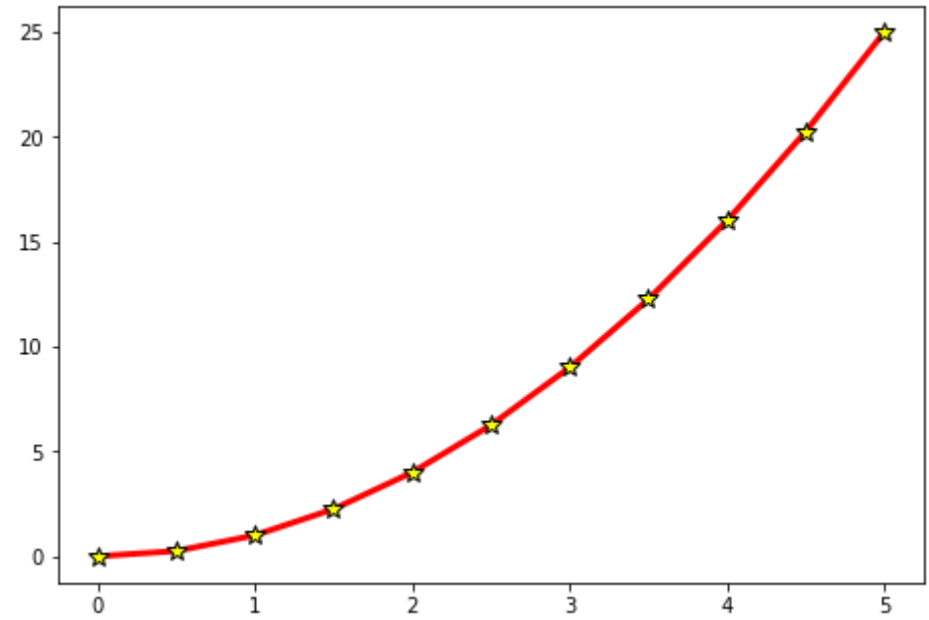

In [15]:

```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax1.plot(x,y, linestyle='-', linewidth=3, marker='o',
         markeredgecolor='black',
         markerfacecolor ='yellow',
         markersize=10)
plt.show()
```




In [16]:

```
fig = plt.figure()
ax1 = fig.add_axes([0,0,1,1])
ax1.plot(x,y, linestyle='-', linewidth=3, marker='*',
         color='red',
         markerfacecolor ='yellow',
         markeredgecolor='black',
         markersize=10)
plt.show()
```




for more line styles refer: https://matplotlib.org/2.0.1/api/lines_api.html

## Result:

Plots and subplots using matplotlib created successfully.

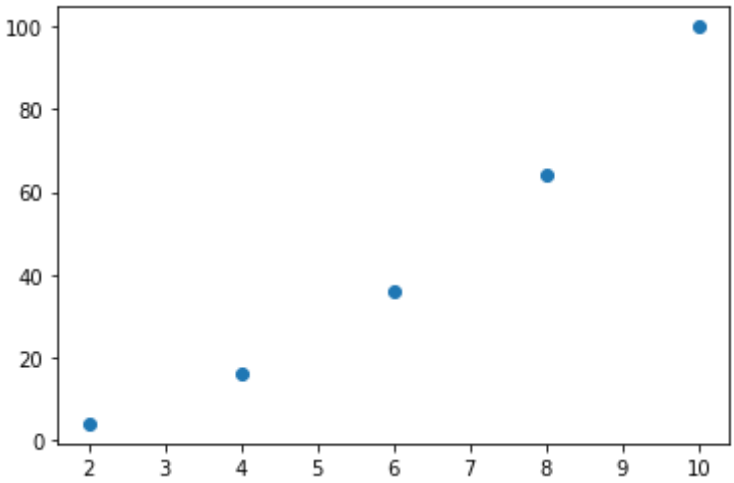
# 12. Create a IPython notebook to visualize different types of plots using matplotlib.

## Importing Libraries

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

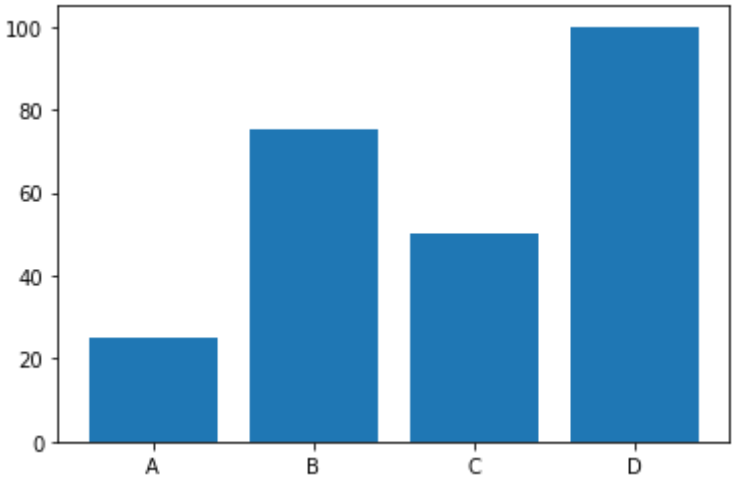
## Scatter Plot

In [2]:
```
x =[2, 4, 6, 8, 10]
y =[4, 16, 36, 64, 100]
plt.scatter(x,y)
plt.show()
```




## Bar Plot

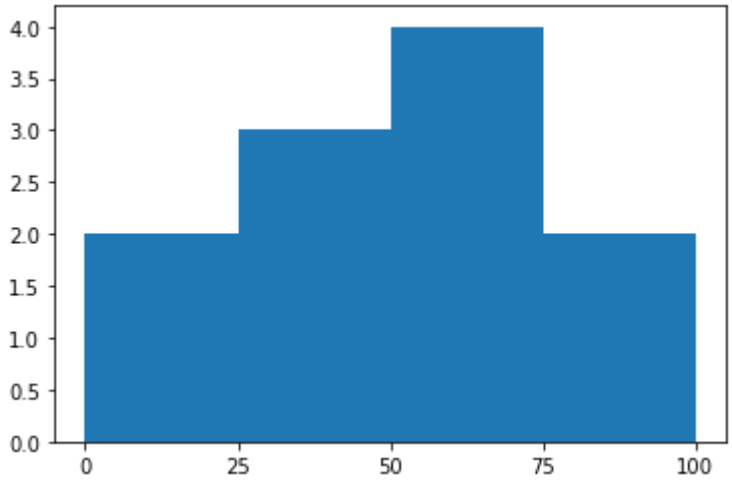
In [3]:
```
x = ['A','B','C','D']
y = [25, 75, 50, 100]
plt.bar(x,y)
plt.show()
```




## Histogram

In [4]:
```
x = [21,43,56,58,45,54,36,20,51,90,85]
```

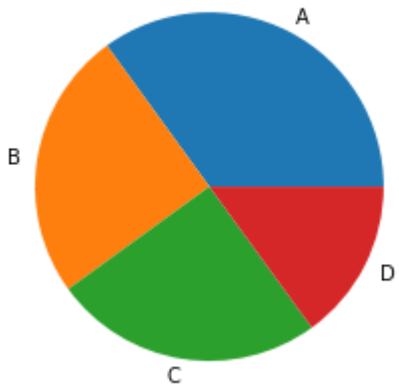
```
plt.hist(x, bins=[0,25,50,75,100])
plt.xticks([0, 25, 50, 75, 100])
plt.show()
```




## Pie Chart

In [5]:

```
x = np.array([35, 25, 25, 15])
mylabels = ['A', 'B', 'C', 'D']
plt.pie(x, labels= mylabels)
plt.show()
```




## Result: Different plots using matplotlib plotted successfully

## 12. Create a IPython notebook to visualize different types of plots on given dataset using seaborn.

In [1]:
```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
```

In [2]:
```
x = np.linspace(0,5,11)
y = x ** 3
```

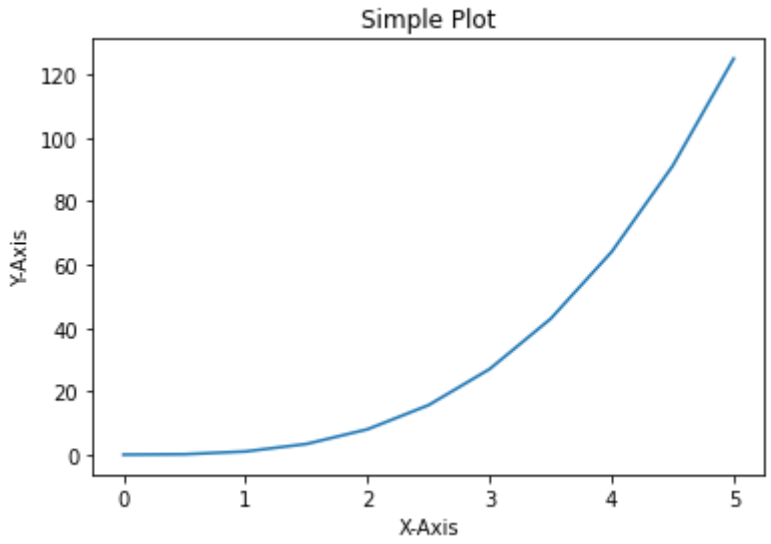
In [3]:
```
x
```

Out[3]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

In [4]:
```
y
```

Out[4]:
```
array([  0.   ,   0.125,   1.   ,   3.375,   8.   ,  15.625,  27.   ,
        42.875,  64.   ,  91.125, 125.   ])
```

In [5]:
```
plt.plot(x,y)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.title('Simple Plot')
plt.show()
```




In [6]:
```
data = pd.Series(np.random.randint(100, 200, size=9),
                 index=[['a','a','a','b','b','c','c','d','d'],
                        [1, 2, 3, 1, 3, 1, 2, 2, 3]])
```

In [7]:
```
data
```

Out[7]:
```
a  1    101
   2    144
   3    198
b  1    127
   3    103
```

```
c  1    197
   2    169
d  2    190
   3    183
dtype: int32
```

In [8]:
```
data.index
```

Out[8]:
```
MultiIndex([('a', 1),
            ('a', 2),
            ('a', 3),
            ('b', 1),
            ('b', 3),
            ('c', 1),
            ('c', 2),
            ('d', 2),
            ('d', 3)],
           )
```

In [9]:
```
data['b']
```

Out[9]:
```
1    127
3    103
dtype: int32
```

In [10]:
```
data['b']
```

Out[10]:
```
1    127
3    103
dtype: int32
```

In [11]:
```
data.loc['b']
```

Out[11]:
```
1    127
3    103
dtype: int32
```

In [12]:
```
data['b':'c']
```

Out[12]:
```
b  1    127
   3    103
c  1    197
   2    169
dtype: int32
```

In [13]:
```
data.loc[['b','d']]
```

Out[13]:
```
b  1    127
   3    103
d  2    190
   3    183
dtype: int32
```

In [14]:
```
data.loc[:,3]
```

Out[14]:
```
a    198
b    103
d    183
dtype: int32
```

In [15]:
```
data.unstack()
```

Out[15]:

| | 1 | 2 | 3 |
|---|---|---|---|
| **a** | 101.0 | 144.0 | 198.0 |
| **b** | 127.0 | NaN | 103.0 |
| **c** | 197.0 | 169.0 | NaN |
| **d** | NaN | 190.0 | 183.0 |

In [16]:
```
data.unstack().stack()
```

Out[16]:
```
a  1    101.0
   2    144.0
   3    198.0
b  1    127.0
   3    103.0
c  1    197.0
   2    169.0
d  2    190.0
   3    183.0
dtype: float64
```

In [17]:
```
frame = pd.DataFrame(np.arange(16).reshape((4, 4)),
                     index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                     columns=[['Hyd', 'Hyd', 'Vizag', 'Vizag'],
                              ['Green', 'Red', 'Green','Blue']])
```

In [18]:
```
frame
```

Out[18]:

| | | Hyd | | Vizag | |
|---|---|---|---|---|---|
| | | **Green** | **Red** | **Green** | **Blue** |
| **a** | **1** | 0 | 1 | 2 | 3 |
| | **2** | 4 | 5 | 6 | 7 |
| **b** | **1** | 8 | 9 | 10 | 11 |
| | **2** | 12 | 13 | 14 | 15 |

In [19]:
```
frame.index.names = ['key1', 'key2']
```

In [20]:
```
frame.columns.names = ['state', 'color']
```

In [21]:
```
frame
```

Out[21]:

| | **state** | **Hyd** | | **Vizag** | |
|---|---|---|---|---|---|
| | **color** | **Green** | **Red** | **Green** | **Blue** |
| **key1** | **key2** | | | | |
| **a** | **1** | 0 | 1 | 2 | 3 |

| state | | Hyd | | Vizag | |
|---|---|---|---|---|---|
| color | | Green | Red | Green | Blue |
| key1 | key2 | | | | |
| | 2 | 4 | 5 | 6 | 7 |
| b | 1 | 8 | 9 | 10 | 11 |
| | 2 | 12 | 13 | 14 | 15 |

In [22]:
```
frame['Hyd']
```

Out[22]:

| color | | Green | Red |
|---|---|---|---|
| key1 | key2 | | |
| a | 1 | 0 | 1 |
| | 2 | 4 | 5 |
| b | 1 | 8 | 9 |
| | 2 | 12 | 13 |

In [23]:
```
frame.loc['a']['Hyd']['Green']
```

Out[23]:
```
key2
1    0
2    4
Name: Green, dtype: int32
```

In [24]:
```
frame['Vizag'].loc['b']
```

Out[24]:

| color | Green | Blue |
|---|---|---|
| key2 | | |
| 1 | 10 | 11 |
| 2 | 14 | 15 |

In [25]:
```
import seaborn as sns
sns.set()
```

In [26]:
```
tips = sns.load_dataset('tips')
```

In [27]:
```
tips.head()
```

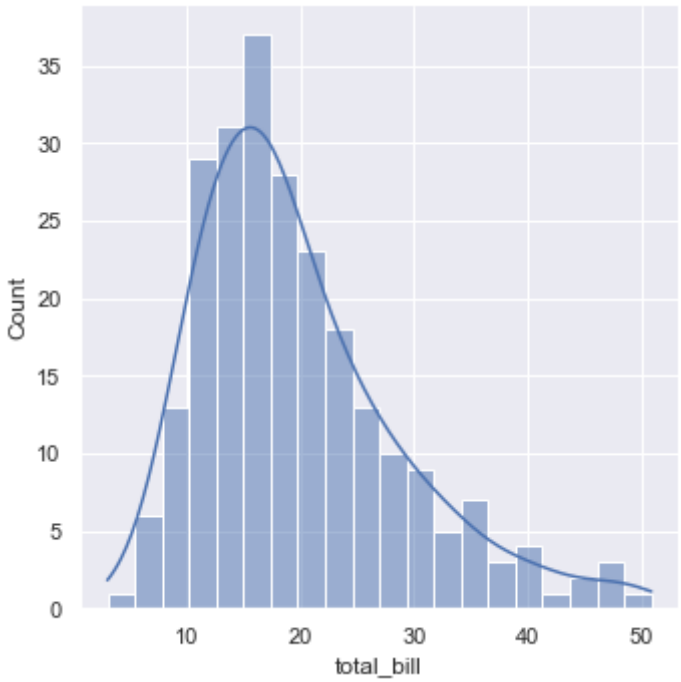
Out[27]:

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 1 | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 2 | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

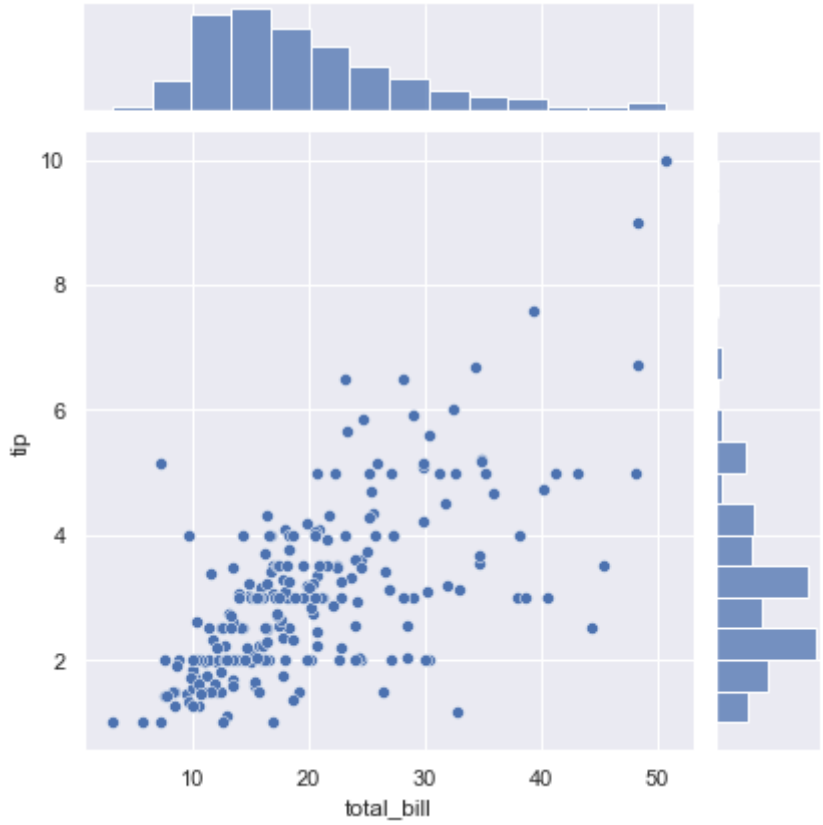
In [28]:
```
sns.displot(tips['total_bill'], bins=20, kde=True)
```

Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x23258dfad90>
```



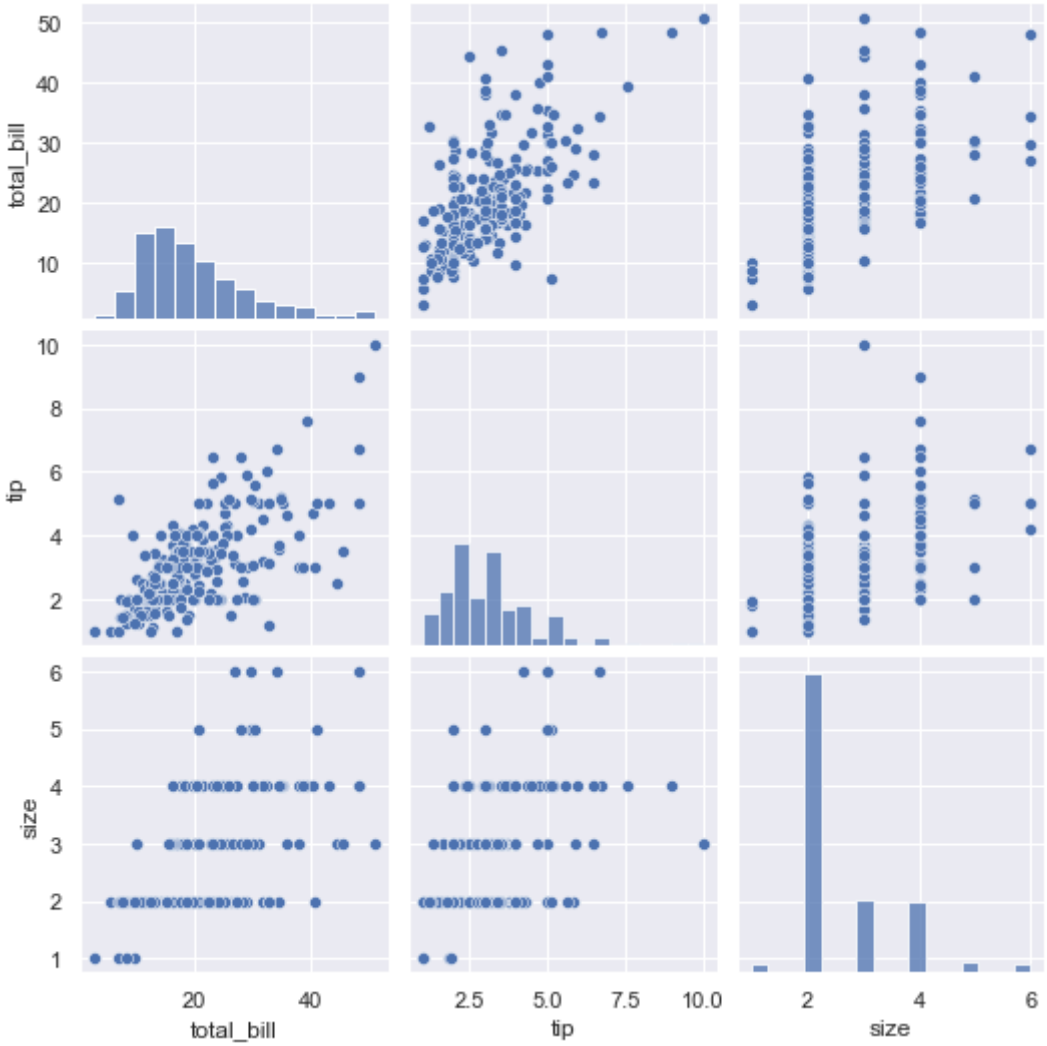

In [29]:
```
sns.jointplot(x='total_bill', y='tip', data=tips)
```

Out[29]:
```
<seaborn.axisgrid.JointGrid at 0x23258e02190>
```



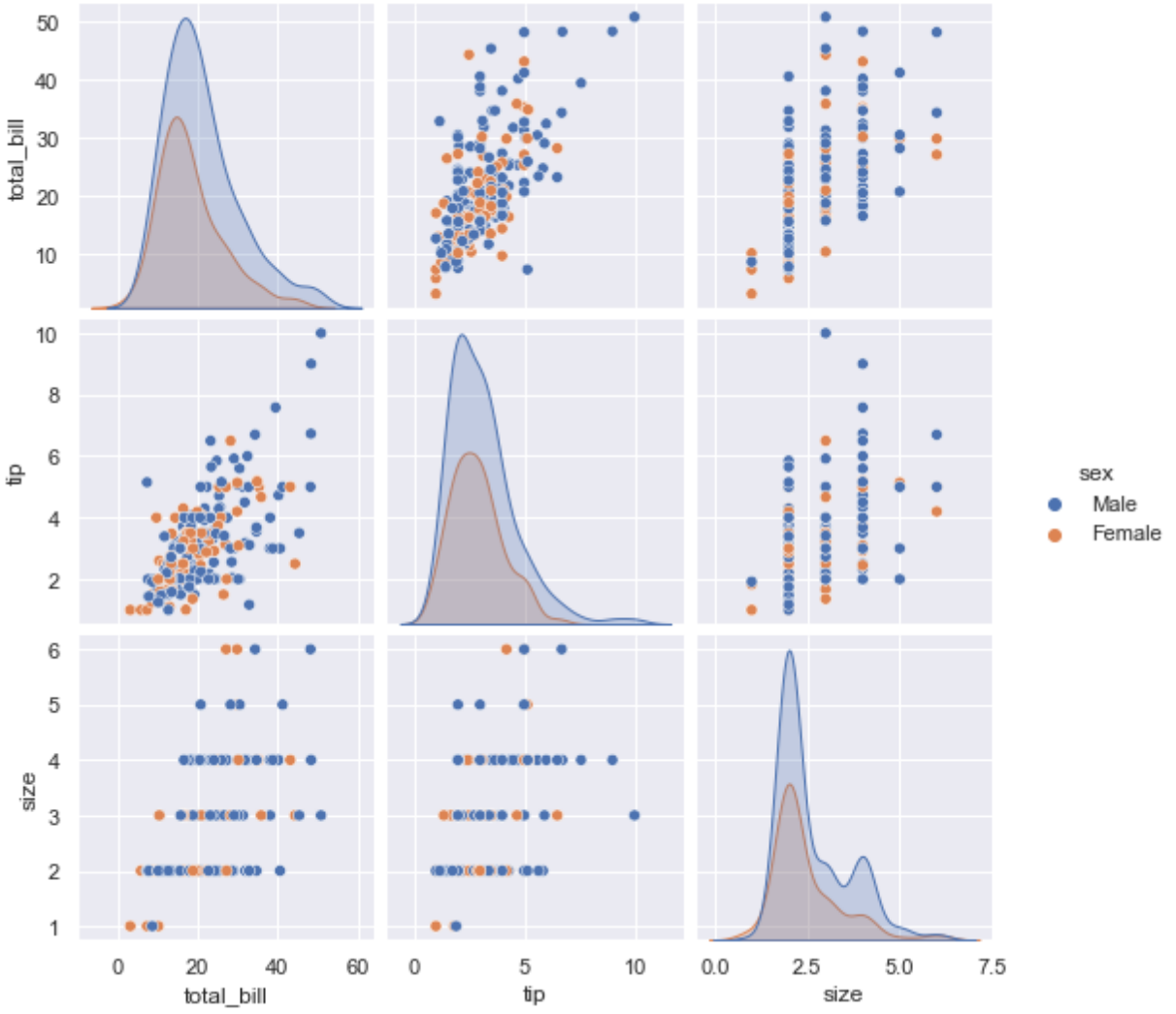

In [30]:
```
sns.pairplot(data=tips)
```

Out[30]:
```
<seaborn.axisgrid.PairGrid at 0x2325e7acb50>
```



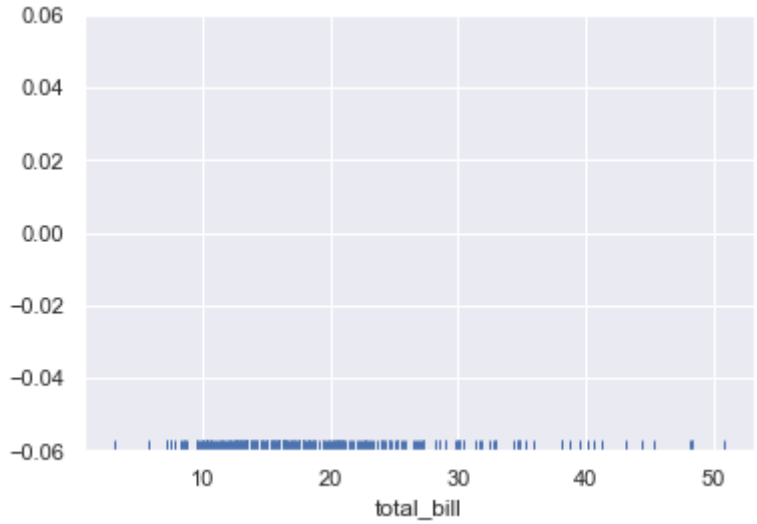

In [31]:
```
sns.pairplot(data=tips, hue='sex')
```

Out[31]:
```
<seaborn.axisgrid.PairGrid at 0x2325ed7c700>
```



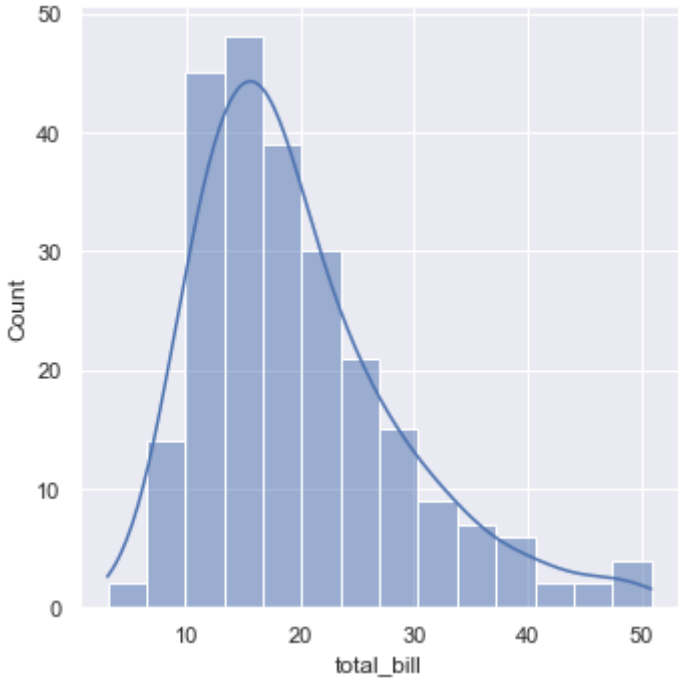

In [32]:
```
sns.rugplot(tips['total_bill'])
```

Out[32]:
```
<AxesSubplot:xlabel='total_bill'>
```



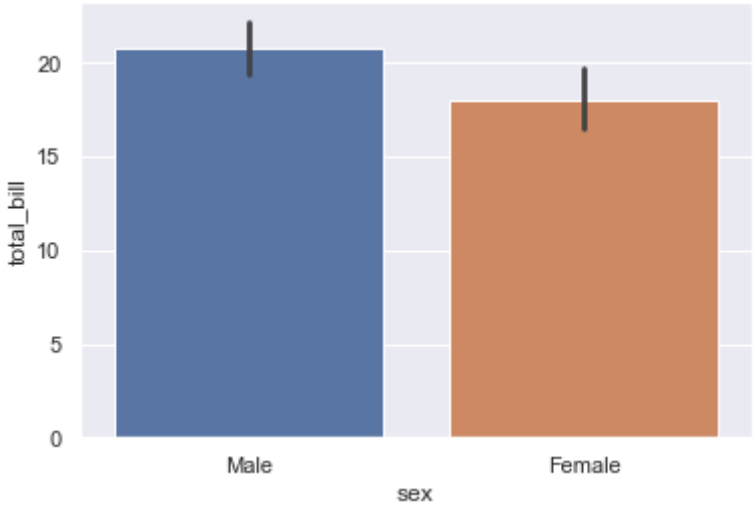

In [33]:
```
sns.displot(tips['total_bill'], kde=True)
```

Out[33]:
```
<seaborn.axisgrid.FacetGrid at 0x2325f459280>
```



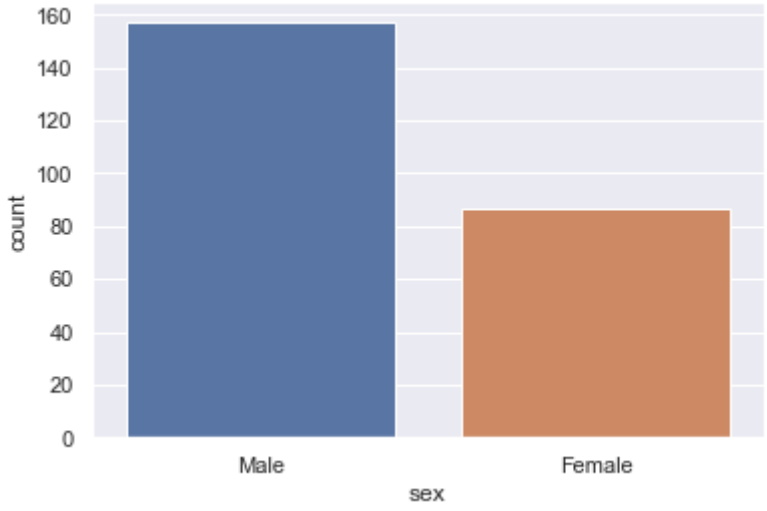

In [34]:
```
sns.barplot(x='sex', y='total_bill', data=tips)
```

Out[34]:
```
<AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



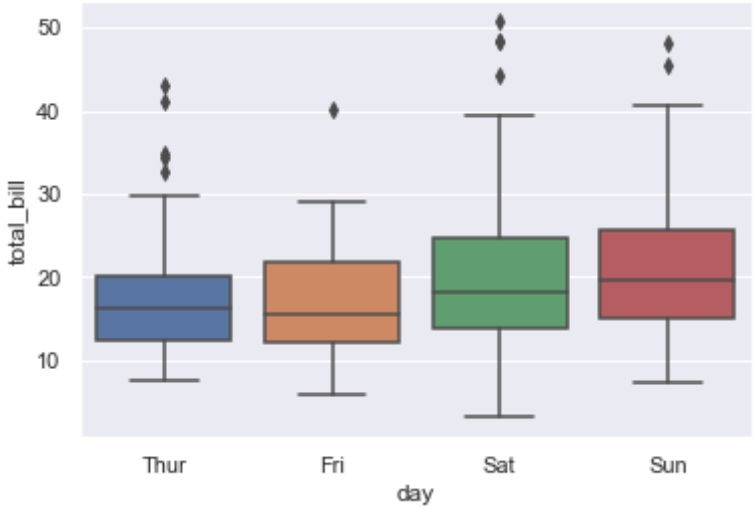

In [35]:
```
sns.countplot(x='sex', data=tips)
```

Out[35]:
```
<AxesSubplot:xlabel='sex', ylabel='count'>
```



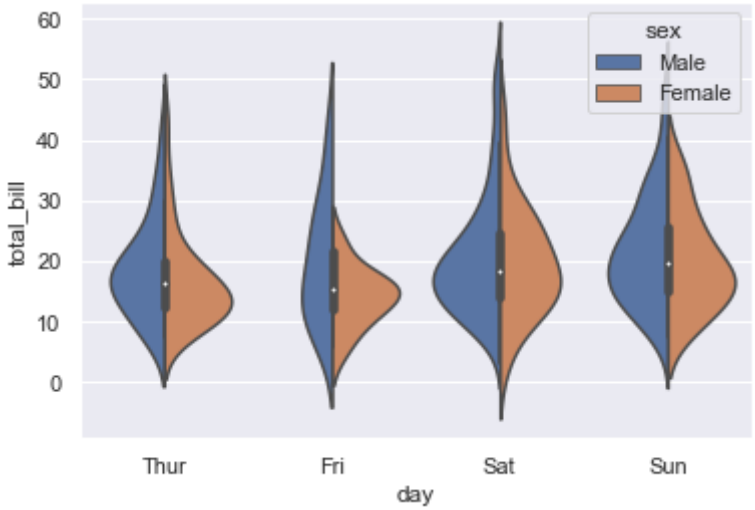

In [36]:
```
sns.boxplot(x='day',y='total_bill', data=tips)
```

Out[36]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



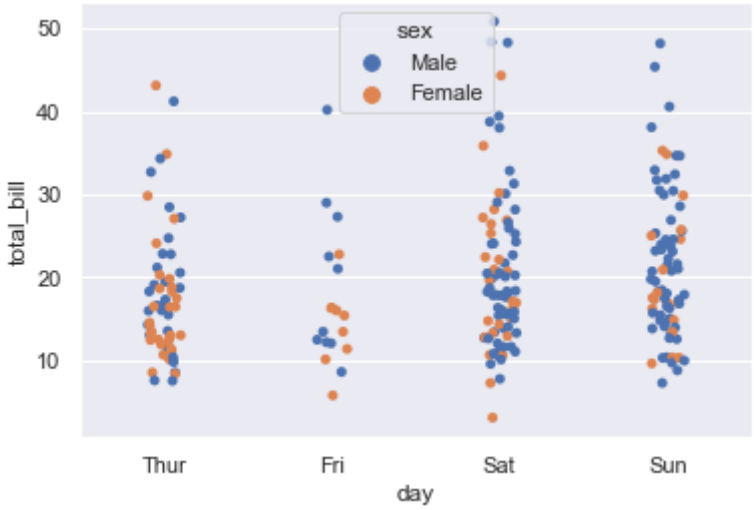

In [37]:
```
sns.violinplot(x='day',y='total_bill', data=tips, hue='sex', split=True)
```

Out[37]:
```
<AxesSubplot:xlabel='day', ylabel='total_bill'>
```



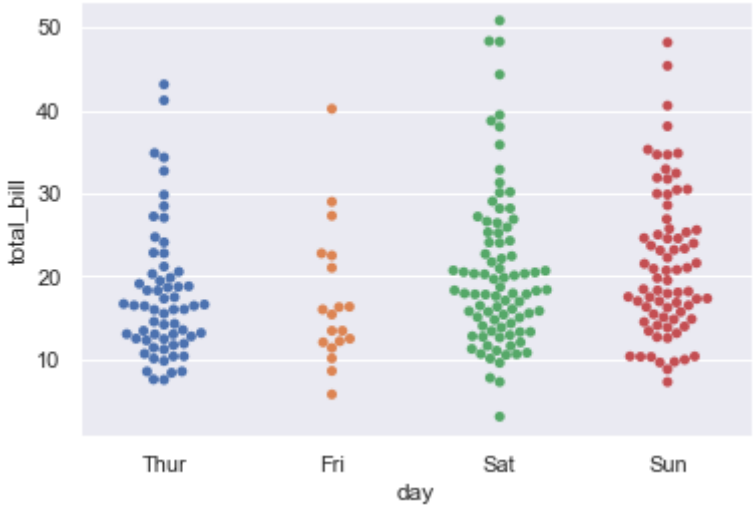

In [38]:
```
sns.stripplot(x='day',y='total_bill', data=tips, hue='sex')
```

Out[38]: <AxesSubplot:xlabel='day', ylabel='total_bill'>




In [39]:
```
sns.swarmplot(x='day',y='total_bill', data=tips)
```

Out[39]: <AxesSubplot:xlabel='day', ylabel='total_bill'>




In [ ]:

# 13. Create a IPython notebook to create Linear Regression Model on given dataset using sklearn

## Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

## Importing the dataset

In [2]:
```
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [3]:
```
print(X)
```

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
 [67532.53 105751.03 304768.73 'Florida']
 [77044.01 99281.34 140574.81 'New York']
 [64664.71 139553.16 137962.62 'California']
 [75328.87 144135.98 134050.07 'Florida']
 [72107.6 127864.55 353183.81 'New York']
 [66051.52 182645.56 118148.2 'Florida']
 [65605.48 153032.06 107138.38 'New York']
 [61994.48 115641.28 91131.24 'Florida']
 [61136.38 152701.92 88218.23 'New York']
 [63408.86 129219.61 46085.25 'California']
 [55493.95 103057.49 214634.81 'Florida']
 [46426.07 157693.92 210797.67 'California']
 [46014.02 85047.44 205517.64 'New York']
 [28663.76 127056.21 201126.82 'Florida']
 [44069.95 51283.14 197029.42 'California']
 [20229.59 65947.93 185265.1 'New York']
 [38558.51 82982.09 174999.3 'California']
 [28754.33 118546.05 172795.67 'California']
```

```
[27892.92 84710.77 164470.71 'Florida']
[23640.93 96189.63 148001.11 'California']
[15505.73 127382.3 35534.17 'New York']
[22177.74 154806.14 28334.72 'California']
[1000.23 124153.04 1903.93 'New York']
[1315.46 115816.21 297114.46 'Florida']
[0.0 135426.92 0.0 'California']
[542.05 51743.15 0.0 'New York']
[0.0 116983.8 45173.06 'California']]
```

## Encoding categorical data

In [4]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])], remainder='
X = np.array(ct.fit_transform(X))
```

In [5]:

```
print(X)
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]
 [0.0 1.0 0.0 67532.53 105751.03 304768.73]
 [0.0 0.0 1.0 77044.01 99281.34 140574.81]
 [1.0 0.0 0.0 64664.71 139553.16 137962.62]
 [0.0 1.0 0.0 75328.87 144135.98 134050.07]
 [0.0 0.0 1.0 72107.6 127864.55 353183.81]
 [0.0 1.0 0.0 66051.52 182645.56 118148.2]
 [0.0 0.0 1.0 65605.48 153032.06 107138.38]
 [0.0 1.0 0.0 61994.48 115641.28 91131.24]
 [0.0 0.0 1.0 61136.38 152701.92 88218.23]
 [1.0 0.0 0.0 63408.86 129219.61 46085.25]
 [0.0 1.0 0.0 55493.95 103057.49 214634.81]
 [1.0 0.0 0.0 46426.07 157693.92 210797.67]
 [0.0 0.0 1.0 46014.02 85047.44 205517.64]
 [0.0 1.0 0.0 28663.76 127056.21 201126.82]
 [1.0 0.0 0.0 44069.95 51283.14 197029.42]
 [0.0 0.0 1.0 20229.59 65947.93 185265.1]
 [1.0 0.0 0.0 38558.51 82982.09 174999.3]
 [1.0 0.0 0.0 28754.33 118546.05 172795.67]
 [0.0 1.0 0.0 27892.92 84710.77 164470.71]
 [1.0 0.0 0.0 23640.93 96189.63 148001.11]
```

```
[0.0 0.0 1.0 15505.73 127382.3 35534.17]
[1.0 0.0 0.0 22177.74 154806.14 28334.72]
[0.0 0.0 1.0 1000.23 124153.04 1903.93]
[0.0 1.0 0.0 1315.46 115816.21 297114.46]
[1.0 0.0 0.0 0.0 135426.92 0.0]
[0.0 0.0 1.0 542.05 51743.15 0.0]
[1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

## Splitting the dataset into the Training set and Test set

In [6]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
```

## Training the Multiple Linear Regression model on the Training set

In [7]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[7]:

```
LinearRegression()
```

## Predicting the Test set results

In [8]:

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),
```

```
[[103015.2  103282.38]
 [132582.28 144259.4 ]
 [132447.74 146121.95]
 [ 71976.1   77798.83]
 [178537.48 191050.39]
 [116161.24 105008.31]
 [ 67851.69  81229.06]
 [ 98791.73  97483.56]
 [113969.44 110352.25]
 [167921.07 166187.94]]
```

# 14. Create a IPython notebook to create Logistic Regression (Binary Classification) Model on given dataset using sklearn.

## Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

## Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

## Splitting the dataset into the Training set and Test set

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_s
```

## Feature Scaling

In [4]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## Training the Logistic Regression model on the Training set

In [5]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[5]:
```
LogisticRegression(random_state=0)
```

## Predicting a new result

In [6]:
```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

# Predicting the Test set results

In [7]:
```
y_pred = classifier.predict(X_test)
```

# Making the Confusion Matrix

In [8]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)
accuracy_score(y_test, y_pred)
```

Out[8]:
```
[[65  3]
 [ 8 24]]
0.89
```

**Result: Logistic regression implemented successfully.**