# ARTIFICIAL INTELLIGENCE

# PRACTICALS

## WEEK 01 :

BASIC LOCAL GIT OPERATIONS :

**Git: configurations**

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --global color.ui true
$ git config --list
```

**Git: starting a repository**

```
$ git init
$ git status
```

**Git: staging files**

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

**Git: committing to a repository**

```
$ git commit -m "Add three files"
$ git reset --soft HEAD^
$ git commit --amend -m <enter your message>
```

**Git: pulling and pushing from and to repositories**

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

**Git: branching**

```
$ git branch
$ git branch <branch-name>
$ git checkout <branch-name>
$ git merge <branch-name>
$ git checkout -b <branch-name>
```

### 1. 1.Creating a Git Repository (in git bash)

- Create a directory to contain the project.

- Go into the new directory.
- Type git init .
- Write some code.
- Type git add to add the files (see the typical use page).
- Type git commit .

Command : mkdir foldername

## 2. Creating a Git Repository in GitHub

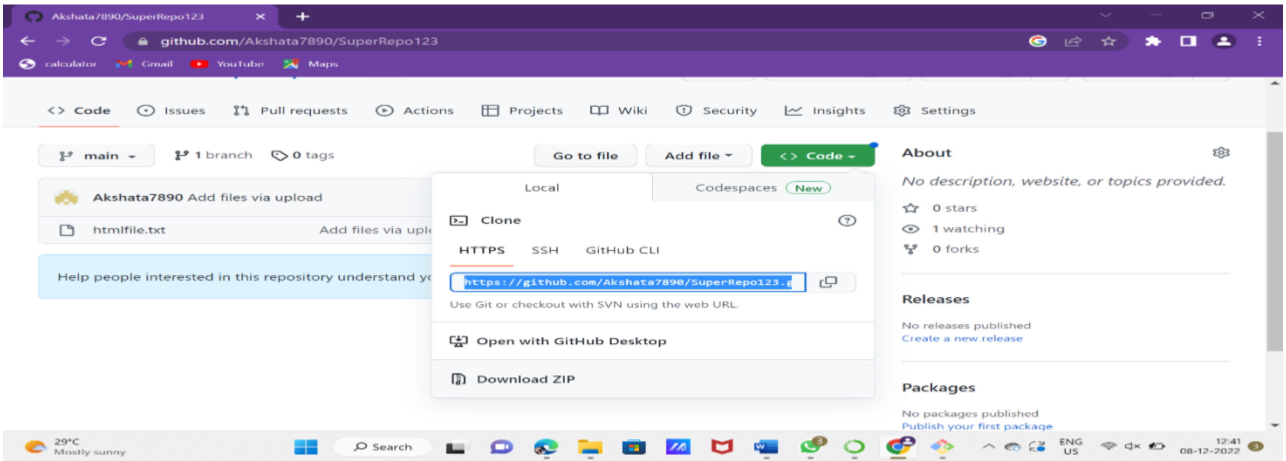
- In the upper-right corner of any page, use the drop-down menu, and select New repository.
- Type a short, memorable name for your repository. ...
- Optionally, add a description of your repository. ...
- Choose a repository visibility. ...
- Select Initialize this repository with a README.
- Click Create repository.

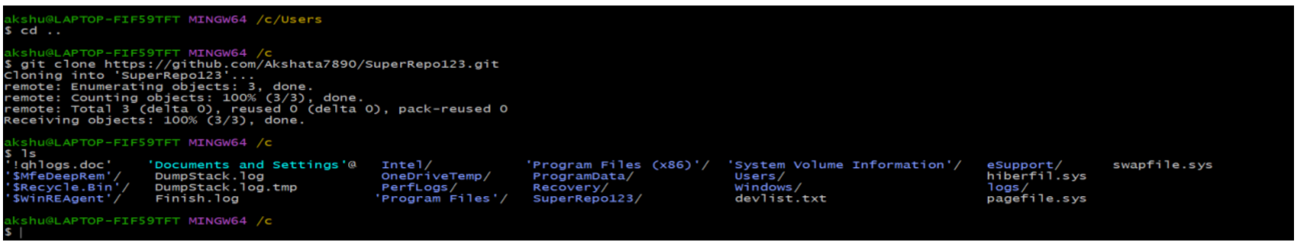
## 3. 2.Cloning a Repository

Just copy the link of the git repository and run the command

After cloning the git repository, the repository will save in your system directly .

Command : git clone 'https://github.com'




## 3. Making and recording changes

Open file and make some changes using following command.

Vim filename

```
akshu@LAPTOP-FIF59TFT MINGW64 /c
$ cd SuperRepo123

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ ls
htmlfile.txt

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ vim htmlfile.txt
```

```
<html>
<head>MY FIRST REPOSITORY and 2nd also</head>
</html>
~
~
```

Save the file using – esc + :

## 4. 4.Staging and committing changes

Use git add command for staging and git commit  for committing changes.

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git add htmlfile.txt
```

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git commit -m "hello"
[main 52e6555] hello
 1 file changed, 2 insertions(+), 2 deletions(-)
```

## 5. 5.Viewing the history of all the changes

To view the content of the file, use cat command.

Command : cat filename

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ cat htmlfile.txt
<html>
<head>MY FIRST REPOSITORY and 2nd also</head>
</html>
```

## 6. Undoing  changes

Make some changes in your file and save it.

```
<html>
<head>MY FIRST REPOSITORY and 2nd also jfieiyfhdlieiheihydfleyielifoleyfhhhhgkudne</head>
</html>
~
~
```

For undoing changes use the following command.

Command : git restore filename

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git restore htmlfile.txt

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ cat htmlfile.txt
<html>
<head>MY FIRST REPOSITORY and 2nd also</head>
</html>
```

.

## Git Branching and merging

### 1.Creating and switching to new branches

To create a new branch use the following code:

Command : git branch branchname

To switch to other branch use the following command.

Command : git checkout branchname

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git branch Super

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git checkout Super
Switched to branch 'Super'

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (Super)
$
```

### 2. Merging local branches together

To merge the local branches , use the following code :

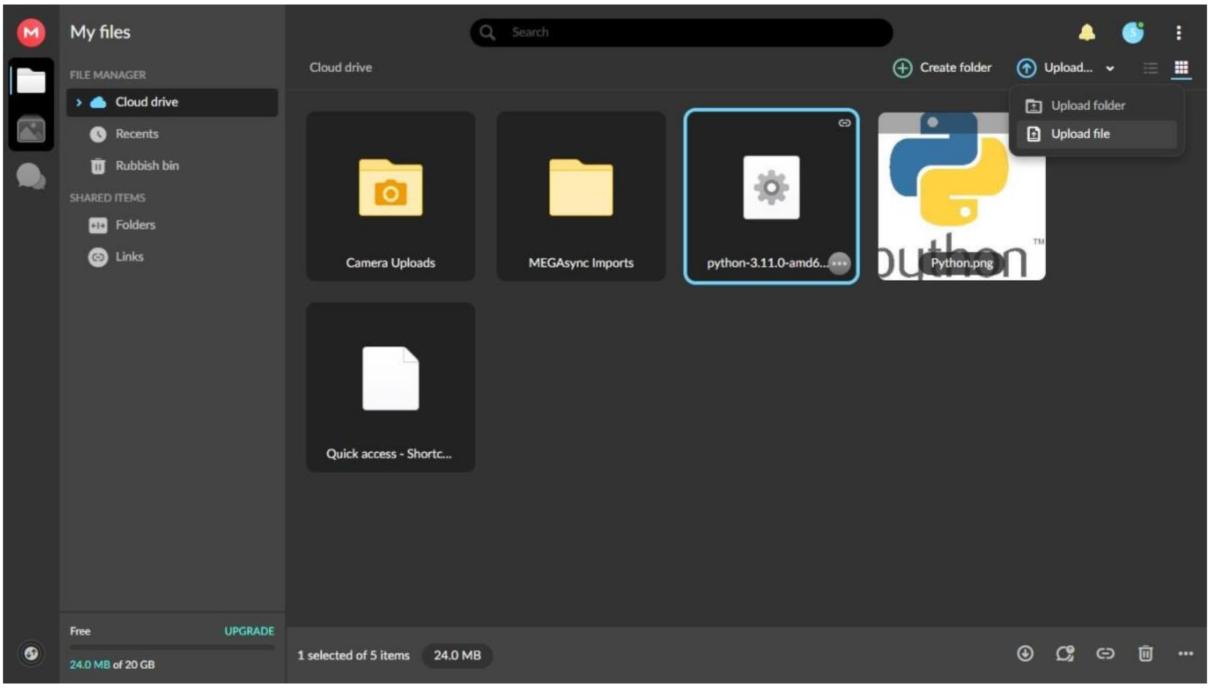
Command : git merge branchname

```
akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git merge Super
Already up to date.

akshu@LAPTOP-FIF59TFT MINGW64 /c/SuperRepo123 (main)
$ git log --oneline --decorate
52e6555 (HEAD -> main, origin/main, origin/HEAD, Super) hello
78e7d26 Add files via upload
```

# Week 2

## Deploy a simple application on the cloud / Deploy one simple web app on a web server using a cloud platform

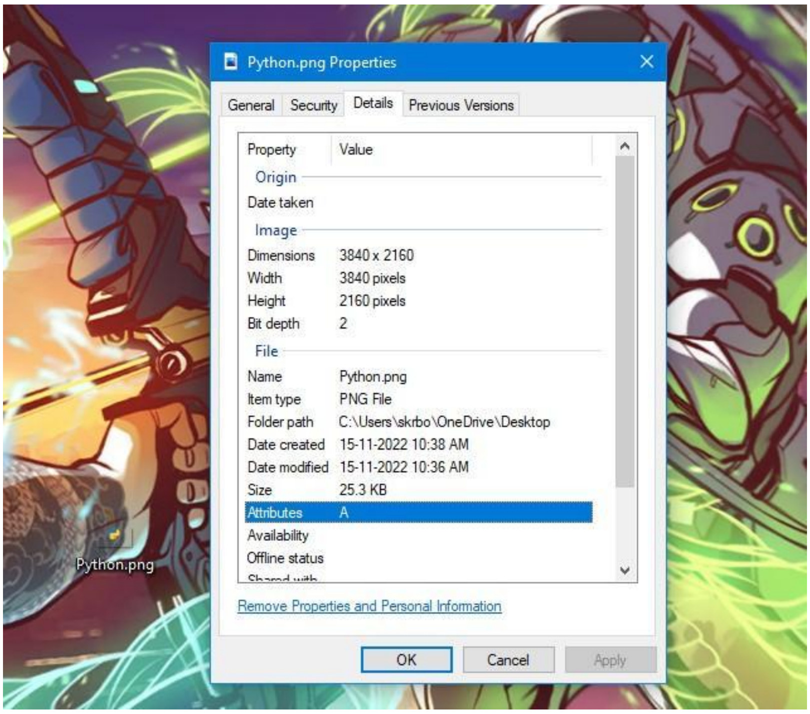
- Download any .exe file(software) from the following website **Python**
- Create a **Mega** account
- Upload the Downloaded .exe file to the cloud



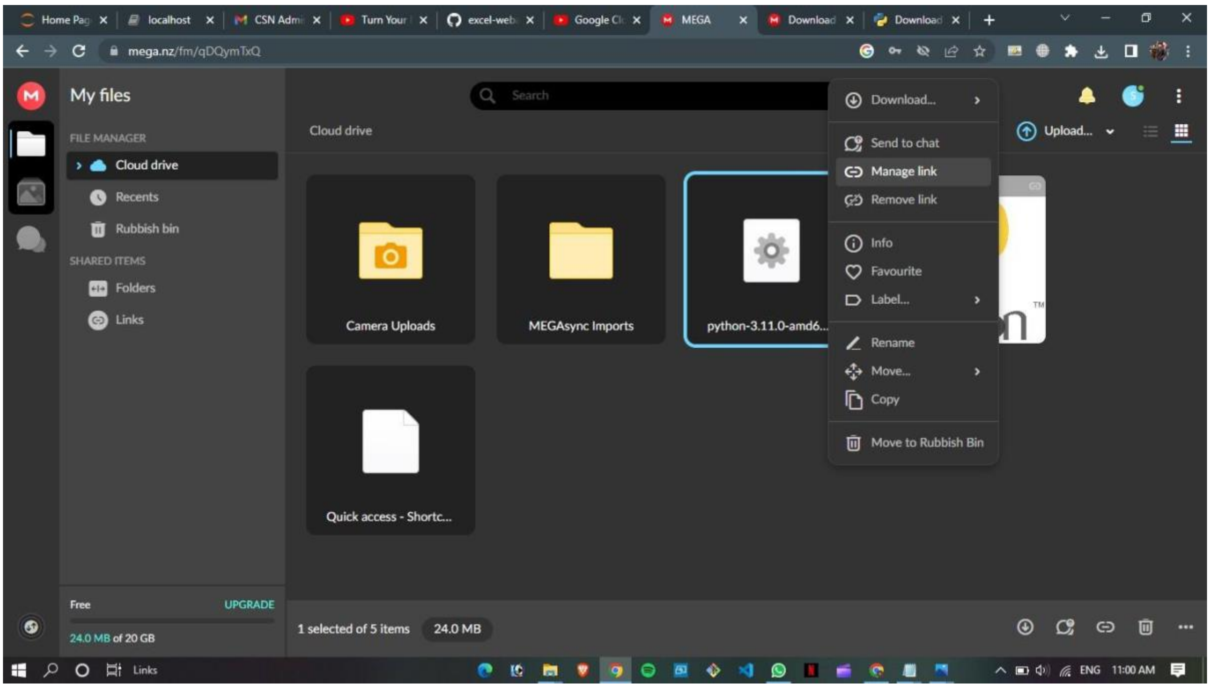

- **Design a simple web page using HTML**

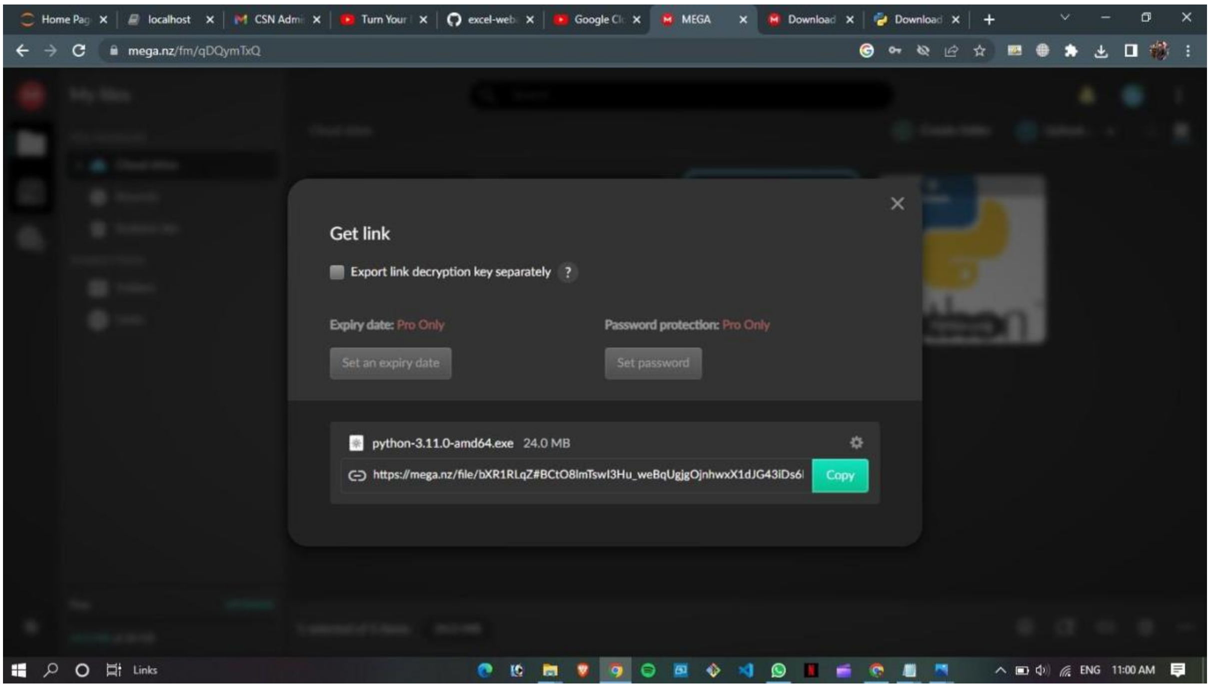
```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Image as link</title>
  </head>
  <body>
    Python(3.11.0) download:<br>
    <a
href="https://mega.nz/file/bXR1RLqZ#BCtO8lmTswl3Hu_weBqUgjgOjnhwxX1dJG43
i Ds6HYU">
      <img src="C:\Users\skrbo\OneDrive\Desktop\Python.png"
      width="800" height="500">
    </a>
  </body>
</html>
```

- Download a [python ](python) image and locate the Folder path and copy it to img src in the HTML code.



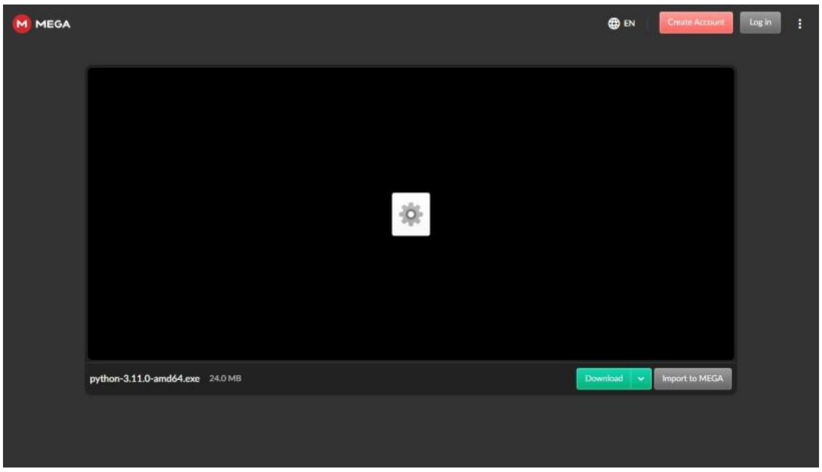

- Now change the href link in your code using the link of the .exe file that you have uploaded in the cloud






- **Now save the file in .html extension and open it using any web browser.**


- Click on the python image and it will redirect the page to cloud storage where you have uploaded your .exe file.




## DATABASE CONNECTIVITY :

Program :

```
import sqlite3

conn = sqlite3.connect('tiger.db')

print ("Opened database successfully")

conn.execute('''CREATE TABLE COMPANY6821
        (ID INT PRIMARY KEY     NOT NULL,
        NAME           TEXT    NOT NULL,
        AGE            INT     NOT NULL,
        ADDRESS        CHAR(50),
        SALARY         REAL);''')
print ("Table created successfully")

#insert operation
conn.execute("INSERT INTO COMPANY6821 (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (1, 'Akshata', 18, 'Shrinivas Nagar', 20000.00 )");

conn.execute("INSERT INTO COMPANY6821 (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (17, 'Hema', 25, 'Rich-Mond ', 65000.00 )");
conn.commit()
print ("Records created successfully")

#Select Operation

cursor = conn.execute("SELECT id, name, address, salary from COMPANY6821")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")
```

```
print ("Operation done successfully")
conn.close()
```

Output :

```
Opened database successfully
Table created successfully
Records created successfully
ID =  1
NAME =  Akshata
ADDRESS =  Shrinivas Nagar
SALARY =  20000.0

ID =  17
NAME =  Hema
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

# WEEK 03 :

## Explore NumPy Modules

### 1. Array Aggregation functions

Program :

```
import numpy as np
a=np.array([1,2,3,4,5])
print("a :",a)
sum=np.sum(a)
print("sum :",sum)
product=np.prod(a)
print("product :",product)
mean=np.mean(a)
print("mean :",mean)
standard_deviation=np.std(a)
print("standard_deviation :",standard_deviation)
variance=np.var(a)
print("variance :",variance)
minimum=np.min(a)
print("minimum value :",minimum)
maximum=np.max(a)
print("maximum value :",maximum)
minimum_index=np.argmin(a)
print("minimum index :",minimum_index)
maximum_index=np.argmax(a)
print("maximum-index :",maximum_index)
median=np.median(a)
print("median :",median)
```

Output :

```
a : [1 2 3 4 5]
sum : 15
product : 120
mean : 3.0
standard_deviation : 1.4142135623730951
variance : 2.0
minimum value : 1
maximum value : 5
minimum index : 0
maximum-index : 4
median : 3.0
```

## 2.Vectorized Operations

Program :

```
# importing the modules
import numpy as np
import timeit

# vectorized sum
print(np.sum(np.arange(15)))

print("Time taken by vectorized sum : ", end = "")
%timeit np.sum(np.arange(15))

# iterative sum
total = 0
for item in range(0, 15):
   total += item
a = total
print("\n" + str(a))

print("Time taken by iterative sum : ", end = "")
%timeit a
```

Output :

```
105
Time taken by vectorized sum : 3.8 µs ± 137 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops ea
ch)

105
Time taken by iterative sum : 18.2 ns ± 0.677 ns per loop (mean ± std. dev. of 7 runs, 100,000,000 lo
ops each)
```

## 3. Use Map, Filter, Reduce and Lambda Functions with NumPy

Program :

```
import numpy as np
from functools import reduce

np.num = [2, 3, 6, 9, 10]
np.num1= [[1, 3, 5], [7, 9], [11, 13, 15]]

cubed = list(map(lambda cube: (cube ** 3),np.num))
print("Using map function", cubed)

even = list(filter(lambda x: (x % 2 == 0), np.num))
print("Using filter function",even)

re = reduce(lambda x, y: x + y, np.num)
print("Using Reduce function " ,re)
```

Output :

```
Using map function [8, 27, 216, 729, 1000]
Using filter function [2, 6, 10]
Using Reduce function  30
```

# Explore Pandas Module :

### 1. Aggregation and Grouping

Program :

```
# import module
import pandas as pd

# Creating our dataset
df = pd.DataFrame([[9, 4, 8, 9],
                   [8, 10, 7, 6],
                   [7, 6, 8, 5]],
                   columns=['Maths', 'English',
                            'Science', 'History'])

# display dataset
print(df)
```

df.agg(['sum', 'min', 'max','mean','median','std','count','size',])

Output :

```
   Maths  English  Science  History
0      9        4        8        9
1      8       10        7        6
2      7        6        8        5
```

| | Maths | English | Science | History |
|---|---|---|---|---|
| **sum** | 24.0 | 20.000000 | 23.000000 | 20.000000 |
| **min** | 7.0 | 4.000000 | 7.000000 | 5.000000 |
| **max** | 9.0 | 10.000000 | 8.000000 | 9.000000 |
| **mean** | 8.0 | 6.666667 | 7.666667 | 6.666667 |
| **median** | 8.0 | 6.000000 | 8.000000 | 6.000000 |
| **std** | 1.0 | 3.055050 | 0.577350 | 2.081666 |
| **count** | 3.0 | 3.000000 | 3.000000 | 3.000000 |
| **size** | 3.0 | 3.000000 | 3.000000 | 3.000000 |

## 2.Time Series Operations

Program :

```
import pandas as pd
import numpy as np
df=pd.DataFrame({"Date":pd.date_range(start="2022-11-01",periods=21,
freq="D"),"temp":np.random.randint(18, 30, size=21)})
print(df.head()
df['temp_lag_1']=df['temp'].shift(1)
print("\nShift Function\n",df.head())
df_weekly =df.resample("w", on="Date").mean()
print("\nResample Function\n",df_weekly.head())
```

Output :

```
        Date  temp
0 2022-11-01    20
1 2022-11-02    18
2 2022-11-03    20
3 2022-11-04    28
4 2022-11-05    23

Shift Function
        Date  temp  temp_lag_1
0 2022-11-01    20         NaN
1 2022-11-02    18        20.0
2 2022-11-03    20        18.0
3 2022-11-04    28        20.0
4 2022-11-05    23        28.0

Resample Function
                 temp  temp_lag_1
Date
2022-11-06  21.833333   21.800000
2022-11-13  21.571429   22.142857
2022-11-20  24.285714   23.714286
2022-11-27  18.000000   22.000000
```

## 3.Pivot and melt function

Program :

```
import pandas as pd

d1 = {"Name": ["patil", "keerthi", "yukthi"], "ID": [1, 2, 3], "Role": ["CEO", "Editor",
"Author"]}
df = pd.DataFrame(d1)
print(df)
print("\n")
df_melted = pd.melt(df)
print(df_melted)
df.pivot(columns='ID')
```

Output :

```
      Name  ID    Role
0    patil   1     CEO
1  keerthi   2  Editor
2   yukthi   3  Author


  variable    value
0     Name    patil
1     Name  keerthi
2     Name   yukthi
3       ID        1
4       ID        2
5       ID        3
6     Role      CEO
7     Role   Editor
8     Role   Author
```

| | Name | | | Role | | |
|---|---|---|---|---|---|---|
| ID | 1 | 2 | 3 | 1 | 2 | 3 |
| 0 | patil | NaN | NaN | CEO | NaN | NaN |
| 1 | NaN | keerthi | NaN | NaN | Editor | NaN |
| 2 | NaN | NaN | yukthi | NaN | NaN | Author |

# 4. Use Map, Filter, Reduce and Lambda Functions with Pandas dataframes

Program :

```
import pandas as pd
from operator import add
from functools import reduce

Coding= {'subject' :['python','java'],  'amount':[1000,2000]}
df = pd.DataFrame(Coding)
print(df)

print("\n  map operation to multiply amount by 2\n")
df['amount'] = df['amount'].map(lambda x: x*2)
print(df)
print("\n")
```

```
print("operation of filter to display only subject column\n")
df2=df.filter(items=['subject'])
print(df2)

print("reduce operation to find total amount\n")

reduce(add,df.amount)
```

Output :

```
  subject  amount
0  python    1000
1    java    2000

  map operation to multiply amount by 2

  subject  amount
0  python    2000
1    java    4000


operation of filter to display only subject column

  subject
0  python
1    java
reduce operation to find total amount

6000
```

## DATA VISUALIZATION WITH PYTHON :

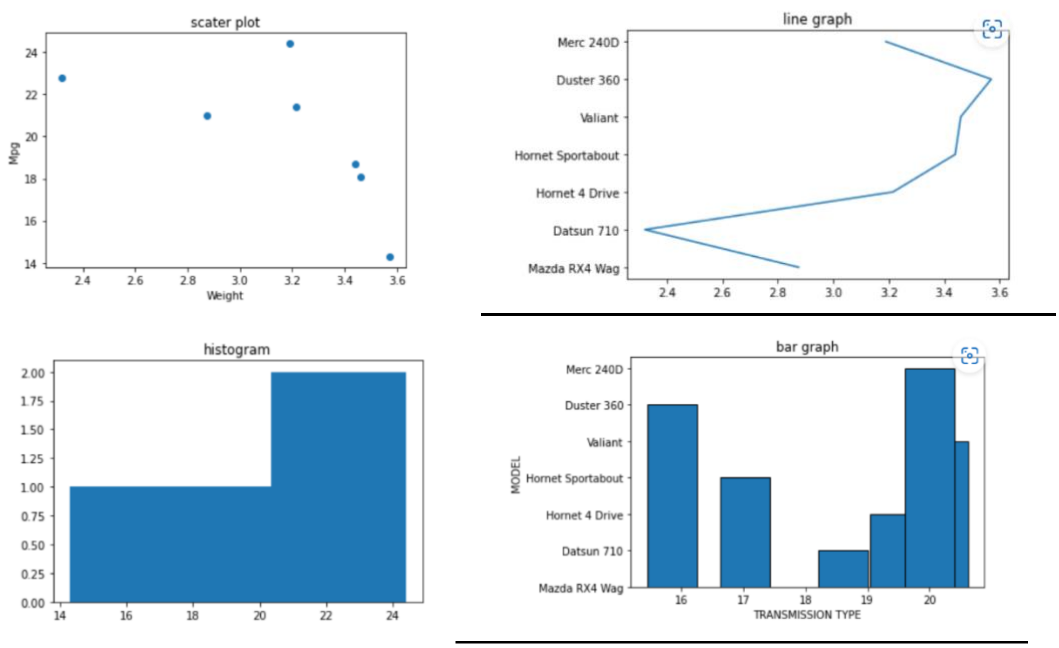
Program :

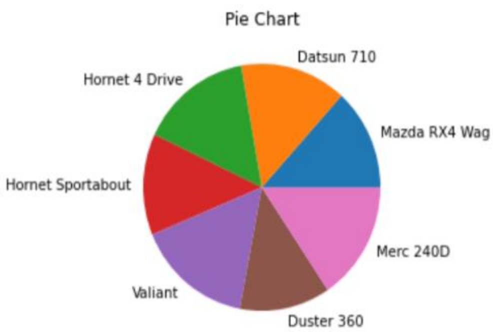
```
import matplotlib
from matplotlib import pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import pandas as pd
import seaborn as sns

df = pd.read_csv('Cars2.csv')
# customizing runtime configuration stored
# in matplotlib.rcParams
#plt.rcParams["figure.figsize"] = [7.00, 3.50]
```

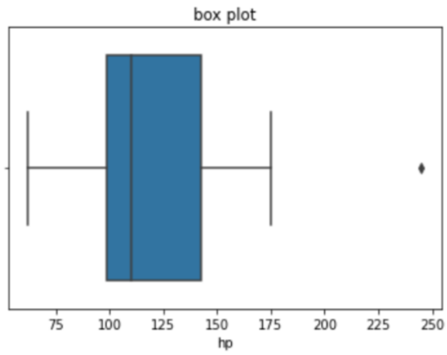
```
#plt.rcParams["figure.autolayout"] = True
fig1 = plt.figure()
x=df.wt
y=df.mpg
plt.xlabel('Weight')
plt.ylabel('Mpg')
plt.title('scater plot')
plt.scatter(x,y)

fig2 = plt.figure()
x = df.wt
y = df.model
plt.title('line graph')
plt.plot(x,y)

fig3 = plt.figure()
y= df.mpg
plt.title('histogram')
plt.hist(y,   bins = 5 ,);

fig4 = plt.figure()
x=df.trans
y=df.model
plt.xlabel('TRANSMISSION TYPE')
plt.ylabel('MODEL')
plt.title('bar graph')
plt.bar(x,y,edgecolor='black' )

fig5 = plt.figure()
plt.pie(a, labels=b)
plt.title("Pie Chart")
cols = ['b','c','g', 'orange']

fig6 = plt.figure()
plt.title('box plot')
sns.boxplot(df['hp'])

def save_image(filename):

   # PdfPages is a wrapper around pdf
   # file so there is no clash and create
   # files with no error.
   p = PdfPages(filename)

   # get_fignums Return list of existing
   # figure numbers
   fig_nums = plt.get_fignums()
   figs = [plt.figure(n) for n in fig_nums]
```

```
    # iterating over the numbers in list
    for fig in figs:

        # and saving the files
        fig.savefig(p, format='pdf')

     # close the object
     p.close()

# name your Pdf file
filename = "multi_plot_image1.pdf"

# call the function
save_image(filename)
```

<u>Output :</u>




Pie Chart
Datsun 710
Hornet 4 Drive
Mazda RX4 Wag
Hornet Sportabout
Merc 240D
Valiant
Duster 360


box plot
75
100
125
150
175
200
225
250
hp

## WEEK 04 :

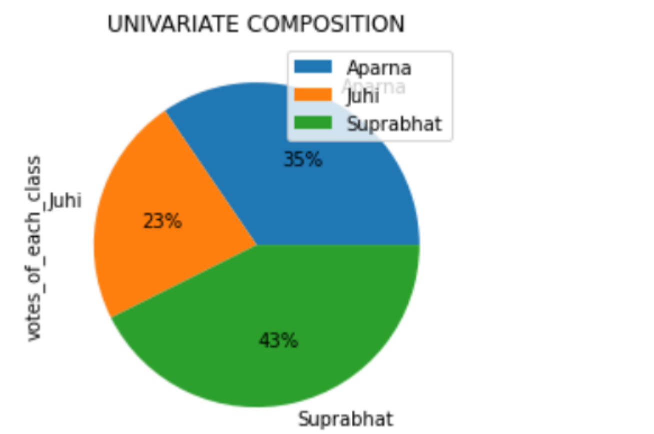
## UNIVARIATE DATA ANALYSIS

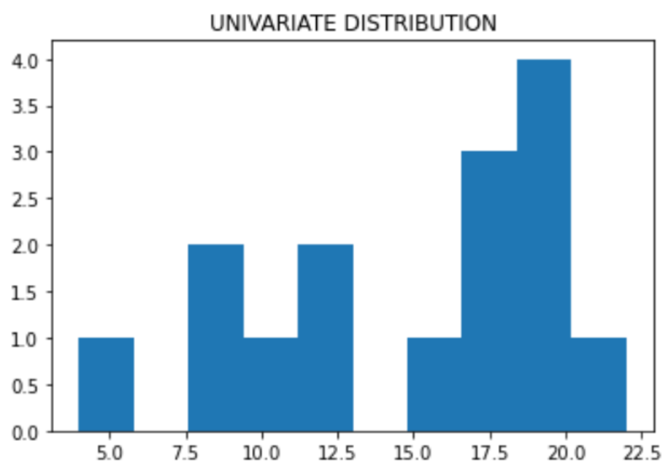
Program :

```
import pandas as pd
import matplotlib.pyplot as plt

# DataFrame of each student and the votes they get
dataframe = pd.DataFrame({'Name': ['Aparna', 'Aparna', 'Aparna',
                                                                  'Aparna',
'Aparna', 'Juhi',
                    'Juhi', 'Juhi', 'Juhi', 'Juhi',
                    'Suprabhat', 'Suprabhat',
                    'Suprabhat', 'Suprabhat',
                    'Suprabhat'],
                    'votes_of_each_class': [12, 9, 17, 19,
                              20, 11, 15, 12,
                              9, 4, 22, 19, 17,
                              19, 18]})



# Plotting the pie chart for above dataframe
dataframe.groupby(['Name']).sum().plot(
         kind='pie', y='votes_of_each_class',
autopct='%1.0f%%')
plt.title("UNIVARIATE COMPOSITION")

plt.show()
plt.title("UNIVARIATE DISTRIBUTION")
plt.hist(dataframe['votes_of_each_class'])

plt.show()
print("Mean:",dataframe['votes_of_each_class'].mean())
```

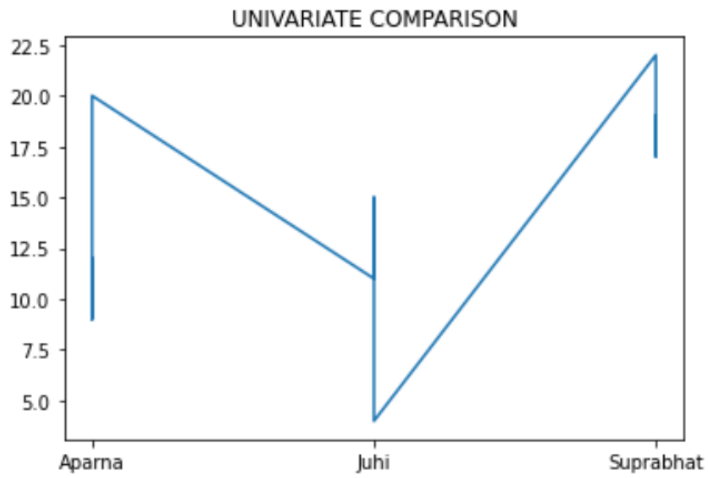
```
print("median:",dataframe['votes_of_each_class'].median())
print("Standrad deviation:",dataframe['votes_of_each_class'].std())

x=dataframe.Name
y=dataframe.votes_of_each_class
plt.title("UNIVARIATE COMPARISON")
plt.plot(x,y)
```

Output :




```
Mean: 14.866666666666667
median: 17.0
Standrad deviation: 5.111145617550876

[<matplotlib.lines.Line2D at 0x11d4b86d070>]
```




## MULTIVARIATE ANALYSIS :

COVARIANCE :

Program :

```
import pandas as pd
X = pd.Series([1,3,5,10,20])
Y = pd.Series([2,4,1,-2,12])
print("The covariance value: ", X.cov(Y))
```

Output :

```
The covariance value: 26.599999999999998
```

CORRELATION :

Program :

```
import pandas as pd
data = pd.read_csv('iris.csv')
data
data.corr()

import pandas as pd
data = pd.DataFrame(np.random.randint(0, 10, size=(5, 3)), columns=['A',
'B', 'C'])
 data
 data.corr()
```

Output :

## Correlation

In [19]:

```
import pandas as pd

data = pd.read_csv('iris.csv')
data
data.corr()
```

Out[19]:

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| **sepal_length** | 1.000000 | 0.626300 | -0.184115 | NaN |
| **sepal_width** | 0.626300 | 1.000000 | -0.188982 | NaN |
| **petal_length** | -0.184115 | -0.188982 | 1.000000 | NaN |
| **petal_width** | NaN | NaN | NaN | NaN |

In [21]:

```
import pandas as pd

data = pd.DataFrame(np.random.randint(0, 10, size=(5, 3)), columns=['A', 'B', 'C'])
data
data.corr()
```

Out[21]:

| | A | B | C |
|---|---|---|---|
| **A** | 1.000000 | -0.339624 | 0.105511 |
| **B** | -0.339624 | 1.000000 | -0.347548 |
| **C** | 0.105511 | -0.347548 | 1.000000 |

## MULTIVARIATE PLOTS

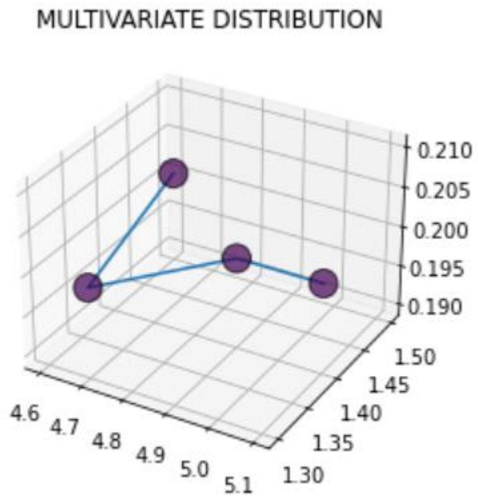
Program :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import plotly.express as px
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

df=pd.read_csv('iris.csv')
df
```

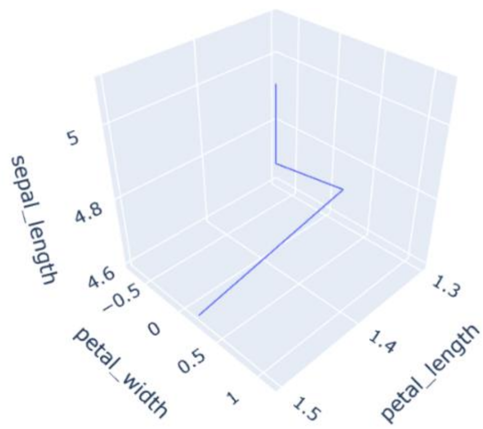
| | sepal_length | sepal_width | petal_length | petal_width | variety |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Setosa |

```
fig=plt.figure()
```

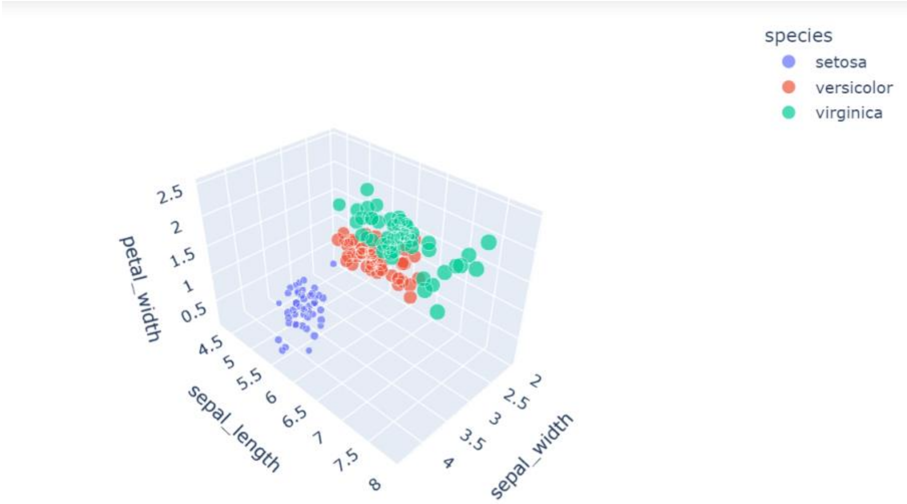
```
x=df.sepal_length
y=df.petal_length
z=df.petal_width
ax=fig.add_subplot(111,projection='3d')
plt.title("MULTIVARIATE DISTRIBUTION")
ax.scatter(x,y,z,linewidth=1,alpha=.7,edgecolor='k',s=200,c=z)
ax.plot(x,y,z)
```



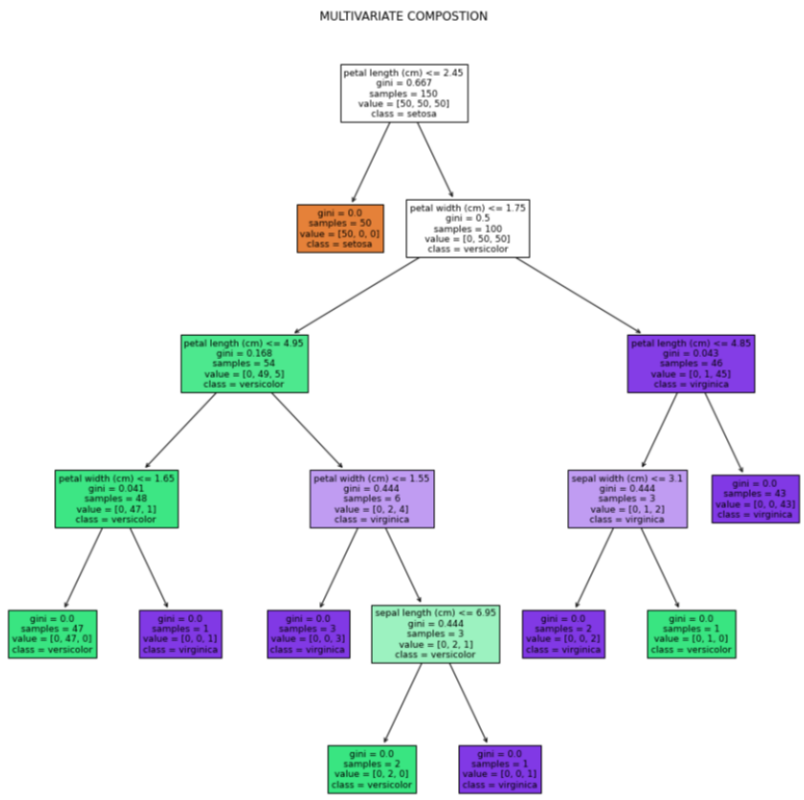

```
print("Multivariate Comparision")
fig= plt.figure()
ax = px.line_3d(df, x="petal_length", y="petal_width", z="sepal_length")
ax.show()
```




```
print("Multivariate Relationship")
df=px.data.iris()
fig=px.scatter_3d(df,x='sepal_width',
          y='sepal_length',
          z='petal_width',
          size='petal_length',
          color='species')
fig.show()
```




```
iris = datasets.load_iris()
X = iris.data
y = iris.target
clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
text_representation = tree.export_text(clf)
print(text_representation)
with open("decistion_tree.log", "w") as fout:
   fout.write(text_representation)
fig = plt.figure(figsize=(15,15))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
plt.title("MULTIVARIATE COMPOSTION")
```




**Eigenvalues** are the special set of scalar values that is associated with the set of linear equations most probably in the matrix equations. The eigenvectors are also termed as characteristic roots. It is a non-zero vector that can be changed at most by its scalar factor after the application of linear transformations.

**Eigenvectors** are a special set of vectors associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic vectors, proper vectors, or latent vectors

**Eigendecomposition** provides us with a tool to decompose a matrix by discovering the eigenvalues and the eigenvectors. This operation can prove useful since it allows certain matrix operations to be easier to perform and it also tells us important facts about the matrix itself.

## LINEAR ALGEBRA USING PYTHON :

**Program :**

```
import numpy as np
from numpy import array
#Scalar has only one unit
number_of_units=50
print("Scalar : ",number_of_units)
#Vector has only 1 row and many columns or 1 column and many rows
A= np.array([1,2,3,4]) #vector1
B = np.array([-4,-3, -2, -1]) #vector2
print("\nVector :\n",A)
```

```
print(B)
#Matrix has N number of columns and rows
C= np.array([[2,-2],[3,1],[1,4]])
print("\nMatrix A:\n",C)
# Finding transpose of a matrix using the function transpose()
print("Matrix A Transpose: \n",np.transpose(C))
#Tensor has n number of Matrix
T = array([
[[1,2,3], [4,5,6], [7,8,9]],
[[11,12,13], [14,15,16], [17,18,19]],
[[21,22,23], [24,25,26], [27,28,29]],
])
print(T.shape)
print("\nTensor:\n",T)
n=np.gradient(C)
print("\nGradient : \n",n)
w, v = np.linalg.eig(T)
mat_norm = np.linalg.norm(T)
print("\nEigen values:\n",w)
print("\nEigen vectors:\n",v)
print("\nMatrix norm:", mat_norm)
```

**Output :**

```
Scalar :  50

Vector :
 [1 2 3 4]
[-4 -3 -2 -1]

Matrix A:
 [[ 2 -2]
 [ 3  1]
 [ 1  4]]
Matrix A Transpose:
 [[ 2  3  1]
 [-2  1  4]]
(3, 3, 3)

Tensor:
 [[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[11 12 13]
  [14 15 16]
  [17 18 19]]

 [[21 22 23]
  [24 25 26]
  [27 28 29]]]

Gradient :
 [array([[ 1. ,  3. ],
       [-0.5,  3. ],
       [-2. ,  3. ]]), array([[-4., -4.],
       [-2., -2.],
       [ 3.,  3.]])]

Eigen values:
 [[ 1.61168440e+01 -1.11684397e+00 -3.38433605e-16]
 [ 4.53965063e+01 -3.96506284e-01  4.43774968e-16]
 [ 7.52392369e+01 -2.39236876e-01  1.00586167e-15]]

Eigen vectors:
 [[[-0.23197069 -0.78583024  0.40824829]
  [-0.52532209 -0.08675134 -0.81649658]
  [-0.8186735   0.61232756  0.40824829]]

 [[-0.45694089 -0.7371869   0.40824829]
  [-0.56993163 -0.03110723 -0.81649658]
  [-0.68292237  0.67497245  0.40824829]]

 [[-0.50588144 -0.72551862  0.40824829]
  [-0.57461613 -0.01878627 -0.81649658]
  [-0.64335081  0.68794608  0.40824829]]]

Matrix norm: 89.74965180990955
```

# WEEK 05 :
## Data Cleaning

### DETECT MISSING VALUES WITH PANDAS DATAFRAME FUNCTIONS: .INFO() AND .ISNA() :

Program :

```
import pandas as pd
df = pd.read_csv('sd.csv')
df.info()
```

Output :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Experience  25 non-null     int64
 1   salary      22 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

Program :

```
import pandas as pd
df = pd.read_csv("Book1.csv")
df
```

:

| | name | roll | marks | grade |
|---|---|---|---|---|
| 0 | xyz | 1 | Nan | A |
| 1 | yxz | 2 | 89 | A |
| 2 | xzy | 3 | Nan | Nan |
| 3 | klm | 4 | 88 | Nan |

```
# detect the missing values
df.isna()
```

| | name | roll | marks | grade |
|---|---|---|---|---|
| 0 | False | False | True | False |
| 1 | False | False | False | False |
| 2 | False | False | True | True |
| 3 | False | False | False | True |

### ESTIMATE AND IMPUTE MISSING VALUES :

Program :

```
#Load libraries
import os
```

```
import pandas as pd
import numpy as np

sal = pd.read_csv("sd.csv")
print("count of NULL values before imputation\n")
print(sal.isnull().sum())
sal.head()
sal.describe()

missing_col = ['salary']

#Technique 2: Using median to impute the missing values
for i in missing_col:
  sal.loc[sal.loc[:,i].isnull(),i]=sal.loc[:,i].mean()


print("count of NULL values after imputation\n")
print(sal.isnull().sum())
sal
```

Output :

```
count of NULL values before imputation

 Experience    0
salary        3
dtype: int64
count of NULL values after imputation

Experience    0
salary        0
dtype: int64
```

|  | Experience | salary |
|---|---|---|
| 0 | 5 | 40000.000000 |
| 1 | 7 | 50000.000000 |
| 2 | 9 | 60000.000000 |
| 3 | 11 | 70000.000000 |
| 4 | 13 | 161363.636364 |
| 5 | 15 | 90000.000000 |
| 6 | 17 | 100000.000000 |
| 7 | 19 | 110000.000000 |
| 8 | 21 | 120000.000000 |
| 9 | 23 | 130000.000000 |
| 10 | 25 | 161363.636364 |
| 11 | 27 | 150000.000000 |
| 12 | 29 | 160000.000000 |
| 13 | 31 | 170000.000000 |
| 14 | 33 | 180000.000000 |
| 15 | 35 | 190000.000000 |
| 16 | 37 | 200000.000000 |
| 17 | 39 | 210000.000000 |
| 18 | 41 | 220000.000000 |
| 19 | 43 | 161363.636364 |
| 20 | 45 | 240000.000000 |
| 21 | 47 | 250000.000000 |
| 22 | 49 | 260000.000000 |
| 23 | 51 | 270000.000000 |
| 24 | 53 | 280000.000000 |

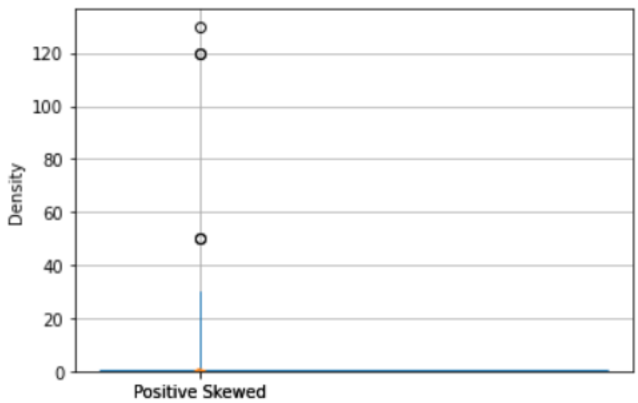
## PERFORM A LOG TRANSFORMATION :

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns

#create a list  of data
data=[1,1,10,10,15,15,20,20,30,50,120,130,120,50,30,30,25,20,20,15,15,13,
11,9,7,6,6,5,5,5,4,4,4,4,3,3,3,3,2,2,2,2,2,2,2,2,1,
    1,1]

#create the pandas Dataframe
df=pd.DataFrame(data,columns=["Positive Skewed"])

df
#Boxplot showing three outliers
df.boxplot(column='Positive Skewed')

#Right skewed data
sns.distplot(df)
```

```
#Creating inpiut data from dataframe on variable positive Skewness with
input values ranging from 1 to 130
inp_array=df
print("Input array:",inp_array)

#Applying log10 transformation with output values ranging from 0 to 24
out_array = np.log10(inp_array)
print("Output array :",out_array)

#Box plot showing No outliers with all them treated by doing Log10
transformation
out_array.boxplot(column='Positive Skewed')

#Right Skewed data transformed to Fairly or else close to normal
distribution using transformations
sns.distplot(out_array)
#if wants to revert back Log10 values to original value for interpretation
purpose then just raise 10 to the power
#Log10 values as shown as below.
original_val = (10**out_array)
print('Original Values : ',original_val)
```

Output :



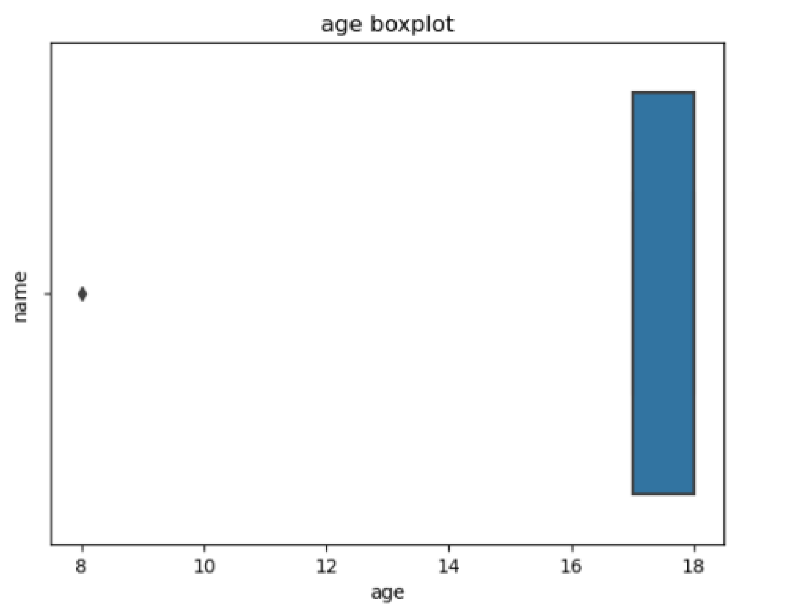

## UNIVARIATE OUTLIER DETECTION :

Program:

```
import matplotlib
```

```
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd

df=pd.read_csv("height.csv")
ax = sns.boxplot(x = df["age"])
ax.set_xlabel("age")
ax.set_ylabel("name")
ax.set_title("age boxplot")
plt.show()

#extract the upper and lower quantiles
height_lq = df["age"].quantile(0.25)
height_uq = df["age"].quantile(0.75)
#extract the inter quartile range
height_iqr = height_uq -height_lq
#get the upper and lower bounds
lower_bound = height_lq - 1.5*height_iqr
upper_bound = height_uq + 1.5*height_iqr
#extract values outside these bounds
Height_outliers = df[(df.age <= lower_bound) | (df.age >= upper_bound)]
Height_outliers
```

Output :




Out[29]:

| | name | age | height |
|---|---|---|---|
| 3 | harshith | 8 | 170 |

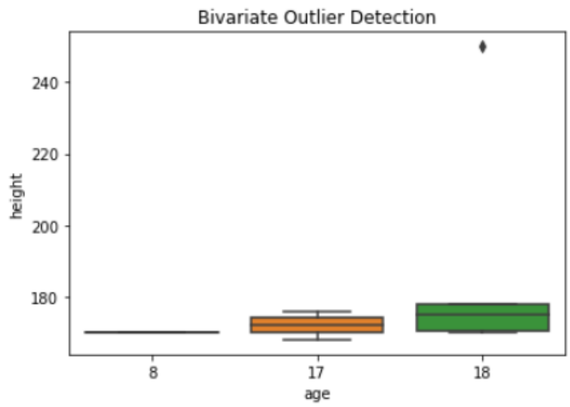
**BIVARIATE OUTLIER DETECTION :**

Program :

```
import matplotlib
```

```
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
df=pd.read_csv("height.csv")
dff=sns.boxplot(x=df['age'], y=df['height'])
dff.set_title('Bivariate Outlier Detection')
plt.show()
```

Output :

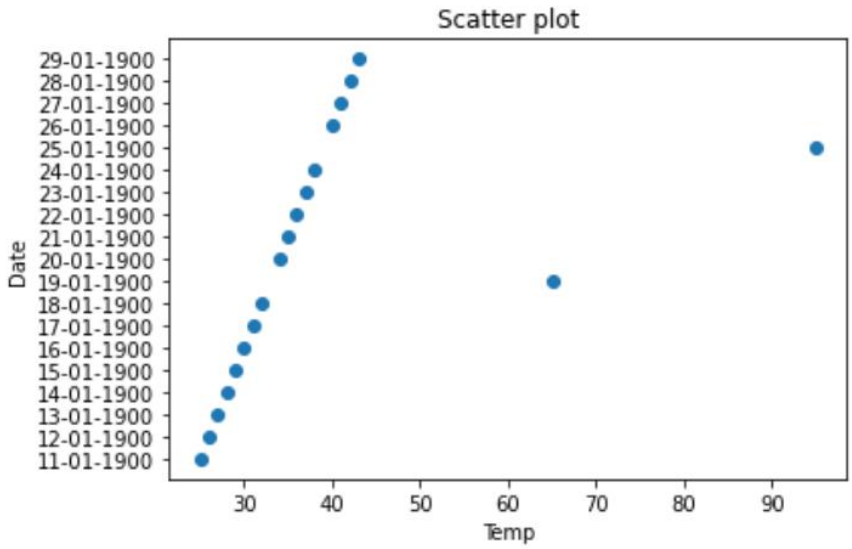



## TIME SERIES OUTLIER DETECTION :

Program :

```
import numpy as np
import pandas as pd

df=pd.read_csv("datetemp2.csv")
df

import matplotlib.pyplot as plt
x = df.temp
y = df.date
plt.scatter(x,y,label="values of x & y")
plt.xlabel('Temp')
plt.ylabel('Date')
plt.title('Scatter plot')
plt.show()
```

Output :




## DATA INTEGRATION

### APPROACHES - ADDING ATTRIBUTES

Program :

```
import pandas as pd
df1=pd.read_csv("stu1.csv")
df2=pd.read_csv("stu2.csv")
df1.head()
df2.head()
df1['Name']='abc','xyz','pqr','klm'
df1.head()
df=pd.merge(df1,df2, on='student_id')
df.head()
```

Output:

In [16]:
```
import pandas as pd
df1 = pd.read_csv("stu1.csv")

df2 = pd.read_csv("stu2.csv")
```

In [17]:
```
df1.head()
```

Out[17]:

| | student_id | marks | city |
|---|---|---|---|
| 0 | 1 | 81 | bangalore |
| 1 | 2 | 87 | mumbai |
| 2 | 3 | 99 | delhi |
| 3 | 4 | 89 | pune |

In [18]:
```
df2.head()
```

Out[18]:

| | student_id | age | gender | grade | employed |
|---|---|---|---|---|---|
| 0 | 1 | 18 | male | 1st | yes |
| 1 | 2 | 19 | female | 2nd | no |
| 2 | 3 | 19 | male | 1st | no |
| 3 | 4 | 20 | male | 2nd | no |
| 4 | 5 | 22 | female | 1st | yes |

In [21]:
```
df1['name']='abc','def','sfs','saf'
df1.head()
```

Out[21]:

| | student_id | marks | city | name |
|---|---|---|---|---|
| 0 | 1 | 81 | bangalore | abc |
| 1 | 2 | 87 | mumbai | def |
| 2 | 3 | 99 | delhi | sfs |
| 3 | 4 | 89 | pune | saf |

In [20]:
```
df = pd.merge(df1, df2, on = 'student_id')
df.head()
```

Out[20]:

| | student_id | marks | city | age | gender | grade | employed |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 81 | bangalore | 18 | male | 1st | yes |
| 1 | 2 | 87 | mumbai | 19 | female | 2nd | no |
| 2 | 3 | 99 | delhi | 19 | male | 1st | no |
| 3 | 4 | 89 | pune | 20 | male | 2nd | no |

**APPROACHES ADDING DATA OBJECTS :**

Program :

```
import pandas as pd
from numpy.random import randint
mark = {'Name':['Harish', 'Rakshitha', 'Manohar', 'Vidya'], 'Maths':[87, 98, 87, 95], 'ITSkills':[83, 99, 84, 76] }
 df = pd.DataFrame(mark)
display(df)
```

| | Name | Maths | IT Skills |
|---|---|---|---|
| 0 | Harish | 87 | 83 |
| 1 | Rakshitha | 98 | 99 |
| 2 | Manohar | 87 | 84 |
| 3 | Vidya | 95 | 76 |

```
df.loc[len(df.index)] = ['Hruthvik', 89, 96]
 display(df)
```

|   | Name | Maths | IT Skills |
|---|---|---|---|
| 0 | Harish | 87 | 83 |
| 1 | Rakshitha | 98 | 99 |
| 2 | Manohar | 87 | 84 |
| 3 | Vidya | 95 | 76 |
| 4 | Hruthvik | 89 | 96 |

**NUMEROSITY DATA REDUCTION :**

Program :

Random sampling Example –

Random sampling to speed up tuning

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
customer_df = pd.read_csv('Customer Churn.csv') 1
print(customer_df.shape)
print(customer_df.Churn.value_counts())

customer_df_rs = customer_df.sample(1000,random_state=1)
y=customer_df_rs['Churn']
  Xs = customer_df_rs.drop(columns=['Churn'])
print(customer_df_rs.shape)

print(customer_df_rs.Churn.value_counts())
```

Output :

## Numerosity Data Reduction

### Random sampling

#### Example – Random sampling to speed up tuning

```
In [46]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         import seaborn as sns
```

```
In [47]: customer_df = pd.read_csv('Customer Churn.csv')
         print(customer_df.shape)
         print(customer_df.Churn.value_counts())

         (3150, 9)
         0    2655
         1     495
         Name: Churn, dtype: int64
```

```
In [48]: customer_df_rs = customer_df.sample(1000,random_state=1)
         y=customer_df_rs['Churn']
         Xs = customer_df_rs.drop(columns=['Churn'])
         print(customer_df_rs.shape)

         (1000, 9)
```

```
In [49]: print(customer_df_rs.Churn.value_counts())

         0    856
         1    144
         Name: Churn, dtype: int64
```

Stratified sampling Example –
Stratified sampling for imbalanced dataset

```
n,s=len(customer_df),1000
print(n,s)
r = s/n
print('Ratio of each Churn class in sample:',r)
sample_df=customer_df.groupby('Churn').apply(lambdasdf:sdf.sample(round(len(sdf)*r)))
 print(sample_df.Churn.value_counts())

customer_df.Churn.value_counts().plot.bar()

sample_df.Churn.value_counts().plot.bar()
```

Output :

### Stratified sampling

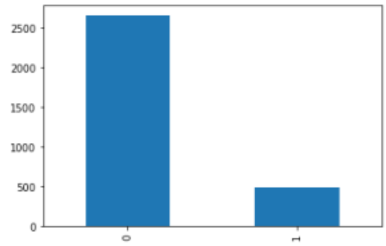
**Example – Stratified sampling for imbalanced dataset**

In [59]:
```
n,s=len(customer_df),1000
print(n,s)
r = s/n
print('Ratio of each Churn class in sample:',r)
sample_df = customer_df.groupby('Churn').apply(lambda sdf: sdf.sample(round(len(sdf)*r)))
print(sample_df.Churn.value_counts())
```

```
3150 1000
Ratio of each Churn class in sample: 0.31746031746031744
0    843
1    157
Name: Churn, dtype: int64
```

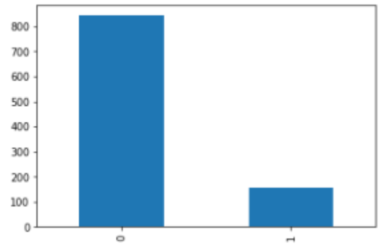
In [60]:
```
customer_df.Churn.value_counts().plot.bar()
```

Out[60]: <AxesSubplot:>




In [61]:
```
sample_df.Churn.value_counts().plot.bar()
```

Out[61]: <AxesSubplot:>




Random Over/Under-sampling

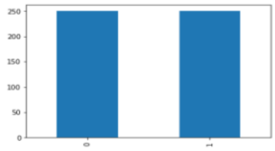
n,s=len(customer_df),500
sample_df = customer_df.groupby('Churn').apply(lambda sdf:
sdf.sample(250)) print(sample_df.Churn.value_counts())
sample_df.Churn.value_counts().plot.bar()
sample_df

Output :

### Random Over/Under-sampling

```
In [65]: n,s=len(customer_df),500
         sample_df = customer_df.groupby('Churn').apply(lambda sdf: sdf.sample(250))
         print(sample_df.Churn.value_counts())

0    250
1    250
Name: Churn, dtype: int64
```

```
In [66]: sample_df.Churn.value_counts().plot.bar()
Out[66]: <AxesSubplot:>
```




```
In [67]: sample_df
Out[67]:
```

| Churn | | Call Failure | Complains | Subscription Length | Seconds of Use | Frequency of use | Frequency of SMS | Distinct Called Numbers | Status | Churn |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1815 | 0 | 0 | 30 | 6195 | 74 | 171 | 23 | 1 | 0 |
| | 410 | 4 | 0 | 35 | 5738 | 87 | 0 | 7 | 1 | 0 |
| | 1455 | 4 | 0 | 33 | 3290 | 68 | 14 | 21 | 1 | 0 |
| | 2086 | 0 | 0 | 33 | 1360 | 38 | 31 | 18 | 0 | 0 |
| | 506 | 0 | 0 | 38 | 1760 | 29 | 264 | 3 | 1 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 2178 | 8 | 0 | 40 | 498 | 11 | 12 | 6 | 1 | 1 |
| | 1599 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 1476 | 2 | 1 | 30 | 2505 | 27 | 0 | 9 | 0 | 1 |
| | 1382 | 0 | 1 | 28 | 0 | 0 | 0 | 0 | 1 | 1 |
| | 368 | 2 | 0 | 40 | 180 | 8 | 11 | 5 | 0 | 1 |

## Normalization:

Program:

```
import pandas as panda
import numpy as numpy
import matplotlib.pyplot as plot
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = panda.read_csv("salary.csv")
df
print(df.shape)
df

df.isnull().any()
df.isnull().sum()
df['Salary'] = df['Salary'].fillna(0)
df

df['YearsExperience'] = df['YearsExperience'].fillna(0)
df

from sklearn import preprocessing
a=nump.array(df['YearsExperience'])
print(a)
print('\n')
b=preprocessing.normalize([a])
print(b)
```

Output :

https://github.com/Akshata7890/Normalizatioin-output1.git

**Standardization :**

Program :

```
from sklearn import preprocessing
import numpy as np
x=np.array([[-500.5],
        [-100.1],
        [0],
        [100.1],
        [900.9]])
scaler=preprocessing.StandardScaler()
standardized_x=scaler.fit_transform(x)
print(x)
print(standardized_x)
```

Output :

```
[[-500.5]
 [-100.1]
 [   0. ]
 [ 100.1]
 [ 900.9]]
[[-1.26687088]
 [-0.39316683]
 [-0.17474081]
 [ 0.0436852 ]
 [ 1.79109332]]
```

# WEEK 06 :

**Explore the options of train_test_split()**

Program :

```
# import modules
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# read the dataset
df = pd.read_csv('Real-estate.csv')

# get the locations
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.05, random_state=0)
print(X)
print("splitted",y)
```

Output :

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

[150 rows x 5 columns]
splitted 0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

**Create a model to analyze the relation between crop yield and rain fall rate (using stats model)**

Program :

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import numpy as np

df=pd.read_csv("CROPYIELD.csv")
```

```
df.head()

x=df[['CROP YIELD']]
y=df[['RAINFALL RATE']]
X=x.astype(float)
Y=y.astype(float)
model=sm.OLS(Y,X).fit()
print(model.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:          RAINFALL RATE   R-squared (uncentered):                   0.650
Model:                            OLS   Adj. R-squared (uncentered):              0.612
Method:                 Least Squares   F-statistic:                              16.74
Date:                Mon, 02 Jan 2023   Prob (F-statistic):                     0.00271
Time:                        12:17:40   Log-Likelihood:                         -52.445
No. Observations:                  10   AIC:                                      106.9
Df Residuals:                       9   BIC:                                      107.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
CROP YIELD     1.1609      0.284      4.092      0.003       0.519       1.803
==============================================================================
Omnibus:                        1.412   Durbin-Watson:                   1.484
Prob(Omnibus):                  0.494   Jarque-Bera (JB):                0.934
Skew:                          -0.481   Prob(JB):                        0.627
Kurtosis:                       1.852   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

**Model Evaluation & testing, Evaluate regression model: Evaluation Metric Coefficient of Determination or R-Squared (R2) and Root Mean Squared Error (RSME):**

Program :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv("User_Data.csv")
# input
x = dataset.iloc[:, [2, 3]].values
# output
y = dataset.iloc[:, 4].values
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_state=0)
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)
print (X_train[0:10, :])
```

```
Out: [[ 0.99180116 -0.71109705]
 [ 1.20875767 -1.26624558]
 [-0.74385087 -0.30893752]
 [-0.96080738  1.22388101]
 [-0.52689437  1.45985059]
 [-0.09298136  1.35076556]
 [-1.39472039  0.50219935]
 [ 0.77484466 -1.31216288]
 [ 0.99180116 -0.72688326]
 [-1.8286334   1.42548698]]
```

```
#Training the model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state = 0)
model.fit(X_train, y_train)
```

out: LogisticRegression(random_state=0)

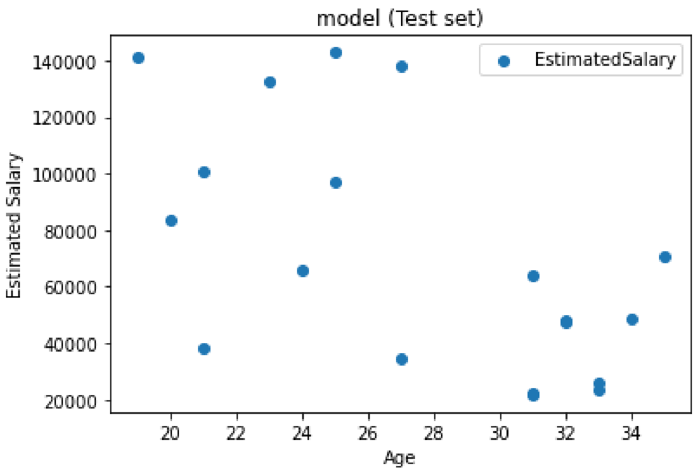
```
y_pred = model.predict(X_test)
#Evauation metrics
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
```

```
Out: Confusion Matrix :
 [[2 0]
 [3 0]]
```

```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

Out: Accuracy :  0.4

```
plt.scatter(x='Age', y='EstimatedSalary',data=dataset)
plt.title('model (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




```
from sklearn.metrics import mean_squared_error
mse=mean_squared_error(y_test,y_pred)
print("MSE: %.2f"% (mse)
```

Out: MSE: 0.60

```
import math
rmse=math.sqrt(mse)
print("RMSE: %.2f"% (rmse))
```

Out: RMSE: 0.77

**Perform data exploration, preprocessing and splitting on datasets like**

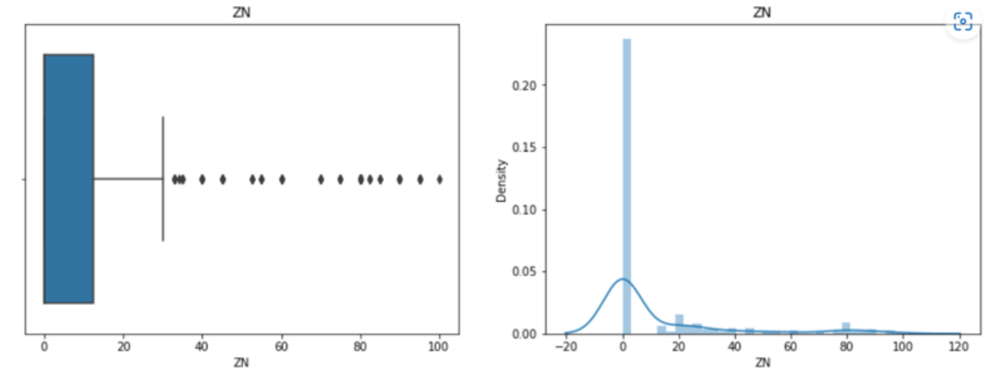
**i) Boston housing price from sci-kit learn datasets**

Program :

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("HousingData.csv")
df.head()

#Processing data
df.isnull().sum()

#Handling missing values by using mean
df.RAD.mean()

df.CRIM.fillna(df.CRIM.mean(),inplace=True)
df.isnull().sum()
```

```
#Exploring the data
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title("ZN")
sns.boxplot(df.ZN)
plt.subplot(1,2,2)
plt.title("ZN")
sns.distplot(df.ZN)
```

: <AxesSubplot:title={'center':'ZN'}, xlabel='ZN', ylabel='Density'>




```
#Handling Outliers
q1=df.ZN.quantile(0.25)
q3=df.ZN.quantile(0.75)
iqr=q3-q1
upper_limit=q3+(1.5*iqr)
lower_limit=q1-(1.5*iqr)

#Number of Outliers in ZN
df.loc[(df.ZN>upper_limit)|(df.ZN<lower_limit)]
df.loc[(df.ZN>upper_limit,"ZN")]=upper_limit
df.loc[(df.ZN<lower_limit,"ZN")]=lower_limit
x = df.drop(columns=['MEDV'], axis=1)
y = df['MEDV']

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_state
=42)
```

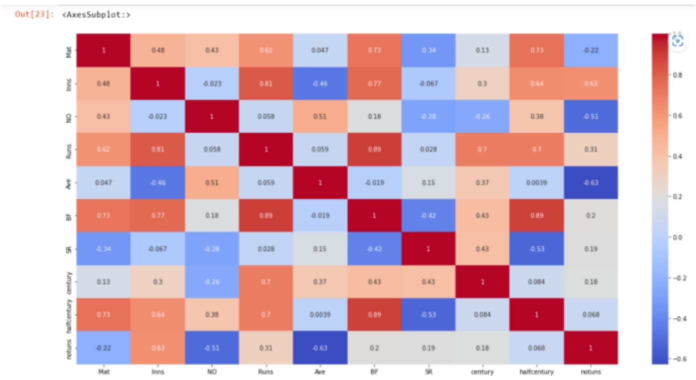
Output : https://github.com/Akshata7890/week-6/blob/main/Boston.ipynb

## ii) Cricket match result - past data

**iii) Performance of a cricket player - past data**

Program :

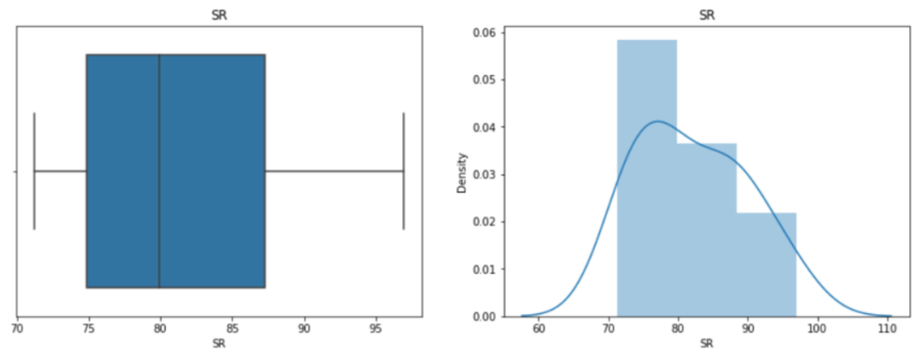
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')

df = pd.read_csv("Cricket_Player_Performance3.csv")
df.head()

# statistical info
df.describe()

# datatype info
df.info()

df1=df.copy()
df1
# check for null values
df1.isnull().sum()
corr = df1.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```



```
def correlation(dataset,threshold):
   col_corr=set()
   corr_matrix=dataset.corr()
   for i in range(len(corr_matrix.columns)):
      for j in range(i):
         if abs(corr_matrix.iloc[i,j])>threshold:
            colname=corr_matrix.columns[i]
            col_corr.add(colname)
   return col_corr
```

```
corr_features=correlation(df1,0.7)
len(set(corr_features))

corr_features
#Feature selection
df1.drop(['SR','halfcentury','BF'],axis=1,inplace=True)
df1

#handling the outliers in 'rm' column
#IQR method
q1=df['SR'].quantile(0.25)
q3=df['SR'].quantile(0.75)
iqr=q3-q1
q1,q3,iqr

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('SR')
sns.boxplot(df['SR'])
plt.subplot(1,2,2)
plt.title('SR')
sns.distplot(df['SR'])
```

```
<AxesSubplot:title={'center':'SR'}, xlabel='SR', ylabel='Density'>
```




```
x = df.drop(columns=['SR'], axis=1)
y = df['SR']
upper_limit=q3+(1.5*iqr)
lower_limit=q1-(1.5*iqr)
upper_limit,lower_limit
x
y
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_state
=5)
x_train
```

```
y_train
```

**iv) Crop yield - past data**

Program :

```
#Data exploration pre-processing & Splitting on Crop yield Dataset
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
df = pd.read_csv("production.csv")
#check for null values
df.isnull().sum()

#Handling the missing value in Rainfall
#fill the 'rice_yield_gap' with mean value since it is numerical value
df['Rainfall'].mean()

df['Rainfall'].fillna(df['Rainfall'].mean(),inplace=True)
df['Rainfall'].isnull().sum()

#Exploratory Data Analysis Visusalizing the data
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('Production')
sns.boxplot(df['Production'])
plt.subplot(1,2,2)
plt.title('Production')
sns.distplot(df['Production'])



plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('Ph')
sns.boxplot(df['Ph'])
plt.subplot(1,2,2)
plt.title('Ph')
sns.distplot(df['Ph'])
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('Rainfall')
sns.boxplot(df['Rainfall'])
plt.subplot(1,2,2)
plt.title('Rainfall')
sns.distplot(df['Rainfall'])


#handling the outliers in 'rm' column
#IQR method
q1=df['Rainfall'].quantile(0.25)
q3=df['Rainfall'].quantile(0.75)
iqr=q3-q1
q1,q3,iqr

upper_limit=q3+(1.5*iqr)
lower_limit=q1-(1.5*iqr)
upper_limit,lower_limit

#number of outliers in 'Production' column
df.loc[(df['Rainfall']>upper_limit)|(df['Rainfall']<lower_limit)

#capping - change the outlier values to upper or lower limit values
df.loc[(df['Rainfall']>upper_limit),'Rainfall']=upper_limit
df.loc[(df['Rainfall']<lower_limit),'Rainfall']=lower_limit

#after removing the outliers in rice_yield_gap column
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('Rainfall')
sns.boxplot(df['Rainfall'])
plt.subplot(1,2,2)
plt.title('Rainfall')
sns.distplot(df['Rainfall'])


#Splitting independent and dependent variables
x = df.drop(columns=['Production'], axis=1)
y = df['Production']

x
y
```

```
x_train
y_train
```

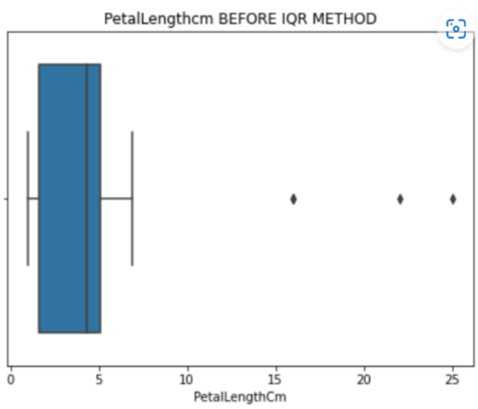
Output : https://github.com/Akshata7890/week-6/blob/main/Crop.ipynb

## WEEK 07:

**Iris dataset from sci-kit learn Perform data exploration, preprocessing and splitting**

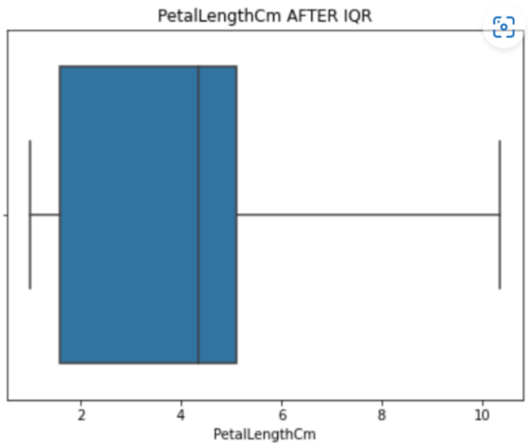
Program :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

df=pd.read_csv("Iris1.csv")
df.head()

df.isnull().sum()
df['PetalLengthCm'].mean()
df['PetalLengthCm'].fillna(df['PetalLengthCm'].mean(),inplace=True)
df['PetalLengthCm'].isnull().sum()

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('PetalLengthcm BEFORE IQR METHOD')
sns.boxplot(df['PetalLengthCm'])
```




```
q1=df['PetalLengthCm'].quantile(0.25)
```

```
q3=df['PetalLengthCm'].quantile(0.75)
iqr=q3-q1
q1,q3,iqr
upper_limit=q3+(1.5*iqr)
lower_limit=q1-(1.5*iqr)
upper_limit,lower_limit


df.loc[(df['PetalLengthCm']>upper_limit),'PetalLengthCm']=upper_limit
df.loc[(df['PetalLengthCm']<lower_limit),'PetalLengthCm']=lower_limit
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.title('PetalLengthCm AFTER IQR')
sns.boxplot(df['PetalLengthCm'])
```




```
x=df.drop(columns=['SepalWidthCm'],axis=1)
y=df['SepalWidthCm']
x_train,x_test,y_test,y_train=train_test_split(x,y,test_size=0.2,random_state
=5)
print(x_test)
print(x_train)
print(y_test)
print(y_train)
```

Output :

**Assessment Build decision tree-based model in python for like Breast cancer Wisconsin(diagnostic) dataset from sci-kit learn Or any classification dataset from UCI, Kaggle**

Program :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data=pd.read_csv('Breast_Cancer.csv')
data.head()

x=data.drop(["diagnosis"],axis=1)
y=data.diagnosis.values

x=(x-np.min(x)) / (np.max(x)-np.min(x))

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state
=42)

dt=DecisionTreeClassifier()
dt.fit(x_train, y_train)
dt.score(x_test,y_test)
```

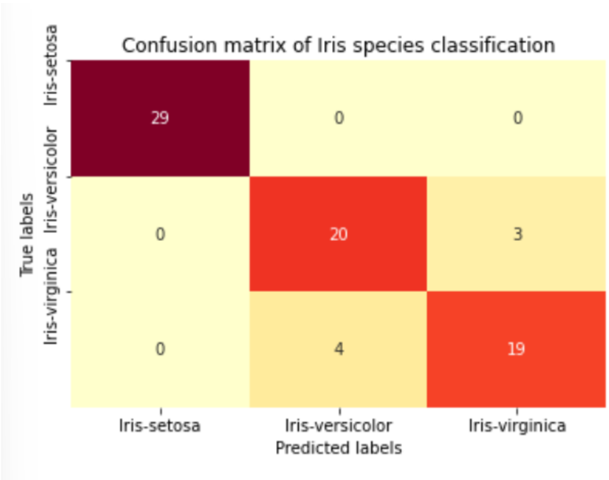
Output: 0.9415204678362573

**Evaluation Metrics for Classification**

Program :

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics, tree
from sklearn.metrics import roc_curve
from sklearn.model_selection import train_test_split
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

iris_dataset = pd.read_csv("Iris.csv")
iris_dataset = iris_dataset.drop(labels = ['Id'], axis=1)
iris_dataset.head()
```

```
iris_values = iris_dataset.values
X,y = iris_values[:,:-1], iris_values[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.5,
random_state = 42)
#Create model
classifier = tree.DecisionTreeClassifier(random_state=42)
classifier.fit(X_train, y_train)
#Make predictions
predictions = classifier.predict(X_test)

conf_matrix = metrics.confusion_matrix(y_test, predictions)
categories = ['Iris-setosa','Iris-versicolor','Iris-virginica']
sns.heatmap(conf_matrix,
annot=True,cmap='YlOrRd',
xticklabels=categories, cbar=False)
plt.yticks(np.arange(3),categories)
plt.ylabel('True labels');
plt.xlabel('Predicted labels');
plt.title('Confusion matrix of Iris species classification');
```




```
print(metrics.classification_report(y_test, predictions, digits=3))
```

```
                 precision    recall  f
1-score   support

    Iris-setosa      1.000     1.000
1.000        29
Iris-versicolor      0.833     0.870
0.851        23
 Iris-virginica      0.864     0.826
0.844        23

       accuracy
0.907        75
      macro avg      0.899     0.899
0.899        75
   weighted avg      0.907     0.907
0.907        75
```

metrics.classification_report(y_test, predictions, digits=3, output_dict=True)

```
{'Iris-setosa': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 29},
 'Iris-versicolor': {'precision': 0.8333333333333334,
  'recall': 0.8695652173913043,
  'f1-score': 0.851063829787234,
  'support': 23},
 'Iris-virginica': {'precision': 0.8636363636363636,
  'recall': 0.8260869565217391,
  'f1-score': 0.8444444444444444,
  'support': 23},
 'accuracy': 0.9066666666666666,
 'macro avg': {'precision': 0.8989898989898991,
  'recall': 0.8985507246376812,
  'f1-score': 0.8985027580772261,
  'support': 75},
 'weighted avg': {'precision': 0.907070707070707,
  'recall': 0.9066666666666666,
  'f1-score': 0.906622537431048,
  'support': 75}}
```

# WEEK 08:

## 1. Build Logistic regression model in python

import pandas as pd

df=pd.read_csv('iris .csv')

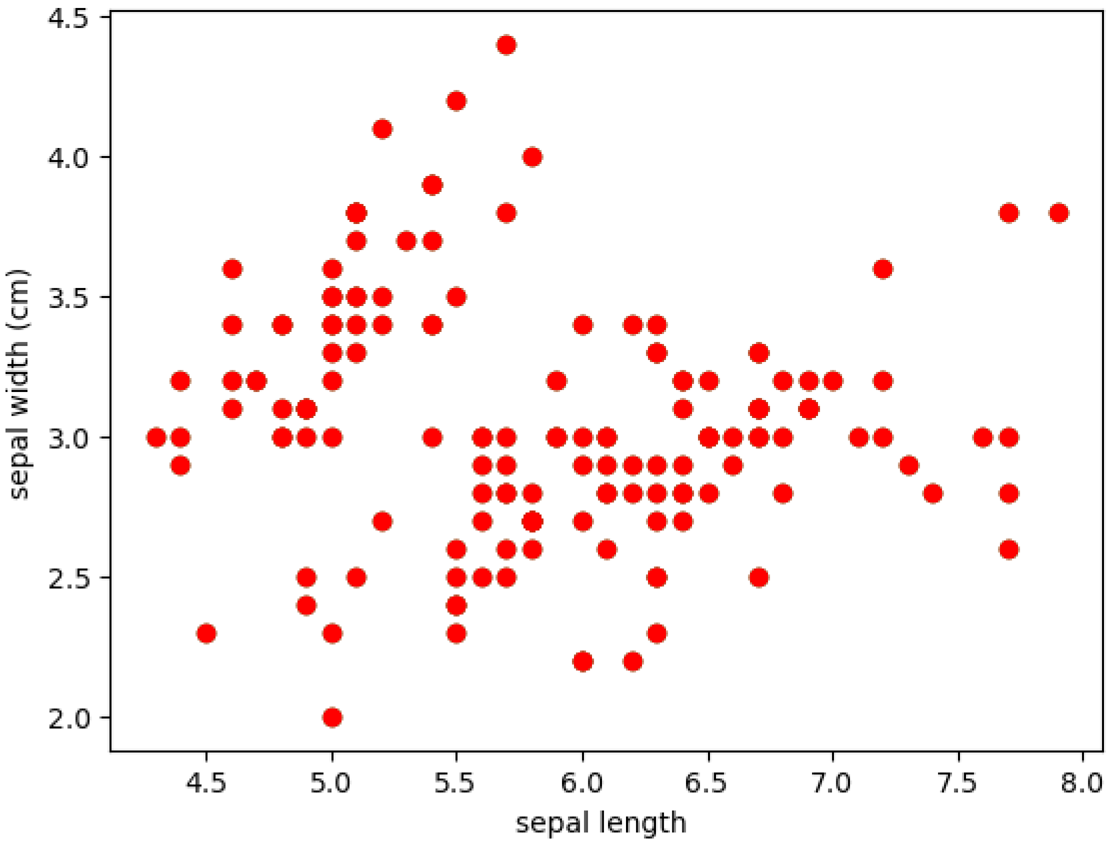
df

from matplotlib import pyplot as plt

plt.xlabel('sepal length ')

plt.ylabel('sepal width (cm)')

plt.scatter(df['sepal_length'],df['sepal_width'],color='green') plt.scatter(df['sepal_length'],df['sepal_width'],color='red',)

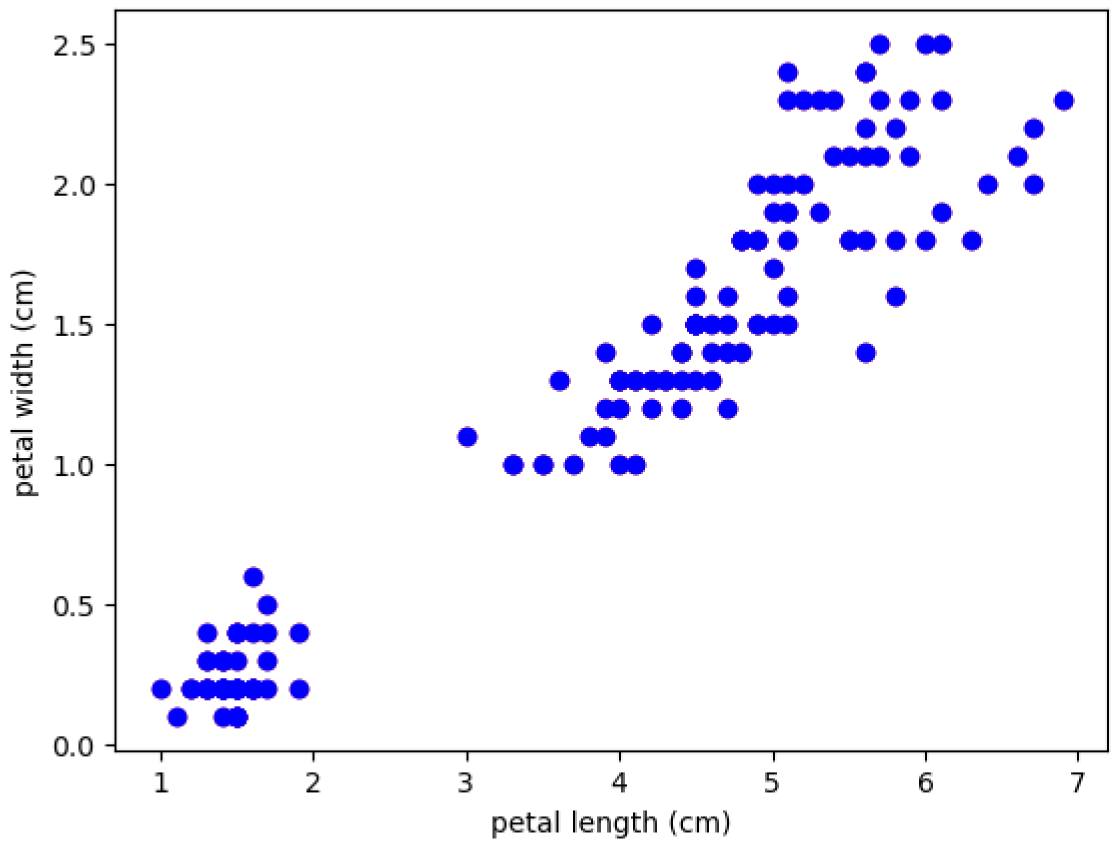



plt.xlabel('petal length (cm)')

plt.ylabel('petal width (cm)')

plt.scatter(df['petal_length'],df['petal_width'],color='red')

plt.scatter(df['petal_length'],df['petal_width'],color='blue')




```
from sklearn.model_selection import train_test_split
from sklearn import linear_model, metrics
x=df.drop(['species'],axis='columns')


y=df.species
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
model = linear_model.LogisticRegression()
model.fit(x, y)
model.score(x_test,y_test)
```

**Out:** 0.9666666666666667

## 2. Build SVM model in python

```
import pandas as pd
df=pd.read_csv('iris .csv')
df
```

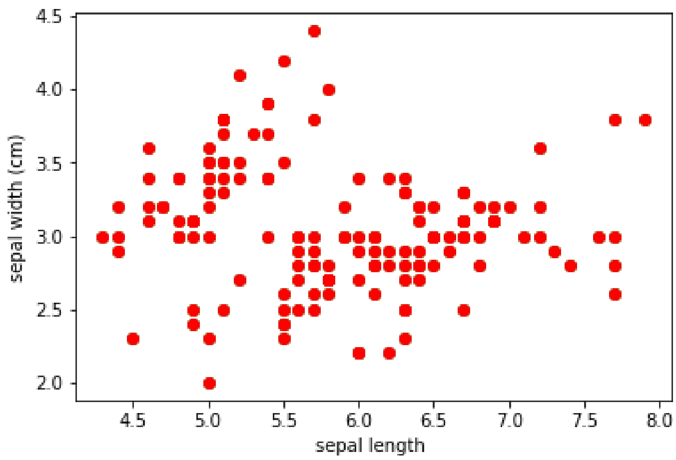
**Out:**

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| ... | ... | ... | ... | ... | ... |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

150 rows × 5 columns

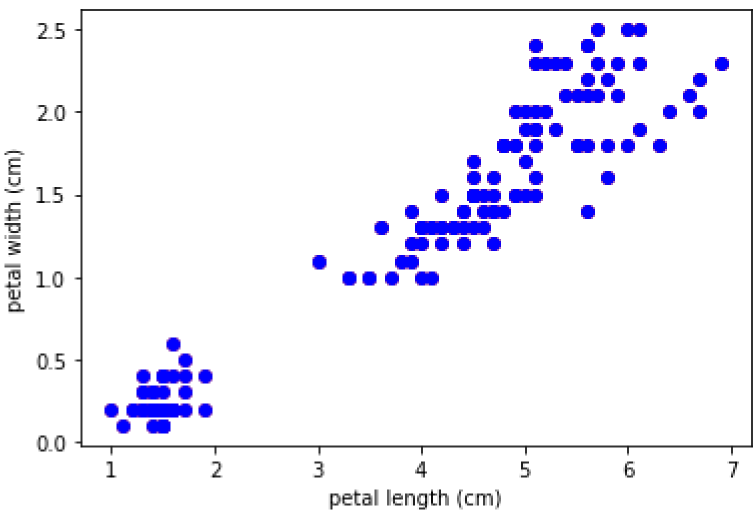
```
from matplotlib import pyplot as plt
plt.xlabel('sepal length ')
plt.ylabel('sepal width (cm)')
plt.scatter(df['sepal_length'],df['sepal_width'],color='green')
plt.scatter(df['sepal_length'],df['sepal_width'],color='red',)
```

**Out:**




```
plt.xlabel('petal length (cm)')

plt.ylabel('petal width (cm)')

plt.scatter(df['petal_length'],df['petal_width'],color='red')

plt.scatter(df['petal_length'],df['petal_width'],color='blue')
```

**Out:**

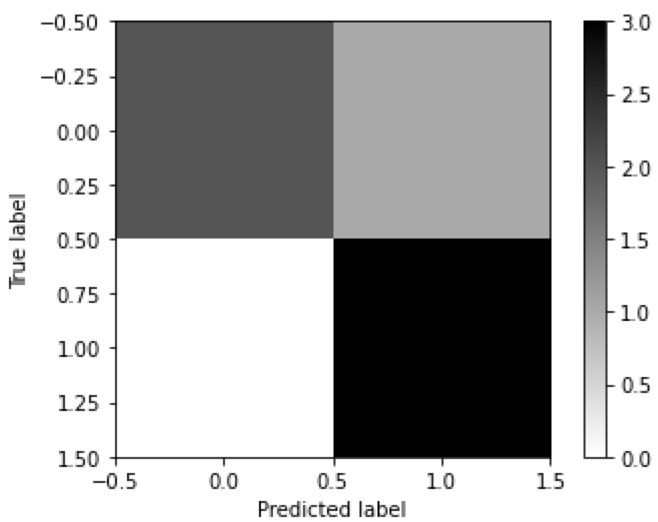



```
from sklearn.model_selection import train_test_split

x=df.drop(['species'],axis='columns')


y=df.species

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)

len(x_train)

len(x_test)


from sklearn.svm import SVC
```

```
model = SVC()
model.fit(x, y)
model.score(x_test,y_test)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import pandas as pd

# true labels
y_true = [0, 0, 0, 1, 1, 1]

# predicted labels
y_pred = [0, 0, 1, 1, 1, 1]

# compute confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

# plot confusion matrix
plt.imshow(conf_matrix, cmap='binary', interpolation='None')
plt.colorbar()
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()


confusion_matrix = pd.crosstab(y_true, y_pred, rownames=['True'], colnames=['Predicted'])
print("\nConfusion matrix\n")
print("\n")


print(confusion_matrix)
print("\n")
```

```
precision = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[0][1])
print("\n")
print("\nPRECISION:\n")
print(precision)

print("\n")
recall = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[1][0])
print("\nRECALL:\n")
print(recall)
recall = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[1][0])
print("\n")
print("\nF1_SCORE:\n")
f1_score = 2 * (precision * recall) / (precision + recall)
print(f1_score)
```

**Out:**




Confusion matrix

```
Predicted  0  1
True
0          2  1
1          0  3
```

PRECISION:

1.0

RECALL:

0.75

F1_SCORE:

0.8571428571428571

## 3. Build Random Forest-based model in python

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import pandas as pd

# Load the dataset

X = pd.read_csv("Breast_cancer_data.csv")

y = X.pop('diagnosis')

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the model

model = RandomForestClassifier()

# Train the model on the training data

model.fit(X_train, y_train)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Calculate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy: ", accuracy)
```

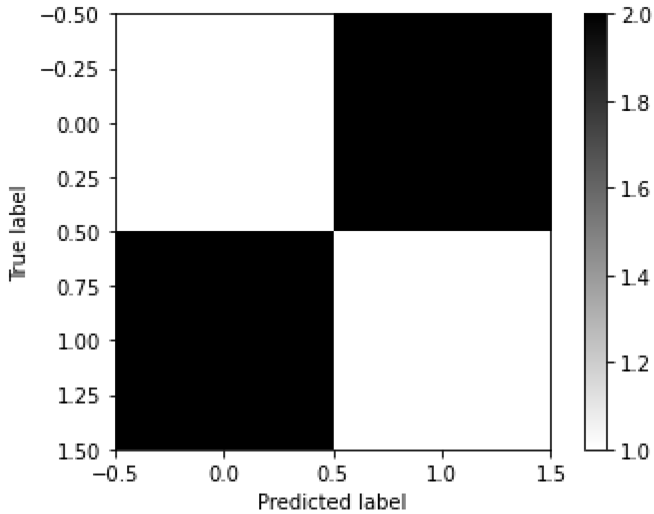
**Out:** Accuracy: 0.9473684210526315

```
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
```

```
import pandas as pd
# true labels
y_true = [1, 0, 1, 0, 1, 0]
# predicted labels
y_pred = [0, 1, 1, 0, 0, 1]

# compute confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
# plot confusion matrix
plt.imshow(conf_matrix, cmap='binary', interpolation='None')
plt.colorbar()
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
confusion_matrix = pd.crosstab(y_true, y_pred, rownames=['True'], colnames=['Predicted'])
print("\nConfusion matrix\n")
print("\n")
print(confusion_matrix)
print("\n")
precision = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[0][1])
print("\n")
print("\nPRECISION:\n")
print(precision)
print("\n")
recall = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[1][0])
print("\nRECALL:\n")
print(recall)
recall = confusion_matrix[1][1] / (confusion_matrix[1][1] + confusion_matrix[1][0])
print("\n")
print("\nF1_SCORE:\n")
f1_score = 2 * (precision * recall) / (precision + recall)
```

print(f1_score)

**Out:**




Confusion matrix

Predicted  0  1

True

0          1  2

1          2  1

PRECISION:

0.3333333333333333

RECALL:

0.3333333333333333

F1_SCORE:

0.3333333333333333

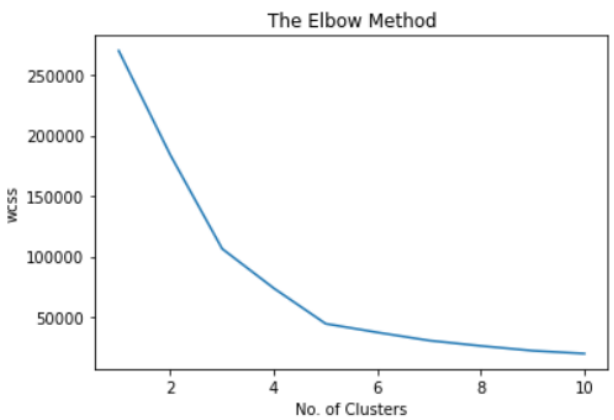
# WEEK 09:

**Evaluation Metrics**

6. **Inertia**

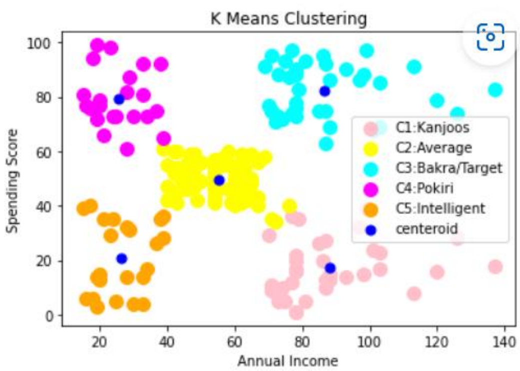
Program :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('Mall_Customers.csv')
data.head()

x = data.iloc[:, 3:5].values

#Evaluating metrics using pre-defined function Inertia

from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters = i)
   kmeans.fit(x)
   wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('No. of Clusters')
plt.ylabel('wcss')
plt.show()
```




```
#Evaluating metric using inertia

km1 = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10,
random_state = 0)
```

```
y_means = km1.fit_predict(x)

plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s = 100, c = 'pink', label =
'C1:Kanjoos')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s = 100, c = 'yellow', label =
'C2:Average')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s = 100, c = 'cyan', label =
'C3:Bakra/Target')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s = 100, c = 'magenta', label =
'C4:Pokiri')
plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s = 100, c = 'orange', label =
'C5:Intelligent')
plt.scatter(km1.cluster_centers_[:,0], km1.cluster_centers_[:, 1], s = 50, c = 'blue' ,
label = 'centeroid')

plt.title('K Means Clustering')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
km.inertia_
km1.cluster_centers_
```




In [8]:

```
km.inertia_
```

Out[8]:

```
44448.45544793371
```

In [9]:

```
km1.cluster_centers_
```

Out[9]:

```
array([[88.2       , 17.11428571],
       [55.2962963 , 49.51851852],
       [86.53846154, 82.12820513],
       [25.72727273, 79.36363636],
       [26.30434783, 20.91304348]])
```

**7. Dunn Index**

Program :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = pd.read_csv('Mall_Customers.csv')
data.head()

x = data.iloc[:, 3:5].values
km = KMeans(n_clusters = 5)
# Change value of n_clusters to 4,5 or 6 to check how Dunn index
changes
y_means = km.fit_predict(x)

import math
def max_intra_cluster_distance(cluster_points,centroid):
    max_dist = float('-inf')
    #print(max_dist)
    for point in cluster_points:
        dist = math.dist(centroid, point)
        if dist > max_dist:
            max_dist = dist
    return max_dist

max_distance = float('-inf')
for i in range(0,5):
    cluster_points = x[y_means == i]
    centroid = list(km.cluster_centers_[i])
    dist = max_intra_cluster_distance(cluster_points,centroid)
    if dist > max_distance:
        max_distance = dist

print("Maximum intra cluster distance",max_distance)
max_intra_cluster_dist = max_distance
```

output : Maximum intra cluster distance 50.46906864807208

```
def min_inter_cluster_distance(centroids):
    min_dist = float('inf')
    for i in range(len(centroids)):
        for j in range(len(centroids)):
            dist = math.dist(centroids[i], centroids[j])
            if(i==j):
```

```
            continue
        else:
            if dist < min_dist:
                min_dist = dist
    return min_dist

centroids = km.cluster_centers_
min_inter_cluster_dist = min_inter_cluster_distance(centroids)
print("Minimum inter cluster distance",min_inter_cluster_dist)
```

output : Minimum inter cluster distance 40.728445567908395

```
dunn_index  #Higher is better
```

output : 0.8069981606340624

**Dimensionality Reduction using PCA in python**

Program :

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

data = pd.read_csv('iris.csv')
data

y = data.pop("Species")
data.head()
data.describe()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = data.copy()
x = scaler.fit_transform(X)
x[:5,:5]
```

```
array([[-1.72054204, -0.90068117,  1.03205722,
-1.3412724 , -1.31297673],
       [-1.69744751, -1.14301691, -0.1249576 ,
-1.3412724 , -1.31297673],
       [-1.67435299, -1.38535265,  0.33784833,
-1.39813811, -1.31297673],
       [-1.65125846, -1.50652052,  0.10644536,
-1.2844067 , -1.31297673],
       [-1.62816394, -1.02184904,  1.26346019,
-1.3412724 , -1.31297673]])
```

```
from sklearn.decomposition import PCA
pca = PCA(random_state=42)
pca.fit(x)
pca.components_
```

Output :

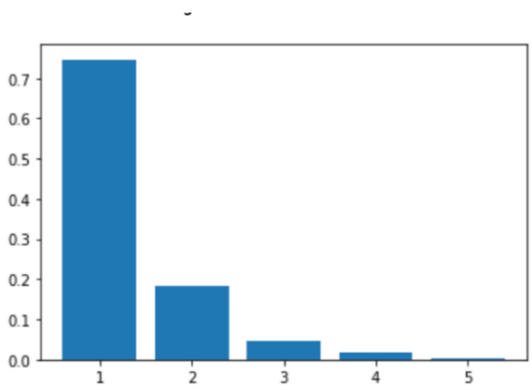
```
array([[ 0.48136016,  0.44844975, -0.23195044,
0.51079205,  0.5024696 ],
       [-0.02275157,  0.38285827,  0.92007839,
0.03074857,  0.07356757],
       [ 0.67406853, -0.64520569,  0.27427786,
-0.13238322,  0.19127876],
       [-0.55978662, -0.40999945,  0.09491665,
0.28817343,  0.65305918],
       [ 0.0067323 , -0.26061932,  0.12416613,
0.79848404, -0.52824072]])
```

```
pca.explained_variance_ratio_
```

Output :

```
array([0.7470533 , 0.18435257, 0.04682624, 0.0
1764767, 0.00412021])
```

```
plt.bar(range(1,len(pca.explained_variance_ratio_)+1),
pca.explained_variance_ratio_)
```




```
pc2 = PCA(n_components=2, random_state=42)
newdata = pc2.fit_transform(x)
newdata.shape
```

Output :

```
(150, 2)
```

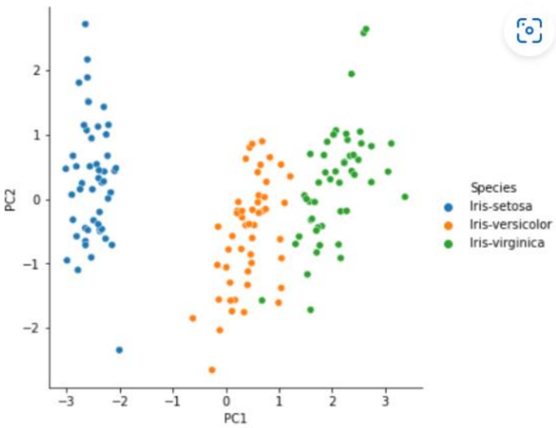
```
df = pd.DataFrame(newdata, columns=["PC1", "PC2"])
df.head()
```

| | PC1 | PC2 |
|---|---|---|
| 0 | -2.816339 | 0.506051 |
| 1 | -2.645527 | -0.651799 |
| 2 | -2.879481 | -0.321036 |
| 3 | -2.810934 | -0.577363 |
| 4 | -2.879884 | 0.670468 |

```
df_final = pd.concat([df, y], axis=1)
df_final.head()
```

| | PC1 | PC2 | Species |
|---|---|---|---|
| 0 | -2.816339 | 0.506051 | Iris-setosa |
| 1 | -2.645527 | -0.651799 | Iris-setosa |
| 2 | -2.879481 | -0.321036 | Iris-setosa |
| 3 | -2.810934 | -0.577363 | Iris-setosa |
| 4 | -2.879884 | 0.670468 | Iris-setosa |

```
import seaborn as sns
sns.pairplot(data=df_final, x_vars=["PC1"], y_vars=["PC2"], hue = "Species",
height=5)
```




# WEEK 10:

## 1. Build a shallow Deep learning model using keras

```
from sklearn.datasets import make_classification

from keras.utils import to_categorical

from keras.models import Sequential
```

```
from keras.layers import Dense, Activation

X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, n_classes=3, random_state=42)

num_classes = 3

y = to_categorical(y, num_classes)

model = Sequential()

model.add(Dense(units=32, input_dim=20))

model.add(Activation('relu'))

model.add(Dense(units=3))

model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X, y, epochs=5, batch_size=32)
```

**Out[10]:**

```
Epoch 1/5

32/32 [==============================] - 1s 2ms/step - loss: 2.0170 - accuracy: 0.3930

Epoch 2/5

32/32 [==============================] - 0s 2ms/step - loss: 1.2383 - accuracy: 0.4890

Epoch 3/5

32/32 [==============================] - 0s 2ms/step - loss: 0.9506 - accuracy: 0.5880

Epoch 4/5

32/32 [==============================] - 0s 2ms/step - loss: 0.8085 - accuracy: 0.6440

Epoch 5/5

32/32 [==============================] - 0s 2ms/step - loss: 0.7183 - accuracy: 0.6830

<keras.callbacks.History at 0x19109514b20>
```

```
loss, accuracy = model.evaluate(X, y)

print('Loss:', loss)

print('Accuracy:', accuracy)
```

**Out:**

```
32/32 [==============================] - 0s 2ms/step - loss: 0.6754 - accuracy: 0.7090

Loss: 0.6754007935523987
```

Accuracy: 0.7089999914169312

## 2. Build a Deep Neural Network model using keras

```
import pandas as pd

data = pd.read_csv('diabetes.csv')

x = data.drop("Outcome", axis=1)

y = data["Outcome"]

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(12, input_dim=8, activation="relu"))

model.add(Dense(12, activation="relu"))

model.add(Dense(1, activation="sigmoid"))

model = Sequential()  #define model

model.add(Dense(12, input_dim=8, activation="relu"))

model.add(Dense(8, activation="relu"))

model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"]) #compile model

model.fit(x,y, epochs=150, batch_size=10)  #training

_, accuracy = model.evaluate(x,y)    #testing

print("Model accuracy: %.2f"% (accuracy*100))

predictions = model.predict(x)     #make predictions

#round the prediction

rounded = [round(x[0]) for x in predictions]
```

**Out:**

**Epoch 1/150**

**77/77 [==============================] - 1s 1ms/step - loss: 4.5659 - accuracy: 0.4557**

**Epoch 2/150**

**77/77 [==============================] - 0s 1ms/step - loss: 1.3151 - accuracy: 0.6094**

**Epoch 3/150**

**77/77 [==============================] - 0s 1ms/step - loss: 1.1676 - accuracy: 0.6146**

**.**

**.**

**.**

**Epoch 148/150**

**77/77 [==============================] - 0s 1ms/step - loss: 0.5451 - accuracy: 0.7135**

**Epoch 149/150**

**77/77 [==============================] - 0s 1ms/step - loss: 0.5476 - accuracy: 0.7109**

**Epoch 150/150**

**77/77 [==============================] - 0s 1ms/step - loss: 0.5412 - accuracy: 0.7253**

**24/24 [==============================] - 0s 950us/step - loss: 0.5274 - accuracy: 0.7266**

**Model accuracy: 72.66**

**24/24 [==============================] - 0s 871us/step**

## 3. Build a Classification model using deep Neural Network model using keras

```
import pandas as pd

data = pd.read_csv('diabetes.csv')

x = data.drop("Outcome", axis=1)

y = data["Outcome"]

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(12, input_dim=8, activation="relu"))

model.add(Dense(12, activation="relu"))

model.add(Dense(1, activation="sigmoid"))

model = Sequential()  #define model
```

```
model.add(Dense(12, input_dim=8, activation="relu"))

model.add(Dense(8, activation="relu"))

model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
#compile model

model.fit(x,y, epochs=150, batch_size=10)  #training

_, accuracy = model.evaluate(x,y)    #testing

print("Model accuracy: %.2f"% (accuracy*100))

predictions = model.predict(x)     #make predictions

#round the prediction

rounded = [round(x[0]) for x in predictions]
```