**Experiment No.: 1**

**Aim**

To implement Matrix operations (using vectorization), transformation using python and SVD using Python

**Questions**

(a) Matrix operations (using vectorization),

(b) transformation using python and

(c) SVD using Python.

**Program and Output**

import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array

print("type: " ,type(a))            # Prints "<class 'numpy.ndarray'>"

print("shape: " ,a.shape)            # Prints "(3,)"

print(a[0], a[1], a[2])   # Prints "1 2 3"

a[0] = 5                  # Change an element of the array

print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array

print("\n shape of b:",b.shape)                     # Prints "(2, 3)"

print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"

a = np.zeros((3,3))   # Create an array of all zeros

print("All zeros matrix:\n  " ,a)

b = np.ones((1,2))    # Create an array of all ones

print("\nAll ones matrix:\n  " ,b)              # Prints "[[ 1.  1.]]"

d = np.eye(2)        # Create a 2x2 identity matrix

print("\n identity matrix: \n",d)

e = np.random.random((2,2))  # Create an array filled with random values

print("\n random matrix: \n",e)

* OUTPUT

shape: (3,)

1 2 3

[5 2 3]

shape of b: (2, 3)

1 2 4

All zeros matrix:

[[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]]

All ones matrix:

[[1. 1.]]

identity matrix:

[[1. 0.]

[0. 1.]]

random matrix:

[[0.19072046 0.82646264]

[0.24096376 0.46100121]]

#vectorized sum

print("Vectorized sum example\n")

x = np.array([[1,2],[3,4]])

print("x:\n " ,x)

print("sum: ",np.sum(x))  # Compute sum of all elements; prints "10"

print("sum axis = 0: " ,np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"

print(" sum axis = 1: " ,np.sum(x, axis=1))  # Compute sum of each row; prints

#matrix dot product

a = np.arange(10000)

b = np.arange(10000)

print("a", a)

print("b", b)

dp = np.dot(a,b)

print("Dot product: \n" ,dp)

#outer product

op = np.outer(a,b)

print("\n Outer product: \n" ,op)

#elementwise product

ep = np.multiply(a, b)

print("\n Element Wise product:  \n" ,ep)

* OUTPUT

Vectorized sum example

x:

[[1 2]

[3 4]]

sum: 10

sum axis = 0: [4 6]

sum axis = 1: [3 7]

a [ 0 1 2 ... 9997 9998 9999]

b [ 0 1 2 ... 9997 9998 9999]

Dot product:

333283335000

Outer product:

[[ 0 0 0 ... 0 0 0]

[ 0 1 2 ... 9997 9998 9999]

[ 0 2 4 ... 19994 19996 19998]

...

[ 0 9997 19994 ... 99940009 99950006 99960003]

[ 0 9998 19996 ... 99950006 99960004 99970002]

[ 0 9999 19998 ... 99960003 99970002 99980001]]

Element Wise product:

[ 0 1 4 ... 99940009 99960004 99980001]

import numpy as np

x = np.array([[1,2], [3,4]])

print("Original x: \n " ,x)    # Prints "[[1 2]

            #          [3 4]]"

print("\nTranspose of x: \n" ,x.T)  # Prints "[[1 3]

* OUTPUT

Original x:

[[1 2]

[3 4]]

Transpose of x:

[[1 3]

[2 4]]

# Singular-value decomposition

from numpy import array

from scipy.linalg import svd

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print("A: \n%s" %A)

# SVD

U, s, VT = svd(A)

print("\nU: \n%s" %U)

print("\ns: \n %s" %s)

print("\nV^T: \n %s" %VT)

* OUTPUT

A:

[[1 2]

[3 4]

[5 6]]

U:

[[-0.2298477 0.88346102 0.40824829]

[-0.52474482 0.24078249 -0.81649658]

[-0.81964194 -0.40189603 0.40824829]]

s:

[9.52551809 0.51430058]

V^T:

[[-0.61962948 -0.78489445]

[-0.78489445 0.61962948]]

**Experiment No.: 2**

**Aim**

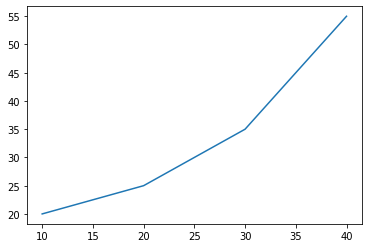
Programs using matplotlib / plotly / bokeh / seaborn for data visualisation.

**Questions**

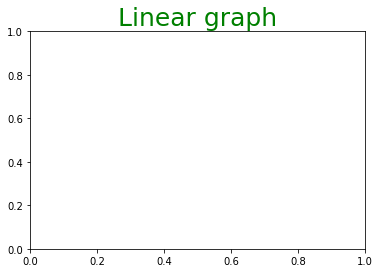
1. Visualization using matplotlib
2. Visualization using seaborn
3. Visualization using plotly

**Program and Output**

import matplotlib.pyplot as plt  
x = [10, 20, 30, 40]  
y = [20, 25, 35, 55]  
plt.plot(x, y)  
plt.show()

  
plt.title("Linear graph")  
plt.show()  
plt.title("Linear graph", fontsize=25, color="green")  
plt.show()





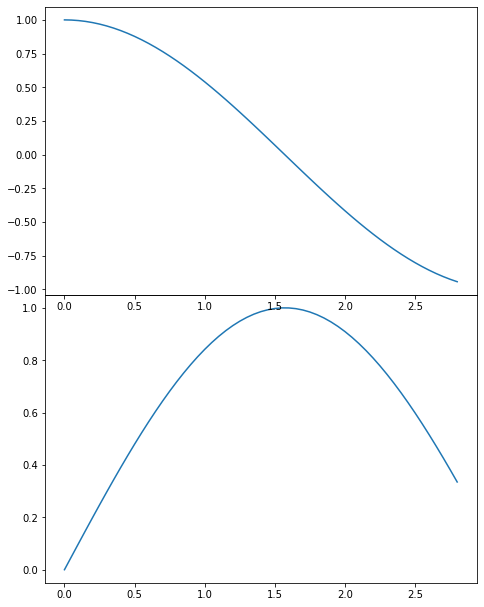
*# Adding label on the y-axis*  
plt.ylabel('Y-Axis')  
*# Adding label on the x-axis*  
plt.xlabel('X-Axis')  
plt.show()

  
plt.ylim(0, 80)  
plt.xticks(x, labels=["one", "two", "three", "four"])  
plt.show()

  
plt.legend(["GFG"])  
plt.show()



import matplotlib.pyplot as plt  
import numpy  
fig = plt.figure()  
*# Generate line graph*  
x = numpy.arange(0, 1.414\*2, 0.05)  
y1 = numpy.sin(x)  
y2 = numpy.cos(x)  
axes1 = fig.add\_axes([0, 0, 1, 1])  
axes1.plot(x, y1)  
axes2 = fig.add\_axes([0, 1, 1, 1])  
axes2.plot(x, y2)  
   
*# Show plot*  
plt.show()



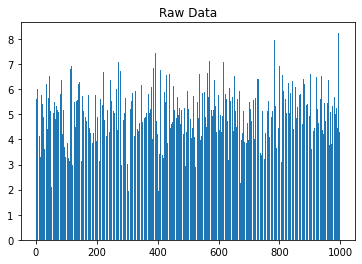
import matplotlib.pyplot as plt  
   
x = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
y1 = [1, 3, 5, 3, 1, 3, 5, 3, 1]  
y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]  
plt.plot(x, y1, label="line L")  
plt.plot(x, y2, label="line H")  
plt.plot()  
   
plt.xlabel("x axis")  
plt.ylabel("y axis")  
plt.title("Line Graph Example")  
plt.legend()  
plt.show()

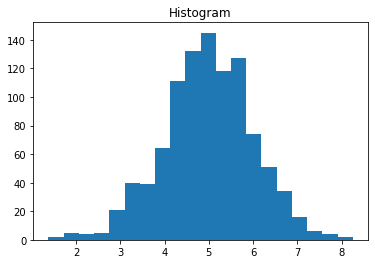


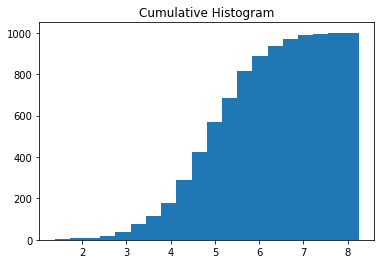
import matplotlib.pyplot as plt  
x1 = [1, 3, 4, 5, 6, 7, 9]  
y1 = [4, 7, 2, 4, 7, 8, 3]  
   
x2 = [2, 4, 6, 8, 10]  
y2 = [5, 6, 2, 6, 2]  
plt.bar(x1, y1, label="Blue Bar", color='b')  
plt.bar(x2, y2, label="Green Bar", color='g')  
plt.plot()  
plt.xlabel("bar number")  
plt.ylabel("bar height")  
plt.title("Bar Chart Example")  
plt.legend()  
plt.show()



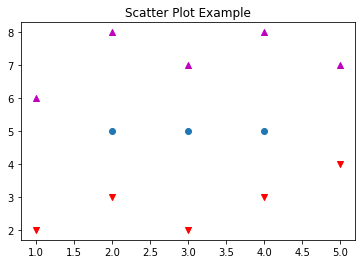
import matplotlib.pyplot as plt  
import numpy as np  
n = 5 + np.random.randn(1000)  
   
m = [m **for** m **in** range(len(n))]  
plt.bar(m, n)  
plt.title("Raw Data")  
plt.show()  
   
plt.hist(n, bins=20)  
plt.title("Histogram")  
plt.show()  
   
plt.hist(n, cumulative=True, bins=20)  
plt.title("Cumulative Histogram")  
plt.show()





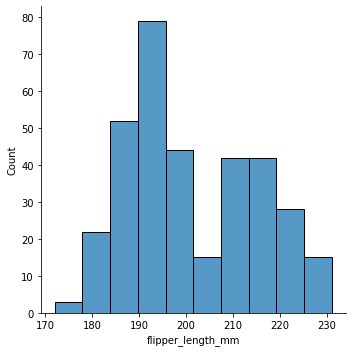


import matplotlib.pyplot as plt  
x1 = [2, 3, 4]  
y1 = [5, 5, 5]  
   
x2 = [1, 2, 3, 4, 5]  
y2 = [2, 3, 2, 3, 4]  
y3 = [6, 8, 7, 8, 7]  
   
plt.scatter(x1, y1)  
plt.scatter(x2, y2, marker='v', color='r')  
plt.scatter(x2, y3, marker='^', color='m')  
plt.title('Scatter Plot Example')  
plt.show()



*#Bubble chart*  
import plotly.express as px  
  
df = px.data.iris()  
  
fig = px.scatter(df, x="sepal\_width", y="sepal\_length",  
 color="species",  
 size='petal\_length',  
 hover\_data=['petal\_width'])  
  
fig.show()  
import seaborn as sns  
penguins = sns.load\_dataset("penguins")  
sns.displot(penguins, x="flipper\_length\_mm")

<seaborn.axisgrid.FacetGrid at 0x7fe3007627d0>



**Experiment No.: 3**

**Aim**

Programs to handle data using pandas.

**Questions**

1. Pandas Series
2. Pandas Dataframe
3. Pandas CSV operations

**Program and Output**

import pandas as pd  
import numpy as np  
print(pd.Series(np.array([1,2,3,4,5,6,7])))

*0 1  
1 2  
2 3  
3 4  
4 5  
5 6  
6 7  
#How to create Series with Mutable index*  
print(pd.Series(np.array([1,2,3,4,5,6,7]), index=['a','b','c','d','e','f','g']))

*a 1  
b 2  
c 3  
d 4  
e 5  
f 6  
g 7*

*#Creating a series from a Dictionary*  
print(pd.Series({'name':1, 'new':10}))

*name 1  
new 10*

*#Print all the values of the Series by multiplying them by 2.*  
print(pd.Series(np.array([1,2,3,4,5,6,7]), index=['a','b','c','d','e','f','g'])\*2)

*a 2  
b 4  
c 6  
d 8  
e 10  
f 12  
g 14*

*#Print Square of all the values of the series.*  
print(pd.Series(np.array([1,2,3,4,5,6,7]), index=['a','b','c','d','e','f','g'])\*\*2)

*a 1  
b 4  
c 9  
d 16  
e 25  
f 36  
g 49*

*#Print all the values of the Series that are greater than2*  
print(pd.Series(np.array([1,2,3,4,5,6,7]), index=['a','b','c','d','e','f','g'])[pd.Series(np.array([1,2,3,4,5,6,7]), index=['a','b','c','d','e','f','g'])>2])

*c 3  
d 4  
e 5  
f 6  
g 7*  
*#Addition of two series*  
print(pd.Series(np.array([1,2,3,4,5,6,7]))+pd.Series(np.array([11,22,33,44,55,66,77])))

*0 12  
1 24  
2 36  
3 48  
4 60  
5 72  
6 84*

*#Print the first and last 5 elements of a series*  
print(pd.Series(np.array([1,2,3,4,5,6,7])).head(5))

*0 1  
1 2  
2 3  
3 4  
4 5*

*#Print the values from index 0 to 5*  
print(pd.Series(np.array([1,2,3,4,5,6,7]))[:5])

*0 1  
1 2  
2 3  
3 4  
4 5*

*#Selection Using loc, iloc index label*  
print(pd.Series(np.array([1,2,3,4,5,6,7])).loc[:2])  
print(pd.Series(np.array([1,2,3,4,5,6,7])).iloc[:2])

*0 1  
1 2  
2 3  
0 1  
1 2*

*#Retrieve subsets of data using slicing*  
print(pd.Series(np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]))[2:14:2])

*2 3  
4 5  
6 7  
8 9  
10 11  
12 13*

*#create Dataframe From Series*

print(pd.DataFrame(pd.Series([1,2,3,4,5,6,7,8,9])))

*0  
0 1  
1 2  
2 3  
3 4  
4 5  
5 6  
6 7  
7 8  
8 9*

*#DataFrame from List of Dictionaries*  
print(pd.DataFrame({'name':[2,4,5,7,8], 'new':[4,6,8,9,1]}))

*name new  
0 2 4  
1 4 6  
2 5 8  
3 7 9  
4 8 1*

*#Display the first 5 rows of data frame*  
print(pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1]}).head())

name new  
*0 2 4  
1 4 6  
2 5 8  
3 7 9  
4 8 1*

*# Select the last two columns of the data frame*  
print(pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1], 'value':[1,2,3,4,5,6,7,8]}).iloc[: , 1:])

*new value  
0 4 1  
1 6 2  
2 8 3  
3 9 4  
4 1 5  
5 7 6  
6 9 7  
7 1 8*

*# Add two data frames*  
print(pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1]})+ pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1]}))

*name new  
0 4 8  
1 8 12  
2 10 16  
3 14 18  
4 16 2  
5 10 14  
6 4 18  
7 8 2*

*# Demonstrate deletion, and renaming of columns*  
val=pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1], 'value':[1,2,3,4,5,6,7,8], 'value2':[1,2,3,4,5,6,7,8]})  
print(val.drop('name',axis=1))  
val.rename(columns={'name':'name2'}, inplace=True)  
print(val)

*new value value2  
0 4 1 1  
1 6 2 2  
2 8 3 3  
3 9 4 4  
4 1 5 5  
5 7 6 6  
6 9 7 7  
7 1 8 8  
 name2 new value value2  
0 2 4 1 1  
1 4 6 2 2  
2 5 8 3 3  
3 7 9 4 4  
4 8 1 5 5  
5 5 7 6 6  
6 2 9 7 7  
7 4 1 8 8*

*#Demonstrate concat, Merge operations in data frame*  
print(pd.concat([pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1]}),pd.DataFrame({'name':[2,4,5,7,8,5,2,4], 'new':[4,6,8,9,1,7,9,1]})], ignore\_index=True))

*name new  
0 2 4  
1 4 6  
2 5 8  
3 7 9  
4 8 1  
5 5 7  
6 2 9  
7 4 1  
8 2 4  
9 4 6  
10 5 8  
11 7 9  
12 8 1  
13 5 7  
14 2 9  
15 4 1*

data1 = {'key': ['K0', 'K1', 'K2', 'K3'],  
 'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],   
 'Age':[27, 24, 22, 32],}   
data2 = {'key': ['K0', 'K1', 'K2', 'K3'],  
 'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],   
 'Qualification':['Btech', 'B.A', 'Bcom', 'B.hons']}   
df = pd.DataFrame(data1)  
df1 = pd.DataFrame(data2)   
*# print(df, "\n\n", df1)*   
print(pd.merge(df, df1, on='key'))

*key Name Age Address Qualification  
0 K0 Jai 27 Nagpur Btech  
1 K1 Princi 24 Kanpur B.A  
2 K2 Gaurav 22 Allahabad Bcom  
3 K3 Anuj 32 Kannuaj B.hons*

*#Write a Pandas program to join the two given dataframes along rows and*  
*# assign all data*  
  
s1 = pd.DataFrame({  
 'student\_id': ['S1', 'S2', 'S3', 'S4', 'S5'],  
 'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],   
 'marks': [200, 210, 190, 222, 199]})  
  
s2 = pd.DataFrame({  
 'student\_id': ['S4', 'S5', 'S6', 'S7', 'S8'],  
 'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston'],   
 'marks': [201, 200, 198, 219, 201]})  
  
print("Original DataFrames:")  
print(s1)  
print("-------------------------------------")  
print(s2)  
print("\ntwo dataframes along rows:")  
result\_data = pd.concat([s1, s2])  
print(result\_data)

*Original DataFrames:  
 student\_id name marks  
0 S1 Danniella Fenton 200  
1 S2 Ryder Storey 210  
2 S3 Bryce Jensen 190  
3 S4 Ed Bernal 222  
4 S5 Kwame Morin 199  
-------------------------------------  
 student\_id name marks  
0 S4 Scarlette Fisher 201  
1 S5 Carla Williamson 200  
2 S6 Dante Morse 198  
3 S7 Kaiser William 219  
4 S8 Madeeha Preston 201  
  
two dataframes along rows:  
 student\_id name marks  
0 S1 Danniella Fenton 200  
1 S2 Ryder Storey 210  
2 S3 Bryce Jensen 190  
3 S4 Ed Bernal 222  
4 S5 Kwame Morin 199  
0 S4 Scarlette Fisher 201  
1 S5 Carla Williamson 200  
2 S6 Dante Morse 198  
3 S7 Kaiser William 219  
4 S8 Madeeha Preston 201*

*#save dataframe to csv file*  
result\_data.to\_csv('output.csv')

data=pd.read\_csv('output.csv')  
print(data)

*Unnamed: 0 student\_id name marks  
0 0 S1 Danniella Fenton 200  
1 1 S2 Ryder Storey 210  
2 2 S3 Bryce Jensen 190  
3 3 S4 Ed Bernal 222  
4 4 S5 Kwame Morin 199  
5 0 S4 Scarlette Fisher 201  
6 1 S5 Carla Williamson 200  
7 2 S6 Dante Morse 198  
8 3 S7 Kaiser William 219  
9 4 S8 Madeeha Preston 201*

**Experiment No.: 4**

**Aim**

Perform Z-score normalization, Min-max normalization using iris dataset

#single Dimensional array

import pandas as pd

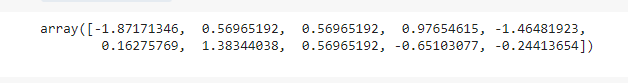
import numpy as np

import scipy.stats as stats

arr= np.array([1,7,7,8,2,6,9,7,4,5])

stats.zscore(arr)

**Output**



#MinMax Scaler

# example of a normalization

from numpy import asarray

from sklearn.preprocessing import MinMaxScaler

# define data

data = asarray([[100, 0.001],

[8, 0.05],

[50, 0.005],

[88, 0.07],

[4, 0.1]])

print(data)

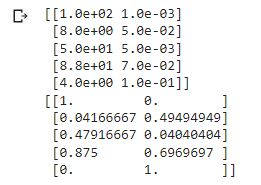
# define min max scaler

scaler = MinMaxScaler()

# transform data

scaled = scaler.fit\_transform(data)

print(scaled)



# example of a normalization

from numpy import asarray

from sklearn.preprocessing import MinMaxScaler

# define data

data = asarray([[100, 0.001],

[8, 0.05],

[50, 0.005],

[88, 0.07],

[4, 0.1]])

print(data)

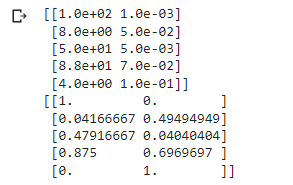
# define min max scaler

scaler = MinMaxScaler()

# transform data

scaled = scaler.fit\_transform(data)

print(scaled)



**Experiment No.: 5**

**Aim**

Implement K-NN Algorithm using iris data set

**Procedure**

# Example of calculating Euclidean distance

from math import sqrt

# calculate the Euclidean distance between two vectors

def euclidean\_distance(row1, row2):

distance = 0.0

for i in range(len(row1)-1):

distance += (row1[i] - row2[i])\*\*2

return sqrt(distance)

# Test distance function

dataset = [[2.7810836,2.550537003,0],

[1.465489372,2.362125076,0],

[3.396561688,4.400293529,0],

[1.38807019,1.850220317,0],

[3.06407232,3.005305973,0],

[7.627531214,2.759262235,1],

[5.332441248,2.088626775,1],

[6.922596716,1.77106367,1],

[8.675418651,-0.242068655,1],

[7.673756466,3.508563011,1]]

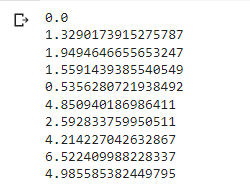
row0 = dataset[0]

for row in dataset:

distance = euclidean\_distance(row0, row)

print(distance)

**Output:**



from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

# Loading data

irisData = load\_iris()

print(irisData)

# Create feature and target arrays

X = irisData.data

y = irisData.target

# Split into training and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size = 0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train)

# Calculate the accuracy of the model

print(knn.score(X\_test, y\_test))

**Experiment No.: 6**

**Aim**

Implement Naive Bayes Algorithm using data set

**Procedure**

import pandas as pd

import numpy as np

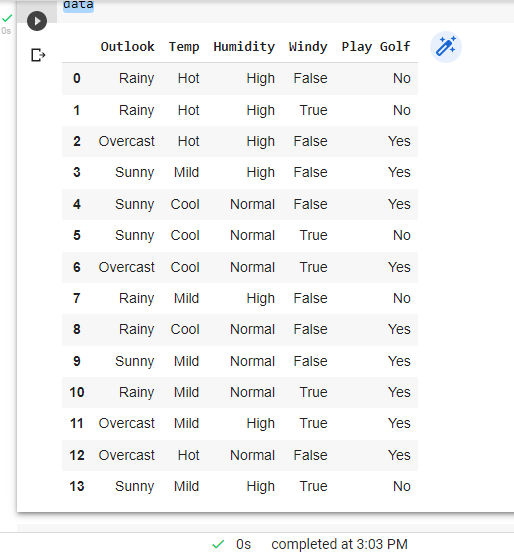
from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

data= pd.read\_csv('golf-dataset.csv')

data

**Output:**

****

label\_encoder = preprocessing.LabelEncoder()

data['Outlook']= label\_encoder.fit\_transform(data['Outlook'])

data['Temp']= label\_encoder.fit\_transform(data['Temp'])

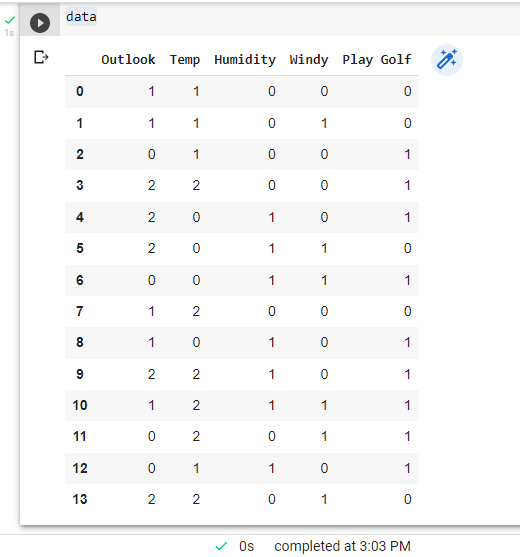
data['Humidity']= label\_encoder.fit\_transform(data['Humidity'])

data['Windy']= label\_encoder.fit\_transform(data['Windy'])

data['Play Golf']= label\_encoder.fit\_transform(data['Play Golf'])

data

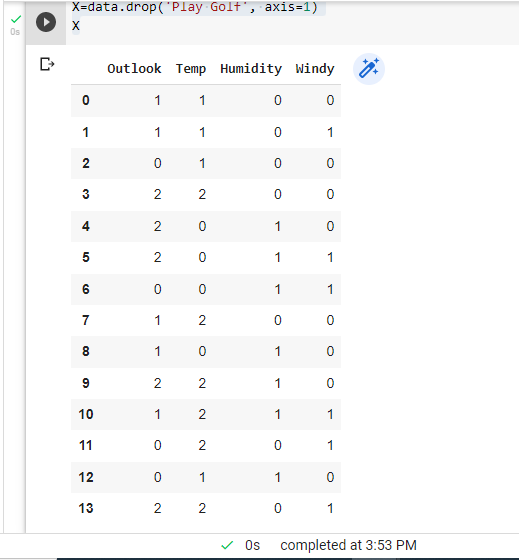
**Output:**

****

X=data.drop('Play Golf', axis=1)

X

**Output:**

****

X=data.drop('Play Golf', axis=1)

Y=data['Play Golf']

Y

**Output:**

****

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=42)

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred = gnb.predict(X\_test)

from sklearn import metrics

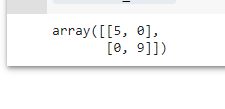
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

from sklearn import metrics

confusion\_matrix = metrics.confusion\_matrix(Y, Y)

confusion\_matrix

**Output:**





**Experiment No.: 7**

**Aim**

Implement Decision Tree using inbuilt function

**Procedure**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

data = load\_iris()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['target'] = data.target

# Step 1: Import the model you want to use

# This was already imported earlier in the notebook so commenting out

# from sklearn.tree import DecisionTreeClassifier

# Step 2: Make an instance of the Model

clf = DecisionTreeClassifier(max\_depth=2,

random\_state=0)

# Step 3: Train the model on the data

clf.fit(X\_train, Y\_train)

# Step 4: Predict labels of unseen (test) data

# Not doing this step in the tutorial

clf.predict(X\_test)

# tree.plot\_tree(clf);

fn = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

cn = ['setosa', 'versicolor', 'virginica']

# fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4, 4), dpi=300)

tree.plot\_tree(clf,

feature\_names=fn,

class\_names=cn,

filled=True

)

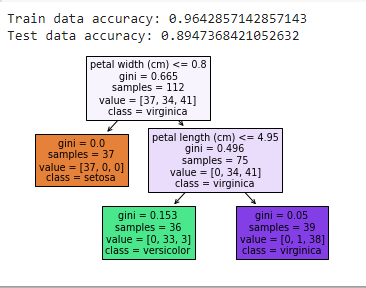
y\_pred = clf.predict(X\_test)

print("Train data accuracy:",accuracy\_score(y\_true = Y\_train, y\_pred=clf.predict(X\_train)))

print("Test data accuracy:",accuracy\_score(y\_true = Y\_test, y\_pred=y\_pred))

plt.show()

**Output**





**Experiment No.: 8**

**Aim**

Implement Linear Regression using built-in functions.

**Procedure**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

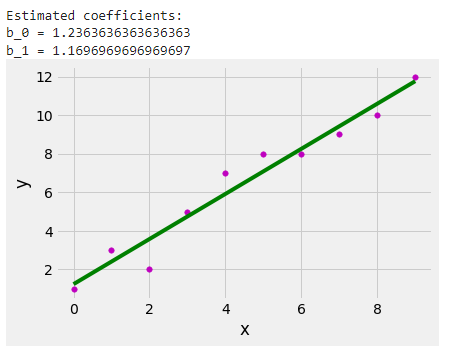
# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output**





**Experiment No.: 9**

**Aim:** Implementation of Multiple Linear Regression

**Procedure**

import numpy as np

from sklearn.linear\_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]

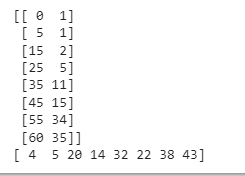
y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

print(x)

print(y)

**Output**

****

model = LinearRegression().fit(x, y)

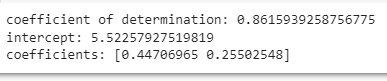
r\_sq = model.score(x, y)

print(f"coefficient of determination: {r\_sq}")

print(f"intercept: {model.intercept\_}")

print(f"coefficients: {model.coef\_}")

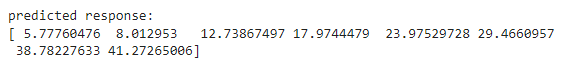
**Output**

****

y\_pred = model.predict(x)

print(f"predicted response:\n{y\_pred}")

**Output**

****

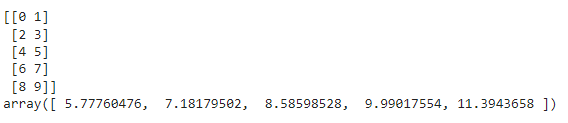
x\_new = np.arange(10).reshape((-1, 2))

print(x\_new)

y\_new = model.predict(x\_new)

y\_new

**Output**

****



**Experiment No.: 10**

**Aim**

Implement K means Clustering

**Procedure**

#finding optimal number of clusters using the elbow method

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

from sklearn.cluster import KMeans

wcss\_list= [] #Initializing the list for the values of WCSS

dataset = pd.read\_csv('iris.csv')

x = dataset.iloc[:, [3, 4]].values

#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state= 42)

kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

#visulaizing the clusters

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster

mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

mtp.scatter(x[y\_predict== 2, 0], x[y\_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

mtp.scatter(x[y\_predict == 3, 0], x[y\_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster

mtp.scatter(x[y\_predict == 4, 0], x[y\_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

mtp.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroid')

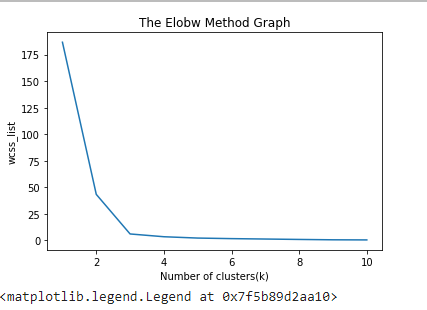
mtp.title('Clusters of customers')

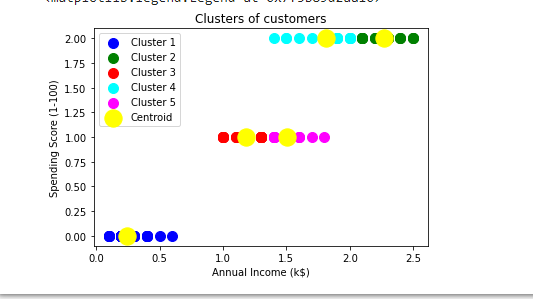
mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

**Output**

****

****



**Experiment No.: 11**

**Aim**

Implement KMeans Clustering algorithm using a dataset.

**Procedure**

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

import pandas as pd

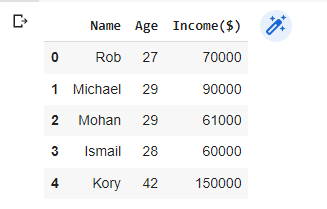
from matplotlib import pyplot as plt

%matplotlib inline

df = pd.read\_csv('income.csv')

df.head()

**Output**



scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])

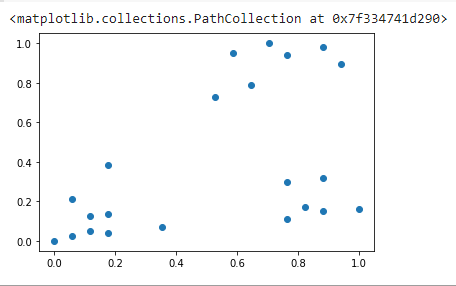
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])

df['Age'] = scaler.transform(df[['Age']])

plt.scatter(df.Age, df['Income($)'])

**Output**



km = KMeans(n\_clusters=3)

y\_predicted = km.fit\_predict(df[['Age', 'Income($)']])

y\_predicted

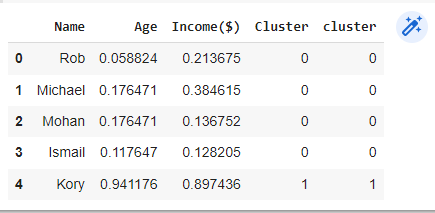
**Output**



df['cluster'] = y\_predicted

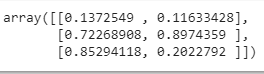
df.head()

**Output**



km.cluster\_centers\_

**Output**



df1 = df[df.cluster==0]

df2 = df[df.cluster==1]

df3 = df[df.cluster==2]

plt.scatter(df1.Age, df1['Income($)'], color = 'green')

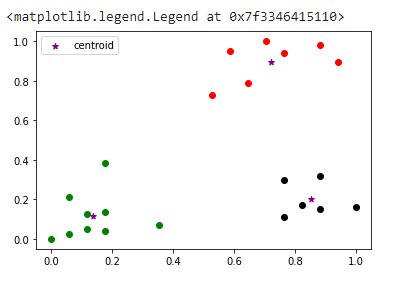
plt.scatter(df2.Age, df2['Income($)'], color = 'red')

plt.scatter(df3.Age, df3['Income($)'], color = 'black')

plt.scatter(km.cluster\_centers\_[:, 0], km.cluster\_centers\_[:, 1], color='purple', marker = '\*', label = 'centroid')

plt.legend()

**Output**

****



**Experiment No.: 12**

**Aim**

Implement Ngrams algorithm

**Procedure**

import nltk

from nltk.util import ngrams

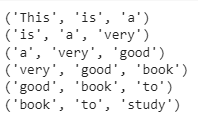
text = "This is a very good book to study";

Ngrams = ngrams(sequence = nltk.wordpunct\_tokenize(text), n=3)

for grams in Ngrams:

print(grams)

**Output**



import requests

from bs4 import BeautifulSoup

import csv

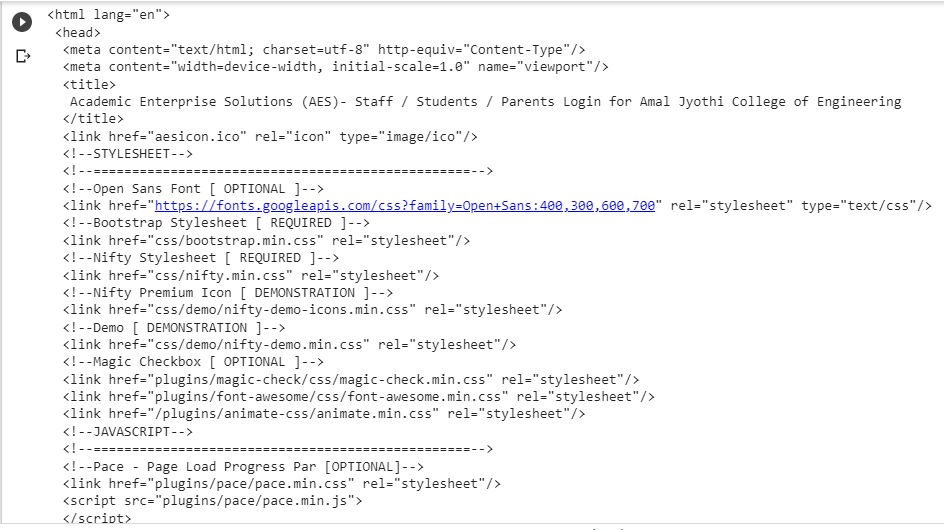
URL = "https://www.aesajce.in/"

r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')

print(soup.prettify())

**Output**

****



**Experiment No.: 13**

**Aim**

**Procedure**

**Output**



**Experiment No.: 14**

**Aim**

**Procedure**

**Output**



**Experiment No.: 15**

**Aim**

**Procedure**

**Output**



**Experiment No.: 16**

**Aim**

**Procedure**

**Output**



**Experiment No.: 17**

**Aim**

**Procedure**

**Output**



**Experiment No.: 18**

**Aim**

**Procedure**

**Output**



**Experiment No.: 19**

**Aim**

**Procedure**

**Output**



**Experiment No.: 20**

**Aim**

**Procedure**

**Output**