**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])   # Creating the 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                  # Changing an element of the array

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

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

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

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

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

print("All zeros matrix:\n  " ,a)

b = np.ones((1,2))    # Createing the array of all ones

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

d = np.eye(2)        # Creating the 2x2 identity matrix

print("\n identity matrix: \n",d)

e = np.random.random((2,2))  # Creating the 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.

**Histogram**

import matplotlib.pyplot as plt

import numpy as np

# Use numpy to generate a bunch of random data in a bell curve around 5.

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)

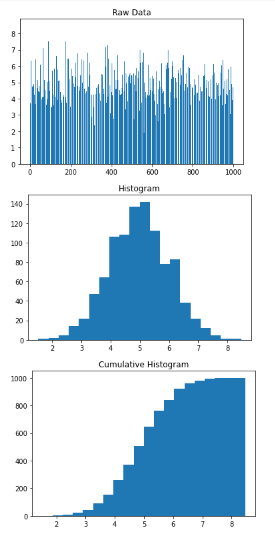
plt.title("Histogram")

plt.show()

plt.hist(n, cumulative=True, bins=20)

plt.title("Cumulative Histogram")

plt.show()



**Distribution Chart**

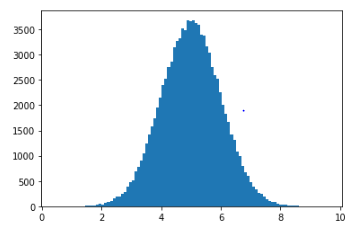
import numpy

import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)

plt.show()



**Bubble Chart**

import matplotlib.pyplot as plt

import numpy as np

# create data

x = np.random.rand(40)

y = np.random.rand(40)

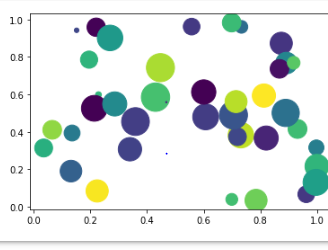
z = np.random.rand(40)

colors = np.random.rand(40)

# use the scatter function

plt.scatter(x, y, s=z\*1000,c=colors)

plt.show()



**Scatter Plot**

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]

# Markers: https://matplotlib.org/api/markers\_api.html

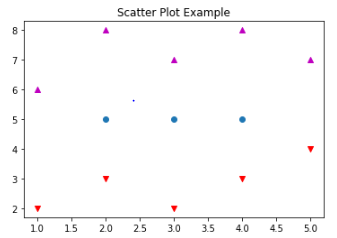
plt.scatter(x1, y1)

plt.scatter(x2, y2, marker='v', color='r')

plt.scatter(x2, y3, marker='^', color='m')

plt.title('Scatter Plot Example')

plt.show()



**Line graph**

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()

**output**



**Bar chart**

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='y')

plt.bar(x2, y2, label="Green Bar", color='r')

plt.plot()

plt.xlabel("bar number")

plt.ylabel("bar height")

plt.title("Bar Chart Example")

plt.legend()

plt.show()

**output**



**Box plot**

plt.figure()

plt.suptitle("Boxplot for X vs Y split into 5 bins")

ax = plt.gca()

df2.boxplot(showmeans=True)

# Rotate x axis text values

for tick in ax.get\_xticklabels():

    tick.set\_rotation(30)

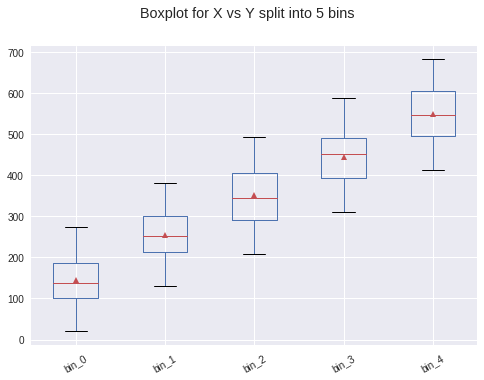
print("\nIn the boxplot below, the box extends from the lower to upper quartile values of the data, with a line at the median.\n \

The whiskers extend from the box to show the range of the data. The triangle indicates the mean value.\n")

**output**

In the boxplot below, the box extends from the lower to upper quartile values of the data, with a line at the median.

 The whiskers extend from the box to show the range of the data. The triangle indicates the mean value.



**Experiment No.: 3**

**Aim**

Programs to handle data using pandas.

**Questions**

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

**Question**

**Q1 - Pandas  Series**

1. How to create Series with nd array
2. How to create Series with Mutable index
3. Creating a series from a Dictionary
4. Print all the values of the Series by multiplying them by 2.
5. Print Square of all the values of the series.
6. Print all the values of the Series that are greater than2
7. Addition of two series
8. Print the first and last 5 elements of a series
9. Print the values from index 0 to 5
10. Selection Using loc, iloc index label
11. Retrieve subsets of data using slicing

**Q2 Dataframe**

1. create Dataframe From Series
2. DataFrame from List of Dictionaries
3. Display the first 5 rows of data frame
4. Select the last two columns of the data frame
5. Add two data frames
6. Demonstrate deletion, and renaming of columns
7. Demonstrate concat, Merge operations in data frame
8. Write a Pandas program to join the two given dataframes along rows and assign all data

**Test Data:**

student\_data1:

  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\_data2:

  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

**Program and Output**

import pandas as pd

d={"name":"Rohit sharma","Role":"captain","Team":"MUmbai","score":50}

s=pd.Series(d)

print(s)

name Rohit sharma

Role captain

Team MUmbai

score 50

import pandas as pd

s=pd.Series([1,2,3,4,5,])

print("multiply the series of numbers with 2")

print("-----------------------------------------------------")

print(s\*2)

print("Finding square of series of numbers")

print("-----------------------------------------------------")

print(s\*\*2)

print("Finding the series of numbers which is greater than 2")

print("-----------------------------------------------------")

print(s[s>2])

multiply the series of numbers with 2

-----------------------------------------------------

0 2

1 4

2 6

3 8

4 10

dtype: int64

Finding square of series of numbers

-----------------------------------------------------

0 1

1 4

2 9

3 16

4 25

dtype: int64

Finding the series of numbers which is greater than 2

-----------------------------------------------------

2 3

3 4

4 5

import pandas as pd

s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])

s2=pd.Series([9,8,7,6,5],index=['a','b','c','d','e'])

s3=pd.Series([55,33,22,11],index=['a','b','c','d'])

print("To add series1 and series2")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print(s1+s2)

print("To add series2 and series3")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print(s2+s3)

print("To add series1 and series2 and Fill the non matching index")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print(s1.add(s3,fill\_value=0))

To add series1 and series2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a 10

b 10

c 10

d 10

e 10

dtype: int64

To add series2 and series3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a 64.0

b 41.0

c 29.0

d 17.0

e NaN

dtype: float64

To add series1 and series2 and Fill the non matching index

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a 56.0

b 35.0

c 25.0

d 15.0

e 5.0

import pandas as pd

import numpy as np

arr=np.array([1,2,3,4,5,6])

s=pd.Series(arr)

print(s.head())

print(s.head(3))

0 1

1 2

2 3

3 4

4 5

0 1

1 2

2 3

import pandas as pd

import numpy as np

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

s=pd.Series(arr)

print(s.tail())

print(s.tail(3))

3 4.0

4 5.0

5 6.0

6 7.0

7 0.8

5 6.0

6 7.0

7 0.8

import pandas as pd

import numpy as np

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

s=pd.Series(arr)

print(s)

print(s.loc[:3])

print(s.loc[2:4])

s.loc[1:4]

0 1.0

1 2.0

2 3.0

3 4.0

4 5.0

5 6.0

6 7.0

7 0.8

2 3.0

3 4.0

4 5.0

1 2.0

2 3.0

3 4.0

4 5.0

import pandas as pd

import numpy as np

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

s=pd.Series(arr)

print(s)

print(s.loc[:3])

print(s.iloc[2:4])

s.iloc[1:4]

0 1.0

1 2.0

2 3.0

3 4.0

4 5.0

5 6.0

6 7.0

7 0.8

0 1.0

1 2.0

2 3.0

3 4.0

2 3.0

3 4.0

1 2.0

2 3.0

3 4.0

import pandas as pd

import numpy as np

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

s=pd.Series(arr)

print(s[1])

print("\n")

print(s[:3])

print(s.iloc[2:4])

s[1:4]

2.0

0 1.0

1 2.0

2 3.0

2 3.0

3 4.0

1 2.0

2 3.0

3 4.0

import pandas as pd

import numpy as np

arr=np.array([1,2,3,4,5,6])

s=pd.Series(arr,index=["one","two","Three","Four","Fifth","sixth"])

print(s)

print("THe index in series are")

print(s.index)

one 1

two 2

Three 3

Four 4

Fifth 5

sixth 6

dtype: int64

THe index in series are

Index(['one', 'two', 'Three', 'Four', 'Fifth', 'sixth'], dtype='object')

import pandas as pd

import numpy as np

arr=np.array([1,2,3,4,5,6])

s=pd.Series(arr,index=['A','B','C','D','E','F'])

print(s)

print(s[1:5:2])

print(s[0:6:2])

A 1

B 2

C 3

D 4

E 5

F 6

B 2

D 4

A 1

C 3

E 5

import pandas as pd

name=pd.Series(['hardik','rohit'])

team=pd.Series(['lions','mumbai'])

dic={'Name':name,'Team':team}

pf=pd.DataFrame(dic)

print(pf)

Name Team

0 hardik lions

1 rohit mumbai

import pandas as pd

dic1=[{'Fname':'Rohit','Lname':'Sharma'},

      {'Fname':'Virat','Lname':'Kholi'},

      {'Fname':'Hardik','Lname':'Pandya'}]

df1=pd.DataFrame(dic1)

print(df1)

Fname Lname

0 Rohit Sharma

1 Virat Kholi

2 Hardik Pandya

import pandas as pd

g=[{'Fname':'Rohit','Lname':'Sharma'},

   {'Fname':'Virat','Lname':'Kholi'},

   {'Fname':'Hardik','Lname':'Pandya'}]

GG=pd.DataFrame(g)

print(GG)

for (row\_index,row\_value) in GG.iteritems():

     print("\n Row index is",row\_index)

     print("Column index")

     print(row\_value)

Fname Lname

0 Rohit Sharma

1 Virat Kholi

2 Hardik Pandya

Row index is Fname

Column index

0 Rohit

1 Virat

2 Hardik

Name: Fname, dtype: object

Row index is Lname

Column index

0 Sharma

1 Kholi

2 Pandya

Name: Lname, dtype: object

import pandas as pd

dic={'empdate':['1995-05-22','1996-02-12','1994-06-02','1999-07-16','1998-07-16','2000-06-15','2004-02-19','1994-05-08'],

      'empid':[66,55,66,77,88,32,15,18],

      'empname':['vishnu','ram','don','thomas','manu','vinay','vijay','akhil']

     }

pf=pd.DataFrame(dic)

print(pf)

print(pf.head())

print(pf.tail())

empdate empid empname

0 1995-05-22 66 vishnu

1 1996-02-12 55 ram

2 1994-06-02 66 don

3 1999-07-16 77 thomas

4 1998-07-16 88 manu

5 2000-06-15 32 vinay

6 2004-02-19 15 vijay

7 1994-05-08 18 akhil

empdate empid empname

0 1995-05-22 66 vishnu

1 1996-02-12 55 ram

2 1994-06-02 66 don

3 1999-07-16 77 thomas

4 1998-07-16 88 manu

empdate empid empname

3 1999-07-16 77 thomas

4 1998-07-16 88 manu

5 2000-06-15 32 vinay

6 2004-02-19 15 vijay

7 1994-05-08 18 akhil

import pandas as pd

dic={'empdate':['1995-05-22','1996-02-12','1994-06-02','1999-07-16','1998-07-16','2000-06-15','2004-02-19','1994-05-08'],

      'empid':[66,55,66,77,88,32,15,18],

      'empname':['vishnu','ram','don','thomas','manu','vinay','vijay','akhil']

     }

pf=pd.DataFrame(dic)

print(pf)

pf[['empid','empname']]

import pandas as pd

dic={'empdate':['1995-05-22','1996-02-12','1994-06-02','1999-07-16','1998-07-16','2000-06-15','2004-02-19','1994-05-08'],

      'empid':[66,55,66,77,88,32,15,18],

      'empname':['vishnu','ram','don','thomas','manu','vinay','vijay','akhil']

     }

pf=pd.DataFrame(dic)

print(pf)

N = 2

# Select last N columns of dataframe

last\_n\_column  = pf.iloc[: , -N:]

print(last\_n\_column)

empdate empid empname

0 1995-05-22 66 vishnu

1 1996-02-12 55 ram

2 1994-06-02 66 don

3 1999-07-16 77 thomas

4 1998-07-16 88 manu

5 2000-06-15 32 vinay

6 2004-02-19 15 vijay

7 1994-05-08 18 akhil

empid empname

0 66 vishnu

1 55 ram

2 66 don

3 77 thomas

4 88 manu

5 32 vinay

6 15 vijay

7 18 akhil

import pandas as pd

s1=pd.Series([1,2,3,4,5])

df=pd.DataFrame(s1)

df.columns=['List1']

df['List2']=20

df['List3']=df['List1']+df['List2']

print(df)

List1 List2 List3

0 1 20 21

1 2 20 22

2 3 20 23

3 4 20 24

4 5 20 25

import pandas as pd

s1=pd.Series([1,2,3,4,5])

df=pd.DataFrame(s1)

df.columns=['List1']

df['List2']=20

print(df)

df.pop('List2')

List1 List2

0 1 20

1 2 20

2 3 20

3 4 20

4 5 20

0 20

1 20

2 20

3 20

4 20

import pandas as pd

dict1={'id':['1','2','3','4','5'],'value1':['a','b','c','d','e'],'value2':['f','g','h','i','j']}

dict2={'id':['2','3','6','7','8'],'value1':['k','m','o','q','s'],'value2':['l','n','p','r','t']}

dict3={'id':['1','2','3','4','5','7','8','9','10','11'],'value3':['12','13','14','15','16','17','15','12','13','23']}

df1=pd.DataFrame(dict1)

df2=pd.DataFrame(dict2)

print('data frame 1\n')

print(df1)

print('data frame 2\n')

print(df2)

df3=pd.concat([df1,df2])  #concatenation

df4=pd.DataFrame(dict3)

print('after concatenation: \n')

print(df4)

df5=pd.merge(df3,df4,on='id')    #merge

print('after merging\n')

print(df5)

id value1 value2

0 1 a f

1 2 b g

2 3 c h

3 4 d i

4 5 e j

data frame 2

id value1 value2

0 2 k l

1 3 m n

2 6 o p

3 7 q r

4 8 s t

after concatenation:

id value3

0 1 12

1 2 13

2 3 14

3 4 15

4 5 16

5 7 17

6 8 15

7 9 12

8 10 13

9 11 23

after merging

id value1 value2 value3

0 1 a f 12

1 2 b g 13

2 2 k l 13

3 3 c h 14

4 3 m n 14

5 4 d i 15

6 5 e j 16

7 7 q r 17

8 8 s t 15

import pandas as pd

student\_data1 = 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]})

student\_data2 = 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(student\_data1)

print("-------------------------------------")

print(student\_data2)

print("\nJoin the said two dataframes along columns:")

result\_data = pd.concat([student\_data1, student\_data2], axis = 1)

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

Join the said two dataframes along columns:

student\_id name marks student\_id name marks

0 S1 Danniella Fenton 200 S4 Scarlette Fisher 201

1 S2 Ryder Storey 210 S5 Carla Williamson 200

2 S3 Bryce Jensen 190 S6 Dante Morse 198

3 S4 Ed Bernal 222 S7 Kaiser William 219

4 S5 Kwame Morin 199 S8 Madeeha Preston 201

**Experiment No.: 4**

**Aim**

Perform Z-score normalization, Min-max normalization

**Procedure and Output**

1. Z-score Normalization

   import pandas as pd

import numpy as np

import scipy.stats as stats

data = np.array([6, 7, 7, 12, 13, 13, 15, 16, 19, 22])

print("\n Data before aplying z-score operation\n",data)     # z-score normalization

new\_data=stats.zscore(data)

print("Normalized Data are:\n",new\_data)

**Output**

Data before aplying z-score operation

 [ 6  7  7 12 13 13 15 16 19 22]

Normalized Data are:

 [-1.39443338 -1.19522861 -1.19522861 -0.19920477  0. 0.

  0.39840954  0.5976143   1.19522861  1.79284291]

2. Min-max 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("\n before normalization\n",data)

# define min max scaler             #min max normalization's another example

scaler = MinMaxScaler()

# transform data

print("\n After applying transformation")

scaled = scaler.fit\_transform(data)

print(scaled)

Output

 before normalization

 [[1.0e+02 1.0e-03]

 [8.0e+00 5.0e-02]

 [5.0e+01 5.0e-03]

 [8.8e+01 7.0e-02]

 [4.0e+00 1.0e-01]]

 After applying transformation

[[1.         0.        ]

 [0.04166667 0.49494949]

 [0.47916667 0.04040404]

 [0.875      0.6969697 ]

 [0.         1.        ]]

**Experiment No.: 5**

**Aim**

 Implement KNN Algorithm using python

**Procedure**

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)

# Locate the most similar neighbors

def get\_neighbors(train, test\_row, num\_neighbors):

distances = list()

for train\_row in train:

dist = euclidean\_distance(test\_row, train\_row)

distances.append((train\_row, dist))

distances.sort(key=lambda tup: tup[1])

neighbors = list()

for i in range(num\_neighbors):

neighbors.append(distances[i][0])

return neighbors

# Make a classification prediction with neighbors

def predict\_classification(train, test\_row, num\_neighbors):

neighbors = get\_neighbors(train, test\_row, num\_neighbors)

output\_values = [row[-1] for row in neighbors]

prediction = max(set(output\_values), key=output\_values.count)

return prediction

# 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]]

prediction = predict\_classification(dataset, dataset[0], 3)

print('Expected %d, Got %d.' % (dataset[0][-1], prediction))

**Output**

Expected 0, Got 0.

**Experiment No.: 6**

**Aim**

Gaussian Naive Bayes classifier

**Procedure**

# load the iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target

# splitting X and y into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

# training the model on training set

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

# making predictions on the testing set

y\_pred = gnb.predict(X\_test)

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

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

**Output**

Gaussian Naive Bayes model accuracy(in %): 95.0