**Experiment No.: 1**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:1/9/22**

**Aim**

Programs to handle data using pandas**.**

**Program and Output**

**1.How to create Series with nd array**

import pandas as pd

import numpy as np

arr=np.array([10,15,18,22])

s = pd.Series(arr)

print(s)

**out put**

0 10

1 15

2 18

3 22

dtype: int64

**2.How to create Series with Mutable index**

import pandas as pd

import numpy as np

arr=np.array(['a','b','c','d'])

s=pd.Series(arr,

index=['first','second','third','fourth'])

print(s)

**out put**

first a

second b

third c

fourth d

dtype: object

**3.Creating a series from a Dictionary**

import pandas as pd

s={'name':'hardik','iplteam':'mi','runs':100}

p=pd.Series(s)

print(p)

**out put**

name hardik

iplteam mi

runs 100

dtype: object

4. Print all the values of the Series by multiplying them by 2

import pandas as pd

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

print(p)

print("multlipling all values in series by 2")

print(p\*2)

**out put**

0 1

1 2

2 3

3 4

4 5

dtype: int64

multlipling all values in series by 2

0 2

1 4

2 6

3 8

4 10

dtype: int64

**5.Print Square of all the values of the series**

import pandas as pd

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

print('..............................................')

print("square of all values")

print(p\*\*2)

')

**out put**

0 1

1 2

2 3

3 4

4 5

dtype: int64

square of all values

0 1

1 4

2 9

3 16

4 25

dtype: int64

**6 Print all the values of the Series that are greater than2**

import pandas as pd

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

print("when the value greater than 2")

print(p[p>2])

print('..............................................')

**out put**

when the value greater than 2

2 3

3 4

4 5

dtype: int64

**7.Addition of two series**

import pandas as pd

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

s2=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])

print(s1)

print(s2)

print(s1+s2)

**out put**

a 1

b 2

c 3

d 4

e 5

dtype: int64

a 1

b 2

c 3

d 4

e 5

dtype: int64

a 2

b 4

c 6

d 8

e 10

dtype: int64

**8. Print the first and last 5 elements of a series**

import pandas as pd

import numpy as np

arr=np.array([10,12,23,3,4,56,57,6,7])

s=pd.Series(arr)

print(s.head(5))

**out put**

0 10

1 12

2 23

3 3

4 4

dtype: int64

**9. Print the values from index 0 to 5**

import pandas as pd

import numpy as np

arr=np.array([10,12,23,3,4,56,57,6,7])

s=pd.Series(arr)

print(s.head(6))

**out put**

0 10

1 12

2 23

3 3

4 4

5 56

dtype: int64

**10.Selection Using loc, iloc index label**

import pandas as pd

import numpy as np

arr=np.array([10,12,23,3,4,56,57,6,7])

s=pd.Series(arr)

print(s)

print(s.loc[:2])

print(s.iloc[3:4])

**out put**

0 10

1 12

2 23

3 3

4 4

5 56

6 57

7 6

8 7

dtype: int64

0 10

1 12

2 23

dtype: int64

3 3

dtype: int64

**11.Retrieve subsets of data using slicing**

import pandas as pd

import numpy as np

arr=np.array([10,12,23,3,4])

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

print(s)

print(s[::-1])

**out put**

A 10

B 12

C 23

D 3

E 4

dtype: int64

E 4

D 3

C 23

B 12

A 10

dtype: int64

**Experiment No.: 2**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:1/9/22**

**Aim**

Dataframe

**1.create Dataframe From Series**

import pandas as pd

s = pd.Series(['a','b','c','d'])

df=pd.DataFrame(s)

print(df)

**out put**

0

0 a

1 b

2 c

3 d

**2 DataFrame from List of Dictionaries**

import pandas as pd

l=[{'Name':'sachin','city':'kerala'},

   {'Name':'virat','city':'tamilnadu'}]

d=pd.DataFrame(l)

print(d)

**out put**

Name city

0 sachin kerala

1. virat tamilnadu

**3.Display the first 5 rows of data frame**

import pandas as pd

empdata = {'empid':[1,2,3,4,5,6],'ename':['Vimal','Sachin','Bav','Kumar','Ravy','Sunil']}

df=pd.DataFrame(empdata)

print(df)

print(df.head(5))

**out put**

empid ename

0 1 Vimal

1 2 Sachin

2 3 Bav

3 4 Kumar

4 5 Ravy

5 6 Sunil

empid ename

0 1 Vimal

1 2 Sachin

2 3 Bav

3 4 Kumar

1. 5 Ravy

**4.Select the last two columns of the data frame**

import pandas as pd

empdata = {'empid':[1,2,3,4,5,6], 'ename':['Vimal','Sachin','Bav','Kumar','Ravy','Sunil']}

df=pd.DataFrame(empdata)

print(df)

df.loc[0:5]

print(df.tail(2))

**out put**

empid ename

0 1 Vimal

1 2 Sachin

2 3 Bav

3 4 Kumar

4 5 Ravy

5 6 Sunil

empid ename

4 5 Ravy

5 6 Sunil

**6. Demonstrate deletion, and renaming of columns**

import pandas as pd dic1= {'id':['1','2','3','4','5'],'value1':['A','C','E','G','I'],'value2':['B','D','F','H','J']} dic2= {'id':['2','3','6','7','8'],'value1':['K','M','O','Q','S'],'value2':['L','N','P','R','T']} dic3= {'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(dic1) df2=pd.DataFrame(dic2) df3=pd.concat([df1,df2]) df4=pd.DataFrame(dic3) df5=pd.merge(df3,df4,on='id') print(df5)

**out put**

id value1 value2 value3 0 1 A B 12 1 2 C D 13 2 2 K L 13 3 3 E F 14 4 3 M N 14 5 4 G H 15 6 5 I J 16 7 7 Q R 17 8 8 S T 15

**7 Demonstrate concat, Merge operations in data frame**

import pandas as pd

s= pd.Series([10,20,30,40])

df=pd.DataFrame(s)

df.columns=['List1']

df['List2']=40

df1=df.drop('List2',axis=1)

df2=df.drop(index=[2,3],axis=0)

print(df)

print(" After deletion::")

print(df1)

print (" After row deletion::")

print(df2)

**out put**

List1 List2

0 10 40

1 20 40

2 30 40

3 40 40

After deletion::

List1

0 10

1 20

2 30

3 40

After row deletion::

List1 List2

0 10 40

1. 20 40

**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

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 rows:")

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

print(result\_data)

**Out put**

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 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

**Experiment No.: 3**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:1/9/22**

**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**

**Identity matrix and random matrix**

import numpy as np

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

print("type: %s" %type(a))

print("shape: %s" %a.shape)

print(a[0], a[1], a[2])

a[0] = 5

print(a)

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

print("\n shape of b:",b.shape)

print(b[0, 0], b[0, 1], b[1, 0])

a = np.zeros((2,2))

print("All zeros matrix:\n %s" %a)

b = np.ones((1,2))

print("\nAll ones matrix:\n %s" %b)

d = np.eye(2)

print("\n identity matrix: \n%s"%d)

e = np.random.random((2,2))

print("\n random matrix: \n%s"%e)

**Output**

type: <class 'numpy.ndarray'>

shape: 3

1 2 3

[5 2 3]

shape of b: (2, 3)

1 2 4

All zeros matrix:

[[0. 0.]

[0. 0.]]

All ones matrix:

[[1. 1.]]

identity matrix:

[[1. 0.]

[0. 1.]]

random matrix:

[[0.8972642 0.58584636]

[0.05954874 0.71295688]]

**Outer product, Dot product, vectorized sum**

import numpy as np

a = np.arange(4)

b = np.arange(4)

print(a)

print(b)

dp = np.dot(a,b)

print("Dot product: %s\n" %dp)

op = np.outer(a,b)

print("\n Outer product: %s\n" %op)

ep = np.multiply(a, b)

print("\n Element Wise product: %s \n" %ep)

print("Vectorized sum example\n")

x = np.array([[1,2],[3,4]])

print("x:\n %s" %x)

print("sum: %s"%np.sum(x))

print("sum axis = 0: %s" %np.sum(x, axis=0))

print(" sum axis = 1: %s" %np.sum(x, axis=1))

**Output**

[0 1 2 3]

[0 1 2 3]

Dot product: 14

Outer product: [[0 0 0 0]

[0 1 2 3]

[0 2 4 6]

[0 3 6 9]]

Element Wise product: [0 1 4 9]

Vectorized sum example

x:

[[1 2]

[3 4]]

sum: 10

sum axis = 0: [4 6]

sum axis = 1: [3 7]

**Transpose**

import numpy as np

x = np.array([[1,2], [3,4]])

print("Original x: \n%s " %x)

print("\nTranspose of x: \n%s" %x.T)

**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

A = array([[1, 1], [1, 1], [1, 1]])

print("A: \n%s" %A)

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.: 4**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:1/9/22**

**Aim**

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

**Program and Output**

**1.Bubble chart**

import matplotlib.pyplot as plt

plt.style.use('seaborn-whitegrid')

import numpy as np

rng = np.random.RandomState(0)

x = rng.randn(100)

y = rng.randn(100)

colors = rng.rand(100)

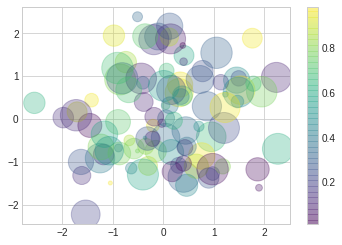
sizes = 1000 \* rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,

            cmap='viridis')

plt.colorbar();  # show color scale

**output**



**2.Distribution chart**

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

num\_points = 20

# x will be 5, 6, 7... but also twiddled randomly

x = 5 + np.arange(num\_points) + np.random.randn(num\_points)

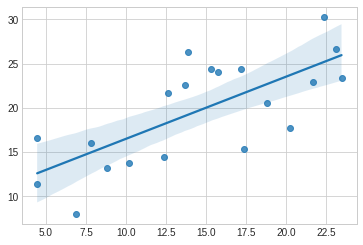
# y will be 10, 11, 12... but twiddled even more randomly

y = 10 + np.arange(num\_points) + 5 \* np.random.randn(num\_points)

sns.regplot(x, y)

plt.show()

**output**



**3.Raw Data, Histogram and Cumulative Histogram**

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")

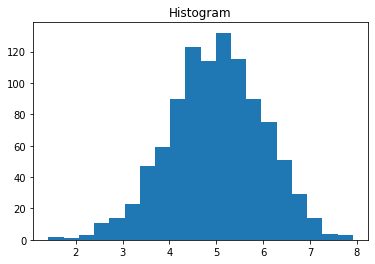
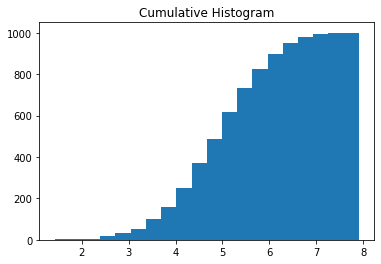
plt.show()

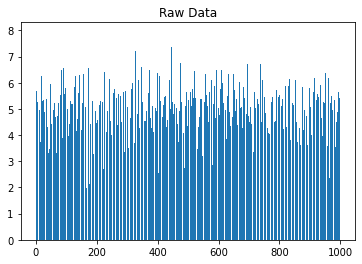
plt.hist(n, cumulative=True, bins=20)

plt.title("Cumulative Histogram")

plt.show()

**output**





**4.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]

plt.scatter(x1, y1)

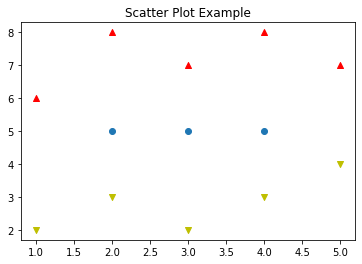
plt.scatter(x2, y2, marker='v', color='y')

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

plt.title('Scatter Plot Example')

plt.show()

**output**



**Experiment No.: 5**

**Name: JOBIN T J**

**Roll No:**

**Batch:MCA B**

**Date:10/9/22**

**Aim**

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

**Program and Output**

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

import numpy as np

X, y = load\_iris(return\_X\_y=True)

print(X)

scaler = MinMaxScaler()

scaler.fit(X)

X\_scaled = scaler.transform(X)

X\_scaled

**Output**

[[5.1 3.5 1.4 0.2]

 [4.9 3.  1.4 0.2]

 [4.7 3.2 1.3 0.2]

 [4.6 3.1 1.5 0.2]

 [5.  3.6 1.4 0.2]

 [5.4 3.9 1.7 0.4]

 [4.6 3.4 1.4 0.3]

 [5.  3.4 1.5 0.2]

 [4.4 2.9 1.4 0.2]

 [4.9 3.1 1.5 0.1]

 [5.4 3.7 1.5 0.2]

……………………..

**Graphical representation**

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1,2)

axes[0].scatter(X[:,0], X[:,1], c=y)

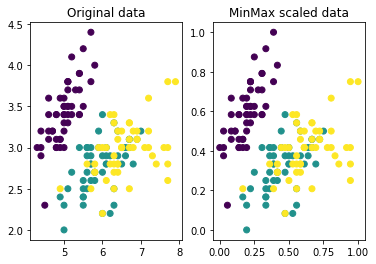
axes[0].set\_title("Original data")

axes[1].scatter(X\_scaled[:,0], X\_scaled[:,1], c=y)

axes[1].set\_title("MinMax scaled data")

plt.show()

**Output**



**Classifier**

from sklearn.neighbors import KNeighborsClassifier

model=KNeighborsClassifier(n\_neighbors=5,metric='minkowski')

model.fit(X\_train, y\_train)

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

model.score(X\_test, y\_test)

**Output**

1.0

**Experiment No.: 6**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:15/9/22**

**Aim**

Implement K-NN Algorithm using iris data set.

**Program and Output**

# Example of making predictions

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.: 7**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:19/9/22**

**Aim**

 Gaussian Naive Bayes classifier

**Program**

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

**Out put**

Gaussian Naive Bayes model accuracy(in %): 95.0

**Experiment No.: 8**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:13/10/22**

**Aim**

Decision tree

**Program and Output**

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

from sklearn import tree

from sklearn.datasets import load\_iris

data = load\_iris()

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

df['target'] = data.target

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df[data.feature\_names], df['target'], random\_state=0)

# 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**

Train data accuracy: 0.9642857142857143

Test data accuracy: 0.8947368421052632

**Experiment No.: 9**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:17/10/22**

**Aim**

Confusion Matrix

**Procedure**

import pandas as pd

import numpy as np

from sklearn import preprocessing

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

data

**Output**

****

**Procedure**

label\_encoder = preprocessing.LabelEncoder()

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

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

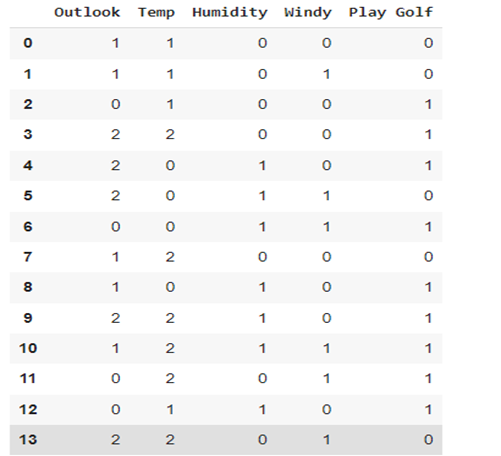
data['Humidity']= label\_encoder.fit\_transform(data['Humidity'])

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

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

data

**Output**

****

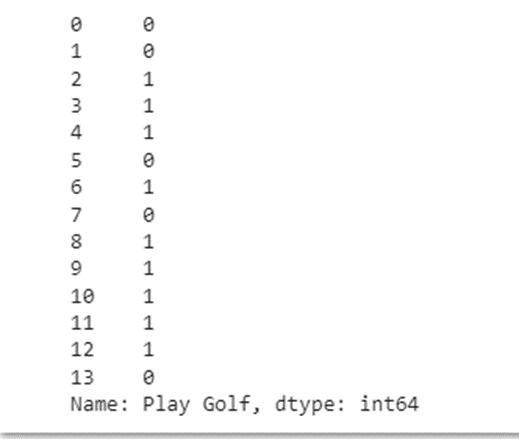
**Procedure**

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

Y=data['Play Golf']

Y

**Output**

****

**Procedure**

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

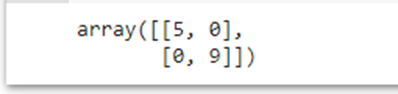
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

from sklearn import metrics

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

confusion\_matrix

**Output**



**Experiment No.: 10**

**Aim**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:27/10/22**

## Linear Regression

**Program and Output**

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\_\_":

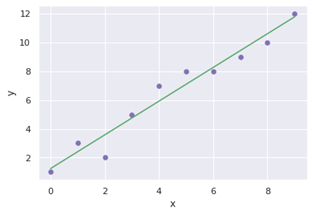
main()

**Output**

Estimated coefficients:

b\_0 = 1.2363636363636363

b\_1 = 1.1696969696969697



**Experiment No.: 11**

**Aim**

**Name: JOBIN T J**

**Roll No:7**

**Batch:MCA B**

**Date:27/10/22**

## Multiple Linear Regression

**Program and Output**

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)

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\_}")

y\_pred = model.predict(x)

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

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

print(x\_new)

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

y\_new

**Output**

[[ 0 1]

[ 5 1]

[15 2]

[25 5]

[35 11]

[45 15]

[55 34]

[60 35]]

[ 4 5 20 14 32 22 38 43]

coefficient of determination: 0.8615939258756775

intercept: 5.52257927519819

coefficients: [0.44706965 0.25502548]

predicted response:

[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

[[0 1]

[2 3]

[4 5]

[6 7]

[8 9]]

array([ 5.77760476, 7.18179502, 8.58598528, 9.99017554, 11.3943658 ])