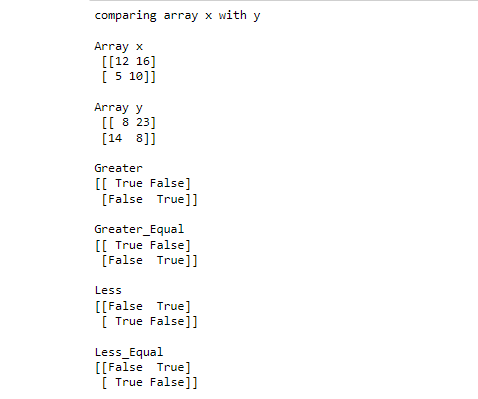
**CYCLE 1**

**Results and Observations**

****

## EXERCISE 1: INTRODUCTION TO NUMPY CO1

**ELEMENT-WISE COMPARISON OF ARRAYS**

**Program No: 3 Date**

**AIM:**Write a NumPy program to create an element-wise comparison(greater,

greater\_equal,less and less\_equal) of two given arrays.

**Theoretical Support**

**numpy.greater()**-To compare and return True if an array is greater than another array.

**numpy.greater\_equal()**-checks whether the elements in a given array (first argument) is

greater than or equal to a specified number(second argument).

**numpy.less(**)-function in Python is used to check, one by one, if the elements of the array x1 are less than the elements of another array x2 that is of the same shape.

**numpy.less\_equal()**-function is used to return the truth value of (x1 =< x2) element-wise.

**Code**

import numpy as np

x = np.array([[12,16],[5,10]])

y = np.array([[8,23],[14,8]])

print("comparing array x with y")

print("\nArray x\n",x)

print("\nArray y\n",y)

print("\nGreater")

print(np.greater(x, y))

print("\nGreater\_Equal")

print(np.greater\_equal(x, y))

print("\nLess")

print(np.less(x, y))

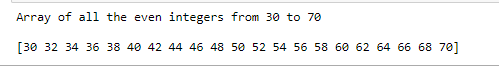
print("\nLess\_Equal")

print(np.less\_equal(x, y))

**Inference**:

numpy comparison functions helps to find the element-wise greater,greater than,lesser,lesser than values of two arrays.Standard mathematical functions for fast operations on entire arrays of data without having to write loops

**Results and Observations**

****

**ARRAY CREATION**

**Program No: 4 Date:**

# **AIM:**Write a NumPy program to create an array of all the even integers from 30 to 70.

**Theoretical Support**

**NumPy arange()** is one of the array creation routines based on numerical ranges. It creates an instance of ndarray with evenly spaced values and returns the reference to it.

**Code**

import numpy as np

x=np.arange(30,71,2)

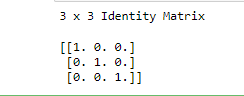
print("Array of all the even integers from 30 to 70\n")

print(x)

**Inference**

Array of even numbers are created easily using numpy.arrange().we can create array of any range of values according to the parameter specified inside the arrange() function.

**Results and Observations**

****

**IDENTITY MATRIX**

**Program No: 5 Date:**

# **AIM:**Write a NumPy program to create a 3x3 identity matrix

**Theoretical Support**

****numpy.identity(n, dtype = None)** :**Return a identity matrix i.e. a square matrix with ones on the main diagonal.

**Code**

import numpy as np

x=np.identity(5)

print('3 x 3 Identity Matrix\n')

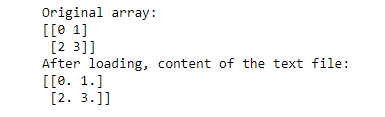
print(x)

**Inference**

identity matrix are created using identity() function according to the parameter.

Identity(5)-creates a matrix with 5 rows and 5 columns with main diagonal elements as 1 and other elements as 0.

**Results and Observations**



**ARRAY TO TEXT FILE**

**Program No: 6 Date:22-11-22**

# **AIM:**Write a NumPy program to save a given array to a text file and load it

**Theoretical Support**

**numpy savetxt** enables you to save a Numpy array to a text file.

Python **numpy loadtxt()** function is used to load the data from a text file and store them in a ndarray.

**Code**

import numpy as np

import os

x = np.arange(4).reshape(2,2)

print("Original array:")

print(x)

header = 'col1 col2'

np.savetxt('temp.txt', x, fmt="%d", header=header)

print("After loading")

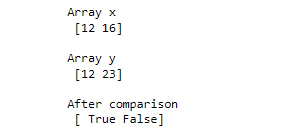
result = np.loadtxt('temp.txt')

print(result)

**Inference**

Read and wirte contents of a file without using normal file functions.

**Results and Observations**



**ARRAY EQUALITY**

**Program No: 7 Date:22-11-22**

# **AIM:**Write a NumPy program to check whether two arrays are equal (element wise) or not.

**Theoretical Support**

The **equal()** function is used to return (x1 == x2) element-wise. Input arrays of the same shape

**Code**

import numpy as np

x=np.array([12,16])

y=np.array([12,23])

print("Array x\n",x)

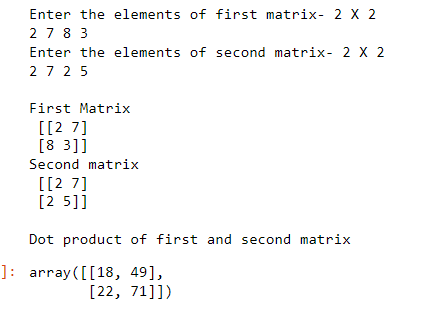
print("\nArray y\n",y)

print("\nAfter comparison\n",np.equal(x,y))

**Inference**

We can compare(element-wise) every elements of a array with another array without using loops.

**Results and Observations**



**EXERCISE 2:MATRIX OPERATIONS(USING VECTOIZATION) AND CO1 TRANSFORMATIONS**

**DOT PRODUCT OF MATRIX**

**Program No: 8 Date:22-11-22**

**AIM:**Write a python program to create two matrices(read values from user) and find the dot product.

**Theoretical Support**

numpy.dot(vector\_a, vector\_b, out = None) returns the dot product of vectors a and b. It can handle 2D arrays but considers them as matrix and will perform matrix multiplication.

C**ode**

import numpy as np

print("Enter the elements of first matrix- 2 X 2 ");

elements = list(map(int, input().split()));

a= np.array(elements).reshape(2,2);

print("Enter the elements of second matrix- 2 X 2");

elements = list(map(int, input().split()));

b= np.array(elements).reshape(2,2);

print("\nFirst Matrix\n",a)

print("Second matrix\n",b)

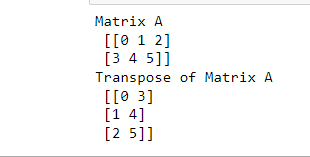
print("\nDot product of first and second matrix");

np.dot(a, b)

**Inference**

We can find the dot product of the 2 matrix using numpy.dot() function.During matrix multiplication,we multiply the values of the rows of matrix a with the values of the columns of matrix b and sum them up.

**Results and Observations**



**TRANSPOSE OF MATRIX**

**Program No: 9 Date:22-11-22**

**AIM:**Write a python program to find the transpose of the matrix.

**Theoretical Support**

With the help of **Numpy numpy.transpose()**, We can perform the simple function of transpose within one line by using **numpy.transpose()** method of Numpy. It can transpose the 2-D arrays on the other hand it has no effect on 1-D arrays. This method transpose the 2-D numpy array.

C**ode**

import numpy as np

a= np.arange(6).reshape((2,3))

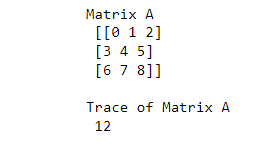
print("Matrix A\n",a)

b=np.transpose(a)

print("Transpose of Matrix A\n",b)

**Inference**

**Results and Observations**



**TRACE OF MATRIX**

**Program No: 10 Date:22-11-22**

**AIM:**Write a python program to find the trace of the matrix.

**Theoretical Support**

**Code**

import numpy as np

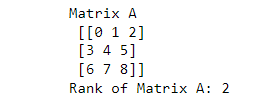
a= np.arange(9).reshape((3,3))

print("Matrix A\n",a)

b=np.trace(a)

print("\nTrace of Matrix A\n",b)

**Results and Observations**



**RANK OF MATRIX**

**Program No: 11 Date:22-11-22**

**AIM:**Write a python program to find the rank of the matrix.

**Theoretical Support**

**Code**

import numpy as np

a= np.arange(9).reshape((3,3))

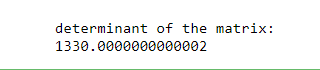
print("Matrix A\n",a)

b=np.linalg.matrix\_rank(a)

print("Rank of Matrix A:",b)

**Inference**

**Results and Observations**

****

**DETERMINANT OF MATRIX**

**Program No: 11 Date:22-11-22**

**AIM:**Write a python program to find the determinant of the matrix.

**Theoretical Support**

**Code**

import numpy as np

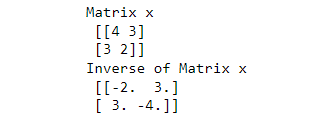
a = np.array([[50,29], [30,44]])

print("\ndeterminant of the matrix:")

print(np.linalg.det(a))

**Inference**

**Results and Observations**

****

**INVERSE OF MATRIX**

**Program No: 12 Date:22-11-22**

**AIM:**Write a python program to find the inverse of the matrix.

**Theoretical Support**

**Code**

import numpy as np

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

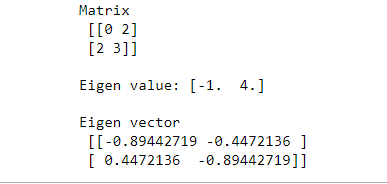
y = np.linalg.inv(x)

print("Matrix x\n",x)

print("Inverse of Matrix x\n",y)

**Inference**

**Results and Observations**



**EIGEN VALUES AND EIGEN VECTORS**

**Program No: 12 Date:22-11-22**

**AIM:**Write a python program to find the eigen values and eigen vectors..

**Theoretical Support**

**Code**

import numpy as np

from numpy.linalg import eig

a = np.array([[0, 2], [2, 3]])

print("Matrix\n",a)

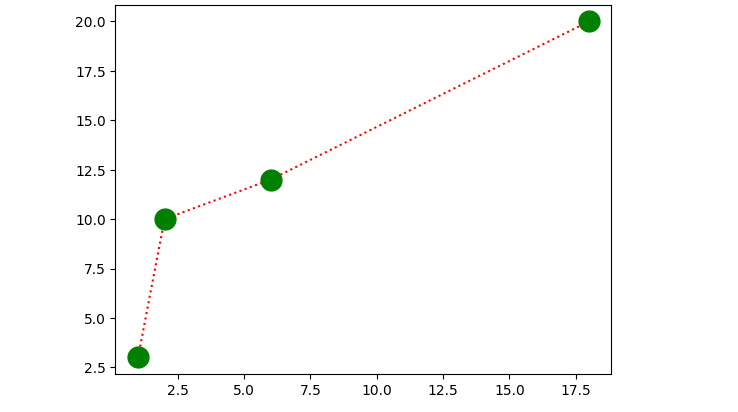
val,vec=eig(a)

print('\nEigen value:', w)

print('\nEigen vector\n', v)

**Inference**

**Results and Observations**

****

**EXERCISE 3:PROGRAM USING MATPLOTLIB**

**LINE DIAGRAM**

**Program No: 13 Date:22-11-22**

**AIM:**Draw a line in a diagram from position (1, 3) to (2, 10) then to (6, 12) and finally to position  (18, 20). (Mark each point with a beautiful green colour and set line colour to red and line style dotted)

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

import numpy as np

xpoints = np.array([1, 2, 6, 18])

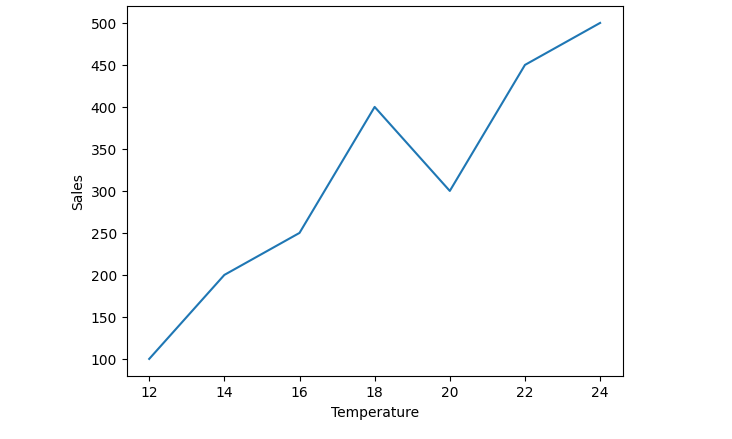
ypoints = np.array([3, 10, 12, 20])

plt.plot(xpoints, ypoints,marker='o',color='r',linestyle=':',mfc='g',mec='g',ms='15')

plt.show()

**Inference**

**Results and Observations**

****

**PLOT FOR THE GIVEN DATA**

**Program No: 14 Date:22-11-22**

**AIM:**Draw a plot for the following data:

|  |  |
| --- | --- |
| **Temperature in degree Celsius** | **Sales** |
| 12 | 100 |
| 14 | 200 |
| 16 | 250 |
| 18 | 400 |
| 20 | 300 |
| 22 | 450 |
| 24 | 500 |

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

import numpy as np

xpoints = [12,14,16,18,20,22,24]

ypoints = [100,200,250,400,300,450,500]

plt.plot(xpoints,ypoints)

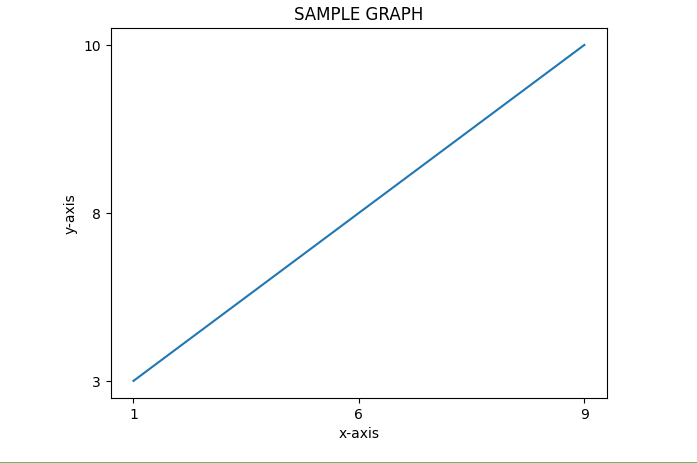
plt.xlabel("Temperature")

plt.ylabel("Sales")

plt.show()

**Inference**

**Results and Observations**



**LINE DIAGRAM USING TEXT FILE**

**Program No: 15 Date:22-11-22**

**AIM:**Write a Python program to draw a line using given axis values taken from a text file, with suitable label in the x axis, y axis and a title

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

with open("plot.txt") as f:

data = f.read()

data = data.split('\n')

x = [row.split(' ')[0] for row in data]

y = [row.split(' ')[1] for row in data]

plt.plot(x,y)

plt.title("SAMPLE GRAPH")

plt.xlabel("x-axis")

plt.ylabel("y-axis")

plt.show()

**Plot.txt**

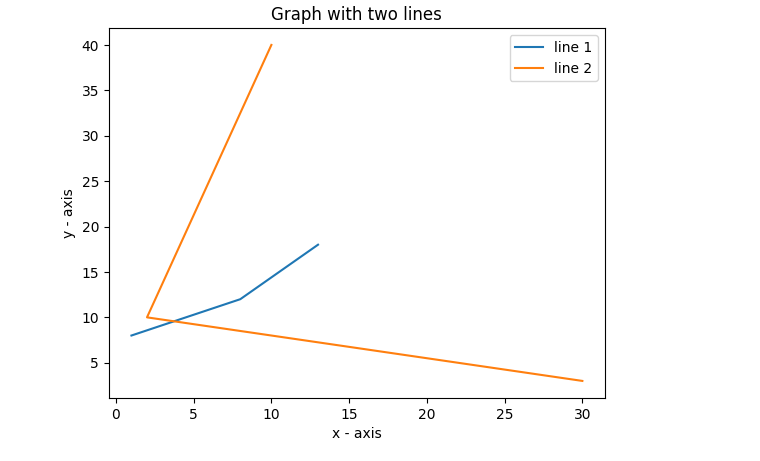
1 3

6 8

9 10

**Inference**

**Results and Observations**



**MULTIPLE LINES ON SAME PLOT**

**Program No: 16 Date:22-11-22**

**AIM:**Write a Python program to plot two or more lines on same plot with suitable legends of each line

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

x1 = [1,8,13]

y1 = [8,12,18]

plt.plot(x1, y1, label = "line 1")

x2 = [10,2,30]

y2 = [40,10,3]

plt.plot(x2, y2, label = "line 2")

plt.title('Graph with two lines')

plt.xlabel('x - axis')

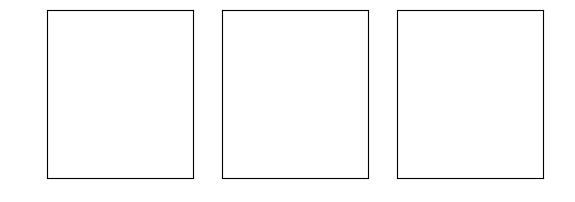
plt.ylabel('y - axis')

plt.legend()

plt.show()

**Inference**

**Results and Observations**

****

**MULTIPLE PLOTS**

**Program No: 17 Date:22-11-22**

**AIM:**Write a Python program to create multiple plots.

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

plt.subplot(2, 3, 4)

plt.xticks(())

plt.yticks(())

plt.subplot(2, 3, 5)

plt.xticks(())

plt.yticks(())

plt.subplot(2, 3, 6)

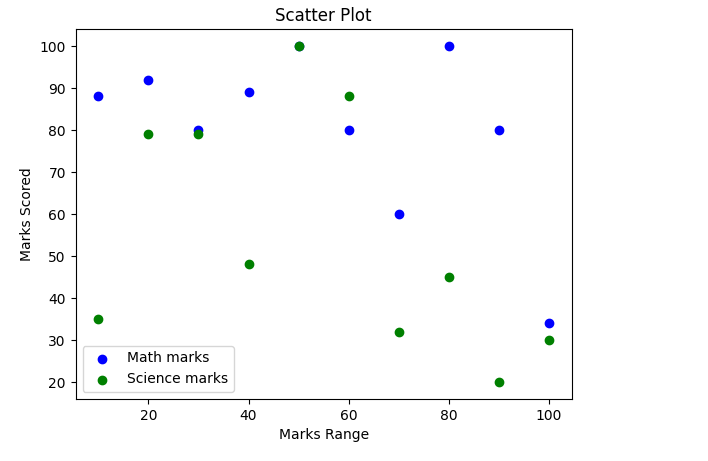
plt.xticks(())

plt.yticks(())

plt.show()

**Inference**

**Results and Observations**



**SCATTER PLOT**

**Program No: 18 Date:22-11-22**

**AIM:**Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students.

**Sample data:**

math\_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

science\_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks\_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

math\_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

science\_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks\_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.scatter(marks\_range, math\_marks, label='Math marks', color='b')

plt.scatter(marks\_range, science\_marks, label='Science marks', color='g')

plt.title('Scatter Plot')

plt.xlabel('Marks Range')

plt.ylabel('Marks Scored')

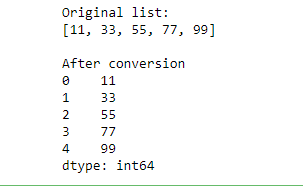
plt.legend()

plt.show(**)**

**Inference**

**CYCLE 2**

**Results and Observations**



**EXERCISE 4:INTRODUCTION TO PANDAS**

**LIST TO SERIES CONVERSION**

**Program No: 19 Date:22-11-22**

**AIM:**Write a python program to implement List-to-Series Conversion.

**Theoretical Support**

**Code**

import pandas as pd

list = [11, 33, 55, 77, 99]

print("Original list:")

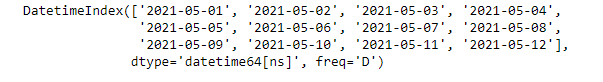
print(list)

print("\nAfter conversion")

print(pd.Series(list))

**Inference**

**Results and Observations**

****

**GENERATING SERIES OF DATES**

**Program No: 20 Date:22-11-22**

**AIM:**Write a python program to Generate the series of dates from 1st May, 2021 to 12th May, 2021 (both inclusive).

**Theoretical Support**

**Code**

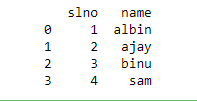
import pandas as pd

result = pd.date\_range(start = '05-01-2021', end = '05-12-2021')

print(result)

**Inference**

**Results and Observations**

****

**DICTIONARY TO DATA FRAME**

**Program No: 21 Date:22-11-22**

**AIM:**Given a dictionary, convert it into corresponding dataframe and display it.

**Theoretical Support**

**Code**

import pandas as pd

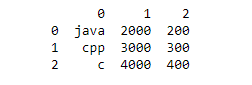
data={'slno':[1,2,3,4],'name':['albin','ajay','binu','sam']}

df=pd.DataFrame(data)

print(df)

**Inference**

**Results and Observations**

****

**LIST TO DATA FRAME**

**Program No: 22 Date:22-11-22**

**AIM:**Given a 2D List, convert it into corresponding dataframe and display it.

**Theoretical Support**

**Code**

import pandas as pd

data = [['java', '2000', '200'],

['cpp','3000','300'],

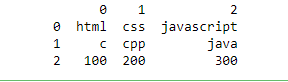
['c', '4000','400']]

data = pd.DataFrame(data)

print(data)

**Inference**

**Results and Observations**

****

**CSV FILE TO DATA FRAME**

**Program No: 23 Date:22-11-22**

**AIM:**Given a CSV file, read it into a dataframe and display it.

**Theoretical Support**

**Code**

import pandas as pd

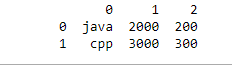
data = pd.read\_csv ('sample.csv',header=None)

df=pd.DataFrame(data)

print(df)

**Inference**

**Results and Observations**

****

**FIRST 2 ROWS OF DATA FRAME**

**Program No: 24 Date:22-11-22**

**AIM:**Given a dataframe, select first 2 rows and output them.

**Theoretical Support**

**Code**

import pandas as pd

data = [['java', '2000', '200'],

['cpp','3000','300'],

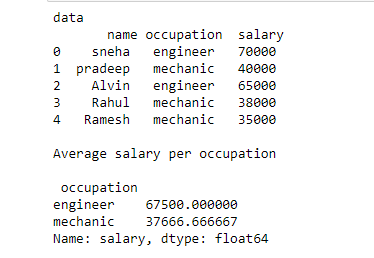
['c', '4000','400']]

df = pd.DataFrame(data)

print(df.head(2))

**Inference**

**Results and Observations**

****

**AVERAGE SALARY PER OCCUPATION**

**Program No: 25 Date:22-11-22**

**AIM:**Given is a dataframe showing name, occupation, salary of people. Find the average salary per occupation.

**Theoretical Support**

**Code**

import pandas as pd

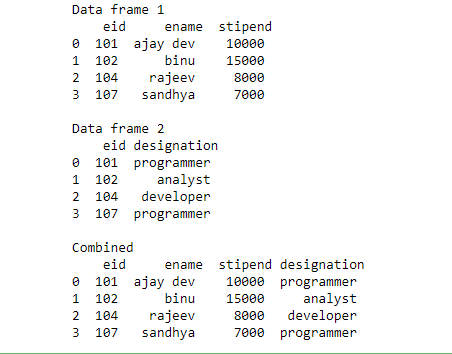
data=pd.read\_csv('people.csv')

print("data\n",data)

print("\nAverage salary per occupation\n\n",data.groupby('occupation')['salary'].mean())

**Inference**

**Results and Observations**

****

**DISPLAYING EMPLOYEE DETAILS FROM DATA FRAME**

**Program No: 26 Date:22-11-22**

**AIM:**Given are 2 dataframes, with one dataframe containing Employee ID (eid), Employee Name (ename) and Stipend (stipend) and the other dataframe containing Employee ID (eid) and designation of the employee (designation). Output the Dataframe containing Employee ID (eid), Employee Name (ename), Stipend (stipend) and Position (position).

**Theoretical Support**

**Code**

import pandas as pd

df1=pd.read\_csv('employee1.csv')

df2=pd.read\_csv('employee2.csv')

print("Data frame 1\n",df1)

print("\nData frame 2\n",df2)

print("\nCombined\n",pd.merge(df1,df2, how = 'inner', on = 'eid'))

**Inference**

**CYCLE 3**

**Results and Observations**

**Exercise_5**

**EXERCISE 5**

**K-NN CLASSIFICATION**

**Program No: 27 Date:22-11-22**

**AIM:**Program to implement **k-NN classification** using any standard dataset available in the public domain and find the accuracy of the algorithm

**Theoretical Support**

**Code**

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

irisData=load\_iris()

x=irisData.data

y=irisData.target

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)

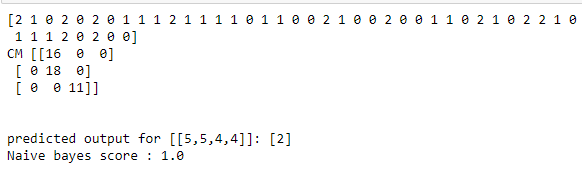
knn.fit(X\_train, y\_train)

print("Accuracy")

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

**Inference**

**Results and Observations**

****

**EXERCISE 6**

**NAIVE BAYES ALGORITHM**

**Program No: 28 Date:22-11-22**

**AIM:**Program to implement **Naïve Bayes Algorithm** using any standard dataset available in the public domain and find the accuracy of the algorithm

**Theoretical Support**

**Code**

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

X,y=load\_iris(return\_X\_y=True)

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.30,random\_state=0)

gnb=GaussianNB()

y\_pred=gnb.fit(X\_train,y\_train).predict(X\_test)

print(y\_pred)

x\_new=[[5,5,4,4]]

y\_new=gnb.fit(X\_train,y\_train).predict(x\_new)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print ("CM", cm)

print("\n")

print("predicted output for [[5,5,4,4]]:",y\_new)

print("Naive bayes score :",gnb.score(X\_test,y\_test))

**Inference**

**Results and Observations**

Exercise_7

**EXERCISE 7**

**DECISION TREE**

**Program No: 29 Date:22-11-22**

**AIM:**Program to implement **decision trees** using any standard dataset available in the public domain and find the accuracy of the algorithm

**Theoretical Support**

**Code**

from sklearn.datasets import load\_iris

from sklearn import metrics

from sklearn import tree

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

iris=load\_iris()

x=iris.data

y=iris.target

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.3,random\_state=1)

clf=DecisionTreeClassifier()

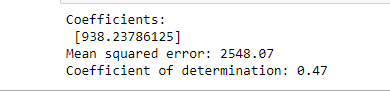
clf=clf.fit(x\_train,y\_train)

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

print("Accuracy: ",metrics.accuracy\_score(y\_test,y\_pred))

**Inference**

**Results and Observations**

****

**EXERCISE 8**

**REGRESSION**

**Program No: 30 Date:22-11-22**

**AIM:**Program to implement linear and multiple **regression** techniques using any standard dataset available in the public domain and evaluate its performance

**Theoretical Support**

**Code**

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

df = datasets.load\_diabetes()

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=True)

diabetes\_X = diabetes\_X[:, np.newaxis, 2]

diabetes\_X\_train = diabetes\_X[:-20]

diabetes\_X\_test = diabetes\_X[-20:]

diabetes\_y\_train = diabetes\_y[:-20]

diabetes\_y\_test = diabetes\_y[-20:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print("Coefficients: \n", regr.coef\_)

print("Mean squared error: %.2f" % mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

print("Coefficient of determination: %.2f" % r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

**Inference**

**Results and Observations**

**Exercise_9**

**EXERCISE 9**

**SUPPORT VECTOR MACHINE**

**Program No: 31 Date:22-11-22**

**AIM:**Program to implement text classification using a **Support vector machine**.

**Theoretical Support**

**Code**

from sklearn.datasets import load\_iris

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

from sklearn import metrics

from sklearn.svm import SVC

iris = load\_iris()

x = iris.data

y = iris.target

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.3,random\_state=1)

classifier = SVC(kernel='linear', random\_state=0)

classifier.fit(x\_train, y\_train)

y\_pred= classifier.predict(x\_test)

print("Accuracy : ",metrics.accuracy\_score(y\_test,y\_pred))

sample = [[1,1,1,2]]

pred = classifier.predict(sample)

pred\_v = [iris.target\_names[p] for p in pred]

print(pred\_v)

**Inference**