**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Name of the Experiment** | **Page No.** |
| 1 | 06.08.2024 | 1. Review of Python Programming 2. Review of python and matrix operations using NumPy 3. To perform Data visualization using Matplotlib 4. Familiarization of Pandas | 2 |
| 2 | 13.08.2024 | K-NN Classifier | 37 |
| 3 | 27.08.2024 | Naïve Bayes Classifier | 42 |
| 4 | 03.09.2024 | Decision Trees Classifier | 45 |
| 5 | 10.09.2024 | 1. Simple Linear Regression 2. Multiple Linear Regression | 49 |
| 6 | 01.10.2024 | CNN (Convolutional Neural Network) Classifier | 55 |
| 7 | 08.10.2024 | Text Classification using Support Vector Machine | 59 |
| 8 | 08.10.2024 | K-Means Clustering | 61 |
| 9 | 15.10.2024 | 1. Simple Web crawler 2. Web Scraping | 64 |
| 10 | 29.10.2024 | NLP Techniques   1. Part of Speech tagging 2. N-gram generation 3. Chunking using NLTK | 69 |

# Experiment 1.A

## Review of Python Programming

**Aim:** To perform a review of python programming.

### SOURCE CODE & OUTPUT:

**Basic Data Types**

print("hello World")



a=5 print(a)

print(type(a))



f=1.5

print(f) print(type(f))



s="hello" print(s) print(type(s))



b=True print(b) print(type(b))



t=True f=False print(t,f)



t=5+3j print(t) print(type(t))



### Arithmetic Operators

a=7 b=3

sum=a+b print(sum)



diff=a-b print(diff)



pro=a\*b print(pro)



quo=a/b print(quo)



iquo=a//b print(iquo)



rem=a % b print(rem)



pow= a\*\*b print(pow)

### Boolean Operations

t=True f=False print(t,f)



p=5>3

print(p)



q=-1<-12.5

print(q)



print(p and q)



print(p or p)



print(not q)



### STRING OPERATIONS

s='hello' u="hello" print(s) print(u)



s1="python" s2='world' s3=s1+' '+s2

print(s3)



s3='%s %s %d' %(s1,s2,1011)

print(s3)



print(len(s3))



print(s3.upper())



print(s3.capitalize())



print(s3.lower())



print('hello world how are you'.split(' '))



print('book'.replace('o','e'))



word='jewellery' print(word.find('well')) print(word.find('is'))



### Control Structures IF-ELSE

number= 123

if number>99 and number<1000 : print('3 digit')

else:

print('Not 3 digit')



response=input('Are you familiar with python ') if response.upper()=="YES":

print("You can skip this course:-)") elif response.upper() == "NO": print("You are at the right place:-)") else:

print('Sorry wrong input :-(')



### FOR LOOP

for x in range(10): print(x,end=' ')



limit=int(input('Enter a limit :')) sum=0

for i in range(1,limit+1):

if i%2!=0:

sum+=i

print("Odd sum="+str(sum))



print(list(range(10)))



print(list(range(1,10)))



print(list(range(1,10,2)))



### WHILE LOOP

number=int(input('Enter number :')) s=0

while number>0: s+=number%10 number=number//10 print(s)



### NESTED LOOP

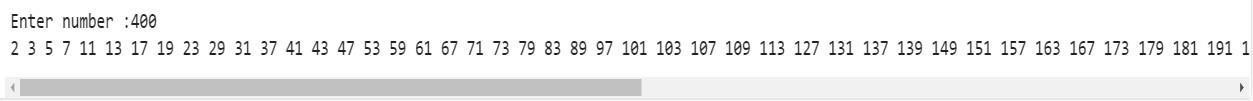
limit=int(input('Enter number :')) for num in range (2,limit+1): is\_divisible=False

k=2

while k<=num//2: if num % k==0: is\_divisible=True break;

k+=1

if not is\_divisible: print(num,end=' ')



### Containers List

mylist=['a','b',1,1.2,True] print(mylist) mylist.append("new") print(mylist)

print(mylist.pop())



mylist.insert(2,'new') print(mylist)



mylist.remove('new') print(mylist)



b=[1,2,3]

print(b) mylist.append(b) print(mylist)



mylist.remove(b)

print(mylist)



mylist.extend(b) print(mylist)



a=[2,3,1,4,5]

a.sort() print(a)



print(list('hello'))



### List Slicing

numbers=[0,1,2,3,4,5,6,7,8,9,10]

print(numbers[1],numbers[-1])



sliced=numbers[5:11] print(sliced)



slice1=numbers[5:] print(slice1)



Sliced=numbers[:7] print(Sliced)



slice2=numbers[-2:] print(slice2)



### List Comprehension

numbers=list(range(1,8)) print(numbers)



square=[]

for i in numbers: square.append(pow(i,2))

print(square)



square=[x\*\*2 for x in numbers] print(square)



odd\_square=[x\*\*2 for x in numbers if x%2!=0] print(odd\_square)



A=[4,6,8,9]

AxA=[(a,b) for a in A for b in A if a!=b ] print(AxA)



### Dictionary

person={'name':'Manu','age':28} print(person['name'])



print('name' in person)



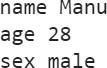
print('sex' in person)



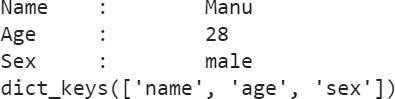
person['sex']='male' print(person)



for item in person: print(item,person[item])



for(key,value)in person.items(): print(key.capitalize(),'\t:\t',value)

print(person.keys())

### FUNCTIONS

**Finding the Square of the number**

def square(number):

return pow(number,2) s=square(5)

print(s)



### To check if a given number is prime

def isPrime(number):

for factor in range(2, (number//2)+1): if number%factor == 0:

return False return True

number = int(input('Enter the number ')) print(isPrime(number))



### Prime in given range

def printPrimes(llimit, ulimit):

for num in range(llimit,ulimit+1):

if isPrime(num)==True: print(num,end=' ')

printPrimes(5,50)



### Swap 2 numbers

def swap(x,y):

t=x x=y y=t

return x,y a=5

b=7 a,b=swap(a,b) print(a,b)



### RESULT:

Review of Python programming was executed successfully.

# Experiment-1B Review of python and matrix operations using NumPy

**AIM:** To perform a review of python and matrix operations using Numpy programming.

### SOURCE CODE & OUTPUT:

import numpy as np

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



print(type(x))



print(x.shape)



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

print(y.shape)



z = np.array([[1+0.j,2+5.j]]) print(z)

print(z.shape)



a = np.zeros((2,3)) print(a)



print(a.shape)



b = np.ones((2,3), dtype=int) print(b)



d = np.eye(3) print(d)



e = np.arange(10) print(e)



e = np.arange(12, 21) print(e)



e = np.arange(5,20,3)

print(e)



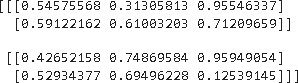
f = np.linspace(1,20,7) print(f)



g = np.random.random((3,4)) print(g)



h = np.random.random((3,4)) print(h.reshape(2,2,3))



x = np.arange(12) print(x) print(x[4])



print(x[-1])



x.resize(3,4) print(x)



print(x[-1,-1])



print(x[2][3])



y = np.arange(1,26) print(y)

print(y[:3])



print(y[10:])



print(y[10:15])



print(y[-5:])



print(y[3:-3])



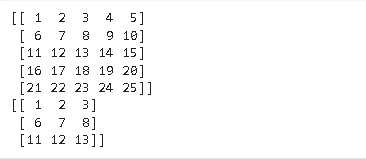
print(y[::3])



print(y.reshape((5,5))) print(y)



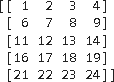
y = y.reshape((5,5)) print(y) print(y[:3,:3])



print(y[2:-1,1:-1])



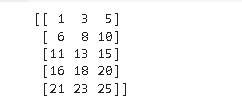
print(y[:,:-1])



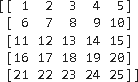
print(y[:,-1])



print(y[::,::2])



print(y)



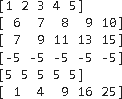
print(y[1::2,1::2])



a = np.arange(1,6) b = np.arange(6,11) print(a)

print(b) print(a+b) print(a-b)

print(b-a) print(a\*\*2)



print(a>3)



a = np.arange(0,4).reshape((2,2)) b = np.eye(2)

print(a\*b)



print(np.dot(a,b))



x = np.arange(1,10).reshape(3,3) print(x)



print(x.sum())



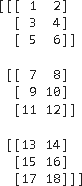
print(x.sum(axis=0))



print(x.sum(axis=1))



x = np.arange(1,19).reshape(3,3,2)



print(x) print(x.sum(axis=1))



x = np.arange(1,10).reshape(3,3) print(x)

print(x.max())



print(x.max(axis=0))



print(x.transpose())



### RESULT:

Review of Python Programming and matrix using Numpy was executed successfully.

# Experiment-1.C

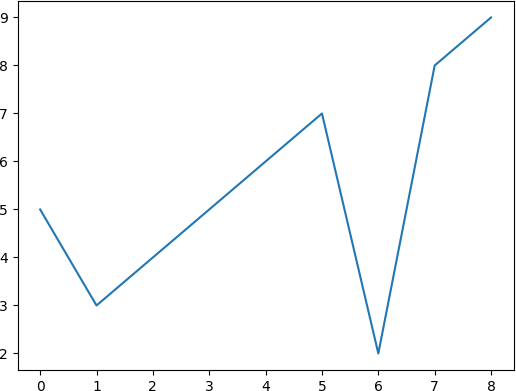
## To perform Data visualization using Matplotlib

**AIM:** To perform Data visualisation using Matplotlib

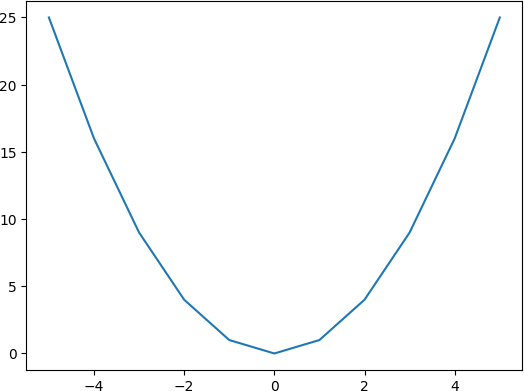
### SOURCE CODE & OUTPUT:

from matplotlib import pyplot as plt y = [5,3,4,5,6,7,2,8,9]

plt.plot(y) plt.show()



x = [-5,-4,-3,-2,-1,0,1,2,3,4,5]

y = [i\*\*2 for i in x] plt.plot(x,y) plt.show()

import numpy as np import math

x = np.arange(-1,1.1,0.1).tolist() y = [i\*\*2 + 5 for i in x]

print(x) print(y) plt.plot(x,y) plt.show()

[-1.0, -0.9, -0.8, -0.7000000000000001, -0.6000000000000001, -

0.5000000000000001, -0.40000000000000013, -0.30000000000000016, -

0.20000000000000018, -0.1000000000000002, -2.220446049250313e-16,

0.09999999999999964, 0.19999999999999973, 0.2999999999999998,

0.3999999999999997, 0.49999999999999956, 0.5999999999999996,

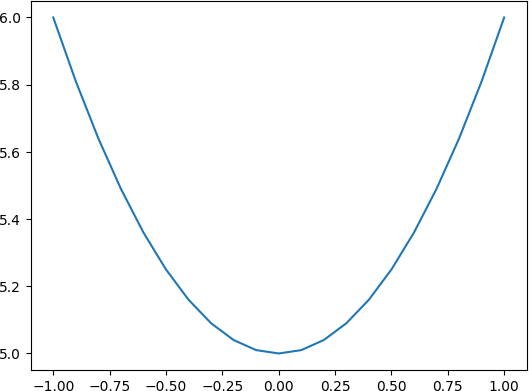
0.6999999999999997, 0.7999999999999996, 0.8999999999999995,

0.9999999999999996]

[6.0, 5.8100000000000005, 5.640000000000001, 5.49, 5.36, 5.25, 5.16,

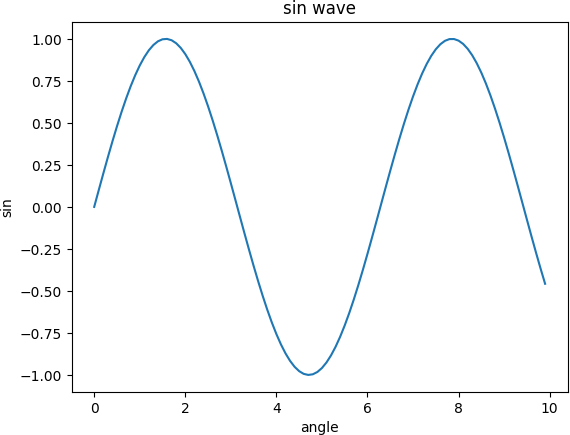
5.09, 5.04, 5.01, 5.0, 5.01, 5.04, 5.09, 5.16, 5.25, 5.359999999999999,

5.489999999999999, 5.64, 5.809999999999999, 5.999999999999999]

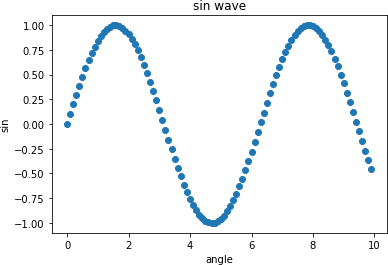


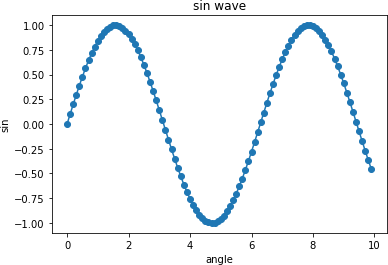
|  |  |  |  |
| --- | --- | --- | --- |
| import numpy as np |  |  |  |
| x = np.arange(0,10,0.1) |  |  |  |
| y = np.sin(x) |  |  |  |
| print(x) |  |  |  |
| print(y) |  |  |  |
| plt.plot(x,y) |  |  |  |
| plt.xlabel('angle') |  |  |  |
| plt.ylabel('sin') |  |  |  |
| plt.title('sin wave') |  |  |  |
| plt.show() |  |  |  |
| [0. 0.1 0.2 0.3 0.4 0.5 | 0.6 0.7 0.8 0.9 1. | 1.1 1.2 1.3 1.4 1.5 | 1.6 1.7 |
| 1.8 1.9 2. 2.1 2.2 2.3 | 2.4 2.5 2.6 2.7 2.8 | 2.9 3. 3.1 3.2 3.3 | 3.4 3.5 |
| 3.6 3.7 3.8 3.9 4. 4.1 | 4.2 4.3 4.4 4.5 4.6 | 4.7 4.8 4.9 5. 5.1 | 5.2 5.3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5.4 5.5 5.6 | 5.7 5.8 5.9 | 6. 6.1 6.2 | 6.3 6.4 6.5 | 6.6 6.7 6.8 | 6.9 7. 7.1 |
| 7.2 7.3 7.4 | 7.5 7.6 7.7 | 7.8 7.9 8. | 8.1 8.2 8.3 | 8.4 8.5 8.6 | 8.7 8.8 8.9 |
| 9. 9.1 9.2 | 9.3 9.4 9.5 | 9.6 9.7 9.8 | 9.9] |  |  |
| [ 0. | 0.09983342 | 0.19866933 | 0.29552021 | 0.38941834 | 0.47942554 |
| 0.56464247 | 0.64421769 | 0.71735609 | 0.78332691 | 0.84147098 | 0.89120736 |
| 0.93203909 | 0.96355819 | 0.98544973 | 0.99749499 | 0.9995736 | 0.99166481 |
| 0.97384763 | 0.94630009 | 0.90929743 | 0.86320937 | 0.8084964 | 0.74570521 |
| 0.67546318 | 0.59847214 | 0.51550137 | 0.42737988 | 0.33498815 | 0.23924933 |
| 0.14112001 | 0.04158066 | -0.05837414 | -0.15774569 | -0.2555411 | -0.35078323 |
| -0.44252044 | -0.52983614 | -0.61185789 | -0.68776616 | -0.7568025 | -0.81827711 |
| -0.87157577 | -0.91616594 | -0.95160207 | -0.97753012 | -0.993691 | -0.99992326 |
| -0.99616461 | -0.98245261 | -0.95892427 | -0.92581468 | -0.88345466 | -0.83226744 |
| -0.77276449 | -0.70554033 | -0.63126664 | -0.55068554 | -0.46460218 | -0.37387666 |
| -0.2794155 | -0.1821625 | -0.0830894 | 0.0168139 | 0.1165492 | 0.21511999 |
| 0.31154136 | 0.40484992 | 0.49411335 | 0.57843976 | 0.6569866 | 0.72896904 |
| 0.79366786 | 0.85043662 | 0.8987081 | 0.93799998 | 0.96791967 | 0.98816823 |
| 0.99854335 | 0.99894134 | 0.98935825 | 0.96988981 | 0.94073056 | 0.90217183 |
| 0.85459891 | 0.79848711 | 0.7343971 | 0.66296923 | 0.58491719 | 0.50102086 |
| 0.41211849 | 0.31909836 | 0.22288991 | 0.12445442 | 0.02477543 | -0.07515112 |
| -0.17432678 | -0.27176063 | -0.36647913 | -0.45753589] | |  |



plt.scatter(x,y) plt.xlabel('angle') plt.ylabel('sin') plt.title('sin wave') plt.show()



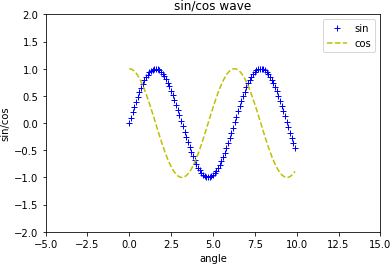
plt.plot(x,y) plt.scatter(x,y) plt.xlabel('angle') plt.ylabel('sin') plt.title('sin wave') plt.show()

plt.plot(x,np.sin(x), 'b+', label='sin')

plt.plot(x,np.cos(x) ,'y--', label='cos') plt.xlabel('angle')

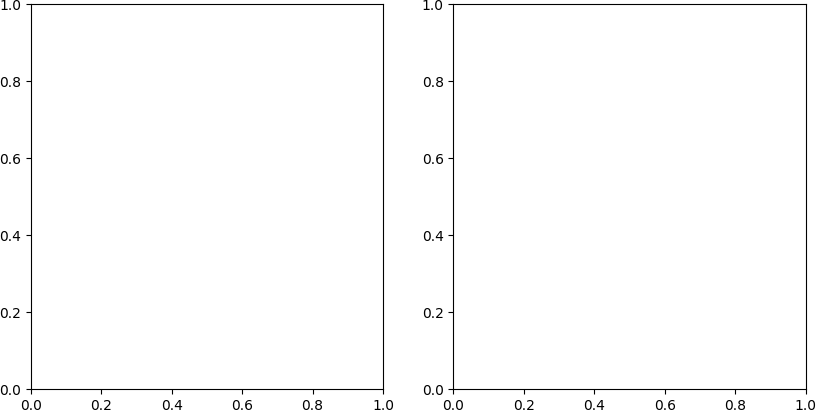
plt.ylabel('sin/cos') plt.title('sin/cos wave') plt.ylim(-2,2)

plt.xlim(-5,15) plt.legend() plt.show()



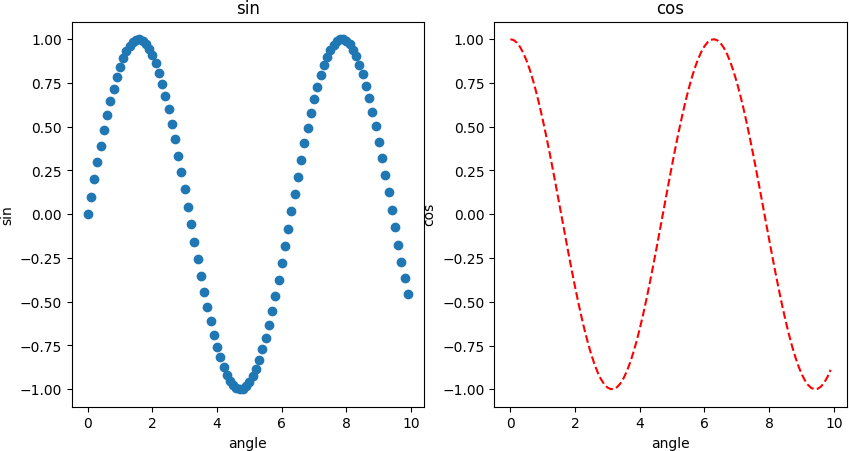
### Subplot

fig, axis = plt.subplots(1,2, figsize=(10,5)) print(axis.shape)



fig, axis = plt.subplots(1,2, figsize=(10,5)) x = np.arange(0,10,0.1) axis[0].plot(x,np.sin(x), 'g--') axis[0].set\_title('sin') axis[0].set\_xlabel('angle') axis[0].set\_ylabel('sin') axis[1].plot(x,np.cos(x), 'r--') axis[1].set\_title('cos') axis[1].set\_xlabel('angle') axis[1].set\_ylabel('cos')

plt.show()



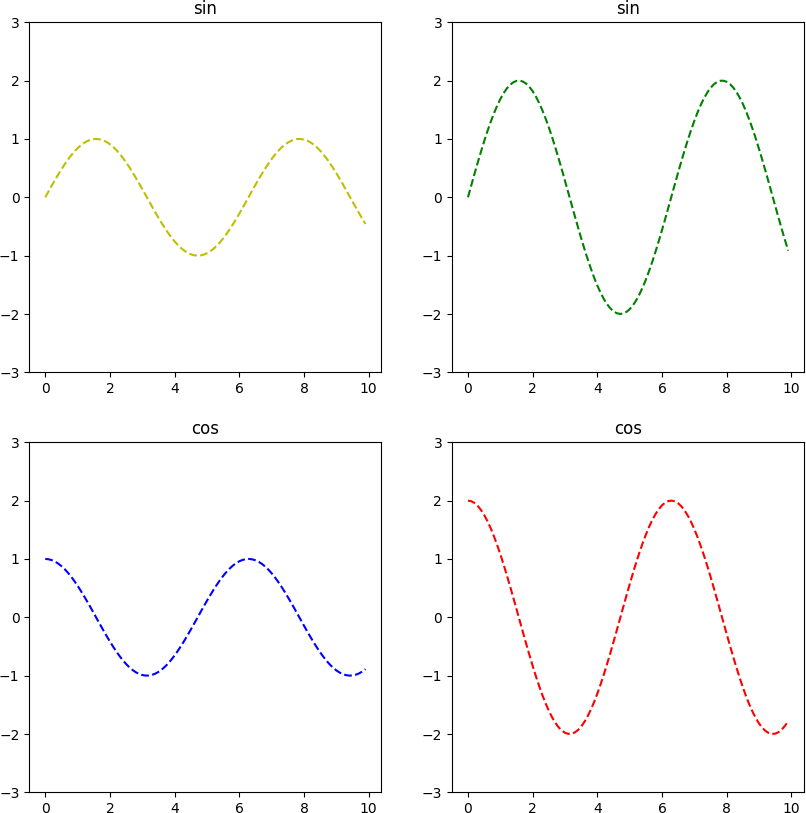
fig, axis = plt.subplots(2,2, figsize=(10,10)) x = np.arange(0,10,0.1) axis[0][0].plot(x,np.sin(x), 'y--') axis[0][0].set\_title('sin')

axis[0][0].set\_ylim(-3,3) axis[0][1].plot(x,2\*np.sin(x), 'g--') axis[0][1].set\_title('sin')

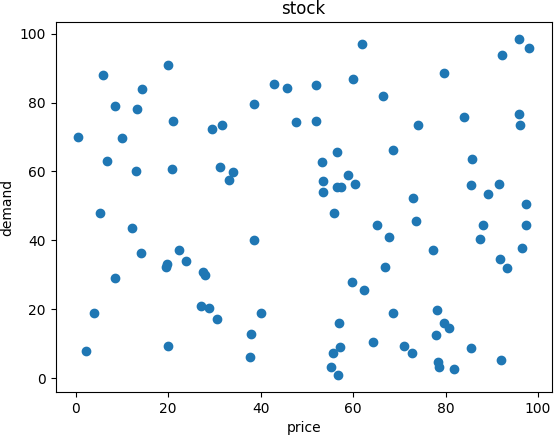
axis[0][1].set\_ylim(-3,3) axis[1][0].plot(x,np.cos(x), 'b--') axis[1][0].set\_title('cos')

axis[1][0].set\_ylim(-3,3) axis[1][1].plot(x,2\*np.cos(x), 'r--') axis[1][1].set\_title('cos')

axis[1][1].set\_ylim(-3,3) plt.show()



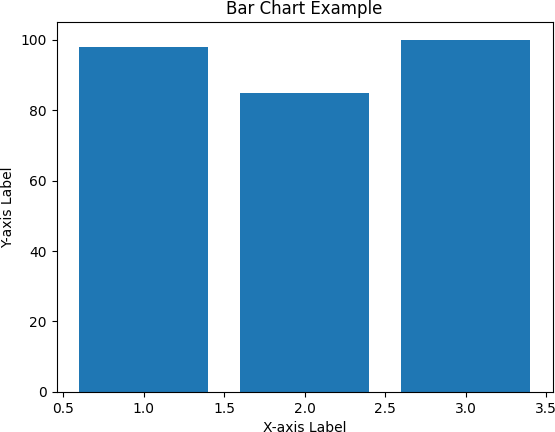
x = np.random.random(100)\*100 y = np.random.random(100)\*100 plt.scatter(x,y)

plt.xlabel('price') plt.ylabel('demand') plt.title('stock') plt.show()

x = np.array([1,2,3]) y = [98,85,100]

plt.bar(x,y)

plt.xlabel('X-axis Label') plt.ylabel('Y-axis Label') plt.title('Bar Chart Example') plt.show()



slice = [12, 25, 50, 36, 19]

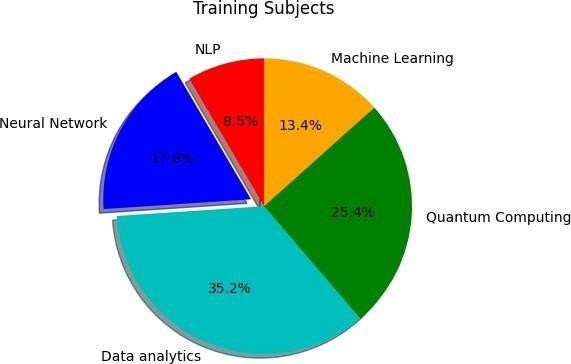
activities = ['NLP', 'Neural Network', 'Data analytics', 'Quantum Computing', 'Machine Learning']

cols = ['r', 'b', 'c', 'g', 'orange'] plt.pie(slice,

labels=activities, colors=cols, startangle=90, shadow=True, explode=(0, 0.1, 0, 0, 0), autopct='%1.1f%%

)

plt.title('Training Subjects') plt.show()



days = [1, 2, 3, 4, 5]

age = [63, 81, 52, 22, 37]

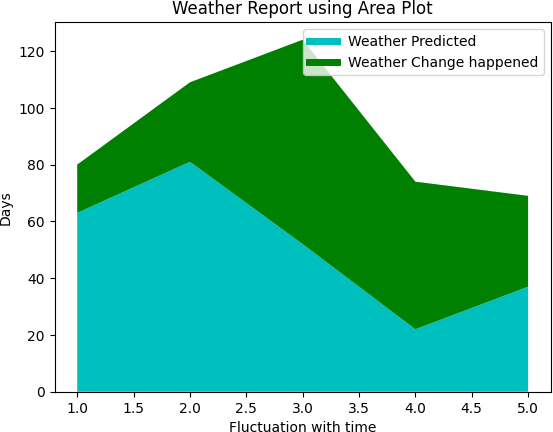
weight = [17, 28, 72, 52, 32]

plt.plot([], [], color='c', label='Weather Predicted', linewidth=5) plt.plot([], [], color='g', label='Weather Change happened', linewidth=5) plt.stackplot(days, age, weight, colors=['c', 'g'])

plt.xlabel('Fluctuation with time') plt.ylabel('Days')

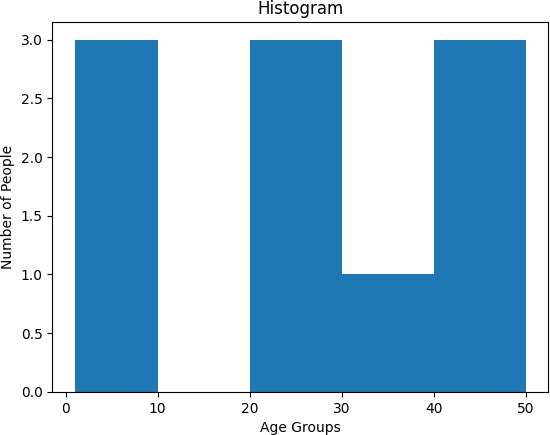
plt.title('Weather Report using Area Plot') plt.legend()

plt.show()



pop = [22, 55, 62, 45, 21, 22, 34, 42, 42, 4, 2, 8]

bins = [1, 10, 20, 30, 40, 50]

plt.hist(pop, bins, rwidth=1) plt.xlabel('Age Groups') plt.ylabel('Number of People')

plt.title('Histogram') plt.show()

### RESULT:

Data Visualization using Matplotlib was executed succesffully.

# Experiment-1.D

## Familiarization of Pandas

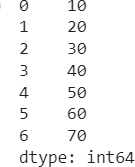
**AIM** :Familarization of Pandas.

### SOURCE CODE & OUTPUT:

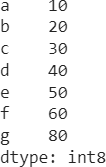
import numpy as np import pandas as pd

### Pandas Series

data = pd.Series([10, 20, 30, 40, 50, 60, 70])

data

data = pd.Series([10, 20, 30, 40, 50, 60, 80], index = ['a','b','c','d','e','f','g'], dtype

= 'int8') data

data.values



array\_data = data.values print(array\_data)



data.index

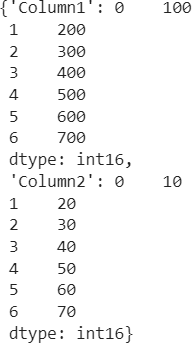
data\_series = {

'Column1' : pd.Series([100, 200, 300, 400, 500, 600, 700], dtype =

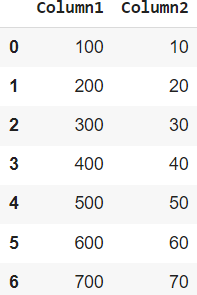
'int16'),

'Column2' : pd.Series([10, 20, 30, 40, 50, 60, 70], dtype = 'int16')

}

data\_series

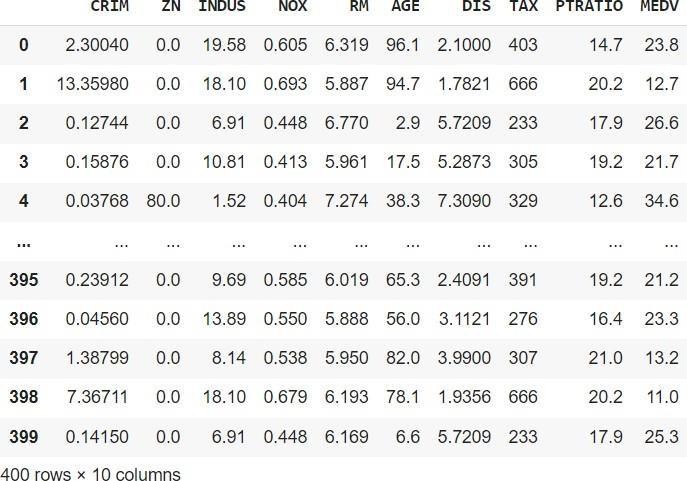
pd.DataFrame(data\_series)



### DataFrame

movies\_df = pd.read\_csv('https://raw.githubusercontent.com/ammishra08/MachineLearning/ master/Datasets/boston\_train.csv', sep = ',')

movies\_df



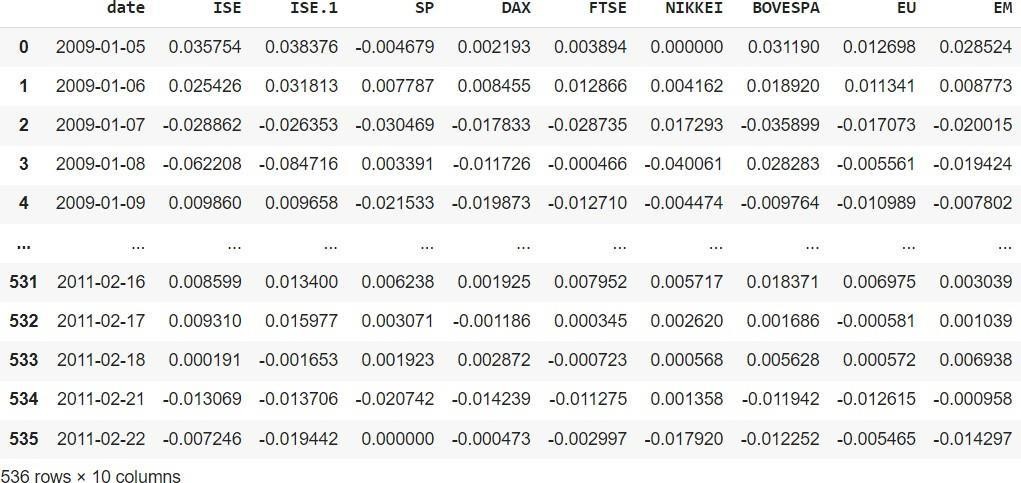
movies\_df.head()



movies\_df.tail()



stock\_data = pd.read\_excel("https://github.com/ammishra08/MachineLearning/raw/master/D atasets/data\_akbilgic.xlsx", header=1)

stock\_data

movies\_df.shape movies\_df.columns

len(movies\_df.columns) print(movies\_df.shape[0], movies\_df.shape[1])

### Data Manipulation

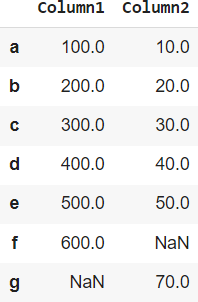
data\_series = {

'Column1' : pd.Series([100, 200, 300, 400, 500, 600], index = ['a','b','c','d','e','f'], dtype = 'int16'),

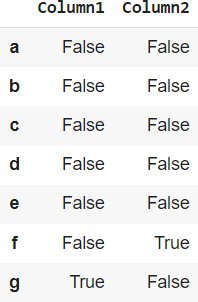
'Column2' : pd.Series([10, 20, 30, 40, 50, 70], index = ['a','b','c','d','e','g'], dtype = 'int16')

}

df = pd.DataFrame(data\_series) df



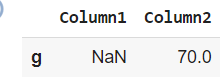
df.isnull()



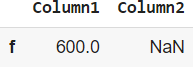
df.isnull().sum()



df.isna().sum()

df.notnull() df[df['Column1'].isnull() == True]

df[df['Column2'].isnull() == True]



### RESULT:

Familiarization of Pandas was executed successfully.

# Experiment-2

## K-NN Classifier

**AIM:** To implement decision tree using any standard dataset available in the public domain and find the accuracy of the algorithm.

**KNN WITH DIABETES DATASET**

### ALGORITHM:

Step-1: Load the dataset- diabetes.csv

Step 2: Pre-process the dataset by replacing zeros suitable mean values. Step 3: Perform the training and testing dataset splitting

Step 4: Determine the number of neighbors for the training dataset. Step-5: Calculate the Euclidean distance of K number of neighbors

Step-6: Take the K nearest neighbors as per the calculated Euclidean distance. Step-7: Among these k neighbors, count the number of the data points in each category.

Step-8: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-9: Calculate the model performance by creating the confusion matrix using the test data and the predicted output

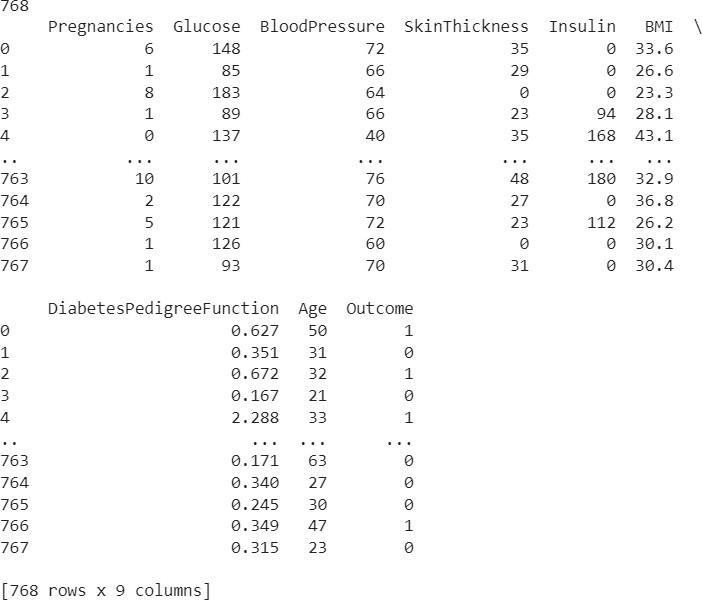
### SOURCE CODE & OUTPUT:

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy\_score

dataset = pd.read\_csv('/content/diabetes.csv')

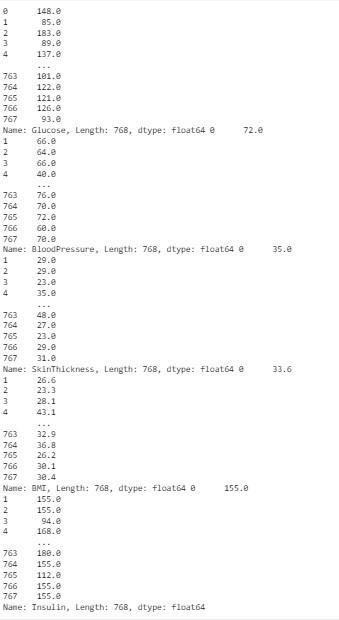
print(len(dataset)) print(dataset)



zero\_not\_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Insulin'] for column in zero\_not\_accepted:

dataset[column] = dataset[column].replace(0, np.NaN) mean = int(dataset[column].mean(skipna=True))# Calculate mean of dataset

dataset[column] = dataset[column].replace(np.NaN, mean) print(dataset['Glucose'],dataset['BloodPressure'],dataset['SkinThickness'],dataset[' BMI'],dataset['Insulin'])



X = dataset.iloc[:, 0:8] y = dataset.iloc[:, 8]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,random\_state=0, test\_size=0.2)

print(len(X\_train)) print(len(y\_train)) print(len(X\_test))

print(len(y\_test))



sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train) X\_test = sc\_X.transform(X\_test)

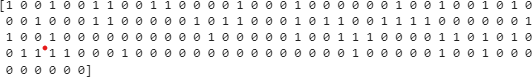
import math math.sqrt(len(y\_test))



classifier = KNeighborsClassifier(n\_neighbors=11,metric='euclidean') classifier.fit(X\_train, y\_train)



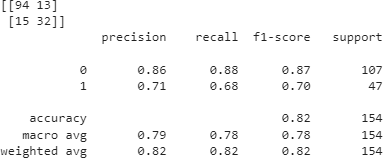
y\_pred=classifier.predict(X\_test) print(y\_pred)



print(accuracy\_score(y\_test,y\_pred)\*100, '%')



from sklearn.metrics import classification\_report, confusion\_matrix print(confusion\_matrix(y\_test, y\_pred)) print(classification\_report(y\_test, y\_pred)



new\_data = [[140, 72, 35, 0, 33.6, 0.627, 45, 1],

[120, 70, 30, 1, 25.2, 0.2, 35, 0]]

predictions = classifier.predict(new\_data) for prediction in predictions:

print(f"Predicted target: {prediction}")



### RESULT

k-NN classification model on diabetes dataset is build and the accuracy of the algorithm is determined.

# Experiment-3

## Naïve Bayes Classifier

**AIM:** To implement Naive Bayes classification using any standard datasetavailable in the public domain and find the accuracy of the algorithm.

### ALGORITHM:

Step 1:start

Step 2: Importing the standard libraries.

Step 3:Load the iris dataset-iris.csv The iris dataset contains the following data50 samples of 3 different species of iris (150 samples total) Measurements: sepal length, sepal width, petal length, petal width The format for the data: (sepal length, sepal width, petal length, petal width)

Step 4:Define x and y and label the fields

Step 5:Split the dataset into Training and testing

Step 6:Preprocess the dataset using StandardScaler StandardScaler removes the mean and scales each feature/variable to unit variance

Step 7:Train the data using GuassianNB model Step 8:Test the data using Test set

Step 9:Create the confusion matrix and Find the accuracy score Step 10:Stop

### SOURCE CODE & OUTPUT:

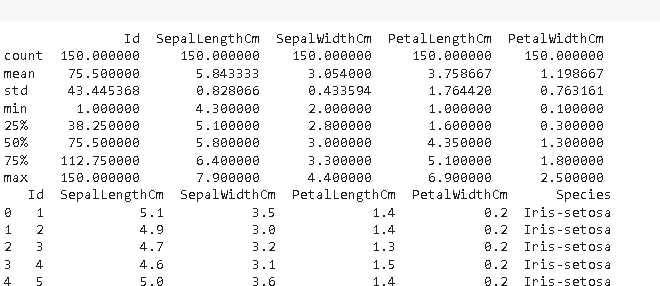
import pandas as pd

from sklearn.preprocessing import LabelEncoder from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix,accuracy\_score from sklearn.model\_selection import train\_test\_split

import sklearn.naive\_bayes

dataset = pd.read\_csv('/content/Iris.csv') print(dataset.describe()) print(dataset.head())



X = dataset.iloc[:, [1, 2, 3, 4]].valuesy = dataset.iloc[:, -1].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=0)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.fit\_transform(X\_test)

classifier = sklearn.naive\_bayes.GaussianNB() classifier.fit(X\_train, y\_train)

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

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

= accuracy\_score(y\_test, y\_pred) print("Confusion Matrix:") print(cm)

print("Accuracy Score:", ac\*100,'%')



new\_data = [[5.1, 3.5, 1.4, 0.2],

[6.2, 3.4, 5.4, 2.3]]

predictions = classifier.predict(new\_data)

for prediction in predictions: print(f"Predicted class: {prediction}")



### RESULT:

Naïve Bayes classification model on Iris dataset is build and the accuracy of the algorithm is determined.

# Experiment-4

## Decision Tree Classifier

**AIM:** To implement decision tree using any standard dataset available in the public domain and find the accuracy of the algorithm.

### ALGORITHM:

Step 1: Import the necessary packages and classes Step 2: Load the Data Set

Step 3: Extract feature matrix and target from the data frame Step 4: Split the data into training and testing sets

Step 5: Create a Decision Tree Classifier

Step 6: Train the classifier on the training data Step 7: Make predictions on the test data

Step 8: Generate a confusion matrix and classification report Step 9: Visualize the decision tree

Step 10: Save the figure as an image

### SOURCE CODE & OUTPUT:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.metrics import classification\_report, confusion\_matrix from sklearn.tree import plot\_tree

from sklearn import tree data=load\_iris() X=data.data y=data.target print(X.shape,y.shape)



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=10)

dtc = DecisionTreeClassifier() dtc.fit(X\_train, y\_train)



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

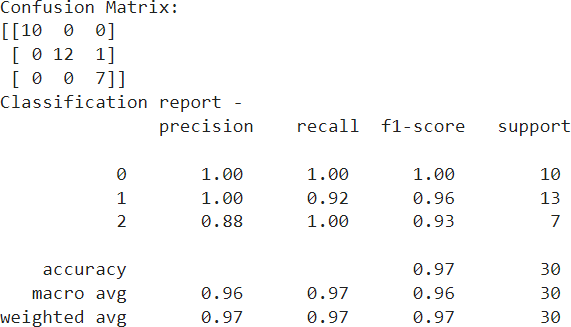
cm = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix:")

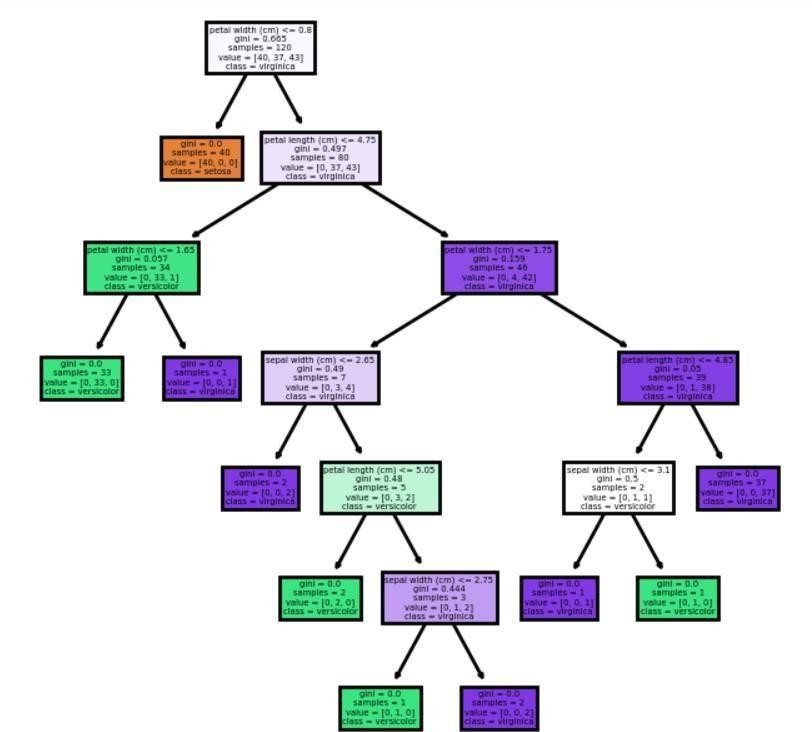
print(cm)

print("Classification report - \n", classification\_report(y\_test,y\_pred)) fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4, 4), dpi=200) tree.plot\_tree(dtc, feature\_names=data.feature\_names, class\_names=data.target\_names, filled=True)

plt.show()

fig.savefig("iris\_tree.png")





new\_data = [[5.1, 3.5, 1.4, 0.2],[6.2, 3.4, 5.4, 2.3]]

predictions = dtc.predict(new\_data) for prediction in predictions:

print(f"Predicted class: {prediction}")



### RESULT:

Decision tree classification model dataset is build and the accuracy of the algorithm is determined.

# Experiment-5.A

## Simple Linear Regression

**AIM:** To predict the salary based on the number of years of experience.

### ALGORITHM:

Step 1: Load the data set

Step 2: Extract the features and labels from the dataframe Step 3: Split the dataset into the Training set and Test set Step 4: Perform data visualization on train and test data

Step 5: Initialize a Linear Regression model and fit the model on the training data

Step 6: Predict on the test data

Step 7: Calculate the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2)

Step 8: Predict on a new data Step 9: Stop

### SOURCE CODE & OUTPUT:

import numpy as np

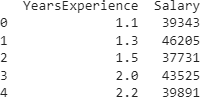
import matplotlib.pyplot as plt import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

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

dataset = pd.read\_csv('/content/Salary\_data.csv') X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 1].values print(dataset.head())



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

plt.scatter(X\_train, y\_train, color='red', label='Actual')

plt.plot(X\_train, regressor.predict(X\_train), color='blue', label='Predicted') plt.title('Salary VS Experience (Training set)')

plt.xlabel('Year of Experience') plt.ylabel('Salary')

plt.legend() # Add a legend to distinguish between actual and predicted data plt.show()

plt.scatter(X\_test, y\_test, color='red', label='Actual')

plt.plot(X\_train, regressor.predict(X\_train), color='blue', label='Predicted') plt.title('Salary VS Experience (Test set)')

plt.xlabel('Year of Experience') plt.ylabel('Salary')

plt.legend() # Add a legend to distinguish between actual and predicted data plt.show()

regressor = LinearRegression() regressor.fit(X\_train, y\_train)



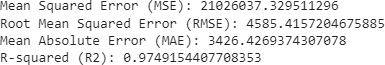
y\_pred = regressor.predict(X\_test)

y\_pred



mse = mean\_squared\_error(y\_test, y\_pred) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error (MSE):", mse) print("Root Mean Squared Error (RMSE):", rmse) print("Mean Absolute Error (MAE):", mae) print("R-squared (R2):", r2)

new\_input = [[5]]

y\_pred = regressor.predict(new\_input) print("Predicted Salary:", y\_pred)

### RESULT:

Predicted the salary based on the number of years of experience using simple linear regression and accuracy of the algorithm is determined.

## Experiment 5.B Multiple Linear Regression

**AIM:** To predict the salary based on the number of years of experience.

### ALGORITHM

Step 1: Load the data set

Step 2: Extract the features and labels from the dataframe, Define X and y Step 3: Split the dataset into the Training set and Test set

Step 4: Initialize a Linear Regression model and fit the model on the training data

Step 5: Predict on the test data Step 6: Perform data Visualization

Step 7: Calculate the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2)

Step 8: Predict on a new data Step 9: Stop

### SOURCE CODE & OUTPUT:

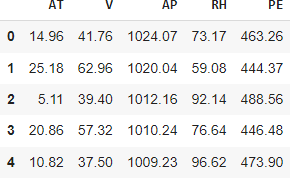
import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

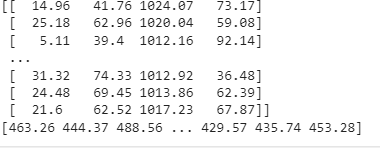
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

data\_df = pd.read\_excel('/content/CCCP.xlsx') data\_df.head()



x = data\_df.drop(['PE'], axis=1).values print(x)

y = data\_df['PE'].values print(y)



x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state=0)

regressor = LinearRegression() regressor.fit(x\_train, y\_train)



y\_pred = regressor.predict(x\_test) print(y\_pred)

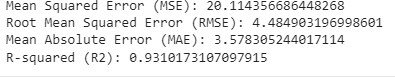
plt.figure(figsize=(15, 10)) plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual') plt.ylabel('Predicted') plt.title('ACTUAL VS PREDICTED')

plt.show()

mse = mean\_squared\_error(y\_test, y\_pred)rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred)r2 = r2\_score(y\_test, y\_pred) print("Mean Squared Error (MSE):", mse) print("Root Mean Squared Error (RMSE):", rmse)print("Mean Absolute Error (MAE):", mae) print("R-squared (R2):", r2)



new\_input = [[14.96, 41.76, 1024.07, 73.17]]

y\_pred = regressor.predict(new\_input) print("Predicted target value:", y\_pred)



### RESULT:

Predicted the salary based on the number of years of experience using Multiple linear regression and accuracy of the algorithm is determined.

## Experiment 6 Convolutional Neural Network

**AIM:** Programs to implement Convolutional Neural Network to classify images from any standard dataset in the public domain using Keras framework.

### ALGORITHM:

Step 1: Import Libraries:

Import the deep learning framework of your choice (e.g., TensorFlow, PyTorch). Import other necessary libraries (e.g., NumPy for numerical operations).

Step 2:Load and Preprocess Data:

Load your dataset (images and corresponding labels). Preprocess the data (normalize, resize, etc.).

Step 3: Define the CNN Architecture:

Define the layers of your CNN, including convolutional layers, pooling layers, fully connected layers, etc.

Step 4: Compile the Model:

Specify the optimizer, loss function, and metrics. Step 5: Train the Model:

Feed the training data into the model and adjust the weights using backpropagation.

Step 6: Evaluate the Model:

Evaluate the performance of the trained model on the test set. Step 7: Make Predictions:

Use the trained model to make predictions on new data.

### SOURCE CODE & OUTPUT:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import cifar10 from PIL import Image

import numpy as np

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data() X\_train, X\_test = X\_train / 255.0, X\_test / 255.0



X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

model = keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

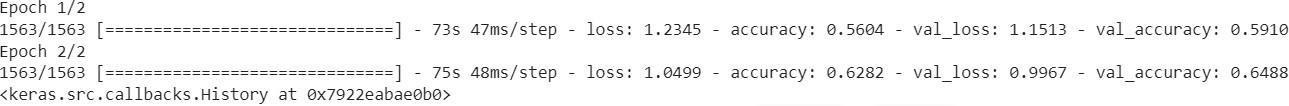
layers.Conv2D(64, (3, 3), activation='relu'), layers.Flatten(),

layers.Dense(64, activation='relu'), layers.Dense(10) # 10 output classes

])

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=2, validation\_data=(X\_test, y\_test))



test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2) print("\nTest accuracy:", test\_ac)

class\_names = [

"Airplane", "Automobile", "Bird",

"Cat",

"Deer",

"Dog",

"Frog",

"Horse",

"Ship", "Truck"

]

# Load and preprocess the image image\_path = '0007.jpeg'

image = Image.open(image\_path).resize((32, 32)) image = np.array(image) / 255.0

image = np.expand\_dims(image, axis=0)

# Make predictions

predictions = model.predict(image)

# Get the predicted class index

predicted\_class\_index = np.argmax(predictions)

# Get the class name from the class names list predicted\_class\_name = class\_names[predicted\_class\_index] print(f'Predicted class: {predicted\_class\_name}')



### RESULT:

CNN classification model dataset is build and the accuracy of the algorithm is determined.

# Experiment-7 Support Vector Machine

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

### ALGORITHM:

Step 1: Import necessary libraries. Step 2: Load the dataset

Step 3: Text Vectorization Step 4: Split the dataset

Step 5: Initialize the SVM model Step 6: Train the model

Step 7: Make Predictions Step 8: Evaluate the model

### SOURCE CODE & OUTPUT:

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, accuracy\_score,confusion\_matrix

newsgroups = fetch\_20newsgroups(subset='all', categories=['sci.space', 'rec.autos'], shuffle=True, random\_state=42)

X, y = newsgroups.data, newsgroups.target vectorizer = TfidfVectorizer(stop\_words='english') X = vectorizer.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,test\_size=0.3, random\_state=42)

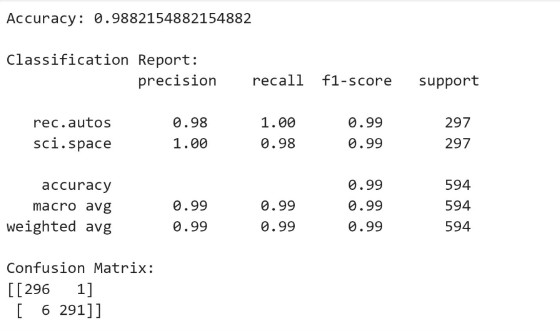
svm = SVC(kernel='linear') svm.fit(X\_train, y\_train)

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

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

print("\nClassification Report:\n", classification\_report(y\_test,y\_pred, target\_names=newsgroups.target\_names))

print("Confusion Matrix:") print(confusion\_matrix(y\_test, y\_pred))



### RESULT:

SVM text classification model dataset is build and the accuracy of the algorithm is determined.

# Experiment-8

## K – Means Clustering

**AIM:** To implement k-means clustering technique using any standard dataset available in the public domain.

### ALGORITHM:

Step 1: Load the Dataset

Step 2: Do the scatter plot and see that clusters are evident Step 3: Create an instance of K-Means

Step 4: Fit and make predictions

Step 5: Create the K-means cluster plot Step 6: Stop

### SOURCE CODE & OUTPUT:

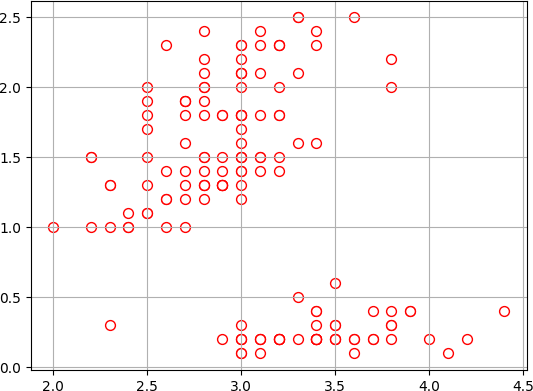
import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans iris = datasets.load\_iris()

X = iris.data y = iris.target

plt.scatter(X[:,1], X[:,3], color='white', marker='o', edgecolor='red', s=50) plt.grid()

plt.show()



kmc = KMeans(n\_clusters=3) y\_kmc = kmc.fit\_predict(X)

plt.scatter(X[y\_kmc == 0, 1], X[y\_kmc == 0, 3], s=50,c='lightgreen', marker='s',

edgecolor='black', label='Cluster 1')

plt.scatter(X[y\_kmc == 1, 1], X[y\_kmc == 1, 3], s=50,c='orange', marker='o',

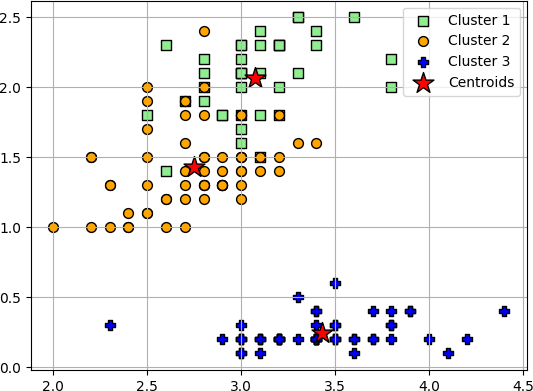
edgecolor='black', label='Cluster 2')

plt.scatter(X[y\_kmc == 2, 1], X[y\_kmc == 2, 3], s=50,c='blue', marker='P',

edgecolor='black', label='Cluster 3')

plt.scatter(kmc.cluster\_centers\_[:, 1], kmc.cluster\_centers\_[:, 3],s=250, marker='\*', c='red', edgecolor='black', label='Centroids')

plt.legend() plt.grid() plt.show()



### RESULT:

K-Means Clustering classification model dataset is build and the accuracy of the algorithm is determined.

# Experiment-9A

## Web Crawler

**AIM:** To implement a simple web crawler

### SOURCE CODE & OUTPUT:

!pip install requests

!pip install bs4

!pip install scrapy import logging

from urllib.parse import urljoin import requests

from bs4 import BeautifulSoup logging.basicConfig(

format='%(asctime)s %(levelname)s:%(message)s', level=logging.INFO

)

class Crawler:

def init (self, urls=[]): self.visited\_urls = [] self.urls\_to\_visit = urls

def download\_url(self, url): return requests.get(url).text

def get\_linked\_urls(self, url, html):

soup = BeautifulSoup(html, 'html.parser') for link in soup.find\_all('a'):

path = link.get('href')

if path and path.startswith('/'): path = urljoin(url, path) yield path

def add\_url\_to\_visit(self, url):

if url not in self.visited\_urls and url not in self.urls\_to\_visit: self.urls\_to\_visit.append(url)

def crawl(self, url):

html = self.download\_url(url)

for url in self.get\_linked\_urls(url, html): self.add\_url\_to\_visit(url)

def run(self):

while self.urls\_to\_visit:

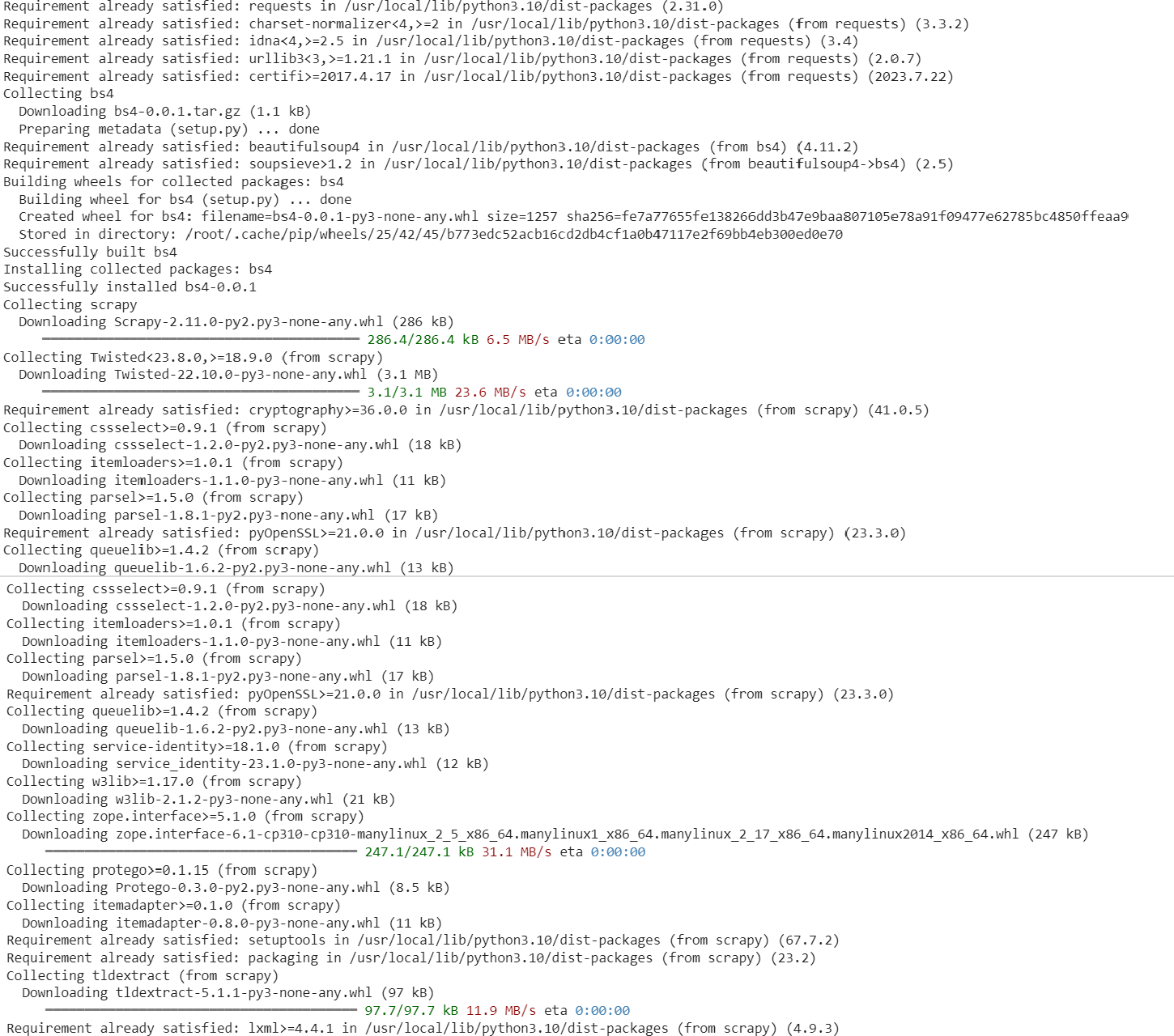
url = self.urls\_to\_visit.pop(0) logging.info(f'Crawling: {url}') try:

self.crawl(url) except Exception:

logging.exception(f'Failed to crawl: {url}') finally:

self.visited\_urls.append(url)

if name == ' main ': Crawler(urls=['https://docs.python.org/']).run()



### RESULT:

Simple Web crawler was executed Successfully.

# Experiment-9B

## Web Scrapping

**AIM:** To implement a program to scrap a web page of any website.

### SOURCE CODE & OUTPUT:

!pip install scrapy import scrapy

from scrapy.crawler import CrawlerProcess

class QuotesSpider(scrapy.Spider):name

= 'quotes'

start\_urls = [['http://quotes.toscrape.com/']](http://quotes.toscrape.com/%27)

def parse(self, response):

for quote in response.css('div.quote'): text = quote.css('span.text::text').get()author = quote.css('small::text').get()

print(f'Text: {text}\nAuthor: {author}\n{"-"\*40}')

next\_page = response.css('li.next a::attr(href)').get()if next\_page:

yield response.follow(next\_page, self.parse)

if name == " main ": process = CrawlerProcess({

'USER\_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)',

})

process.crawl(QuotesSpider) process.start()



### RESULT

Program to scrap a web page of a website was executed Successfully.

# Experiment-10A

## Parts of Speech Tagging

**AIM:** To demonstrate how to preprocess and analyze text data by tokenizing, removing stopwords, and performing part-of-speech tagging.

### SOURCE CODE & OUTPUT:

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize nltk.download('stopwords')

nltk.download('punkt') nltk.download('averaged\_perceptron\_tagger') stop\_words = set(stopwords.words('english'))

txt = "The quick brown fox jumps over the lazy dog. " \

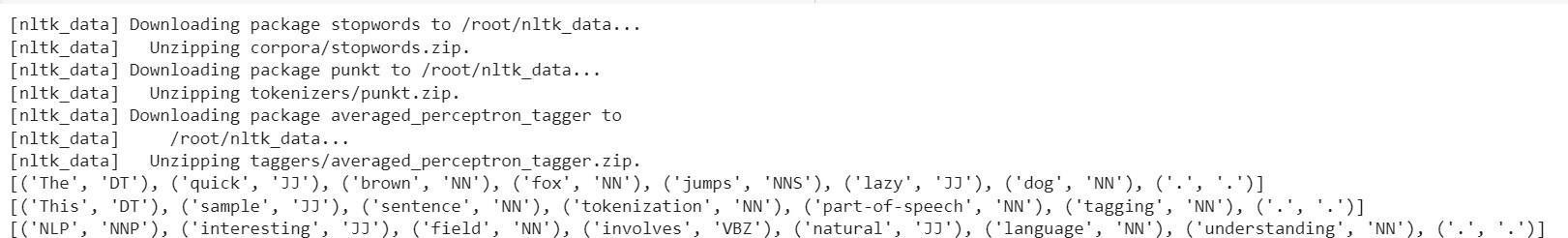
"This is a sample sentence for tokenization and part-of-speech tagging. " \ "NLP is an interesting field that involves natural language understanding."

tokenized = sent\_tokenize(txt) for i in tokenized:

wordsList = nltk.word\_tokenize(i)

wordsList = [w for w in wordsList if not w in stop\_words] tagged = nltk.pos\_tag(wordsList)

print(tagged)



# Experiment-10B

## N-gram generation

**AIM:** The program to preprocess sentiment-labeled financial news data, including loading the dataset, splitting it into training and testing sets, removing punctuation, and demonstrating the generation of N-grams for text classification tasks.

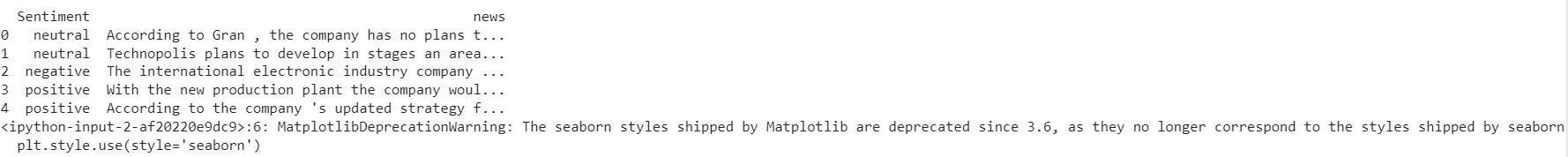
### SOURCE CODE & OUTPUT:

import numpy as np import pandas as pd

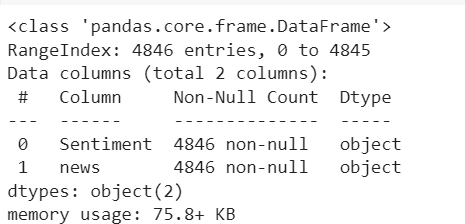
import matplotlib.pyplot as plt plt.style.use(style='seaborn') colnames = ['Sentiment', 'news']

df = pd.read\_csv('/content/all-data - all-data.csv', encoding="ISO-8859-1", names=colnames, header=None)

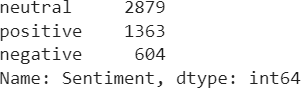
print(df.head())



df.info()



df['Sentiment'].value\_counts()



y = df['Sentiment'].values y.shape

x = df['news'].values x.shape



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) print(x\_train.shape)

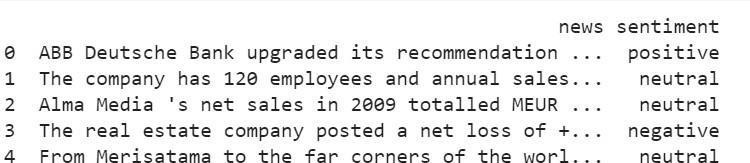
print(y\_train.shape) print(x\_test.shape) print(y\_test.shape)



df1 = pd.DataFrame(x\_train)

df1 = df1.rename(columns={0: 'news'}) df2 = pd.DataFrame(y\_train)

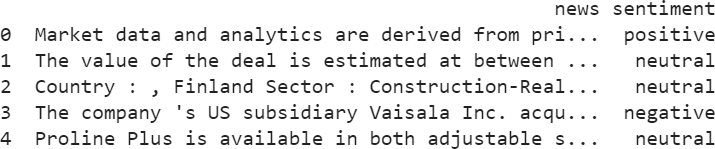
df2 = df2.rename(columns={0: 'sentiment'}) df\_train = pd.concat([df1, df2], axis=1) print(df\_train.head())



df3 = pd.DataFrame(x\_test)

df3 = df3.rename(columns={0: 'news'}) df4 = pd.DataFrame(y\_test)

df4 = df2.rename(columns={0: 'sentiment'}) df\_test = pd.concat([df3, df4], axis=1) print(df\_test.head())



import string

def remove\_punctuation(text): if type(text) == float:

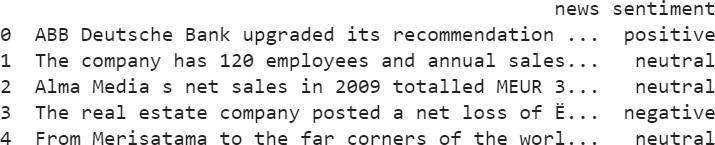
return text ans = ""

for i in text:

if i not in string.punctuation: ans += i

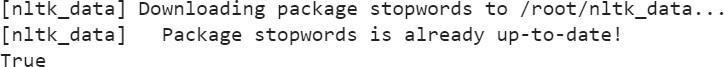
return ans

df\_train['news'] = df\_train['news'].apply(lambda x: remove\_punctuation(x)) df\_test['news'] = df\_test['news'].apply(lambda x: remove\_punctuation(x)) print(df\_train.head())



import nltk

from nltk.corpus import stopwords nltk.download('stopwords')



def generate\_N\_grams(text, ngram=1):

words = [word for word in text.split(" ") if word not in set(stopwords.words('english'))]

print("Sentence after removing stopwords:", words) temp = zip(\*[words[i:] for i in range(0, ngram)]) ans = [' '.join(ngram) for ngram in temp]

return ans

print(generate\_N\_grams("The sun rises in the east", 2))



print(generate\_N\_grams("The sun rises in the east", 3))



print(generate\_N\_grams("The sun rises in the east", 4))



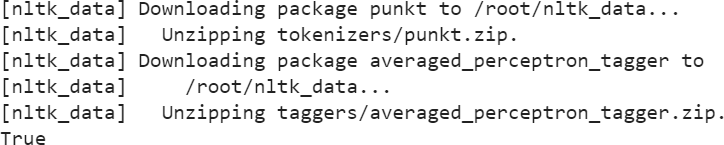
# Experiment-10 C Chunking

**AIM:** The program to read sentences from the 'news' column of the 'all-data.csv' file and perform Noun Phrase (NP) chunking on each sentence using natural language processing techniques.

### SOURCE CODE & OUTPUT:

import pandas as pd import nltk nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')



colnames = ['Sentiment', 'news']

df = pd.read\_csv('/content/all-data - all-data.csv', encoding="ISO-8859-1", names=colnames, header=None)

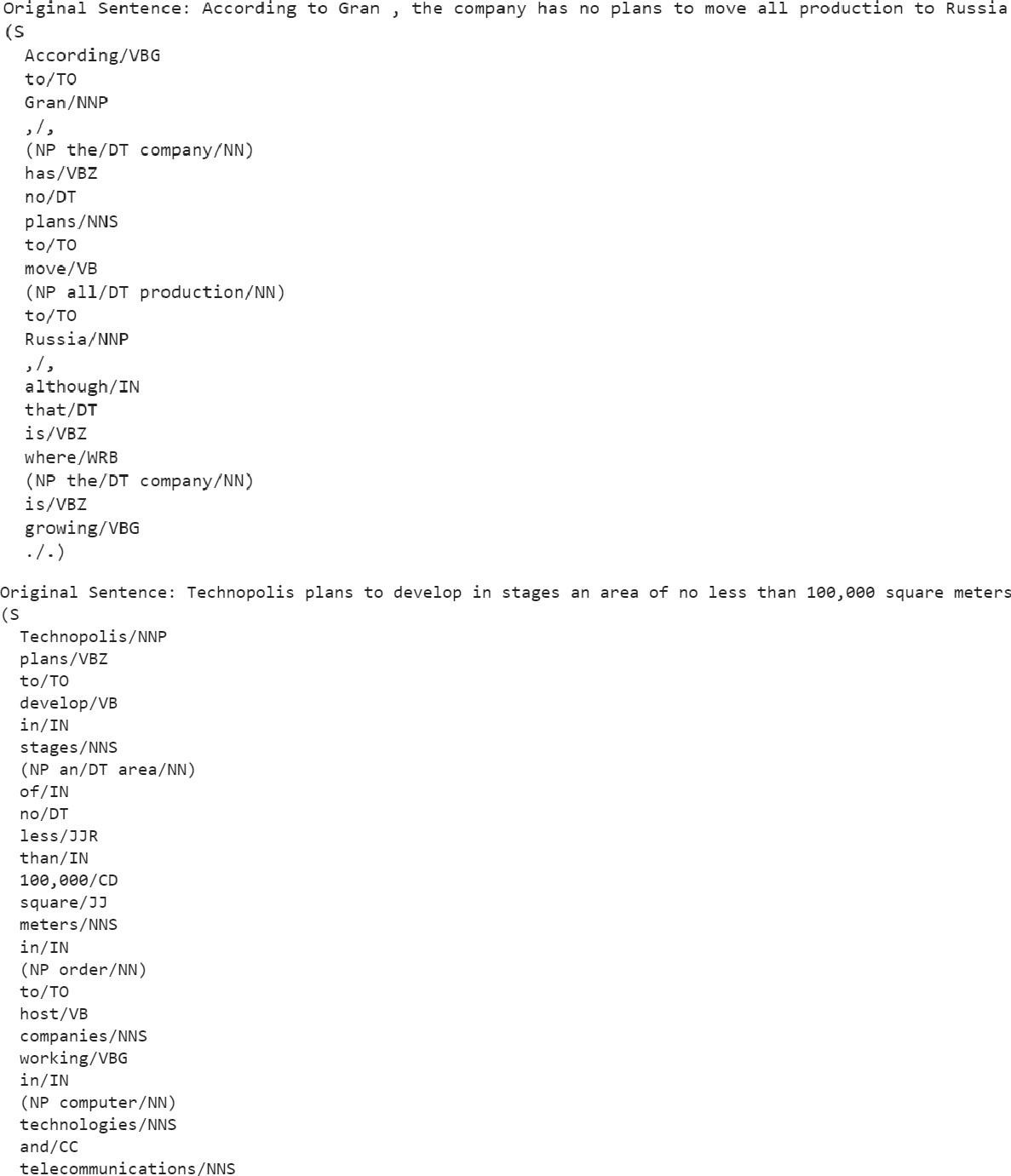
sentences\_for\_chunking = df['news'].head(3) def perform\_chunking(sentence):

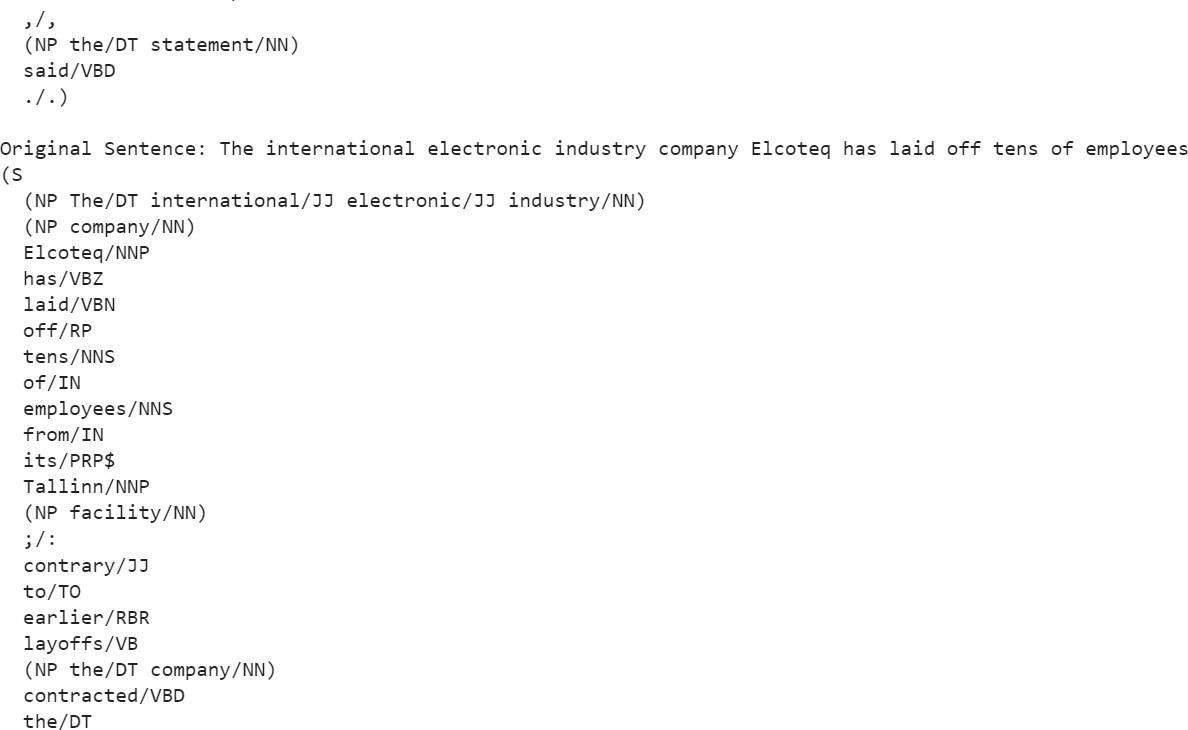
tokens = nltk.word\_tokenize(sentence) pos\_tags = nltk.pos\_tag(tokens) grammar = "NP: {<DT>?<JJ>\*<NN>}"

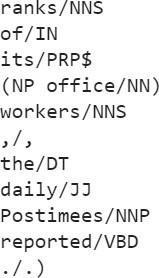
chunk\_parser = nltk.RegexpParser(grammar) chunks = chunk\_parser.parse(pos\_tags)

print(chunks)

for sentence in sentences\_for\_chunking: print("\nOriginal Sentence:", sentence) perform\_chunking(sentence)







### RESULT:

Parts of speech tagging,N-gram generation,Chunking was executed successfully.