**CYCLE 1**

# Program to Print all non-Prime Numbers in an Interval

# PROGRAM:

first=int(input("Enter the First Limit:"))

last=int(input("Enter the last limit"))

for num in range(first,last + 1):

if num > 1:

for i in range(2,num):

if(num % i == 0):

print(num)

break

**OUTPUT:**



1. **Program to print the first N Fibonacci numbers.**

**PROGRAM:**

n = int (input("Enter the number of terms needed in the Fibonacci series: "))

if (n<0):

print ("Enter a positive number")

else:

f1, f2 = 0, 1

if n == 1:

print (f1)

elif n == 2:

print (f1,f2)

else:

print (f1,f2, end = ' ')

for i in range (3, n+1):

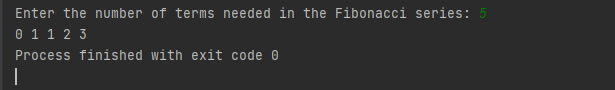
f3 = f1 + f2

print (f3, end = ' ')

f1 = f2

f2 = f3

**OUTPUT:**



1. **Program to find the roots of a quadratic equation (rounded to 2 decimal places)**

**PROGRAM:**

import cmath

a = int(input("Enter the value of a :"))

b = int(input("Enter the value of b :"))

c = int(input("Enter the value of c :"))

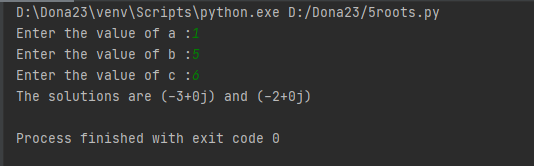
d = (b\*\*2) - (4\*a\*c)

sol1 = (-b-cmath.sqrt(d))/(2\*a)

sol2 = (-b+cmath.sqrt(d))/(2\*a)

print('The solutions are {0} and {1}'.format(sol1,sol2))

**OUTPUT:**



1. **Program to check whether a given number is perfect number or not(sum of factors =number)**

**PROGRAM:**

n = int(input(" Please Enter any Number: "))

Sum = 0

for i in range(1, n):

if(n % i == 0):

Sum = Sum + i

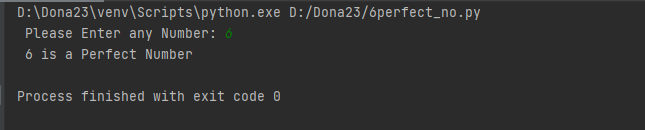
if (Sum == n):

print(" %d is a Perfect Number" %n)

else:

print(" %d is not a Perfect Number" %n)

**OUTPUT:**



1. **Program to display amstrong numbers upto 1000**

**PROGRAM:**

lower = int(input("Enter lower range: "))

upper = int(input("Enter upper range: "))

for num in range(lower, upper + 1):

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* 3

temp //= 10

if num == sum:

print(num)

**OUTPUT:**



1. **Write a program to perform bubble sort on a given set of elements.**

**PROGRAM:**

def bubbleSort(arr):

for i in range(n - 1):

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

arr = []

n = int(input("Enter limit :"))

print("Enter elements :")

for i in range(0,n):

arr.append(int(input()))

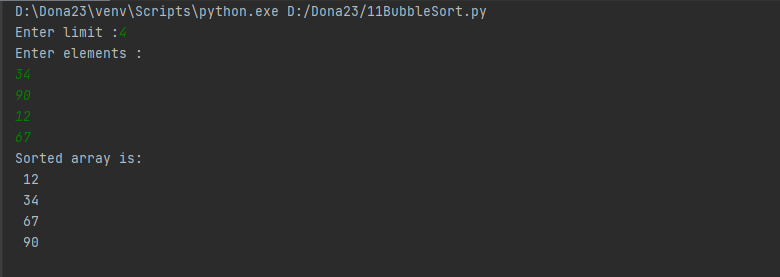
bubbleSort(arr)

print("Sorted array is:")

for i in range(len(arr)):

print("% d" % arr[i])

**OUTPUT:**



1. **Write a Python program that accept a positive number and subtract from this number the sum of its digits and so on. Continues this operation until the number is positive**

**PROGRAM:**

def repeat\_times(n):

s = 0

n\_str = str(n)

while (n > 0):

n -= sum([int(i) for i in list(n\_str)])

n\_str = list(str(n))

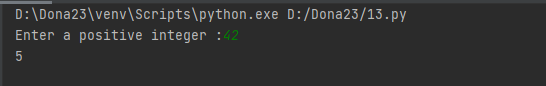
s += 1

return s

n=int(input("Enter a positive integer :"))

print(repeat\_times(n))

**OUTPUT:**



1. **Write a Python program that accepts a 10-digit mobile number, and find the digits which are absent in a given mobile number**

**PROGRAM:**

mobile = input('Please enter a mobile number: ' )

all = '0123456789'

print('Missing digits are ', set(all) - set(mobile))

**OUTPUT:**



**CYCLE 2**

1. **Create a 2-dimensional array (2X3) with elements belonging to complex data type and print it. Also display**

**a. the no: of rows and columns**

**b. dimension of an array**

**c. reshape the same array to 3X2**

**PROGRAM:**

import numpy as np

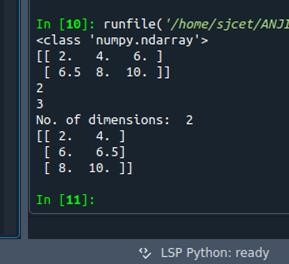
x = np.array([[2, 4, 6], [6.5, 8, 10]])

print(type(x)) print(x)

numOfRows = np. size(x, 0) print(numOfRows) numOfColumns = np. size(x, 1) print(numOfColumns)

print("No. of dimensions: ", x.ndim) rs=np.reshape(x, (3, 2))

print(rs)

**OUTPUT:**

1. **Create an one dimensional array using arange function containing 10 elements.**

**Display**

1. **First 4 elements**
2. **Last 6 elements**
3. **Elements from index 2 to 7**

**PROGRAM**

import numpy as np

a = np.arange(1, 11, 1)

print(a)

first\_element = a[:4]

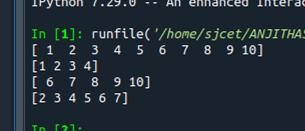
print(first\_element)

first\_element1 = a[5:]

print(first\_element1)

first\_element2 = a[1:7]

print(first\_element2)

 **OUTPUT**

1. **Create an 1D array with arange containing first 15 even numbers as elements**
2. **Elements from index 2 to 8 with step 2(also demonstrate the same using slice function)**
3. **Last 3 elements of the array using negative index**
4. **Alternate elements of the array**
5. **Display the last 3 alternate elements**

**PROGRAM:**

import numpy as np

a = np.arange(0, 15, 2) print(a)

print("Elements from index 2 to 8 with step 2")

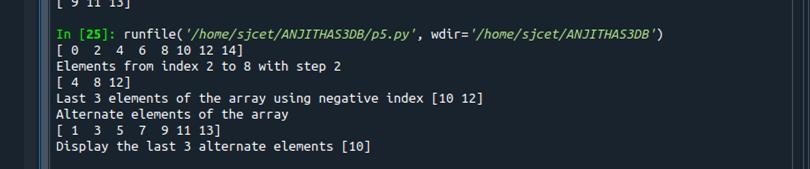
s2 = slice(2, 8, 2) print(a[s2])

print("Last 3 elements of the array using negative index",a[-3:-1]) print("Alternate elements of the array")

ab = np.arange(1, 15, 2) print(ab)

print("Display the last 3 alternate elements",a[-3:-1:2])

**OUTPUT:**



1. **Create a 2 Dimensional array with 4 rows and 4 columns.**
   1. **Display all elements excluding the first row**
   2. **Display all elements excluding the last column**
   3. **Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row**
   4. **Display the elements of 2 nd and 3 rd column**
   5. **Display 2 nd and 3 rd element of 1 st row**

**PROGRAM**

import numpy as np

x = np.array([[2, 4, 6,1], [6, 8, 10,1],[1, 2, 1,1], [1, 1, 1,1]])

print(x)

print("Display all elements excluding the first row") print(x[1:])

print("Display all elements excluding the last column") print(x[:, :3])

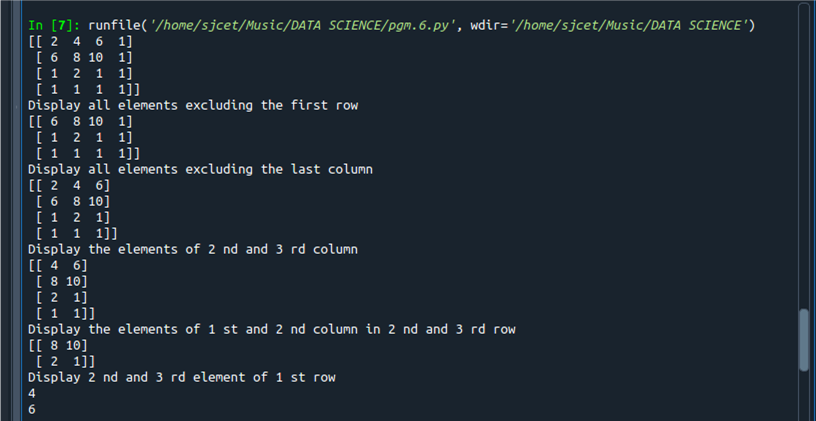
print("Display the elements of 2 nd and 3 rd column") print(x[:, 1:3])

print("Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row")

print(x[1:3,1:3])

print("Display 2 nd and 3 rd element of 1 st row") print(x[0,1])

print(x[0,2])

**OUTPUT**

1. **Create two 2D arrays using array object and**
2. **Add the 2 matrices and print it**
3. **Subtract 2 matrices**
4. **Multiply the individual elements of matrix**
5. **Divide the elements of the matrices**
6. **Perform matrix multiplication**
7. **Display transpose of the matrix**
8. **Sum of diagonal elements of a matrix**

**PROGRAM:**

import numpy as np

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 + M2

print("Matrix addition") print(M3)

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 - M2

print("Matrix Substract") print(M3)

M1 = np.array([[3, 6], [14, 21]])

M2 = np.array([[9, 27], [11, 22]])

M3 = M1 / M2

print("Divide the elements of the matrices") print(M3)

M1 = np.array([[3, 6], [5, -10]])

M2 = np.array([[9, -18], [11, 22]])

M3 = M1 \* M2

print("Multiply the individual elements of matrix") print(M3)

M1 = np.array([[3, 6], [5, -10]])

M2 = np.array([[9, -18], [11, 22]]) M3 = M1.dot(M2)

print("matrix multiplication") print(M3)

M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])

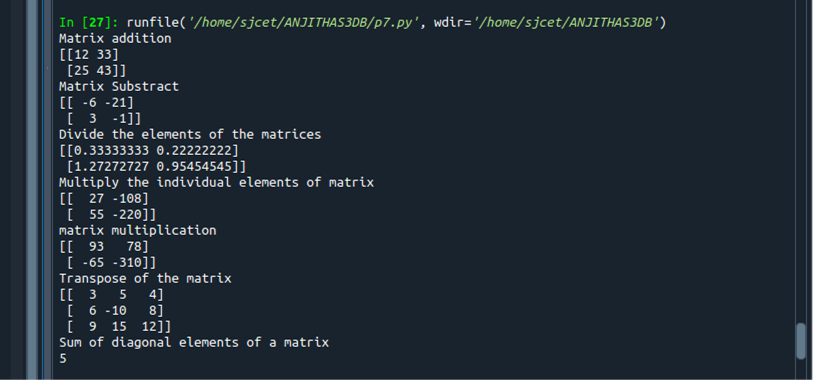
M2 = M1.transpose() print("Transpose of the matrix")

print(M2)

M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])

print("Sum of diagonal elements of a matrix") print(np.trace(M1))

**OUTPUT**



1. **Create a square matrix with random integer values(use randint()) and use appropriate functions to find:**
   1. **Inverse**
   2. **rank of matrix**
   3. **Determinant**
   4. **transform matrix into 1D array**
   5. **eigen values and vectors**

**PROGRAM**

import numpy as np

import numpy as nf

from numpy.linalg import eig

mat = np.random.randint(10, size=(3, 3))

array = nf.random.randint(10, size=(3, 3))

print(mat)

M\_inverse = np.linalg.inv(mat)

print("inverse of the array")

print(M\_inverse)

rank = np.linalg.matrix\_rank(mat)

print("Rank of the given Matrix ")

print(rank)

det= np.linalg.det(mat)

print("determinant of the given Matrix ")

print(det)

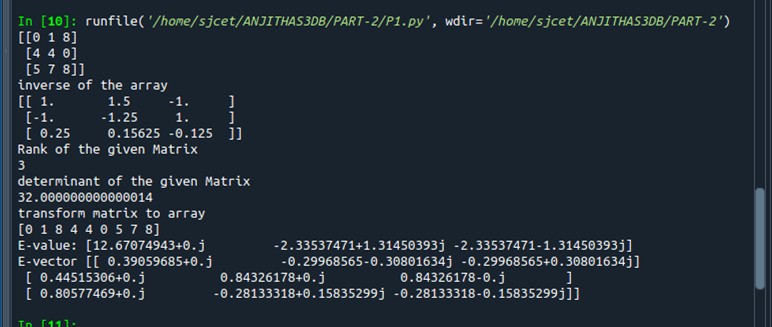
arr=mat.flatten()

print("transform matrix to array ")

print(arr) w,v=eig(array)

print('E-value:', w)

print('E-vector', v)

**OUTPUT**

1. **Create a matrix X with suitable rows and columns**
2. **Display the cube of each element of the matrix using different methods (use multiply(), \*, power(), \*\*)**
3. **Display identity matrix of the given square matrix.**
4. **Display each element of the matrix to different powers.**
5. **Create a matrix Y with same dimension as X and perform the operation X2+2Y**

**PROGRAM**

import numpy as np

matrix=np.random.randint(0,10,4).reshape(2,2)

print("Display the cube of each element of the matrix using different methods (use multiply(), \*, power(),\*\*)")

x=np.power(matrix,3)

print("power()",x)

y=np.multiply(matrix,(matrix\*matrix))

print("multiply()")

print(y)

z=matrix\*matrix\*matrix print("\*\*")

print(z)

cube=matrix\*3

print("\*")

print(cube)

print("Display identity matrix of the given square matrix.")

identity=np.identity(2,dtype=int)

print(identity)

print("Display each element of the matrix to different powers.")

dpow=np.power(matrix,matrix)

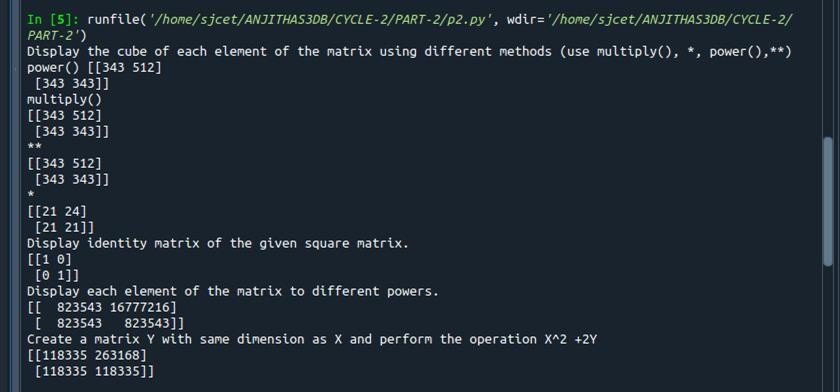
print(dpow)

print("Create a matrix Y with same dimension as X and perform the operation X^2+2Y")

a=np.add((np.power(x,2)),(np.multiply(y,2)))

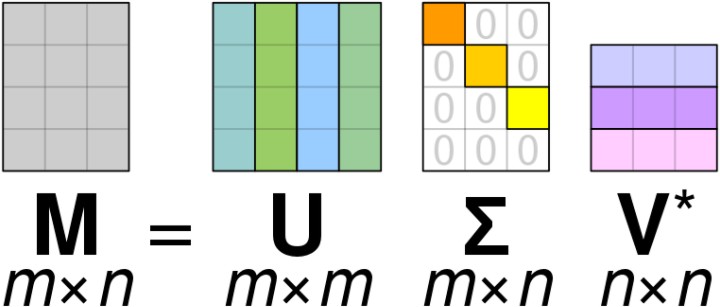
print(a)

**OUTPUT**



1. **Write a program to find out the value of X using solve(), given A and b as above Singular value Decomposition Matrix decomposition, also known as matrix fac- torization, involves describing a given matrix using its constituent elements.The Sin- gular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.**

**M= U ∑V^T**



### **M-is original matrix we want to decompose**

* **U-is left singular matrix (columns are left singular vectors). U columns contain eigenvectors of matrix MMᵗ**
* **Σ-is a diagonal matrix containing singular (eigen) values.**
* **V-is right singular matrix (columns are right singular vectors). V columns con- tain eigenvectors of matrix MᵗM**

**Numpy provides a function for performing svd, which decomposes the given matrix into 3 matrices.**

**PROGRAM**

import numpy as np

A = np.array([[2, 1, -2],

[3, 0, 1],[1, 1, -1]])

b=np.array([[3],[5],[-2]])

inv=np.linalg.inv(A)

x=np.linalg.solve(inv,b)

print(x)

**OUTPUT**



1. **Write a program to perform the SVD of a given matrix. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.**

**PROGRAM**

from numpy import array from scipy.linalg import svd from numpy import diag

from numpy import dot from numpy import zeros

A = array([[1, 2], [3, 4],[5,6]])

print(A)

# SVD

U, s, VT = svd(A)

print("first" ,U)

print("second",s)

print("3rd" ,VT)

Sigma = zeros((A.shape[0], A.shape[1]))

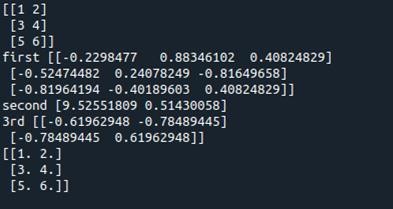
# populate Sigma with n x n diagonal matrix

Sigma[:A.shape[1], :A.shape[1]] = diag(s)

# reconstruct matrix

B = U.dot(Sigma.dot(VT))

print(B)

**OUTPUT**

**CYCLE 3**

1. **Sarah bought a new car in 2001 for $24,000. The dollar value of her car**

|  |  |  |
| --- | --- | --- |
| **Value Year** | **changed each year as shown in the**  **of Sarah's Car Value** | **table below.** |
| **2001** | **$24,000** |  |
| **2002** | **$22,500** |  |
| **2003** | **$19,700** |  |
| **2004** | **$17,500** |  |
| **2005** | **$14,500** |  |
| **2006** | **$10,000** |  |
| **2007** | **$ 5,800** |  |
|  |  |  |

**Represent the following information using a line graph with following style properties**

**X- axis – Year, Y –axis - Car Value, title –Value Depreciation (left Aligned)**

**Line Style dashdot and Line-color should be red, point using \* symbol with green color and size 20 . Subplot() provides multiple plots in one figure.**

**PROGRAM:**

import matplotlib.pyplot as plt

import numpy as np

xpoints = np.array([2001, 2002,2003,2004,2005,2006,2007])

ypoints = np.array([24000, 22500,19700,17500,14500,10000,5800])

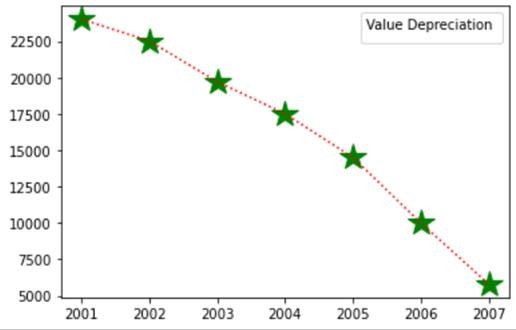
plt.plot(xpoints, ypoints, '\*g',ms = 20)

plt.plot(xpoints, ypoints, ':r')

leg = plt.legend(title="Value Depreciation ")

leg.\_legend\_box.align = "left"

plt.show()



1. **Following table gives the daily sales of the following items in a shop**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mon** | **Tues** | **Wed** | **Thurs** | **Fri** |
| **300** | **450** | **150** | **400** | **650** |
| **400** | **500** | **350** | **300** | **500** |

**Use subplot function to draw the line graphs with grids(color as blue and line style dotted) for the above information as 2 separate graphs in two rows**

* **Properties for the Graph 1:**
  + **X label- Days of week**
  + **Y label-Sale of Drinks**
  + **Title-Sales Data1 (right aligned)**
  + **Line –dotted with cyan color**
  + **Points- hexagon shape with color magenta and outline black**
* **Properties for the Graph 2:**
  + **X label- Days of Week**
  + **Y label-Sale of Food**
  + **Title-Sales Data2 ( center aligned)**
  + **Line –dashed with yellow color**

**Points- diamond shape with color green and outline red**

**PROGRAM**

## import matplotlib.pyplot as plt

## import numpy as np

#plot 1:

x = np.array(['mon', 'tue', 'wed', 'thur','fri'])

y = np.array([300, 450, 150, 400,65])

plt.subplot(1, 2, 1)

plt.title("Sales Data1")

plt.xlabel("Days of week")

plt.ylabel("Sale of Drinks")

plt.plot(x,y,':c')

plt.plot(x,y,'Hm',mec = 'k')

plt.grid(color = 'blue', linestyle = 'dotted')

#plot 2:

c = np.array(['mon', 'tue', 'wed', 'thur','fri'])

v = np.array([400, 500, 350, 300,500])

plt.subplot(1, 2, 2) plt.title("Sales Data2")

plt.xlabel("Days of Week")

plt.ylabel("Sale of Food")

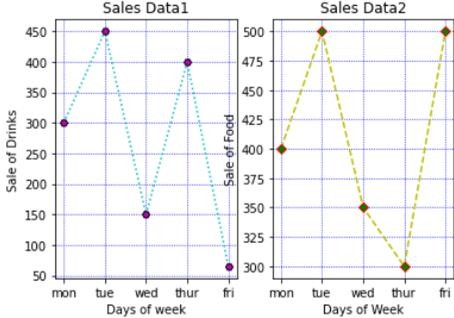
plt.plot(c,v,'--y')

plt.plot(c,v,'Dg',mec = 'r')

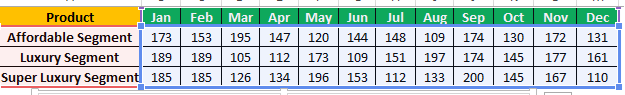
plt.grid(color = 'blue', linestyle = 'dotted')

plt.show()

**OUTPUT**



1. **Create scatter plot for the below data:(use Scatter function)**



**Create scatter plot for each Segment with following properties within one**

* **X Label- Months of Year with font size 18**
* **Y-Label- Sales of Segments**
* **Title –Sales Data**
* **Color for Affordable segment- pink**
* **Color for Luxury Segment- Yellow**
* **Color for Super luxury segment-blue**

**PROGRAM**

import matplotlib.pyplot as plt import numpy as np

plt.title("Sales Data")

plt.xlabel("Months of Year"size=18) plt.ylabel("Sale of Food")

x = np.array(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']) y1 = np.array([173,153,195,147,120,144,148,109,174,130,172,131])

plt.scatter(x,y1, color = 'hotpink')

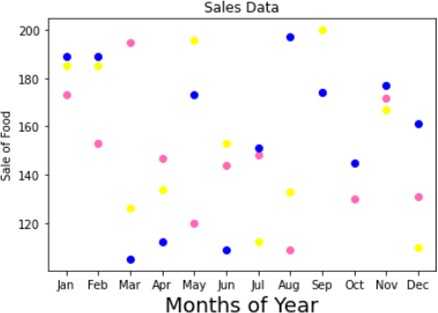
y2 = np.array([185,185,126,134,196,153,112,133,200,145,167,110])

plt.scatter(x, y2, color = 'yellow')

y3 = np.array([189,189,105,112,173,109,151,197,174,145,177,161])

plt.scatter(x, y3, color = 'blue') plt.show()

**OUTPUT**



1. **Display the above data using multiline plot( 3 different lines in same graph)**

* **Display the description of the graph in upper right corner(use legend())**
* **Use different colors and line styles for 3 different lines**

**PROGRAM**

import matplotlib.pyplot as plt import numpy as np

x = np.array(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])

y1 = np.array([173,153,195,147,120,144,148,109,174,130,172,131])

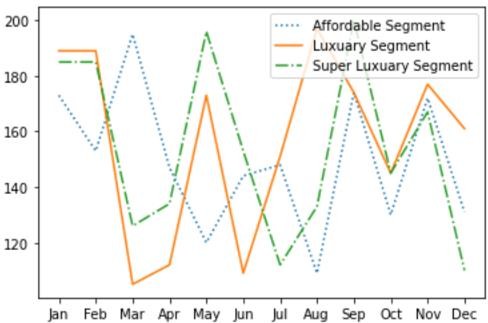
y2= np.array([189,189,105,112,173,109,151,197,174,145,177,161])

y3= np.array([185,185,126,134,196,153,112,133,200,145,167,110])

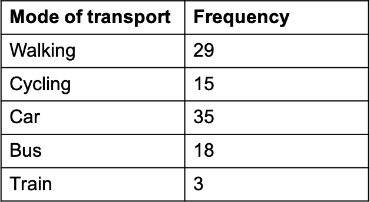
plt.plot(x,y1,label = "Affordable Segment",ls=':') plt.plot(x,y2,label = "Luxuary Segment",ls="-") plt.plot(x,y3,label = "Super Luxuary Segment",ls="-.") plt.legend()

plt.show()

**OUTPUT**



1. **100 students were asked what their primary mode of transport for getting to school was. The results of this survey are recorded in the table below. Construct a bar graph representing this information.**



**Create a bar graph with**

* **X axis -mode of Transport and Y axis ‘frequency’**
* **Provide appropriate labels and title**
* **Width .1, color green**

**PROGRAM**

## import matplotlib.pyplot as plt

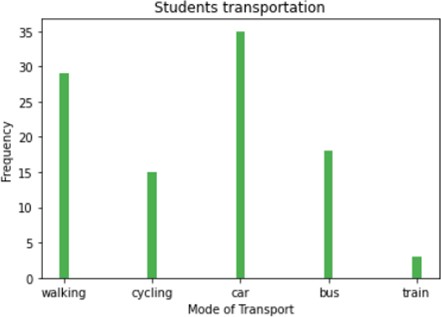
## import numpy as np

## plt.title("Students transportation") plt.xlabel("Mode of Transport") plt.ylabel("Frequency")

x = np.array(["walking","cycling","car","bus","train"])

y = np.array([29,15,35,18,3])

plt.bar(x, y, color = "#4CAF50",width = 0.1) plt.show()

**OUTPUT**

1. **We are provided with the height of 30 cherry trees. The height of the trees (in inches): 61, 63, 64, 66, 68, 69, 71, 71.5, 72, 72.5,73,73.5, 74, 74.5, 76, 76.2,76.5, 77, 77.5, 78, 78.5, 79, 79.2, 80, 81, 82, 83, 84, 85,87. Create a histogram with a bin size of 5**

**PROGRAM**

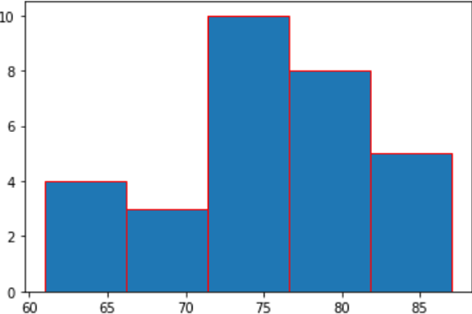
import matplotlib.pyplot as plt

height=[61,63,64,66,68,69,71,71.5,72,72.5,73,73.5,74,74.5,76,76.2,76.5,77,77.5,78,78.5,79,79.2,80,81,82,83,84,85,87]

plt.hist(height, edgecolor="red", bins=5)

plt.show()

**OUTPUT**



**Data Handling using ‘Pandas’ and Data Visualization using ‘Seaborn’**

**7. Use appropriate functions in pandas to display**

**1.Shap of the data set**

2. **First 5 and last five rows of data set(head and tail)**

**3.Size of dataset**

**4.No:of samples available for each variety**

**5.Description of the data set( use describe**

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

import pandas as pd

col=['sepal\_length','sepal\_width','petal\_length','petal\_width','type'] iris=pd.read\_csv("iris.csv",names=col)

print("First five rows") print(iris.head()) print("\*\*\*\*\*\*\*\*\*")

print("columns",iris.columns) print("\*\*\*\*\*\*\*\*\*")

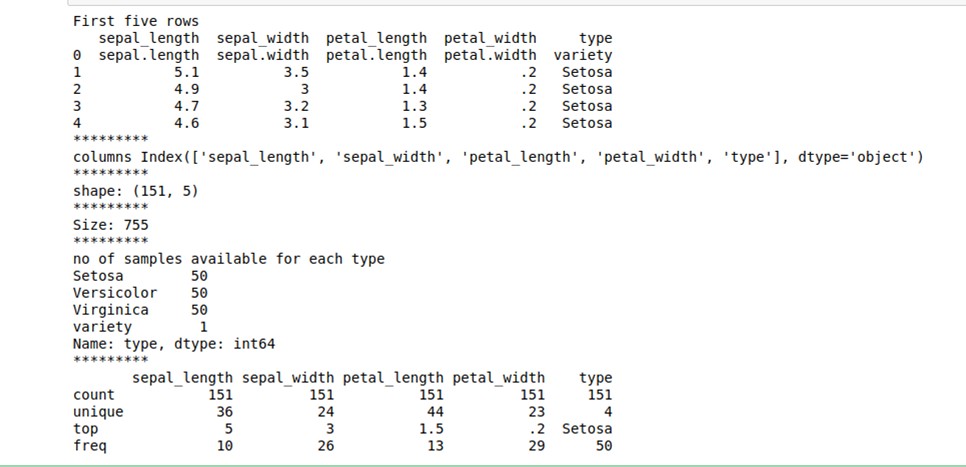
print("shape:",iris.shape) print("\*\*\*\*\*\*\*\*\*")

print("Size:",iris.size) print("\*\*\*\*\*\*\*\*\*")

print("no of samples available for each type")

print(iris["type"].value\_counts()) print("\*\*\*\*\*\*\*\*\*")

print(iris.describe())

**OUTPUT**

1. **Use pairplot() function to display pairwise relationships between attributes. Try different kind of plots {*‘scatter’, ‘kde’, ‘hist’, ‘reg’}* and different kind of markers**

**PROGRAM**

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

iris = sns.load\_dataset('iris')

my\_data\_frame = pd.DataFrame(iris)

g = sns.pairplot(my\_data\_frame)

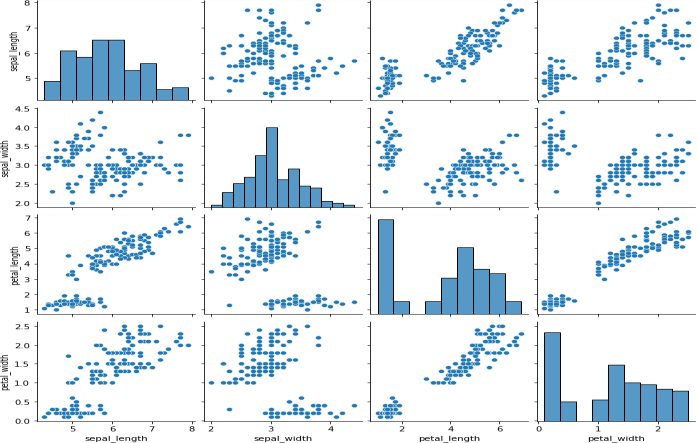
g = sns.pairplot(iris, kind="reg", hue="species")

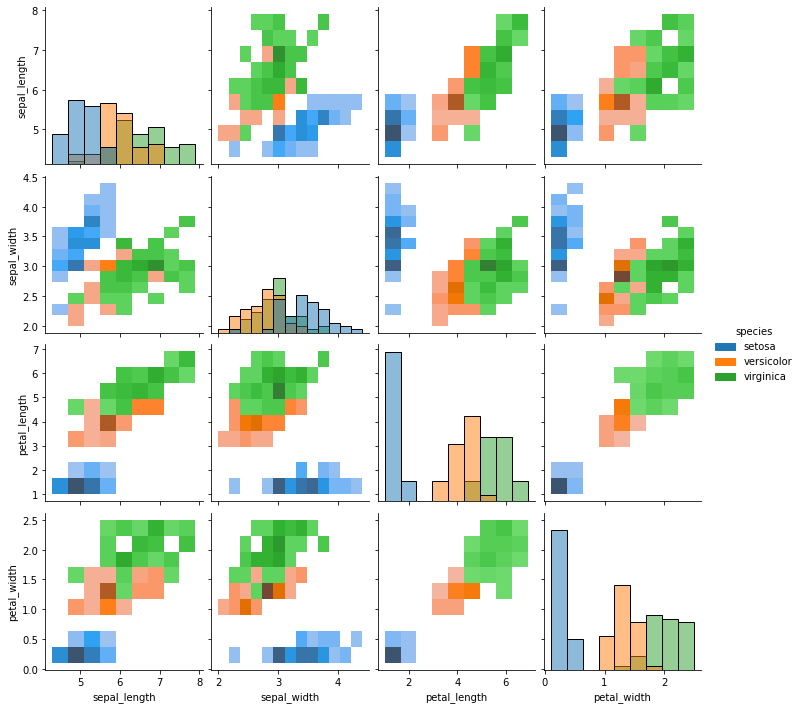
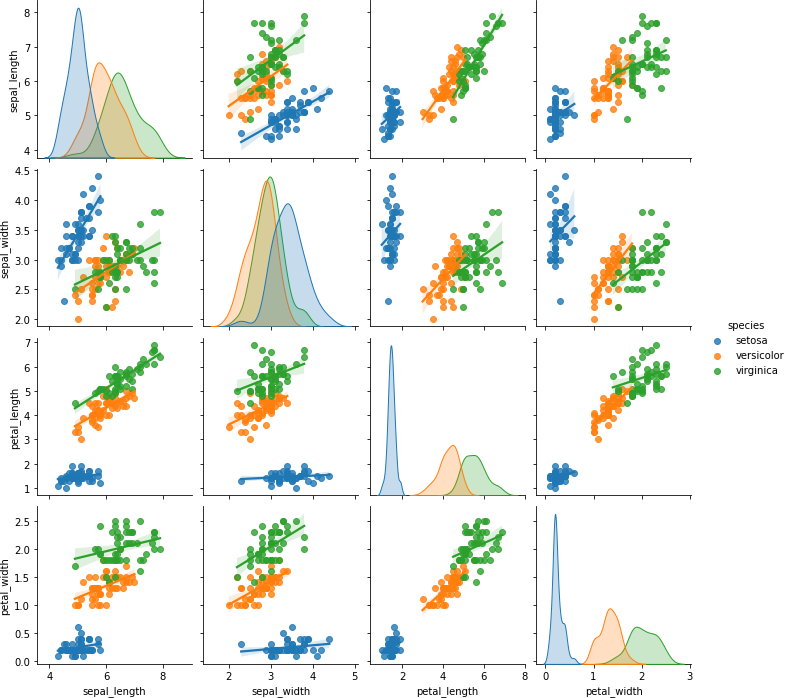
g = sns.pairplot(iris, kind="hist", hue="species")

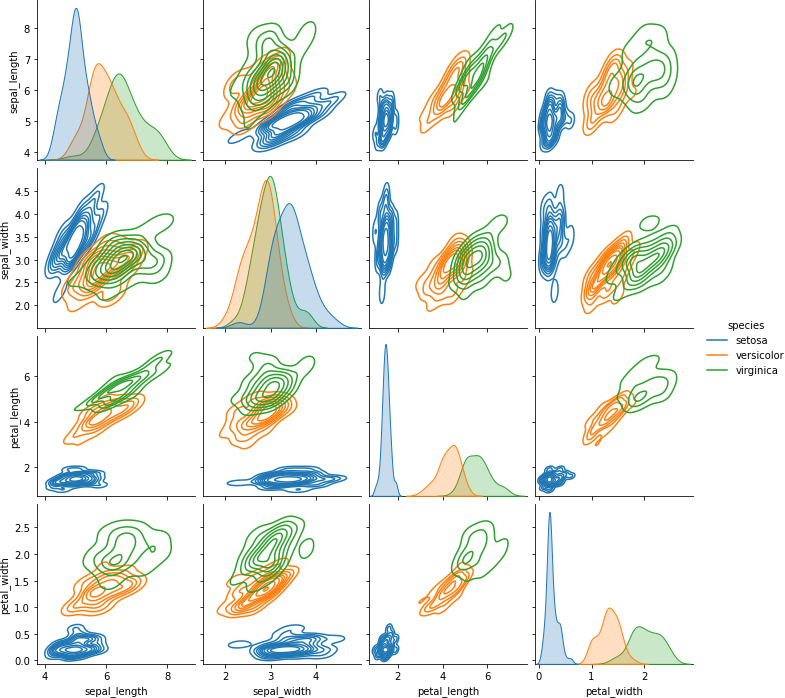
g = sns.pairplot(iris, kind="kde", hue="species")

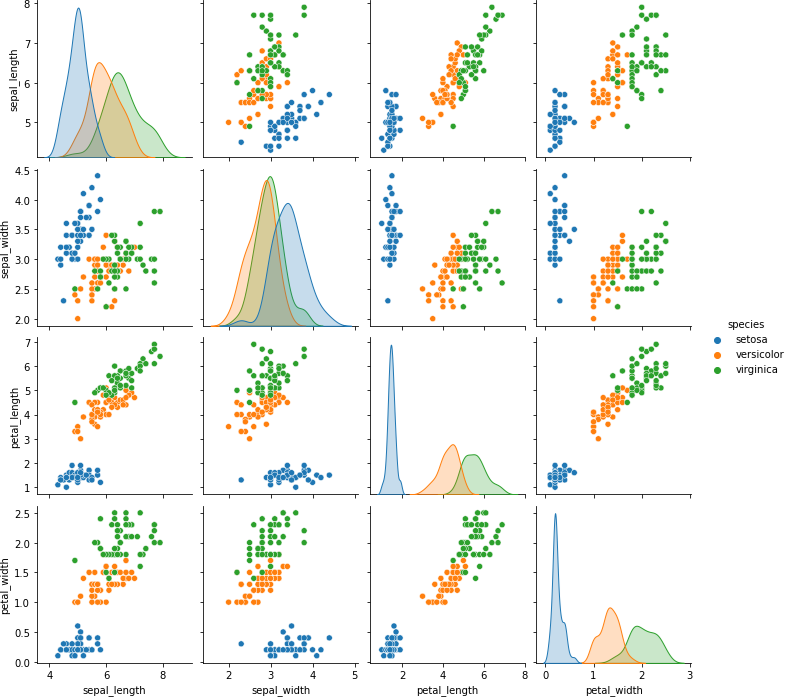
g = sns.pairplot(iris, kind="scatter", hue="species")

**OUTPUT**









1. **Using the iris data set,get familiarize with functions: 1)displot()**

**PROGRAM**

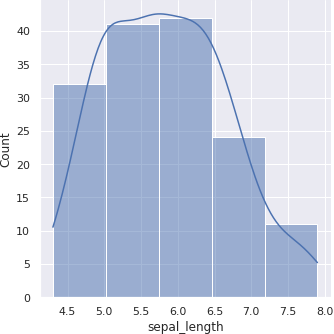
import seaborn as sns import pandas

Import matplotlib.pyplot as plt sns.get\_dataset\_names()

df = sns.load\_dataset('iris') df.head()

sns.displot(x = 'sepal\_length',kde=True,bins = 5 , data =df)

**OUTPUT**



1. **histplot()**

**PROGRAM**

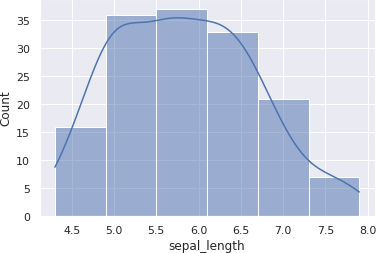
import seaborn as sns import pandas

import matplotlib.pyplot as plt sns.get\_dataset\_names()

df = sns.load\_dataset('iris')

df.head()

sns.histplot(x = 'sepal\_length',kde=True,bins = 6 , data =df)

**OUTPUT**

**3)relplot()**

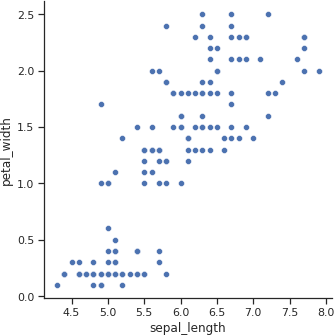
**PROGRAM**

import seaborn as sns

sns.set(style ="ticks")

dataset = sns.load\_dataset('iris') col=['sepal\_length','sepal\_width','petal\_length','petal\_width','type']

sns.relplot(x ="sepal\_length", y ="petal\_width", data = dataset)

**OUTPUT**

**CYCLE 4**

1. **KNN Algorithm**

**Using the iris data set implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k. Also find the accuracy level.**

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

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

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

from sklearn.neighbors import KNeighborsClassifier

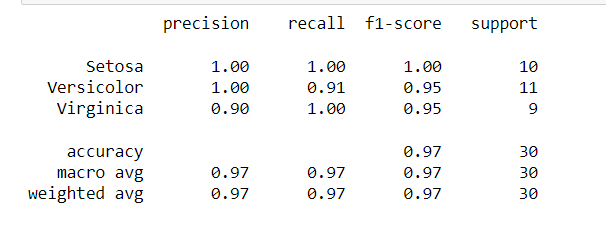
classifier = KNeighborsClassifier(n\_neighbors=5)

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

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

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

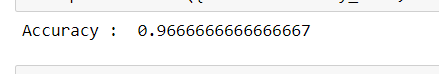
print(classification\_report(y\_test, y\_pred))



from sklearn.metrics import accuracy\_score

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

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})



1. **Download another data set suitable for the KNN and implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k.**

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("cancer.csv")

dataset.head()

dataset.info()

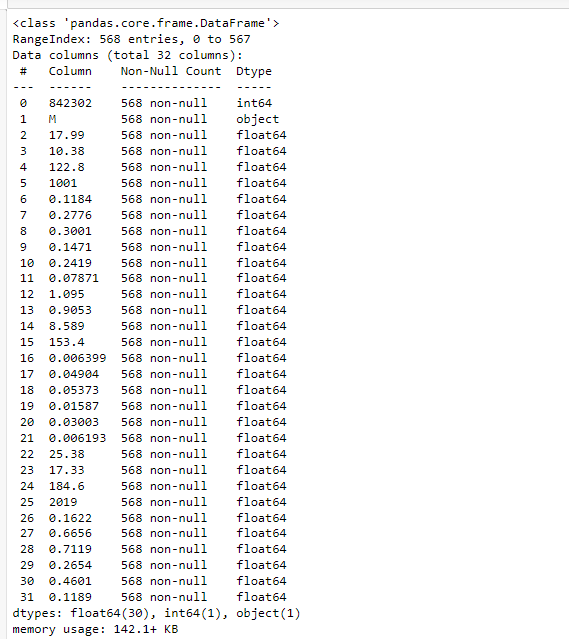
X = dataset.iloc[:, 2:35].values

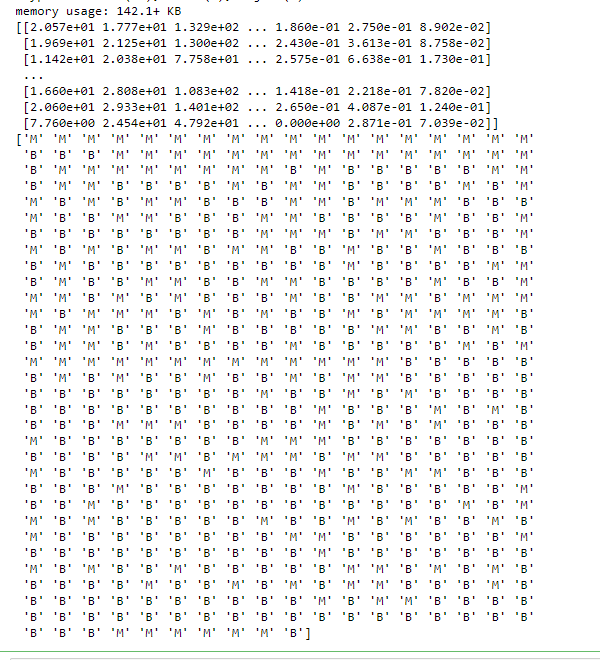
print(X)

y = dataset.iloc[:, 1].values

print(y)

**OUTPUT:**





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

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

from sklearn.neighbors import KNeighborsClassifier

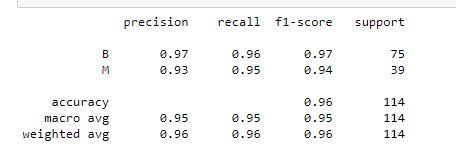
classifier = KNeighborsClassifier(n\_neighbors=5)

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

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

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

print(classification\_report(y\_test, y\_pred))



from sklearn.metrics import accuracy\_score

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

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})



1. **Naive Bayes Classification Algorithm**

**Using iris data set, implement naive bayes classification for different naive Bayes classification algorithms. ((i) gaussian (ii) bernoulli etc)**

* **Find out the accuracy level w.r.t to each algorithm**
* **Display the no:of mislabeled classification from test data set**
* **List out the class labels of the mismatching records**

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

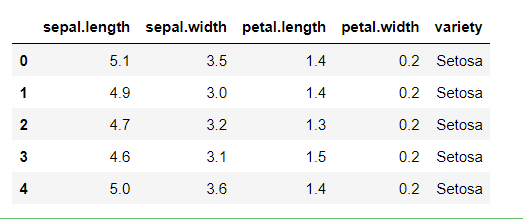
import pandas as pd

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

X = dataset.iloc[:,:4].values

y = dataset['variety'].values

dataset.head(5)



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

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

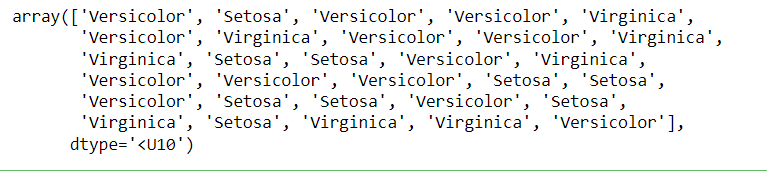
from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

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

y\_pred



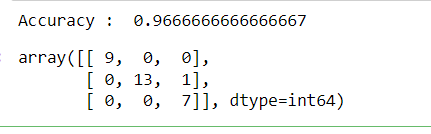
from sklearn.metrics import confusion\_matrix

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

from sklearn.metrics import accuracy\_score

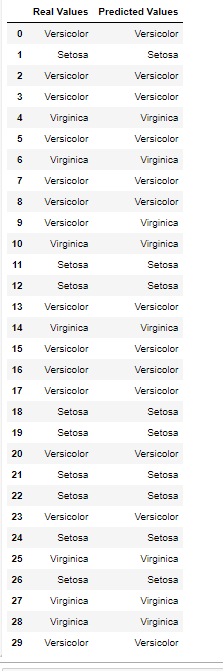
print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

cm



df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

df



1. **Decision Tree Algorithm**

**Use car details CSV file and implement decision tree algorithm**

* 1. **Find out the accuracy level.**
  2. **Display the no: of mislabelled classification from test data set**
  3. **List out the class labels of the mismatching records**

**PROGRAM**

import os

import numpy as np

import pandas as pd

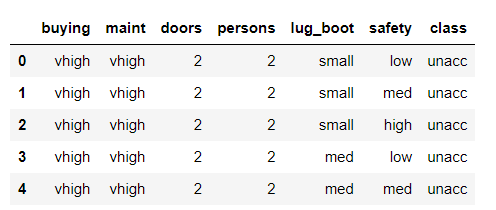
import numpy as np, pandas as pd

import matplotlib.pyplot as plt

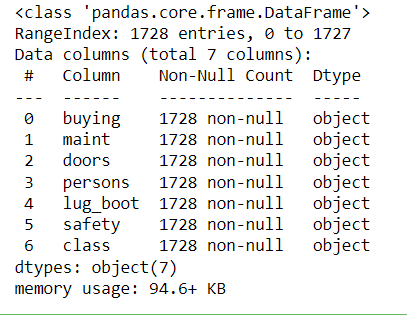
from sklearn import tree, metrics, model\_selection

data = pd.read\_csv('car.csv',names=['buying','maint','doors','persons','lug\_boot','safety','class'])

data.head()



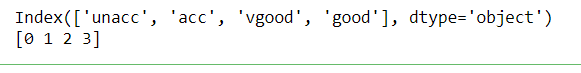
data.info()



data['class'],class\_names = pd.factorize(data['class'])

print(class\_names)

print(data['class'].unique())



data['buying'],\_ = pd.factorize(data['buying'])

data['maint'],\_ = pd.factorize(data['maint'])

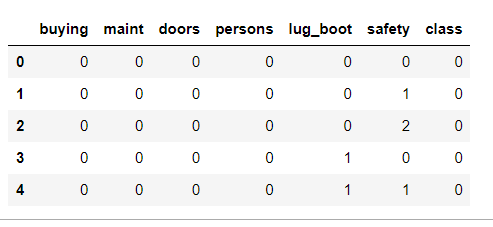
data['doors'],\_ = pd.factorize(data['doors'])

data['persons'],\_ = pd.factorize(data['persons'])

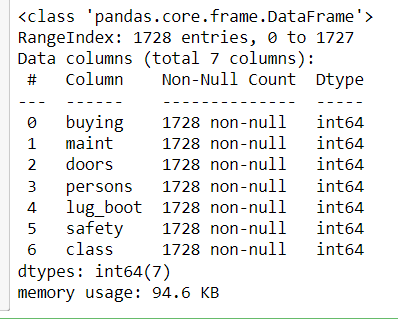
data['lug\_boot'],\_ = pd.factorize(data['lug\_boot'])

data['safety'],\_ = pd.factorize(data['safety'])

data.head()



data.info()



X = data.iloc[:,:-1]

y = data.iloc[:,-1]

# split data randomly into 70% training and 30% test

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.3, random\_state=0)

# train the decision tree

dtree = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=3, random\_state=0)

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



# use the model to make predictions with the test data

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

# how did our model perform?

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print('Accuracy: {:.2f}'.format(accuracy))



count\_misclassified = (y\_test != y\_pred).sum()

print('Misclassified samples: {}'.format(count\_misclassified))



1. **Implement Simple and multiple linear regression for the data sets ‘student\_score.csv’ and ‘company\_data .csv’ respectively**

**Simple Linear Regression**

**PROGRAM**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

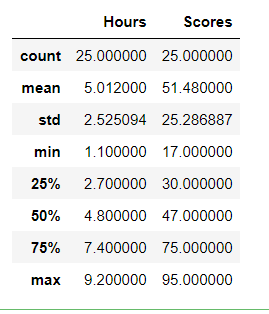
#data set contains details of no.of hours spend by students for studt and their marks

student = pd.read\_csv('student\_scores.csv')

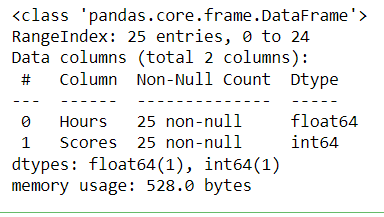
student.head()



student.describe()



student.info()



import matplotlib.pyplot as plt

Xax=student.iloc[:,0]

Yax=student.iloc[:,1]

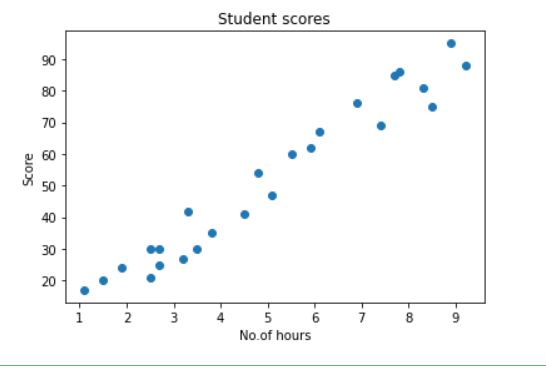
plt.scatter(Xax,Yax)

plt.xlabel("No.of hours")

plt.ylabel("Score")

plt.title("Student scores")

plt.show()



#Perform the simple linear regression model

#Equation: Y=w0+w1.x

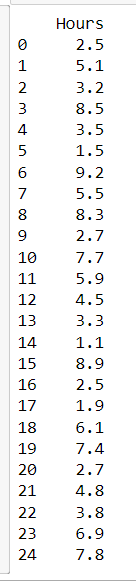
#Here Y(marks)=w0+w1.x

#Create x as hours and Y as marks

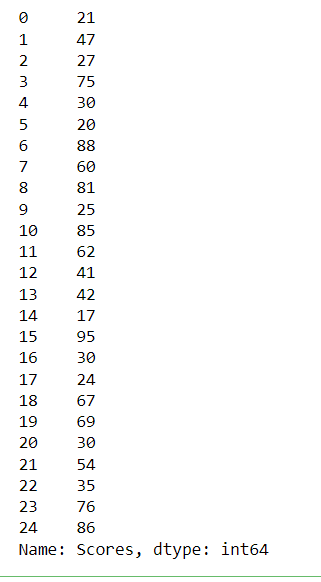
X = student.iloc[:, :-1]

y = student.iloc[:, 1]

print(X)



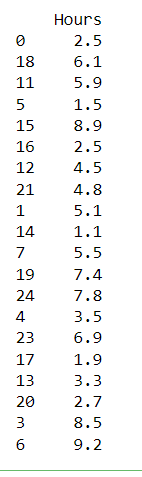
print(y)



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

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

print(X\_train)



from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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



print(regressor.intercept\_)



print(regressor.coef\_)



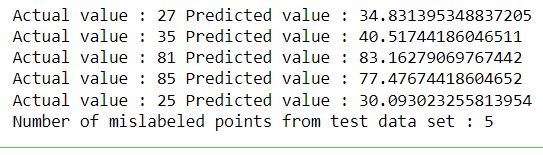
y\_pred = regressor.predict(X\_test)

for(i,j) in zip(y\_test,y\_pred):

if i!=j:

print("Actual value :",i,"Predicted value :",j)

print("Number of mislabeled points from test data set :", (y\_test != y\_pred).sum())

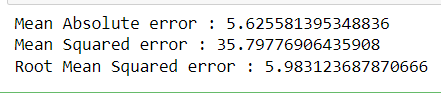


from sklearn import metrics

print("Mean Absolute error :", metrics.mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared error :", metrics.mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared error :", np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))



import matplotlib.pyplot as plt

c=X\_test['Hours'].count()

xax=np.arange(c)

print(xax)

X\_axis = np.arange(len(xax))

plt.bar(X\_axis-0.2, y\_test, 0.6, label='Actual')

plt.bar(X\_axis+0.2, y\_pred, 0.6, label='Predicted')

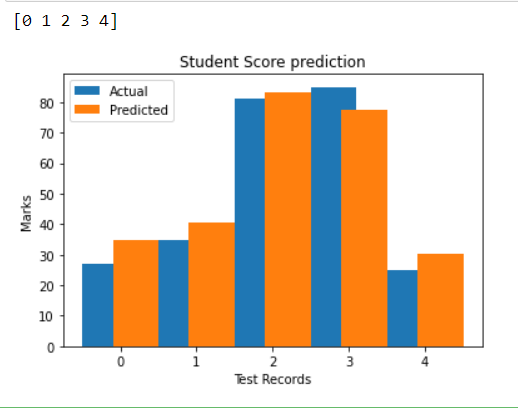
plt.xlabel("Test Records")

plt.ylabel("Marks")

plt.title("Student Score prediction")

plt.legend()

plt.show()



1. **Multiple Linear Regression**

**PROGRAM**

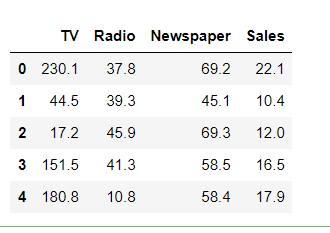
import pandas as pd

import numpy as np

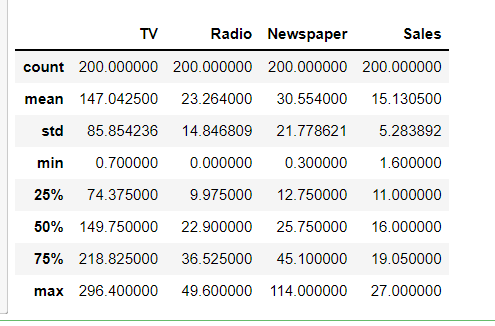
import matplotlib.pyplot as plt

advertising = pd.read\_csv('Company\_data.csv')

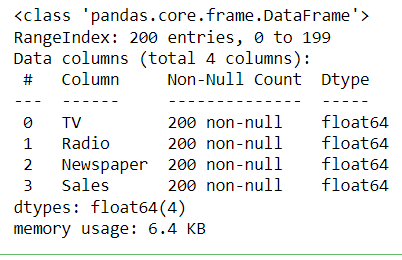
advertising.head()



advertising.describe()



advertising.info()



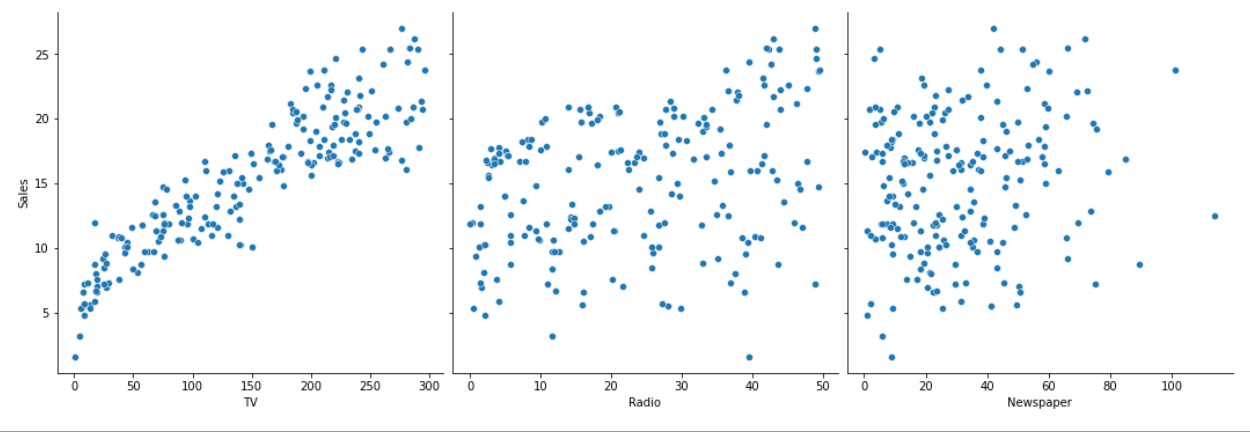
import matplotlib.pyplot as plt

import seaborn as sns

sns.pairplot(advertising, x\_vars=['TV', 'Radio', 'Newspaper'],

y\_vars='Sales', height=5, aspect=1, kind='scatter')

plt.show()



#perform the multiple linear regression model

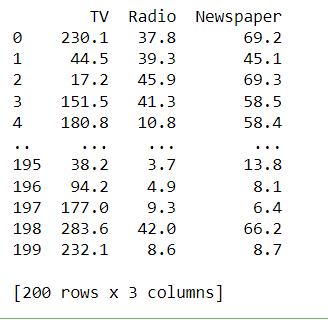
#Equation : Y=w0+w1.x1 + w2.x2 + w3.x3

#Here Y(sales)=w0+w1.x1(TV)+w2.x2(Radio)+w3.x3(Newspaper)

#create x and Y as sales

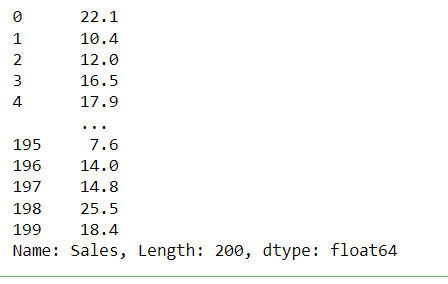
X = advertising.iloc[:, :-1]

print(X)



y = advertising.iloc[:, -1]

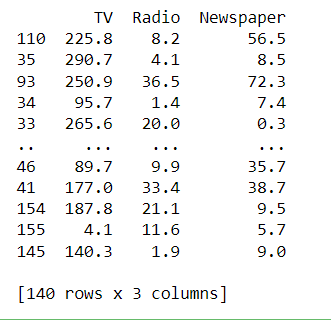
print(y)



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

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

print(X\_train)



from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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



print(regressor.intercept\_)



print(regressor.coef\_)



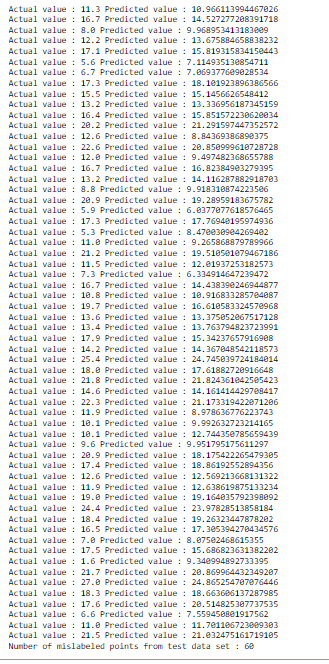
y\_pred = regressor.predict(X\_test)

for(i,j) in zip(y\_test,y\_pred):

if i!=j:

print("Actual value :",i,"Predicted value :",j)

print("Number of mislabeled points from test data set :", (y\_test != y\_pred).sum())



from sklearn import metrics

print("Mean Absolute error :", metrics.mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared error :", metrics.mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared error :", np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))



import matplotlib.pyplot as plt

c=X\_test['TV'].count()

xax=np.arange(c)

print(xax)

X\_axis = np.arange(len(xax))

plt.bar(X\_axis-0.2, y\_test, 0.6, label='Actual')

plt.bar(X\_axis+0.2, y\_pred, 0.6, label='Predicted')

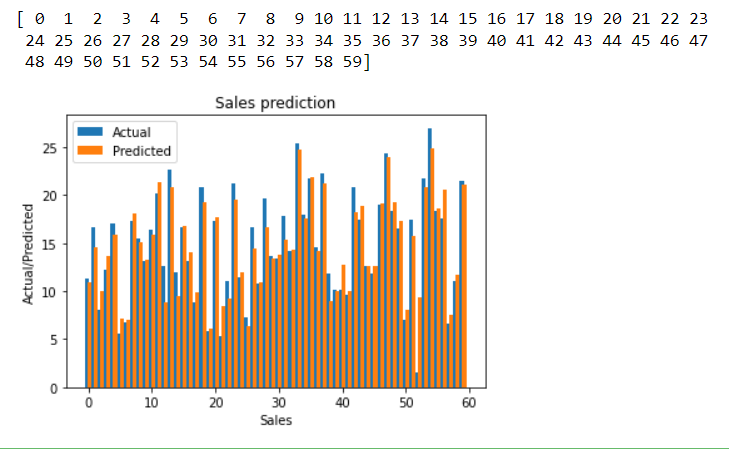
plt.xlabel("Sales")

plt.ylabel("Actual/Predicted")

plt.title("Sales prediction")

plt.legend()

plt.show()



1. **Neural Networks**

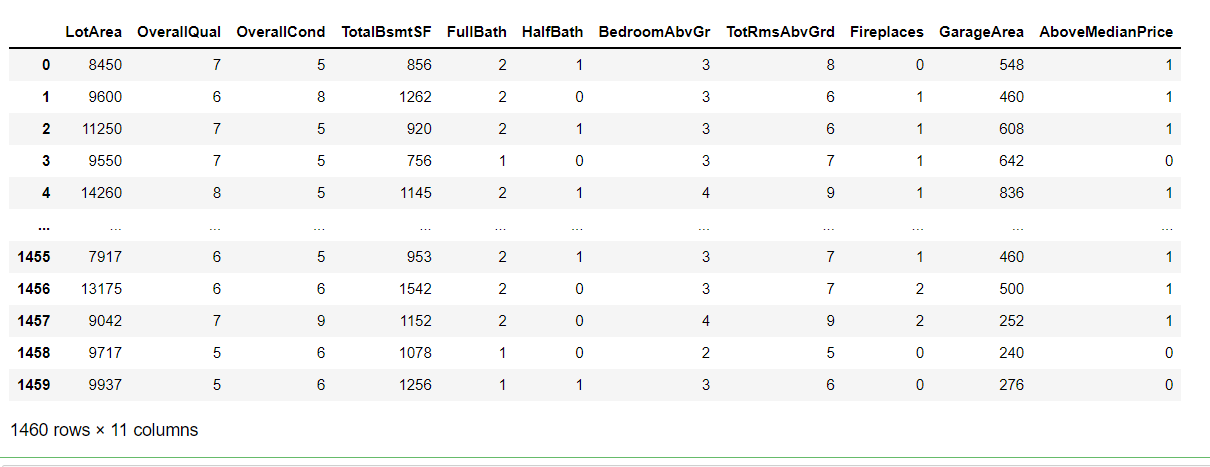
**Create a neural network for the given ‘houseprice.csv’ to predict the whether price of the house is above or below median value or not**

**PROGRAM:**

import pandas as pd

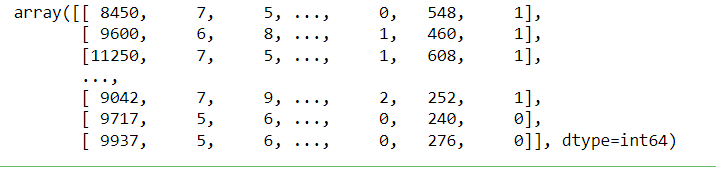
df = pd.read\_csv('housepricedata.csv')

df



dataset = df.values

dataset



X = dataset[:,0:10]

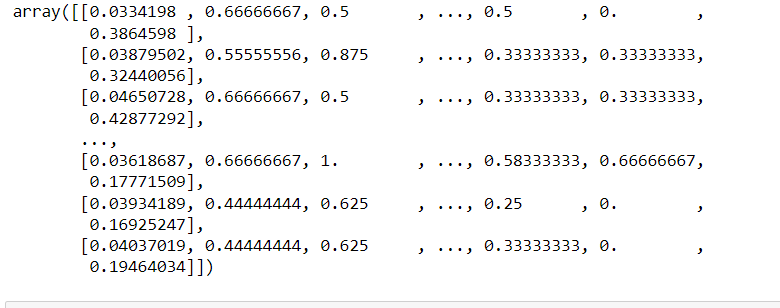
Y = dataset[:,10]

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler()

X\_scale = min\_max\_scaler.fit\_transform(X)

X\_scale



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

X\_train, X\_val\_and\_test, Y\_train, Y\_val\_and\_test = train\_test\_split(X\_scale, Y, test\_size=0.3)

X\_val, X\_test, Y\_val, Y\_test = train\_test\_split(X\_val\_and\_test, Y\_val\_and\_test, test\_size=0.5)

print(X\_train.shape, X\_val.shape, X\_test.shape, Y\_train.shape, Y\_val.shape, Y\_test.shape)



from keras.models import Sequential

from keras.layers import Dense

model = Sequential([

Dense(32, activation='relu', input\_shape=(10,)),

Dense(32, activation='relu'),

Dense(1, activation='sigmoid'),

])

model.compile(optimizer='sgd',

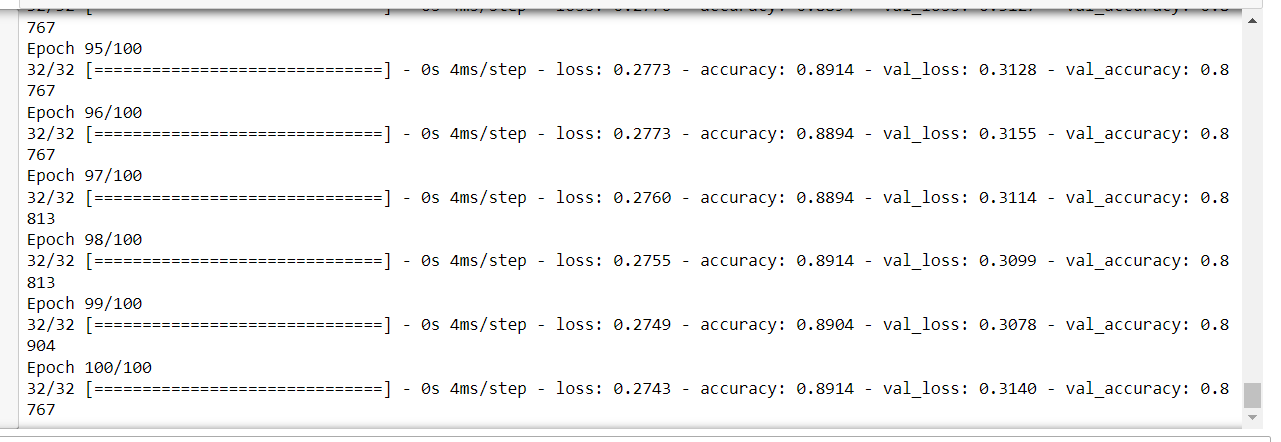
loss='binary\_crossentropy',

metrics=['accuracy'])

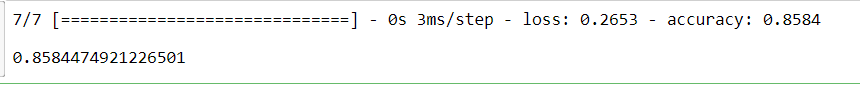
hist = model.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



model.evaluate(X\_test, Y\_test)[1]



import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])

plt.plot(hist.history['val\_loss'])

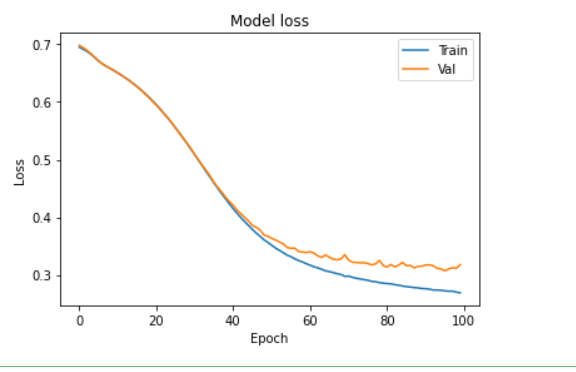
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()



plt.plot(hist.history['accuracy'])

plt.plot(hist.history['val\_accuracy'])

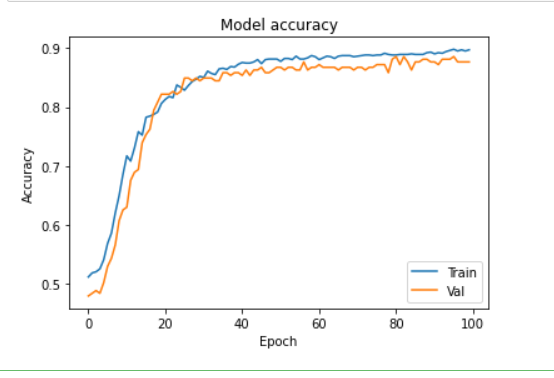
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()



model\_2 = Sequential([

Dense(1000, activation='relu', input\_shape=(10,)),

Dense(1000, activation='relu'),

Dense(1000, activation='relu'),

Dense(1000, activation='relu'),

Dense(1, activation='sigmoid'),

])

model\_2.compile(optimizer='adam',

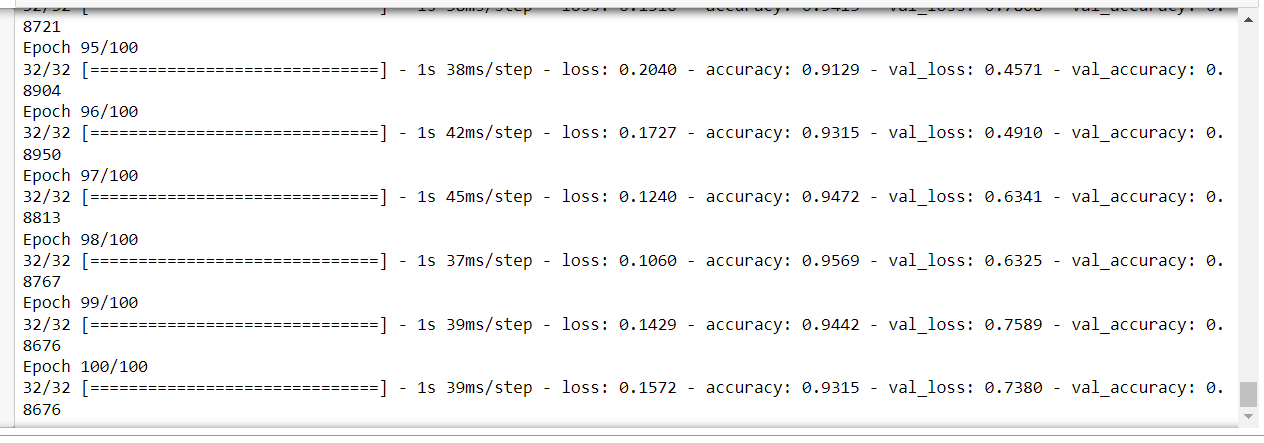
loss='binary\_crossentropy',

metrics=['accuracy'])

hist\_2 = model\_2.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



plt.plot(hist\_2.history['loss'])

plt.plot(hist\_2.history['val\_loss'])

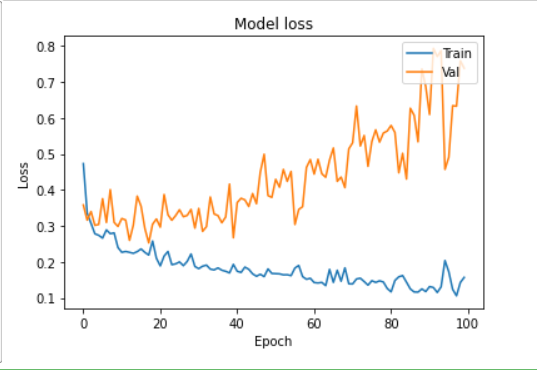
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()



plt.plot(hist\_2.history['accuracy'])

plt.plot(hist\_2.history['val\_accuracy'])

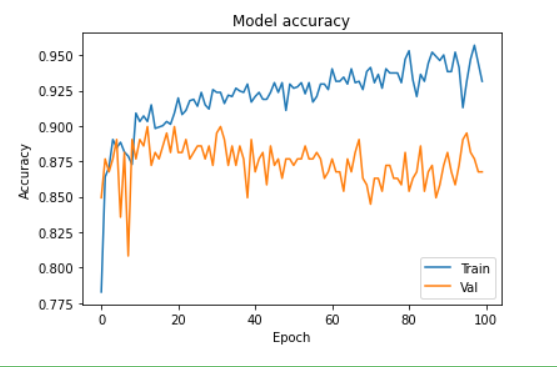
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()



from keras.layers import Dropout

from keras import regularizers

model\_3 = Sequential([

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01), input\_shape=(10,)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1, activation='sigmoid', kernel\_regularizer=regularizers.l2(0.01)),

])

model\_3.compile(optimizer='adam',

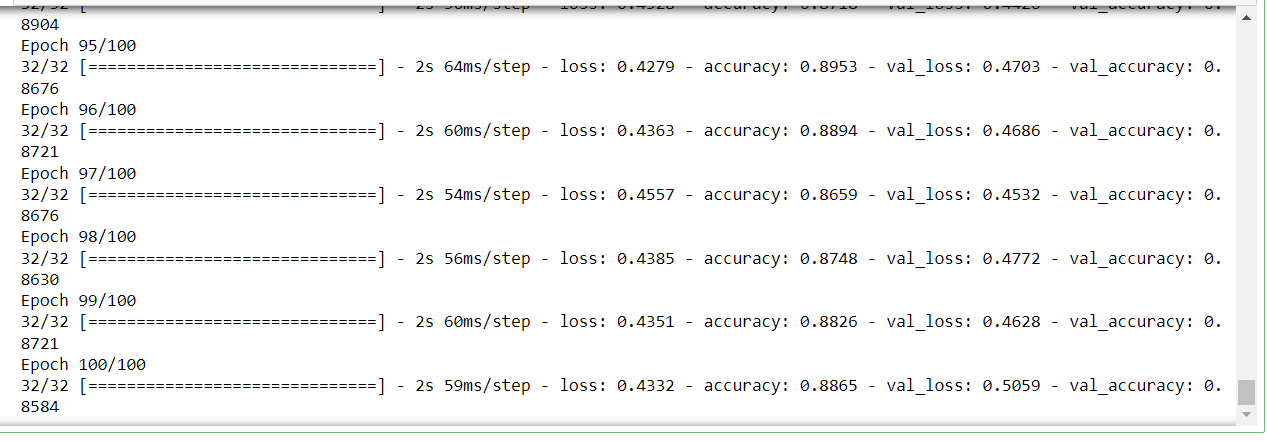
loss='binary\_crossentropy',

metrics=['accuracy'])

hist\_3 = model\_3.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



plt.plot(hist\_3.history['loss'])

plt.plot(hist\_3.history['val\_loss'])

plt.title('Model loss')

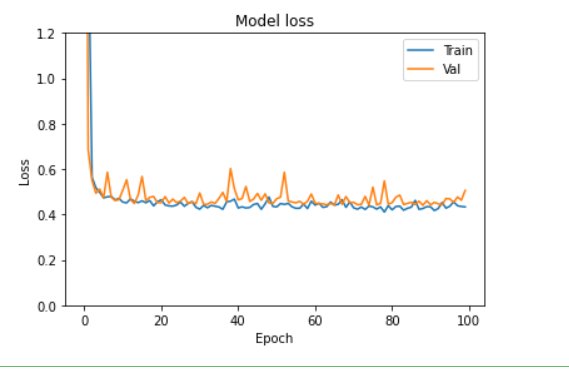
plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.ylim(top=1.2, bottom=0)

plt.show()



plt.plot(hist\_3.history['accuracy'])

plt.plot(hist\_3.history['val\_accuracy'])

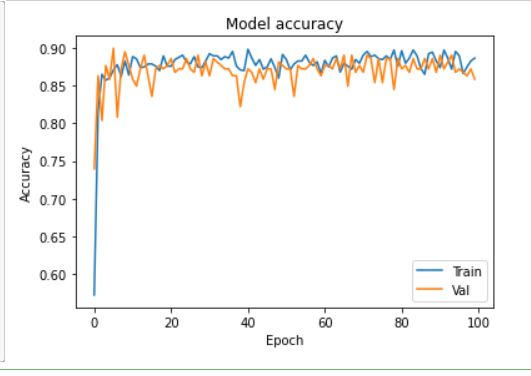
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()



**CYCLE 5**

Given a data set of support tickets. Each ticket also has an associated "urgency score" of between 0 and 3, and where 0 is "very urgent" and 3 is "not urgent". It would be useful if we could have a machine guess how urgent a ticket is, based on the description, so the urgent tickets can be resolved first

1. **For the given data set, perform text classification using SVM and find out the accuracy of the model.**

**PROGRAM:**

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

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

from sklearn.svm import LinearSVC

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

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

tickets = f.read().strip().split("\n")

with open("labels\_4.txt") as f:

labels = f.read().strip().split("\n")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tickets, labels, test\_size=0.1, random\_state=1337)

vectorizer = CountVectorizer()

svm = LinearSVC()

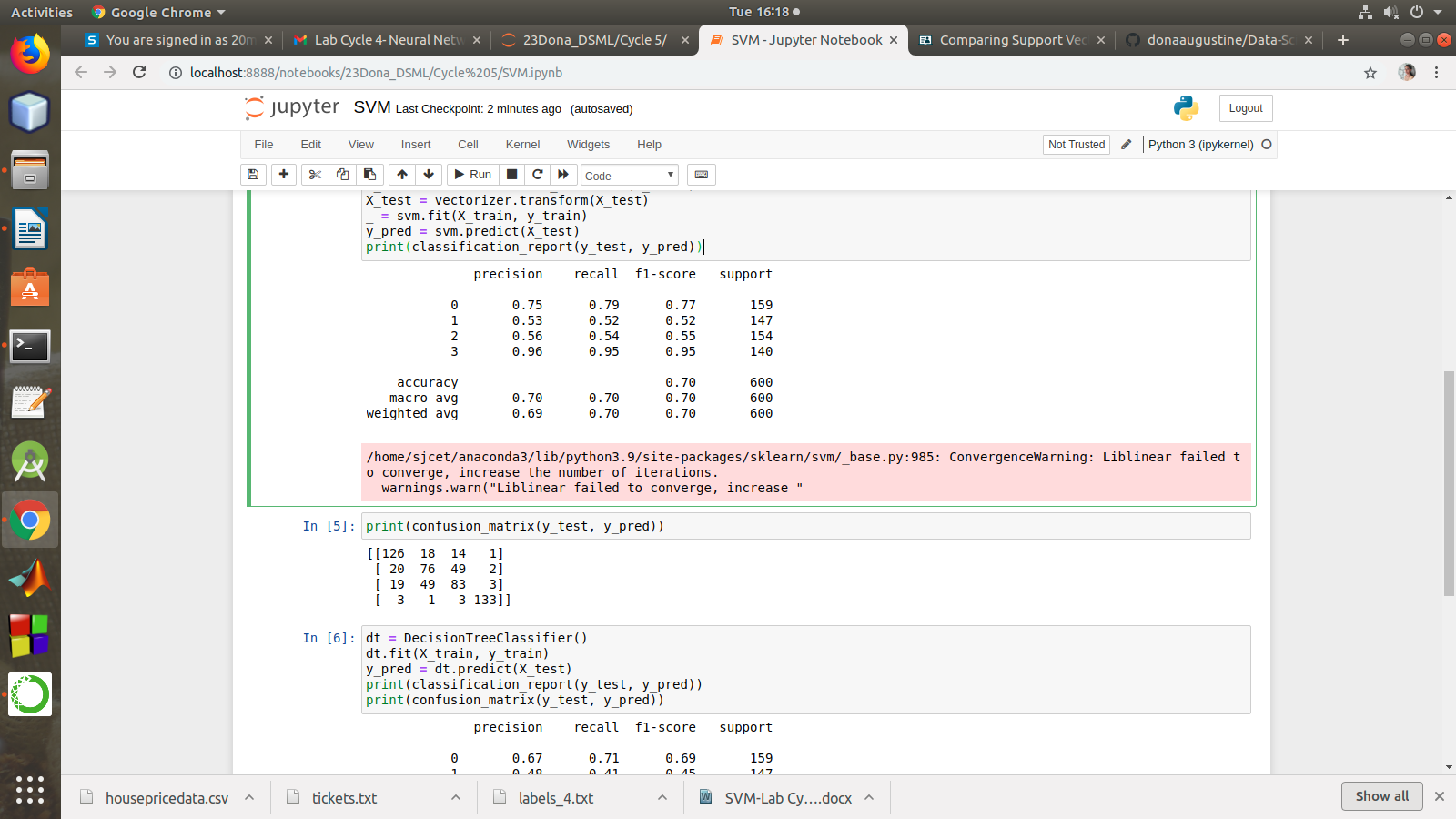
X\_train = vectorizer.fit\_transform(X\_train)

X\_test = vectorizer.transform(X\_test)

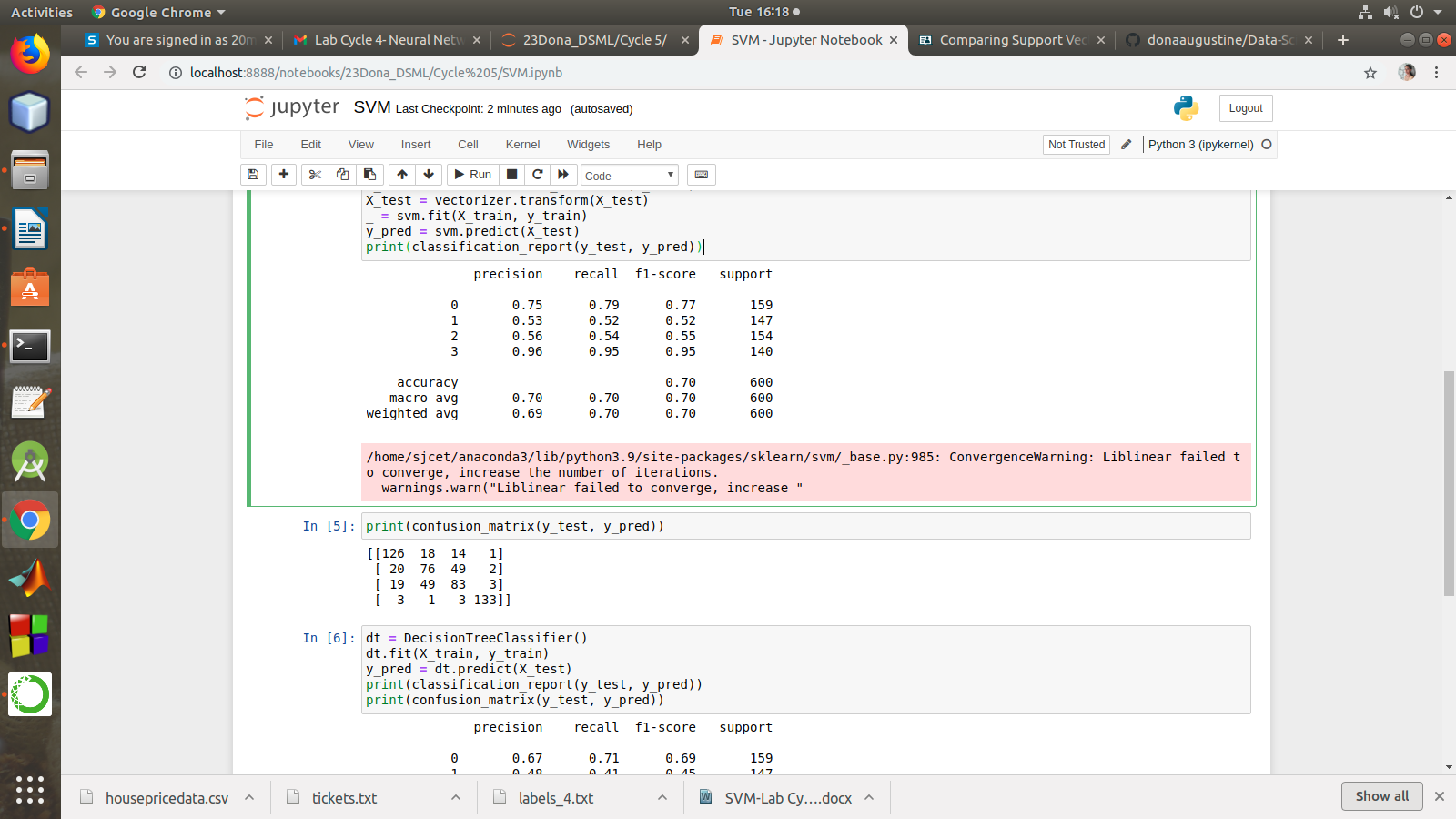
\_ = svm.fit(X\_train, y\_train)

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

print(classification\_report(y\_test, y\_pred))



print(confusion\_matrix(y\_test, y\_pred))



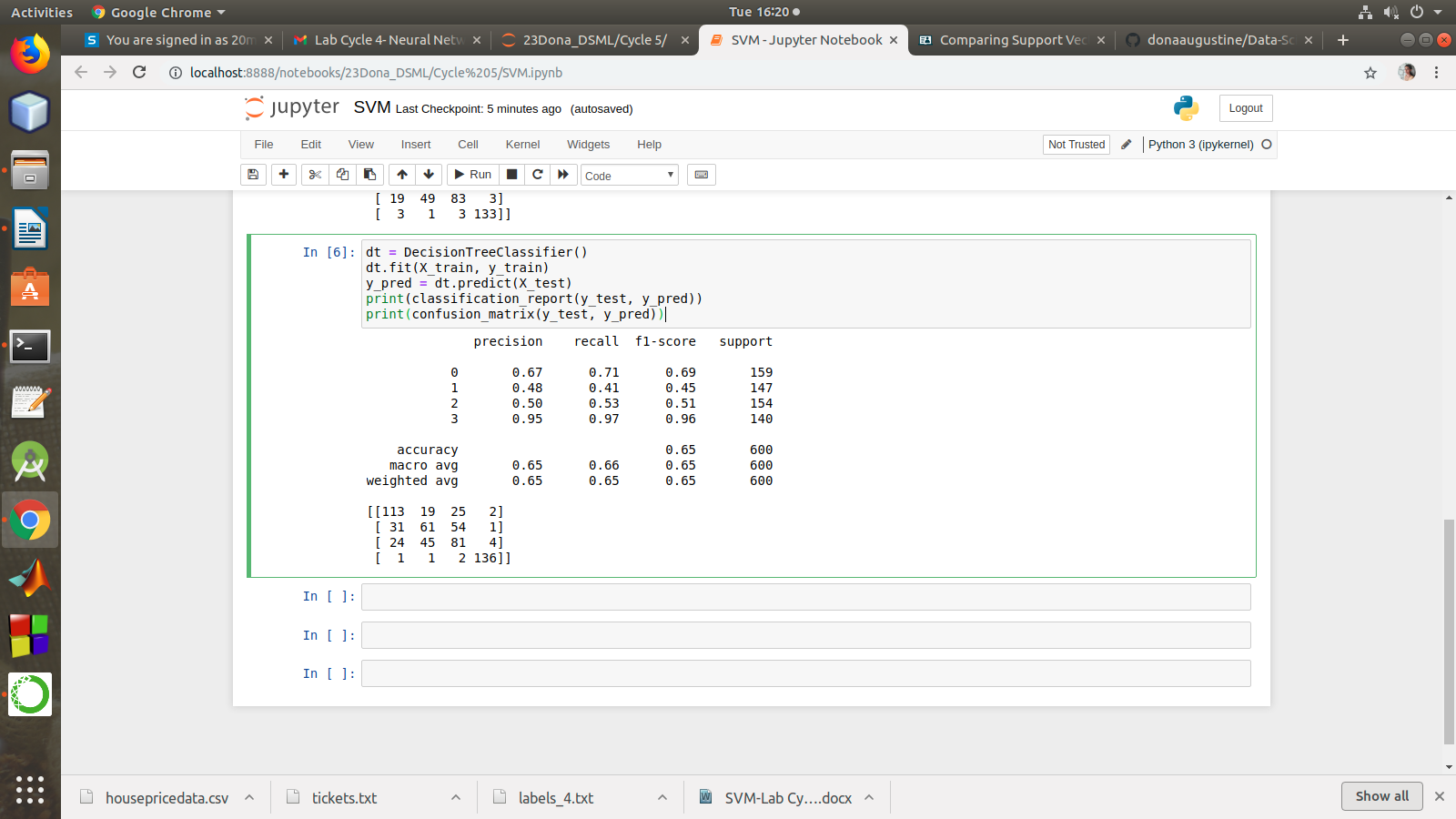
dt = DecisionTreeClassifier()

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

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

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))



1. **K-means**

**Given dataset contains 200 records and five columns, two of which describe the customer’s annual income and spending score. The latter is a value from 0 to 100. The higher the number, the more this customer has spent with the company in the past:**

**Functions to familiarize**:

* The purpose of Kmeans.fit() is to train the model with data.
* The purpose of Kmeans.predict() is to apply a trained model to data

**Using k means clustering create 6 clusters of customers based on their spending pattern.**

* Visualize the same in a scatter plot with each cluster in a different color scheme.
* Display the cluster labels of each point.(print cluster indexes)
* Display the cluster centers.
* Use different values of K and visualize the same using scatter plot

**PROGRAM:**

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

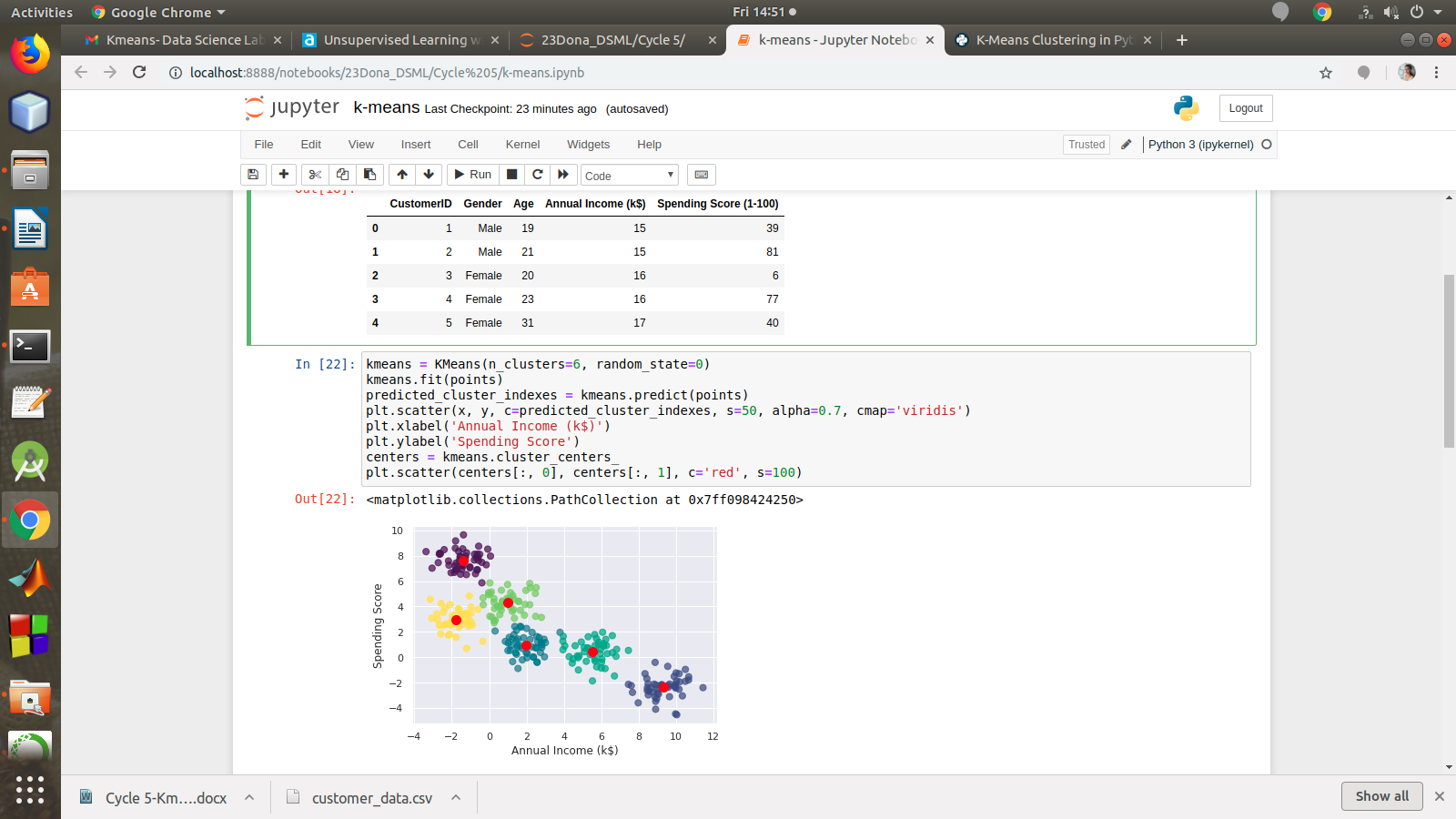
sns.set()

%matplotlib inline

import pandas as pd

customers = pd.read\_csv('customer\_data.csv')

customers.head()



points = customers.iloc[:, 3:5].values

x = points[:, 0]

y = points[:, 1]

plt.scatter(x, y, s=50, alpha=0.7)

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

plt.ylabel('Spending Score')



kmeans = KMeans(n\_clusters=6, random\_state=0)

kmeans.fit(points)

predicted\_cluster\_indexes = kmeans.predict(points)

print(predicted\_cluster\_indexes)

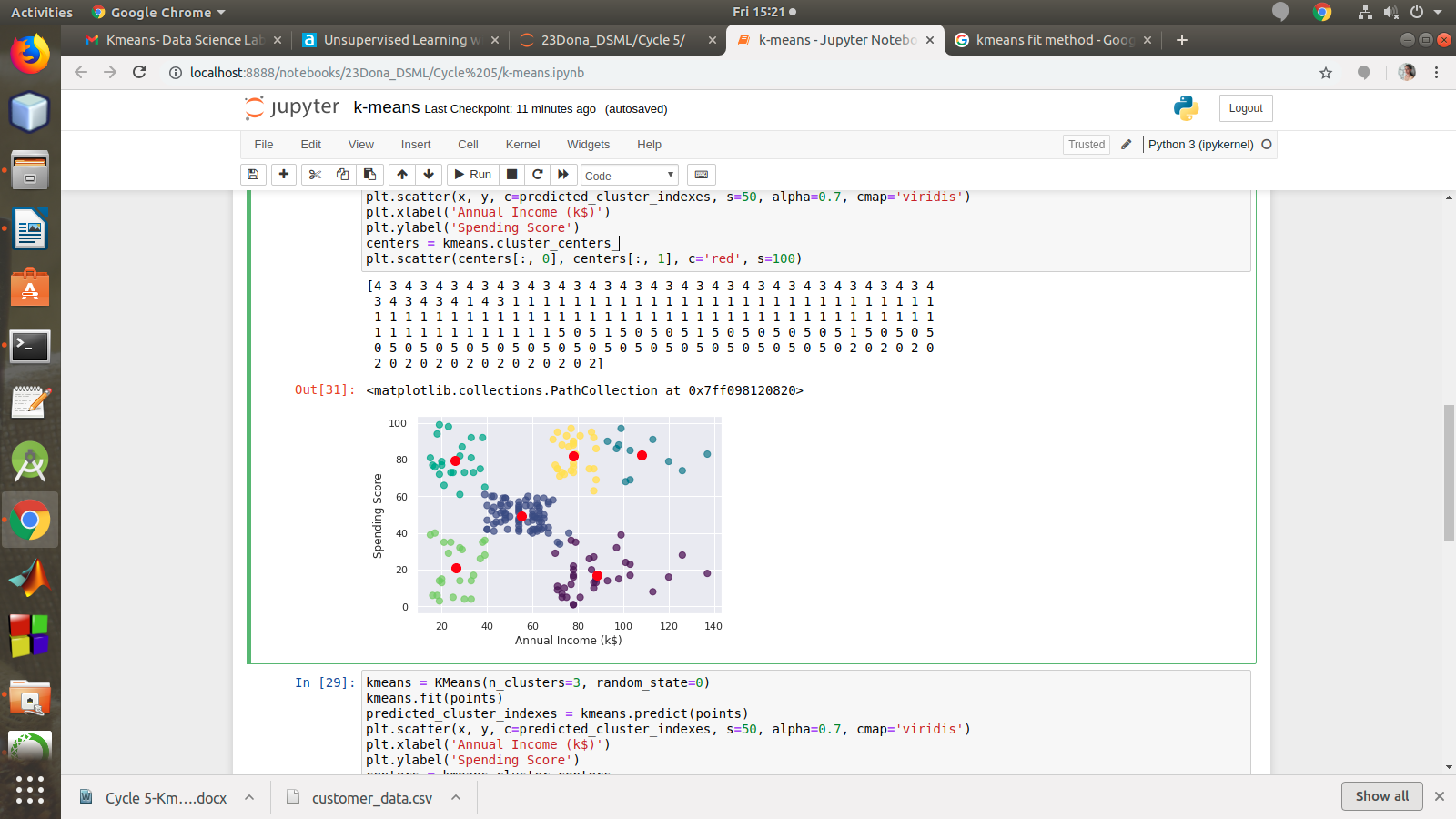
plt.scatter(x, y, c=predicted\_cluster\_indexes, s=50, alpha=0.7, cmap='viridis')

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

plt.ylabel('Spending Score')

centers = kmeans.cluster\_centers\_

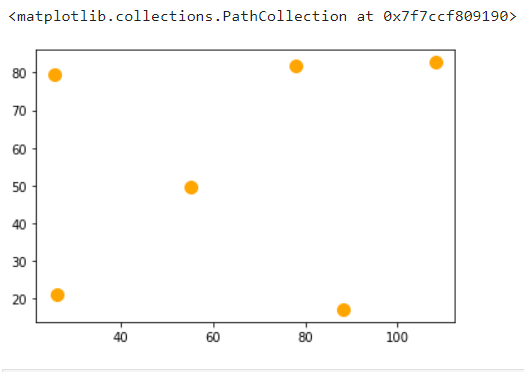
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)



#Display the cluster centers.

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c=&#39;orange&#39;, s=100)



**#Use different values of K and visualize the same using scatter plot.**

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans.fit(points)

predicted\_cluster\_indexes = kmeans.predict(points)

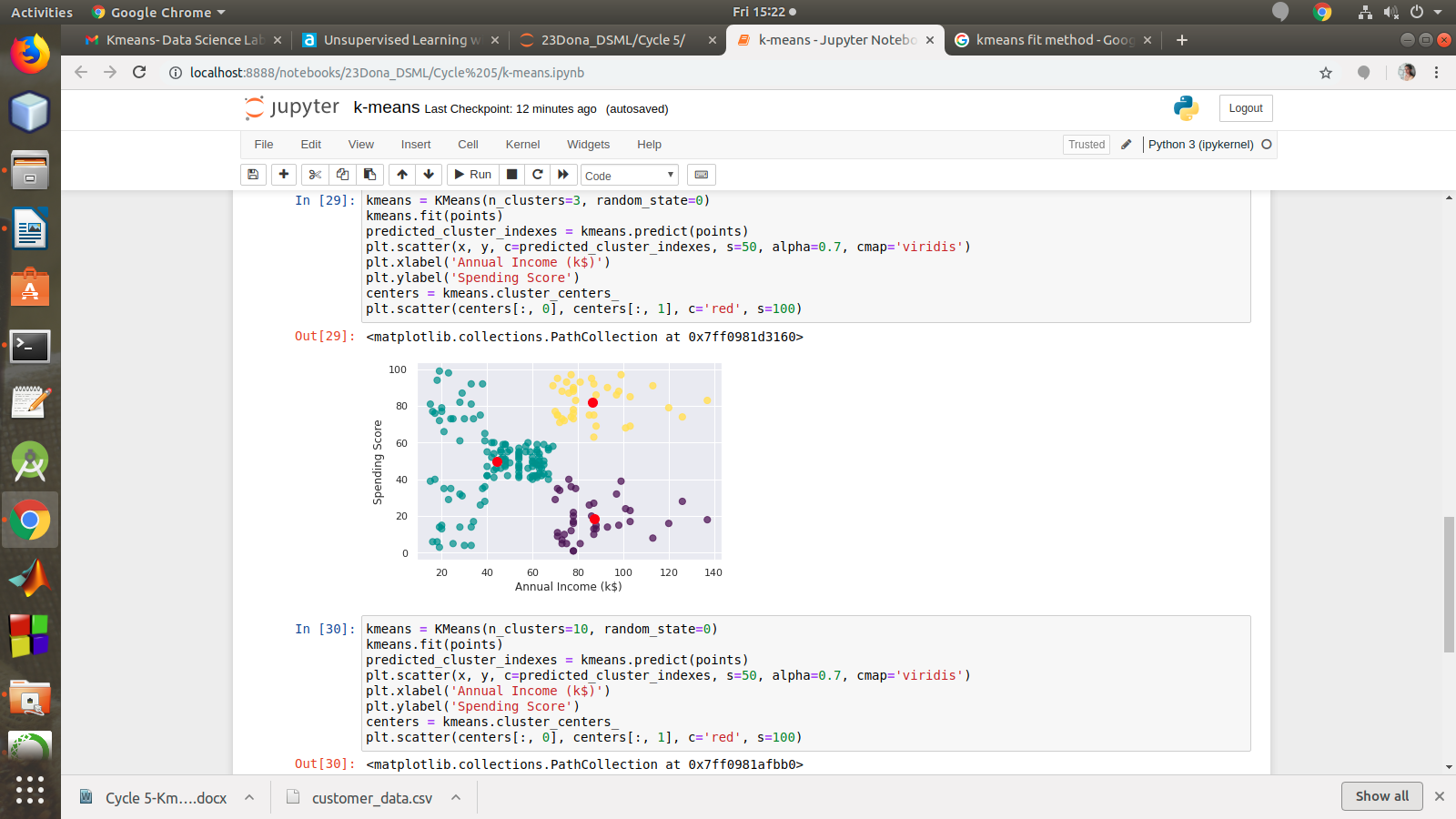
plt.scatter(x, y, c=predicted\_cluster\_indexes, s=50, alpha=0.7, cmap='viridis')

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

plt.ylabel('Spending Score')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)



**from** sklearn.cluster **import** KMeans

kmeans **=** KMeans(n\_clusters**=**8, random\_state**=**0)

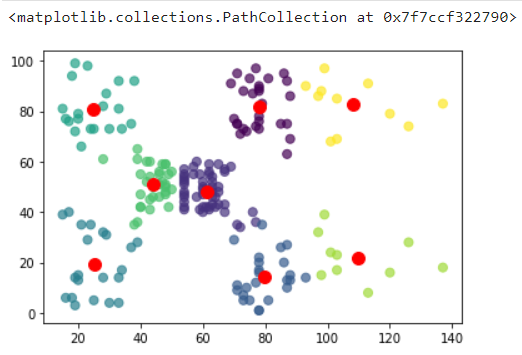
kmeans**.**fit(points)

predicted\_cluster\_indexes **=** kmeans**.**predict(points)

plt**.**scatter(x, y, c**=**predicted\_cluster\_indexes, s**=**50, alpha**=**0.7, cmap**=**'viridis')

centers **=** kmeans**.**cluster\_centers\_

plt**.**scatter(centers[:, 0], centers[:, 1], c**=**'red', s**=**100)



1. **NLP**

**For a given text,perform word and sentence tokenization.Remove the stop words from given text and create n-grams for different values of n.**

**PROGRAM:**

import nltk

nltk.download('punkt')

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import sent\_tokenize,word\_tokenize

text1 = "The data set given satisfies the requirement for model generation. This is used in Data Science Lab."

print(sent\_tokenize(text1))

**OUTPUT:**

['The data set given satisfies the requirement for model generation.', 'This is used in Data Science Lab.']

**PROGRAM:**

print(word\_tokenize(text1))

**OUTPUT:**

['The', 'data', 'set', 'given', 'satisfies', 'the', 'requirement', 'for', 'model', 'generation', '.', 'This', 'is', 'used', 'in', 'Data', 'Science', 'Lab', '.']

**PROGRAM:**

import nltk

nltk.download('stopwords')

print(stopwords.words('english'))

**OUTPUT:**

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself',

'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't",

'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

**PROGRAM:**

text = word\_tokenize(text1)

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

print(text)

**OUTPUT:**

['The', 'data', 'set', 'given', 'satisfies', 'requirement', 'model', 'generation', '.', 'This', 'used', 'Data', 'Science', 'Lab', '.']

**PROGRAM:**

import nltk

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

print(nltk.pos\_tag(text))

**OUTPUT:**

[('The', 'DT'), ('data', 'NN'), ('set', 'NN'), ('given', 'VBN'), ('satisfies', 'NNS'), ('requirement', 'VBP'), ('model', 'NN'), ('generation', 'NN'), ('.', '.'), ('This', 'DT'), ('used', 'VBN'), ('Data', 'NNP'), ('Science', 'NNP'), ('Lab', 'NNP'), ('.', '.')]

**PROGRAM:**

temp=zip(\*[text[i:] for i in range(0,2)])

ans=[' '.join(ngram) for ngram in temp]

print(ans)

**OUTPUT:**

['The data', 'data set', 'set given', 'given satisfies', 'satisfies requirement', 'requirement model', 'model generation', 'generation .', '. This', 'This used', 'used Data', 'Data Science', 'Science Lab', 'Lab .']

**PROGRAM:**

temp=zip(\*[text[i:] for i in range(0,4)])

ans=[' '.join(ngram) for ngram in temp]

print(ans)

**OUTPUT:**

['The data set given', 'data set given satisfies', 'set given satisfies requirement', 'given satisfies requirement model', 'satisfies requirement model generation', 'requirement model generation .',

'model generation . This', 'generation . This used', '. This used Data', 'This used Data Science', 'used Data Science Lab', 'Data Science Lab .']