**AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY**

**AMITY UNIVERSITY**



**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LAB FILE**

**Submitted by:** **Submitted to:**

Aman Satyam Dr. Aakanshi Gupta

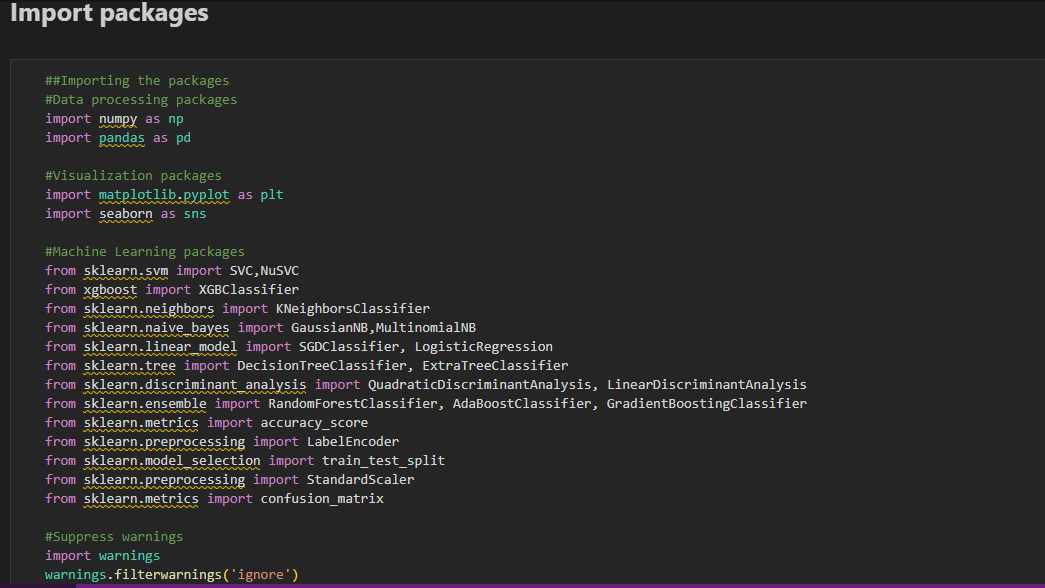
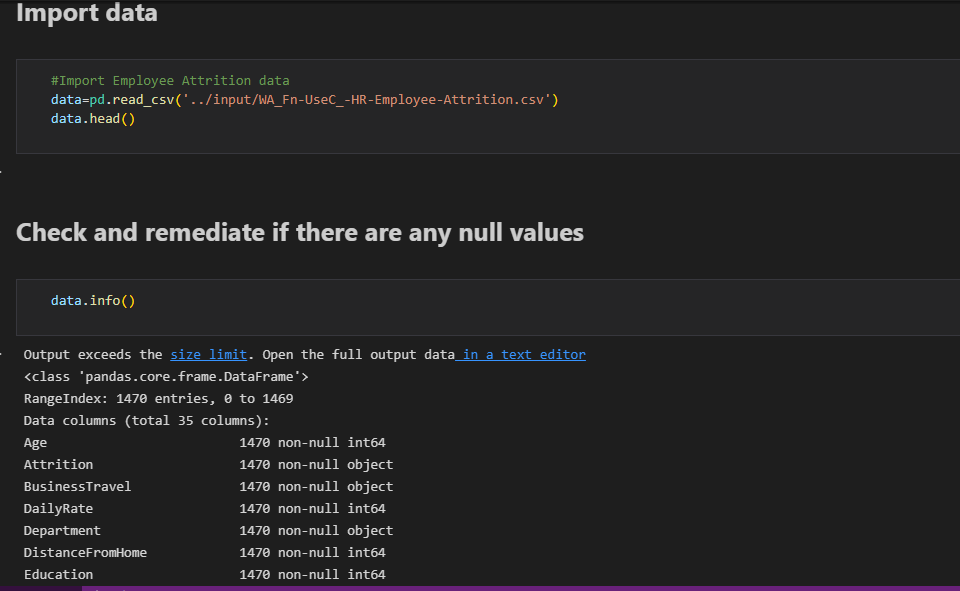
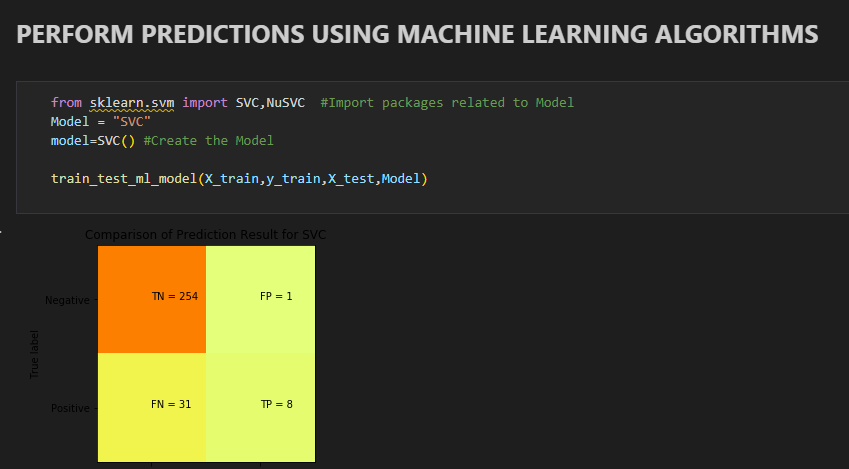
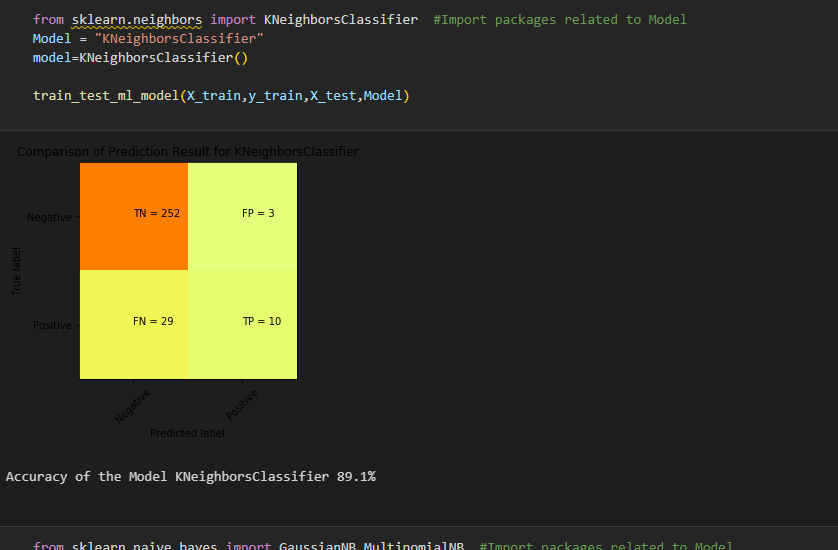
Enrollment Number: A2305219642 Associate Professor

B.tech CSE CSE, DEPT

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **EXPERIMENT** | **DATE** | **PG.NO** | **REMARKS** |
| **1.** | Use Python to predict employee attribution in a firm and help them plan their man power(take dataset from Kaggle) | 21/07/2022 | 3-7 |  |
| **2.** | Write a program in Python for Linear Regression | 04/08/2022 | 8-9 |  |
| **3.** | Create customer clusters using different market strategies on a data set | 11/08/2022 | 10-15 |  |
| **4.** | Write a program to implement K-means Clustering | 25/08/2022 | 16-17 |  |
| **5.** | Make a movie recommendation system | 01/09/2022 | 18-21 |  |
| **6.** | Write a program to implement Logistic regression | 15/09/2022 | 22-23 |  |
| **7.** | Develop a prediction mechanism to predict which employee can go on a leave in a company in near future | 22/09/2022 | 24-28 |  |
| **8.** | Write a program to implement a movie recommendation system | 06/10/2022 | 29-31 |  |
| **9.** | Recognizing alphabet using SVM | 20/10/2022 | 32-38 |  |
| **10.** | Write a program to implement decision tree classification algorithm | 03/11/2022 | 39-42 |  |

# **1.Use python to predict employee attrition in a firm and help them plan their manpower. (take data set from kaggle)**

**** ****       

# **2.WAP in Python for the Linear regression**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

df=pd.read\_csv(r'D:/Saksham 5th sem/Data Science/Latest Covid-19 India Status.csv')

print(df.head())

df1=df[['Total Cases']]

df2=df['Discharged']

print(df1.head())

print(df2.head())

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df1, df2, test\_size = 0.3)

from sklearn.linear\_model import LinearRegression

slr = LinearRegression()

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

print("Intercept: ", slr.intercept\_)

print("Coefficient: ", slr.coef\_)

y\_pred\_slr= slr.predict(x\_test)

print("Prediction for test set: {}".format(y\_pred\_slr))

slr\_diff = pd.DataFrame({'Actual value': y\_test, 'Predicted value': y\_pred\_slr})

print(slr\_diff.head())

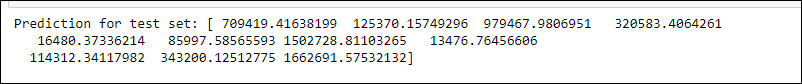
plt.scatter(x\_test,y\_test)

plt.plot(x\_test, y\_pred\_slr, 'Red')

plt.show()

Text

Description automatically generated



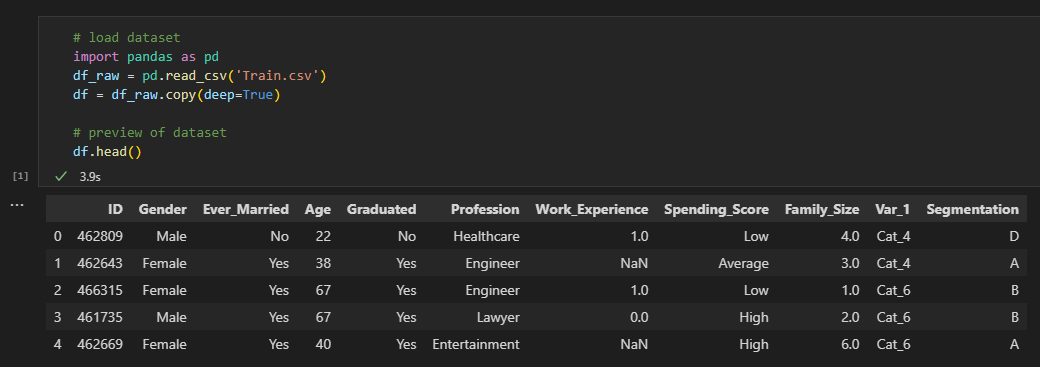
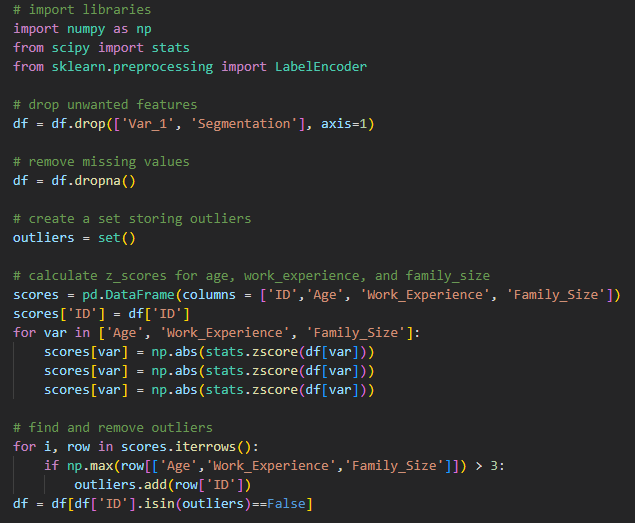
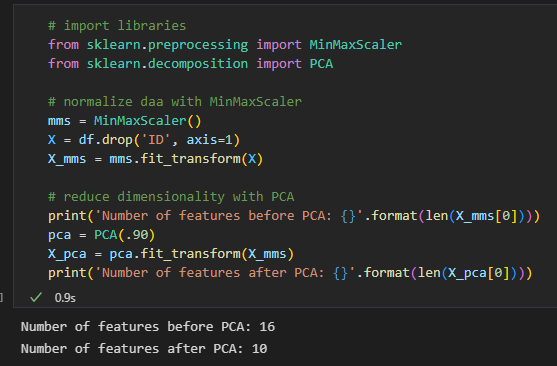
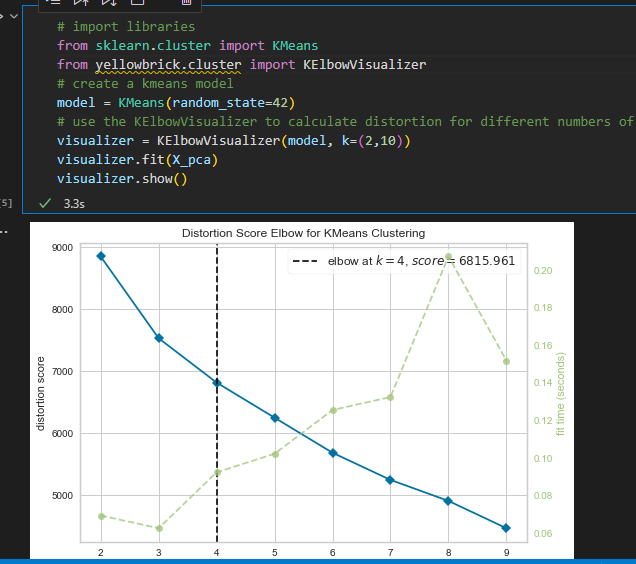
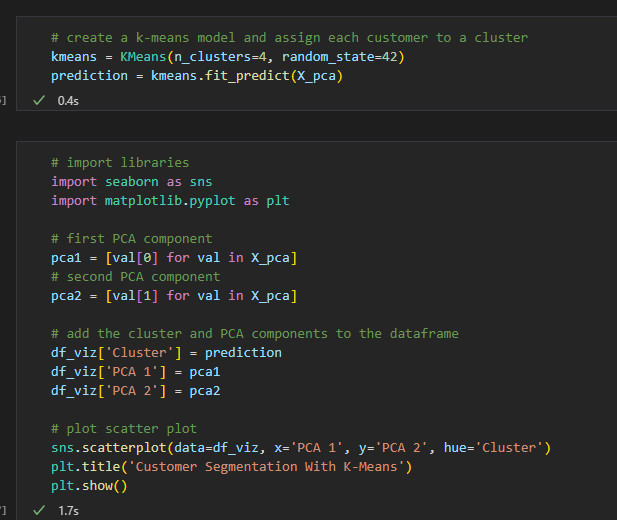
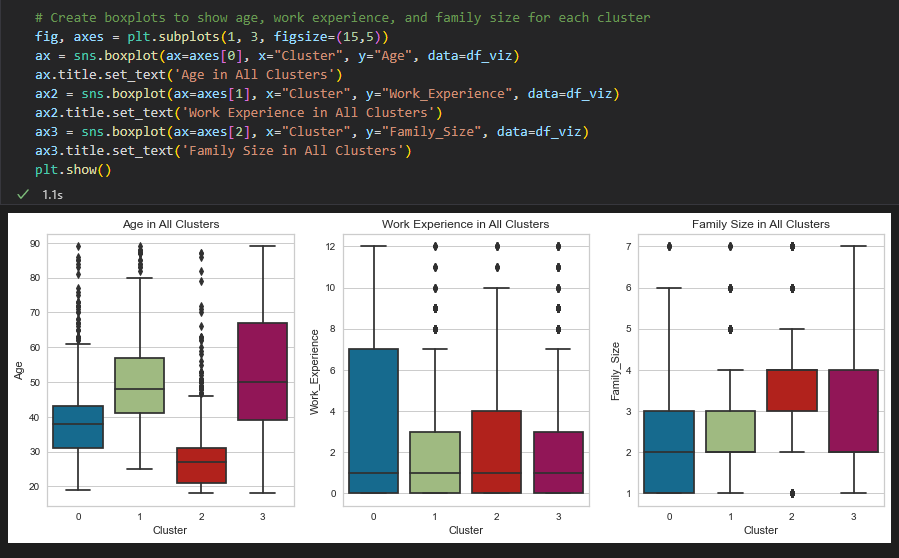
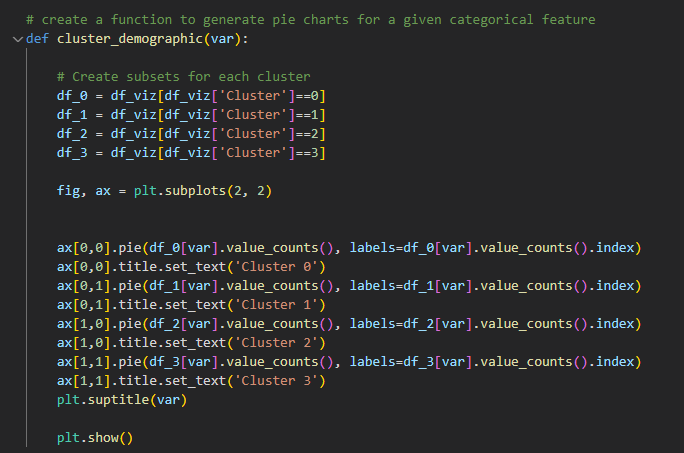
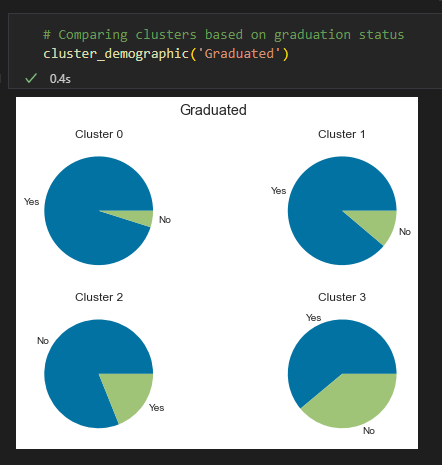
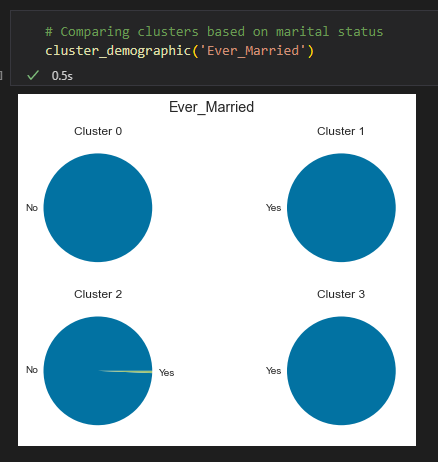
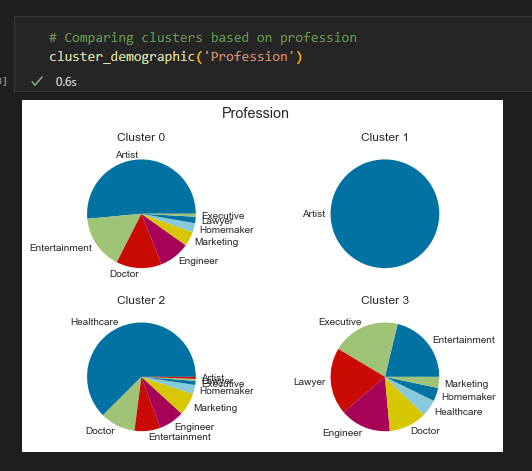
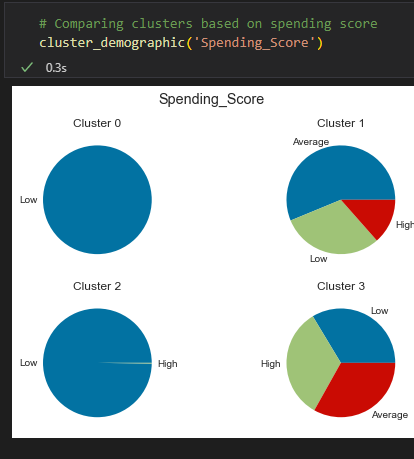
Text

Description automatically generated

Chart, line chart

Description automatically generated

# **3. Create customer clusters using different market strategies on a data set.**

**** ****          

# **4.WAP to implement K-means clustering**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import datasets

iris = datasets.load\_iris()

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

iris\_df.head()

from sklearn.cluster import KMeans

wcss = []

x = data.iloc[:, [0, 1, 2, 3]].values

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++',

max\_iter = 300, n\_init = 10, random\_state = 0)

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n\_clusters = 3, init = 'k-means++',

max\_iter = 300, n\_init = 10, random\_state = 0)

y\_kmeans = kmeans.fit\_predict(x)

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1],

s = 100, c = 'red', label = 'Iris-setosa')

plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1],

s = 100, c = 'blue', label = 'Iris-versicolour')

plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1],

s = 100, c = 'green', label = 'Iris-virginica')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1],

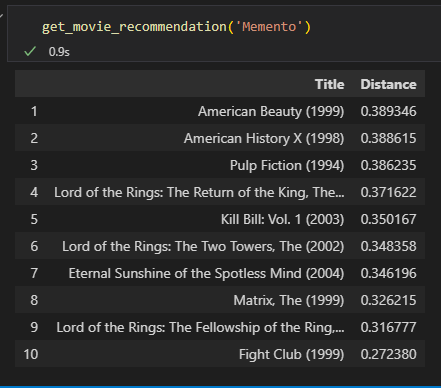
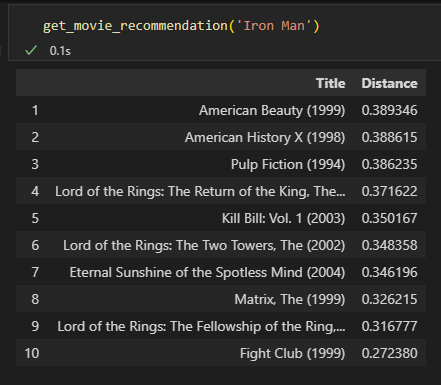
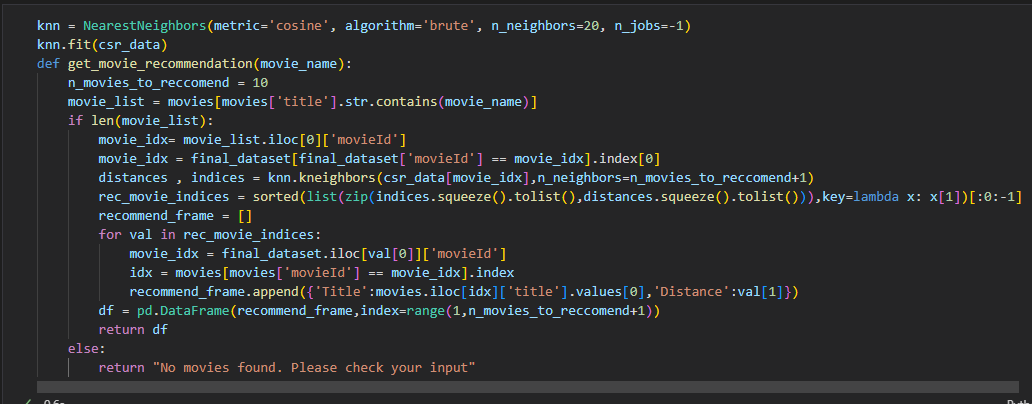
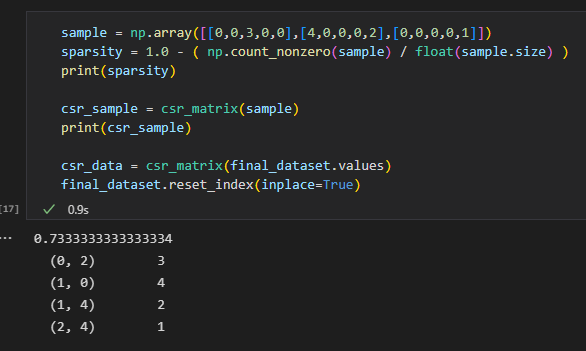
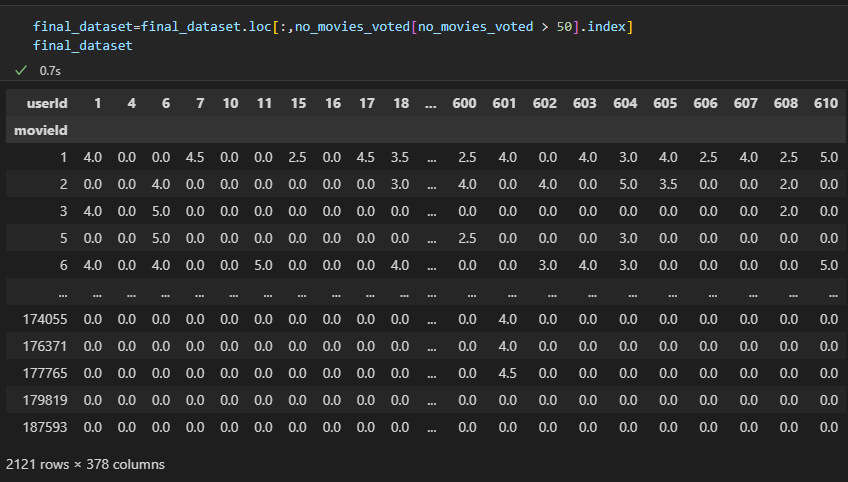
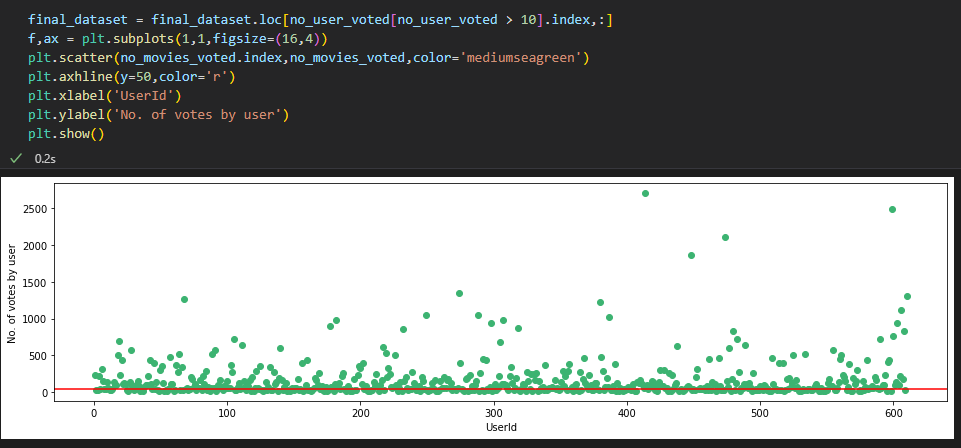
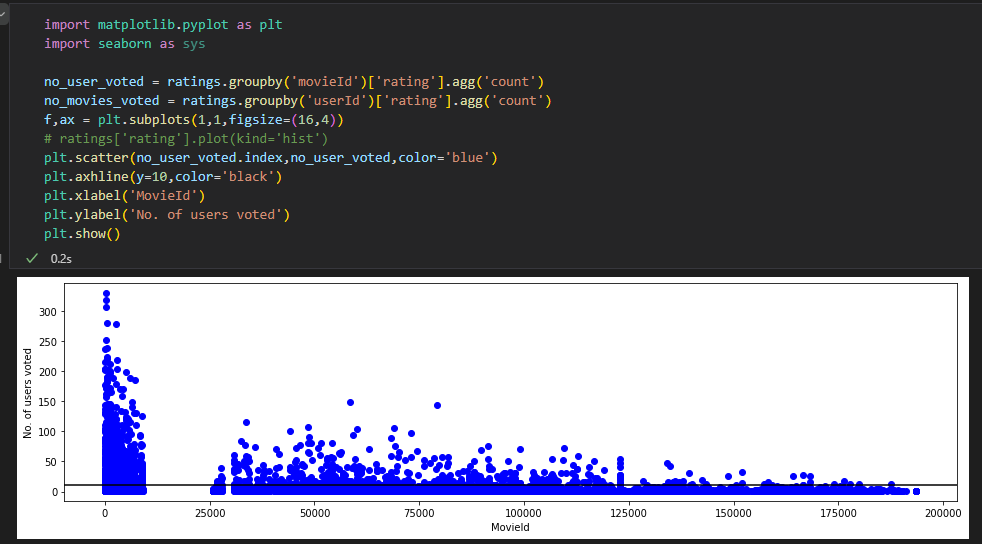
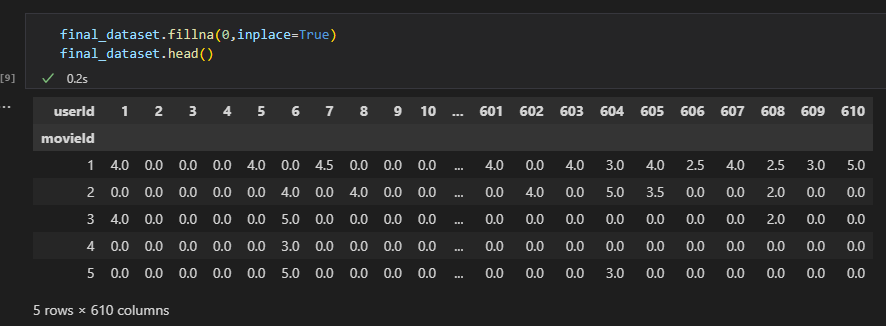
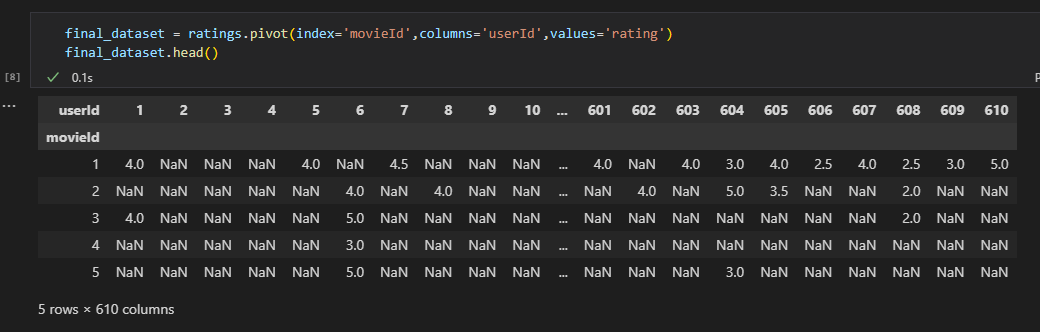
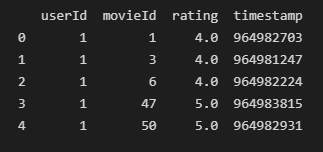
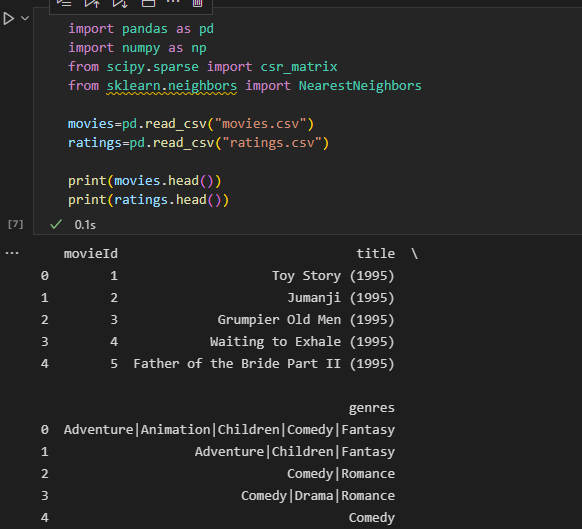
s = 100, c = 'yellow', label = 'Centroids')

plt.legend()

Chart, scatter chart, bubble chart

Description automatically generated

# **5. Make a movie recommendation system.**

****

# **6. WAP to implement logistic regression**

import pandas as pd

col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

pima = pd.read\_csv("pima-indians-diabetes.csv", header=None, names=col\_names)

feature\_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

X = pima[feature\_cols]

y = pima.label

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=16)

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(random\_state=16)

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

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

from sklearn import metrics

cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

cnf\_matrix

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

class\_names=[0,1]

fig, ax = plt.subplots()

tick\_marks = np.arange(len(class\_names))

plt.xticks(tick\_marks, class\_names)

plt.yticks(tick\_marks, class\_names)

sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

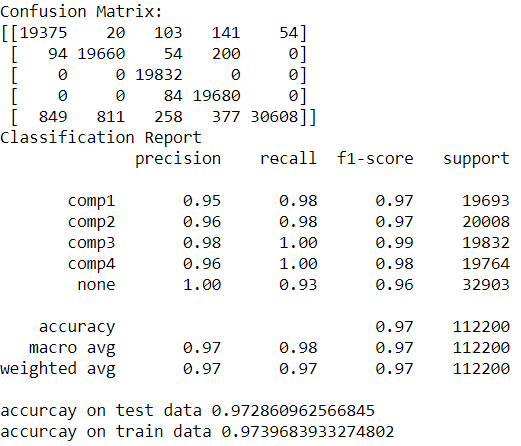
ax.xaxis.set\_label\_position("top")

plt.tight\_layout()

plt.title('Confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')



# **7. Develop a prediction mechanism to predict which employee can go on leave in a company in near future**

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_excel('Job.xlsx')

data=dataset.iloc[:,6].values

df=pd.DataFrame()

i=0

while i<len(data):

if(data[i]==1):

a=[dataset.iloc[i,:].values]

df1=pd.DataFrame(a)

df=df.append(df1)

i=i+1

else:

i=i+1

df=df.reset\_index()

df=df.drop('index',1)

# Preprocessing

X=dataset.iloc[:,[4,7,9]].values

X=pd.DataFrame(X)

y=dataset.iloc[:,[6]].values

from sklearn.preprocessing import OneHotEncoder,LabelEncoder

label=LabelEncoder()

X[2]=label.fit\_transform(X[2])

one=OneHotEncoder(categorical\_features=[2])

X=one.fit\_transform(X).toarray()

from sklearn.cross\_validation import train\_test\_split

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

# Plotting Of Graph

plt.hist(df[0])

plt.hist(df[1])

plt.hist(df[2])

plt.hist(df[3])

plt.hist(df[4])

plt.hist(df[5])

# Reason For Leaving The Company

Reason1=pd.DataFrame()

i=0

WithorNotWorkAccident=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4):

a=[dataset.iloc[i,[7,9]].values]

df1=pd.DataFrame(a)

Reason1=Reason1.append(df1)

i=i+1

WithorNotWorkAccident=WithorNotWorkAccident+1

else:

i=i+1

Reason1=Reason1.reset\_index()

Reason1=Reason1.drop('index',1)

Reason2=pd.DataFrame()

i=0

WithWorkAccident=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],5].values==1):

a=[dataset.iloc[i,[7,9]].values]

df1=pd.DataFrame(a)

Reason2=Reason2.append(df1)

i=i+1

WithWorkAccident=WithWorkAccident+1

else:

i=i+1

Reason2=Reason2.reset\_index()

Reason2=Reason2.drop('index',1)

Reason2WithoutAcc=pd.DataFrame()

i=0

WithoutWorkAccident=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],5].values==0):

a=[dataset.iloc[i,[7,9]].values]

df1=pd.DataFrame(a)

Reason2WithoutAcc=Reason2WithoutAcc.append(df1)

i=i+1

WithoutWorkAccident=WithoutWorkAccident+1

else:

i=i+1

Reason2WithoutAcc=Reason2WithoutAcc.reset\_index()

Reason2WithoutAcc=Reason2WithoutAcc.drop('index',1)

Reason3=pd.DataFrame()

i=0

WithHigherProjects=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],2].values>3):

a=[dataset.iloc[i,[7,8,9]].values]

df1=pd.DataFrame(a)

Reason3=Reason3.append(df1)

i=i+1

WithHigherProjects=WithHigherProjects+1

else:

i=i+1

Reason3=Reason3.reset\_index()

Reason3=Reason3.drop('index',1)

Reason4=pd.DataFrame()

i=0

WithPromotion=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],2].values>3 and dataset.iloc[[i],7].values==1):

a=[dataset.iloc[i,[9]].values]

df1=pd.DataFrame(a)

Reason4=Reason4.append(df1)

i=i+1

WithPromotion=WithPromotion+1

else:

i=i+1

Reason4=Reason4.reset\_index()

Reason4=Reason4.drop('index',1)

Reason5=pd.DataFrame()

i=0

WithLowerSalary=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],2].values>3 and dataset.iloc[[i],9].values=='low'):

a=[dataset.iloc[i,[7]].values]

df1=pd.DataFrame(a)

Reason5=Reason5.append(df1)

i=i+1

WithLowerSalary=WithLowerSalary+1

else:

i=i+1

Reason5=Reason5.reset\_index()

Reason5=Reason5.drop('index',1)

Reason6=pd.DataFrame()

i=0

WithHigherTime=0

while i<len(data):

if(dataset.iloc[[i],0].values> 0.5 and dataset.iloc[[i],1].values > 0.5 and dataset.iloc[[i],4].values> 4 and dataset.iloc[[i],2].values>3 and dataset.iloc[[i],3].values>200):

a=[dataset.iloc[i,[9]].values]

df1=pd.DataFrame(a)

Reason6=Reason6.append(df1)

i=i+1

WithHigherTime=WithHigherTime+1

else:

i=i+1

Reason6=Reason6.reset\_index()

Reason6=Reason6.drop('index',1)

b=[WithHigherProjects,WithLowerSalary,WithHigherTime,WithPromotion,WithWorkAccident,WithorNotWorkAccident,WithoutWorkAccident]

Visual=pd.DataFrame(b)

ax = Visual[0].plot(kind='bar', title ="V comp", figsize=(15, 10), legend=True, fontsize=12)

ax.set\_xlabel("Factors Affecting Employess", fontsize=12)

ax.set\_ylabel("Values", fontsize=12)

plt.show()

# Regression Models

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

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

from sklearn.tree import DecisionTreeClassifier

Classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)

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

y\_pred\_decision = Classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

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

from sklearn.ensemble import RandomForestClassifier

Classifier1 = RandomForestClassifier(n\_estimators = 5, criterion = 'entropy', random\_state = 0)

Classifier1.fit(X\_train, y\_train)

y\_pred\_RandomForest = Classifier1.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm1 = confusion\_matrix(y\_test, y\_pred\_RandomForest)

from sklearn.linear\_model import LogisticRegression

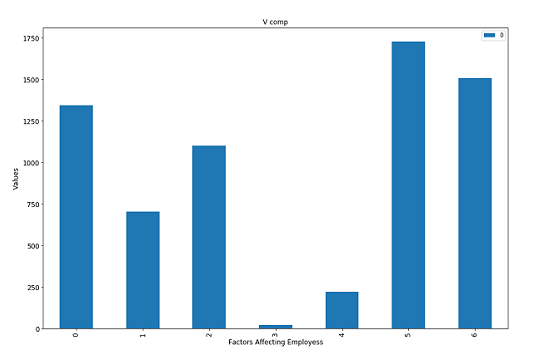
classifier2=LogisticRegression(random\_state=0)

classifier2.fit(X\_train,y\_train)

y\_pred\_LR=classifier2.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm2 = confusion\_matrix(y\_test, y\_pred\_LR)



# **8.WAP to implement a movie recommendation system**

import pandas as pd

column\_names = ['user\_id', 'item\_id', 'rating', 'timestamp']

path = ‘C://wp-content/uploads/file.tsv'

df = pd.read\_csv(path, sep='\t', names=column\_names)

df.head()

movie\_titles = pd.read\_csv('https://media.geeksforgeeks.org/wp-content/uploads/Movie\_Id\_Titles.csv')

movie\_titles.head()

ratings = pd.DataFrame(data.groupby('title')['rating'].mean())

ratings['num of ratings'] = pd.DataFrame(data.groupby('title')['rating'].count())

ratings.head()

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('white')

%matplotlib inline

plt.figure(figsize =(10, 4))

ratings['num of ratings'].hist(bins = 70)

# plot graph of 'ratings' column

plt.figure(figsize =(10, 4))

ratings['rating'].hist(bins = 70)

moviemat = data.pivot\_table(index ='user\_id',

columns ='title', values ='rating')

moviemat.head()

ratings.sort\_values('num of ratings', ascending = False).head(10)

similar\_to\_starwars = moviemat.corrwith(starwars\_user\_ratings)

similar\_to\_liarliar = moviemat.corrwith(liarliar\_user\_ratings)

corr\_starwars = pd.DataFrame(similar\_to\_starwars, columns =['Correlation'])

corr\_starwars.dropna(inplace = True)

corr\_starwars.head()

corr\_starwars.sort\_values('Correlation', ascending = False).head(10)

corr\_starwars = corr\_starwars.join(ratings['num of ratings'])

corr\_starwars.head()

corr\_starwars[corr\_starwars['num of ratings']>100].sort\_values('Correlation', ascending = False).head()

Graphical user interface

Description automatically generated with low confidence

Graphical user interface

Description automatically generated

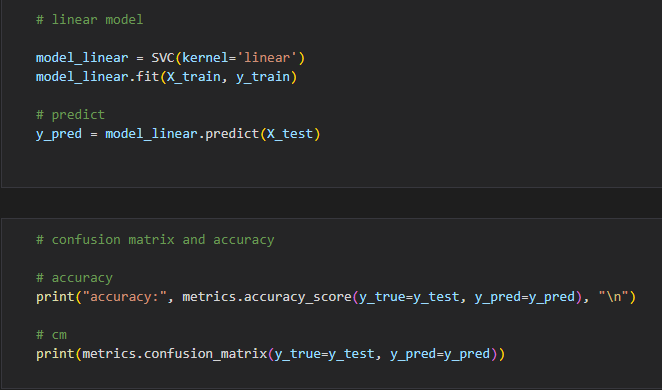
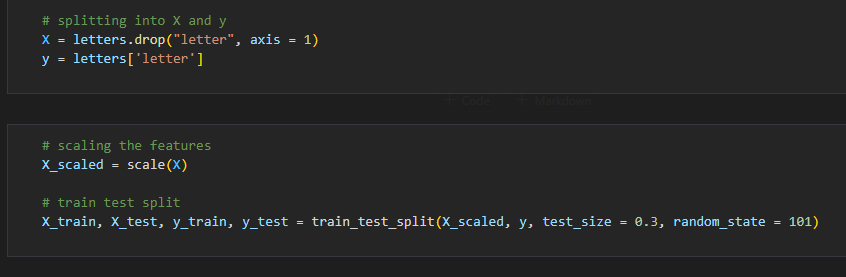
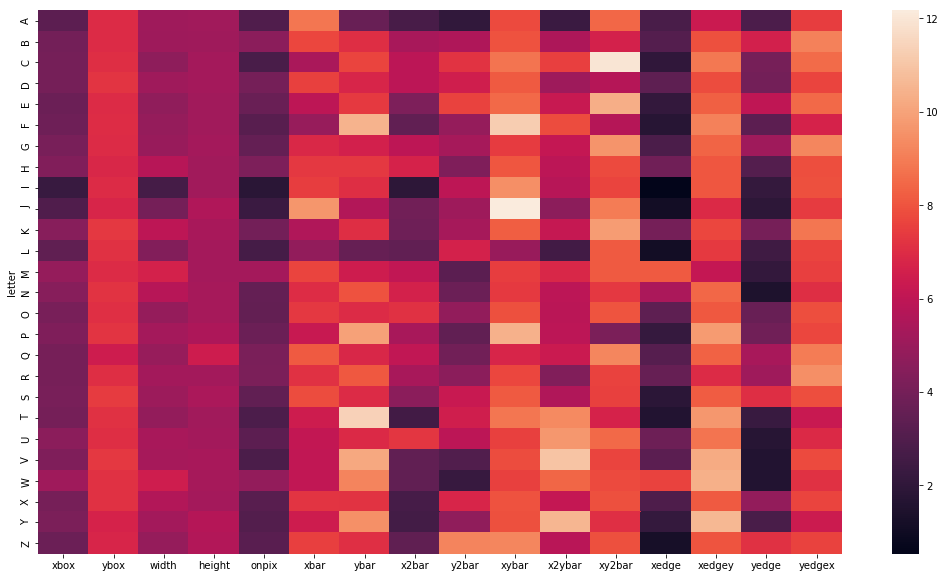
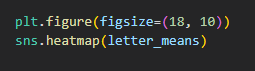
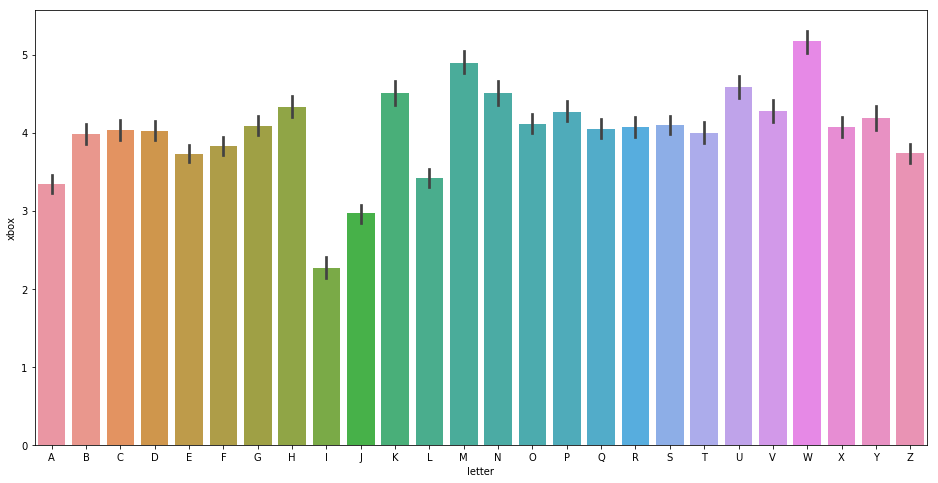
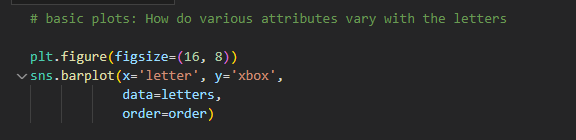
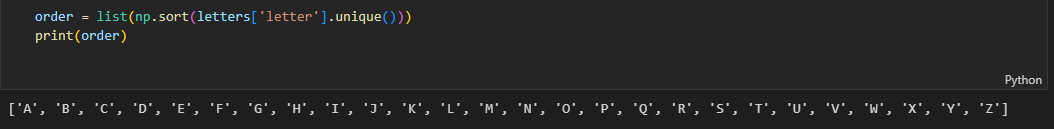
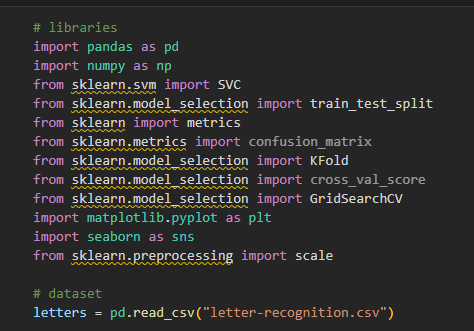
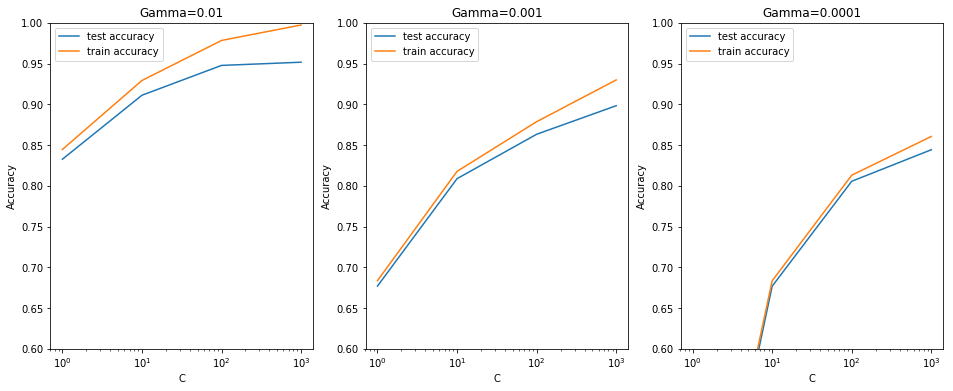
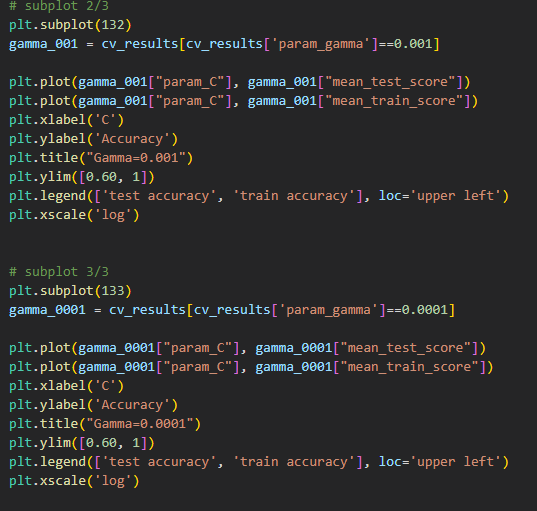
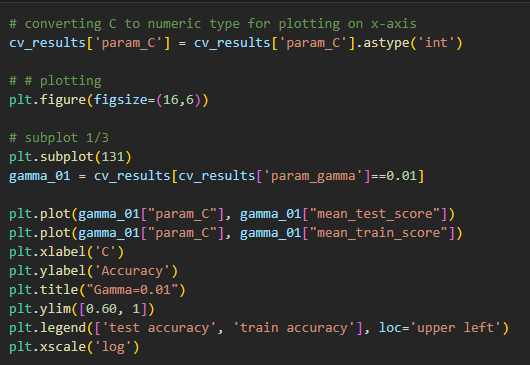
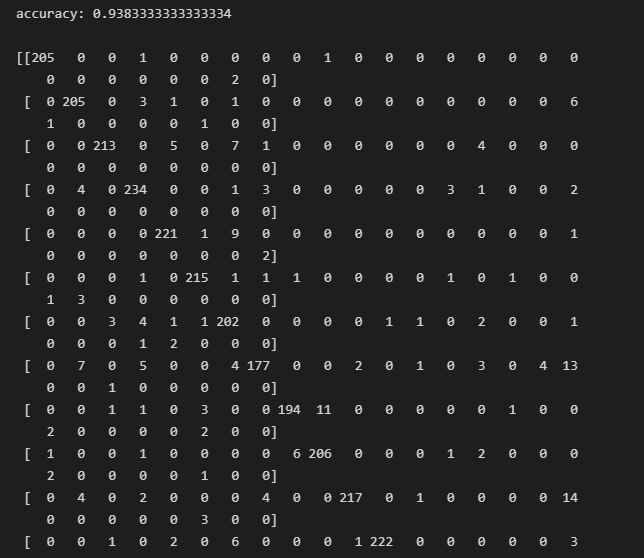
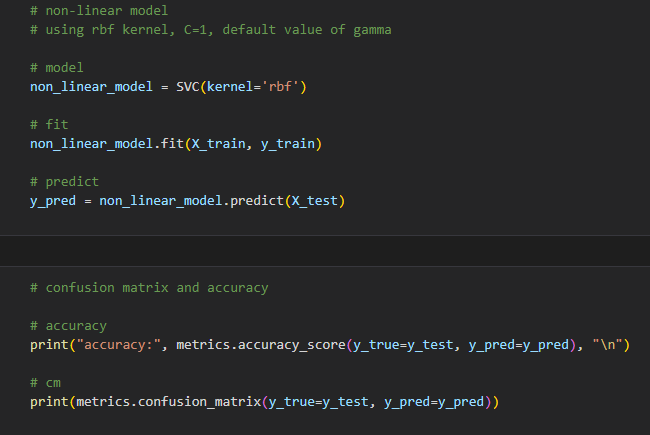
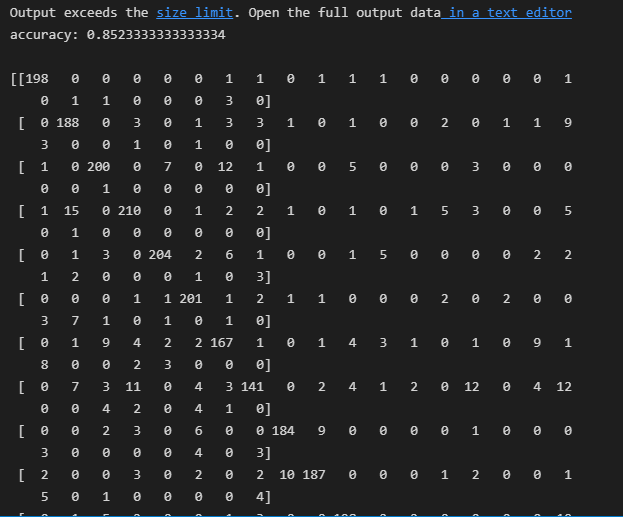
Chart, histogram

Description automatically generated

Table

Description automatically generated

# **9. Recognizing alphabets using SVM.**

** **

# **10.WAP to implement decision tree classification algorithm**

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

def importdata():

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-'+

'databases/balance-scale/balance-scale.data',

sep= ',', header = None)

print ("Dataset Length: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

print ("Dataset: ",balance\_data.head())

return balance\_data

def splitdataset(balance\_data):

# Separating the target variable

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:, 0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

def train\_using\_gini(X\_train, X\_test, y\_train):

clf\_gini = DecisionTreeClassifier(criterion = "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

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

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

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

def main():

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

print("Results Using Gini Index:")

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

if \_\_name\_\_=="\_\_main\_\_":

main()

Calendar

Description automatically generated

Table

Description automatically generated

Calendar

Description automatically generated