**Source Code for Project Exam-1**

**Task 1**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.medol\_selection import trian\_tset\_split

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

from sklearn.neighbors import KNeighborsClassifier

# Read file

dat = pd.read\_csv("creditcard.csv")

# trianing and tseting of the datset

x\_dat = dat.drop("Class", axis=1)

y\_dat = dat.Class

X\_trian, X\_tset, Y\_trian, Y\_tset = trian\_tset\_split(x\_dat, y\_dat, tset\_size=0.3, random\_state=0)

# Visual representation of the sample

count = dat["Class"].value\_counts()

sample = ["Real", "Fraud"]

plt.title("Transactions")

plt.pie(count, labels=sample, autopct='%0.4f%%')

plt.show()

# Classification using Naive Bayes

medol = GaussianNB()

medol.fit(x\_dat, y\_dat)

predict = medol.predict(X\_tset)

accuracy = accuracy\_score(Y\_tset, predict) \* 100

print("Naive Bayes classification:")

print("tset dat accuracy %: " + str(accuracy))

print("Confusion matrix: ")

print(confusion\_matrix(Y\_tset, predict))

# Undersampling as a solution to the unbalanced datset

medol = KNeighborsClassifier()

medol.fit(x\_dat, y\_dat)

predict = medol.predict(X\_tset)

accuracy = accuracy\_score(Y\_tset, predict) \* 100

print("kNN classification:")

print("tset dat accuracy %: " + str(accuracy))

print("Confusion matrix: ")

print(confusion\_matrix(Y\_tset, predict))

**Task 2**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn

import metrics

import numpy as np

import seaborn as sns

#feeding the data Customers.csv to data variable

dat = pd.read\_csv('Customers.csv')

# Null values condition check

nulls = pd.DatFrame(dat.isnull().sum().sort\_values(ascending=False)[:5])

nulls.columns = ['Null Count']

nulls.index.name = 'Feature'

print(nulls)

print(50\*"==")

# Handling the null values if it has any

dat = dat.select\_dtypes(include=

[np.number]).interpolate().dropna()

# Using elbow method to find the good no. of clusters

wcss= []

#Taking only the last two columns that is spending and income

x = dat.iloc[:,2:]

print(x)

#Visualising the dat

sns.FacetGrid(x, height=4).map(plt.scatter, 'Annual Income (k$)', 'Spending Score (1-100)').add\_legend()

plt.title('before clustering the dat')

plt.show()

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.xlabel('Number of Clusters')

plt.ylabel('wcss')

plt.title('Elbow Graph')

plt.show()

# part - 2:

#From above plot we found that for no of clusters = 5 the graph is steadily decreasing

km =KMeans(n\_clusters=5, random\_state=0)

km.fit(x)

kmeans=km.predict(x)

# predict the cluster for the data point

y\_cluster\_kmeans = km.predict(x)

score = metrics.silhouette\_score(x, y\_cluster\_kmeans)

print("Silhouette score is :",score)

x['res'] = y\_cluster\_kmeans

print(y\_cluster\_kmeans)

sns.FacetGrid(x,hue="res", height=4).map(plt.scatter, 'Annual Income (k$)', 'Spending Score (1-100)').add\_legend()

plt.title('After clustering')

plt.show()

#Part -3

#From the first plot we can see that there the density of the values are

#formed at five different points so we can infer directly from the graph

#that we can use 5 clusters that is what we got from the elbow graph

**Task 3**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import linear\_medol

from sklearn.metrics import mean\_squared\_error

from sklearn.medol\_selection import trian\_tset\_split

# reading the dat

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

# dropping precipitation type, formatted date, summary and daily summary columns

df = df.drop(['Precip Type', 'Formatted Date', 'Summary', 'Daily Summary'], axis=1)

# including only numerical features

numeric\_features = df.select\_dtypes(include=[np.number])

print(numeric\_features)

# finding the null values in the dat set

null\_values =

pd.DatFrame(df.isnull().sum().sort\_values(ascending=False))

null\_values.columns = ['Null Count']

null\_values.index.name = 'Feature'

print(null\_values)

# finding top 3 correlated features wrt temperature

correlation = numeric\_features.corr()

print('The top 3 correlated features \n')

print(correlation['Temperature (C)'].sort\_values(ascending=False)[:4], '\n')

# visualizing the dat set columns

df[df.dtypes[(df.dtypes == "float64") | (df.dtypes == "int64")].index.values].hist(figsize=[11, 11])

plt.show()

# Plotting Temperature vs Apparent Temperature, Temperature vs Wind Speed, Temperature vs Wind Bearing

y\_axis\_dat1 = df["Apparent Temperature (C)"]

y\_axis\_dat2 = df["Wind Speed (km/h)"]

y\_axis\_dat3 = df["Wind Bearing (degrees)"]

x\_dat = df["Temperature (C)"]

fig = plt.figure()

plt.rcParams['figure.figsize'] = (11, 11)

axis1 = fig.add\_subplot(1, 3, 1)

axis2 = fig.add\_subplot(1, 3, 2)

axis3 = fig.add\_subplot(1, 3, 3)

axis1.plot(x\_dat, y\_axis\_dat1, label='dat 1')

axis2.plot(x\_dat, y\_axis\_dat2, label='dat 2')

axis3.plot(x\_dat, y\_axis\_dat3, label='dat 3')

axis1.set\_xlabel('Temperature (C)')

axis1.set\_ylabel('Apparent Temperature (C)')

axis2.set\_xlabel('Temperature (C)')

axis2.set\_ylabel('Wind Speed (km/h)')

axis3.set\_xlabel('Temperature (C)')

axis3.set\_ylabel('Wind Bearing (degrees)')

plt.show()

# trianing medol with all features

x\_trian\_dat = df.drop("Temperature (C)", axis=1)

y\_trian\_dat = df["Temperature (C)"]

# splitting the dat into trian and tset dat

x\_trian\_dat, x\_tset\_dat, y\_trian\_dat, y\_tset\_dat = trian\_tset\_split(x\_trian\_dat, y\_trian\_dat, tset\_size=0.3, random\_state=0)

# trianing the medol

lr = linear\_medol.LinearRegression()

medol = lr.fit(x\_trian\_dat, y\_trian\_dat)

print("R Squared value is: ", medol.score(x\_tset\_dat, y\_tset\_dat))

y\_predictions = medol.predict(x\_tset\_dat)

print('Root Mean Square Error is: ', mean\_squared\_error(y\_tset\_dat, y\_predictions))

# trianing medol with only top 3 correlated features

x\_trian\_dat = df[["Apparent Temperature (C)", "Visibility (km)", "Wind Bearing (degrees)"]]

y\_trian\_dat = df["Temperature (C)"]

# splitting the dat into trian and tset dat

x\_trian\_dat, x\_tset\_dat, y\_trian\_dat, y\_tset\_dat = trian\_tset\_split(x\_trian\_dat, y\_trian\_dat, tset\_size=0.3, random\_state=0)

# trianing the medol with top 3 correlated features wrt temperature

lr = linear\_medol.LinearRegression()

medol = lr.fit(x\_trian\_dat, y\_trian\_dat)

print("R Squared value is: ", medol.score(x\_tset\_dat, y\_tset\_dat))

y\_predictions = medol.predict(x\_tset\_dat)

print('Root Mean Square Error is: ', mean\_squared\_error(y\_tset\_dat, y\_predictions))

**Task 4**

# importing libraries

import matplotlib

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings("ignore")

from sklearn.feature\_extraction.text

import CountVectorizer, TfidfVectorizer

spam\_dat = pd.read\_csv('spam.csv', encoding='latin-1')

print(spam\_dat)

print("\*"\*50)

spam\_dat = spam\_dat.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'], axis=1)

print(spam\_dat)

print("\*"\*50)

# Count Vectorizer vs. Tfidf

# Tfidf

from sklearn.medol\_selection import trian\_tset\_split

vect = TfidfVectorizer()

spam\_dat1 = vect.fit\_transform(spam\_dat.Text)

X\_trian, X\_tset, y\_trian, y\_tset = trian\_tset\_split(spam\_dat1,spam\_dat['Class'], tset\_size=0.2)

print(X\_trian)

print(50\*"++")

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

mnnb = MultinomialNB()

mnnb.fit(X\_trian,y\_trian)

feature\_names = np.array(vect.get\_feature\_names())

y\_pred = mnnb.predict(X\_tset)

print('Accuracy: %.2f%%' % (accuracy\_score(y\_tset, y\_pred) \* 100))

# Count Vectorizer

from sklearn.feature\_extraction.text import CountVectorizer

vect = CountVectorizer()

spam\_dat2 = vect.fit\_transform(spam\_dat.Text)

X\_trian, X\_tset, y\_trian, y\_tset = trian\_tset\_split(spam\_dat2,spam\_dat['Class'], tset\_size=0.2)

medol = MultinomialNB()

medol.fit(X\_trian, y\_trian)

feature\_names = np.array(vect.get\_feature\_names())

y\_pred = medol.predict(X\_tset)

print('Accuracy: %.2f%%' % (accuracy\_score(y\_tset, y\_pred) \* 100))

**Task 5**

# Importing necessary libraries

import warnings

warnings.filterwarnings("ignore")

import pandas as pd

# Reading and loading the dat set

# Dat set considered here is cars.csv

dat = pd.read\_csv('cars.csv', delimiter=',', header=None, skiprows=1, names=['mpg','cylinders','cubicinches','hp','weightlbs','time-to-60','year','brand'])

print(dat.head())

print('='\*50)

# (a) Performing the dat analysis on the dat set

# Count the number of classes in the target 'brand' or top 3 correlated features are printed

print("The top 3 correlated features are given below")

print(dat['brand'].value\_counts(dropna=False))

print('='\*50)

# Handling the Null Values

print("Displaying columns with null values")

nulls = pd.DatFrame(dat.isnull().sum().sort\_values(ascending=False))

nulls.columns = ['Features']

nulls.index.name = 'Nulls count'

print(nulls)

print("No null values are found")

# No nulls were found, so it is not necessary to delete any null values

print('='\*50)

# Visualize dat to analyze our feature correlations

import seaborn as sns

import matplotlib.pyplot as plt

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'mpg').add\_legend()

plt.show()

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'cylinders').add\_legend()

plt.show()

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'cubicinches').add\_legend()

plt.show()

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'hp').add\_legend()

plt.show()

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'weightlbs').add\_legend()

plt.show()

sns.FacetGrid(dat, hue='brand', height=4).map(plt.scatter, 'brand', 'time-to-60').add\_legend()

plt.show()

# Encoding non-numeric features

from sklearn.preprocessing import LabelEncoder

dat = dat.apply(LabelEncoder().fit\_transform)

# Split dat into trian and tset

from sklearn.medol\_selection import trian\_tset\_split

x = dat.drop(['brand'], axis=1)

y = dat['brand']

x\_trian, x\_tset, y\_trian, y\_tset = trian\_tset\_split(x, y, tset\_size =0.2, random\_state=0)

# (b) Applying the three types of classifiers on the dat set

# Naive Bayes method

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report

nb = GaussianNB()

nb.fit(x\_trian, y\_trian)

# Evaluate medol

#.score() for trian dat calculate the difference between y\_trian from medol and accuracy measure y\_trian

score = nb.score(x\_trian, y\_trian)

print('Naive Bayes accuracy trianing score: ', score)

print('Classification report:')

y\_pred = nb.predict(x\_tset)

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

print()

print('='\*50)

# KNN method

from sklearn.neighbors import KNeighborsClassifier

import numpy as np

# Setup arrays to store trianing and tset accuracies

neighbors = np.arange(1, 9)

trian\_accuracy = np.empty(len(neighbors))

tset\_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):

# Setup a knn classifier with k neighbors

knn = KNeighborsClassifier(n\_neighbors=k)

# Fit the medol

knn.fit(x\_trian, y\_trian)

# Compute accuracy on the trianing set

trian\_accuracy[i] = knn.score(x\_trian, y\_trian)

# Compute accuracy on the tset set

tset\_accuracy[i] = knn.score(x\_tset, y\_tset)

#Generate plot

plt.title('k-NN Varying number of neighbors')

plt.plot(neighbors, tset\_accuracy, label='tseting Accuracy')

plt.plot(neighbors, trian\_accuracy, label='trianing accuracy')

plt.legend()

plt.xlabel('Number of neighbors')

plt.ylabel('Accuracy')

plt.show()

# At k=3, the value remains mostly unchanged

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(x\_trian, y\_trian)

#Evaluate medol

score = knn.score(x\_trian, y\_trian)

print('K-Neighbors accuracy trianing score: ', score)

print('Classification report:')

y\_pred = knn.predict(x\_tset)

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

print()

print('='\*50)

#SVM method

from sklearn.svm import SVC

svc = SVC()

svc.fit(x\_trian, y\_trian)

#Evaluate medol

score = svc.score(x\_trian, y\_trian)

print('Support Vector Machines score: ', score)

print('Classification report:')

y\_pred = svc.predict(x\_tset)

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

print('='\*50)

print("From the three classifiers, knn gives the better result")

print('='\*50)

# (c) Applying SVM with linear and non-linear kernel to check for the better performance

#SVM method (Linear)

from sklearn.svm import SVC

svc = SVC(kernel='linear')

svc.fit(x\_trian, y\_trian)

#Evaluate medol

score = svc.score(x\_trian, y\_trian)

print('Support Vector Machines (Linear) score: ', score)

print('Classification report:')

y\_pred = svc.predict(x\_tset)

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

print('='\*50)

#SVM method (Non-Linear)

from sklearn.svm import SVC

svc = SVC(kernel='rbf')

svc.fit(x\_trian, y\_trian)

#Evaluate medol

score = svc.score(x\_trian, y\_trian)

print('Support Vector Machines (Non-Linear) score: ', score)

print('Classification report:')

y\_pred = svc.predict(x\_tset)

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

print('='\*50)

print("SVM with Linear Kernel gives the more accurate result when compared with non-linear kernel")

print('='\*50)