**PROGRAM 11** **CREDIT SCORE CLASSIFICATION**

"""

Credit Score Classification

There are three credit scores that banks and credit card companies use to label their customers:

Good

Standard

Poor

A person with a good credit score will get loans from any bank and financial institution. For the task of Credit Score Classification, we need a labelled dataset with credit scores.

"""

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

import plotly.io as pio

pio.templates.default = "plotly\_white"

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("D:/GEO/BE COURSES/2022 dec/LAB\DATASET/Credit-Score-Data/Credit Score Data/CREDITSCORE.csv")

print(data.head())

print(data.info())

#the dataset has any null values or not:

print(data.isnull().sum())

#The dataset doesn’t have any null values. As this dataset is labelled, let’s have a look at the Credit\_Score column values:

data["Credit\_Score"].value\_counts()

data.shape

data.describe()

#Data Exploration

"""

The dataset has many features that can train a Machine Learning model for credit score classification.

Let’s explore all the features one by one.

I will start by exploring the occupation feature to know if the occupation of the person affects credit scores:

"""

fig = px.box(data,

x="Credit\_Score",

y="Interest\_Rate",

color="Credit\_Score",

title="Credit Scores Based on the Average Interest rates",

color\_discrete\_map={'Poor':'red',

'Standard':'yellow',

'Good':'green'})

fig.update\_traces(quartilemethod="exclusive")

fig.show()

data["Credit\_Mix"] = data["Credit\_Mix"].map({"Standard": 1,

"Good": 2,

"Bad": 0})

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

x = np.array(data[["Annual\_Income", "Monthly\_Inhand\_Salary",

"Num\_Bank\_Accounts", "Num\_Credit\_Card",

"Interest\_Rate", "Num\_of\_Loan",

"Delay\_from\_due\_date", "Num\_of\_Delayed\_Payment",

"Credit\_Mix", "Outstanding\_Debt",

"Credit\_History\_Age", "Monthly\_Balance"]])

#x=data.iloc[:,0:10]

x=np.array(x)

#y = np.array(data[["Credit\_Score"]])

y=data. Credit\_Score

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.33,

random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain, ytrain)

print("Credit Score Prediction : ")

a = float(input("Annual Income: "))

b = float(input("Monthly Inhand Salary: "))

c = float(input("Number of Bank Accounts: "))

d = float(input("Number of Credit cards: "))

e = float(input("Interest rate: "))

f = float(input("Number of Loans: "))

g = float(input("Average number of days delayed by the person: "))

h = float(input("Number of delayed payments: "))

i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")

j = float(input("Outstanding Debt: "))

k = float(input("Credit History Age: "))

l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])

print("Predicted Credit Score = ", model.predict(features))

OBSERVATION:

Credit Score Prediction :

Annual Income: 500000

Monthly Inhand Salary: 6000

Number of Bank Accounts: 1

Number of Credit cards: 1

Interest rate: 1

Number of Loans: 2

Average number of days delayed by the person: 1

Number of delayed payments: 2

Credit Mix (Bad: 0, Standard: 1, Good: 3) : 1

Outstanding Debt: 1

Credit History Age: 1

Monthly Balance: 5000

Predicted Credit Score = ['Standard']

PROGRAM12: IRIS FLOWER CLASSIFICATION USING KNN

# -\*- coding: utf-8 -\*-

"""

Created on Fri Jan 27 09:32:02 2023

KNN

@author: GEOMOL GEORGE

"""

import numpy as np

import pandas as pd

dataset = pd.read\_csv("D:/GEO/BE COURSES/2022 dec/LAB/IRIS.csv")

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

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

dataset.shape

#splitting the dataset into the Training set and Test set

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, random\_state = 42)

a=X\_train.shape

b=X\_test.shape

c=y\_train.shape

d=y\_test.shape

#Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

#Training the K-Nearest Neighbors (K-NN) Classification model on the Training set

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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

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

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

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

print(cm)

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

OBSERVATION:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

PROGRAM 13 CAR PRICE PREDICTION

# -\*- coding: utf-8 -\*-

"""

Created on Tue Jan 24 10:02:09 2023

@author: GEOMOL GEORGE

car price prediction

"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.tree import DecisionTreeRegressor

data = pd.read\_csv("CarPrice.csv")

data.head()

data.shape

data.isnull().sum()

#So this dataset doesn’t have any null values, now let’s look at some of the other important insights to get

#an idea of what kind of data we’re dealing with:

data.info()

data.describe()

data.CarName.unique()

sns.set\_style("whitegrid")

plt.figure(figsize=(15, 10))

sns.distplot(data.price)

plt.show()

#Now let’s have a look at the correlation among all the features of this dataset:

print(data.corr())

plt.figure(figsize=(20, 15))

correlations = data.corr()

sns.heatmap(correlations, cmap="hot", annot=True)

plt.show()

#Training a Car Price Prediction Model

#predict = "price"

x=np.array(data[["symboling", "wheelbase", "carlength",

"carwidth", "carheight", "curbweight",

"enginesize", "boreratio", "stroke",

"compressionratio", "horsepower", "peakrpm",

"citympg", "highwaympg", "price"]])

y=np.array(data.price)

#x = np.array(data.drop([predict], 1))

#=data.iloc[:,0:15]

#y = np.array(data[predict])

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

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2)

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(xtrain, ytrain)

predictions = model.predict(xtest)

from sklearn.metrics import mean\_absolute\_error

model.score(xtest, predictions)

# Calculate and display Mean Absolute Error

mae = mean\_absolute\_error(ytest, predictions)

print("Mean Absolute Error:", mae)

OBSERVATION:

Mean Absolute Error: 328.3536585365854

**PROGRAM 14:HOUSE PRICE PREDICTION**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

dataset = pd.read\_excel("HousePricePrediction.xlsx")

# Printing first 5 records of the dataset

print(dataset.head(5))

dataset.shape

obj = (dataset.dtypes == 'object')

object\_cols = list(obj[obj].index)

print("Categorical variables:",len(object\_cols))

int\_ = (dataset.dtypes == 'int')

num\_cols = list(int\_[int\_].index)

print("Integer variables:",len(num\_cols))

fl = (dataset.dtypes == 'float')

fl\_cols = list(fl[fl].index)

print("Float variables:",len(fl\_cols))

plt.figure(figsize=(12, 6))

sns.heatmap(dataset.corr(),

cmap = 'BrBG',

fmt = '.2f',

linewidths = 2,

annot = True)

unique\_values = []

for col in object\_cols:

unique\_values.append(dataset[col].unique().size)

plt.figure(figsize=(10,6))

plt.title('No. Unique values of Categorical Features')

plt.xticks(rotation=90)

sns.barplot(x=object\_cols,y=unique\_values)

plt.figure(figsize=(18, 36))

plt.title('Categorical Features: Distribution')

plt.xticks(rotation=90)

index = 1

for col in object\_cols:

y = dataset[col].value\_counts()

plt.subplot(11, 4, index)

plt.xticks(rotation=90)

sns.barplot(x=list(y.index), y=y)

index += 1

dataset.drop(['Id'],

axis=1,

inplace=True)

dataset['SalePrice'] = dataset['SalePrice'].fillna(dataset['SalePrice'].mean())

new\_dataset = dataset.dropna()

new\_dataset.isnull().sum()

from sklearn.preprocessing import OneHotEncoder

s = (new\_dataset.dtypes == 'object')

object\_cols = list(s[s].index)

print("Categorical variables:")

print(object\_cols)

print('No. of. categorical features: ',len(object\_cols))

OH\_encoder = OneHotEncoder(sparse=False)

OH\_cols = pd.DataFrame(OH\_encoder.fit\_transform(new\_dataset[object\_cols]))

OH\_cols.index = new\_dataset.index

OH\_cols.columns = OH\_encoder.get\_feature\_names()

df\_final = new\_dataset.drop(object\_cols, axis=1)

df\_final = pd.concat([df\_final, OH\_cols], axis=1)

from sklearn.metrics import mean\_absolute\_error

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

X = df\_final.drop(['SalePrice'], axis=1)

Y = df\_final['SalePrice']

# Split the training set into

# training and validation set

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(X, Y, train\_size=0.8, test\_size=0.2, random\_state=0)

#Model and Accuracy

#svm

from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import mean\_absolute\_percentage\_error

model\_SVR = svm.SVR()

model\_SVR.fit(X\_train,Y\_train)

Y\_pred = model\_SVR.predict(X\_valid)

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

#Random forest

from sklearn.ensemble import RandomForestRegressor

model\_RFR = RandomForestRegressor(n\_estimators=10)

model\_RFR.fit(X\_train, Y\_train)

Y\_pred = model\_RFR.predict(X\_valid)

mean\_absolute\_percentage\_error(Y\_valid, Y\_pred)

#LinearRegression

from sklearn.linear\_model import LinearRegression

model\_LR = LinearRegression()

model\_LR.fit(X\_train, Y\_train)

Y\_pred = model\_LR.predict(X\_valid)

print ('mean\_absolute\_percentage\_error')

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

OBSERVATION: mean\_absolute\_percentage\_error

0.18741683841599993

**PROGRAM 15: NAÏVE IRIS Classification**

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

iris = datasets.load\_iris()

gnb = GaussianNB()

mnb = MultinomialNB()

y\_pred\_gnb = gnb.fit(iris.data, iris.target).predict(iris.data)

cnf\_matrix\_gnb = confusion\_matrix(iris.target, y\_pred\_gnb)

print('Gaussian Naive Bayes classifier Confusion Matrix')

print(cnf\_matrix\_gnb)

y\_pred\_mnb = mnb.fit(iris.data, iris.target).predict(iris.data)

cnf\_matrix\_mnb = confusion\_matrix(iris.target, y\_pred\_mnb)

print('Multinomial Naive Bayes classifier Confusion Matrix')

print(cnf\_matrix\_mnb)

OBSERVATION:

Gaussian Naive Bayes classifier Confusion Matrix

[[50 0 0]

[ 0 47 3]

[ 0 3 47]]

Multinomial Naive Bayes classifier Confusion Matrix

[[50 0 0]

[ 0 46 4]

[ 0 3 47]]