EXP NO: 01

FIND S-ALGORITHM

import csv

a = []

with open("C:/Users/ramya/Downloads/enjoysport (1) (1).csv", 'r') as csvfile:

next(csvfile)

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\nThe total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

print("\nThe initial hypothesis is : ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

print ("\nInstance ", i+1, "is", a[i], " and is Positive Instance")

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

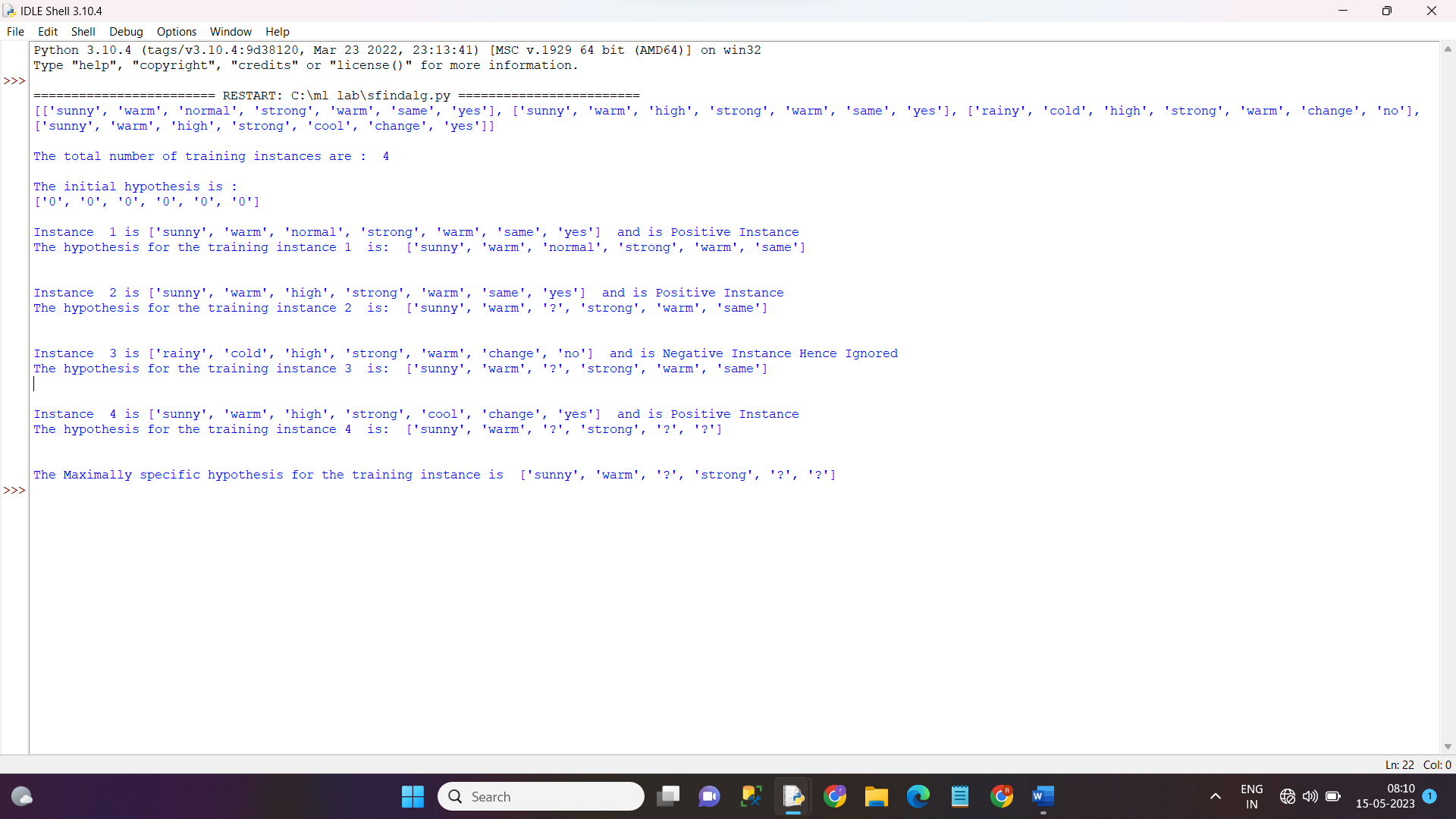
print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")

if a[i][num\_attribute] == 'no':

print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")

print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")

print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)

OUTPUT: 

EXP NO:02

import numpy as np

import pandas as pd

data = pd.read\_csv("C:\ml lab\enjoysport (1) (1).csv")

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":

print("Instance is Positive ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

specific\_h[x] ='?'

general\_h[x][x] ='?'

if target[i] == "no":

print("Instance is Negative ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

general\_h[x][x] = specific\_h[x]

else:

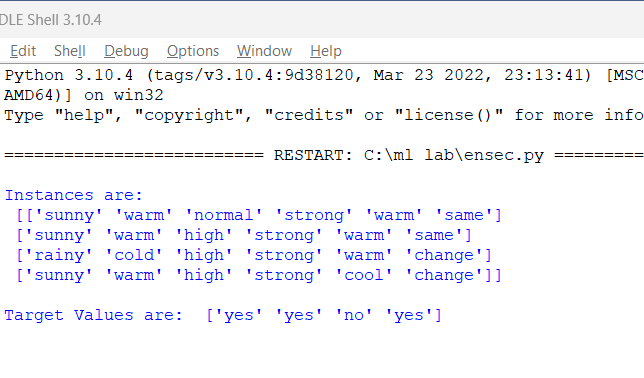
general\_h[x][x] = '?'

print("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

OUTPUT:



EXP NO : 03

import pandas as pd

import math

import numpy as np

data = pd.read\_csv('C:/ml lab/3-dataset.csv')

features = [feat for feat in data]

features.remove("answer")

# Create a class named Node with four members children, value, isLeaf and pred.

class Node:

def \_\_init\_\_(self):

self.children = []

self.value = ""

self.isLeaf = False

self.pred = ""

# Define a function called entropy to find the entropy oof the dataset

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

# Define a function named info\_gain to find the gain of the attribute

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

#print ("\n",uniq)

gain = entropy(examples)

#print ("\n",gain)

for u in uniq:

subdata = examples[examples[attr] == u]

#print ("\n",subdata)

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

#print ("\n",gain)

return gain

# Define a function named ID3 to get the decision tree for the given dataset

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

#print ("\n",examples)

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

#print ("\nMax feature attr",max\_feat)

uniq = np.unique(examples[max\_feat])

#print ("\n",uniq)

for u in uniq:

#print ("\n",u)

subdata = examples[examples[max\_feat] == u]

#print ("\n",subdata)

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

# Define a function named printTree to draw the decision tree

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

# Define a function named classify to classify the new example

def classify(root: Node, new):

for child in root.children:

if child.value == new[root.value]:

if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred)

exit

else:

classify (child.children[0], new)

# Finally, call the ID3, printTree and classify functions

root = ID3(data, features)

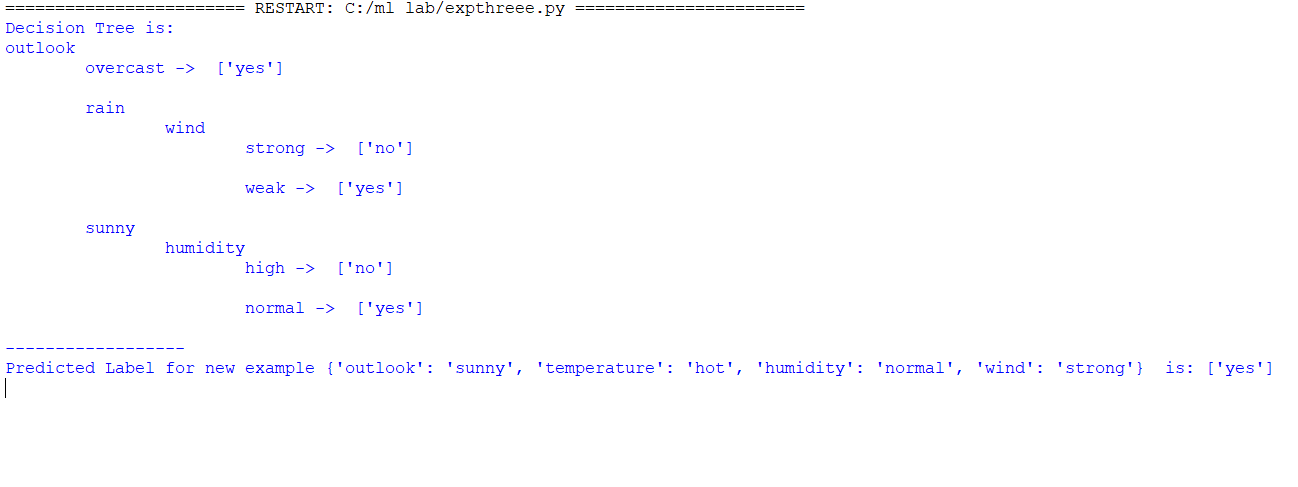
print("Decision Tree is:")

printTree(root)

print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}

classify (root, new)

output:

EXP NO:-04

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

def sigmoid (x):

return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

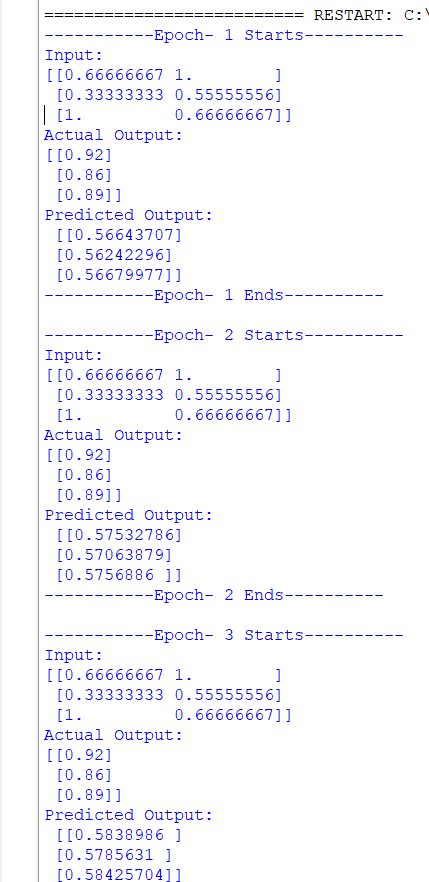
print ("-----------Epoch-", i+1, "Ends----------\n")

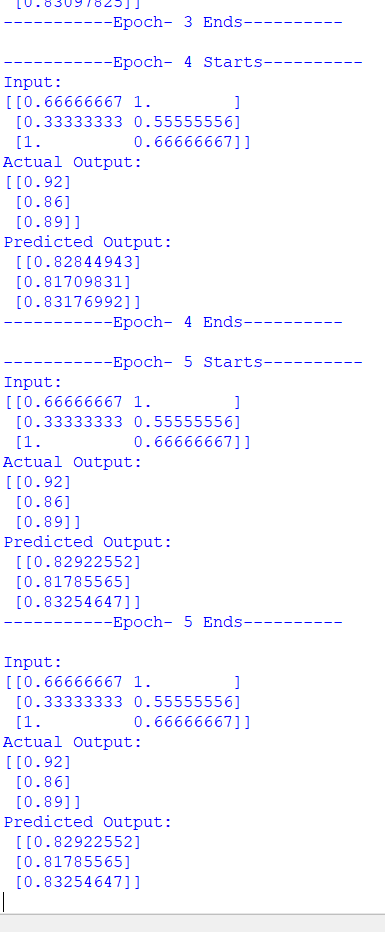
print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

OUTPUT:





EXP NO :05

import numpy as np

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/mobile\_prices.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)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.neighbors import KNeighborsClassifier

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

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

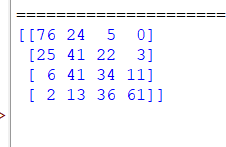
y\_pred = classifier.predict(X\_test)

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

print(cm)

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

OUTPUT:



EXP NO:06

import numpy as np

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/breastcancer.csv')

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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.naive\_bayes import GaussianNB

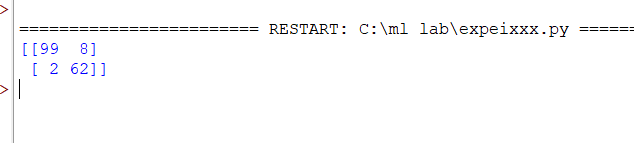
classifier = GaussianNB()

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

GaussianNB(priors=None, var\_smoothing=1e-09)

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

OUTPUT:

:

EXP NO:07

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/Salary\_Data (1).csv')

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

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

dataset.head()

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

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

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

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

pd.DataFrame(data={'Actuals': y\_test, 'Predictions': y\_pred})

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

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

OUTPUT:



EXP NO:08

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/Position\_Salaries.csv')

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

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

from sklearn.linear\_model import LinearRegression

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

from sklearn.preprocessing import PolynomialFeatures

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg\_2 = LinearRegression()

lin\_reg\_2.fit(X\_poly, y)

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg.predict(X), color = 'blue')

plt.title('Truth or Bluff (Linear Regression)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue')

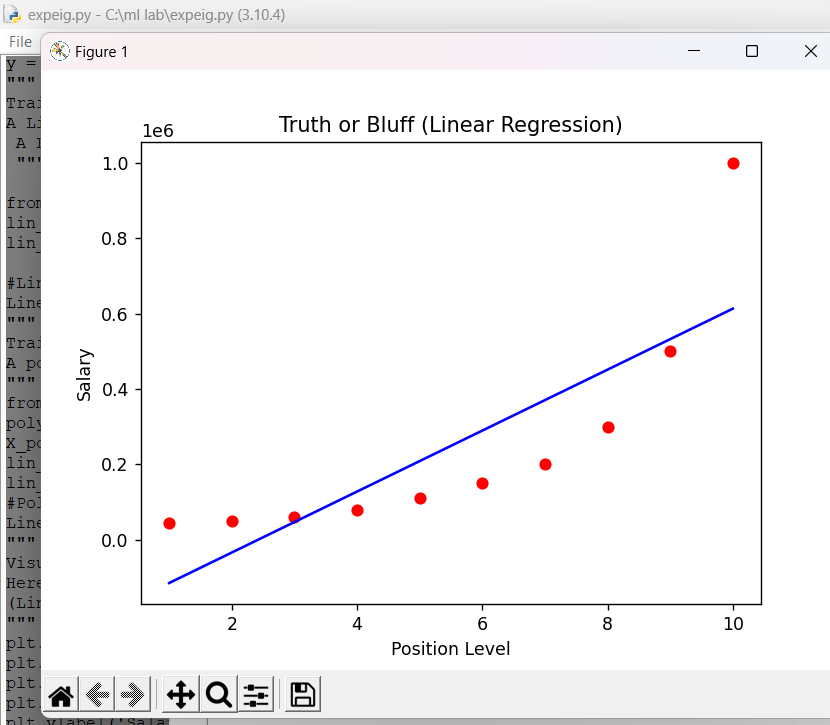
plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

OUTPUT:



EXP NO:09

import numpy as np

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/breastcancer.csv')

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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 2)

from sklearn.preprocessing import StandardScaler

rom sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

#Logistic Regression (LR) classifier model

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=0, solver='warn', tol=0.0001, verbose=0,

warm\_start=False)

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

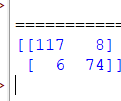
y\_pred = classifier.predict(X\_test)

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

print(cm)

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

output:



EXP NO:12

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

iris = pd.read\_csv('C:/ml lab/IRIS.csv')

#first five rows of this dataset:

print(iris.head())

print(iris.describe())

#The target labels of this dataset are present in the species column, let’s have a quick look at the target labels:

print("Target Labels", iris["species"].unique())

import plotly.io as io

io.renderers.default='browser'

import plotly.express as px

fig = px.scatter(iris, x="sepal\_width", y="sepal\_length", color="species")

fig.show()

x = iris.drop("species", axis=1)

y = iris["species"]

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,random\_state=42)

from sklearn.neighbors import KNeighborsClassifier

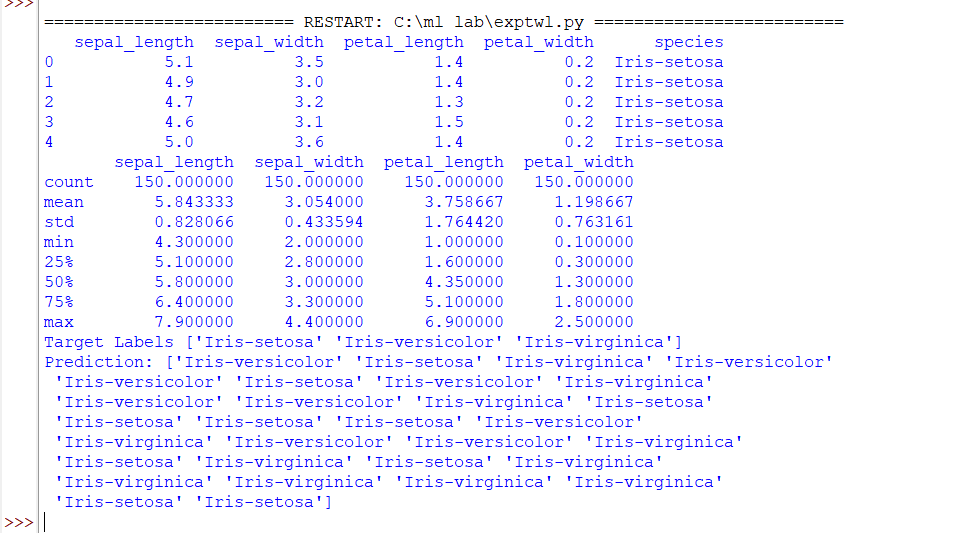
knn = KNeighborsClassifier()

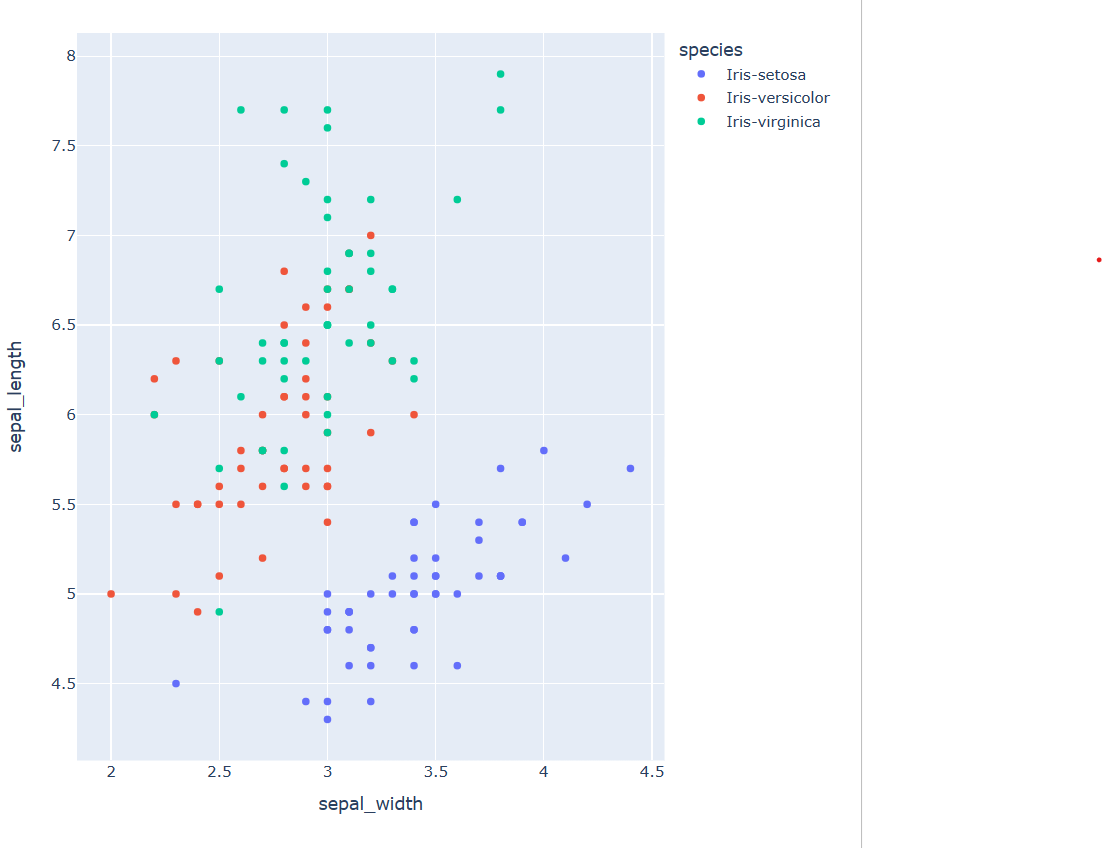
knn.fit(x\_train, y\_train)

prediction = knn.predict(x\_test)

print("Prediction: {}".format(prediction))

OUTPUT:





EXP NO :13

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('C:/ml lab/CarPrice.csv')

data.head()

data.shape

data.isnull().sum()

data.info()

data.describe()

data.CarName.unique()

sns.set\_style("whitegrid")

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

sns.histplot(data.price)

plt.show()

print(data.corr())

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

correlations = data.corr()

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

plt.show()

llllllllllllllllllllllllllllllllllllpredict = "price"

data = data[["symboling", "wheelbase", "carlength",

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

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

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

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

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

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()

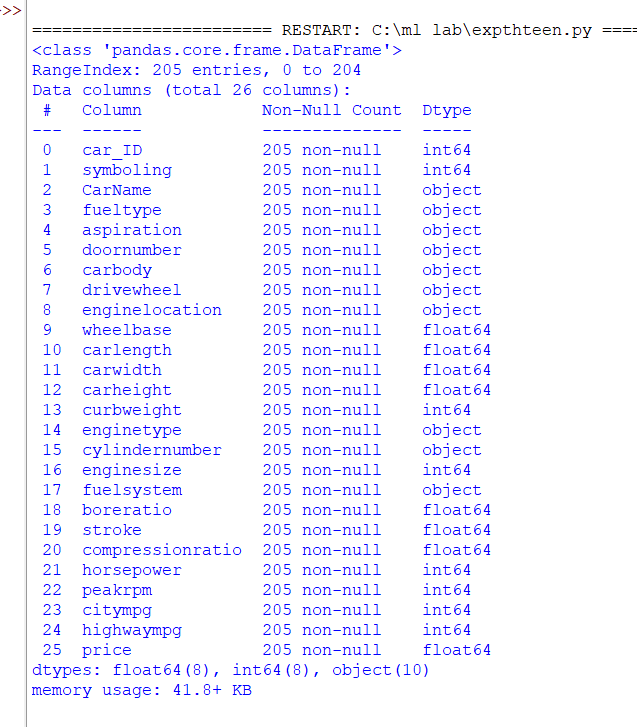
model.fit(xtrain, ytrain)

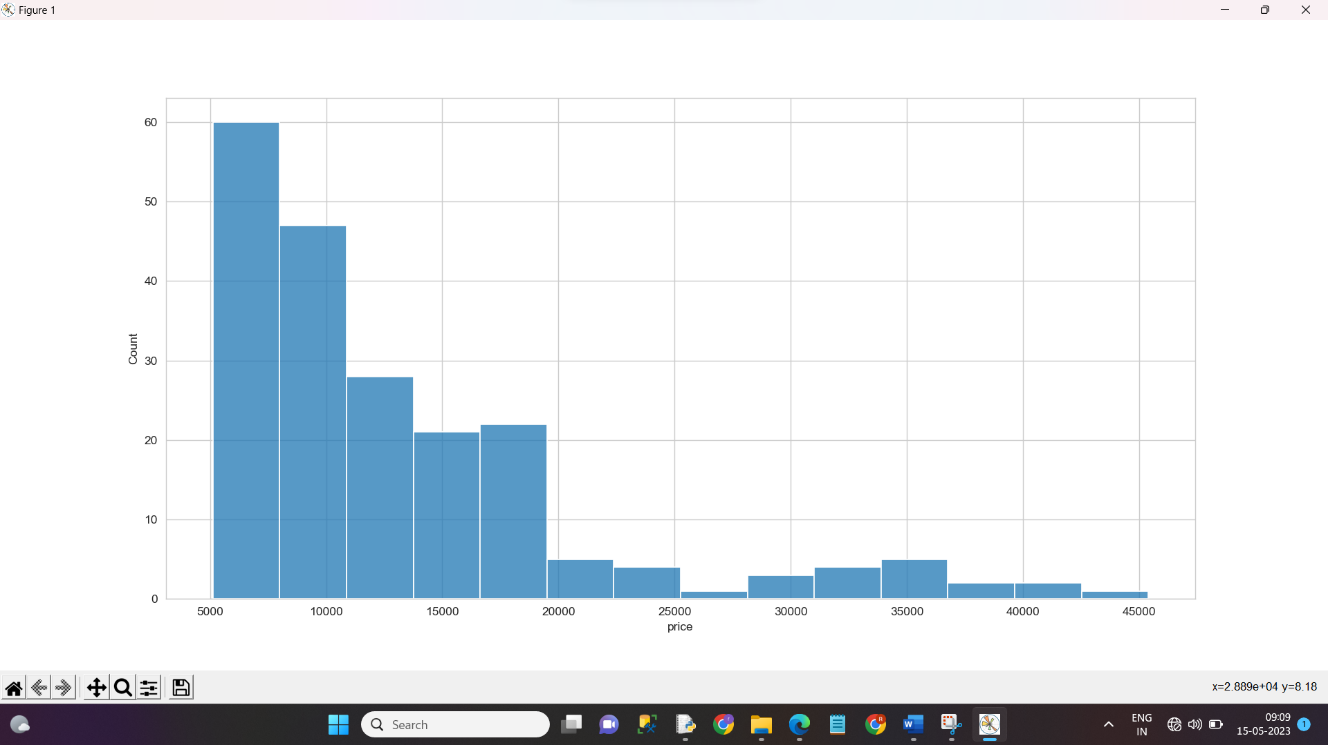
predictions = model.predict(xtest)

from sklearn.metrics import mean\_absolute\_error

model.score(xtest, predictions)

OUTPUT:





EXP NO:14

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

dataset = pd.read\_csv('C:/ml lab/HousePricePrediction.csv')

from sklearn.feature\_extraction.text import CountVectorizer

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\_out()

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

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

text\_data = ['This is the first document.',

'This document is the second document.',

'And this is the third one.',

'Is this the first document?']

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(text\_data)

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']

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

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))

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)

from sklearn.linear\_model import LinearRegression

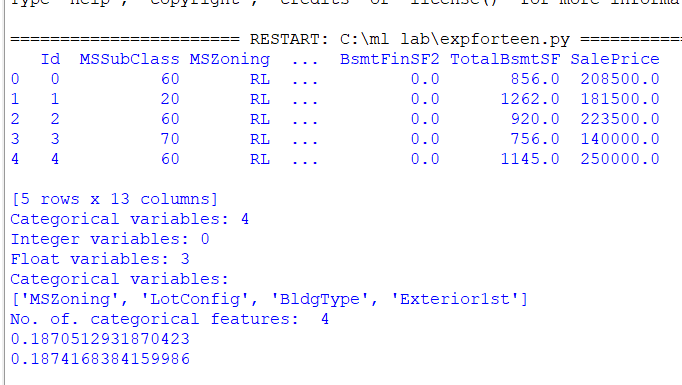
model\_LR = LinearRegression()

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

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

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

OUTPUT:



EXPNO:15

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)

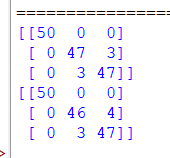
print(cnf\_matrix\_gnb)

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

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

print(cnf\_matrix\_mnb)

OUTPUT:



EXP NO:16

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import PassiveAggressiveClassifier

from sklearn.metrics import classification\_report

iris = pd.read\_csv('C:/ml lab/IRIS.csv')

print(iris.head())

x = iris.drop("species", axis=1)

y = iris["species"]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=10, random\_state=42)

decision\_tree = DecisionTreeClassifier()

logistic\_regression = LogisticRegression(max\_iter=1000) # Increase max\_iter value

k\_nearest\_classifier = KNeighborsClassifier()

bernoulli\_naive\_bayes = BernoulliNB()

passive\_aggressive = PassiveAggressiveClassifier()

k\_nearest\_classifier.fit(x\_train, y\_train)

decision\_tree.fit(x\_train, y\_train)

logistic\_regression.fit(x\_train, y\_train)

passive\_aggressive.fit(x\_train, y\_train)

data1 = {

"Classification Algorithms": [

"KNN Classifier",

"Decision Tree Classifier",

"Logistic Regression",

"Passive Aggressive Classifier"

],

"Score": [

k\_nearest\_classifier.score(x, y),

decision\_tree.score(x, y),

logistic\_regression.score(x, y),

passive\_aggressive.score(x, y)

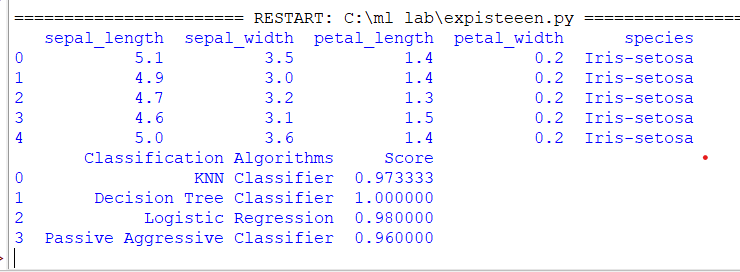
]

}

score = pd.DataFrame(data1)

print(score)

OUTPUT:



EXP NO:17

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.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

sns.set()

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv('C:/ml lab/mobile\_prices.csv')

print(data.head())

plt.figure(figsize=(12, 10))

sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation

x = data.iloc[:, :-1].values

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

x = StandardScaler().fit\_transform(x)

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

# Logistic Regression algorithm provided by Scikit-learn:

from sklearn.linear\_model import LogisticRegression

lreg = LogisticRegression()

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

y\_pred = lreg.predict(x\_test)

#accuracy of the model:

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

print("Accuracy of the Logistic Regression Model: ",accuracy)

#predictions made by the model:

print(y\_pred)

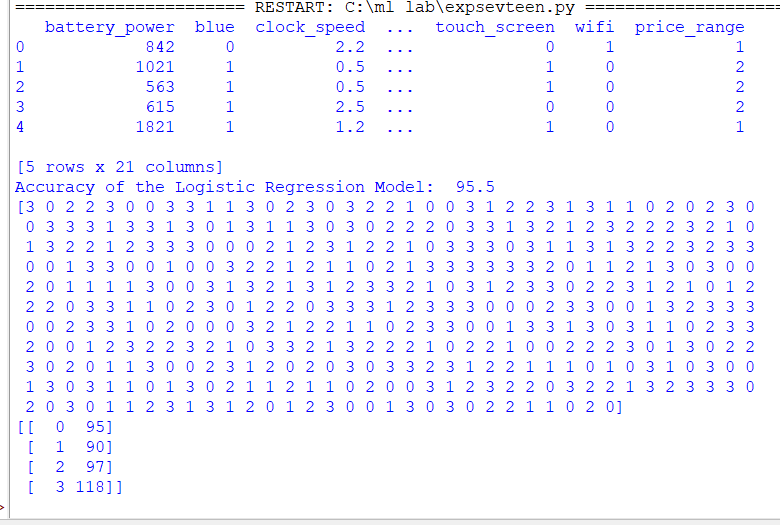
#Let’s have a look at the number of mobile phones classified for each price range:

(unique, counts) = np.unique(y\_pred, return\_counts=True)

price\_range = np.asarray((unique, counts)).T

print(price\_range)

OUTPUT:



EXP NO:11

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('C:/ml lab/CREDITSCORE.csv')

print(data.head())

print(data.info())

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

fig = px.box(data,

x="Occupation",

color="Credit\_Score",

title="Credit Scores Based on Occupation",

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

'Standard':'yellow',

'Good':'green'})

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Annual\_Income",

color="Credit\_Score",

title="Credit Scores Based on Annual Income",

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

'Standard':'yellow',

'Good':'green'})

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

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Monthly\_Inhand\_Salary",

color="Credit\_Score",

title="Credit Scores Based on Monthly Inhand Salary",

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

'Standard':'yellow',

'Good':'green'})

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

fig.show()

fig = px.box(data,

x="Credit\_Score",

y="Num\_Bank\_Accounts",

color="Credit\_Score",

title="Credit Scores Based on Number of Bank Accounts",

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

'Standard':'yellow',

'Good':'green'})

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

fig.show()

# impact on credit scores based on the number of credit cards you have:

fig = px.box(data,

x="Credit\_Score",

y="Num\_Credit\_Card",

color="Credit\_Score",

title="Credit Scores Based on Number of Credit cards",

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

'Standard':'yellow',

'Good':'green'})

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

fig.show()

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"]])

y = np.array(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))

output:

========================================================================= RESTART: C:/ml lab/explaavan.py ===========================================================================

ID Customer\_ID ... Monthly\_Balance Credit\_Score

0 5634 3392 ... 312.494089 Good

1 5635 3392 ... 284.629162 Good

2 5636 3392 ... 331.209863 Good

3 5637 3392 ... 223.451310 Good

4 5638 3392 ... 341.489231 Good

[5 rows x 28 columns]

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 100000 entries, 0 to 99999

Data columns (total 28 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 100000 non-null int64

1 Customer\_ID 100000 non-null int64

2 Month 100000 non-null int64

3 Name 100000 non-null object

4 Age 100000 non-null float64

5 SSN 100000 non-null float64

6 Occupation 100000 non-null object

7 Annual\_Income 100000 non-null float64

8 Monthly\_Inhand\_Salary 100000 non-null float64

9 Num\_Bank\_Accounts 100000 non-null float64

10 Num\_Credit\_Card 100000 non-null float64

11 Interest\_Rate 100000 non-null float64

12 Num\_of\_Loan 100000 non-null float64

13 Type\_of\_Loan 100000 non-null object

14 Delay\_from\_due\_date 100000 non-null float64

15 Num\_of\_Delayed\_Payment 100000 non-null float64

16 Changed\_Credit\_Limit 100000 non-null float64

17 Num\_Credit\_Inquiries 100000 non-null float64

18 Credit\_Mix 100000 non-null object

19 Outstanding\_Debt 100000 non-null float64

20 Credit\_Utilization\_Ratio 100000 non-null float64

21 Credit\_History\_Age 100000 non-null float64

22 Payment\_of\_Min\_Amount 100000 non-null object

23 Total\_EMI\_per\_month 100000 non-null float64

24 Amount\_invested\_monthly 100000 non-null float64

25 Payment\_Behaviour 100000 non-null object

26 Monthly\_Balance 100000 non-null float64

27 Credit\_Score 100000 non-null object

dtypes: float64(18), int64(3), object(7)

memory usage: 21.4+ MB

None

ID 0

Customer\_ID 0

Month 0

Name 0

Age 0

SSN 0

Occupation 0

Annual\_Income 0

Monthly\_Inhand\_Salary 0

Num\_Bank\_Accounts 0

Num\_Credit\_Card 0

Interest\_Rate 0

Num\_of\_Loan 0

Type\_of\_Loan 0

Delay\_from\_due\_date 0

Num\_of\_Delayed\_Payment 0

Changed\_Credit\_Limit 0

Num\_Credit\_Inquiries 0

Credit\_Mix 0

Outstanding\_Debt 0

Credit\_Utilization\_Ratio 0

Credit\_History\_Age 0

Payment\_of\_Min\_Amount 0

Total\_EMI\_per\_month 0

Amount\_invested\_monthly 0

Payment\_Behaviour 0

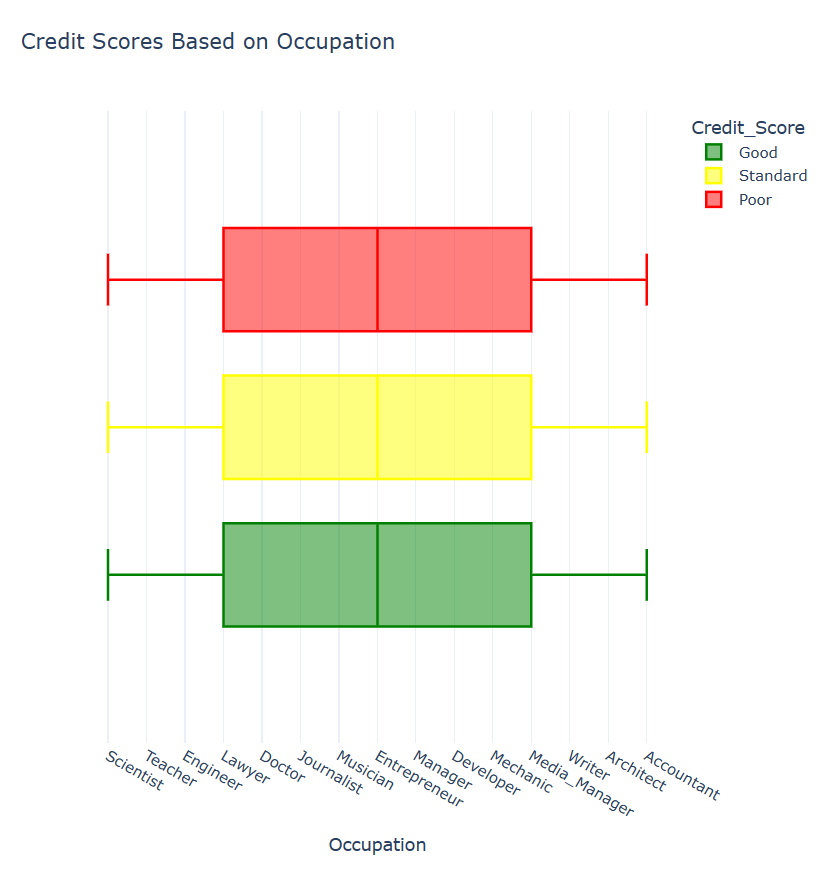
Monthly\_Balance 0

Credit\_Score 0

dtype: int64

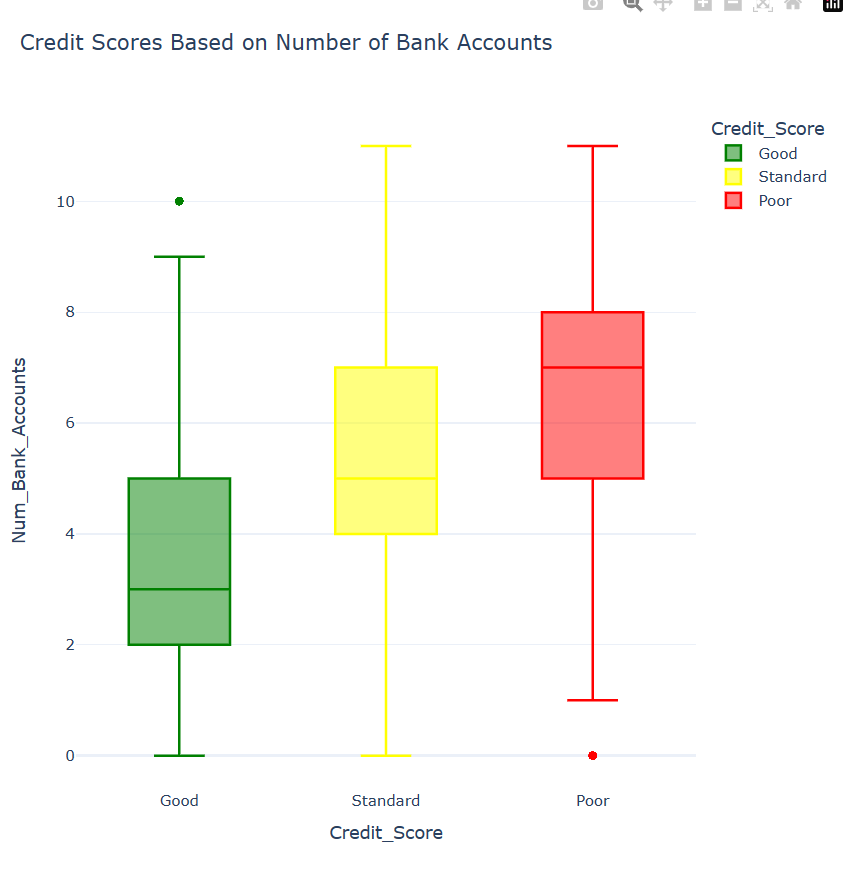
Credit Score Prediction :

Annual Income:









EXP NO:18

from sklearn import datasets

import numpy as np

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

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data[:, [2, 3]]

y = iris.target

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

X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

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

ppn = Perceptron(eta0=0.1, random\_state=1)

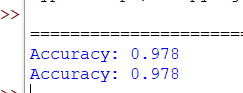
ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

print('Accuracy: %.3f' % ppn.score(X\_test\_std, y\_test))

output:



EXPNO:19

import numpy as np

import pandas as pd

dataset = pd.read\_csv('C:/ml lab/breastcancer.csv')

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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

built using a decision tree classifier.

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

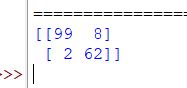
y\_pred = classifier.predict(X\_test)

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

print(cm)

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

OUTPUT:



EXPNO:10

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv('C:/ml lab/3-dataset.csv')

X = pd.get\_dummies(dataset.iloc[:, :-1])

y = dataset['answer'].map({'no': 0, 'yes': 1})

plt.figure(figsize=(14, 7))

colormap = np.array(['red', 'lime'])

plt.subplot(1, 3, 1)

plt.title('Real')

plt.scatter(X['outlook\_sunny'], X['humidity\_high'], c=colormap[y])

gmm = GaussianMixture(n\_components=2, random\_state=0).fit(X)

y\_cluster\_gmm = gmm.predict(X)

plt.subplot(1, 3, 3)

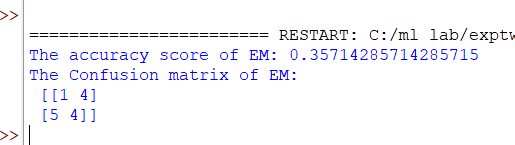
plt.title('GMM Classification')

plt.scatter(X['outlook\_sunny'], X['humidity\_high'], c=colormap[y\_cluster\_gmm])

print('The accuracy score of EM:', metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n', metrics.confusion\_matrix(y, y\_cluster\_gmm))

output:



EXPNO:20

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

import plotly.io as io

io.renderers.default = 'browser'

data = pd.read\_csv('C:/ml lab/futuresale prediction.csv')

print(data.head())

print(data.sample(5))

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

import plotly.express as px

import plotly.graph\_objects as go

figure = px.scatter(data\_frame=data, x="Sales", y="TV", size="TV", trendline="ols")

figure.show()

figure = px.scatter(data\_frame=data, x="Sales", y="Newspaper", size="Newspaper", trendline="ols")

figure.show()

figure = px.scatter(data\_frame=data, x="Sales", y="Radio", size="Radio", trendline="ols")

figure.show()

correlation = data.corr()

print(correlation["Sales"].sort\_values(ascending=False))

x = np.array(data.drop(["Sales"], axis=1))

y = np.array(data["Sales"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))

features = np.array([[230.1, 37.8, 69.2]]) # Modify the values according to your needs

print(model.predict(features))

OUTPUT:

