**PROGRAM 1:**

# Step 1 : Importing dataset and libraries

import csv

a = []

with open('enjoysport.csv','r') as csvfile:

for i in csv.reader(csvfile):

a.append(i)

# Step 2 : Finding total no. of attributes and decalring initial hypothesis

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

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

# Step 3 : Main Algorithm

print("Initial Hypothesis, H0 : ")

print(hypothesis)

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

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

print("\nInstance ", i+1, "is", a[i], "is +ve")

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("Hypothesis " , i+1 , " : ", hypothesis)

print()

elif(a[i][num\_attribute] == 'no'):

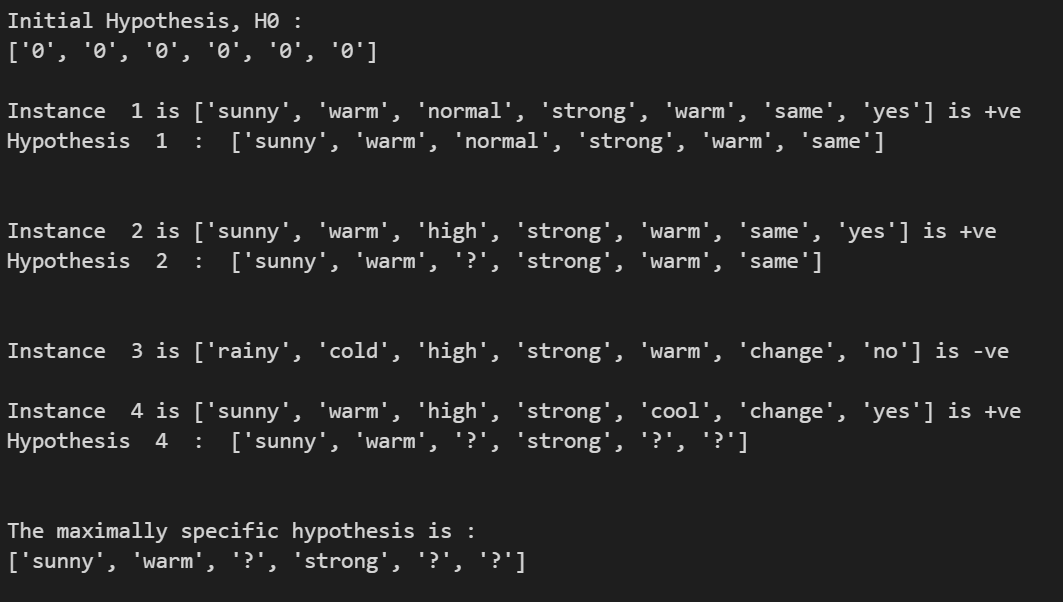
print("\nInstance ", i+1, "is", a[i], "is -ve ")

print()

print("The maximally specific hypothesis is : ")

print(hypothesis)

**OUTPUT:**

****

**PROGRAM 2:**

### Step 1 : Importing necessary libraries and dataset

import csv

a = []

with open('enjoysport.csv','r') as dataset:

reader = csv.reader(dataset)

for row in reader:

a.append(row).

print(row)

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

### Step 2 : Declaring initial hypothesis - general and specific

s = ['0'] \* num\_attributes

g = ['?'] \* num\_attributes

print("Most Specific hypothesis S0 : " + str(s))

print("Most General hypothesis G0 : " + str(g))

### Step 3 : Creating a version space

It will contain the final valid hypothesis for the given data.

version\_space = []

### Step 4 : Writing the main algorithm

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

if(a[i][num\_attributes] == 'yes'):

print("Instance " + str(i+1) + " +ve ")

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

if (s[j] == '0' or s[j] == a[i][j]):

s[j] = a[i][j]

else:

s[j] = '?'

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

for k in range(1,len(version\_space)):

if(version\_space[k][j] != '?' and version\_space[k][j]!=s[j]):

del version\_space[k]

print("S" + str(i+1) , s)

print("G" + str(i+1) , version\_space)

if(a[i][num\_attributes] == 'no'):

print("Instance " + str(i+1) + " -ve ")

print("S" + str(i+1),s)

print("G" + str(i+1))

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

if(s[j]!=a[i][j] and s[j] !='?'):

g[j] = s[j]

#appending the generic hypothesis

version\_space.append(g)

#resetting the generic hypothesis to [?,?,?,?,?,?]

g = ['?']\*num\_attributes

print(version\_space)

print()

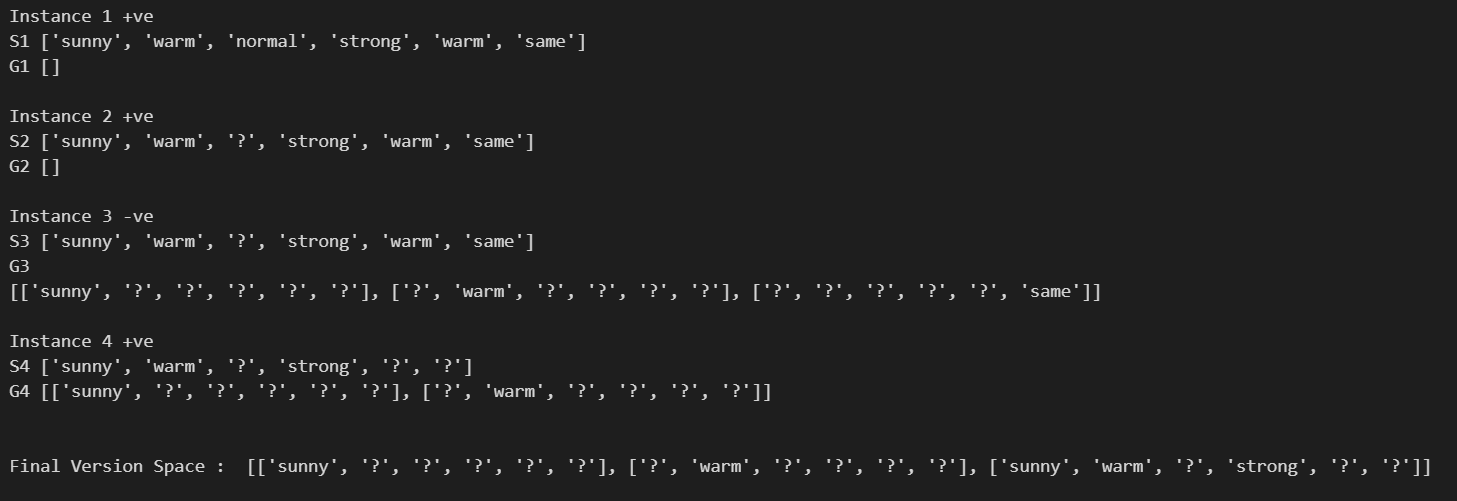
# appending the specific hypothesis

version\_space.append(s)

print()

print("Final Version Space : " , version\_space)

**OUTPUT:**

****

**PROGRAM 4:**

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)

Y = Y/100

def sigmoid(x):

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

def sigmoid\_grad(x):

return x\*(1-x)

epoch = 1000

eta = 0.2

input\_neurons = 2

hidden\_neurons = 3

output\_neurons = 1

wh = np.random.uniform(size=(input\_neurons,hidden\_neurons))

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

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

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

for i in range(epoch):

h\_ip = np.dot(X,wh) + bh

h\_act = sigmoid(h\_ip)

o\_ip = np.dot(h\_act,wout) + bout

output = sigmoid(o\_ip)

Eo = Y - output

outgrad = sigmoid\_grad(output)

d\_output = Eo\*outgrad

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

hiddengrad = sigmoid\_grad(h\_act)

d\_hidden = Eh \* hiddengrad

wout += h\_act.T.dot(d\_output)\*eta

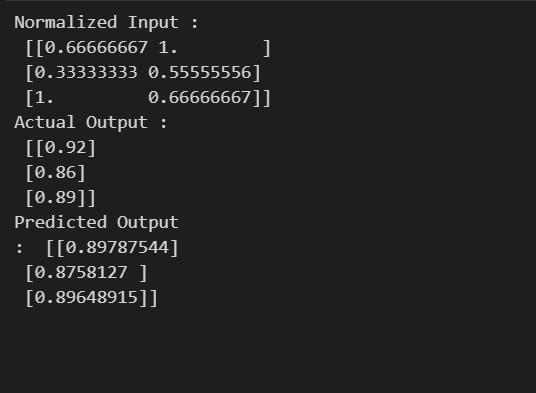
wh+=X.T.dot(d\_output)\*eta

print("Normalized Input : \n", str(X))

print("Actual Output : \n", str(Y))

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

**OUTPUT:**



**PROGRAM 5:**

### Step 1 : Importing Library and dataset

import math as m

result = [[9.2,85,8,"pass"],

[8,80,7,"pass"],

[8.5,81,8,"pass"],

[6,45,5,"fail"],

[6.5,50,4,"fail"],

[8.2,72,7,"pass"],

[5.8,38,5,"fail"],

[8.9,91,9,"pass"]]

g = [7.6,60,8]

k = int(input("Enter K : "))

no\_attr = len(result[0]) - 1

distance = []

### Step 3 : Finding distances

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

x = 0

for j in range(0,no\_attr):

x = x + m.pow(g[j]- result[i][j], 2)

#we append the distances of every instance on the main list(result) to use it in future

result[i].append(m.sqrt(x))

distance.append(m.sqrt(x))

# we sort the distance list to find the nearest k distances

distance.sort()

### Step 4 : Finding nearest distances

NN = []

pass\_ = 0

fail\_ = 0

for i in range(0,k):

NN.append(distance[i])

for j in range(0,k):

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

if(result[i][len(result[0]) - 1] == NN[j]):

if(result[i][len(result[0]) - 2] == "pass"):

pass\_ = pass\_ + 1

else:

fail\_ = fail\_ + 1

### Step 5 : Printing the nearest neighbours and result

print("Nearest Neighbours (distances): " + str(NN))

if(pass\_ > fail\_ ):

print("Outcome : Pass")

else:

print("Outcome : Fail")

**OUTPUT:**

Nearest Neighbours (distances): [10.82635672791175, 12.056533498481228, 15.380507143784303, 20.028979005431108, 21.01927686672403]

Outcome : Pass

**PROGRAM 6:**

import csv

a = []

with open('play\_tennis.csv','r') as dataset:

for i in csv.reader(dataset):

a.append(i)

a.pop(0)

print(a)

case = []

no\_attributes = len(a[0]) - 2

for i in range(0,no\_attributes):

x = input("Attribute " + str(i+1))

case.append(x)

print("The given case is : " + str(case))

positive = 0

negative = 0

# finding positive and negative instances

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

if(a[i][len(a[i]) - 1] == "Yes"):

positive = positive + 1

if(a[i][len(a[i]) - 1] == "No"):

negative = negative + 1

print(positive)

print(negative)

#finding positive and negative probabilities

prob\_pos = positive/len(a)

prob\_neg = negative/len(a)

NB\_pos = prob\_pos

NB\_neg = prob\_neg

j = 1

count\_pos = 0

count\_neg = 0

for i in range(1,no\_attributes+1):

count\_pos = 0

count\_neg = 0

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

if case[i-1] in a[j]:

if(a[j][len(a[0])-1] == "Yes"):

count\_pos = count\_pos + 1

if(a[j][len(a[0])-1] == "No"):

count\_neg = count\_neg + 1

# print(count\_pos,count\_neg)

x = count\_pos/positive

y = count\_neg/negative

NB\_pos = NB\_pos \* x

NB\_neg = NB\_neg \* y

if(NB\_pos > NB\_neg) :

print(str(case) + " corresponds to YES")

else:

print(str(case) + "corresponds to NO")

**OUTPUT :**

['Sunny', 'Cool', 'Normal', 'Strong'] corresponds to YES

**PROGRAM 7:**

import matplotlib.pyplot as plt

from scipy import stats

import pandas as pd

dataset = pd.read\_csv('Position\_Salaries.csv')

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

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

print("levels :", x)

print("Salaries : ", y)

std\_err = stats.linregress(x,y)

def myfunc(x):

return slope\*x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x,y)

plt.plot(x,mymodel)

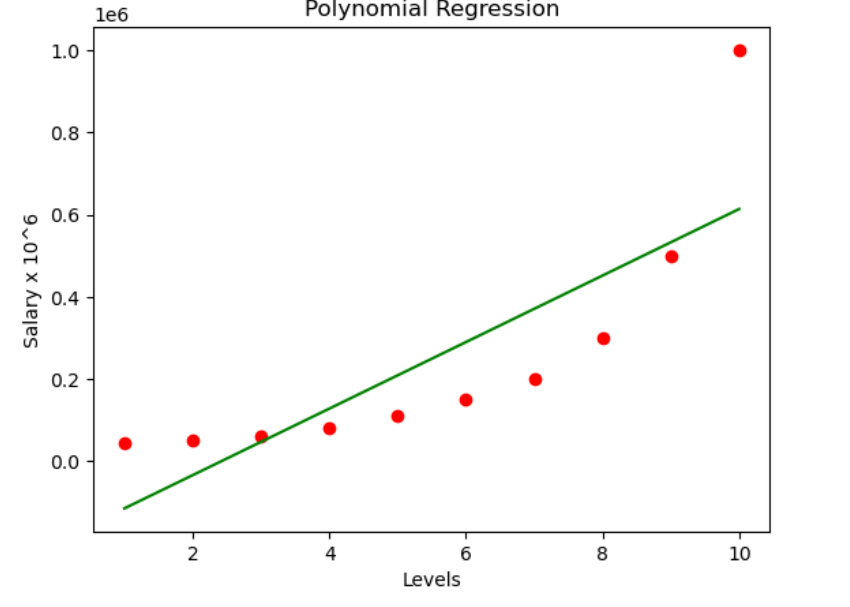
plt.title('Salary vs Experience')

plt.xlabel('Years of experience')

plt.ylabel('Salary')

plt.show()

**OUTPUT:**



**PROGRAM 8:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

datas = pd.read\_csv('Position\_Salaries.csv')

x = datas.iloc[:, 1:2].values

y = datas.iloc[:, 2].values

from sklearn.linear\_model import LinearRegression

lin = LinearRegression()

lin.fit(x, y)

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)

x\_poly = poly.fit\_transform(x)

poly.fit(x\_poly,y)

lin = LinearRegression()

lin.fit(x\_poly,y)

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

plt.plot(x, lin.predict(poly.fit\_transform(x)), color = 'green')

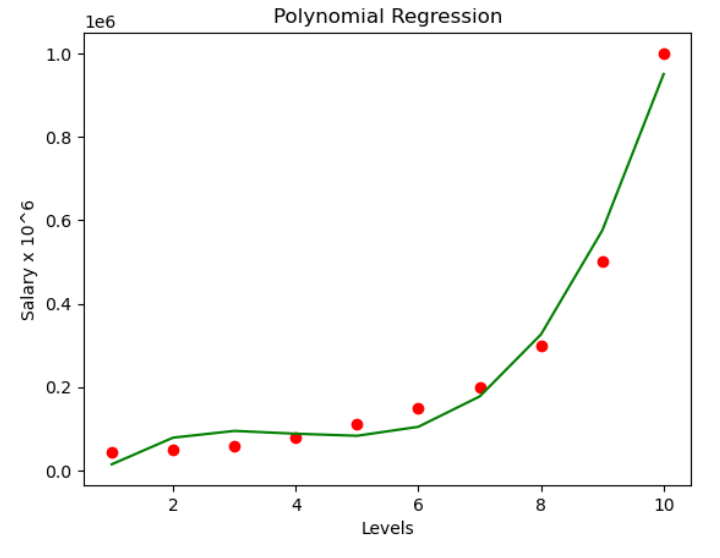
plt.title('Polynomial Regression')

plt.xlabel('Levels')

plt.ylabel('Salary x 10^6')

plt.show()

**OUTPUT:**



**PROGRAM 9:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("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

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)

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:**

[[117 8]

[ 6 74]]

0.9317073170731708

**PROGRAM 10:**

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

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

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

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

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

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

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.petal\_length,X.petal\_width,c=colormap[y])

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

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

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.petal\_length,X.petal\_width,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:**

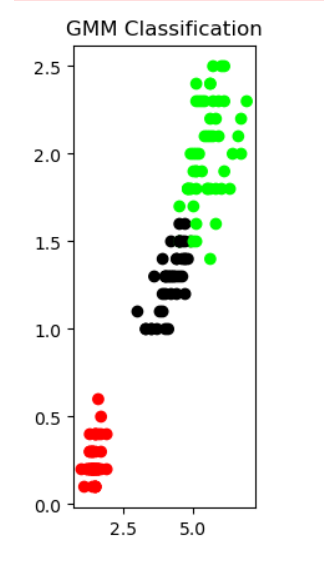
The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]

[ 0 5 45]

[ 0 50 0]]

****

**PROGRAM 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"

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

print(data.head())

print(data.info())

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: "))

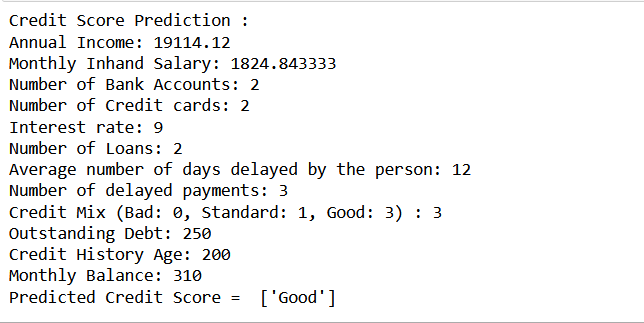
k = float(input("Credit History Age: "))

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 :**

****

**PROGRAM 12 :**

**# Importing libraries and functions**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

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

**# Dataset Exploration**

print(iris.head())

print()

print(iris.describe())

**# Identifying the unique values of the result.**

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

import plotly.io as io

import plotly.express as px

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

fig.show()

**#Seggregating dataset**

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=0)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=1)

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

x\_new = np.array([[6, 2.9, 1, 0.2]])

prediction = knn.predict(x\_new)

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

**OUTPUT:**

Prediction: ['Iris-setosa']

**PROGRAM 13:**

**#Importing packages and functions**

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

**#Importing the dataset**

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

**#Data Exploration**

data.head()

data.shape

data.isnull().sum() #Checking if the dataset has NULL Values

data.info()

data.describe()

data.CarName.unique()

**#Analysing correlations & using heatmap**

print(data.corr())

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

correlations = data.corr()

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

plt.show()

**#Training a Car Price Prediction Model**

predict = "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:**

1.0

**PROGRAM 14:**

**#Importing Libraries and functions**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**#Importing Dataset**

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

**#Exploring 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']

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))

#LinearRegression

from sklearn.linear\_model import LinearRegression

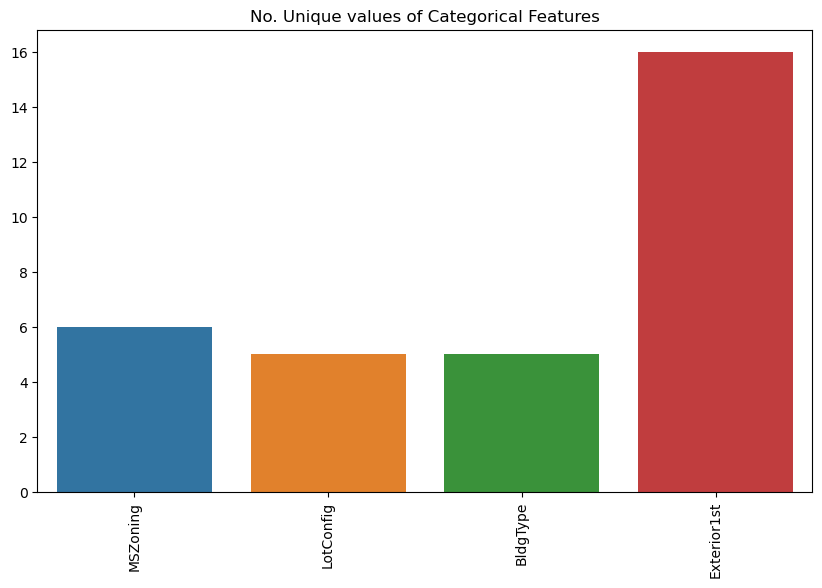
model\_LR = LinearRegression()

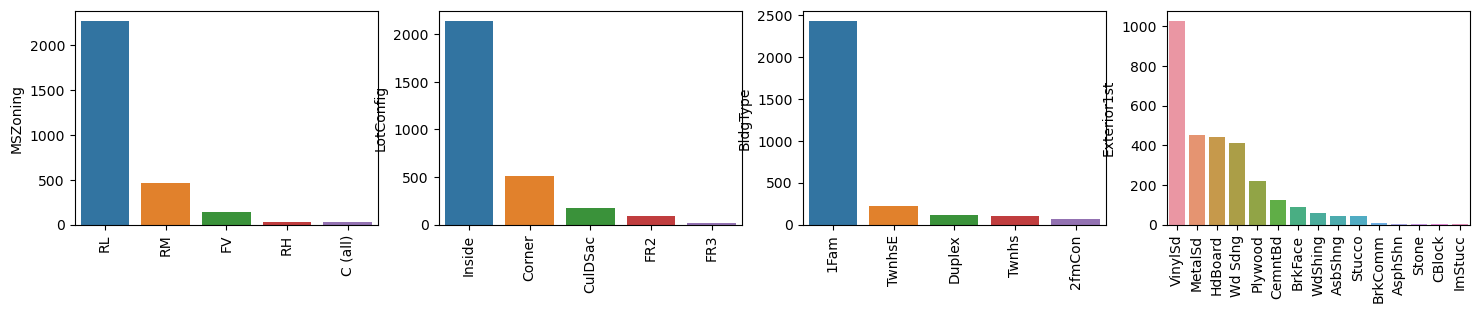
model\_LR.fit(X\_train, Y\_train)

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

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

**OUTPUT:**

****

****

**PROGRAM 15:**

**#Import Necessary Libraries and functions**

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

**#Load the iris dataset**

iris = datasets.load\_iris()

**#GaussianNB and MultinomialNB Models**

gnb = GaussianNB()

mnb = MultinomialNB()

**#Train both GaussianNB and MultinomialNB Models and print their confusion matrices**

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

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

print("Confusion Matrix of GNB \n",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("Confusion Matrix of MNB \n",cnf\_matrix\_mnb)

**OUTPUT:**

Confusion Matrix of GNB

[[50 0 0]

[ 0 47 3]

[ 0 3 47]]

Confusion Matrix of MNB

[[50 0 0]

[ 0 46 4]

[ 0 3 47]]

**PROGRAM 16:**

import numpy

import pandas as pd

import numpy as np

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("D:/GEO/BE COURSES/LAB/DATASET/IRIS.csv")

print(iris.head())

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..10,random\_state=42)

#x = np.array(data[["Age", "EstimatedSalary"]])

#y = np.array(data[["Purchased"]])

#xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.10, random\_state=42)

decisiontree = DecisionTreeClassifier()

logisticregression = LogisticRegression()

knearestclassifier = KNeighborsClassifier()

#svm\_classifier = SVC()

bernoulli\_naiveBayes = BernoulliNB()

passiveAggressive = PassiveAggressiveClassifier()

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

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

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

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

data1 = {"Classification Algorithms": ["KNN Classifier", "Decision Tree Classifier",

"Logistic Regression", "Passive Aggressive Classifier"],

"Score": [knearestclassifier.score(x,y), decisiontree.score(x, y),

logisticregression.score(x, y), passiveAggressive.score(x,y) ]}

score = pd.DataFrame(data1)

score

**OUTPUT:**

|  | **Classification Algorithms** | **Score** |
| --- | --- | --- |
| **0** | KNN Classifier | 0.973333 |
| **1** | Decision Tree Classifier | 1.000000 |
| **2** | Logistic Regression | 0.980000 |
| **3** | Passive Aggressive Classifier | 0.826667 |
|  |  |  |

**PROGRAM 17:**

**#importing necessary libraries**

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

**#importing dataset**

data = pd.read\_csv("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)

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

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

print(price\_range)

**OUTPUT:**

[[ 0 95]

[ 1 90]

[ 2 97]

[ 3 118]]

**PROGRAM 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:**

Accuracy: 0.978

Accuracy: 0.978

**PROGRAM 19:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("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

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

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

print(cm)

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

**OUTPUT:**

[[99 8]

[ 2 62]]

0.9415204678362573

**PROGRAM 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("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"], 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 = [[TV, Radio, Newspaper]]

features = np.array([[230.1, 37.8, 69.2]])

print(model.predict(features))

**OUTPUT:**

Corelations :

Sales 1.000000

TV 0.901208

Radio 0.349631

Newspaper 0.157960

Name: Sales, dtype: float64

Score : 0.9059011844150826

[21.37254028]