PROGRAM1:

```
#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:

```
Initial Hypothesis, H0 :
['0', '0', '0', '0', '0', '0']

Instance  1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] is +ve
Hypothesis  1  :  ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']


Instance  2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] is +ve
Hypothesis  2  :  ['sunny', 'warm', '?', 'strong', 'warm', 'same']


Instance  3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] is -ve

Instance  4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] is +ve
Hypothesis  4  :  ['sunny', 'warm', '?', 'strong', '?', '?']


The maximally specific hypothesis is :
['sunny', 'warm', '?', 'strong', '?', '?']
```

PROGRAM2:

```
###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:

```
Instance 1 +ve
S1 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
G1 []

Instance 2 +ve
S2 ['sunny', 'warm', '?', 'strong', 'warm', 'same']
G2 []

Instance 3 -ve
S3 ['sunny', 'warm', '?', 'strong', 'warm', 'same']
G3
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 +ve
S4 ['sunny', 'warm', '?', 'strong', '?', '?']
G4 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]


Final Version Space :  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['sunny', 'warm', '?', 'strong', '?', '?']]
```

PROGRAM4:

```
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:

```
Normalized Input :
 [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
 [[0.92]
 [0.86]
 [0.89]]
Predicted Output
:  [[0.89787544]
 [0.8758127 ]
 [0.89648915]]
```

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:

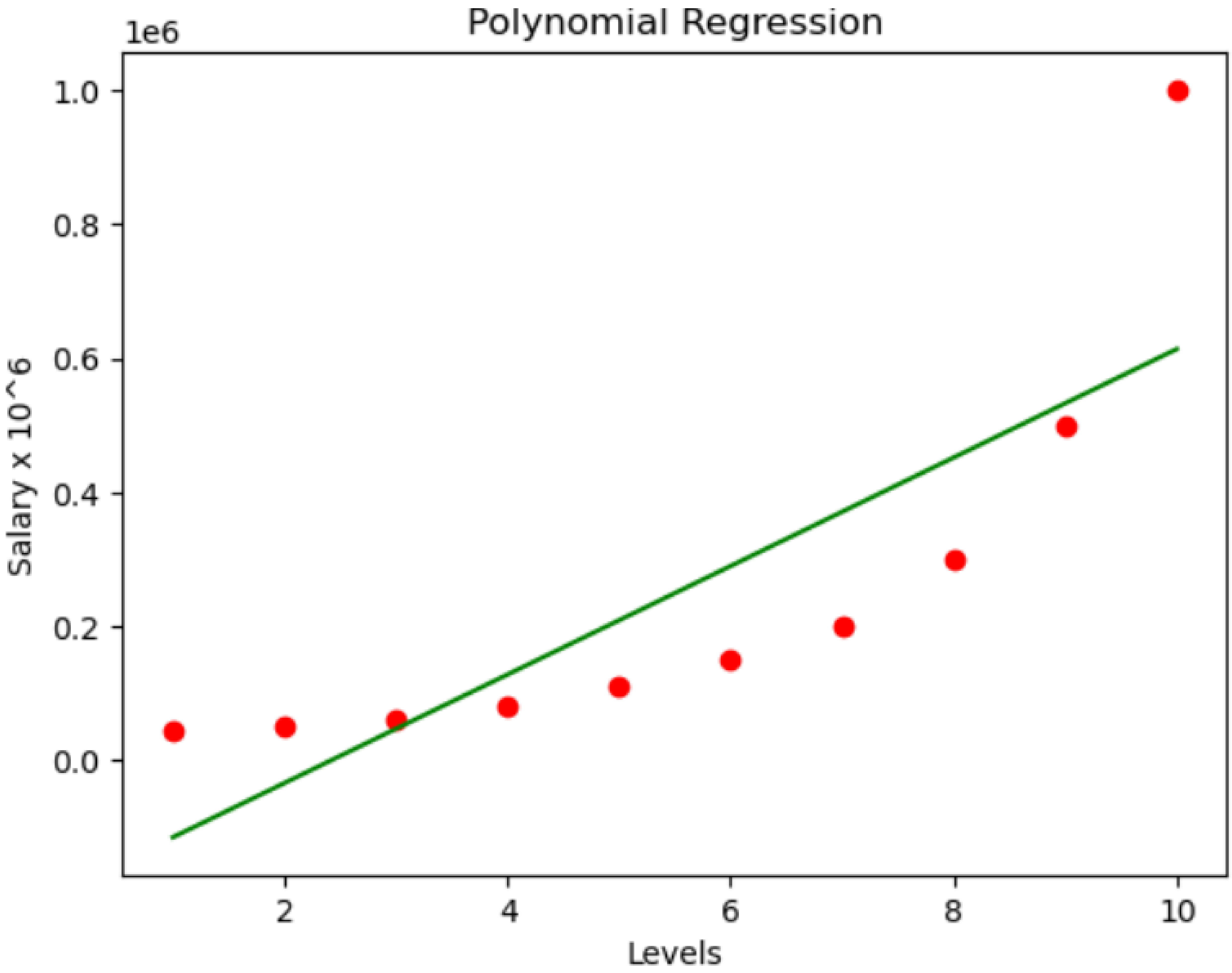
```
['Sunny', 'Cool', 'Normal', 'Strong'] corresponds to
YES
```

PROGRAM7:

```
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:



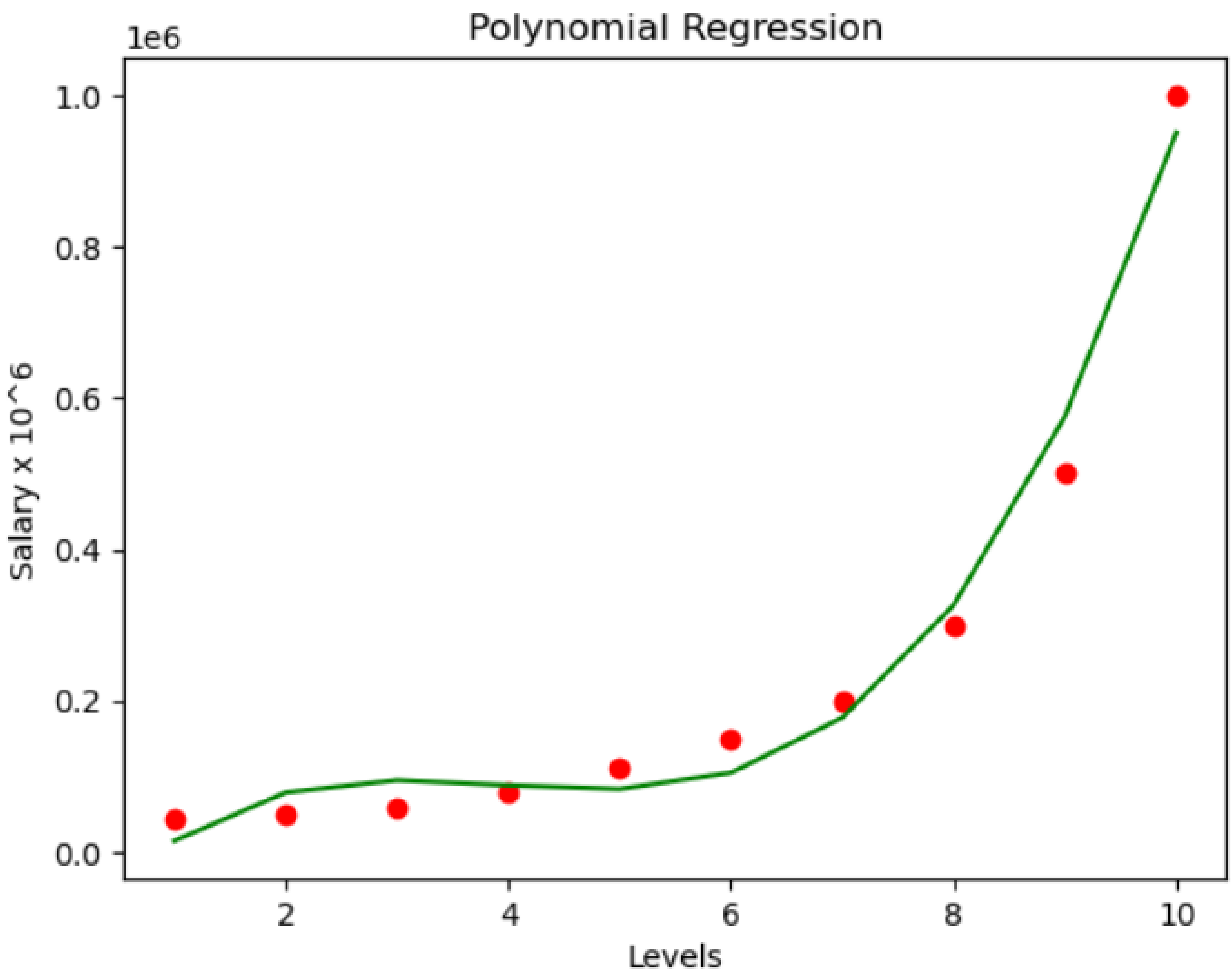
Polynomial Regression
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Salary x 10^6
2
4
6
8
10
Levels

PROGRAM8:

```
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:



PROGRAM9:

```
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
```

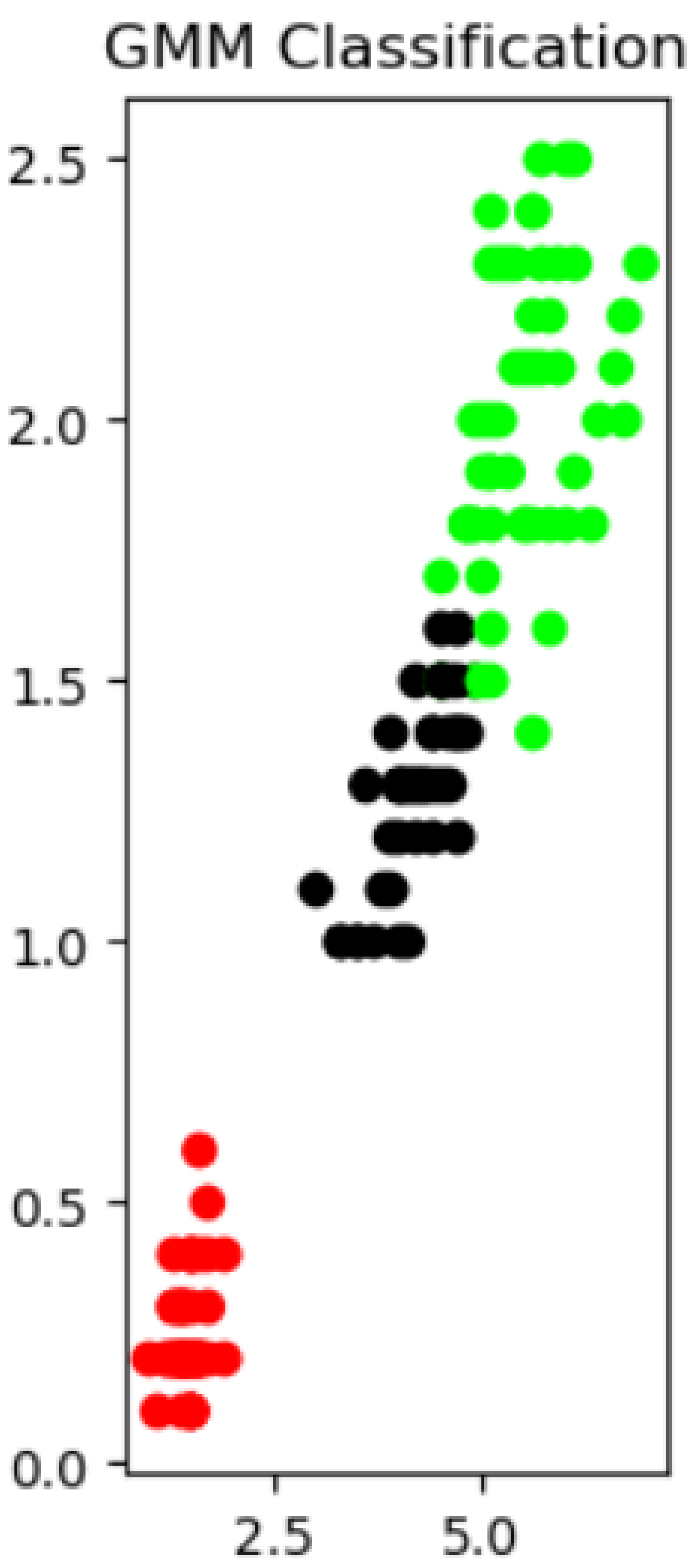
PROGRAM10:

```
fromsklearn.clusterimport KMeans
fromsklearn.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]]
```



PROGRAM11:

```
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:

```
Credit Score Prediction :
Annual Income: 19114.12
Monthly Inhand Salary: 1824.843333
Number of Bank Accounts: 2
Number of Credit cards: 2
Interest rate: 9
Number of Loans: 2
Average number of days delayed by the person: 12
Number of delayed payments: 3
Credit Mix (Bad: 0, Standard: 1, Good: 3) : 3
Outstanding Debt: 250
Credit History Age: 200
Monthly Balance: 310
Predicted Credit Score =  ['Good']
```

PROGRAM12:

#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())

```python
#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']
```

PROGRAM13:

#Importingpackagesandfunctions

```
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
```

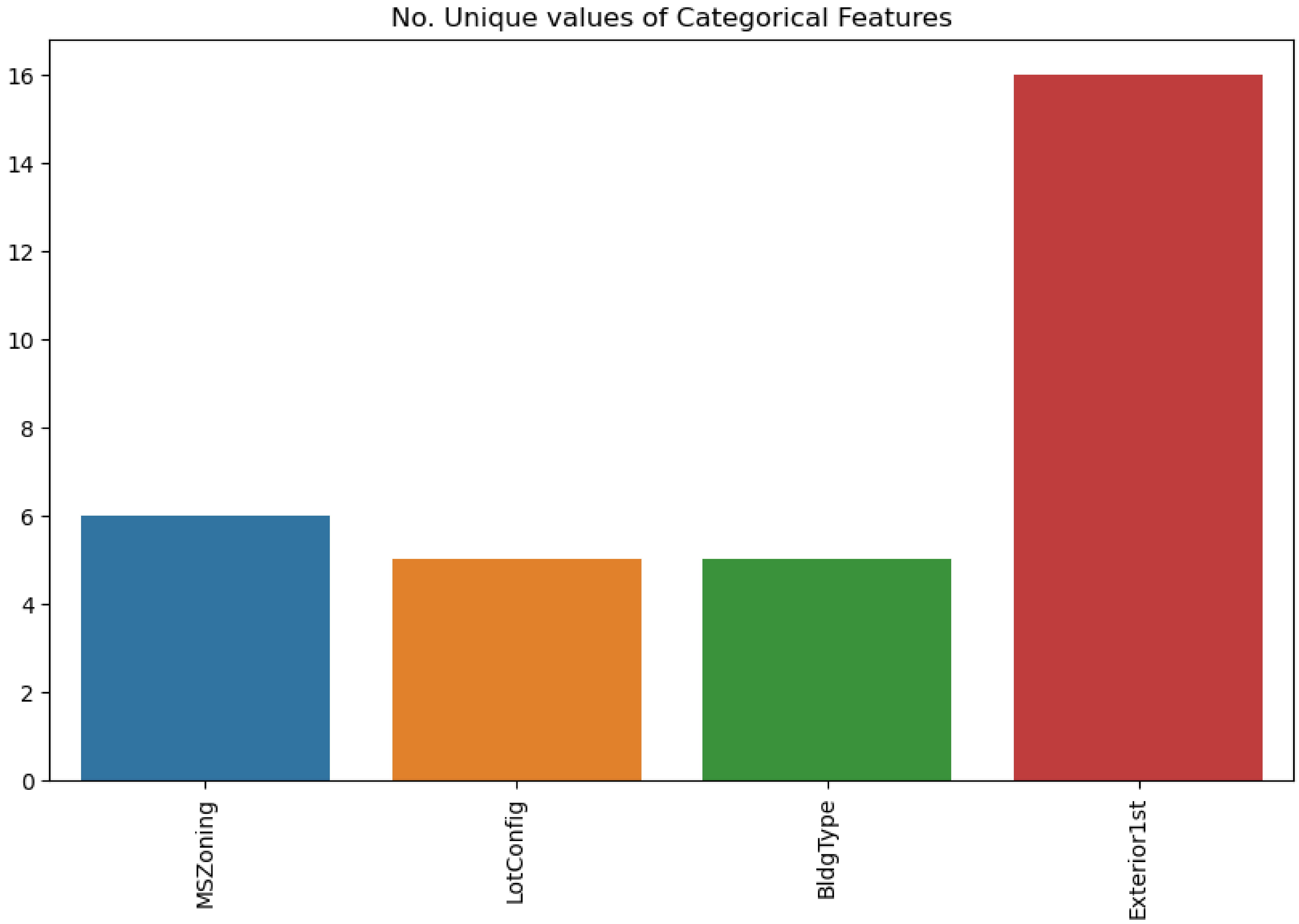
PROGRAM14:

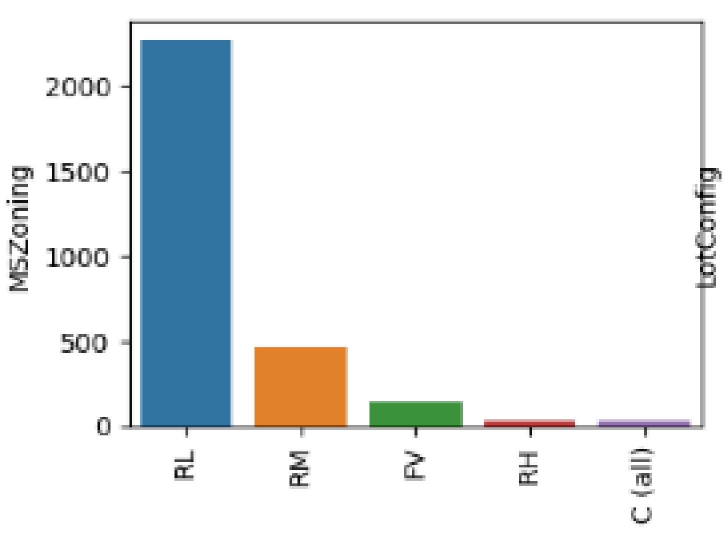
```
#Importing Libraries and functions
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Importing Dataset
dataset = pd.read_csv("HousePricePrediction.csv")
#Exploring dataset
print(dataset.head(5))
dataset.shape
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
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))
```

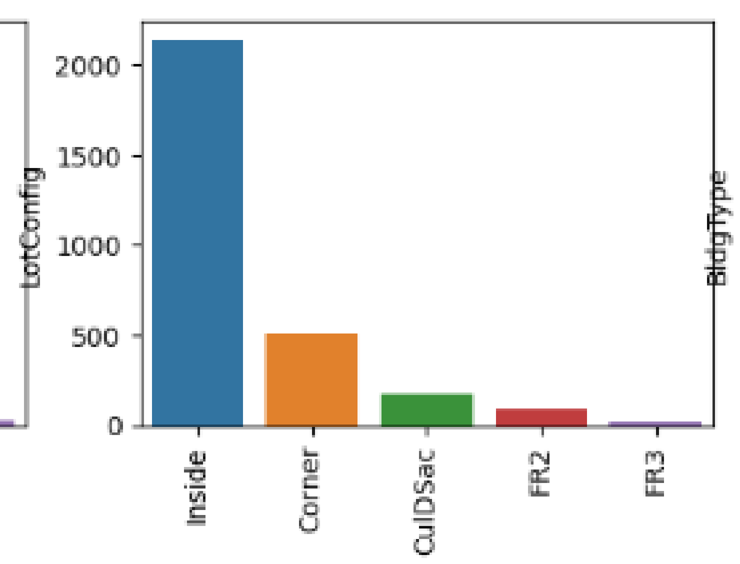
```
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)

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)
```

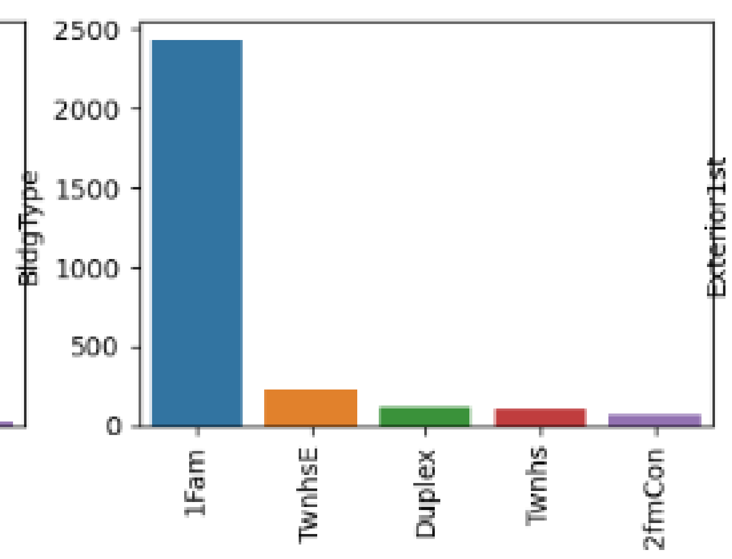
```
df_final = pd.concat([df_final, OH_cols], axis=1)
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

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))

#Linear Regression
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))
```

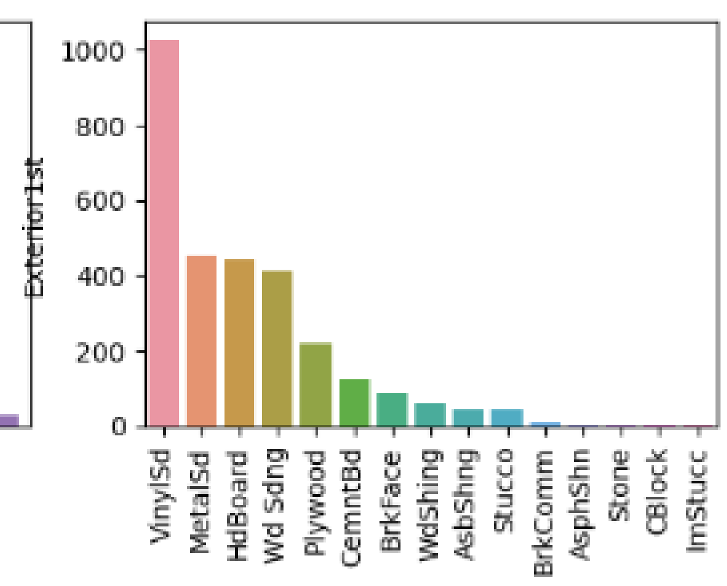
OUTPUT:


No. Unique values of Categorical Features
16
14
12
10
8
6
4
2
0
MSZoning
LotConfig
BldgType
Exterior1st


MSZoning
2000
1500
1000
500
0
RL
RM
FV
RH
C (all)


LotConfig
2000
1500
1000
500
0
Inside
Corner
CulDSac
FR2
FR3


BldgType
2500
2000
1500
1000
500
0
1Fam
TwnhsE
Duplex
Twnhs
2fmCon


Exterior1st
1000
800
600
400
200
0
VinylSd
MetalSd
HdBoard
Wd Sdng
Plywood
CemntBd
BrkFace
WdShing
AsbShng
Stucco
BrkComm
AsphShn
Stone
CBlock
ImStucc

PROGRAM15:

#ImportNecessaryLibrariesandfunctions

```
fromsklearn.naive_bayes importGaussianNB

fromsklearn.naive_bayes importMultinomialNB

fromsklearn import datasets

fromsklearn.metricsimport confusion_matrix
```

#Loadtheirisdataset

```
iris = datasets.load_iris()
```

#GaussianNBandMultinomialNBModels

```
gnb = GaussianNB()

mnb = MultinomialNB()
```

#TrainbothGaussianNBandMultinomialNBModelsandprinttheirconfusionmatrices

```
y_pred_gnb = gnb.fit(iris.data, iris.target).predict(iris.data)

cnf_matrix_gnb = confusion_matrix(iris.target, y_pred_gnb)

print("Confusion MatrixofGNB \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 MatrixofMNB \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]]
```

PROGRAM16:

```
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

PROGRAM17:

#importingnecessarylibraries

```
import numpyas np
import pandasas pd
import matplotlib.pyplot as plt
import seabornas sns
fromsklearn.model_selectionimport train_test_split
fromsklearn.preprocessing import StandardScaler
fromsklearn.linear_model import LogisticRegression
fromsklearn.metricsimport accuracy_score
```

#importingdataset

```
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)
```

#datapreparation

```
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]]
```

PROGRAM18:

```
fromsklearn import datasets
import numpyas np
fromsklearn.model_selection import train_test_split
fromsklearn.linear_model import Perceptron
fromsklearn.preprocessing import StandardScaler
fromsklearn.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
```

PROGRAM20:

```
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]
```