1)

CODE :

```
import csv
a = []
with open('enjoysport.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 :

[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The total number of training instances are : 4

The initial hypothesis is :

['0', '0', '0', '0', '0', '0']

Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

The hypothesis for the training instance 1 is: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] and is Positive Instance

The hypothesis for the training instance 2 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] and is Negative Instance Hence Ignored

The hypothesis for the training instance 3 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] and is Positive Instance

The hypothesis for the training instance 4 is: ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is ['sunny', 'warm', '?', 'strong', '?', '?']

2)

CODE :

```
import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/1 and 2/enjoysport (1).csv"))

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

print(concepts)

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

print(target)

def learn(concepts, target):

    specific_h = concepts[0].copy()

    print("initialization of specific_h and general_h")

    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in

range(len(specific_h))]

    print(general_h)

    for i, h in enumerate(concepts):

        if target[i] == "yes":

            for x in range(len(specific_h)):

                if h[x]!= specific_h[x]:

                    specific_h[x] ='?'

                    general_h[x][x] ='?'

                print(specific_h)

        print(specific_h)

        if target[i] == "no":

            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(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val ==['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

OUTPUT :

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same'
```

3)

CODE :

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/4-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)
```

4)

CODE :

```
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
#Sigmoid Function
def sigmoid (x):
   return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)
#Variable initialization
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
#weight and bias initialization
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))
#draws a random range of numbers uniformly of dim x*y
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)
    #Backpropagation
    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 :

```
-----------Epoch- 1 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
```

```
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.83995147]
 [0.81801839]
 [0.84045961]]
-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84054554]
 [0.818602  ]
 [0.84105669]]
-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
```

```
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84113015]
 [0.81917659]
 [0.84164425]]
-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84170553]
 [0.81974236]
 [0.8422225 ]]
-----------Epoch- 4 Ends----------
```

8)

CODE :

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/Position_Salaries.csv")

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

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

"""

Training the Linear Regression model on the Whole dataset

A Linear regression algorithm is used to create a model.

 A LinearRegression function is imported from sklearn.linear_model library.

 """


from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()

lin_reg.fit(X, y)


#Linear Regression classifier model

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

from sklearn.linear_model import LinearRegression

reg = LinearRegression(normalize_X=True)


"""

Training the Polynomial Regression model on the Whole dataset

A polynomial regression algorithm is used to create a model.

"""

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)

#Polynomial Regression classifier model

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

"""

Visualising the Linear Regression results

Here scatter plot is used to visualize the results. The title of the plot is set to Truth or Bluff (Linear Regression), xlabel is set to Position Level , and ylabel is set to Salary.

"""

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()

#Visualising the Polynomial Regression results

"""

The title of the plot is set to Truth or Bluff (Polynomial Regression), xlabel is set to Position level, and ylabel is set to Salary.

"""

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()

9)

CODE :

```
import numpy as np

import pandas as pd


#"Importing the dataset

# divide the dataset into concepts and targets. Store the concepts into X and targets into y.

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/breastcancer.csv")

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

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


#Splitting the dataset into the Training set and Test

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)

"""

Training the Logistic Regression (LR) Classification model on the Training set

Once the dataset is scaled, next, the Logistic Regression (LR) classifier algorithm is used to create a model.

The hyperparameters such as random_state to 0 respectively.

 The remaining hyperparameters Logistic Regression (LR) are set to default values.

 """

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)

#Display the results (confusion matrix and accuracy)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance
of the model

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 :

```
[[117   8]
 [  6  74]]
```

11)

CODE :

```
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:/Users/Asus/OneDrive/Documents/ML/DATASET/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 Exploration

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 :

```
   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

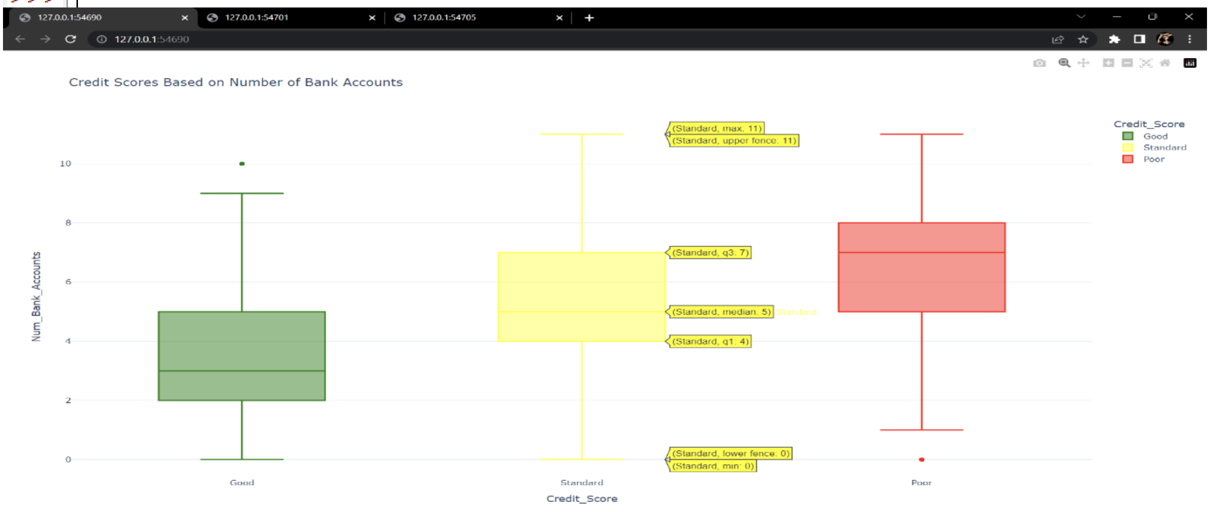
Payment_Behaviour 0

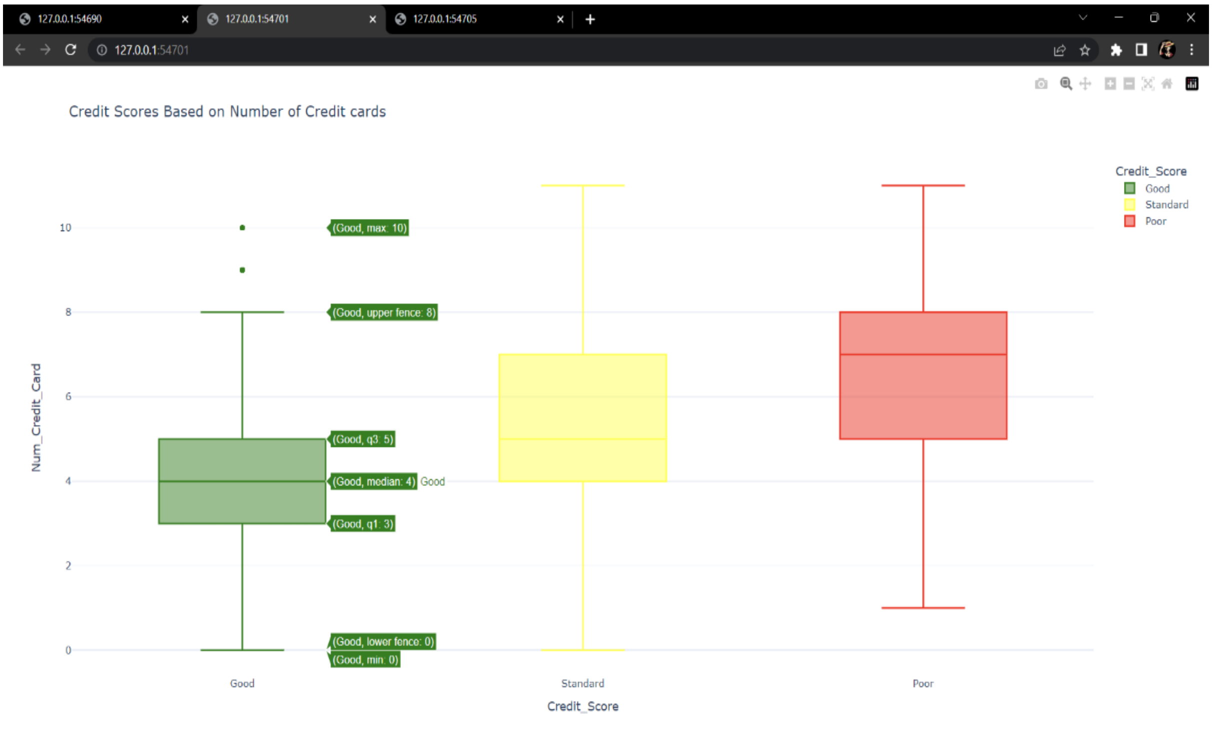
Monthly_Balance 0

Credit_Score 0

dtype: int64

```
Credit Score Prediction :
Annual Income: 450000
Monthly Inhand Salary: 2500
Number of Bank Accounts: 2
Number of Credit cards: 0
Interest rate: 2
Number of Loans: 0
Average number of days delayed by the person: 0
Number of delayed payments: 0
Credit Mix (Bad: 0, Standard: 1, Good: 3) : 3
Outstanding Debt: 1
Credit History Age: 20
Monthly Balance: 1500
Predicted Credit Score =  ['Good']
>>>
```




127.0.0.1:54690
127.0.0.1:54701
127.0.0.1:54705
127.0.0.1:54701
Credit Scores Based on Number of Credit cards
Credit_Score
Good
Standard
Poor
Num_Credit_Card
(Good, max: 10)
(Good, upper fence: 8)
(Good, q3: 5)
(Good, median: 4) Good
(Good, q1: 3)
(Good, lower fence: 0)
(Good, min: 0)
Credit_Score


127.0.0.1:54690
127.0.0.1:54701
127.0.0.1:54705
127.0.0.1:54705
Credit Scores Based on the Average Interest rates
Credit_Score
Good
Standard
Poor
Interest_Rate
(Poor, max: 34)
(Poor, upper fence: 34)
(Poor, q3: 27)
(Poor, median: 21) Poor
(Poor, q1: 15)
(Poor, lower fence: 1)
(Poor, min: 1)
Credit_Score

12)

CODE :

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

iris = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/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())

#plot the data using a scatter plot which will plot the iris species according to the sepal length and sepal width:

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()

#Iris Classification Model

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))
```

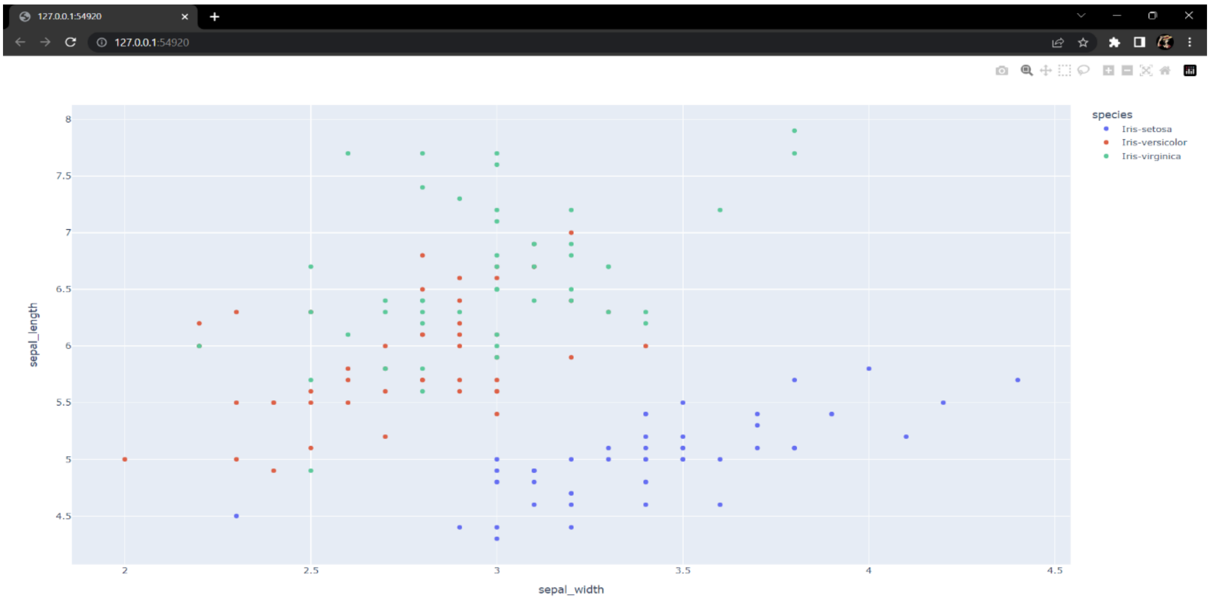
OUTPUT :

ng KNN

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Target Labels ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']13)



13)

CODE :

```
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:/Users/Asus/OneDrive/Documents/ML/DATASET/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="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 :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
```

14)

CODE :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/HousePricePrediction.csv")
# 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
```

16)

CODE :

```
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("C:/Users/Asus/OneDrive/Documents/ML/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 :

lgorithms

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

17)

CODE :

```
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:/Users/Asus/OneDrive/Documents/ML/DATASET/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 :

```
   battery_power  blue  clock_speed  ...  touch_screen  wifi  price_range
0            842     0          2.2  ...             0     1            1
1           1021     1          0.5  ...             1     0            2
2            563     1          0.5  ...             1     0            2
3            615     1          2.5  ...             0     0            2
4           1821     1          1.2  ...             1     0            1

[5 rows x 21 columns]
Accuracy of the Logistic Regression Model:  95.5
[3 0 2 2 3 0 0 3 3 1 1 3 0 2 3 0 3 2 2 1 0 0 3 1 2 2 3 1 3 1 1 0 2 0 2 3 0
 0 3 3 3 1 3 3 1 3 0 1 3 1 1 3 0 3 0 2 2 2 0 3 3 1 3 2 1 2 3 2 2 2 3 2 1 0
 1 3 2 2 1 2 3 3 3 0 0 0 2 1 2 3 1 2 2 1 0 3 3 3 0 3 1 1 3 1 3 2 2 3 2 3 3
 0 0 1 3 3 0 0 1 0 0 3 2 2 1 2 1 1 0 2 1 3 3 3 3 3 3 2 0 1 1 2 1 3 0 3 0 0
 2 0 1 1 1 1 3 0 0 3 1 3 2 1 3 1 2 3 3 2 1 0 3 1 2 3 3 0 2 2 3 1 2 1 0 1 2
 2 2 0 3 3 1 1 0 2 3 0 1 2 2 0 3 3 3 1 2 3 3 3 0 0 0 2 3 3 0 0 1 3 2 3 3 3
 0 0 2 3 3 1 0 2 0 0 0 3 2 1 2 2 1 1 0 2 3 3 0 0 1 3 3 1 3 0 3 1 1 0 2 3 3
 2 0 0 1 2 3 2 2 3 2 1 0 3 3 2 1 3 2 2 2 1 0 2 2 1 0 0 2 2 2 3 0 1 3 0 2 2
 3 0 2 0 1 1 3 0 0 2 3 1 2 0 2 0 3 0 3 3 2 3 1 2 2 1 1 1 0 1 0 3 1 0 3 0 0
 1 3 0 3 1 1 0 1 3 0 2 1 1 2 1 1 0 2 0 0 3 1 2 3 2 2 0 3 2 2 1 3 2 3 3 3 0
 2 0 3 0 1 1 2 3 1 3 1 2 0 1 2 3 0 0 1 3 0 3 0 2 2 1 1 0 2 0]
[[  0  95]
 [  1  90]
 [  2  97]
 [  3 118]]
```

20)

CODE :

```
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:/Users/Asus/OneDrive/Documents/ML/DATASET/DATASET/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 :

```
     TV  Radio  Newspaper  Sales
0  230.1   37.8      69.2   22.1
1   44.5   39.3      45.1   10.4
2   17.2   45.9      69.3   12.0
3  151.5   41.3      58.5   16.5
4  180.8   10.8      58.4   17.9
       TV  Radio  Newspaper  Sales
68   237.4   27.5      11.0   18.9
66    31.5   24.6       2.2   11.0
139  184.9   43.9       1.7   20.7
150  280.7   13.9      37.0   16.1
86    76.3   27.5      16.0   12.0
TV         0
Radio      0
Newspaper    0
Sales      0
dtype: int64
```

18)

CODE :

```
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 :

```
n
Accuracy: 0.978
Accuracy: 0.978
```

19)

CODE :

```
import numpy as np

import pandas as pd

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/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)

#Feature Scaling

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)

#Naive Bayes classifier model

GaussianNB(priors=None, var_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

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]]
```

2)

['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

steps of Candidate Elimination Algorithm 2

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

steps of Candidate Elimination Algorithm 3

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' 'warm' 'same']

['sunny' 'warm' '?' 'strong' '?' 'same']

['sunny' 'warm' '?' 'strong' '?' '?']

['sunny' 'warm' '?' 'strong' '?' '?']

steps of Candidate Elimination Algorithm 4

['sunny' 'warm' '?' 'strong' '?' '?']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

4)

```
-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84227188]
 [0.82029951]
 [0.84279166]]
-----------Epoch- 5 Ends----------

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84227188]
 [0.82029951]
 [0.84279166]]
```

7)

CODE :

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/Position_Salaries.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})

#Visualising the Training set results Here scatter plot is used to visualize the results.

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()
```

10}

CODE :

```
#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("C:/Users/Asus/OneDrive/Documents/ML/DATASET/4-dataset.csv" ,
names=names)

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'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# GMM PLOT

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))
```

13)

7 drivewheel 205 non-null object

8 enginelocation 205 non-null object

9 wheelbase 205 non-null float64

10 carlength 205 non-null float64

11 carwidth 205 non-null float64

12 carheight 205 non-null float64

13 curbweight 205 non-null int64

14 enginetype 205 non-null object

15 cylindernumber 205 non-null object

16 enginesize 205 non-null int64

17 fuelsystem 205 non-null object

18 boreratio 205 non-null float64

19 stroke 205 non-null float64

20 compressionratio 205 non-null float64

21 horsepower 205 non-null int64

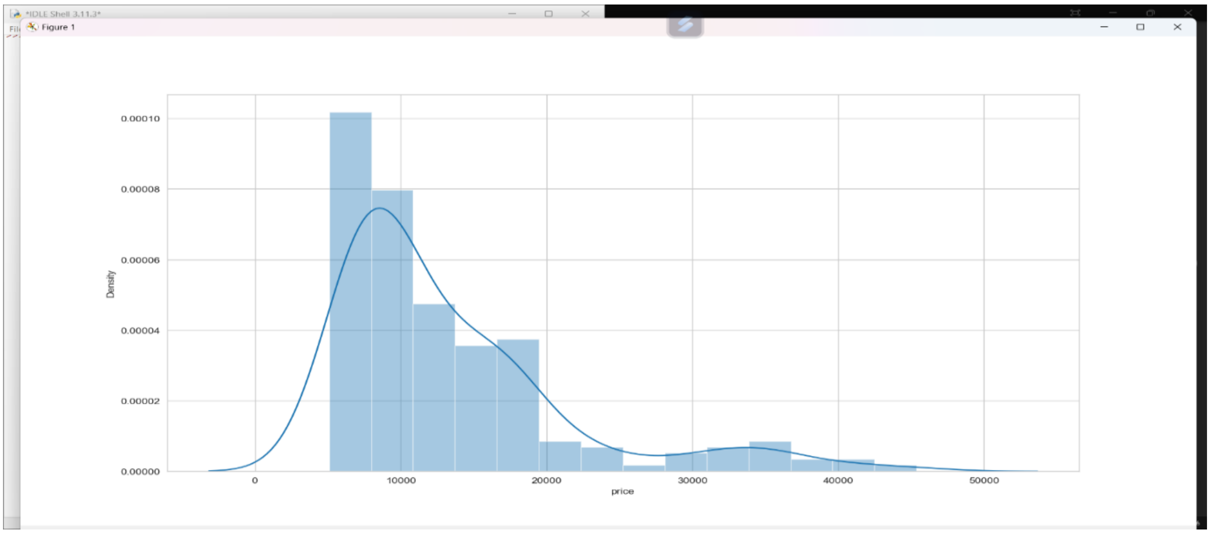
22 peakrpm 205 non-null int64

23 citympg 205 non-null int64

24 highwaympg 205 non-null int64

25 price 205 non-null float64

dtypes: float64(8), int64(8), object(10)



5)

CODE :

```
import numpy as np

import pandas as pd

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/breastcancer.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)

#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)
```

OUTPUT :

```
[[78  1]

 [ 5 53]]
```

6)

CODE :

```
import numpy as np

import pandas as pd

#Importing the dataset

dataset = pd.read_csv("C:/Users/Asus/OneDrive/Documents/ML/DATASET/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)

#Feature Scaling

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)

#Naive Bayes classifier model

GaussianNB(priors=None, var_smoothing=1e-09)

#Display the results (confusion matrix and accuracy)

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]]
```

14)

```
# 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)
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(Y_valid, Y_pred))
```

OUTPUT :

```
Id  MSSubClass MSZoning  ...  BsmtFinSF2 TotalBsmtSF SalePrice
0   0        60      RL  ...        0.0       856.0  208500.0
1   1        20      RL  ...        0.0      1262.0  181500.0
2   2        60      RL  ...        0.0       920.0  223500.0
3   3        70      RL  ...        0.0       756.0  140000.0
4   4        60      RL  ...        0.0      1145.0  250000.0
[5 rows x 13 columns]
Categorical variables: 4          Integer variables: 0          Float variables: 3
```

15)

CODE :

```
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 :

```
[[50  0  0]

 [ 0 47  3]

 [ 0  3 47]]

[[50  0  0]

 [ 0 46  4]

 [ 0  3 47]]
```