INDEX

LIST OF EXPERIMENTS

EX 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Aim: To implement and demonstrate the "find-s" algorithm for finding the most specific hypothesis based on a given set of training data samples.

Algorithm:

1. Load Data set
2. Initialize h to the most specific hypothesis in H
3. For each positive training instance x
   - For each attribute constraint ai in h
     - if the constraint ai is satisfied by x
       Then do nothing.

     else replace ai in h by the next more general constraint that is satisfied by x
4. Output hypothesis h

DataSet:

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	cold	change	yes

Program:-

```
import csv

# Create the empty List to store the values
a = []

# The open() method  is used to open files and return a file object.
# use csv.reader object to read the CSV file
with open('/content/sample_data/tennis1.csv', 'r') as csvfile:
   for row in csv.reader(csvfile):
      a.append(row)
   print(a)

# To Check how many instances are there in the data set
print("\n The total number of training instances are : ",len(a))

# assign the total number of features excluding the target value. Hence, len(h[0])-1
num_attribute = len(a[0])-1

# Initialize  h  to  the  most  specific  hypothesis  in  H
hypothesis = ['0']*num_attribute
print("\n The initial hypothesis is : \n", hypothesis)

# For  each  positive  training  instance  x
#  For  each  attribute  constraint  ai  in  h
#   If  the  constraint  ai  is  satisfied  by  x
#           Then  do  nothing
#   Else  replace  ai  in  h  by  the  next  more  general  constraint  that  is  satisfied  by  x
```

```
for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        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("\n The hypothesis for the training instance {} is :\n”. format(i+1), hypothesis)
```

```
print("\n The Maximally specific hypothesis for the training instances is: \n",
hypothesis )
```

Output:

The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainny', 'cold', 'high', 'strong', 'warm', 'change', 'mo']
['sunny', 'warm', 'high', 'strong', 'cold', 'change', 'yes']
length of a = 4

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

For Training instance No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
For Training instance No:1 the hypothesis is
['sunny', 'warm', '?', 'strong', 'warm', 'same']
For Training instance No:2 the hypothesis is
['sunny', 'warm', '?', 'strong', 'warm', 'same']
For Training instance No:3 the hypothesis is
['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples :

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

Result:

Thus the "find-s" algorithm for finding the most specific hypothesis based on a given set of training data samples is implemented and demonstrated.

- **illustrate this algorithm, assume the learner is given the sequence of training examples from the EnjoySport task**

• The first step of FIND-S is to initialize h to the most specific hypothesis in H

S.No	Sky	AirTemp	Humidity	Wind	Water	Forecast	En Sp
1	Sunny	Warm	Normal	Strong	Warm	Same	Y
2	Sunny	Warm	High	Strong	Warm	Same	Y
3	Rainy	Cold	High	Strong	Warm	Change	N
4	Sunny	Warm	High	Strong	Cool	Change	Y

h - (Ø, Ø, Ø, Ø, Ø, Ø)

Consider the first training example

x1 = [Sunny Warm Normal Strong Warm Same], +

Observing the first training example, it is clear that hypothesis h is too specific.

None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example

h1 = [Sunny Warm Normal Strong Warm Same]

Consider the second training example

x2 = [Sunny, Warm, High, Strong, Warm, Same], +

The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

h2 = [Sunny Warm ? Strong Warm Same]

Consider the third training example

x3 = [Rainy, Cold, High, Strong, Warm, Change], -

Upon encountering the third training the algorithm makes no change to h. Because the FIND-S algorithm simply ignores every negative example.

h3 = [Sunny Warm ? Strong Warm Same]

Consider the fourth training example

x4 = [Sunny Warm High Strong Cool Change], +

The fourth example leads to a further generalization of h

h4 = [Sunny Warm ? Strong ? ?]

EX 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Aim: To implement and demonstrate Candidate elimination algorithm to output a description of the set of all hypotheses consistent

Candidate-Elimination Algorithm:

1. Load data set
2. G <-maximally general hypotheses in H
3. S <- maximally specific hypotheses in H
4. For each training example d=<x,c(x)>

Case 1 : If d is a positive example

Remove from G any hypothesis that is inconsistent with d

For each hypothesis s in S that is not consistent with d

- Remove s from S.
- Add to S all minimal generalizations h of s such that

o h consistent with d

o Some member of G is more general than h

• Remove from S any hypothesis that is more general than another hypothesis in S

Case 2: If d is a negative example

Remove from S any hypothesis that is inconsistent with d

For each hypothesis g in G that is not consistent with d

- Remove g from G.
- Add to G all minimal specializations h of g such that

o h consistent with d

o Some member of S is more specific than h

• Remove from G any hypothesis that is less general than another hypothesis in G

DataSet:

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	cold	change	yes

Program:-

```
import csv
with open("/content/sample_data/tennis1.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)
    s=data[1][:-1]
    g=[['?' for i in range(len(s))] for j in range(len(s))]

for i in data:
        if i[-1]=="yes":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'
        elif i[-1]=="no":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
        print(s)
        print(g)
gh=[]
    for i in g:
        for j in i:
            if j!='?':
```

```
            gh.append(i)
            break
    print("\nFinal specific hypothesis:\n",s)
    print("\nFinal general hypothesis:\n",gh)
```

Output:

Steps of Candidate Elimination Algorithm 1
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Steps of Candidate Elimination Algorithm 4
['sunny', 'warm', '?', 'strong', '?', '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final specific hypothesis:
['sunny', 'warm', '?', 'strong', '?', '?']

Final general hypothesis:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Result:
Thus the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples is implemented and demonstrated.

CANDIDATE-ELIMINATION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

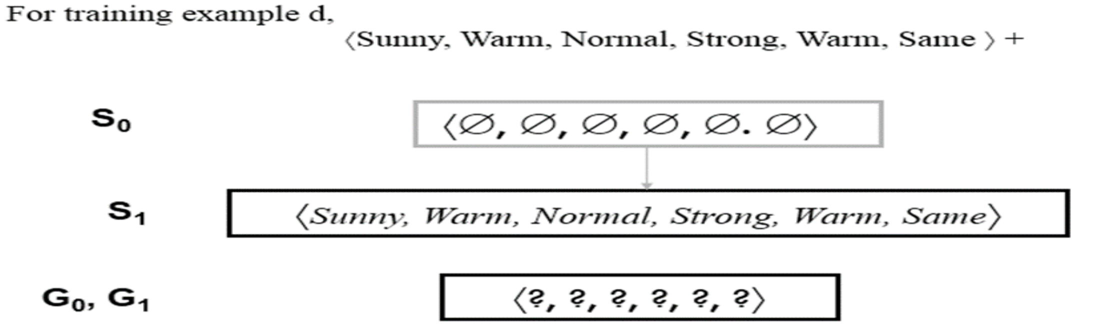
$$G_0 \langle ?,\ ?,\ ?,\ ?,\ ?,\ ? \rangle$$

Initializing the S boundary set to contain the most specific (least general) hypothesis

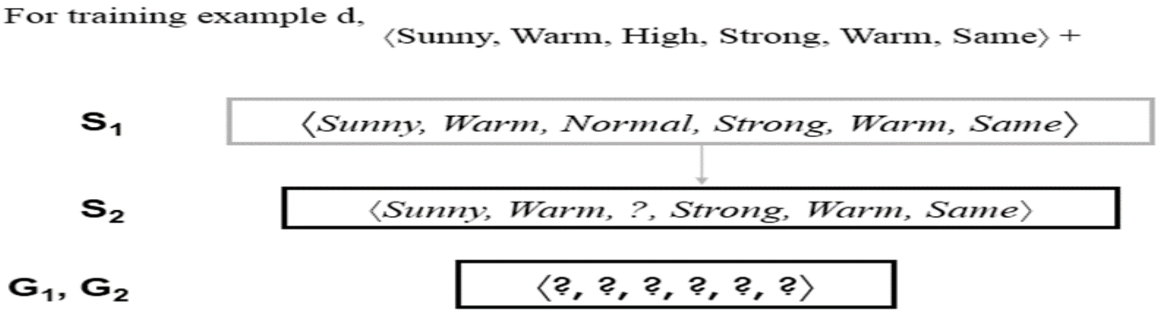
$$S_0 \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$$

- When the first training example is presented, the **CANDIDATE-ELIMINATION** algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.

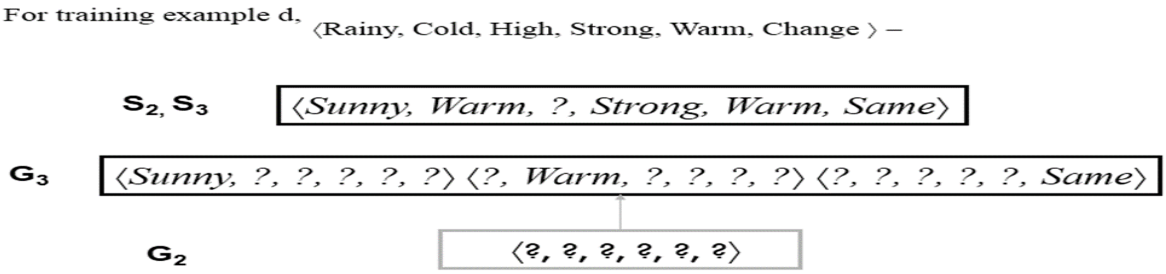
● The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
● No update of the G boundary is needed in response to this training example because Go correctly covers this example



● **When the second training example is observed**, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2=G1=G0



Consider the third training example. This negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
● The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example



Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?
For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

Consider the fourth training example:

For training example d,

⟨Sunny, Warm, High, Strong, Cool Change ⟩ +

S_3 ⟨*Sunny, Warm, ?, Strong, Warm, Same*⟩

S_4 ⟨*Sunny, Warm, ?, Strong, ?, ?*⟩

G_4 ⟨*Sunny, ?, ?, ?, ?, ?*⟩ ⟨*?, Warm, ?, ?, ?, ?*⟩

G_3 ⟨*Sunny, ?, ?, ?, ?, ?*⟩ ⟨*?, Warm, ?, ?, ?, ?*⟩ ⟨*?, ?, ?, ?, ?, Same*⟩

This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example.

After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.

S_4 ⟨*Sunny, Warm, ?, Strong, ?, ?*⟩

⟨*Sunny, ?, ?, strong , ?, ?*⟩ ⟨*Sunny, Warm, ?, ?, ?, ?*⟩ ⟨*?, Warm, ?, Strong, ?, ?*⟩

G_4 ⟨*Sunny, ?, ?, ?, ?, ?*⟩ ⟨*?, Warm, ?, ?, ?, ?*⟩

EX 3. Write a program to demonstrate Association rule process on dateset using Apriori algorithm

Aim: To demonstrate association rule process on dateset using Apriori algorithm.

Algorithm:
Step 1: Determine the level of transactional database support and establish the minimal degree of assistance and dependability.
Step 2: Take all of the transaction's supports that are greater than the standard or chosen support value.
Step 3: Look for all rules with greater precision than the cutoff or baseline standard, in these subgroups.
Step 4: It is best to arrange the rules in ascending order of strength.

Formulas:

1.

$$Support(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

2.

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

3.

$$Lift(\{X\} \rightarrow \{Y\}) = \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

Dataset:- Transaction List

Step 1: **min_support =0.3**
so support_count = min_support * No of Transactions = 0.3*12 =3.6 = 4

Milk,Egg,Bread,Butter
Milk,Butter,Egg,Ketchup
Bread,Butter,Ketchup
Milk,Bread,Butter
Bread,Butter,Cookies
Milk,Bread,Butter,Cookies
Milk,Cookies
Milk,Bread,Butter
Bread,Butter,Egg,Cookies
Milk,Butter,Bread
Milk,Bread,Butter
Milk,Bread,Cookies,Ketchup

Support count =4

1-item Sets	Frequency
Milk	9
Bread	10
Butter	10
Egg	3
Ketchup	3
Cookies	5

Frequent 1- item Sets	Frequency
Milk	9
Bread	10
Butter	10
Cookies	5

Step 2:

2-item Sets	Frequency
Milk, Bread	7
Milk, Butter	7
Milk, Cookies	3
Bread, Butter	9
Bread, Cookies	3
Butter, Cookies	4

Frequent 2-item Sets	Frequency
Milk, Bread	7
Milk, Butter	7
Bread, Butter	9
Butter, Cookies	4

Step 3:

Frequent 3-item Sets	Frequency
Milk,Bread, Butter	6

3-item Sets	Frequency
Milk,Bread, Butter	6
Milk,Bread, Cookies	1
Milk, Butter, Cookies	3
Bread, Butter,Cookies	2

This is called non-empty subset:
{{Milk},{Bread},{Butter},{Milk,Bread},{Milk.Butter},{Bread,Butter}}

Minimum_support = 30% and Minimum_confidence = 60%

Rule	Support	Confidence	Lift
Milk ==> Butter	9 / 12 = 0.75	7 / 9 = 0.78	(7/9) / (10 /12) = 0.94
Milk ==> Bread	9 / 12 = 0.75	7 / 9= 0.78	(7/9) / (10 /12) = 0.94
Bread ==> Butter	10 / 12 = 0.83	7 / 10 = 0.70	(9/10) / (10/12) = 1.08
('Butter','Milk') ==>('Bread')	6 / 12 = 0.50	6 / 7 = 0.86	(6/7) / (10/12) = 1.03
('Bread', 'Milk') ==> ('Butter')	6 / 12 = 0.50	6 / 7 = 0.86	(6/7) / (10/12) = 1.03
('Bread', 'Butter') ==> ('Milk')	6 / 12 = 0.50	6 / 9 = /0.67	(6/9) / (9/12) = 0.89
('Milk') ==> ('Bread', 'Butter')	6 / 12 = 0.50	6 / 9 = 0.67	(6/9) / (9/12) = 0.89
('Butter') ==>('Bread', 'Milk')	6 / 12 = 0.50	6 / 10 = 0.60	(6/10) / (7/12) = 1.03
('Bread') ==> ('Butter', 'Milk')	6 / 12 = 0.50	6 / 10 = 0.60	(6/10) / 7/12) = 1.03

Program:-

```
from efficient_apriori import apriori
import pandas as pd
# Load the dataset
store=pd.read_csv('/content/sample_data/Day3.csv',names=['product'],header=None)
print(store,"\n")
print("############################################################\n")
# Split the dataset into list
#transactions=list(store['product'].apply(lambda x: x.split(",")))
transactions = [x.split(",") for x in store['product']]
# Print each item and support
itemsets,rules = apriori(transactions, min_support=0.3, min_confidence=0.6)
for i in itemsets:
   split_dicts = [{item: support} for item, support in itemsets[i].items()]
   for d in split_dicts:
     itemset_str = ', '.join(list(d.keys())[0])
     support = list(d.values())[0]
     print("{:<20} {:<15}".format(itemset_str, support))
print("\n############################################################\n")


print( "{:<20} {:<25} {:<15} {:<15} {:<15}".format("Antecedent (lhs)", "Consequent (rhs)",
"Support", "Confidence", "Lift"))
# Print each rule
for rule in rules:
```

```
if rule.support >=0.3 :
   print("{:<20} ==> {:<20} {:<15.4f} {:<15.4f} {:<10.4f}".format(str(rule.lhs), str(rule.rhs),
rule.support, rule.confidence, rule.lift))
```

Output:

```
              product
0       Milk,Egg,Bread,Butter
1       Milk,Butter,Egg,Ketchup
2       Bread,Butter,Ketchup
3       Milk,Bread,Butter
4       Bread,Butter,Cookies
5       Milk,Bread,Butter,Cookies
6       Milk,Cookies
7       Milk,Bread,Butter
8       Bread,Butter,Egg,Cookies
9       Milk,Butter,Bread
10      Milk,Bread,Butter
11      Milk,Bread,Cookies,Ketchup

#################################################################
Milk                    9
Bread                  10
Butter                 10
Cookies                 5
Bread, Butter           9
Bread, Cookies          4
Bread, Milk             7
Butter, Milk            7
Bread, Butter, Milk     6

#################################################################
```

Antecedent (lhs)		Consequent (rhs)	Support	Confidence	Lift
('Butter',)	==>	('Bread',)	0.7500	0.9000	1.0800
('Bread',)	==>	('Butter',)	0.7500	0.9000	1.0800
('Cookies',)	==>	('Bread',)	0.3333	0.8000	0.9600
('Milk',)	==>	('Bread',)	0.5833	0.7778	0.9333
('Bread',)	==>	('Milk',)	0.5833	0.7000	0.9333
('Milk',)	==>	('Butter',)	0.5833	0.7778	0.9333
('Butter',)	==>	('Milk',)	0.5833	0.7000	0.9333
('Butter', 'Milk')	==>	('Bread',)	0.5000	0.8571	1.0286
('Bread', 'Milk')	==>	('Butter',)	0.5000	0.8571	1.0286
('Bread', 'Butter')	==>	('Milk',)	0.5000	0.6667	0.8889
('Milk',)	==>	('Bread', 'Butter')	0.5000	0.6667	0.8889
('Butter',)	==>	('Bread', 'Milk')	0.5000	0.6000	1.0286
('Bread',)	==>	('Butter', 'Milk')	0.5000	0.6000	1.0286

Result:

Thus the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples is implemented and demonstrated.

EX 4. Write a program to Implement any two regression (Linear, Logistic, multiple linear).

Aim: To implement linear and logistic regression

Algorithm:
A) Linear Regression:

1. Import the packages and classes needed.
2. Load the dataset and provide data to work with .
3. Create a Linear regression model and fit it with existing data.
4. Apply the model for predictions.
5. Evaluate the model.
6. Visualize the results

Program:

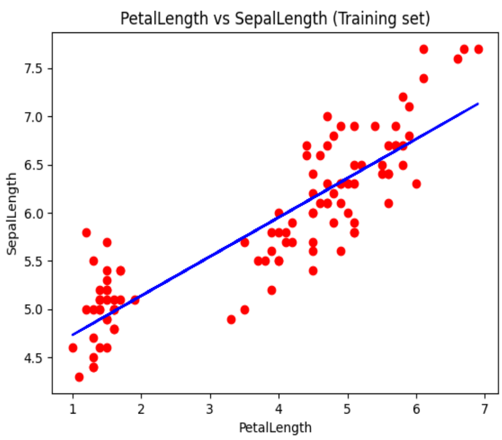
```
# import the required packages and classes
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the iris dataset
iris = pd.read_csv('/content/sample_data/Iris.csv')
print("First five Records:\n\n",iris.head())
# Select and store the X independent attribute and Y dependent attribute
y =iris[['SepalLengthCm']]
x =iris[['PetalLengthCm']]

# Split the data into training and testing sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

# Train the Linear Regression model
lr= LinearRegression()
lr.fit(x_train,y_train)

# Evaluate the model
y_pred = lr.predict(x_test)
y_test.head(), y_pred[0:5]

# Print the result of the Evaluated model
print("Slope: ",lr.coef_)
print("Intercept: ",lr.intercept_)
print("Mean Square Error: ", mean_squared_error(y_test,y_pred))
r2 = r2_score(y_test, y_pred)
print('R2 score: ', r2)

#Visualising the Results
import matplotlib.pyplot as plt
#Visualising the Training set results
plt.scatter(x_train, y_train, color = 'red')
plt.plot(x_train, lr.predict(x_train), color = 'blue')
plt.title('PetalLength vs SepalLength (Training set)')
plt.xlabel('PetalLength')
plt.ylabel('SepalLength')
plt.show()
#Visualising the Test set results
plt.scatter(x_test, y_test, color = 'red')
plt.plot(x_train, lr.predict(x_train), color = 'blue')
plt.title('PetalLength vs SepalLength (Test set)')
plt.xlabel('PetalLength')
plt.ylabel('SepalLength')
plt.show()
```

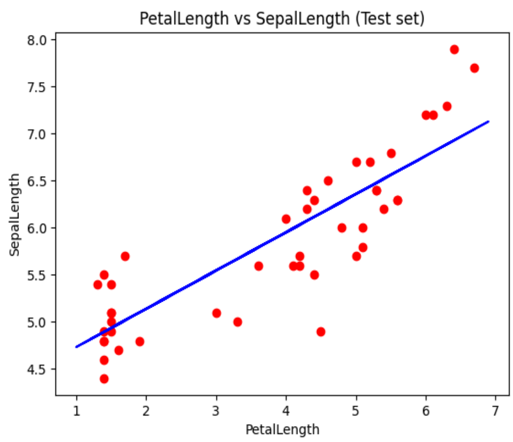
Output:

First five Records:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Slope: [[0.40635235]]
Intercept: [4.32530038]
Mean Square Error: 0.19394089514863622
R2 score: 0.735759347169413





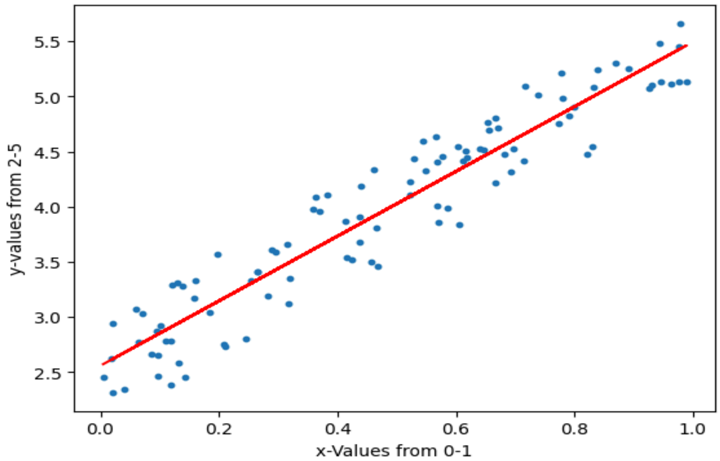
Simple Program:

```
# Importing Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# generate random data-set
np.random.seed(0)
x = np.random.rand(100, 1)

#Generate a 2-D array with 100 rows, each row containing 1 random numbers:
y = 2 + 3 * x + np.random.rand(100, 1)
regression_model = LinearRegression()

# Model initialization
regression_model.fit(x, y)
# Fit the data(train the model)
y_predicted = regression_model.predict(x) # Predict
# model evaluation
rmse = mean_squared_error(y, y_predicted)
r2 = r2_score(y, y_predicted)
# printing values
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
# plotting values # data points
plt.scatter(x, y, s=10)
```

```
plt.xlabel('x-Values from 0-1')
plt.ylabel('y-values from 2-5')
# predicted values
plt.plot(x, y_predicted, color='r')
plt.show()
```

Output:

Slope: [[2.93655106]]
Intercept: [2.55808002]
Root mean squared error: 0.07623324582875007
R2 score: 0.9038655568672764



Algorithm:
B) Logistic Regression:

1. Import the packages and classes needed.
2. Load the dataset and provide data to work with .
3. Create a Logistic regression model and fit it with existing data.
4. Apply the model for predictions.
5. Evaluate the model.
6. Visualize the results

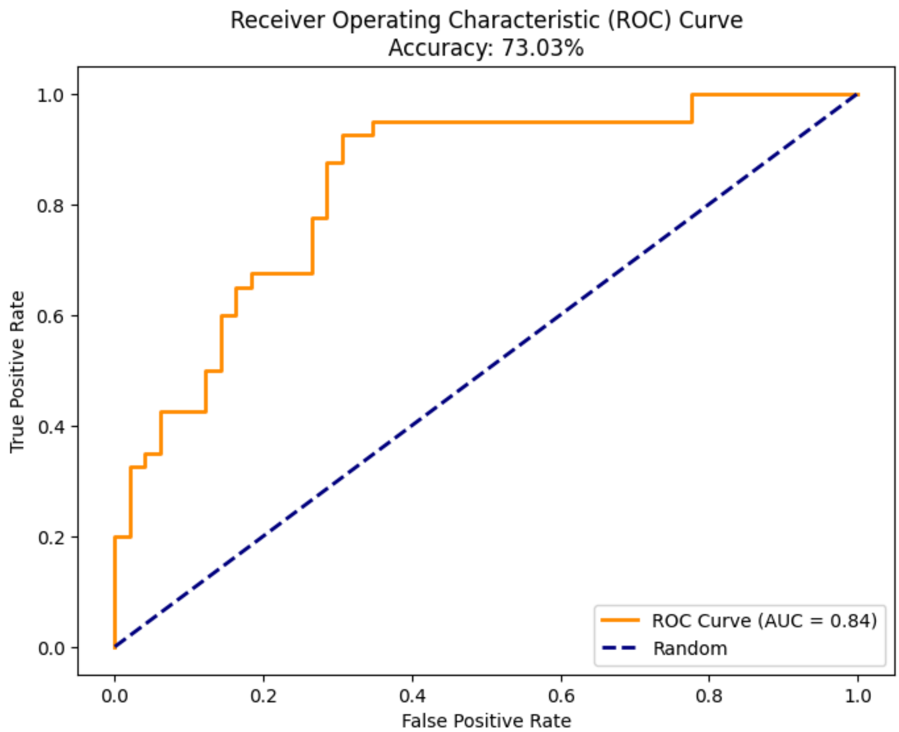
Program:

```
# import the required packages and classes
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
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, classification_report, confusion_matrix,
roc_curve, auc

 Load the diabetes dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# Convert the target variable to binary (1 for diabetes, 0 for no diabetes)
y_binary = (y > np.median(y)).astype(int)

# Split the data into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(
X, y_binary, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot ROC Curve
y_prob = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve\nAccuracy:
{:.2f}%'.format(accuracy * 100))
plt.legend(loc="lower right")
plt.show()
```

Output:

```
Accuracy: 73.03%
Confusion Matrix:
 [[36 13]
 [11 29]]

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        49
           1       0.69      0.72      0.71        40

    accuracy                           0.73        89
   macro avg       0.73      0.73      0.73        89
weighted avg       0.73      0.73      0.73        89
```



Simple program:

```
import numpy
from sklearn import linear_model

#Reshaped for Logistic function.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69,
5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Train the Logistic Regression model
logr = linear_model.LogisticRegression()
logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
print(predicted)

train_acc = logr.score(X, y)
print("The Accuracy for Training Set is {}".format(train_acc*100))
```

Output:

[0]
The Accuracy for Training Set is 91.66666666666666

Result:

Thus the linear regression and logistic regression is implemented and demonstrated.

EX 5. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

Aim: To demonstrate the working of the decision tree based ID3 algorithm
.

Algorithm:

Step 1: Data Preprocessing: Clean and preprocess the data. Handle missing values and convert categorical variables into numerical representations if needed.
Step 2: Selecting the Root Node: Calculate the entropy of the target variable (class labels) based on the dataset. The formula for entropy is:
Entropy(S) = -Σ (p_i * log2(p_i)) where **p_i** is the proportion of instances belonging to class i.
Step 3: Calculating Information Gain:
For each attribute in the dataset, calculate the information gain when the dataset is split on that attribute. The formula for information gain is:

Information Gain(S, A) = Entropy(S) - Σ ((|S_v| / |S|) * Entropy(S_v))
where **S_v** is the subset of instances for each possible value of attribute **A**, and **|S_v|** is the number of instances in that subset.
Step 4: Selecting the Best Attribute: Choose the attribute with the highest information gain as the decision node for the tree.
Step 5: Splitting the Dataset: Split the dataset based on the values of the selected attribute.
Step 6: Repeat the Process: Recursively repeat steps 2 to 5 for each subset until a stopping criterion is met (e.g., the tree depth reaches a maximum limit or all instances in a subset belong to the same class).

Dataset: id3.csv

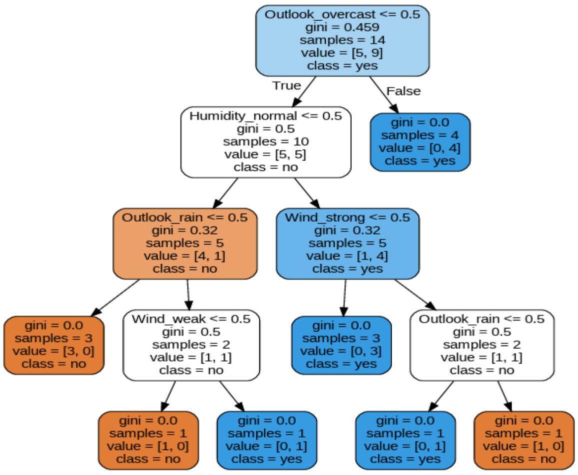
Outlook	Temperature	Humidity	Wind	Play
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Program:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.preprocessing import OneHotEncoder
from IPython.display import display

# Load the dataset
df = pd.read_csv("/content/sample_data/id3.csv")
# Perform one-hot encoding for categorical variables
df_encoded = pd.get_dummies(df[['Outlook', 'Temperature', 'Humidity', 'Wind']])
# Extract features and target variable
X = df_encoded
y = df['Play']
# Create decision tree classifier
clf = DecisionTreeClassifier()
# Train the classifier
clf.fit(X, y)

# Export the decision tree to a Graphviz DOT file
export_graphviz(clf, out_file="decision_tree.dot", feature_names=X.columns,
class_names=clf.classes_, filled=True, rounded=True)
# Render the decision tree using Graphviz
with open("decision_tree.dot") as f:
    dot_graph = f.read()
graph = graphviz.Source(dot_graph)
graph.render("/content/decision_tree")
# Display the decision tree
```

```
display(graph)

# Example usage to predict using a sample
sample_data = {
    'Outlook': ['sunny'],
    'Temperature': ['hot'],
    'Humidity': ['high'],
    'Wind': ['weak']
}
sample_df = pd.DataFrame(sample_data)
# Perform one-hot encoding for sample DataFrame
sample_encoded = pd.get_dummies(sample_df.reindex(columns=X.columns, fill_value=0))
# Predict using the sample DataFrame
prediction = clf.predict(sample_encoded)
print("Prediction for sample: ", prediction)
```

Output:



Prediction for sample: ['yes']

```
import math
import pandas as pd
from operator import itemgetter
class DecisionTree:
    def __init__(self, df, target, positive, parent_val, parent):
        self.data = df
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = ' '
    def _get_entropy(self, data):
        p = sum(data[self.target]==self.positive)
        n = data.shape[0] - p
        p_ratio = p/(p+n)
        n_ratio = 1 - p_ratio
        entropy_p = -p_ratio*math.log2(p_ratio) if p_ratio != 0 else 0
        entropy_n = - n_ratio*math.log2(n_ratio) if n_ratio !=0 else 0
        return entropy_p + entropy_n
```

```
    def _get_gain(self, feat):
        avg_info=0
        for val in self.data[feat].unique():
            avg_info+=self._get_entropy(self.data[self.data[feat] == val])*
sum(self.data[feat]==val) /self.data.shape[0]
        return self._get_entropy(df) - avg_info

    def _get_splitter(self):
        self.splitter = max(self.gains, key = itemgetter(1))[0]

    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features]
            self._get_splitter()
            residual_columns = [k for k in self.data.columns if k != self.splitter]
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter]==val][residual_columns]
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val, self.splitter)
                tmp_node.update_nodes()
                self.childs.append(tmp_node)
        else:
            positive_count = sum(self.data[self.target] == self.positive)
            total_count = self.data.shape[0]
            self.decision = 'yes' if positive_count > total_count / 2 else 'no'

def print_tree(node, indent=""):
    if not node:
        return
    if node.decision:
        print(indent + f"{node.parent}: {node.parent_val} ==> {node.decision}")
    else:
        print(indent + f"{node.parent}: {node.parent_val} ==> ")
    for child in node.childs:
        print_tree(child, indent + "  ")

df = pd.read_csv("/content/sample_data/id3.csv")
print(df)
dt = DecisionTree(df, 'Play', 'yes', '', '')
dt.update_nodes()
print_tree(dt)
```

Output:

```
:  ==>
  Outlook: sunny ==>
    Humidity: high ==> no
    Humidity: normal ==> yes
  Outlook: overcast ==> yes
  Outlook: rain ==>
    Wind: weak ==> yes
    Wind: strong ==> no
```

Result:

Thus a program to demonstrate the working of decision tree based ID3 algorithm was written and implemented

EX 6: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Aim::

To develop a program that implements the naïve Bayesian classifier using a sample training dataset in .CSV format, and to evaluate the accuracy of the classifier using several test datasets.

Algorithm:

STEP 1: Load the training data set from the CSV file into a list of dictionaries, where each dictionary represents a single instance (row) in the data set and the keys represent the attribute names (columns) and the values represent the corresponding attribute values for that instance.

STEP 2: Determine the class variable for each instance in the training data set and add it as a new key-value pair to the corresponding dictionary.

STEP 3: Create a dictionary to store the prior probabilities for each class variable in the training data set. The key-value pairs should be of the form {class_variable:prior_probability}.

STEP 4: For each attribute in the training data set, create a dictionary to store the conditional probabilities for each attribute value given each class variable. The key-value pairs should be of the form {(attribute, attribute_value, class_variable):conditional_probability}.

STEP 5: Compute the prior probabilities for each class variable by counting the number of instances in the training data set that belong to each class variable and dividing by the total number of instances.
STEP 6: For each attribute in the training data set, compute the conditional probabilities for each attribute value given each class variable by counting the number of instances in the training data set that have that attribute value and belong to each class variable, and dividing by the number of instances that belong to that class variable.
STEP 7: Load the test data sets from CSV files into lists of dictionaries, following the same format as the training data set.
STEP 8: For each instance in each test data set, compute the posterior probability for each class variable given the attribute values in that instance, using the Naive Bayesian formula:P(class variable | attribute _values) = P(class variable) *product(P(attribute_value | class variable) for attribute_value in attribute_values)
STEP 9: Determine the predicted class variable for each instance in each test data set as the class variable with the highest posterior probability.
STEP 10: Compare the predicted class variables to the actual class variables in each test data set to compute the accuracy of the classifier.
STEP 11: Output the accuracy for each test dataset.

To use the Naive Bayes classifier in Python using scikit-learn (sklearn), follow these steps:
1. Import the necessary libraries: **from sklearn.naive_bayes import GaussianNB**
2. Create an instance of the Naive Bayes classifier: **classifier = GaussianNB()**
3. Fit the classifier to your training data: **classifier.fit(X_train, y_train)**
4. Predict the target values for your test data: **y_pred = classifier.predict(X_test)**
5. Evaluate the performance of the classifier: accuracy = **classifier.score(X_test, y_test)**

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

Outlook	Temperature	Humidity	Wind	Play Tennis
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes

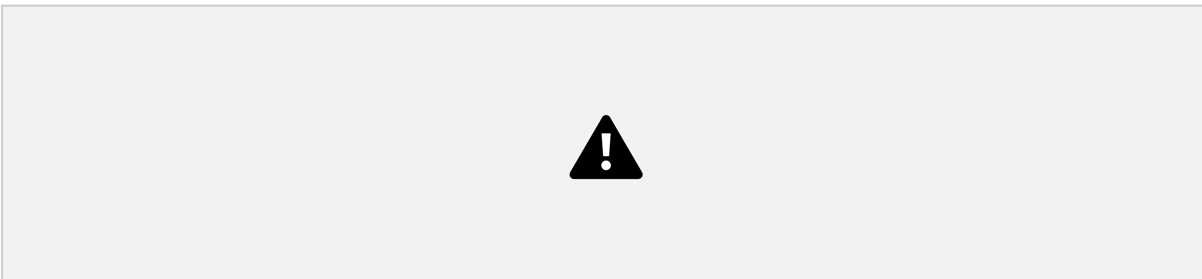
Sunny	**Cool**	**Normal**	**True**	**No**
Overcast	**Cool**	**Normal**	**True**	**Yes**
Rainy	**Mild**	**High**	**False**	**No**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Sunny	**Mild**	**Normal**	**False**	**Yes**
Rainy	**Mild**	**Normal**	**True**	**Yes**
Overcast	**Mild**	**High**	**True**	**Yes**
Overcast	**Hot**	**Normal**	**False**	**Yes**
Sunny	**Mild**	**High**	**True**	**No**

Test dataset :
Today= (Outlook = Sunny,Temperature = Hot, Humidity = Normal, Wind = false〉
play tennis =?
T2 = (Outlook = Rainy, Temp = Hot, Humidity = High, Wind = False) play tennis =?

<u>Naive Bayes Classifier:</u>

P(play tennis = yes) = 9/14 = 0.64
P(play tennis = no) = 5/14 = 0.36

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$



1. Find proportional probabilities as:

P(outlook = “sunny” | playTennis = “yes”) = 3/9 = 0.33
P(outlook = “sunny” | playTennis= “no”) = 2/5 = 0.4
P(Temp = “Hot” | playTennis = “yes”) = 3/9 = 0.33
P(Temp = “Hot” | playTennis = “no”) = 1/5 = 0.2
P(humidity = “high” | playTennis = “yes) = 3/9 =0.33
P(humidity = “high” | playTennis = “no”) = 4/5 = 0.8
P(windy = “weak” | playTennis = “yes”) = 6/9 = 0.67
P(windy= “weak” | playTennis = “no”) = 2/5 = 0.4

P(Yes | today) = (3/9) * (2/9) * (6/9) * (6/9) * (9/14) = 0.33*0.22*0.67* 0.67*0.63 = 0.021
P(No | today)   = (3/5) * {2/5) * (1/5) * (2/5) * (5/14) = 0.6 * 0.4 * 0.2 * 0.4 * 0.36 = 0.0069

Since P(Yes | today) > P(No | today)
So, prediction that golf would be played is ‘Yes’.

2. Find proportional probabilities as:

P(outlook = “rainy” | playTennis = “yes”) = 3/9 = 0.33
P(outlook = “rainy” | playTennis= “no”) = 2/5 = 0.4
P(Temp = “cool” | playTennis = “yes”) = 2/9 = 0.22
P(Temp = “cool” | playTennis = “no”) = 2/5 = 0.4
P(humidity = “high” | playTennis = “yes) = 3/9 =0.33
P(humidity = “high” | playTennis = “no”) = 4/5 = 0.8
P(windy = “true” | playTennis = “yes”) = 3/9 = 0.33
P(windy= “true” | playTennis = “no”) = 3/5 = 0.6

P(Yes | t2) = (3/9) * (2/9) * (6/9) * (3/9) * (9/14) = 0.33 * 0.22 * 0.33 * 0.33 * 0.63 = 0.005
P(No | t2)   = (2/5) * {2/5) * (1/5) * (3/5) * (5/14) = 0.4 * 0.4 * 0.8 * 0.6 * 0.36 = 0.028

Since P(Yes | t2) < P(No | t2)
So, prediction that golf would be played is ‘No’.

Program:

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
data =pd.read_csv('/content/sample_data/tennisdata.csv')
```

```
print("The first 5 Values of data is :\n", data.head())
from sklearn.preprocessing import LabelEncoder
data=data.apply(LabelEncoder().fit_transform)
data.head()
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())
y = data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
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)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

Output:

```
The first 5 Values of data is :
     Outlook Temperature Humidity  Windy PlayTennis
0      Sunny         Hot     High  False         No
1      Sunny         Hot     High   True         No
2   Overcast         Hot     High  False        Yes
3      Rainy        Mild     High  False        Yes
4      Rainy        Cool   Normal  False        Yes

The First 5 values of the train data is
    Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0

The First 5 values of train output is
 0    0
1    0
2    1
3    1
4    1
Name: PlayTennis, dtype: int64

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666
```

Result:

Thus a program to implement Naive Bayesian classifier was written and executed successfully

7. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Aim:

To develop a program that builds an Artificial Neural Network and implements the back propagation algorithm and test by using appropriate data sets

Algorithm:

Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs .

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the steps stopping condition is to be false do step 3 to 10.

Step 3: For each training pair do step 4 to 9 (Feed-Forward).

Step 4: Each input unit receives the signal unit and transmitsthe signal xi signal to all the units.

Step 5: Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input zinj = v0j + Σxivij (i=1 to n)

Applying activation function zj = f(zinj) and sends this signals to all units in the layer about i.e output units

For each output l=unit yk = (k=1 to m) sums its weighted input signals.

yink = w0k + Σ ziwjk (j=1 to a)

and applies its activation function to calculate the output signals. yk = f(yink)

Backpropagation Error :

Step 6: Each output unit yk (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as: δk = (tk – yk) + yink

Step 7: Each hidden unit Zj (j=1 to a) sums its input from all units in the layer above

δinj = Σ δj wjk

The error information term is calculated as : δj = δinj + zinj

Updation of weight and bias :

Step 8: Each output unit yk (k=1 to m) updates its bias and weight (j=1 to a).

The weight correction term is given by : Δ wjk = α δk zj and the bias correction term is given by Δwk = α δk.

therefore wjk(new) = wjk(old) + Δ wjk w0k(new) = wok(old) + Δ wok

for each hidden unit zj (j=1 to a) update its bias and weights (i=0 to n) the weight connection term Δ vij = α δj xi and the bias connection on term Δ v0j = α δj

Therefore vij(new) = vij(old) + Δvij v0j(new) = v0j(old) + Δv0j

Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

Program:

```
import numpy as np
class NeuralNetwork:
  def __init__(self, input_size, hidden_size, output_size):
    self.input_size = input_size
    self.hidden_size = hidden_size
```

```
    self.output_size = output_size
    # Initialize weights and biases
    self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
    self.bias_hidden = np.zeros((1, self.hidden_size))
    self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)
    self.bias_output = np.zeros((1, self.output_size))

    def sigmoid(self, x):
      return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward(self, X):
        # Forward pass
      self.hidden_layer_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
      self.hidden_layer_output = self.sigmoid(self.hidden_layer_input)
      self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output)
                                                                      + self.bias_output
      self.output = self.sigmoid(self.output_layer_input)
      return self.output

    def backward(self, X, y, output, learning_rate):
        # Backpropagation
        error = y - output
        output_delta = error * self.sigmoid_derivative(output)
        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_layer_output)
        # Update weights and biases
        self.weights_hidden_output += self.hidden_layer_output.T.dot(output_delta) *
                                                                      learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate
        self.weights_input_hidden += X.T.dot(hidden_delta) * learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output, learning_rate)
            if epoch % 100 == 0:
                print(f'Epoch {epoch}: Error {np.mean(np.square(y - output))}')

# Example usage:
input_size = 1
hidden_size = 1
output_size = 1

# Initialize neural network
nn = NeuralNetwork(input_size, hidden_size, output_size)

# Example training data
X = np.array([[5]]) # Input
y = np.array([[1]]) # Target output

# Train the neural network
nn.train(X, y, epochs=1000, learning_rate=0.1)

# Make predictions
```

```
predictions = nn.forward(X)
print("Predictions:", predictions)
```

```
# Example usage:
# Initialize neural network
input_size = 2
hidden_size = 3
output_size = 1
nn = NeuralNetwork(input_size, hidden_size, output_size)
# Example training data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# Train the neural network
nn.train(X, y, epochs=1000, learning_rate=0.1)
# Make predictions
predictions = nn.forward(X)
print("Predictions:", predictions)
```

Output:

```
Epoch 0: Error 0.3008608536764959
Epoch 100: Error 0.24656063190137567
Epoch 200: Error 0.24365431038885452
Epoch 300: Error 0.24071478359394938
Epoch 400: Error 0.2374846988945396
Epoch 500: Error 0.23381288477820023
Epoch 600: Error 0.22960397595417048
Epoch 700: Error 0.22480692491055732
Epoch 800: Error 0.21942217923276036
Epoch 900: Error 0.213510750088039
Predictions: [[0.36569489]
 [0.61052463]
 [0.52170034]
 [0.56086062]]
```

Result:

Thus a program that builds an Artificial Neural Network is written and the back propagation algorithm is implemented and tested by using appropriate data sets.

8. Write a program to demonstrate clustering rule process on dataset using simple

k-means

Aim:

To demonstrate clustering rule process on dataset using k-means method

Algorithm:

1. Choose the number of clusters k:

   The first step in k-means is to pick the number of clusters, k.

2. Select k random points from the data as centroids:

   Randomly select the centroid for each cluster.

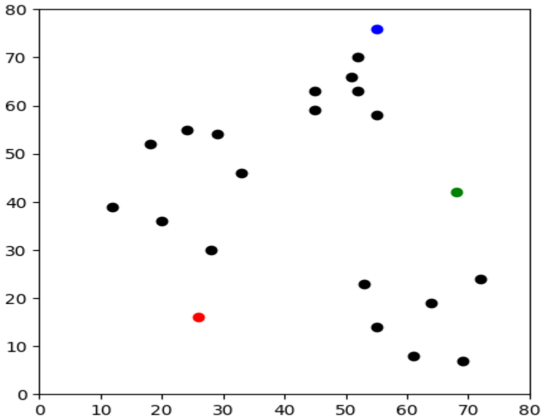
3. Assign all the points to the closest cluster centroid
4. Recompute the centroids of newly formed clusters
5. Repeat steps 3 and 4We then repeat steps 3 and 4:
6. Stopping Criteria for K-Means Clustering
   a) Centroids of newly formed clusters do not change
   b) Points remain in the same cluster
   c) Maximum number of iterations is reached

Program:

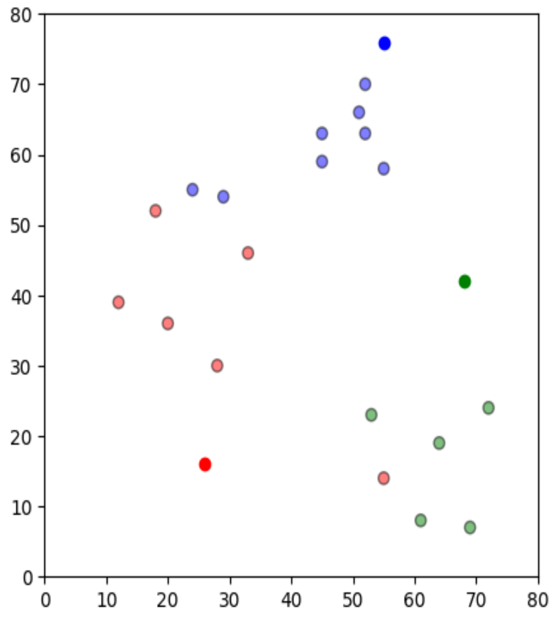
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
 %matplotlib inline
df= pd.DataFrame({
   'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],
   'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 08, 19, 07, 24]    })
np.random.seed(200)
k = 3
centroids={ i+1: [np.random.randint(0, 80), np.random.randint(0, 80)] for i in range(k) }
print(centroids)
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap={1:'r', 2:'g', 3:'b'}
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()


def assignment(df, centroids):
    for i in centroids.keys():
      df['distance_from_{}'.format(i)]=(
            np.sqrt((df['x']- centroids[i][0]) ** 2 + (df['y']- centroids[i][1]) ** 2 )
     )
   centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.keys()]
   df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
   df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
   df['color'] = df['closest'].map(lambda x: colmap[x])
   return df

df = assignment(df, centroids)
print(df.head())

fig = plt.figure(figsize=(5,5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha = 0.5, edgecolor = 'k')
for i in centroids.keys():
   plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

import copy
old_centroids = copy.deepcopy(centroids)

def update(k):
```

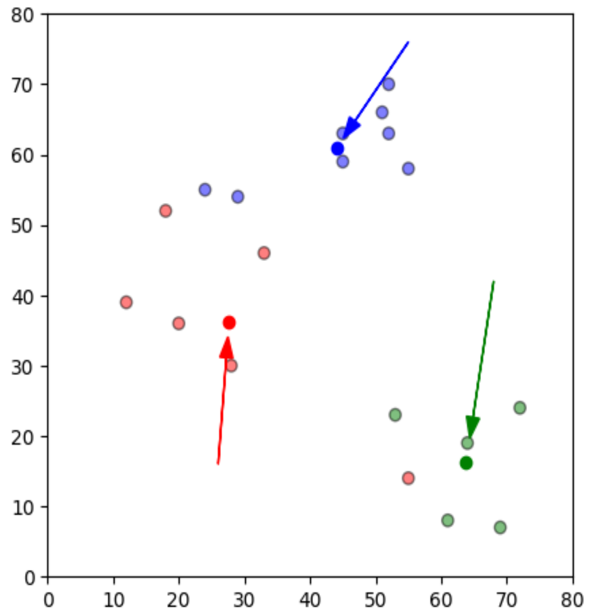
```
  for i in centroids.keys():
    centroids[i][0] = np.mean(df[df['closest'] == i]['x'] )
    centroids[i][1] = np.mean(df[df['closest'] == i]['y'] )
  return k

centroids = update(centroids)
fig = plt.figure(figsize=(5,5))
ax = plt.axes()
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
  plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
for i in old_centroids.keys():
   old_x = old_centroids[i][0]
   old_y = old_centroids[i][1]
   dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
   dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
   ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i], ec=colmap[i])
plt.show()

df= assignment(df,centroids)
fig = plt.figure(figsize=(5,5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha = 0.5, edgecolor = 'k')
for i in centroids.keys():
   plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
while True:
   closest_centroids = df['closest'].copy(deep=True)
   centroids = update(centroids)
   df = assignment(df, centroids)
   if closest_centroids.equals(df['closest']):
      break
fig = plt.figure(figsize=(5,5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha = 0.5, edgecolor = 'k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

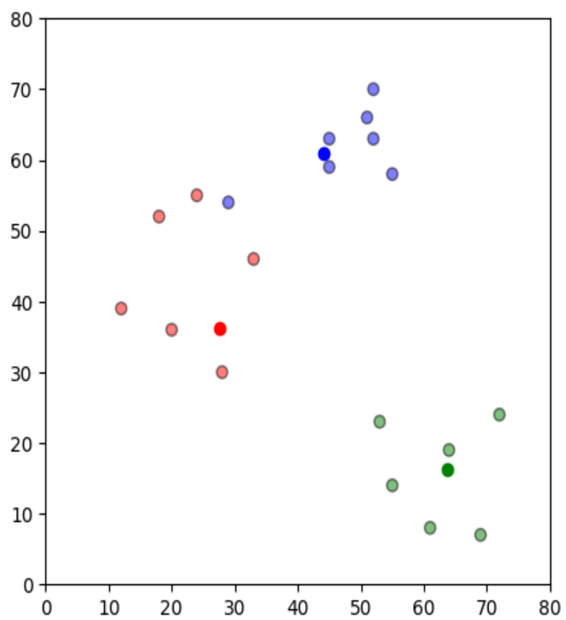
Output:

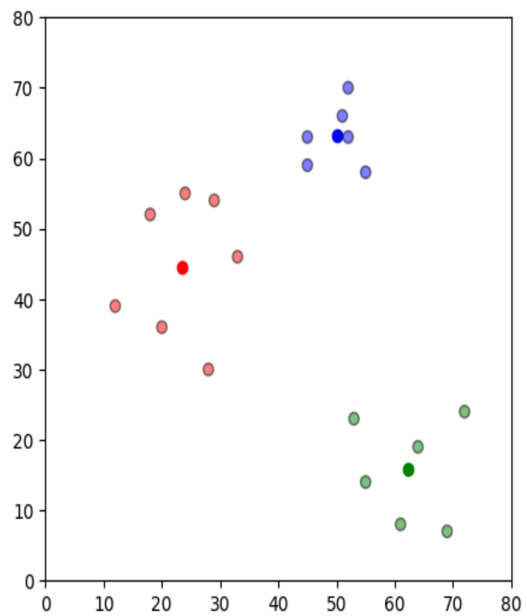


	x	y	dist_from_1	dist_from_2	dist_from_3	closest	color
0	12	39	26.925824	56.080300	56.727418	1	r
1	20	36	20.880613	8.373546	53.150729	1	r
2	28	30	14.142136	41.761226	53.338541	1	r
3	18	52	36.878178	50.990195	44.102154	1	r
4	29	54	38.118237	40.804412	34.058773	3	b







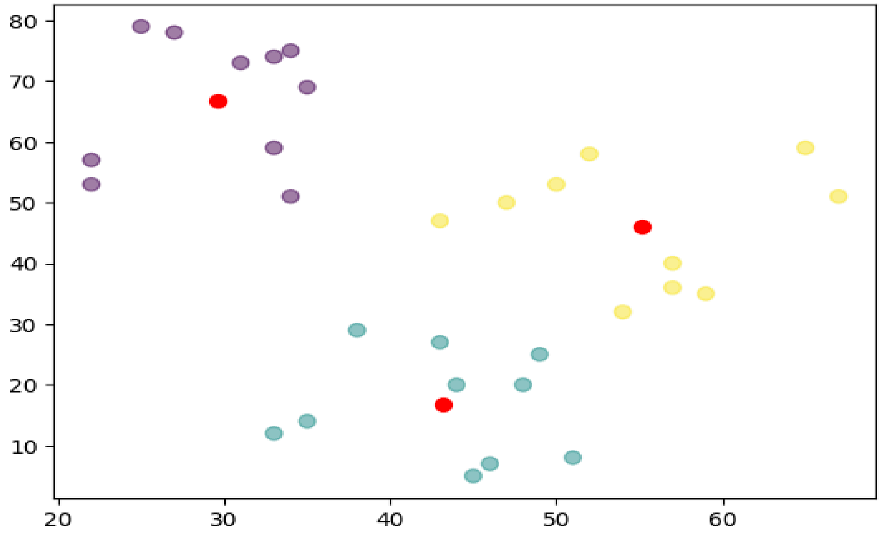


(or)

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv("/content/sample_data/kmeansdata.csv")
kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
plt.scatter(df['X'], df['Y'], c=kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

Output:

```
[[29.6 66.8]
 [43.2 16.7]
 [55.1 46.1]]
```



Result:

Thus a program to demonstrate clustering rule process on dateset using k-means method was written and executed successfully

9. Write a program to predict the winning team in IPL matches

Aim::
To write a program to predict the winning team in IPL matches.

Program:

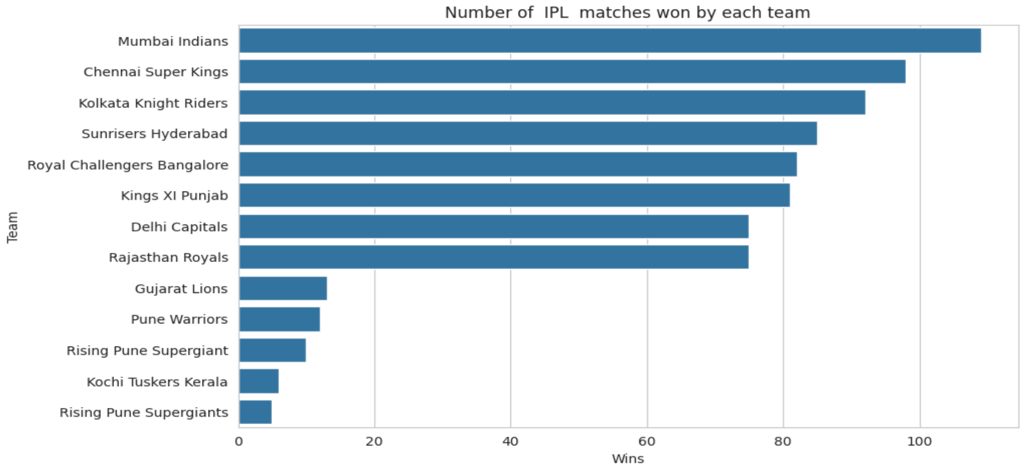
```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style("whitegrid")
import matplotlib.pyplot as plt
import sklearn

data = pd.read_csv("/content/sample_data/matches.csv")
data.describe()

data.isnull().sum()
data = data.iloc[:,:-1]
data.dropna(inplace=True)
data["team1"].unique()

#for Delhi Capitals
data['team1']=data['team1'].str.replace('Delhi Daredevils','Delhi Capitals')
data['team2']=data['team2'].str.replace('Delhi Daredevils','Delhi Capitals')
data['winner']=data['winner'].str.replace('Delhi Daredevils','Delhi Capitals')
#for sunrisers Hyderabad
data['team1']=data['team1'].str.replace('Deccan Chargers','Sunrisers Hyderabad')
data['team2']=data['team2'].str.replace('Deccan Chargers','Sunrisers Hyderabad')
data['winner']=data['winner'].str.replace('Deccan Chargers','Sunrisers Hyderabad')

#Number of IPL matches won by each team.
plt.figure(figsize = (10,6))
sns.countplot(y = 'winner',data = data,order= data['winner'].value_counts().index)
plt.xlabel('Wins')
plt.ylabel('Team')
plt.title('Number of  IPL  matches won by each team')
```

```
#Total number of matches played in a different stadium
plt.figure(figsize = (10,6))
sns.countplot(y = 'venue',data = data,order = data['venue'].value_counts().iloc[:10].index)
plt.xlabel('No of matches',fontsize=12)
plt.ylabel('Venue',fontsize=12)
plt.title('Total Number of matches played in different stadium')

#The decision was taken by the toss winning team.
plt.figure(figsize = (10,6))
sns.countplot(x = "toss_decision", data=data)
plt.xlabel('Toss Decision',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.title('Toss Decision')

x = ["city", "toss_decision", "result", "dl_applied"]
for i in x:
  print("------------")
  print(data[i].unique())
  print(data[i].value_counts())

# dropping some of the features that don’t affect our result.

data.drop(["id", "season","city","date","player_of_match","venue","umpire1","umpire2"],
axis=1, inplace=True)
data.describe()

X = data.drop(["winner"], axis=1)
y = data["winner"]
print(y.unique())

X = pd.get_dummies(X, ["team1","team2", "toss_winner", "toss_decision", "result"],
drop_first = True)

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
print(y)
y = le.fit_transform(y)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size = 0.8)

from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB

model_df={}
def model_val(model,X,y):
  X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=42)
  model.fit(X_train,y_train)
  y_pred=model.predict(X_test)
  print(f"{model} accuracy is {accuracy_score(y_test,y_pred)}")
  print(y_pred[0:5],y_test[0:5])

model = DecisionTreeClassifier()
model_val(model,X,y)
```

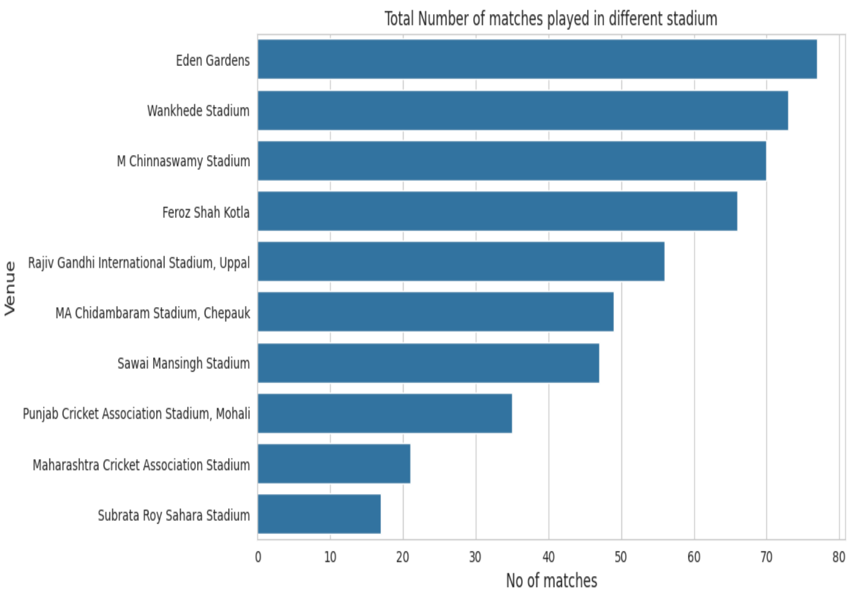
model = GaussianNB()
model_val(model,X,y)

Output:

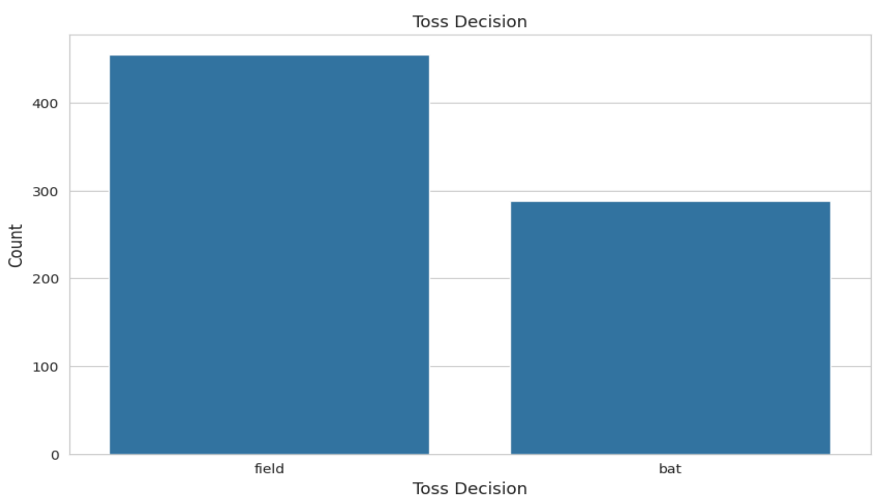
Text(0.5,1.0,'Number of IPL matches won by each team')



Text(0.5, 1.0, 'Total Number of matches played in different stadium')



Text(0.5, 1.0, 'Toss Decision')



DecisionTreeClassifier() accuracy is 0.959731543624161

[5 11 3 5 6] [5 11 3 5 6]

GaussianNB() accuracy is 0.2214765100671141

[3 9 8 5 8] [5 11 3 5 6]

Result:

Thus a program to predict the winning team in IPL matches is written and executed.

10. Write a program to predict the eligibility of a customer for loan disbursement.

Aim::
To write a program to predict the eligibility of a customer for loan disbursement

Program:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import  cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
# Convert dataset into dataframe
data = pd.read_csv("/content/drive/MyDrive/Loan_status.csv")
#print the first 5 and last 5 dataframes in dataset
print(data.head())
print(data.tail())
#print the no of rows and colums of dataset
print("No of rows",data.shape[0])
print("No of columns",data.shape[1])
# print the details of dataset
print(data.info() )
# get the information of noll values count'
data.isnull().sum()
```

```
#.sum()*100/len(data)

# get the information of noll values count'
data.isnull().sum()*100/len(data)

# handle the missing value if less than 5% drop or handle
columns=['Gender','Married','Dependents','LoanAmount','Loan_Amount_Term']
data = data.dropna(subset=columns)
data.isnull().sum()*100/len(data)

data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])
print(data.isnull().sum()*100/len(data))
# handling categorical colums ex dependents - 3+ modified to 4
data['Dependents'] = data['Dependents'].replace(to_replace='3+', value='4')
data['Dependents'].unique()


label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])
data['Married'] = label_encoder.fit_transform(data['Married'])
data['Education'] = label_encoder.fit_transform(data['Education'])
data['Self_Employed'] = label_encoder.fit_transform(data['Self_Employed'])
data['Property_Area'] = label_encoder.fit_transform(data['Property_Area'])
data['Loan_Status'] = label_encoder.fit_transform(data['Loan_Status'])
data = data.drop("Loan_ID",axis=1)
print(data.head())
X=data.drop('Loan_Status',axis=1)
y= data['Loan_Status']
cols =['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term']
St = StandardScaler()
X[cols]= st.fit_transform(X[cols])
print(X[cols])
model_df={}

def model_val(model,X,y):
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
   model.fit(X_train, y_train)
   y_pred = model.predict (X_test)
   print(f"{model} accuracy is {accuracy_score(y_test, y_pred)}")
   score = cross_val_score(model, X, y, cv=5)
   print(f"{model} avg cross val score is {np.mean(score)}")
   model_df[model]=round(np.mean(score)*100, 2)
model = LogisticRegression()
model_val(model, X, y)
model = DecisionTreeClassifier()
```

```
model_val(model, X, y)
model = GaussianNB()
model_val(model, X, y)
import joblib
Nb = GaussianNB()
nb.fit(X,y)
joblib.dump(nb, 'Loan_status_predict')
Model = joblib.load('Loan_status_predict')
print(y.head(10))
df = X.iloc[6:7]
result = model.predict(df)
if result==1:
    print('loan approved')
else:
    print('loan not approved')
print(model_df)
```

Output:

LogisticRegression() accuracy is **0.7927927927927928**
LogisticRegression() avg cross val score is **0.802964782964783**
DecisionTreeClassifier() accuracy is **0.7027027027027027**
DecisionTreeClassifier() avg cross val score is **0.6979852579852579**
GaussianNB() accuracy is **0.8288288288288288**
GaussianNB() avg cross val score is **0.7866830466830466**

1 0
2 1
3 1
4 1
5 1
6 1
7 0
8 1
9 0
10 1
Name: Loan_Status, dtype: int64
loan not approved
{ LogisticRegression() : **80.3,** DecisionTreeClassifier() : **69.8**, GaussianNB() : **78.67 }**

Result:

Thus a program to predict the eligibility of a customer for loan disbursement is written and executed.