EXPERIMENT 1:

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

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance xFor each attribute constraint ai in h

If the constraint ai is satisfied by xThen do nothing

Else replace ai in h by the next more general constraint that is satisfied by x

3. **Output** hypothesis h

Training Examples:

Example Sky AirTemp Humidity Wind Water Forecast EnjoySport1 Sunny Warm Normal Strong Warm Same Yes

2 Sunny Warm High Strong Warm Same Yes 3 Rainy Cold High Strong Warm Change No 4 Sunny Warm High Strong Cool Change Yes

Program:

```
import csv
num_attributes = 6a = []
print("\n The Given Training Data Set \n")with
open('enjoysport.csv', 'r') as csvfile:
 reader = csv.reader(csvfile)for row in
 reader:
 a.append (row)print(row)
print("\n The initial value of hypothesis: ")hypothesis =
['0'] * num_attributes print(hypothesis)
for j in range(0,num_attributes):
 hypothesis[j] = a[0][j];
print("\n Find S: Finding a Maximally Specific Hypothesis\n")for i in
range(0,len(a)):
 if a[i][num_attributes]=='yes':
 for j in range(0,num_attributes):
 if a[i][j]!=hypothesis[j]:
 hypothesis[j]='?'
 ?'else :
 hypothesis[j]= a[i][j]
 print(" For Training instance No:{0} the hypothesis is
".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given TrainingExamples :\n")
print(hypothesis)
```

Data Set:

sunny warm normal strong warm same yessunny warm
high strong warm same yes rainy cold high strong warm
change no sunny warm high strong cool change yes

Output:

The Given Training Data Set
['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 initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
Find S: Finding a Maximally Specific HypothesisFor Training Example No:0 the hypothesis is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same'] For Training Example No:1 the hypothesis is
['sunny', 'warm', '?', 'strong', 'warm', 'same'] For Training Example No:2 the hypothesis is'sunny', 'warm', '?', 'strong', 'warm', 'same'] For Training Example No:3 the hypothesis is
'sunny', 'warm', '?', 'strong', '?', '?']
The Maximally Specific Hypothesis for a given Training Examples:['sunny', 'warm', '?', 'strong', '?', '?']

WEEK-2: EXPERIMENT 2 :

AIM: Python Implementation of Candidate-Elimination

Below is the algorithm for Candidate-Elimination

- Firstly, read the data from the CSV file.
- Initialize General and Specific Hypothesis.
- If the example is positive, [Follow Find-S algorithm]
- If attribute == hypothesis value then do nothing.Else
- Make the attribute more general i.e replaces the attribute with?
- If the example is negative
- Make the generalized hypothesis more specific.

Below is the code for Candidate-Elimination

Contents in candidate.csv

```
sky, air temp,  humidity, wind, water, for cast, enjoy sport.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.
```

program

```
import numpy as npimport
pandas as pd
# Reading the data from CSV file data =
pd.read_csv('candidate.csv') concepts =
np.array(data.iloc[:,:-1])print("\nInstances
are:\n",concepts)target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def train(concepts, target):
    # Initializing general and specific hypothesisspecific_h =
    concepts[0].copy()
    print("\nInitialization of specific hypothesis and general hypothesis")print("\nSpecific
    Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]print("\nGeneric Boundary:
    ",general_h)
    for i, val in enumerate(concepts):
        print("\nInstance", i+1 , "is ", val)#positive
        example
        if target[i] == "yes": print("Instance is Positive
            ") for x in range(len(specific_h)):
                if val[x]!= specific_h[x]:specific_h[x]
                    ='?' general_h[x][x] ='?'
        #negative example if target[i]
        ]== "no":
```

```
        print("Instance is Negative ") for x in
        range(len(specific_h)):
            if val[x]!= specific_h[x]: general_h[x][x] =
                specific_h[x]

            else:
                general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h) print("Generic
        Boundary after ", i+1, "Instance is ", general_h)print("\n")

    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 = train(concepts, target)# displaying
Specific_hypothesis
print("Final Specific_h: ", s_final, sep="\n")# displaying
Generalized_Hypothesis print("Final General_h: ", g_final,
sep="\n")
```

OUTPUT:

```
Instances are:
[['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame'] ['\twarm'
'\thigh' '\tstrong' '\twarm' '\tsame' '\tyes.'] ['\tcold' '\thigh'
'\tstrong' '\twarm' '\tchange' '\tno.'] ['\twarm' '\thigh' '\tstrong'
'\tcool' '\tchange' '\tyes.']]Target Values are:  ['\tyes.' nan nan
nan]
Initialization of specific hypothesis and general hypothesis
Specific Boundary:  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 1 is  ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']
Specific Bundary after 1 Instance is ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame'] Generic Boundary
after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 2 is  ['\twarm' '\thigh' '\tstrong' '\twarm' '\tsame' '\tyes.']
Specific Bundary after 2 Instance is ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame'] Generic Boundary
after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 3 is  ['\tcold' '\thigh' '\tstrong' '\twarm' '\tchange' '\tno.']
Specific Bundary after 3 Instance is ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame'] Generic Boundary
after 3 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 4 is  ['\twarm' '\thigh' '\tstrong' '\tcool' '\tchange' '\tyes.']
Specific Bundary after 4 Instance is ['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame'] Generic Boundary
after 4 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?','?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
```

['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['\twarm' '\tnormal' '\tstrong' '\t' 'warm' '\tsame']Final
General_h:
[]

Experiment-3:

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

data set

```
outlook,temperature,humidity,windy,play
sunny,hot,high,false,no
sunny,hot,high,true,no
overcast,hot,high,false,yes
rain,mild,high,false,yes
rain,cool,normal,false,yes
rain,cool,normal,true,no
overcast,cool,normal,true,yes
sunny,mild,high,false,no
sunny,cool,normal,false,yes
rain,mild,normal,false,yes
sunny,mild,normal,true,yes
overcast,mild,high,true,yes
overcast,hot,normal,false,yes
rain,mild,high,true,no
```

Program-

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text

# Load the dataset from the CSV file
data = pd.read_csv('dataset.csv')

# One-hot encode the categorical features
data = pd.get_dummies(data, columns=['outlook', 'temperature', 'humidity', 'windy'],
drop_first=True)

# Separate features (X) and target (y)
X = data.drop('play', axis=1)
y = data['play']

# Create a Decision Tree classifier using ID3-like criterion
dtree = DecisionTreeClassifier(criterion='entropy')

# Fit the model to the data
dtree.fit(X, y)

# Print the decision tree rules
tree_rules = export_text(dtree, feature_names=list(X.columns))
print("Decision Tree Rules:")
print(tree_rules)

# Classify a new sample (remember to one-hot encode it)
new_sample = pd.DataFrame({'outlook_overcast': [0],
                           'outlook_rain': [1],
                           'outlook_sunny': [0],
                           'temperature_cool': [0],
```

```
                    'temperature_hot': [1],
                    'temperature_mild': [0],
                    'humidity_normal': [1],
                    'humidity_high': [0],
                    'windy_true': [0]})

predicted_class = dtree.predict(new_sample)
print("\nPredicted class for new sample:", predicted_class[0])
```

output-

```
|--- humidity_normal <= 0.50
|   |--- outlook_sunny <= 0.50
|   |   |--- outlook_rain <= 0.50
|   |   |   |--- class: yes
|   |   |--- outlook_rain >  0.50
|   |   |   |--- windy_True <= 0.50
|   |   |   |   |--- class: yes
|   |   |   |--- windy_True >  0.50
|   |   |   |   |--- class: no
|   |--- outlook_sunny >  0.50
|   |   |--- class: no
|--- humidity_normal >  0.50
|   |--- windy_True <= 0.50
|   |   |--- class: yes
|   |--- windy_True >  0.50
|   |   |--- outlook_rain <= 0.50
|   |   |   |--- class: yes
|   |   |--- outlook_rain >  0.50
|   |   |   |--- class: no
```

```
    predicted_class = dtree.predict(new_sample)
Feature names unseen at fit time:
- humidity_high
- outlook_overcast
- temperature_cool
- windy_true
Feature names seen at fit time, yet now missing:
- windy_True
```

Experiment-4:

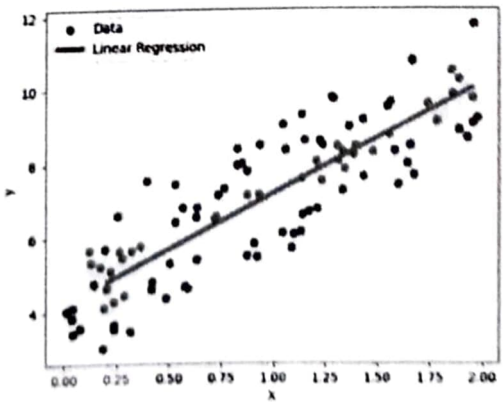
Exercises to solve the real-world problems using the following machine learning methods:

a) Linear Regression b) Logistic Regression c) Binary Classifier

a) Linear Regression: Predicting House Prices

program-

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE) to evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Plot the data and the regression line
plt.scatter(X, y, label="Data")
plt.plot(X_test, y_pred, color='red', linewidth=3, label="Linear Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



b) Logistic Regression: Email Spam Detection

Program-

```
# Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Generate some sample data (replace with your dataset)
X, y = np.random.rand(100, 2), np.random.randint(0, 2, 100)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and display classification report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
```

output-

```
Accuracy: 0.4
Confusion Matrix:
[[ 8  0]
 [12  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.40      1.00      0.57         8
           1       0.00      0.00      0.00        12

    accuracy                           0.40        20
   macro avg       0.20      0.50      0.29        20
weighted avg       0.16      0.40      0.23        20

○ karthik@KARTHIKs-iMac Ml programs %
```

c) Binary Classifier (e.g., Support Vector Machine): Spam vs. Non-Spam

program-

```
# Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Generate some sample data (replace with your dataset)
X, y = np.random.rand(100, 2), np.random.randint(0, 2, 100)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the Support Vector Machine classifier
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and display classification report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
```

output-

```
Accuracy: 0.5
Confusion Matrix:
[[ 0 10]
 [ 0 10]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.50      1.00      0.67        10

    accuracy                           0.50        20
   macro avg       0.25      0.50      0.33        20
weighted avg       0.25      0.50      0.33        20

karthik@KARTHIKs-iMac Ml programs %
```

Experiment-5:

Develop a program for Bias, Variance, Remove duplicates , Cross Validation

data set-

Feature1	Feature2	Target
0.699121252	0.468074753	0.741269036
0.426545885	0.810380384	0.911906157
0.832834111	0.267649446	0.541757077
0.497681085	0.917919707	0.979771874
0.643239664	0.862130474	0.105824393
0.726679637	0.325280707	0.365451779
0.261989406	0.947877329	0.509971774
0.215136508	0.457552325	0.099166155
0.201004086	0.362712476	0.073823546

program-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score

# Step 1: Generate sample data and save it as a CSV file
data = {
    'Feature1': np.random.rand(100),
    'Feature2': np.random.rand(100),
    'Target': np.random.rand(100)
}

df = pd.DataFrame(data)
df.to_csv('sample_data.csv', index=False)

# Step 2: Remove duplicates from the data
def remove_duplicates(input_file, output_file):
    df = pd.read_csv(input_file)
    df.drop_duplicates(inplace=True)
    df.to_csv(output_file, index=False)

input_file = 'sample_data.csv'
output_file = 'sample_data_no_duplicates.csv'
remove_duplicates(input_file, output_file)
```

```
# Step 3: Perform cross-validation and measure bias and variance
def bias_variance_tradeoff(input_file):
    df = pd.read_csv(input_file)
    X = df[['Feature1', 'Feature2']]
    y = df['Target']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)

    # Measure bias and variance using cross-validation
    mse = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
    bias = np.mean(-mse)
    variance = np.var(-mse, ddof=1)

    print(f'Bias: {bias}')
    print(f'Variance: {variance}')

bias_variance_tradeoff(output_file)
```

output-

Experiment-6:

Write a program to implement Categorical Encoding, One-hot Encoding

program-

```
import pandas as pd

# Sample data
data = {'Category': ['A', 'B', 'A', 'C', 'B']}
df = pd.DataFrame(data)

# Categorical encoding using pandas
category_mapping = {'A': 0, 'B': 1, 'C': 2}
df['Category_Encoded'] = df['Category'].map(category_mapping)

print(df)
```

```
import pandas as pd

# Sample data
data = {'Category': ['A', 'B', 'A', 'C', 'B']}
df = pd.DataFrame(data)

# One-hot encoding using pandas
df_encoded = pd.get_dummies(df, columns=['Category'], prefix=['Category'])

print(df_encoded)
```

output-

```
Bias: 0.09425988265197807
Variance: 9.837820879983709e-06
karthik@KARTHIKs-iMac Ml programs % /usr/local/bin/python3 "/Users/karthik/Documents/Ml programs/6a.py"
  Category  Category_Encoded
0        A                 0
1        B                 1
2        A                 0
3        C                 2
4        B                 1
karthik@KARTHIKs-iMac Ml programs % /usr/local/bin/python3 "/Users/karthik/Documents/Ml programs/6b.py"
   Category_A  Category_B  Category_C
0        True       False       False
1       False        True       False
2        True       False       False
3       False       False        True
4       False        True       False
karthik@KARTHIKs-iMac Ml programs %
```

Experiment-7:

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

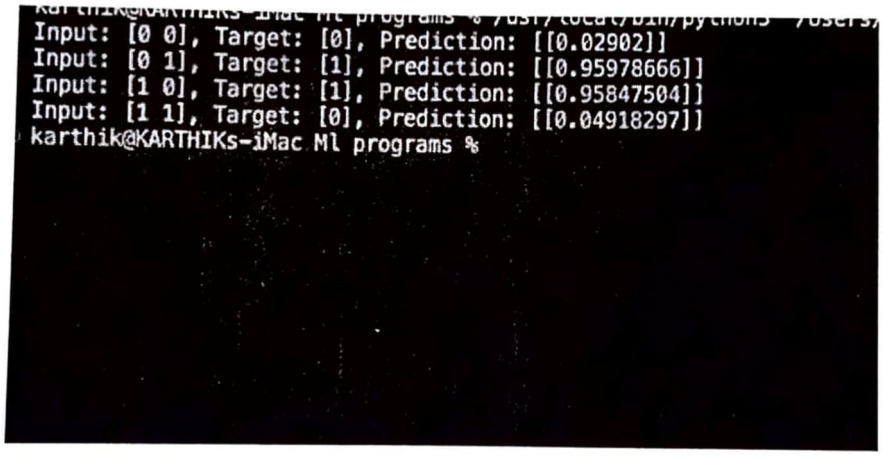
program-

```
import numpy as np

# Define the sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# Define the neural network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights with random values
        self.weights_input_hidden = np.random.uniform(-1, 1, (input_size,
hidden_size))
        self.weights_hidden_output = np.random.uniform(-1, 1, (hidden_size,
output_size))

        # Initialize biases to zero
        self.bias_hidden = np.zeros((1, hidden_size))
        self.bias_output = np.zeros((1, output_size))

    def forward(self, inputs):
        # Calculate the dot product of inputs and weights for the hidden layer
        self.hidden_layer_input = np.dot(inputs, self.weights_input_hidden) +
self.bias_hidden
        self.hidden_layer_output = sigmoid(self.hidden_layer_input)

        # Calculate the dot product of the hidden layer output and weights for
the output layer
        self.output_layer_input = np.dot(self.hidden_layer_output,
self.weights_hidden_output) + self.bias_output
        self.output_layer_output = sigmoid(self.output_layer_input)

        return self.output_layer_output
```

```
def backward(self, inputs, targets, learning_rate):
        # Calculate the error at the output layer
        output_error = targets - self.output_layer_output
        output_delta = output_error *
sigmoid_derivative(self.output_layer_output)

        # Calculate the error at the hidden layer
        hidden_layer_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_layer_delta = hidden_layer_error *
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 += inputs.T.dot(hidden_layer_delta) *
learning_rate
        self.bias_hidden += np.sum(hidden_layer_delta, axis=0, keepdims=True)
* learning_rate

    def train(self, inputs, targets, learning_rate, epochs):
        for _ in range(epochs):
            for i in range(len(inputs)):
                input_data = inputs[i:i+1]
                target = targets[i:i+1]
                self.forward(input_data)
                self.backward(input_data, target, learning_rate)

    def predict(self, inputs):
        return self.forward(inputs)

# Create a toy dataset
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
targets = np.array([[0], [1], [1], [0]])

# Create and train the neural network
input_size = 2
hidden_size = 4
output_size = 1
learning_rate = 0.1
epochs = 10000

nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(inputs, targets, learning_rate, epochs)
```

```
# Test the trained neural network
for i in range(len(inputs)):
    input_data = inputs[i]
    target = targets[i]
    prediction = nn.predict(input_data)
    print(f"Input: {input_data}, Target: {target}, Prediction: {prediction}")
```

output-



Experiment-8:

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

program-

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target # Target labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize the k-NN classifier with a specific 'k' value
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Fit the classifier on the training data
knn_classifier.fit(X_train, y_train)

# Predict the labels for the test data
y_pred = knn_classifier.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print both correct and wrong predictions
for i in range(len(y_test)):
    if y_pred[i] == y_test[i]:
        print(f"Sample {i + 1}: Predicted={iris.target_names[y_pred[i]]},
Actual={iris.target_names[y_test[i]]} (Correct)")
    else:
        print(f"Sample {i + 1}: Predicted={iris.target_names[y_pred[i]]},
Actual={iris.target_names[y_test[i]]} (Wrong)")
```

output-

```
Accuracy: 100.00%
Sample 1: Predicted=versicolor, Actual=versicolor (Correct)
Sample 2: Predicted=setosa, Actual=setosa (Correct)
Sample 3: Predicted=virginica, Actual=virginica (Correct)
Sample 4: Predicted=versicolor, Actual=versicolor (Correct)
Sample 5: Predicted=versicolor, Actual=versicolor (Correct)
Sample 6: Predicted=setosa, Actual=setosa (Correct)
Sample 7: Predicted=versicolor, Actual=versicolor (Correct)
Sample 8: Predicted=virginica, Actual=virginica (Correct)
Sample 9: Predicted=versicolor, Actual=versicolor (Correct)
Sample 10: Predicted=versicolor, Actual=versicolor (Correct)
Sample 11: Predicted=virginica, Actual=virginica (Correct)
Sample 12: Predicted=setosa, Actual=setosa (Correct)
Sample 13: Predicted=setosa, Actual=setosa (Correct)
Sample 14: Predicted=setosa, Actual=setosa (Correct)
Sample 15: Predicted=setosa, Actual=setosa (Correct)
Sample 16: Predicted=versicolor, Actual=versicolor (Correct)
Sample 17: Predicted=virginica, Actual=virginica (Correct)
Sample 18: Predicted=versicolor, Actual=versicolor (Correct)
Sample 19: Predicted=versicolor, Actual=versicolor (Correct)
Sample 20: Predicted=virginica, Actual=virginica (Correct)
Sample 21: Predicted=setosa, Actual=setosa (Correct)
Sample 22: Predicted=virginica, Actual=virginica (Correct)
Sample 23: Predicted=setosa, Actual=setosa (Correct)
Sample 24: Predicted=virginica, Actual=virginica (Correct)
Sample 25: Predicted=virginica, Actual=virginica (Correct)
Sample 26: Predicted=virginica, Actual=virginica (Correct)
Sample 27: Predicted=virginica, Actual=virginica (Correct)
Sample 28: Predicted=virginica, Actual=virginica (Correct)
Sample 29: Predicted=setosa, Actual=setosa (Correct)
Sample 30: Predicted=setosa, Actual=setosa (Correct)
Sample 31: Predicted=setosa, Actual=setosa (Correct)
Sample 32: Predicted=setosa, Actual=setosa (Correct)
Sample 33: Predicted=versicolor, Actual=versicolor (Correct)
Sample 34: Predicted=setosa, Actual=setosa (Correct)
Sample 35: Predicted=setosa, Actual=setosa (Correct)
Sample 36: Predicted=virginica, Actual=virginica (Correct)
Sample 37: Predicted=versicolor, Actual=versicolor (Correct)
Sample 38: Predicted=setosa, Actual=setosa (Correct)
Sample 39: Predicted=setosa, Actual=setosa (Correct)
Sample 40: Predicted=setosa, Actual=setosa (Correct)
Sample 41: Predicted=virginica, Actual=virginica (Correct)
Sample 42: Predicted=versicolor, Actual=versicolor (Correct)
Sample 43: Predicted=versicolor, Actual=versicolor (Correct)
Sample 44: Predicted=setosa, Actual=setosa (Correct)
Sample 45: Predicted=setosa, Actual=setosa (Correct)
karthik@KARTHIKs-iMac Ml programs %
```

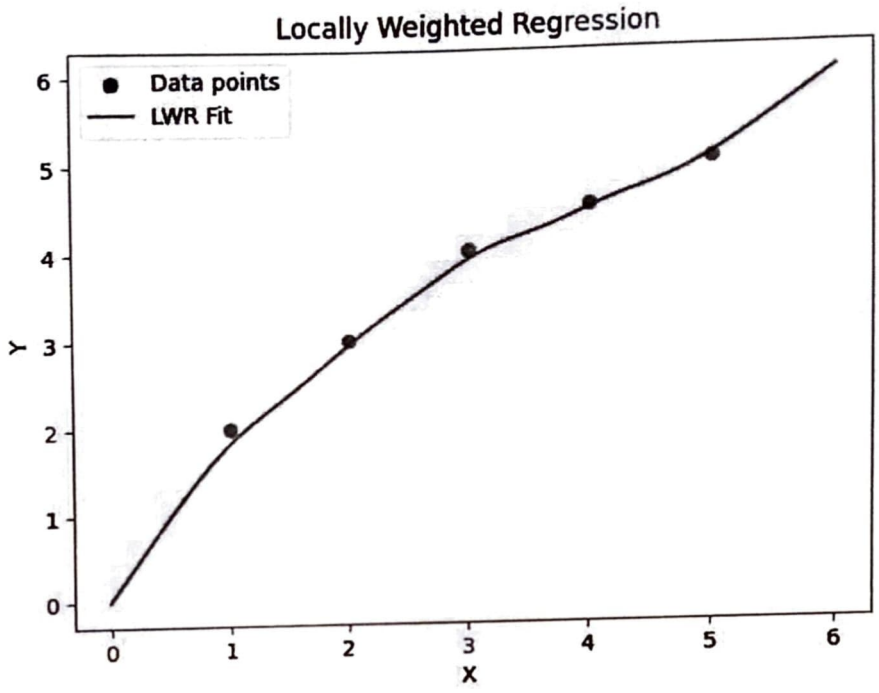
Experiment-9:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

program-

```
import numpy as np
import matplotlib.pyplot as plt

# Sample dataset
X = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
Y = np.array([2.0, 3.0, 4.0, 4.5, 5.0])

# LWR function
def locally_weighted_regression(x, X, Y, tau):
    m = len(X)
    y_pred = np.zeros_like(x)

    for i in range(len(x)):
        weights = np.exp(-(x[i] - X)**2 / (2 * tau**2))
        W = np.diag(weights)
        X_2D = X.reshape(-1, 1)  # Make X 2D
        Y_2D = Y.reshape(-1, 1)  # Make Y 2D
        theta = np.linalg.inv(X_2D.T @ W @ X_2D) @ X_2D.T @ W @ Y_2D
        y_pred[i] = x[i] * theta[0]

    return y_pred

# Set bandwidth (tau)
tau = 0.5

# Predict using LWR
x_pred = np.linspace(0, 6, 100)  # Range of x values for prediction
y_pred = locally_weighted_regression(x_pred, X, Y, tau)

# Plot the results
plt.scatter(X, Y, label="Data points")
plt.plot(x_pred, y_pred, color='red', label="LWR Fit")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.title("Locally Weighted Regression")
plt.show()
```

output-


Locally Weighted Regression
Data points
LWR Fit
Y
X
0
1
2
3
4
5
6

Experiment-11:

Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

dataset-

A	B	C	D	E	F	G	H	I	J	K
feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	target
1.764052346	0.400157	0.978738	2.240893	1.867558	-0.97728	0.950088	-0.15136	-0.10322	0.410599	1
0.144043571	1.454274	0.761038	0.121675	0.443863	0.333674	1.494079	-0.20516	0.313068	-0.8541	1
-2.552989816	0.653619	0.864436	-0.74217	2.269755	-1.45437	0.045759	-0.18718	1.532779	1.469359	1
0.154947426	0.378163	-0.88779	-1.9808	-0.34791	0.156349	1.230291	1.20238	-0.38733	-0.3023	1
-1.048552965	-1.42002	-1.70627	1.950775	-0.50965	-0.43807	-1.2528	0.77749	-1.6139	-0.21274	0
-0.895466561	0.386902	-0.51081	-1.18063	-0.02818	0.428332	0.066517	0.302472	-0.63432	-0.36274	1
-0.672460448	-0.35955	-0.81315	-1.72628	0.177426	-0.40178	-1.6302	0.462782	-0.9073	0.051945	0
0.729090562	0.128983	1.139401	-1.23483	0.402342	-0.68481	-0.8708	-0.57885	-0.31155	0.056165	1
-1.165149841	0.900826	0.465662	-1.53624	1.488252	1.895889	1.17878	-0.17992	-1.07075	1.054452	0
-0.403178947	1.222445	0.208275	0.976639	0.356366	0.706573	0.0105	1.78587	0.126912	0.401989	0
1.883150697	-1.34776	-1.27048	0.969397	-1.17312	1.943621	-0.41362	-0.74745	1.922942	1.480515	0
1.86755896	0.906045	-0.86123	1.910065	-0.268	0.802456	0.947252	-0.15501	0.614079	0.922207	0
0.376425531	-1.0994	0.298238	1.326386	-0.69457	-0.14963	-0.43515	1.849264	0.672295	0.407462	1
-0.769916074	0.539249	-0.67433	0.031831	-0.63585	0.676433	0.576591	-0.2083	0.396007	-1.09306	1
-1.491257593	0.439392	0.166673	0.635031	2.383145	0.944479	-0.91282	1.117016	-1.31591	-0.46158	1
-0.068241605	1.713343	-0.74475	-0.82644	-0.09845	-0.66348	1.126636	-1.07993	-1.14747	-0.43782	0
-0.498032451	1.929532	0.949421	0.087551	-1.22544	0.844363	-1.00022	-1.54477	1.18803	0.316943	0
0.920858824	0.318728	0.856831	-0.65103	-1.03424	0.681595	-0.80341	-0.68955	-0.45553	0.017479	1
-0.353993911	-1.37495	-0.64362	-2.2234	0.625231	-1.60206	-1.10438	0.052165	-0.73956	1.543015	1
-1.29285691	0.267051	-0.03928	-1.16809	0.523277	-0.17155	0.771791	0.823504	2.163236	1.336528	0
-0.369181838	-0.23938	1.09966	0.655264	0.640132	-1.61696	-0.02433	-0.73803	0.279925	-0.09815	1
0.910178908	0.317218	0.786328	-0.46642	-0.94445	-0.41005	-0.01702	0.379152	2.259309	-0.04226	1
-0.955945	-0.34598	-0.4636	0.481481	-1.5408	0.063262	0.156507	0.232181	-0.59732	-0.23792	0
-1.424060909	-0.49332	-0.54286	0.41605	-1.15618	0.781198	1.494485	-2.06999	0.426259	0.676908	0
-0.637437026	-0.39727	-0.13288	-0.29779	-0.30901	-1.676	1.152332	1.079619	-0.81336	-1.46642	1
0.521064876	-0.57579	0.141953	-0.31933	0.691539	0.694749	-0.7256	-1.38336	-1.58294	0.610379	0
-1.188859258	-0.50682	-0.59631	-0.05257	-1.93628	0.188779	0.523891	0.088422	-0.31089	0.0974	0
0.399046346	-2.77259	1.955912	0.390093	-0.65241	-0.39095	0.493742	-0.1161	-2.03068	2.064493	0
-0.110540657	1.020173	-0.69205	1.536377	0.286344	0.608844	-1.04525	1.211145	0.689818	1.301846	1
-0.62808756	-0.48103	2.303917	-1.06002	-0.13595	1.136891	0.097725	0.582954	-0.39945	0.370056	0
-1.306526852	1.658131	-0.11816	-0.68018	0.666383	-0.46072	-1.33426	-1.34672	0.693773	-0.15957	1

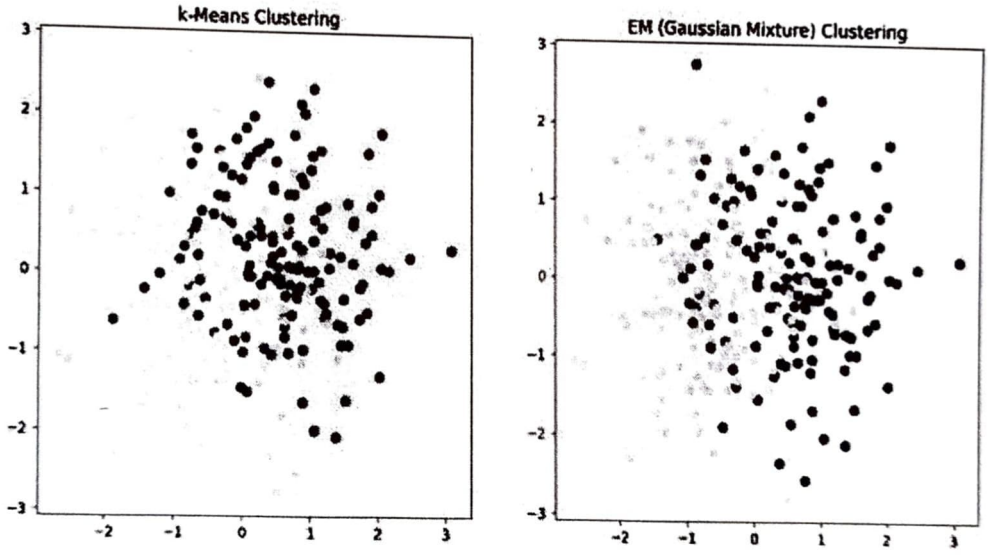
program-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler

# Load the Heart Disease dataset (you can replace this with your
dataset)
data = pd.read_csv('heart_disease.csv')
```

```
# Preprocess the data (you may need to adapt this depending on
your dataset)
X = data.drop('target', axis=1) # Features
y = data['target']          # Target variable

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply k-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# Apply EM clustering
gmm = GaussianMixture(n_components=2, random_state=42)
gmm_labels = gmm.fit_predict(X_scaled)

# Compare the results
# You can use various clustering evaluation metrics here, like
silhouette_score, adjusted_rand_score, etc.
from sklearn.metrics import silhouette_score,
adjusted_rand_score

kmeans_silhouette = silhouette_score(X_scaled, kmeans_labels)
gmm_silhouette = silhouette_score(X_scaled, gmm_labels)

kmeans_ari = adjusted_rand_score(y, kmeans_labels)
gmm_ari = adjusted_rand_score(y, gmm_labels)

print("k-Means Silhouette Score:", kmeans_silhouette)
print("EM (Gaussian Mixture) Silhouette Score:", gmm_silhouette)

print("k-Means Adjusted Rand Index:", kmeans_ari)
print("EM (Gaussian Mixture) Adjusted Rand Index:", gmm_ari)

# Plot the results (you can adapt this based on your dataset's
dimensionality)
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans_labels,
cmap='viridis')
plt.title("k-Means Clustering")

plt.subplot(1, 2, 2)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=gmm_labels,
cmap='viridis')
plt.title("EM (Gaussian Mixture) Clustering")

plt.show()
```

oputput-



Experiment-15:

Write a program to Implement Principle Component Analysis

program-

```
import numpy as np

# Generate a sample dataset (replace this with your own data)
np.random.seed(0)
data = np.random.rand(100, 3)  # 100 samples with 3 features

# Step 1: Standardize the data (mean=0, variance=1)
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)
standardized_data = (data - mean) / std_dev

# Step 2: Calculate the covariance matrix
cov_matrix = np.cov(standardized_data, rowvar=False)

# Step 3: Compute the eigenvalues and eigenvectors of the
covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 4: Sort eigenvalues and corresponding eigenvectors in
descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

# Step 5: Select the top k eigenvectors to retain k principal
components
k = 2  # Choose the number of principal components
top_k_eigenvectors = eigenvectors[:, :k]

# Step 6: Project the data onto the new feature space
pca_data = standardized_data.dot(top_k_eigenvectors)

# Now 'pca_data' contains the data in the reduced feature space
with k principal components
```

```
# Optional: Visualize the data

import matplotlib.pyplot as plt

plt.scatter(pca_data[:, 0], pca_data[:, 1])
plt.title('PCA of Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

output-