BMS COLLEGE OF ENGINEERING

(Autonomous College under VTU)

Bull Temple Road, Basavanagudi, Bangalore - 560019

A lab report on

"Algorithms & AI Laboratory"

[MCSL106]

Submitted in partial fulfillment of the requirements for the award of degree

MASTER OF TECHNOLOGY

IN

COMPUTER NETWORK ENGINEERING

By

PRAJWAL V
24TMPSCN07

Under the guidance of

Dr SHEELA S V
Professor

Department of Information Science and Engineering

2024-2025

BMS COLLEGE OF ENGINEERING

(Autonomous College under VTU)
Bull Temple Road, Basavanagudi,
Bangalore - 560019

Department of Information Science and Engineering

CERTIFICATE

This is to certify that the laboratory report entitled " **Algorithms & AI Laboratory**" **[MCSL106]** is a bona-fide work carried out by **Prajwal V (24TMPSCN07)** in partial fulfilment for the award of degree of Master of technology in **Computer Networking Engineering** from **Visvesvaraya Technological University**, **Belgaum** during the year **2024-2025**.

Dr Sheela S V	**Dr M K Nalini**	**Dr Bheemsha Arya**
Professor	**Professor and HOD**	**Principal**

Examiners

Name of the Examiner	**Signature of the Examiner**
1.	
2.	

INDEX

1) Implement Naive Bayes models and Bayesian networks. (Demonstrate the diagnosis of heart patients using standard heart disease data set etc)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import networkx as nx

# Manually entering a small dataset
data = {
   "age": [29, 45, 34, 60, 50, 41, 52, 39, 48, 59],
   "cholesterol": [200, 240, 210, 280, 260, 230, 300, 220, 250, 270],
   "bp": [120, 140, 130, 150, 140, 135, 160, 125, 145, 155],
   "heart_disease": [0, 1, 0, 1, 1, 0, 1, 0, 1, 1]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Features and target
X = df[["age", "cholesterol", "bp"]]
y = df["heart_disease"]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Predictions
y_pred = nb_model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Visualization of Results
plt.scatter(df["cholesterol"], df["bp"], c=df["heart_disease"], cmap="coolwarm", label="Data
Points")
plt.xlabel("Cholesterol")
plt.ylabel("Blood Pressure")
plt.title("Heart Disease Diagnosis")
plt.legend(["No Disease", "Disease"], loc="best")
plt.show()
# Bayesian Network Visualization
G = nx.DiGraph()
```

```
G.add_edges_from([
    ("age", "heart_disease"),
    ("cholesterol", "heart_disease"),
    ("bp", "heart_disease")
])

plt.figure(figsize=(8, 6))
nx.draw(G, with_labels=True, node_size=3000, node_color="lightblue", font_size=10,
font_weight="bold")
plt.title("Bayesian Network for Heart Disease Diagnosis")
plt.show()
```

Output:

```
PS C:\Users\Admin\Desktop\mtech\ai\code> python -u "c:\Users\Admin\Desktop\mtech\ai\code\AI_lab_exp1.py"
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
Sample instances from the dataset are given below:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal  target
0   52    1   0       125   212    0        1      168      0      1.0      2   2     3       0
1   53    1   0       140   203    1        0      155      1      3.1      0   0     3       0
2   70    1   0       145   174    0        1      125      1      2.6      0   0     3       0
3   61    1   0       148   203    0        1      161      0      0.0      2   1     3       0
4   62    0   0       138   294    1        1      106      0      1.9      1   3     2       0

Attributes and datatypes:
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

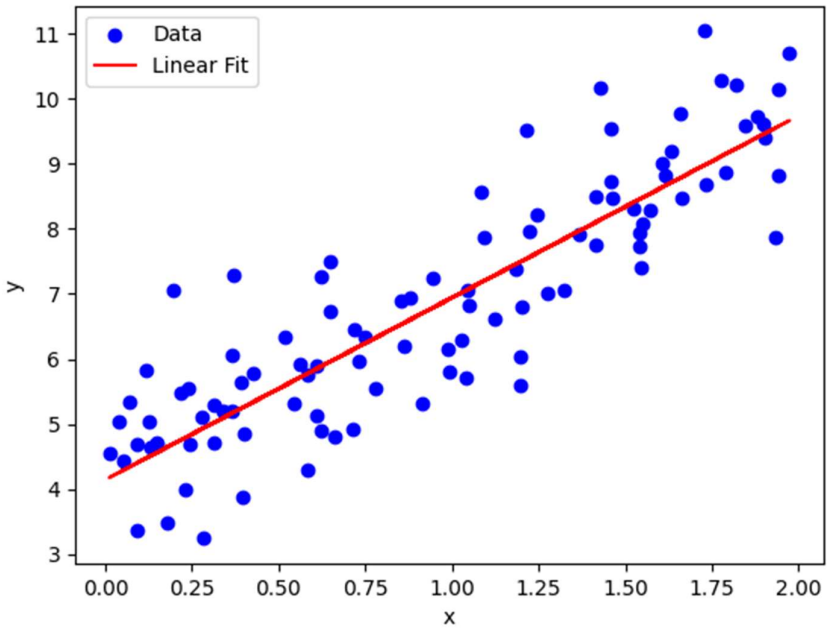
```
Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence = restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.4354 |
+-----------------+---------------------+
| heartdisease(1) |              0.5646 |
+-----------------+---------------------+

2. Probability of HeartDisease given evidence = cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3832 |
+-----------------+---------------------+
| heartdisease(1) |              0.6168 |
+-----------------+---------------------+
PS C:\Users\Admin\Desktop\mtech\ai\code>
```

2) Implement a simple linear regression algorithm to predict a continuous target variable based on a given dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

class SimpleLinearRegression:
    def __init__(self):
        self.slope=0
        self.intercept=0

    def fit(self,x,y):
        x_mean=np.mean(x)   #calc slope and intercept using formula
        y_mean=np.mean(y)

        num=np.sum((x-x_mean)*(y-y_mean))
        deno=np.sum((x-x_mean)**2)

        self.slope=num/deno
        self.intercept=y_mean-self.slope*x_mean

    def predict(self,x):
        return self.slope*x+self.intercept  #predict target variable

    def evaluate(self,x,y):
        y_pred=self.predict(x)  #evaluate model using mean squared error
        mse=np.mean((y-y_pred)**2)
        return mse

#generate a synthetic dataset for demo
np.random.seed(42)
x=2*np.random.rand(100,1).flatten()
y=4+3*x+np.random.randn(100)

#split dataset into training and testing sets
x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2, random_state=42)

#train and evaluate the model
model=SimpleLinearRegression()
model.fit(x_train, y_train)

print(f"slope: {model.slope}")
print(f"Intercept: {model.intercept}")

mse =model.evaluate(x_test, y_test)
print(f"Mean Squared Error on Test Set:{mse}")

#plot the results
plt.scatter(x,y,color='blue',label='Data')
```

```
plt.plot(x, model.predict(x), color='red', label='Linear Fit')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

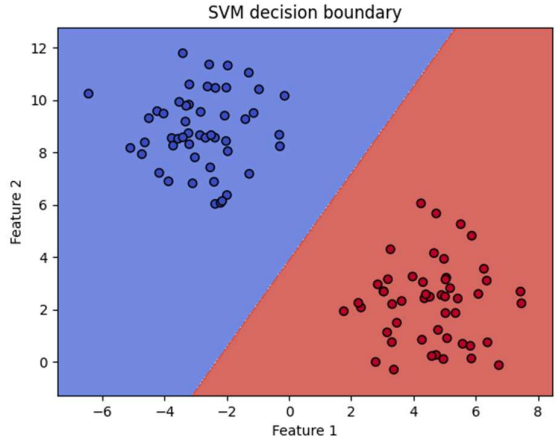
Output:



3) Develop a program to implement a Support Vector Machine for binary classification. Use a sample dataset and visualize the decision boundary.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.svm import SVC

# Generate a synthetic dataset for binary classification
x, y = make_blobs(n_samples=100, centers=2, random_state=42, cluster_std=1.5)

# Train a support vector machine model
model = SVC(kernel='linear')
model.fit(x, y)

# Define a function to plot the decision boundary
def plot_decision_boundary(x, y, model):
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    z = z.reshape(xx.shape)

    plt.contourf(xx, yy, z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', cmap=plt.cm.coolwarm)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('SVM decision boundary')
    plt.show()  # Add parentheses here to display the plot

# Plot the decision boundary
plot_decision_boundary(x, y, model)
```

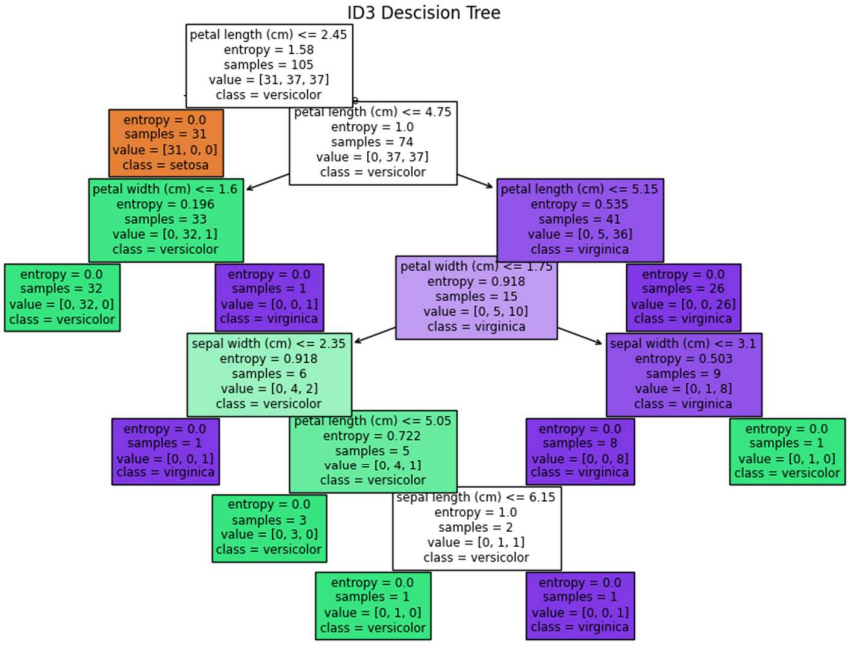
Output:



4) Write a program to demonstrate the ID3 decision tree algorithm using an appropriate dataset for classification

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

data= load_iris()
x=data.data #featurematrix
y=data.target

#split 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)

id3_tree=DecisionTreeClassifier(criterion='entropy', random_state=42)

id3_tree.fit(x_train, y_train)

y_pred=id3_tree.predict(x_test)

accuracy=accuracy_score(y_test, y_pred)
print(f'Accuracy of the ID# Decision tree:{accuracy*100: .2f}%')

print('\nDecision Tree Rules:')
print(export_text(id3_tree,feature_names=data.feature_names))


plt.figure(figsize=(12,8))
plot_tree(id3_tree, feature_names=data.feature_names, class_names=list(data.target_names),
filled=True)
plt.title("ID3 Descision Tree")
plt.show()
```

Output:




```
Decision Tree Rules:
|--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.60
|   |   |   |--- class: 1
|   |   |--- petal width (cm) >  1.60
|   |   |   |--- class: 2
|   |--- petal length (cm) >  4.75
|   |   |--- petal length (cm) <= 5.15
|   |   |   |--- petal width (cm) <= 1.75
|   |   |   |   |--- sepal width (cm) <= 2.35
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  2.35
|   |   |   |   |   |--- petal length (cm) <= 5.05
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- petal length (cm) >  5.05
|   |   |   |   |   |   |--- sepal length (cm) <= 6.15
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- sepal length (cm) >  6.15
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |--- petal width (cm) >  1.75
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  5.15
|   |   |   |--- class: 2

PS C:\Users\Admin\Desktop\Prajwal\PG-1st sem\AI\Lab_prgs>
```

5) Implement a KNN algorithm for regression tasks instead of classification. Use a small dataset, and predict continuous values based on the average of the nearest neighbours.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

data=np.array([[1200, 200], [1500, 250], [1700, 200], [2100, 400],
               [2300, 450], [2500, 500]])

x=data[:,0].reshape(-1, 1)
y=data[:,1]

x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3, random_state=42)

def knn_regression(x_train, y_train, x_test, k=3):
    predictions=[]

    for test_point in x_test:
        distances= np.sqrt(np.sum((x_train- test_point)**2, axis=1))

        nearest_indices=np.argsort(distances)[:k]

        nearest_value=y_train[nearest_indices]
        prediction=np.mean(nearest_value)
        predictions.append(prediction)

    return np.array(predictions)

y_pred=knn_regression(x_train, y_train, x_test, k=3)

mse=mean_squared_error(y_test, y_pred)
print(f"Mean squared error:{mse:.2f}")

print("\nTest Predictions:")
for i,(size, actual, pred) in enumerate(zip(x_test.flatten(),y_test, y_pred)):
    print(f"House Size:{size}, Actual price: {actual}, Predicted proce:{pred:.2f}")
```

Output:

```
Mean squared error:16250.00

Test Predictions:
House Size:1200, Actual price: 200, Predicted proce:350.00
House Size:1500, Actual price: 250, Predicted proce:350.00
PS C:\Users\Admin\Desktop\Prajwal\PG-1st sem\AI\Lab_prgs>
```

6) Implement the k-Nearest Neighbour algorithm to classify the Iris dataset, printing both correct and incorrect predictions.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from collections import Counter

def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1-point2)**2))

# k-nearest neighbour
def knn_classify(x_train, y_train, x_test, k=3):
    predictions = []

    for test_point in x_test:
        distance = [euclidean_distance(test_point, train_point) for train_point in x_train]

        nearest_indices = np.argsort(distance)[:k]
        nearest_labels = [y_train[i] for i in nearest_indices]

        most_common = Counter(nearest_labels).most_common(1)[0][0]
        predictions.append(most_common)

    return predictions

# Load the iris dataset
data = load_iris()
x = data.data
y = data.target

# Split dataset into training and testing set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

k = 3
predictions = knn_classify(x_train, y_train, x_test, k=k)

print("\nPrediction Results:")
correct = 0
incorrect = 0

for i, (pred, actual) in enumerate(zip(predictions, y_test)):
    if pred == actual:
        correct += 1
        print(f'Test Sample {i+1}: Correct (Predicted: {pred}, Actual: {actual})")
    else:
        incorrect += 1
        print(f'Test Sample {i+1}: Incorrect (Predicted: {pred}, Actual: {actual})")
print(f"\nTotal Correct Predictions: {correct}")
print(f'Total Incorrect Predictions: {incorrect}")
```

print(f"Accuracy: {correct / len(y_test) * 100:.2f}%")

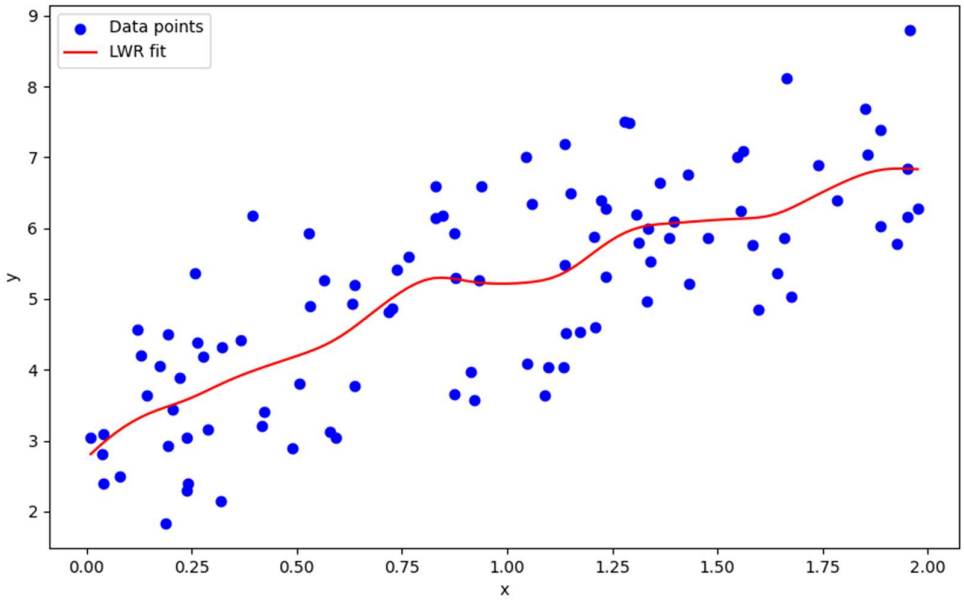
Output:

```
Prediction Results:
Test Sample 1: Correct (Predicted: 1, Actual: 1)
Test Sample 2: Correct (Predicted: 0, Actual: 0)
Test Sample 3: Correct (Predicted: 2, Actual: 2)
Test Sample 4: Correct (Predicted: 1, Actual: 1)
Test Sample 5: Correct (Predicted: 1, Actual: 1)
Test Sample 6: Correct (Predicted: 0, Actual: 0)
Test Sample 7: Correct (Predicted: 1, Actual: 1)
Test Sample 8: Correct (Predicted: 2, Actual: 2)
Test Sample 9: Correct (Predicted: 1, Actual: 1)
Test Sample 10: Correct (Predicted: 1, Actual: 1)
Test Sample 11: Correct (Predicted: 2, Actual: 2)
Test Sample 12: Correct (Predicted: 0, Actual: 0)
Test Sample 13: Correct (Predicted: 0, Actual: 0)
Test Sample 14: Correct (Predicted: 0, Actual: 0)
Test Sample 15: Correct (Predicted: 0, Actual: 0)
Test Sample 16: Correct (Predicted: 1, Actual: 1)
Test Sample 17: Correct (Predicted: 2, Actual: 2)
Test Sample 18: Correct (Predicted: 1, Actual: 1)
Test Sample 19: Correct (Predicted: 1, Actual: 1)
Test Sample 20: Correct (Predicted: 2, Actual: 2)
Test Sample 21: Correct (Predicted: 0, Actual: 0)
Test Sample 22: Correct (Predicted: 2, Actual: 2)
Test Sample 23: Correct (Predicted: 0, Actual: 0)
Test Sample 24: Correct (Predicted: 2, Actual: 2)
Test Sample 25: Correct (Predicted: 2, Actual: 2)
Test Sample 26: Correct (Predicted: 2, Actual: 2)
Test Sample 27: Correct (Predicted: 2, Actual: 2)
Test Sample 28: Correct (Predicted: 2, Actual: 2)
Test Sample 29: Correct (Predicted: 0, Actual: 0)
Test Sample 30: Correct (Predicted: 0, Actual: 0)
Test Sample 31: Correct (Predicted: 0, Actual: 0)
Test Sample 32: Correct (Predicted: 0, Actual: 0)
Test Sample 33: Correct (Predicted: 1, Actual: 1)
Test Sample 34: Correct (Predicted: 0, Actual: 0)
Test Sample 35: Correct (Predicted: 0, Actual: 0)
Test Sample 36: Correct (Predicted: 2, Actual: 2)
Test Sample 37: Correct (Predicted: 1, Actual: 1)
Test Sample 38: Correct (Predicted: 0, Actual: 0)
Test Sample 39: Correct (Predicted: 0, Actual: 0)
Test Sample 40: Correct (Predicted: 0, Actual: 0)
Test Sample 41: Correct (Predicted: 2, Actual: 2)
Test Sample 42: Correct (Predicted: 1, Actual: 1)
Test Sample 43: Correct (Predicted: 1, Actual: 1)
Test Sample 44: Correct (Predicted: 0, Actual: 0)
Test Sample 45: Correct (Predicted: 0, Actual: 0)

Total Correct Predictions: 45
Total Incorrect Predictions: 0
Accuracy: 100.00%
PS C:\Users\Admin\Desktop\Prajwal\PG-1st sem\AI\Lab_prgs>
```

7) Develop a program to implement the non-parametric Locally Weighted Regression algorithm, fitting data points and visualizing results.

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

def locally_weighted_regression(x,y, tau, x_query):
   m= len(x)
   w=np.exp( -np.sum((x-x_query)**2, axis=1)/(2*tau**2))
   w=np.diag(w)

   x_b= np.c_[np.ones((m,1)),x]
   theta= np.linalg.pinv(x_b.T @ w @ x_b) @x_b.T @ w @y
   x_query_b= np.array([1, x_query])
   return x_query_b @ theta

np.random.seed(0)
x=2*np.random.rand(100,1)
y=(3+ 2*x +np.random.randn(100,1)).flatten()

x=x.flatten()

sort_idx= np.argsort(x)
x=x[sort_idx]
y=y[sort_idx]

tau=0.1
x_test= np.linspace(x.min(), x.max(), 100)

#perform predictions
y_pred =[locally_weighted_regression(x[:, np.newaxis], y, tau, x_query) for x_query in
x_test]

#visualization
plt.figure(figsize=(10,6))
plt.scatter(x,y,color='blue',label='Data points')
plt.plot(x_test, y_pred, color='red', label='LWR fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Output:

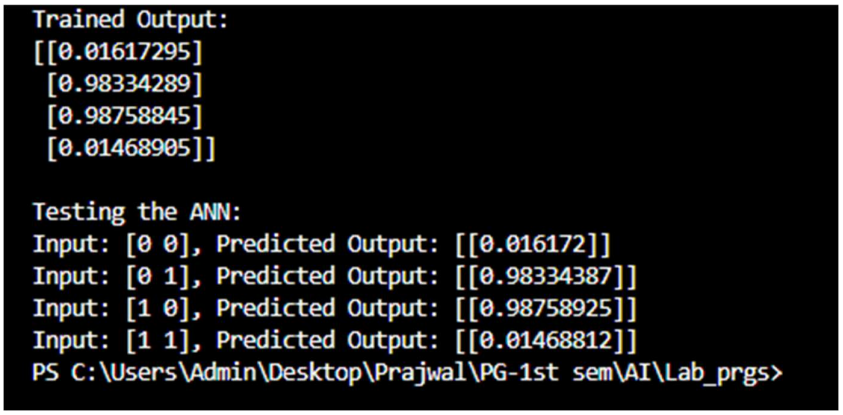

Data points
LWR fit
y
x
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2
3
4
5
6
7
8
9

8) Build an Artificial NeuralNetwork by implementing the Backpropagation algorithm and test it with suitable datasets.

```
import numpy as np

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

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

# Define input data and expected output
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])  # XOR output

np.random.seed(42)

# Set the architecture of the neural network
input_layer_neurons = X.shape[1]  # 2 inputs
hidden_layer_neurons = 4
output_neurons = 1

# Initialize weights and biases
weights_input_hidden = np.random.uniform(size=(input_layer_neurons,
hidden_layer_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))

learning_rate = 0.5

# Training the neural network
for epoch in range(10000):
    # Forward pass
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)

    # Compute the error
    error = y - predicted_output

    # Backpropagation
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_hidden_output += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
```

```
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

# Display the trained output
print("Trained Output:")
print(predicted_output)

# Testing the trained neural network
print("\nTesting the ANN:")
for i, test_input in enumerate(X):
    hidden_layer_input = np.dot(test_input, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)
    print(f"Input: {test_input}, Predicted Output: {predicted_output}")
```

Output:



9) Implement a Q-learning algorithm to navigate a simple grid environment, defining the reward structure and analysing agent performance.

```
import numpy as np

grid_size=(5,5)
num_states= grid_size[0]* grid_size[1]
actions= ["up","down","left","right"]
num_actions= len(actions)

def state_to_coordinates(state):
    return divmod(state, grid_size[1])

def coordinates_to_state(x,y):
    return x * grid_size[1] + y

rewards= np.full(num_states, -1)
terminal_states= coordinates_to_state(4,4)
rewards[terminal_states]= 100

def take_action(state, action):
    x,y= state_to_coordinates(state)
    if action =="up":
        x= max(0, x-1)
    elif action =="down":
        x= min(grid_size[0]-1, x+1)
    elif action =="left":
        y= max(0, y-1)
    elif action =="right":
        y= min(grid_size[1]-1, y+1)

    return coordinates_to_state(x,y)

q_table= np.zeros((num_states, num_actions))
alpha= 0.1
gamma= 0.9
epsilon= 0.2
num_episodes= 500

for epsiode in range(num_episodes):
    state=np.random.randint(0, num_states)

    while state!= terminal_states:
        if np.random.rand()< epsilon:
            action= np.random.randint(0, num_actions)

        else:
            action= np.argmax(q_table[state])
        next_state= take_action(state, actions[action])
        reward= rewards[next_state]
```

```
        best_next_action= np.max(q_table[next_state])
        q_table[state, action] += alpha*(reward+gamma+best_next_action-q_table[state,
action])
        state= next_state

policy= np.argmax(q_table, axis=1)
policy_grid= np.array([actions[a] for a in policy]).reshape(grid_size)

print("Optimal Policy")
print(policy_grid)
print("\nQ-Table")
print(q_table)
```

Output:

```
Optimal Policy
[['right' 'down' 'down' 'right' 'down']
 ['right' 'down' 'left' 'right' 'down']
 ['right' 'down' 'down' 'down' 'down']
 ['right' 'right' 'down' 'down' 'down']
 ['right' 'right' 'right' 'right' 'up']]

Q-Table
[[-8.70290000e-02 -7.79707352e-02 -8.00000000e-02  6.37917304e+01]
 [ 8.47703336e+00  9.40414239e+01  1.14834241e+01  9.31289604e+00]
 [ 3.60399761e+00  5.86180778e+01  1.30134810e+01  7.56175863e-02]
 [ 7.61795696e+00  9.47769289e+00  2.08166181e+00  5.00975521e+01]
 [ 1.01715758e+01  9.26059970e+01  6.26407484e+00  8.36226900e+00]
 [ 7.48188073e+00  2.67893908e+01  7.47155860e+00  7.54365860e+01]
 [ 1.77062463e+01  1.00341527e+02  2.71327084e+01  3.28866872e+01]
 [ 5.42148659e+00  2.23786792e+01  9.40079590e+01  1.78275096e+01]
 [-3.91900000e-02  1.65917422e+01  9.21708587e+00  8.13597021e+01]
 [ 2.29260221e+01  1.00222707e+02  1.20750168e+01  4.70705418e+01]
 [-5.01701000e-02 -4.80000000e-02  1.61818314e+01  9.57144133e+01]
 [ 9.20783640e+00  1.00499676e+02  3.61092969e+01  4.81955064e+01]
 [-4.99198000e-02  9.73230894e+01  4.24287203e+01  6.12229784e+00]
 [ 1.40675849e+01  9.58459270e+01 -3.17100000e-02  1.42287526e+01]
 [ 5.86141567e+01  1.00739937e+02  3.08668552e+01  4.40936208e+01]
 [-3.91000000e-02 -3.81000000e-02  1.43662402e+01  9.56219118e+01]
 [ 7.14073941e+01  6.76483894e+01  5.13190720e+01  1.00599995e+02]
 [ 5.35694085e+01  1.00700000e+02  7.39001065e+01  6.90412593e+01]
 [ 2.88285839e+01  1.00796558e+02  4.81759951e+01  4.39510764e+01]
 [ 4.24364899e+01  1.00895915e+02  4.44713313e+01  2.62937966e+01]
 [ 7.86533968e+00 -4.00000000e-02 -4.00000000e-02  8.86568233e+01]
 [ 3.46459516e+01  2.69143945e+01  2.43546906e+01  1.00567305e+02]
 [ 8.83521877e+01  7.64152855e+01  8.11980577e+01  1.00800000e+02]
 [ 9.33965070e+01  9.43325791e+01  8.89886857e+01  1.00900000e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

10) Write a python program

a. **to perform tokenization by word and sentence using nltk.**
b. **to eliminate stop words using nltk.**
c. **to perform stemming using nltk.**
d. **to perform Parts of Speech tagging using nltk.**

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk import pos_tag

# Sample text
text = "NLTK is a leading platform for building Python programs to work with human
language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such
as WordNet."

# a. Tokenization by word and sentence using nltk
# Sentence tokenization
sentences = sent_tokenize(text)
print("Sentence Tokenization:")
print(sentences)

# Word tokenization
words = word_tokenize(text)
print("\nWord Tokenization:")
print(words)

# b. Eliminate stop words using nltk
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word.lower() not in stop_words]
print("\nFiltered Words (Stop Words Removed):")
print(filtered_words)

# c. Perform stemming using nltk
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_words]
print("\nStemmed Words:")
print(stemmed_words)

# d. Perform Parts of Speech tagging using nltk
pos_tags = pos_tag(words)
print("\nPart of Speech Tagging (Original Words):")
print(pos_tags)
```

```
# If you want POS tagging after stopword removal, use:
filtered_pos_tags = pos_tag(filtered_words)
print("\nPart of Speech Tagging (After Stopword Removal):")
print(filtered_pos_tags)
```

Output:

```
Sentence Tokenization:
['NLTK is a leading platform for building Python programs to work with human language data.', 'It provides easy-to-use inter
faces to over 50 corpora and lexical resources such as WordNet.']

Word Tokenization:
['NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'programs', 'to', 'work', 'with', 'human', 'language
', 'data', '.', 'It', 'provides', 'easy-to-use', 'interfaces', 'to', 'over', '50', 'corpora', 'and', 'lexical', 'resources',
 'such', 'as', 'WordNet', '.']

Filtered Words (Stop Words Removed):
['NLTK', 'leading', 'platform', 'building', 'Python', 'programs', 'work', 'human', 'language', 'data', '.', 'provides', 'eas
y-to-use', 'interfaces', '50', 'corpora', 'lexical', 'resources', 'WordNet', '.']

Stemmed Words:
['nltk', 'lead', 'platform', 'build', 'python', 'program', 'work', 'human', 'languag', 'data', '.', 'provid', 'easy-to-us',
'interfac', '50', 'corpora', 'lexic', 'resourc', 'wordnet', '.']

Part of Speech Tagging:
[('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('leading', 'VBG'), ('platform', 'NN'), ('for', 'IN'), ('building', 'VBG'), ('
Python', 'NNP'), ('programs', 'NNS'), ('to', 'TO'), ('work', 'VB'), ('with', 'IN'), ('human', 'JJ'), ('language', 'NN'), ('d
ata', 'NNS'), ('.', '.'), ('It', 'PRP'), ('provides', 'VBZ'), ('easy-to-use', 'JJ'), ('interfaces', 'NNS'), ('to', 'TO'), ('
over', 'IN'), ('50', 'CD'), ('corpora', 'NNS'), ('and', 'CC'), ('lexical', 'JJ'), ('resources', 'NNS'), ('such', 'JJ'), ('as
', 'IN'), ('WordNet', 'NNP'), ('.', '.')]
PS C:\Users\Admin\Desktop\mtech\ai\code>
```