MALNAD COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi)

HASSAN-573202, KARNATAKA, INDIA

MASTER OF TECHNOLOGY (M.Tech)

In

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Laboratory Manual

Course: Artificial Intelligence Laboratory

Course Code: 25MCS16

Department of Computer Science and Engineering Malnad College of Engineering

2025-2027

1. Implement a simple linear regression algorithm to predict a continuous target variable based on a given dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
data=pd.read_csv('Experience-Salary.csv')
print(data.head())
```

```
   exp(in months)  salary(in thousands)
0       18.290293              16.521825
1       17.023407              11.666234
2       26.343613              23.167255
3       19.105834              20.877145
4       27.742516              23.166236
```

```
print(data.isnull().sum())
```

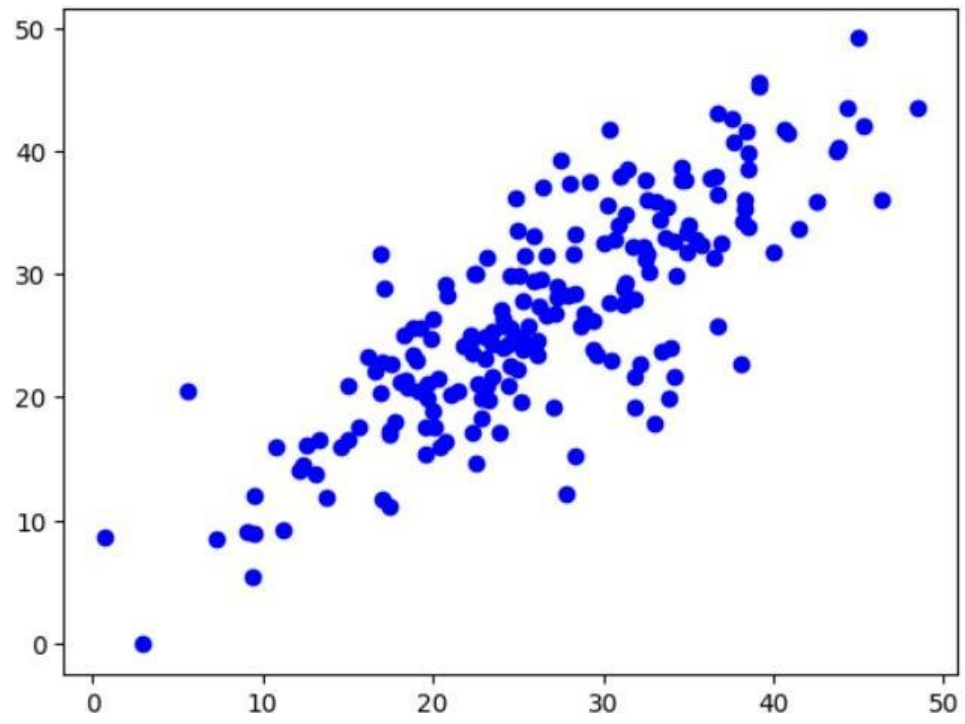
```
exp(in months)          0
salary(in thousands)    0
dtype: int64
```

```
data=data.dropna()
x = data[['exp(in months)']].values
y = data['salary(in thousands)'].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=43
)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
import pandas as pd
```

```
data = pd.DataFrame({
    'y_test': y_test,
    'y_pred': y_pred
})
print(data)
```

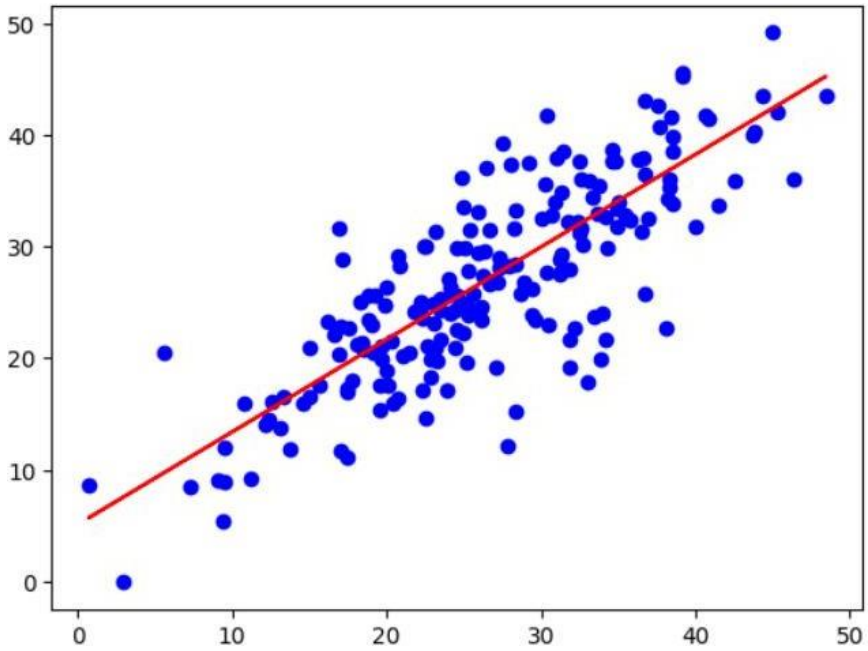
```
        y_test     y_pred
0    19.872404  33.184023
1    33.568944  25.797656
2    14.010704  15.180821
3    32.266118  31.398530
4    22.840362  19.247506
..         ...        ...
195  15.425008  21.351266
196  36.133632  25.677778
197  32.850665  34.452602
198  24.364536  24.570929
199  32.627963  33.463493
```

```
plt.scatter(x_test, y_test, color='blue', label='Actual')
```




```
plt.scatter(x_test, y_test, color='blue', label='Actual')
```

plt.plot(x_test, model.predict(x_test), color='red')



rmse = np.sqrt(mean_squared_error(y_test, y_pred))

r2 = r2_score(y_test, y_pred)

print(r2)

print(rmse)

```
0.6493660735045069
5.219513100434918
```

mse = mean_squared_error(y_test, y_pred)

print("MSE:", mse)

```
MSE: 27.243317005611733
```

from sklearn.metrics import mean_absolute_error

```
mae = mean_absolute_error(y_test, y_pred)
print("MAE:", mae)
```

```
MAE: 4.0235450266378985
```

```
n = len(y_test)
k = x_test.shape[1]

adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
print("Adjusted R2:", adj_r2)
```

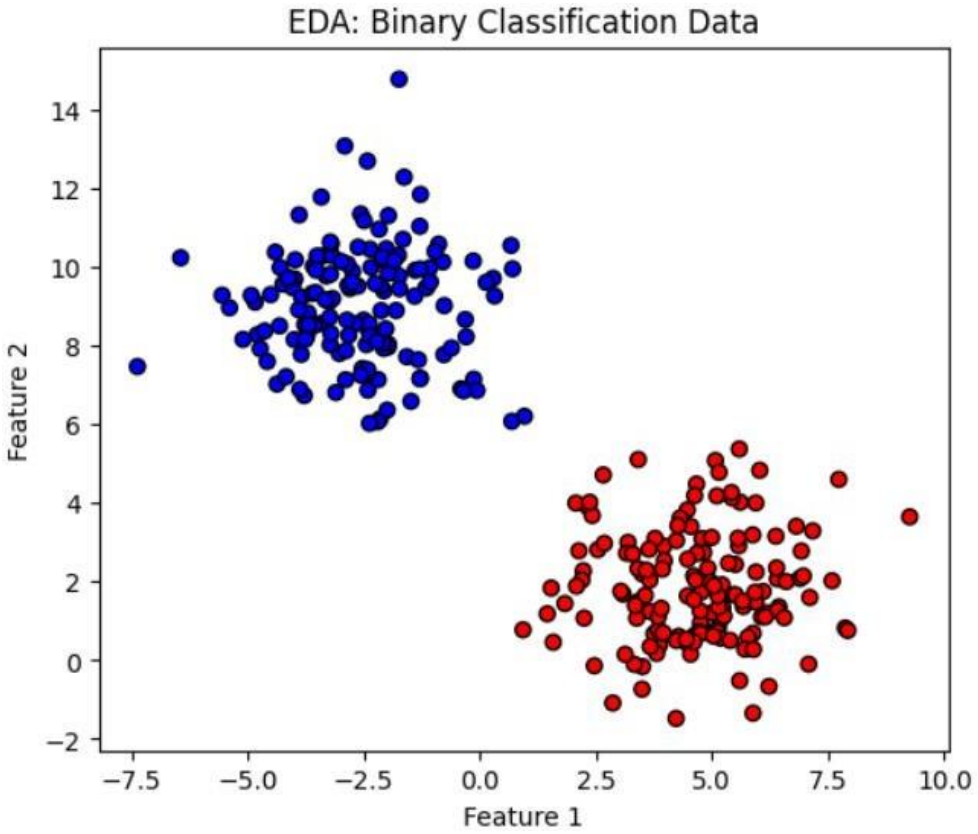
```
Adjusted R2: 0.647595195087863
```

```
experience = float(input("Enter the experience: "))
predicted_salary = model.predict([[experience]])

print("\nExperience entered:", experience)
print("Predicted Salary:", predicted_salary[0])
```

```
Enter the experience: 60

Experience entered: 60.0
Predicted Salary: 54.833914057811086
```

2.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
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
X, y = make_blobs(
    n_samples=300,
    centers=2,
    cluster_std=1.5,
    random_state=42
)
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
plt.figure(figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("EDA: Binary Classification Data")
plt.show()
```




```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.25, random_state=42)

svm_model = SVC(kernel='linear', C=1.0)

svm_model.fit(X_train, y_train)

def plot_decision_boundary(model, X, y):
    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.linspace(x_min, x_max, 200),
        np.linspace(y_min, y_max, 200)
    )
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(6, 5))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
```

```
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolors='k')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("SVM Decision Boundary")
    plt.show()
plot_decision_boundary(svm_model, X, y)
y_pred = svm_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0

Confusion Matrix:
 [[35  0]
 [ 0 40]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        35
           1       1.00      1.00      1.00        40

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75
```

3.Develop a simple case-based reasoning system that stores instances of past cases. Implement a retrieval method to find the most similar cases and make predictions based on them.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
data=pd.read_csv('cbr_classification_dataset.csv')
print(data.head())
print("\nMissing Values:")
print(data.isnull().sum())
data=data.dropna()
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
print("X shape:", X.shape)
print("y shape:", y.shape)
scaler = StandardScaler()
x_scaled = scaler.fit_transform(X)
np.array([[5,6]])
new_case=np.array([[5,6]])
new_case_scaled = scaler.transform(new_case)
new_case_scaled
from sklearn.metrics.pairwise import euclidean_distances
distances = euclidean_distances(new_case_scaled, x_scaled)[0]
print(distances)
data['distance']=distances
```

```
print(data)
```

```
   Feature1  Feature2    Label  distance
0       1.0       2.0  Class A  2.432976
1       2.0       3.0  Class B  1.824732
2       3.0       3.0  Class A  1.558423
3       6.0       6.0  Class B  0.424492
4       4.5       5.5  Class B  0.304122
5       1.5       4.2  Class A  1.679945
6       2.2       2.5  Class B  1.933228
7       3.4       9.8  Class A  1.789285
8       8.5       4.6  Class B  1.606023
9       6.7       7.2  Class A  0.891080
```

k=5

similar_data=data.sort_values(by='distance').head(k)

print(similar_data)

```
    Feature1  Feature2    Label  distance
4       4.5       5.5  Class B  0.304122
3       6.0       6.0  Class B  0.424492
9       6.7       7.2  Class A  0.891080
2       3.0       3.0  Class A  1.558423
8       8.5       4.6  Class B  1.606023
```

predicted=similar_data['Label'].mode()[0]

print(predicted)

```
Class B
```

```
print(predicted[0:1])
```

```
C
```

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

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
df = pd.read_csv("iris.csv")
print("\nMissing Values:")
print(df.isnull().sum())
label_encoder = LabelEncoder()
df["species_encoded"] = label_encoder.fit_transform(df["species"])
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df["species_encoded"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = DecisionTreeClassifier(
    criterion="entropy",
    max_depth=4,
    random_state=42
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
print(y_pred)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print(cm)
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
print("\nClassification Report:\n")

print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
```

```
Classification Report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
from sklearn.tree import export_text

tree_rules = export_text(model, feature_names=list(X.columns))

print(tree_rules)
```

```
|--- petal_length <= 2.45
|   |--- class: 0
|--- petal_length >  2.45
|   |--- petal_length <= 4.75
|   |   |--- petal_width <= 1.65
|   |   |   |--- class: 1
|   |   |--- petal_width >  1.65
|   |   |   |--- class: 2
|   |--- petal_length >  4.75
|   |   |--- petal_width <= 1.75
|   |   |   |--- petal_length <= 4.95
|   |   |   |   |--- class: 1
|   |   |   |--- petal_length >  4.95
|   |   |   |   |--- class: 2
|   |   |--- petal_width >  1.75
|   |   |   |--- petal_length <= 4.85
|   |   |   |   |--- class: 2
|   |   |   |--- petal_length >  4.85
|   |   |   |   |--- class: 2
```

5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test it with suitable datasets.

```
import numpy as np
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],
              [1],
              [1],
              [0]])
lr = 0.1
epochs = 10000
input_neurons = 2
hidden_neurons = 2
output_neurons = 1
np.random.seed(1)
W1 = np.random.randn(input_neurons, hidden_neurons)
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.randn(hidden_neurons, output_neurons)
b2 = np.zeros((1, output_neurons))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
for _ in range(epochs):
    hidden = sigmoid(np.dot(X, W1) + b1)
    output = sigmoid(np.dot(hidden, W2) + b2)
    error = y - output
    output_delta = error * output * (1 - output)
```

```
    hidden_delta = output_delta.dot(W2.T) * hidden * (1 - hidden)
    W2 += hidden.T.dot(output_delta) * lr
    b2 += np.sum(output_delta, axis=0, keepdims=True) * lr
    W1 += X.T.dot(hidden_delta) * lr
    b1 += np.sum(hidden_delta, axis=0, keepdims=True) * lr
print("Final Output:")
print(output.round(3))
print("\nPredicted XOR:")
print((output > 0.5).astype(int))
```

```
Final Output:
[[0.047]
 [0.945]
 [0.944]
 [0.074]]

Predicted XOR:
[[0]
 [1]
 [1]
 [0]]
```

6. 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

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

knn=KNeighborsRegressor(n_neighbors=3)

knn.fit(X_train,y_train)

y_pred=knn.predict(X_test)

sample_input=np.array([[1500,3,5]])     #Example squarefeet, bedroom , age

sample_input_scaled=scaler.transform(sample_input)

predicted_price=knn.predict(sample_input_scaled)

print("Predicted price (average of nearest Neighbours ) for
input",sample_input,"is",predicted_price[0])
```

```
Predicted price (average of nearest Neighbours ) for input [[1500    3    5]] is 499.6666666666667
```

```
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 20.933333333333326
Mean Squared Error: 1022.5333333333326
Root Mean Squared Error: 31.977075121613804
R² Score: 0.9178451189271981
```

7. Create a program that calculates different distance metrics (Euclidean and Manhattan) between two points in a dataset. Allow the user to input two points and display the calculated distances.

```
import math
x1, y1 = map(int, input('Enter the coordinates of point 1: ').split())
x2, y2 = map(int, input('Enter the coordinates of point 2: ').split())
def euclidean_distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
def manhattan_distance(x1, y1, x2, y2):
    return abs(x2 - x1) + abs(y2 - y1)
print("Euclidean  Distance:",  euclidean_distance(x1,y1,x2,y2))
print("Manhattan Distance:", manhattan_distance(x1,y1,x2,y2))
```

```
Enter the coordinates of point 1: 4 1
Enter the coordinates of point 2: 3 0
Euclidean Distance: 1.4142135623730951
Manhattan Distance: 2
```

8. Implement the k-Nearest Neighbor algorithm to classify the Iris dataset, printing both correct and incorrect predictions.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
data=pd.read_csv('iris.csv')
print("\nMissing Values:")
print(data.isnull().sum())
data=data.dropna()
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (150, 4)
y shape: (150,)
```

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

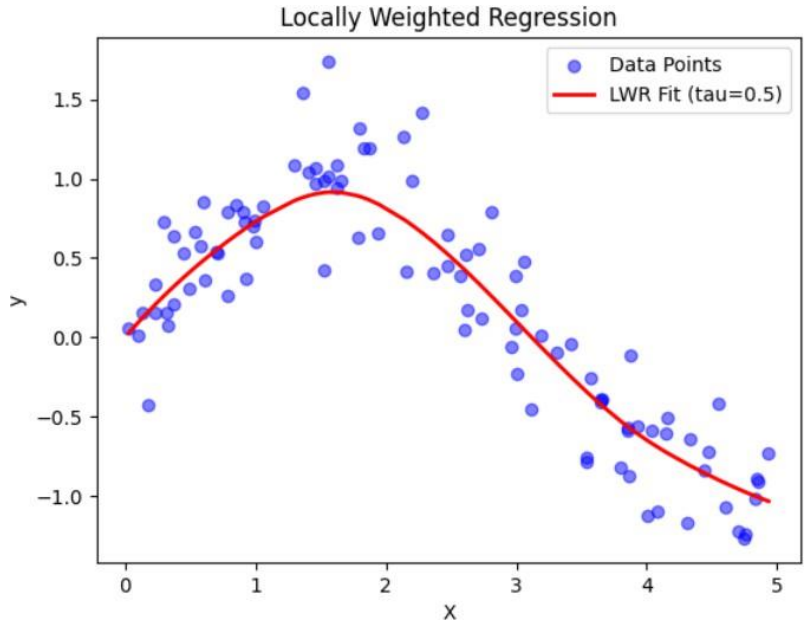
```
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

9.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 gaussian_kernel(x, xi, tau):
    return np.exp(-np.linalg.norm(x - xi) ** 2 / (2 * tau ** 2))
def locally_weighted_regression(X, y, tau):
    m = X.shape[0]
    predictions = np.zeros(m)
    X_augmented = np.hstack((np.ones((m, 1)), X))

    for i in range(m):
        W = np.diag([gaussian_kernel(X[i], X[j], tau) for j in range(m)])
        theta = np.linalg.pinv(X_augmented.T @ W @ X_augmented) @ \
                X_augmented.T @ W @ y
        predictions[i] = np.array([1, X[i][0]]) @ theta
    return predictions
np.random.seed(42)
X = np.sort(5 * np.random.rand(100, 1), axis=0)   # Inputs in range [0, 5]
y = np.sin(X).ravel() + 0.3 * np.random.randn(100)  # Noisy sine data
tau = 0.5
y_pred = locally_weighted_regression(X, y, tau)
plt.scatter(X, y, color='blue', label='Data Points', alpha=0.5)
plt.plot(X, y_pred, color='red', label=f'LWR Fit (tau={tau})', linewidth=2)
plt.title('Locally Weighted Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
```

plt.show()



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

```
import numpy as np
import random
import matplotlib.pyplot as plt
GRID_SIZE = 5
START, GOAL = (0, 0), (4, 4)
OBSTACLES = [(1, 1), (2, 2), (3, 1), (0, 3)]
ACTIONS = [(-1, 0), (0, 1), (1, 0), (0, -1)]  # Up, Right, Down, Left

ALPHA, GAMMA, EPSILON, EPISODES = 0.1, 0.9, 0.2, 500
q_table = np.zeros((GRID_SIZE, GRID_SIZE, 4))
performance_history = []
for e in range(EPISODES):
    state = START
    total_reward = 0
    while state != GOAL:
```

```
        x, y = state
        # Choose action (Epsilon-greedy)
        action = random.randint(0, 3) if random.random() < EPSILON else
np.argmax(q_table[x, y])
        dx, dy = ACTIONS[action]
        nx, ny = x + dx, y + dy
        if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE and (nx, ny) not in
OBSTACLES:
            next_s = (nx, ny)
        else:
            next_s = state


        reward = 10 if next_s == GOAL else -1
        total_reward += reward
        q_table[x, y, action] += ALPHA * (reward + GAMMA *
np.max(q_table[next_s]) - q_table[x, y, action])
        state = next_s


    performance_history.append(total_reward)
state, path = START, [START]
while state != GOAL and len(path) < 20:
    action = np.argmax(q_table[state])
    state = (max(0, min(GRID_SIZE-1, state[0] + ACTIONS[action][0])),
             max(0, min(GRID_SIZE-1, state[1] + ACTIONS[action][1])))
    path.append(state)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(performance_history, color='tab:blue')
```

```
plt.title("Performance: Total Reward per Episode")
plt.xlabel("Episode"); plt.ylabel("Reward")
plt.subplot(1, 2, 2)
grid_view = np.zeros((GRID_SIZE, GRID_SIZE))
for obs in OBSTACLES: grid_view[obs] = -1
grid_view[GOAL] = 1
plt.imshow(grid_view, cmap='RdYlGn')
py, px = zip(*path) # Unzip path for plotting
plt.plot(px, py, marker='o', color='blue', label='Agent Path')
plt.title("Learned Optimal Path")
plt.legend()
plt.tight_layout()
plt.show()
```