INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No** | **Date** | **Practicals** | **Signature** |
| 1 |  | Implement simple linear regression model on a standard data set and plot the least square regression fit. Comment on the result. |  |
| 2 |  | Implement multiple regression model on a standard data set and plot the least square regression fit. Comment on the result. |  |
| 3 |  | Fit a classification model using K Nearest Neighbour (KNN) Algorithm on a given data set. |  |
| 4 |  | For a given data set, perform the following:  (i) Perform the polynomial regression and make a plot of the resulting polynomial fit to the data.  (ii) Fit a step function and perform cross validation to choose the optimal number of cuts. Make a plot of the fit to the data. |  |
| 5 |  | For a given data set, do the following:  (i) Fit a classification tree (ii) Fit a regression tree. |  |
|  |  |  |  |

**PRACTICAL NO: 1**

**Aim:** Implement simple linear regression model on a standard data set and plot the least square regression fit. Comment on the result.

**INPUT:- 1st import housing.csv**

import pandas as pd

# Load the dataset to inspect its contents

file\_path = 'housing.csv'

data = pd.read\_csv(file\_path)

# Display the first few rows of the dataset to understand its structure

data.head()

**OUTPUT:-**



**INPUT:-**

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Selecting RM as the independent variable and MEDV as the dependent variable

X = data[['RM']]  # Independent variable (average number of rooms)

y = data['MEDV']  # Dependent variable (median home value)

# Creating and fitting the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predicting values using the model

y\_pred = model.predict(X)

# Plotting the data points and the regression line

plt.figure(figsize=(10, 6))

plt.scatter(X, y, color='blue', label='Data points', alpha=0.6)

plt.plot(X, y\_pred, color='red', label='Regression line')

# Adding labels and title

plt.xlabel('Average Number of Rooms (RM)')

plt.ylabel('Median Home Value (MEDV) in USD')

plt.title('Least Squares Regression Fit: RM vs MEDV')

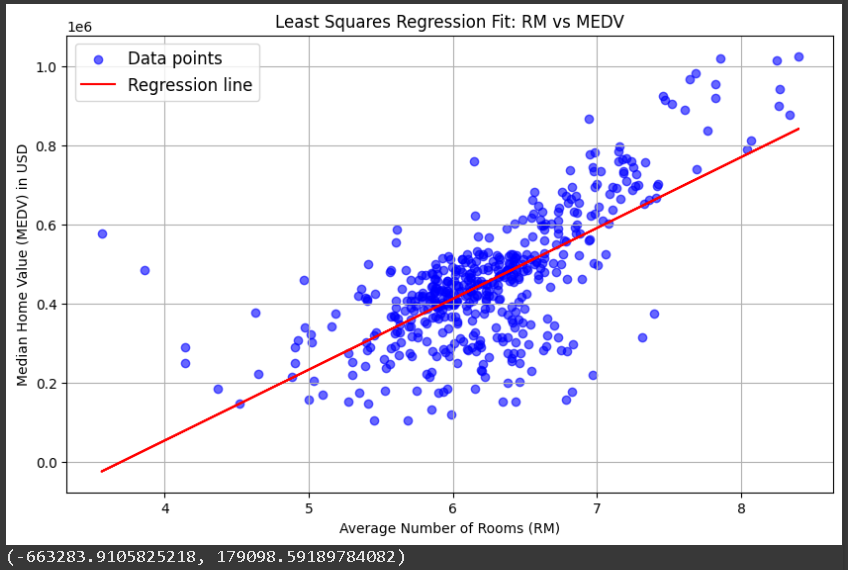
plt.legend()

plt.grid(True)

plt.show()

model.coef\_[0]

**OUTPUT:-**



(179098.59189784082)

**COMMENT:-**

The plot shows a positive link between the number of rooms (RM) and home value (MEDV), with the red line showing the trend.

According to the model, each extra room increases home value by about $179,099. However, the negative starting point suggests that this simple model doesn’t fully capture all aspects of the data, especially for homes with very few rooms.

**PRACTICAL NO: 2**

**Aim:** Implement multiple regression model on a standard data set and plot the least square regression fit. Comment on the result.

**INPUT:-**

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

    x = []

    y = []

    random\_x1 = np.random.rand()

    random\_x2 = np.random.rand()

    for i in range(n):

        x1 = i

        x2 = i/2 + np.random.rand()\*n

        x.append([1, x1, x2])

        y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1) #This line was also indented

    return np.array(x), np.array(y)

# Define 'n' before calling generate\_dataset

n = 200

x, y = generate\_dataset(n)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.add\_subplot(projection ='3d')

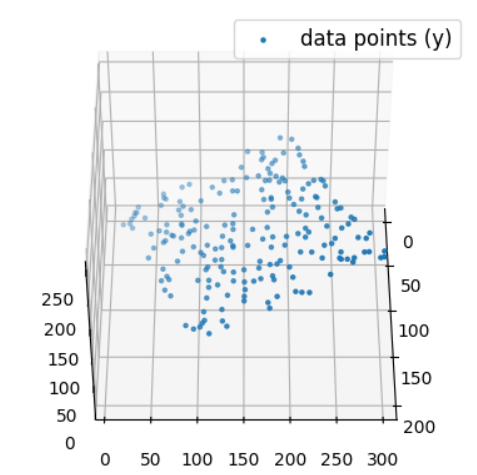
ax.scatter(x[:, 1], x[:, 2], y, label ='data points (y)', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

**OUTPUT:-**

**COMMENT:-**

The R² score of 0.67 suggests that the

model explains about 67% of the variance

in home values. The plot shows predicted

values clustering around the actual values

line (red line), indicating a reasonable fit.

However, the spread around this line

highlights that there are still factors

influencing home prices.

**PRACTICAL NO: 3**

**Aim:** Fit a classification model using K Nearest Neighbour (KNN) Algorithm on a given data set.

**INPUT:- 1st import dataset housing.csv**

# Import necessary modules

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

# Load housing.csv dataset

data = pd.read\_csv('housing.csv')

# Create binary target variable based on the median value of MEDV

data['MEDV\_class'] = (data['MEDV'] > data['MEDV'].median()).astype(int)

# Define features and target arrays

X = data[['RM', 'LSTAT', 'PTRATIO']]  # Features

y = data['MEDV\_class']  # Target (binary)

# Split into training and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features (important for KNN)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Set up K values (number of neighbors) to test

neighbors = np.arange(1, 9)

train\_accuracy = np.empty(len(neighbors))

test\_accuracy = np.empty(len(neighbors))

# Loop over K values

for i, k in enumerate(neighbors):

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    # Compute training and test data accuracy

    train\_accuracy[i] = knn.score(X\_train, y\_train)

    test\_accuracy[i] = knn.score(X\_test, y\_test)

# Generate plot

plt.plot(neighbors, test\_accuracy, label='Testing dataset Accuracy', color='blue')

plt.plot(neighbors, train\_accuracy, label='Training dataset Accuracy', color='orange')

plt.legend()

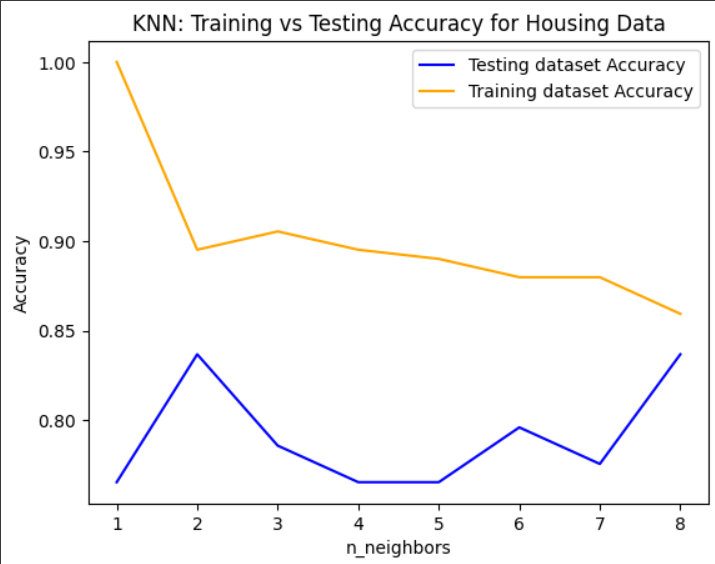
plt.xlabel('n\_neighbors')

plt.ylabel('Accuracy')

plt.title('KNN: Training vs Testing Accuracy for Housing Data')

plt.show()

**OUTPUT:-**



**PRACTICAL NO: 4**

**Aim:** For a given data set, perform the following:

(i) Perform the polynomial regression and make a plot of the resulting polynomial fit to the data.

**INPUT:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import cross\_val\_score

from sklearn.tree import DecisionTreeRegressor

# Generate sample data

np.random.seed(0)

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel()

y[::5] += 3 \* (0.5 - np.random.rand(16))

# Polynomial Regression

degree = 3  # Degree of the polynomial

poly\_features = PolynomialFeatures(degree=degree, include\_bias=False)

X\_poly = poly\_features.fit\_transform(X)

poly\_reg = LinearRegression()

poly\_reg.fit(X\_poly, y)

# Plot polynomial fit

X\_plot = np.linspace(0, 5, 100).reshape(-1, 1)

X\_plot\_poly = poly\_features.transform(X\_plot)

y\_plot = poly\_reg.predict(X\_plot\_poly)

plt.figure(figsize=(8, 6))

plt.scatter(X, y, label="Data")

plt.plot(X\_plot, y\_plot, color="red", label="Polynomial Regression")

plt.title("Polynomial Regression")

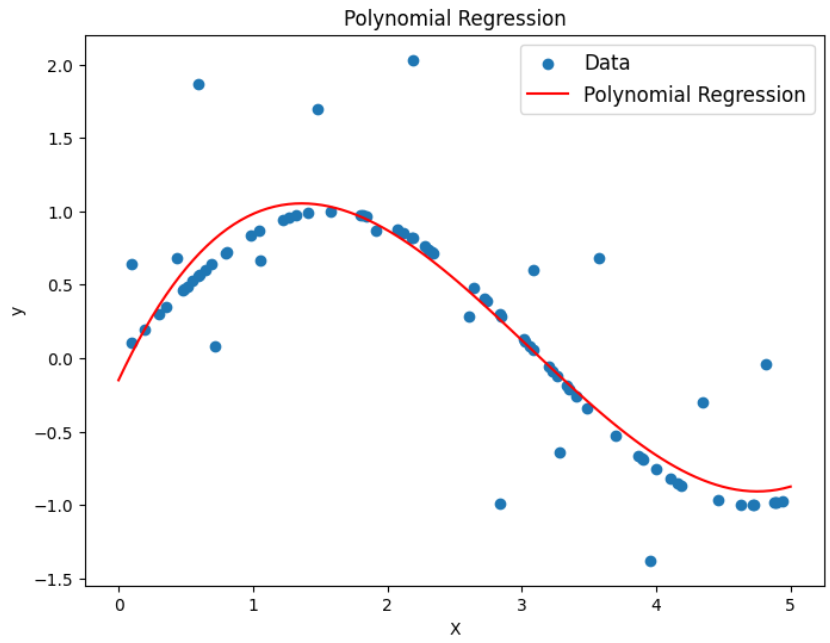
plt.xlabel("X")

plt.ylabel("y")

plt.legend()

plt.show()

**OUTPUT:-**



**INPUT:-** (ii) Fit a step function and perform cross validation to choose the optimal number of cuts. Make a plot of the fit to the data.

# Step Function with Cross-Validation

max\_cuts = 10  # Maximum number of cuts to consider

cv\_scores = []

for cuts in range(1, max\_cuts + 1):

    regressor = DecisionTreeRegressor(max\_depth=1)  # Depth=1 creates a step function

    scores = cross\_val\_score(regressor, X, y, cv=5, scoring='neg\_mean\_squared\_error')

    cv\_scores.append(scores.mean())

optimal\_cuts = np.argmax(cv\_scores) + 1  # Add 1 to get the actual number of cuts

# Fit the step function with optimal cuts

regressor = DecisionTreeRegressor(max\_depth=1)

regressor.fit(X, y)

# Plot step function fit

X\_plot = np.linspace(0, 5, 100).reshape(-1, 1)

y\_plot = regressor.predict(X\_plot)

plt.figure(figsize=(8, 6))

plt.scatter(X, y, label="Data")

plt.plot(X\_plot, y\_plot, color="green", label="Step Function")

plt.title("Step Function with Cross-Validation")

plt.xlabel("X")

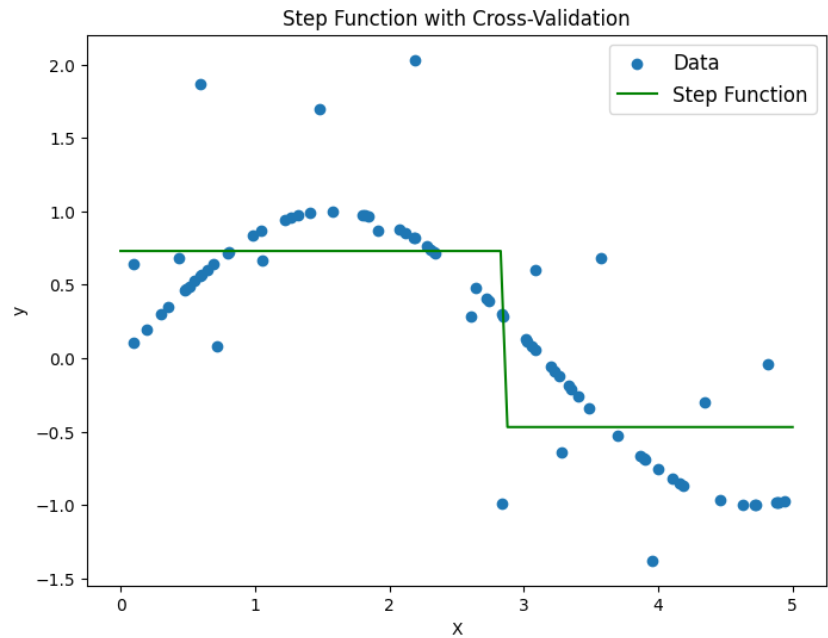
plt.ylabel("y")

plt.legend()

plt.show()

print(f"Optimal number of cuts: {optimal\_cuts}")

**OUTPUT:-**



**PRACTICAL NO: 5**

**Aim:** For a given data set, do the following: (i) Fit a classification tree

**INPUT:-**

import matplotlib.pyplot as plt

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot\_tree

# Load the iris dataset

iris = load\_iris()

X = iris.data  # Features

y = iris.target  # Target variable (species)

# 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, random\_state=42)

# Classification Tree

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

# Plot the classification tree

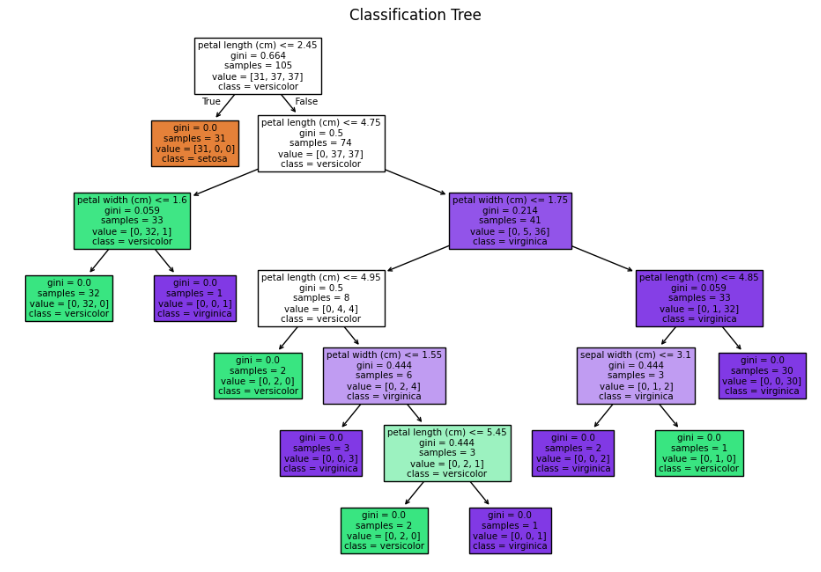
plt.figure(figsize=(12, 8))

plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.title("Classification Tree")

plt.show()

**OUTPUT:-**



**INPUT:** (ii) Fit a regression tree.

# Regression Tree (using sepal length as target)

y\_reg = X[:, 0]  # Sepal length as target for regression

X\_reg = X[:, 1:]  # Remaining features as predictors

reg = DecisionTreeRegressor(random\_state=42)

reg.fit(X\_reg, y\_reg)

# Plot the regression tree

plt.figure(figsize=(12, 8))

plot\_tree(reg, feature\_names=iris.feature\_names[1:], filled=True)

plt.title("Regression Tree (Predicting Sepal Length)")

plt.show()

**OUTPUT:-**

