SAI VIDYA INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi and Govt. of Karnataka)
Accredited by NBA, New Delhi (CSE, ISE, ECE), NAAC-'A' Grade
Rajanukunte, Bengaluru – 560 064
Tel: 080-2846 8196, email: hodcse@saividya.ac.in web: www.saividya.ac.in

MOTTO

"Learn to lead"

VISION

Contribute dedicated, skilled, intelligent Computer Engineers to architect strong India and the world.

MISSION

- ***To provide quality education and skilled training to produce dedicated engineers and managers***
- ***To promote research, innovation and ethical practices by creating supportive environment***
- ***To undertake collaborative projects with academia and industry that transform young minds into socially responsible citizen and globally competent professionals***
- ***To enhance personality traits which lead to entrepreneurship qualities among the students***

MACHINE LEARNING LAB(BCSL606)

(As per Visvesvaraya Technological University Syllabus)

Compiled by:

Prof. Salma Itagi
Asst.Professor,
Dept. of CSE

Prof. Jayaprada S Hiremath,
Asst.Professor,
Dept. of CSE

Prof. Yashaswini D M
Asst. Professor,
Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
2024-25

Disclaimer

The information contained in this document is the proprietary and exclusive property of Sai Vidya Institute of Technology, Bengaluru except as otherwise indicated. No part of this document, in whole or in part, may be reproduced, stored, transmitted, or used for course material development purposes without the prior written permission of Sai Vidya Institute of Technology, Bengaluru. The information contained in this document is subject to change without notice. The information inthis document is provided for informational purposes only.

Trademark

Edition: 2024-25

Document Owners

The primary contacts for questions regarding this document are:

Authors:

1. Prof. Salma Itagi
2. Jayaprada S Hiremath

Department: Computer Science & Engineering

Contact email id:

1. salma.itagi@saividya.ac.in
2. jayaprada.shiremath@saividya.ac.in
3. yashaswini.dm@saividya.ac.in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

- *Contribute dedicated, skilled, intelligent Computer Engineers to architect strong India and the world*

MISSION

- *To promote activities that imparts high quality technical education in core domains of computer engineering and also interdisciplinary areas.*
- *To imbibe innovative projects and research based skills using software to solve real time problems for building modern India and the World.*
- *To create a conducive environment for life long learning with ethical responsibilities to produce globally competent computer professionals and entrepreneurs.*

PROGRAM EDUCATIONAL OBJECTIVE

PEO 1 *: Graduates will have the expertise in analyzing real time problems and providing appropriate solutions related to Computer Science & Engineering.*
PEO 2 *: Graduates will have the knowledge of fundamental principles and innovative technologies to succeed in higher studies, and research.*
PEO 3 *: Graduates will continue to learn and to adapt technology developments combined with deep awareness of ethical responsibilities in profession.*

Program Outcomes

1 **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2 **Problem Analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3 **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4 **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5 **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6 **The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7 **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8 **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9 **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.s
10 **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11 **Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12 **Life-Long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO 1:Demonstrate the knowledge and understanding of working principles,design ,implement and test and evaluate the hardware components of a computer system. software.

PSO 2: Apply standard Software engineering practices and strategies project development.

PSO3: Demonstrate the knowledge of Discrete Mathematucs and Data Managemnt and Data Engineering.

.

Machine Learning lab		Semester	6
Course Code	BCSL606	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	100
Examination type (SEE)	Practical		

Course objectives:

- To become familiar with data and visualize univariate, bivariate, and multivariate data using statistical techniques and dimensionality reduction.
- To understand various machine learning algorithms such as similarity-based learning, regression, decision trees, and clustering.
- To familiarize with learning theories, probability-based models and developing the skills required for decision-making in dynamic environments.

Sl.NO	Experiments
1	Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset. **Book 1: Chapter 2**
2	Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset. **Book 1: Chapter 2**
3	Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2. **Book 1: Chapter 2**
4	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples. **Book 1: Chapter 3**
5	Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of *x* in the range of [0,1]. Perform the following based on dataset generated. a. Label the first 50 points $\{x_1,\ldots\ldots,x_{50}\}$ as follows: if ($x_i \le 0.5$), then $x_i \in \text{Class}_1$, else $x_i \in \text{Class}_1$ b. Classify the remaining points, $x_{51},\ldots\ldots,x_{100}$ using KNN. Perform this for *k=1,2,3,4,5,20,30* **Book 2: Chapter – 2**
6	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs **Book 1: Chapter – 4**
7	Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression. **Book 1: Chapter – 5**
8	Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample. **Book 2: Chapter – 3**

9	Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets. **Book 2: Chapter – 4**
10	Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result. **Book 2: Chapter – 4**

Course outcomes (Course Skill Set):
At the end of the course the student will be able to:

- Illustrate the principles of multivariate data and apply dimensionality reduction techniques.
- Demonstrate similarity-based learning methods and perform regression analysis.
- Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning.
- Implement the clustering algorithms to share computing resources.

Program 1:

Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

To develop a program that creates histograms and box plots for all numerical features of the California Housing dataset, we'll follow these steps:

1. **Load the dataset**: We'll load the California Housing dataset, which is available through libraries like sklearn.
2. **Generate histograms**: We'll use histograms to visualize the distribution of each numerical feature.
3. **Generate box plots**: We'll create box plots to identify any outliers in the dataset.
4. **Analyze the results**: We'll look for trends in the histograms and box plots.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target  # Adding target variable (house value) for context

# Step 2: Plot histograms for all numerical features
def plot_histograms(df):
  num_features = df.select_dtypes(include=[np.number]).columns   # Get numerical
columns
  for feature in num_features:
    plt.figure(figsize=(8, 6))
    sns.histplot(df[feature], kde=True, bins=30)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()

# Step 3: Plot box plots for all numerical features
def plot_boxplots(df):
num_features = df.select_dtypes(include=[np.number]).columns   # Get numerical
```

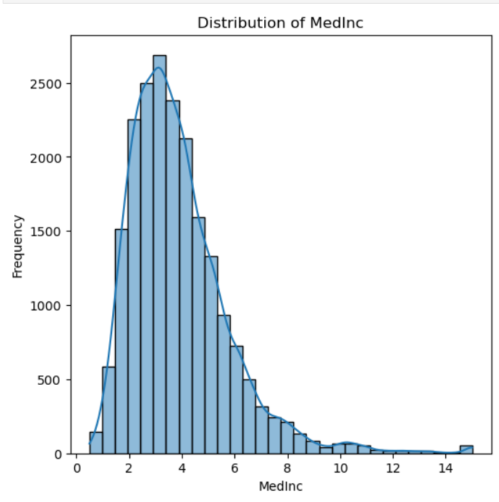
```
columns
    for feature in num_features:
        plt.figure(figsize=(8, 6))
        sns.boxplot(x=df[feature])
        plt.title(f'Box Plot of {feature}')
        plt.xlabel(feature)
        plt.show()

# Step 4: Analyze distribution and outliers
def analyze_features(df):
    num_features = df.select_dtypes(include=[np.number]).columns    # Get numerical
columns
    summary_stats = df[num_features].describe().transpose()  # Summary statistics
    print("Summary Statistics of the Numerical Features:")
    print(summary_stats)

    # Identify potential outliers by considering values beyond 1.5*IQR for each feature
    for feature in num_features:
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        outliers = df[(df[feature] < lower_bound) | (df[feature] > upper_bound)]
        print(f"\nOutliers for {feature}: {len(outliers)}")
        if len(outliers) > 0:
            print(outliers[feature].head())

# Running the functions
plot_histograms(df)
plot_boxplots(df)
analyze_features(df)
```

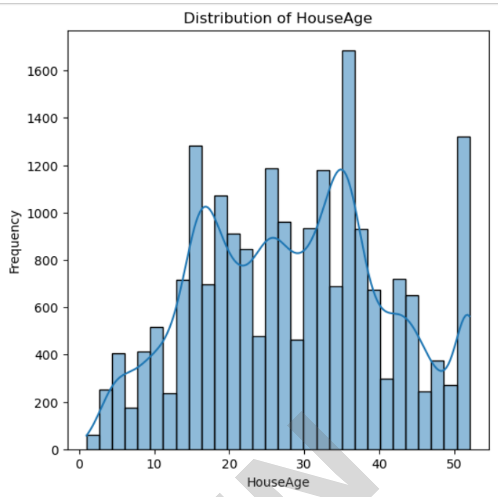
Output:

- **Histograms**: For each feature, a histogram with a KDE curve will be displayed. This will allow you to visually inspect whether a feature is normally distributed, skewed, bimodal, etc.
- **Distribution Analysis**: For each feature, you will see the following printed statistics:
    - Mean, Median, Std Dev (Standard Deviation) to check the central tendency and spread.
    - Skewness and Kurtosis to determine the shape of the distribution.
    - Min, Max, and quartiles to understand the range and spread of the data.

```
Summary Statistics of the Numerical Features:
              count        mean          std         min        25%
MedInc      20640.0    3.870671     1.899822    0.499900    2.563400
HouseAge    20640.0   28.639486    12.585558    1.000000   18.000000
AveRooms    20640.0    5.429000     2.474173    0.846154    4.440716
AveBedrms   20640.0    1.096675     0.473911    0.333333    1.006079
Population  20640.0 1425.476744  1132.462122    3.000000  787.000000
AveOccup    20640.0    3.070655    10.386050    0.692308    2.429741
Latitude    20640.0   35.631861     2.135952   32.540000   33.930000
Longitude   20640.0 -119.569704     2.003532 -124.350000 -121.800000
target      20640.0    2.068558     1.153956    0.149990    1.196000

                    50%          75%           max
MedInc         3.534800     4.743250     15.000100
HouseAge      29.000000    37.000000     52.000000
AveRooms       5.229129     6.052381    141.909091
AveBedrms      1.048780     1.099526     34.066667
Population  1166.000000  1725.000000  35682.000000
AveOccup       2.818116     3.282261   1243.333333
Latitude      34.260000    37.710000     41.950000
Longitude   -118.490000  -118.010000   -114.310000
target         1.797000     2.647250      5.000010

Outliers for MedInc: 681
0       8.3252
1       8.3014
131    11.6017
134     8.2049
135     8.4010
Name: MedInc, dtype: float64

Outliers for HouseAge: 0
Outliers for AveRooms: 511
73     1.714286
155    8.972868
511    8.928358
512    9.210227
514    9.122715
Name: AveRooms, dtype: float64

Outliers for AveBedrms: 1424
41    1.248996
57    1.372951
59    0.754386
61    1.260870
62    1.557377
Name: AveBedrms, dtype: float64

Outliers for Population: 1196
95     3469.0
185    4367.0
283    4985.0
460    3337.0
485    3276.0
Name: Population, dtype: float64

Outliers for AveOccup: 711
89      4.658824
91     10.272727
92      5.617647
200     4.666667
270    12.234043
Name: AveOccup, dtype: float64

Outliers for Latitude: 0

Outliers for Longitude: 0

Outliers for target: 1071
89    5.00001
```

```
140    4.83300
459    5.00001
489    4.89600
493    5.00001
Name: target, dtype: float64
```





Program 2: Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.

To compute the correlation matrix and visualize the relationships between pairs of features, we will use the following steps:

1. **Compute the correlation matrix**: This will allow us to understand the relationships between all numerical features.
2. **Visualize the correlation matrix** using a heatmap: This will give us an easy-to-interpret visual representation of the correlation values.
3. **Create a pair plot** to visualize pairwise relationships between features, which will help us to see the distribution and potential relationships visually.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target  # Adding target variable (house value)

# Step 2: Compute the correlation matrix
def compute_correlation_matrix(df):
    correlation_matrix = df.corr()   # Compute correlation matrix for all numerical
features
    return correlation_matrix

# Step 3: Visualize the correlation matrix using a heatmap
def plot_correlation_heatmap(correlation_matrix):
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
    plt.title('Correlation Matrix Heatmap')
    plt.show()
```

```
# Step 4: Create a pair plot to visualize pairwise relationships
def plot_pair_plot(df):
    # Selecting a subset of features to make the plot more readable
    sns.pairplot(df, diag_kind='kde', kind='scatter', height=2.5)
    plt.suptitle('Pair Plot of Features', y=1.02)
    plt.show()
```

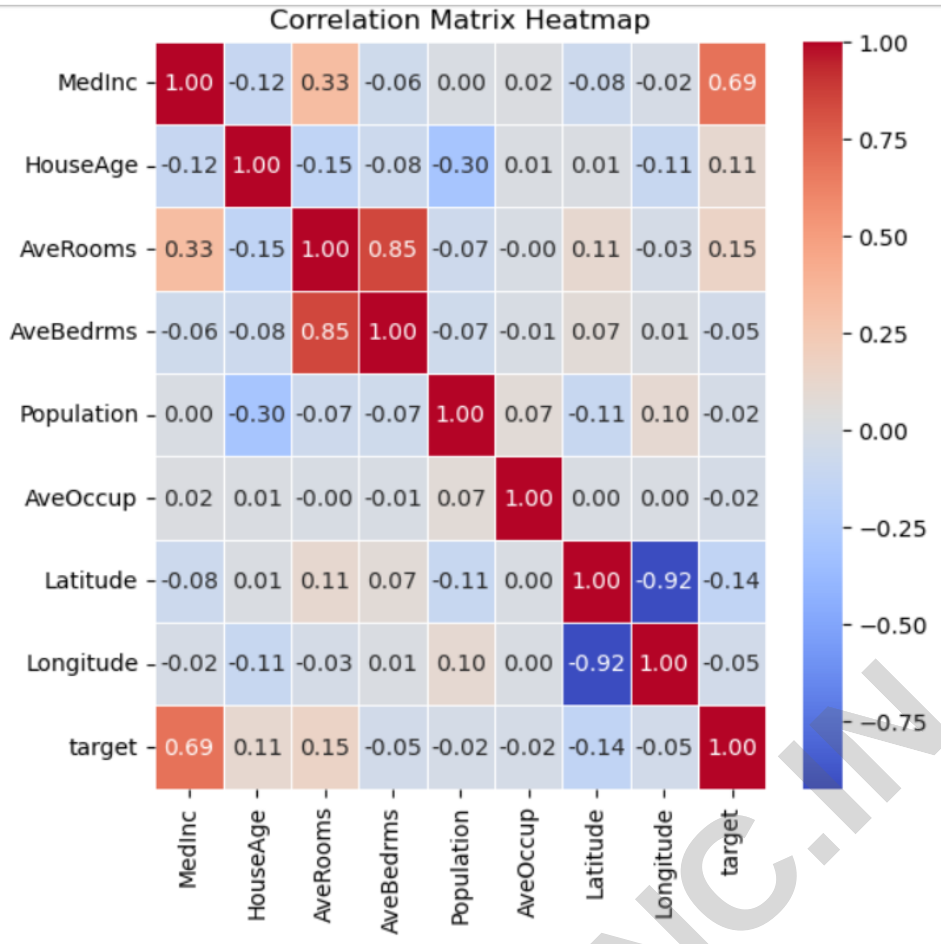
```
# Step 4: Create a pair plot to visualize pairwise relationships
def plot_pair_plot(df):
    # Selecting a subset of features to make the plot more readable
    sns.pairplot(df, diag_kind='kde', kind='scatter', height=2.5)
    plt.suptitle('Pair Plot of Features', y=1.02)
    plt.show()
```

```
# Step 5: Running the program
correlation_matrix = compute_correlation_matrix(df)
plot_correlation_heatmap(correlation_matrix)
plot_pair_plot(df)
```

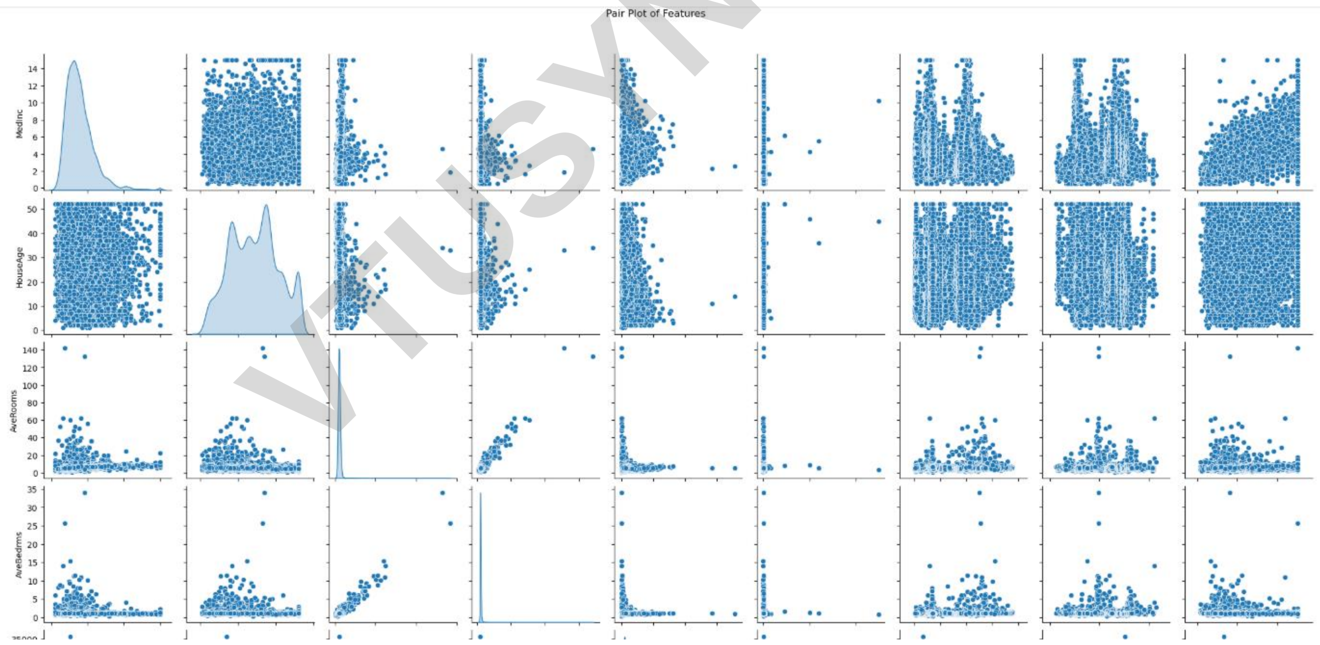
Output:

1. **Correlation Matrix Heatmap**:
    - A heatmap will be displayed showing the correlation between each pair of features. Features with high correlation (either positive or negative) will be highlighted.
2. **Pair Plot**:
    - A pair plot will be generated showing scatter plots for each pair of features. The diagonal of the plot will show the distribution of each feature with a KDE plot.

In this matrix, the values represent the Pearson correlation coefficient between the pairs of features. For instance, MedInc (Median Income) has a strong positive correlation with the target variable (house value) at 0.68.



	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
MedInc	1.00	-0.12	0.33	-0.06	0.00	0.02	-0.08	-0.02	0.69
HouseAge	-0.12	1.00	-0.15	-0.08	-0.30	0.01	0.01	-0.11	0.11
AveRooms	0.33	-0.15	1.00	0.85	-0.07	-0.00	0.11	-0.03	0.15
AveBedrms	-0.06	-0.08	0.85	1.00	-0.07	-0.01	0.07	0.01	-0.05
Population	0.00	-0.30	-0.07	-0.07	1.00	0.07	-0.11	0.10	-0.02
AveOccup	0.02	0.01	-0.00	-0.01	0.07	1.00	0.00	0.00	-0.02
Latitude	-0.08	0.01	0.11	0.07	-0.11	0.00	1.00	-0.92	-0.14
Longitude	-0.02	-0.11	-0.03	0.01	0.10	0.00	-0.92	1.00	-0.05
target	0.69	0.11	0.15	-0.05	-0.02	-0.02	-0.14	-0.05	1.00



Program 3:

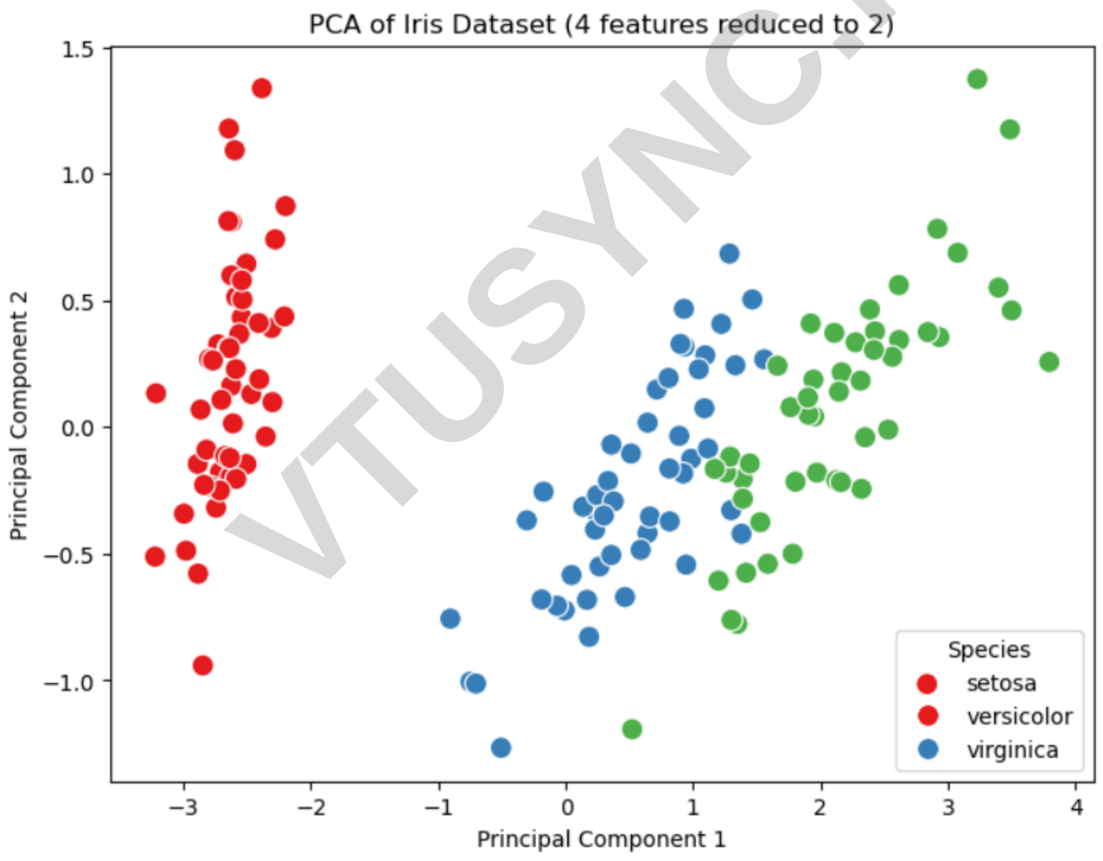
Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

To implement **Principal Component Analysis (PCA)** and reduce the dimensionality of the Iris dataset from 4 features to a lower-dimensional space (for example, reducing from 4 features to 2 features), you can use the sklearn.decomposition.PCA class. PCA is a technique used for dimensionality reduction by projecting the data into a lower-dimensional space while preserving as much variance as possible.

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target  # Adding target variable (species)

# Step 2: Perform PCA to reduce the dimensionality from 4 features to 2
def apply_pca(df, n_components=2):
    # Extracting the features (X) and target (y)
    X = df.iloc[:, :-1].values  # All columns except the target
    y = df.iloc[:, -1].values   # Target (species)

    # Initialize PCA with the desired number of components (2 in this case)
    pca = PCA(n_components=n_components)

    # Fit and transform the data
    principal_components = pca.fit_transform(X)

    # Convert the principal components to a DataFrame
    pca_df = pd.DataFrame(data=principal_components, columns=[f'PC{i+1}' for i in
range(n_components)])
    pca_df['target'] = y  # Add target variable (species) to the dataframe
    return pca_df
# Step 3: Visualize the results
def visualize_pca(pca_df):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='PC1', y='PC2', hue='target', palette='Set1', data=pca_df, s=100)
    plt.title('PCA of Iris Dataset (4 features reduced to 2)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(title='Species', labels=iris.target_names)
    plt.show()

# Step 4: Running the program
pca_df = apply_pca(df)
visualize_pca(pca_df)
```

Output:

The program will produce a scatter plot where the data points are represented as dots in a 2D space, with the axes corresponding to the two principal components (PC1 and PC2). The data points will be color-coded by species, allowing you to see how the species are distributed in the 2D PCA space.



Program 4:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

The **Find-S algorithm** is a simple method used in machine learning to learn a concept or hypothesis from a set of training examples. The goal of the algorithm is to find the most specific hypothesis that is consistent with all positive examples in the dataset.

Steps of the Find-S Algorithm:

1. **Initialize the hypothesis**: Start with the most specific hypothesis, which is equivalent to the first positive training example.
2. **For each positive example**: If the example is consistent with the hypothesis, no change is made. If it is not, generalize the hypothesis to be consistent with this example by adjusting it to match the example (i.e., changing the values of the hypothesis features to match the example).
3. **Output the hypothesis**: After processing all positive examples, the final hypothesis will be the most specific one that is consistent with all the positive training examples.

```
import pandas as pd
# Step 1: Read the CSV file into a pandas DataFrame
def read_data(filename):
    df = pd.read_csv(filename)
    return df

# Step 2: Implement the Find-S algorithm
def find_s_algorithm(df):
    # Assume the class label is in the last column (index -1)
    # Separate the features (X) and the class labels (y)
    X = df.iloc[:, :-1].values  # All columns except the last one (features)
    y = df.iloc[:, -1].values   # Last column (class labels)

    # Step 3: Find the first positive example and initialize the hypothesis
    hypothesis = list(X[0])   # Start with the first positive example (most specific 
hypothesis)

    # Step 4: For each positive example, generalize the hypothesis if necessary
    for i in range(1, len(X)):
        if y[i] == 1:  # Positive example
            for j in range(len(hypothesis)):
                # Generalize hypothesis to match the current example
                if hypothesis[j] != X[i][j]:
                    hypothesis[j] = '?'  # Use '?' to generalize (match any value)

    return hypothesis
```

```
# Step 5: Display the learned hypothesis
def display_hypothesis(hypothesis):
    print("Learned Hypothesis:")
    print(" - " + " AND ".join(str(feature) for feature in hypothesis))
# Step 6: Main function to run the program
def main():
    # Replace 'train.csv' with your actual CSV file name
    filename = 'train.csv'

    # Read the data
    df = read_data(filename)

    # Apply the Find-S algorithm
    hypothesis = find_s_algorithm(df)

    # Display the learned hypothesis
    display_hypothesis(hypothesis)

if __name__ == "__main__":
    main()
```

Output: Learned Hypothesis:

For the above dataset, if we assume the dataset has positive examples as "PlayTennis = Yes", the program will output a hypothesis such as:

```
Learned Hypothesis:
 - Sky = Sunny AND AirTemp = Hot AND Humidity = High AND Wind = ?
```

This means that the learned hypothesis is that `Sky = Sunny`, `AirTemp = Hot`, and `Humidity = High` are specific conditions for playing tennis, while `Wind` can be any value (`?`).

- Sunny AND Hot AND High AND Weak

Program 5:
Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100
values of *x* in the range of [0,1]. Perform the following based on dataset generated.
a. Label the first 50 points {*x1,……,x50*} as follows: if (xi ≤ 0.5), then *xi* ε Class1, else *xi* ε Class1
b. Classify the remaining points, *x51,……,x100* using KNN. Perform this for *k=1,2,3,4,5,20,30*

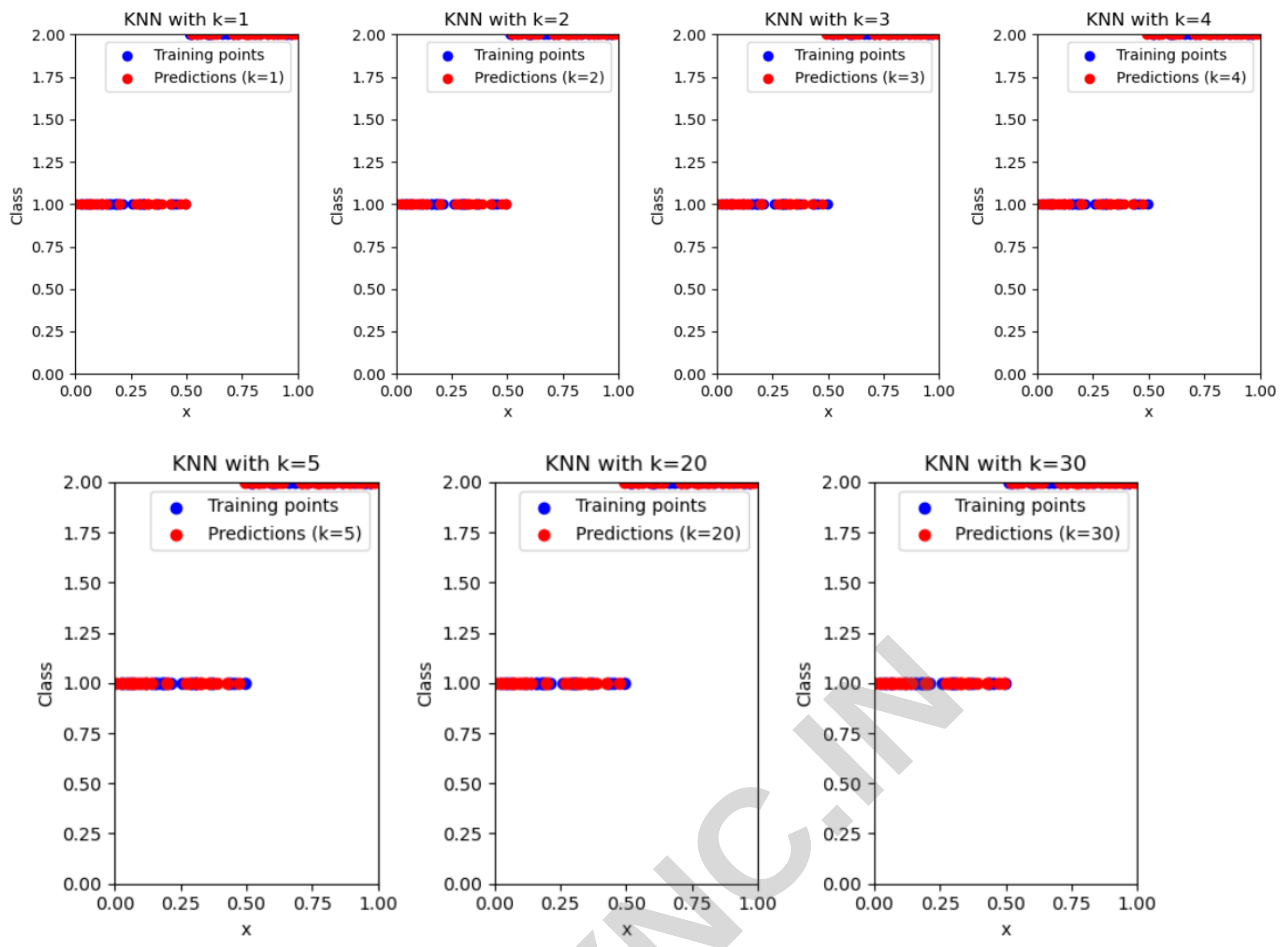
The **k-Nearest Neighbors (KNN)** algorithm is a simple and intuitive classification technique. It works by assigning a class to a data point based on the majority class of its k-nearest neighbors. Here's a step-by-step approach to implement KNN for the problem where we generate 100 random values of x in the range [0, 1], label the first 50 values, and classify the remaining points using KNN for various values of k.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Generate the data
np.random.seed(42)  # For reproducibility
X_train = np.random.rand(50, 1)  # First 50 points (features)
y_train = np.array([1 if x <= 0.5 else 2 for x in X_train])  # Labels for the first 50 points

X_test = np.random.rand(50, 1)  # Points to be classified (x51 to x100)
y_test = []  # To store the predictions

# Step 2: Define a function to perform KNN and classify the test points
def classify_knn(X_train, y_train, X_test, k_values):
    predictions = {}  # To store the results for different k values

    # Iterate through different k values
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)  # Fit the KNN model on the training data
        y_pred = knn.predict(X_test)  # Predict the class labels for the test data
        predictions[k] = y_pred  # Store the predictions for this k

    return predictions
```

```
# Step 3: Define the k values to test and classify the points
k_values = [1, 2, 3, 4, 5, 20, 30]
predictions = classify_knn(X_train, y_train, X_test, k_values)

# Step 4: Plot the results
plt.figure(figsize=(12, 8))

# Plot for each value of k
for i, k in enumerate(k_values, 1):
    plt.subplot(2, 4, i)
    plt.scatter(X_train, y_train, color='blue', label='Training points')   # Training points
(X1 to X50)
    plt.scatter(X_test, predictions[k], color='red', label=f'Predictions (k={k})')   # Test
points (X51 to X100)

    plt.title(f'KNN with k={k}')
    plt.xlabel('x')
    plt.ylabel('Class')
    plt.xlim(0, 1)
    plt.ylim(0, 2)
    plt.legend()

plt.tight_layout()
plt.show()
```

Output:

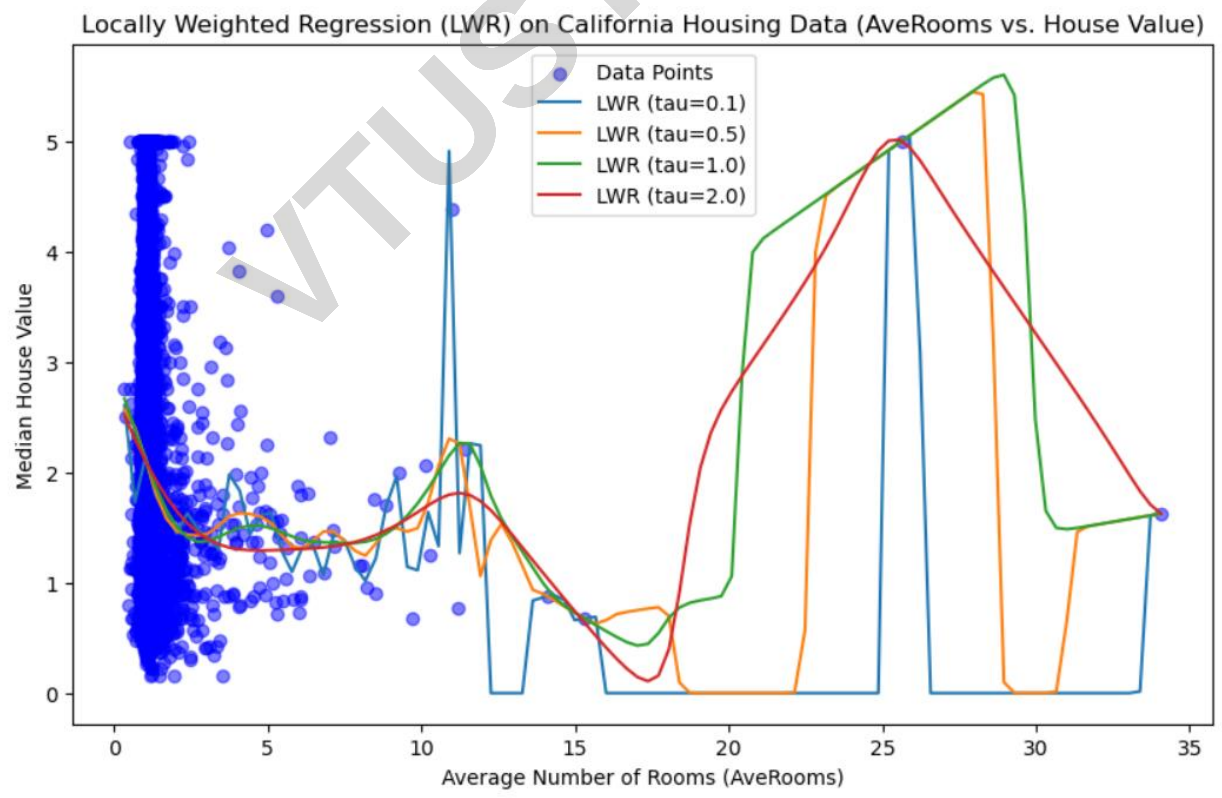
1. The output will be a plot showing multiple subplots, each corresponding to a different value of k. In each subplot:
2. **Blue dots** represent the training points (x1 to x50), which are labeled as Class 1 or Class 2.
3. **Red dots** represent the test points (x51 to x100), which are classified according to the KNN algorithm for each value of k.
4. **Interpretation:**
5. **k=1**: The decision boundary will be very specific, and individual test points will be classified based on their closest neighbor in the training set.
6. **k=5, k=20, k=30**: The decision boundary becomes smoother as k increases, and the classification becomes less sensitive to individual data points. The predictions will be based on a larger neighborhood.
7. **k=2, k=3, k=4**: For these values, the decision boundaries will be less specific compared to k=1, but more complex compared to larger values of k.


KNN with k=1
Training points
Predictions (k=1)
KNN with k=2
Training points
Predictions (k=2)
KNN with k=3
Training points
Predictions (k=3)
KNN with k=4
Training points
Predictions (k=4)
KNN with k=5
Training points
Predictions (k=5)
KNN with k=20
Training points
Predictions (k=20)
KNN with k=30
Training points
Predictions (k=30)
Class
x
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Program 6: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression

# Step 1: Load the California Housing dataset
california_housing = fetch_california_housing()
X = california_housing.data[:, 3].reshape(-1, 1)  # Use the 'AveRooms' feature (average
number of rooms)
y = california_housing.target  # Target variable (median house value)

# Step 2: Define the Locally Weighted Regression function with regularization
def locally_weighted_regression(X_train, y_train, X_test, tau=1.0, epsilon=1e-5):
    m = len(X_train)
    y_pred = np.zeros(len(X_test))

    for i, x in enumerate(X_test):
        x = x[0]  # Extract scalar value from the 2D array
        # Compute the weight matrix (Gaussian kernel)
        weights = np.exp(-np.sum((X_train - x)**2, axis=1) / (2 * tau**2))
        W = np.diag(weights)
  # Solve the weighted least squares problem: (X^T W X + epsilon I) (theta) = (X^T W y)
  X_train_augmented = np.hstack((np.ones((m, 1)), X_train))  # Add bias term (x0 = 1)
        XTX = X_train_augmented.T @ W @ X_train_augmented
        XTX_reg = XTX + epsilon * np.eye(X_train_augmented.shape[1])    # Add
regularization term
        theta = np.linalg.inv(XTX_reg) @ (X_train_augmented.T @ W @ y_train)

        # Make the prediction for the test point
        X_test_augmented = np.array([1, x])  # Add bias term (x0 = 1)
        y_pred[i] = X_test_augmented @ theta
return y_pred

# Step 3: Generate predictions on a range of test values
X_test = np.linspace(min(X), max(X), 100).reshape(-1, 1)
```

```
# Step 4: Perform Locally Weighted Regression for different tau values
tau_values = [0.1, 0.5, 1.0, 2.0]
plt.figure(figsize=(10, 6))

# Scatter plot of the original data
plt.scatter(X, y, color='blue', alpha=0.5, label='Data Points')

# Perform LWR and plot for different tau values
for tau in tau_values:
    y_pred = locally_weighted_regression(X, y, X_test, tau)
    plt.plot(X_test, y_pred, label=f'LWR (tau={tau})')

# Step 5: Customize the plot
plt.title("Locally Weighted Regression (LWR) on California Housing Data (AveRooms vs. House Value)")
plt.xlabel("Average Number of Rooms (AveRooms)")
plt.ylabel("Median House Value")
plt.legend()
plt.show()
```



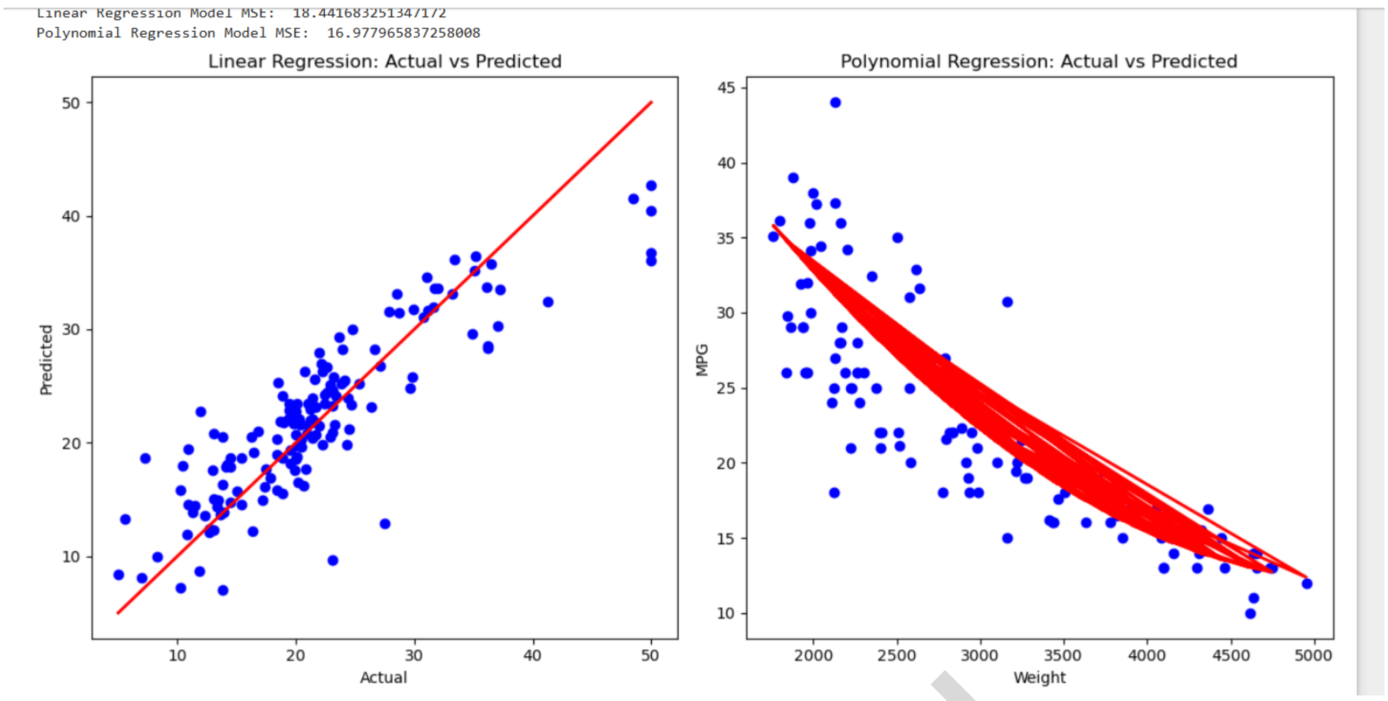
Program 7: Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle


data_url = "http://lib.stat.cmu.edu/datasets/boston"
boston= pd.read_csv(data_url,  sep=r"\s+", skiprows=22, header=None)

data = np.hstack([boston.values[::2, :], boston.values[1::2, :2]])
target = boston.values[1::2, 2]

# ---- Linear Regression with Boston Housing Dataset ----
# Load the Boston Housing Dataset
#boston = load_boston()
X_boston = data.astype(float)  # Features matrix
y_boston = target.astype(float)

# Shuffle and split the data into train and test sets
X_boston, y_boston = shuffle(X_boston, y_boston, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_boston, y_boston, test_size=0.3, random_state=42)
# Initialize and train the linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear_model.predict(X_test)
```

```
# Calculate Mean Squared Error (MSE) for Linear Regression
mse_linear = mean_squared_error(y_test, y_pred_linear)

# ---- Polynomial Regression with Auto MPG Dataset ----
# Load the Auto MPG dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin', 'car_name']
mpg_data = pd.read_csv(url, delim_whitespace=True, header=None, names=column_names)

# Clean the data (handle missing values)
mpg_data = mpg_data.replace("?", np.nan)
mpg_data['horsepower'] = mpg_data['horsepower'].astype(float)
mpg_data = mpg_data.dropna()

# Select features and target for Polynomial Regression
X_mpg = mpg_data[['weight']]  # Using 'weight' as the feature
y_mpg = mpg_data['mpg']       # 'mpg' as the target

# Split into train and test sets
X_train_mpg, X_test_mpg, y_train_mpg, y_test_mpg = train_test_split(X_mpg, y_mpg, test_size=0.3, random_state=42)

# Initialize PolynomialFeatures with degree 3 (cubic polynomial)
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train_mpg)
X_test_poly = poly.transform(X_test_mpg)

# Initialize and train the polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train_mpg)

# Predict on the test set
y_pred_poly = poly_model.predict(X_test_poly)

# Calculate Mean Squared Error (MSE) for Polynomial Regression
```

```
mse_poly = mean_squared_error(y_test_mpg, y_pred_poly)

# ---- Visualization and Results ----

# Linear Regression Results
print("Linear Regression Model MSE: ", mse_linear)

# Plotting Linear Regression results
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')

# Polynomial Regression Results
print("Polynomial Regression Model MSE: ", mse_poly)

# Plotting Polynomial Regression results
plt.subplot(1, 2, 2)
plt.scatter(X_test_mpg, y_test_mpg, color='blue')
plt.plot(X_test_mpg, y_pred_poly, color='red', linewidth=2)
plt.title('Polynomial Regression: Actual vs Predicted')
plt.xlabel('Weight')
plt.ylabel('MPG')

# Show plots
plt.tight_layout()
plt.show()
```

Output:



```
Linear Regression Model MSE:  18.441683251347172
Polynomial Regression Model MSE:  16.977965837258008
```

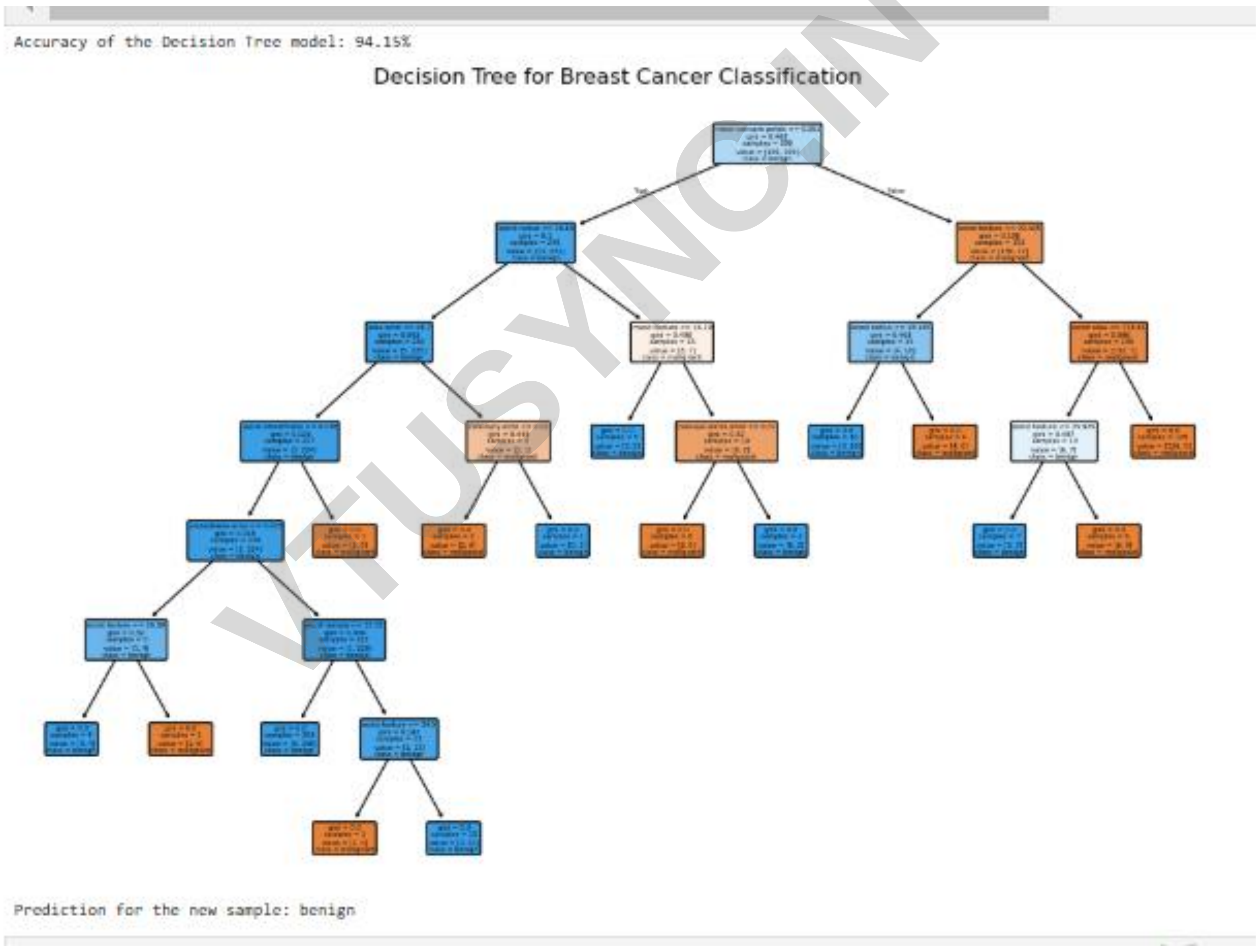
Linear Regression: Actual vs Predicted
Predicted
Actual
Polynomial Regression: Actual vs Predicted
MPG
Weight

Program 8:
Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree

# ---- Load Breast Cancer Dataset ----
# Load the Breast Cancer dataset from sklearn
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Labels (malignant or benign)

# ---- Split the data into train and test sets ----
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# ---- Train the Decision Tree Classifier ----
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# ---- Evaluate the model ----
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Decision Tree model: {accuracy * 100:.2f}%")
# ---- Visualize the Decision Tree ----
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, filled=True, feature_names=data.feature_names,
class_names=data.target_names, rounded=True)
```

```
plt.title("Decision Tree for Breast Cancer Classification")
plt.show()

# ---- Classify a new sample ----
# Let's define a new sample with hypothetical feature values
new_sample = np.array([[15.2, 20.3, 100.1, 0.1, 0.3, 0.2, 0.1, 0.2, 0.3, 0.2, 0.1, 0.4, 0.5, 0.3,
0.1, 1.2, 1.3, 0.7, 0.6, 1.1, 1.1, 0.5, 0.3, 0.1, 1.2, 1.3, 0.5, 1.3, 1.0, 1.5]])  # Example values

# Predict the class for the new sample
prediction = clf.predict(new_sample)
print(f"Prediction for the new sample: {data.target_names[prediction][0]}")
```

Output:



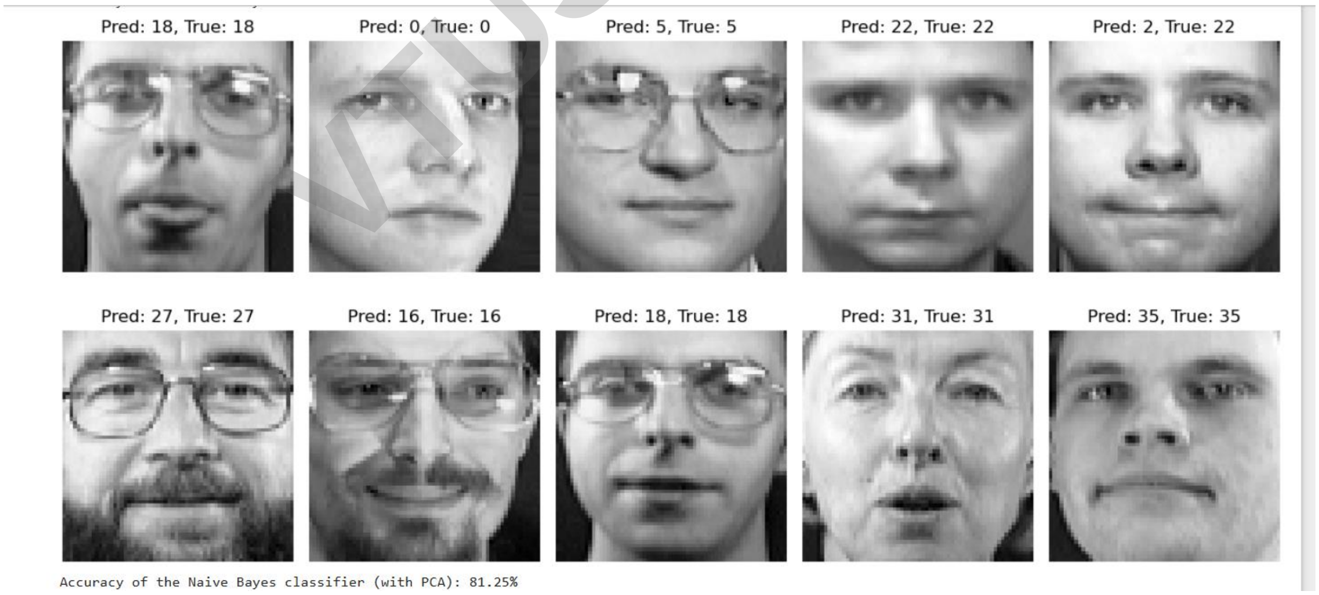
```
Accuracy of the Decision Tree model: 94.15%
```

```
Prediction for the new sample: benign
```

Program 9:

Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA

# ---- Load Olivetti Face Dataset ----
# Load the dataset from sklearn
olivetti_faces = datasets.fetch_olivetti_faces(shuffle=True, random_state=42)
X = olivetti_faces.data  # The 64x64 flattened image data (features)
y = olivetti_faces.target  # The target labels (0-39 for 40 individuals)

# ---- Preprocessing ----
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ---- Train the Naive Bayes Classifier ----
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# ---- Evaluate the Model ----
y_pred = nb_classifier.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Naive Bayes classifier: {accuracy * 100:.2f}%")
# ---- Visualize the results ----
# Plot some of the test images with their predicted labels
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(64, 64), cmap='gray')  # Reshape to 64x64 image
    ax.set_title(f"Pred: {y_pred[i]}, True: {y_test[i]}")
    ax.axis('off')
plt.tight_layout()
plt.show()

# ---- Optional: Dimensionality Reduction for better visualization ----
# Sometimes PCA (Principal Component Analysis) can help improve performance
# when used with Naive Bayes, especially for image data.

# Reduce the dimensions to 100 using PCA
pca = PCA(n_components=100, whiten=True, random_state=42)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train the Naive Bayes classifier again on PCA-reduced data
nb_classifier.fit(X_train_pca, y_train)

# Evaluate the classifier on PCA-reduced test data
y_pred_pca = nb_classifier.predict(X_test_pca)

# Calculate the accuracy with PCA
accuracy_pca = accuracy_score(y_test, y_pred_pca)
print(f"Accuracy of the Naive Bayes classifier (with PCA): {accuracy_pca * 100:.2f}%")
```

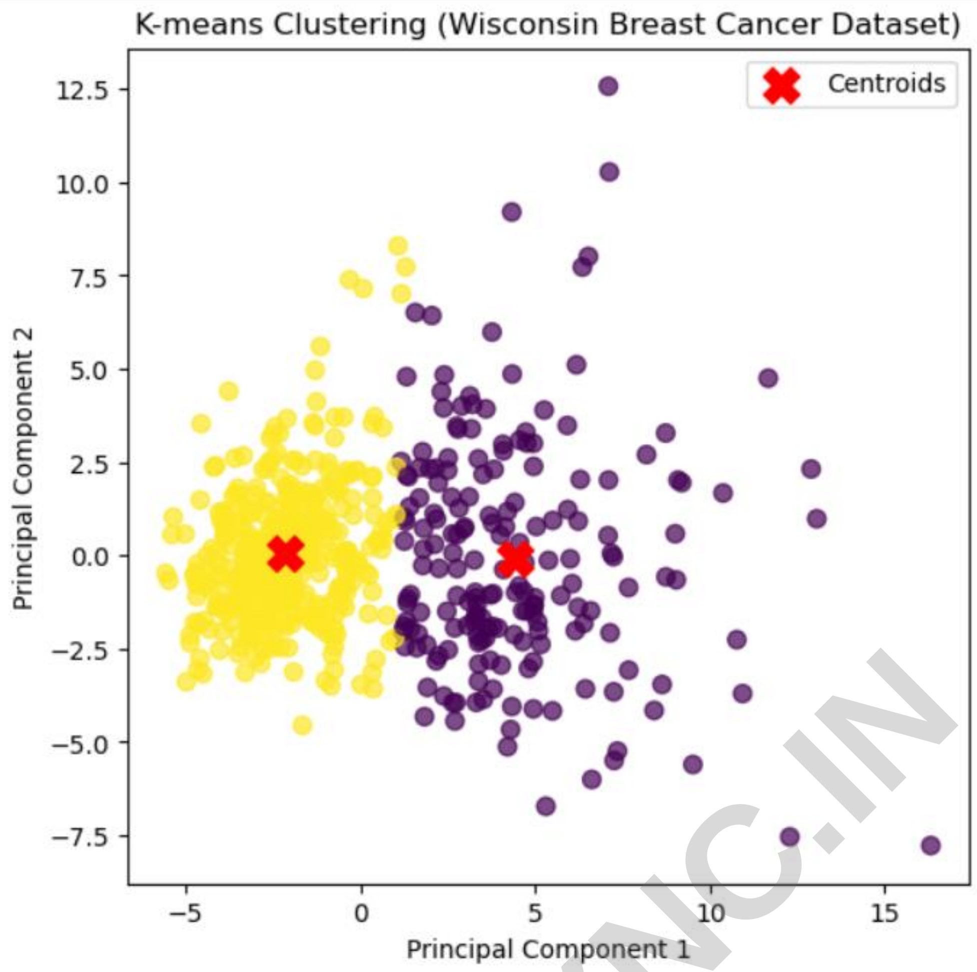


```
Accuracy of the Naive Bayes classifier (with PCA): 81.25%
```

Program 10:
Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

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

# ---- Load Wisconsin Breast Cancer Dataset ----
# Load the dataset from sklearn
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Labels (0: malignant, 1: benign)

# ---- Preprocessing ----
# Standardizing the features to have mean=0 and variance=1
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---- Apply K-means Clustering ----
# Applying K-means with 2 clusters (since breast cancer can be malignant or benign)
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_scaled)

# Get the cluster labels assigned by K-means
y_kmeans = kmeans.predict(X_scaled)

# ---- Visualize the Clustering Result ----
# We will use the first two principal components for visualization (2D plot)
from sklearn.decomposition import PCA
# Reduce the dimensions to 2 for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot the clusters
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis', s=50, alpha=0.7)

# Mark the centroids of the clusters
centers = kmeans.cluster_centers_
centers_pca = pca.transform(centers)

plt.scatter(centers_pca[:, 0], centers_pca[:, 1], c='red', s=200, marker='X',
label='Centroids')

# Add title and labels
plt.title("K-means Clustering (Wisconsin Breast Cancer Dataset)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.show()

# ---- Evaluate the clustering result ----
# We can compare the clustering labels with the actual labels to see how well it
performs
from sklearn.metrics import confusion_matrix, accuracy_score

# Accuracy based on comparison with actual labels
accuracy = accuracy_score(y, y_kmeans)
print(f"Clustering Accuracy: {accuracy * 100:.2f}%")

# Confusion Matrix
cm = confusion_matrix(y, y_kmeans)
print(f"Confusion Matrix:\n{cm}")
```



```
Clustering Accuracy: 91.21%
Confusion Matrix:
[[175  37]
 [ 13 344]]
```

Machine Learning Viva Questions

1. **What is machine learning?**
2. **Define supervised learning**
3. **Define unsupervised learning**
4. **Define semi supervised learning**
5. **Define reinforcement learning**
6. **What do you mean by hypotheses?**
7. **What is classification?**
8. **What is clustering?**
9. **Define precision, accuracy and recall**
10. **Define entropy**
11. **Define regression**
12. **How KNN is different from k-means clustering**
13. **What is concept learning?**
14. **Define specific boundary and general boundary**
15. **Define target function**
16. **Define decision tree**
17. **What is ANN**
18. **Explain gradient descent approximation**
19. **State Bayes theorem**
20. **Define Bayesian belief networks**
21. **Differentiate hard and soft clustering.**
22. **Define variance**
23. **What is inductive machine learning?**
24. **Why K nearest neighbor algorithm is lazy learning algorithm**
25. **Why naïve Bayes is naïve**
26. **Mention classification algorithms**
27. **Define pruning**
28. **Differentiate Clustering and classification**
29. **Mention clustering algorithms**
30. **Define Bias**
31. **What is learning rate? Why it is need**
32. **Explain how a ROC curve works.**
33. **Explain the difference between L1 and L2 regularization.**
34. **What's the difference between a generative and discriminative model?**
35. **What cross-validation technique would you use on a time series dataset?**
36. **What's the F1 score? How would you use it?**

37. How would you handle an imbalanced dataset?
38. When should you use classification over regression?
39.How do you ensure you're not overfitting with a model?
40.Why do we use CNN Model.