

# MACHINE LEARNING

## LAB MANUAL

**ACADEMIC YEAR : 2024-25**

**COURSE CODE : BCSL606**

**CLASS : 6<sup>th</sup> SEM**



**Name of the Student:** \_\_\_\_\_

**USN:** \_\_\_\_\_

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SHRIDEVI INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**Sira Road, NH-4, Maralenahalli, Karnataka 572106**

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		
<b>Course objectives:</b>			
<ul style="list-style-type: none"><li>To become familiar with data and visualize univariate, bivariate, and multivariate data using statistical techniques and dimensionality reduction.</li><li>To understand various machine learning algorithms such as similarity-based learning, regression, decision trees, and clustering.</li><li>To familiarize with learning theories, probability-based models and developing the skills required for decision-making in dynamic environments.</li></ul>			
<b>Sl.NO</b>	<b>Experiments</b>		
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.  <b>Book 1: Chapter 2</b>		
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.  <b>Book 1: Chapter 2</b>		
3	Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.  <b>Book 1: Chapter 2</b>		
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.  <b>Book 1: Chapter 3</b>		
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 $\{x_1, \dots, x_{50}\}$ as follows: if $(x_i \leq 0.5)$ , then $x_i \in \text{Class}_1$ , else $x_i \in \text{Class}_2$ b. Classify the remaining points, $x_{51}, \dots, x_{100}$ using KNN. Perform this for $k=1,2,3,4,5,20,30$  <b>Book 2: Chapter – 2</b>		
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  <b>Book 1: Chapter – 4</b>		
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.  <b>Book 1: Chapter – 5</b>		
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.  <b>Book 2: Chapter – 3</b>		

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.  <b>Book 2: Chapter – 4</b>
10	Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.  <b>Book 2: Chapter – 4</b>
<b>Course outcomes (Course Skill Set):</b> At the end of the course the student will be able to: <ul style="list-style-type: none"> <li>• Illustrate the principles of multivariate data and apply dimensionality reduction techniques.</li> <li>• Demonstrate similarity-based learning methods and perform regression analysis.</li> <li>• Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning.</li> <li>• Implement the clustering algorithms to share computing resources.</li> </ul>	
<b>Assessment Details (both CIE and SEE)</b>  The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together  <b>Continuous Internal Evaluation (CIE):</b> CIE marks for the practical course are <b>50 Marks</b> . The split-up of CIE marks for record/ journal and test are in the ratio <b>60:40</b> . <ul style="list-style-type: none"> <li>• Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.</li> <li>• Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.</li> <li>• Total marks scored by the students are scaled down to <b>30 marks</b> (60% of maximum marks).</li> <li>• Weightage to be given for neatness and submission of record/write-up on time.</li> <li>• Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.</li> <li>• In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.</li> <li>• The suitable rubrics can be designed to evaluate each student's performance and learning ability.</li> <li>• The marks scored shall be scaled down to <b>20 marks</b> (40% of the maximum marks).</li> </ul> The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.	

## 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.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv(r'C:\Users\vijay\Desktop\Machine Learning Course
Batches\FDP_ML_6thSem_VTU\Datasets\housing.csv')
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

	population	households	median_income	median_house_value	ocean_proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	496.0	177.0	7.2574	352100.0	NEAR BAY
3	558.0	219.0	5.6431	341300.0	NEAR BAY
4	565.0	259.0	3.8462	342200.0	NEAR BAY

```
df.shape
```

```
(20640, 10)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 20640 entries, 0 to 20639
```

```
Data columns (total 10 columns):
```

#	Column	Non-Null Count	Dtype
0	longitude	20640 non-null	float64
1	latitude	20640 non-null	float64
2	housing_median_age	20640 non-null	float64
3	total_rooms	20640 non-null	float64
4	total_bedrooms	20433 non-null	float64
5	population	20640 non-null	float64
6	households	20640 non-null	float64

```
7  median_income      20640 non-null  float64
8  median_house_value  20640 non-null  float64
9  ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
df.nunique()
```

```
longitude      844
latitude       862
housing_median_age    52
total_rooms     5926
total_bedrooms    1923
population     3888
households     1815
median_income   12928
median_house_value  3842
ocean_proximity      5
dtype: int64
```

### *Data Cleaning*

```
df.isnull().sum()
```

```
longitude      0
latitude       0
housing_median_age    0
total_rooms     0
total_bedrooms    207
population     0
households     0
median_income   0
median_house_value  0
ocean_proximity  0
dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

```
df['total_bedrooms'].median()
```

```
435.0
```

```
# Handling missing values
```

```
df['total_bedrooms'].fillna(df['total_bedrooms'].median(), inplace=True)
```

### *Feature Engineering*

```
for i in df.iloc[:,2:7]:
    df[i] = df[i].astype('int')
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-122.23	37.88	41	880	129	
1	-122.22	37.86	21	7099	1106	
2	-122.24	37.85	52	1467	190	
3	-122.25	37.85	52	1274	235	
4	-122.25	37.85	52	1627	280	

	population	households	median_income	median_house_value	ocean_proximity	
0	322	126	8.3252	452600.0	NEAR BAY	
1	2401	1138	8.3014	358500.0	NEAR BAY	
2	496	177	7.2574	352100.0	NEAR BAY	
3	558	219	5.6431	341300.0	NEAR BAY	
4	565	259	3.8462	342200.0	NEAR BAY	

## Disciptive Statistics

```
df.describe().T
```

	count	mean	std	min	\
longitude	20640.0	-119.569704	2.003532	-124.3500	
latitude	20640.0	35.631861	2.135952	32.5400	
housing_median_age	20640.0	28.639486	12.585558	1.0000	
total_rooms	20640.0	2635.763081	2181.615252	2.0000	
total_bedrooms	20640.0	536.838857	419.391878	1.0000	
population	20640.0	1425.476744	1132.462122	3.0000	
households	20640.0	499.539680	382.329753	1.0000	
median_income	20640.0	3.870671	1.899822	0.4999	
median_house_value	20640.0	206855.816909	115395.615874	14999.0000	

	25%	50%	75%	max
longitude	-121.8000	-118.4900	-118.01000	-114.3100
latitude	33.9300	34.2600	37.71000	41.9500
housing_median_age	18.0000	29.0000	37.00000	52.0000
total_rooms	1447.7500	2127.0000	3148.00000	39320.0000
total_bedrooms	297.0000	435.0000	643.25000	6445.0000
population	787.0000	1166.0000	1725.00000	35682.0000
households	280.0000	409.0000	605.00000	6082.0000
median_income	2.5634	3.5348	4.74325	15.0001
median_house_value	119600.0000	179700.0000	264725.00000	500001.0000

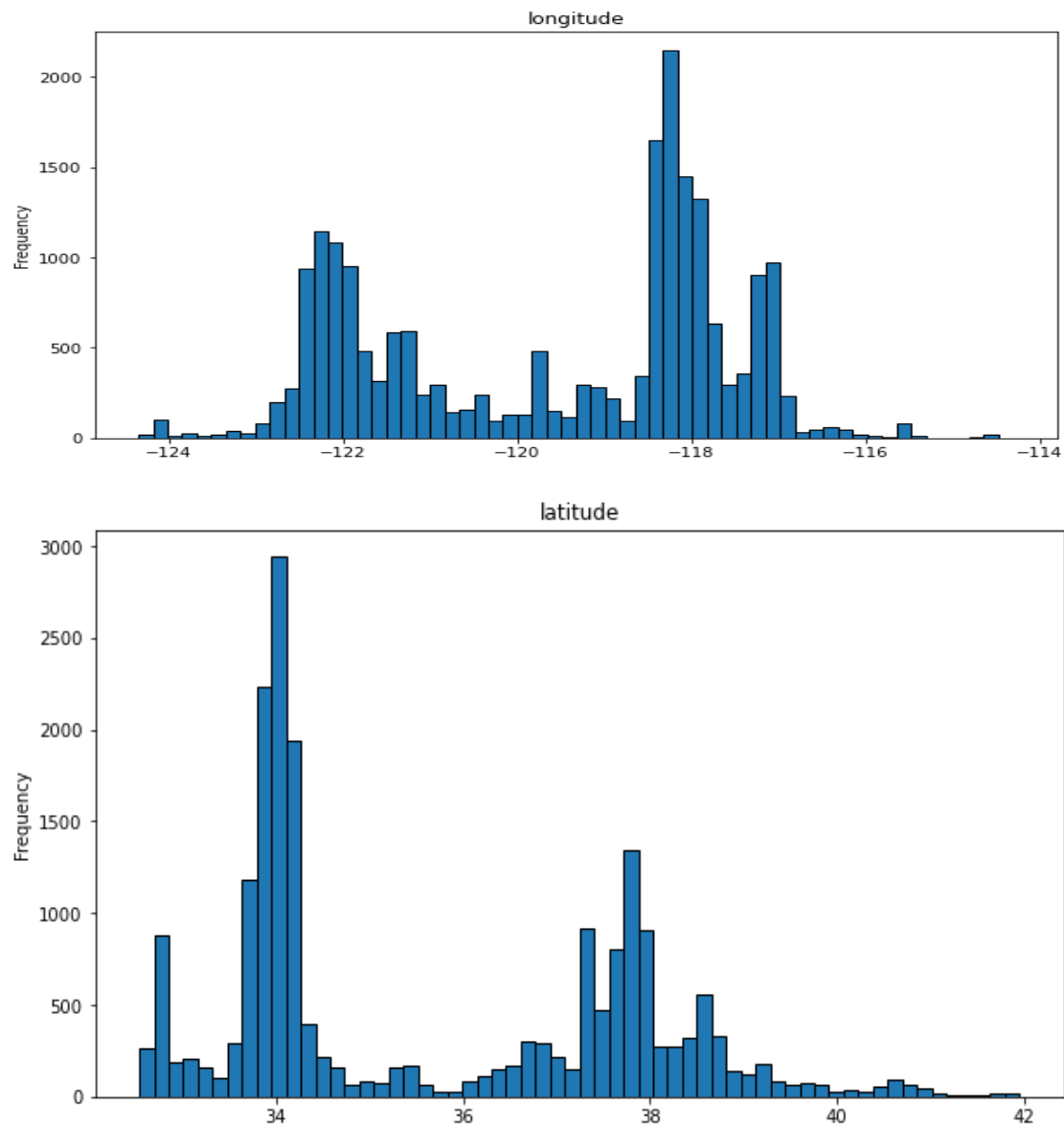
```
Numerical = df.select_dtypes(include=[np.number]).columns
print(Numerical)
```

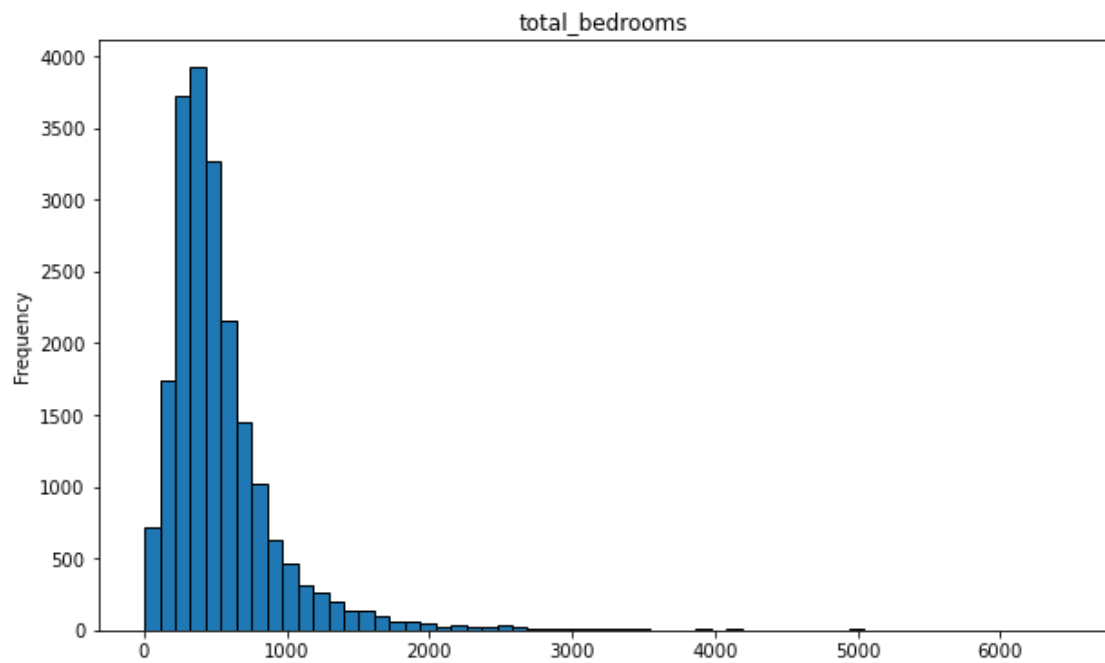
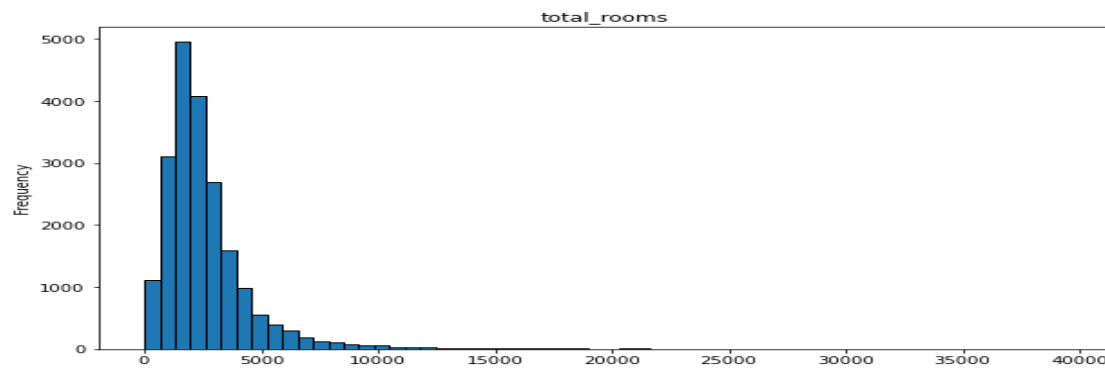
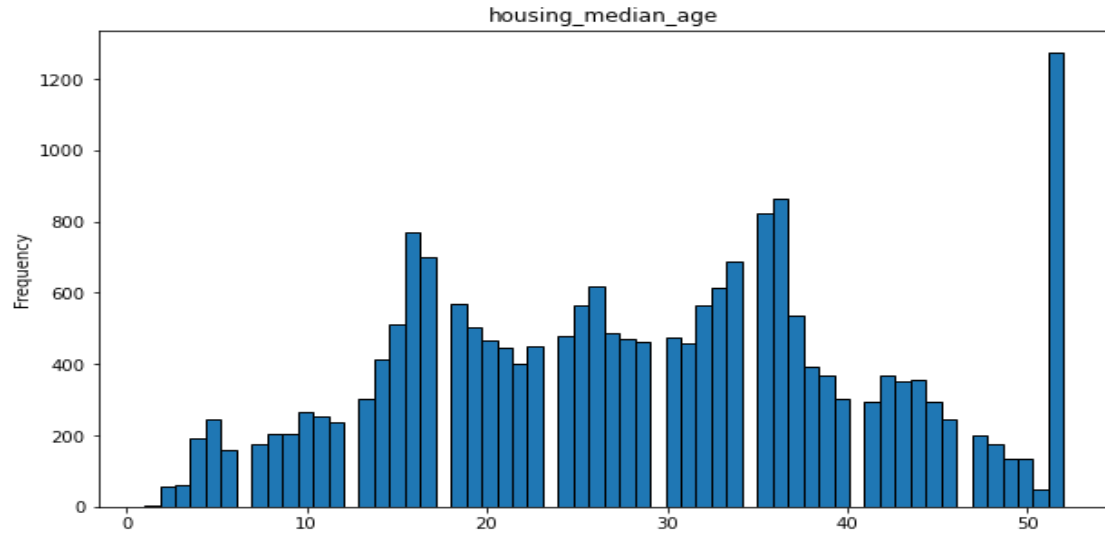
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
      'total_bedrooms', 'population', 'households', 'median_income',
      'median_house_value'],
      dtype='object')
```

## Uni-Variate Analysis

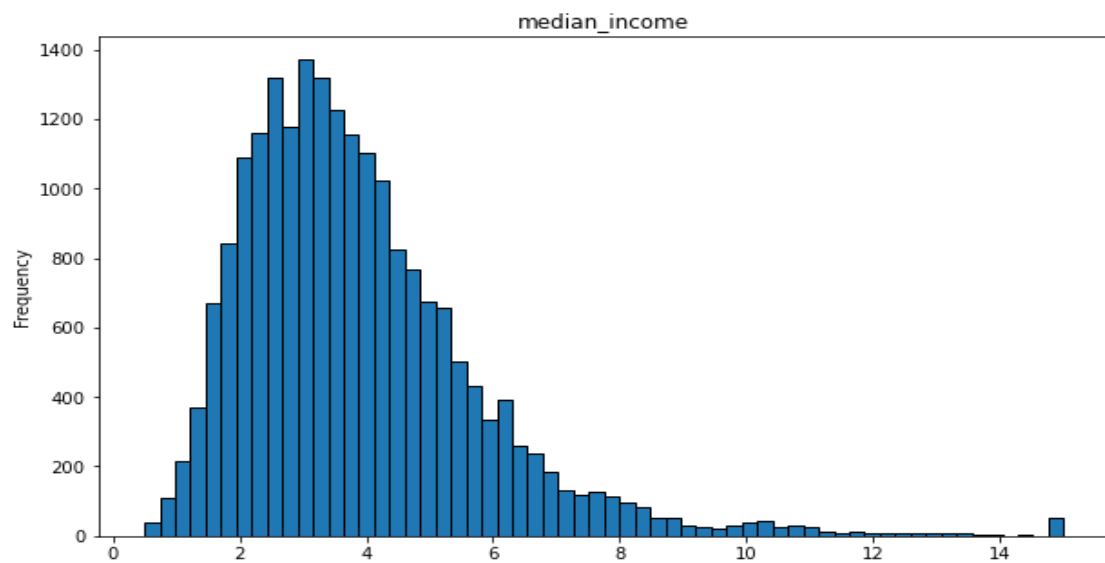
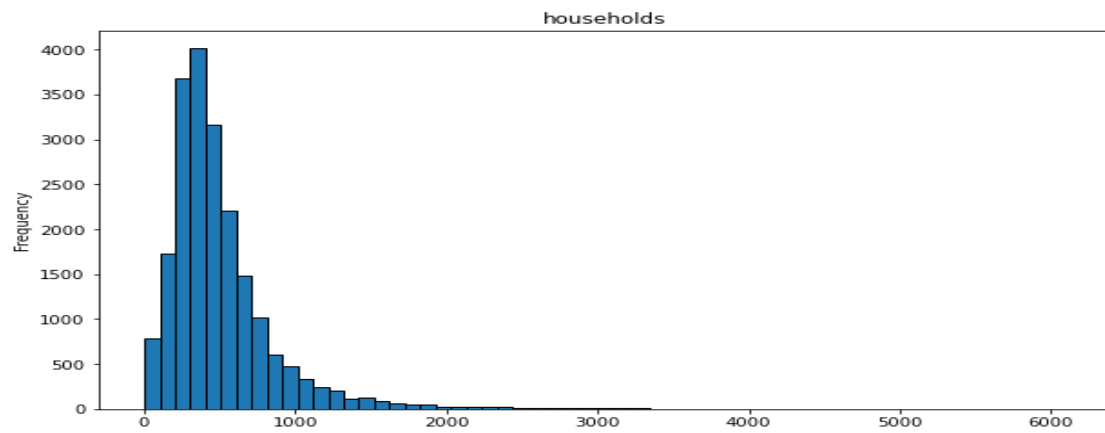
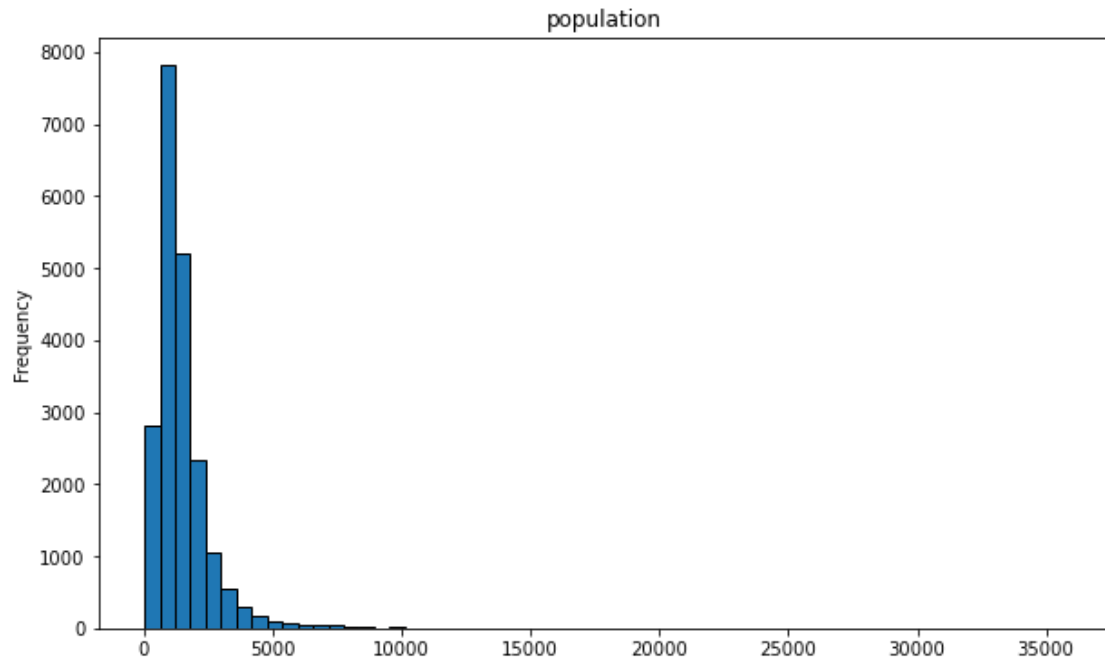
```
for col in Numerical:
    plt.figure(figsize=(10, 6))
```

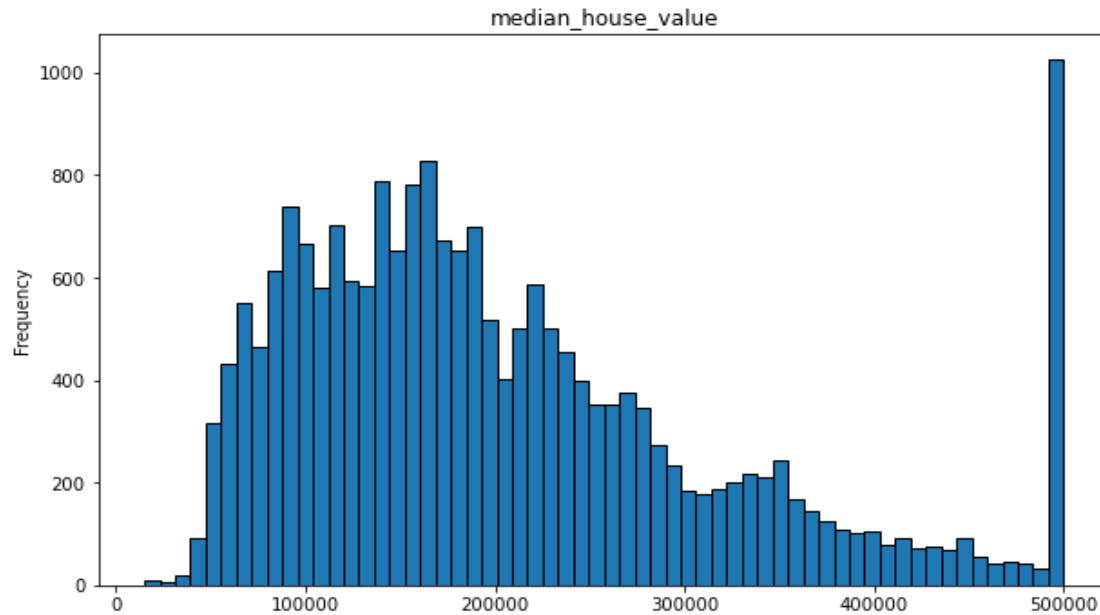
```
df[col].plot(kind='hist', title=col, bins=60, edgecolor='black')  
plt.ylabel('Frequency')  
plt.show()
```







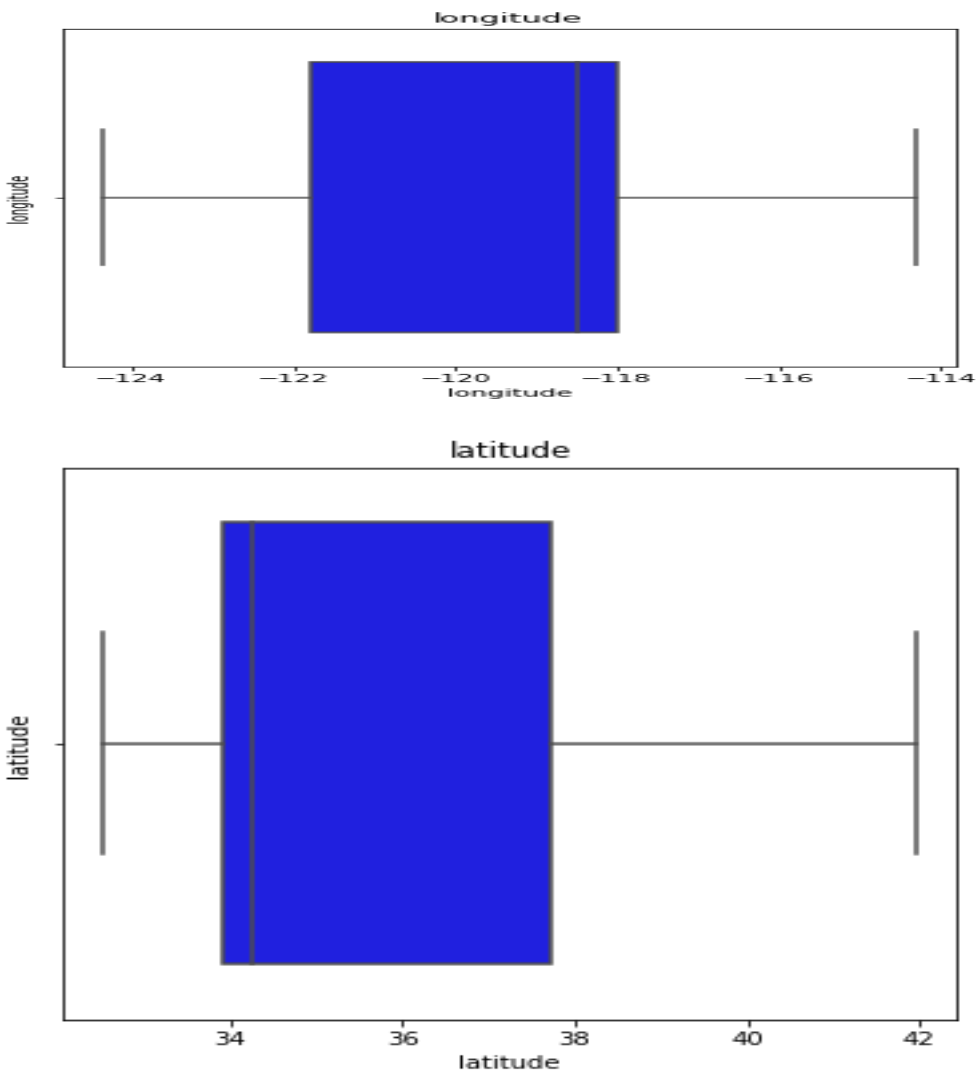


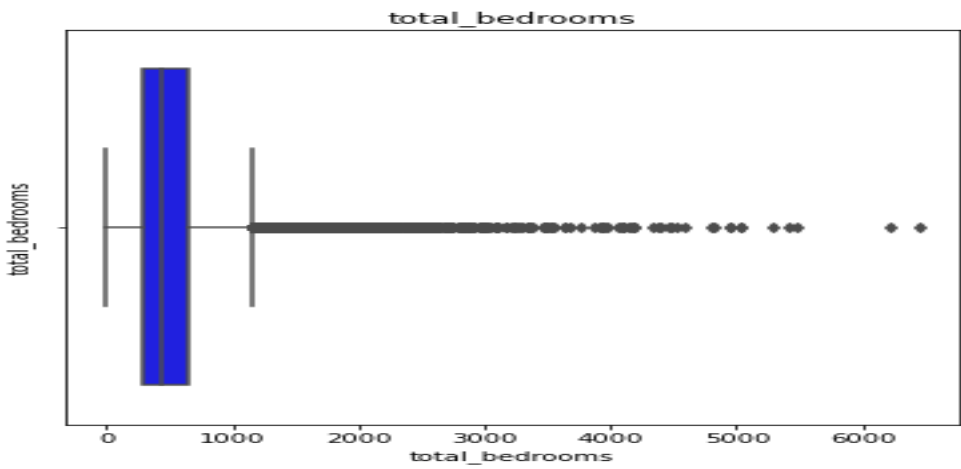
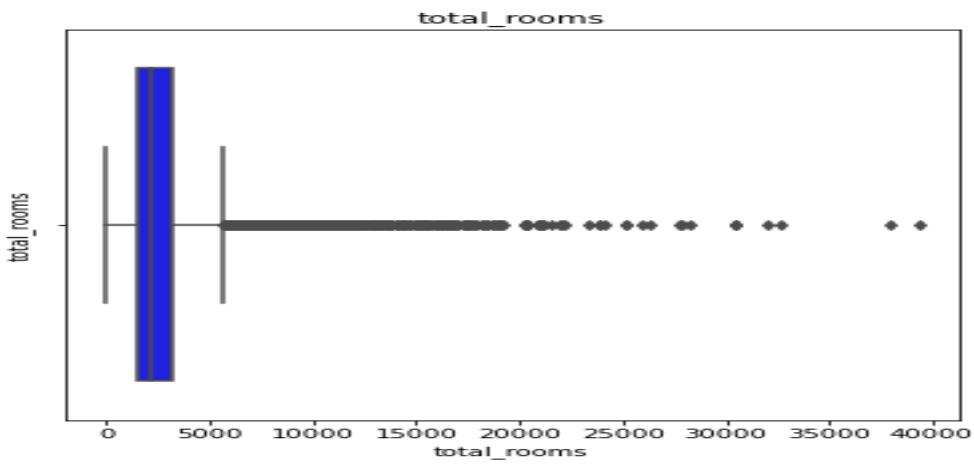
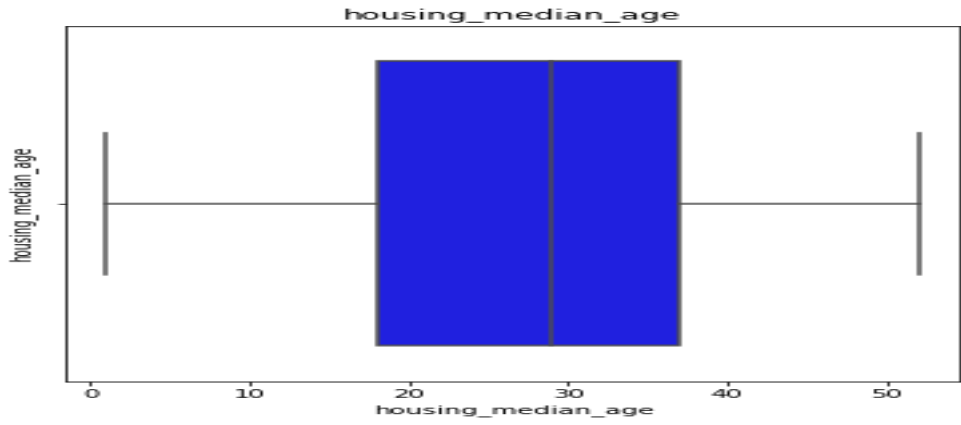


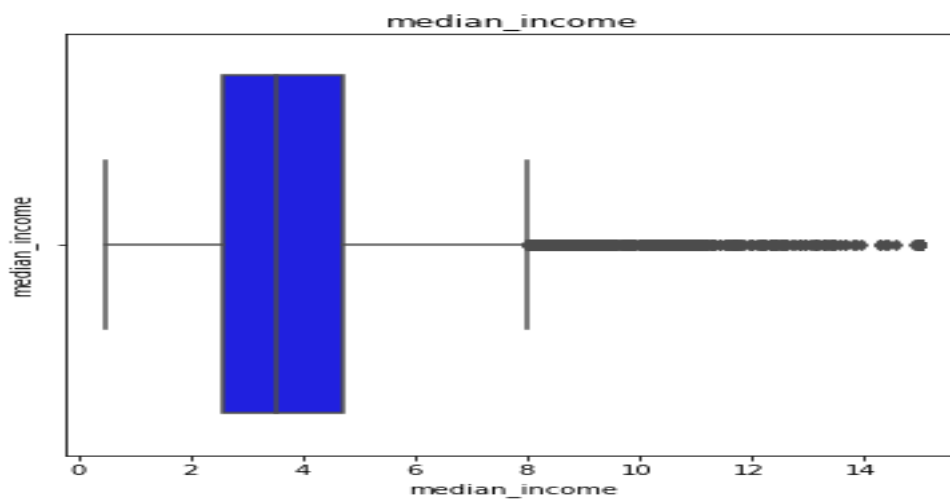
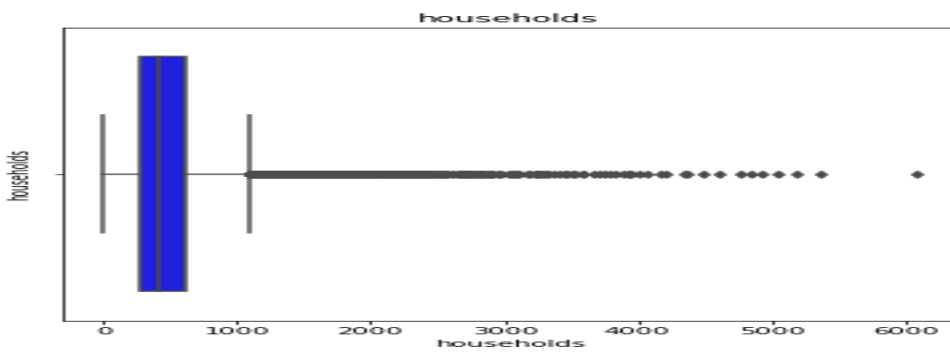
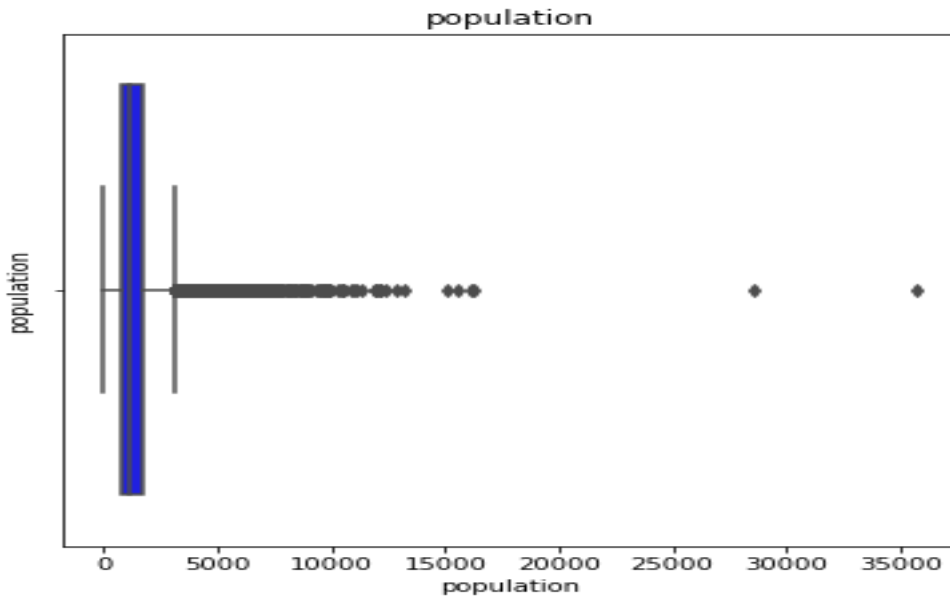
1. Longitude:
2. The dataset contains houses located in specific regions (possibly coastal areas or urban zones) as indicated by the bimodal peaks. Houses are not uniformly distributed across all longitudes.
3. Latitude:
4. Similar to longitude, the latitude distribution shows houses concentrated in particular zones. This suggests geographic clustering, possibly around major cities.
5. Housing Median Age:
6. Most houses are relatively older, with the majority concentrated in a specific range of median ages. This might imply that housing development peaked during certain decades.
7. Total Rooms:
8. The highly skewed distribution shows most houses have a lower total number of rooms. A few properties with a very high number of rooms could represent outliers (e.g., mansions or multi-unit buildings).
9. Median Income:
10. Most households fall within a low-to-mid income bracket. The steep decline after the peak suggests a small proportion of high-income households in the dataset.
11. Population:

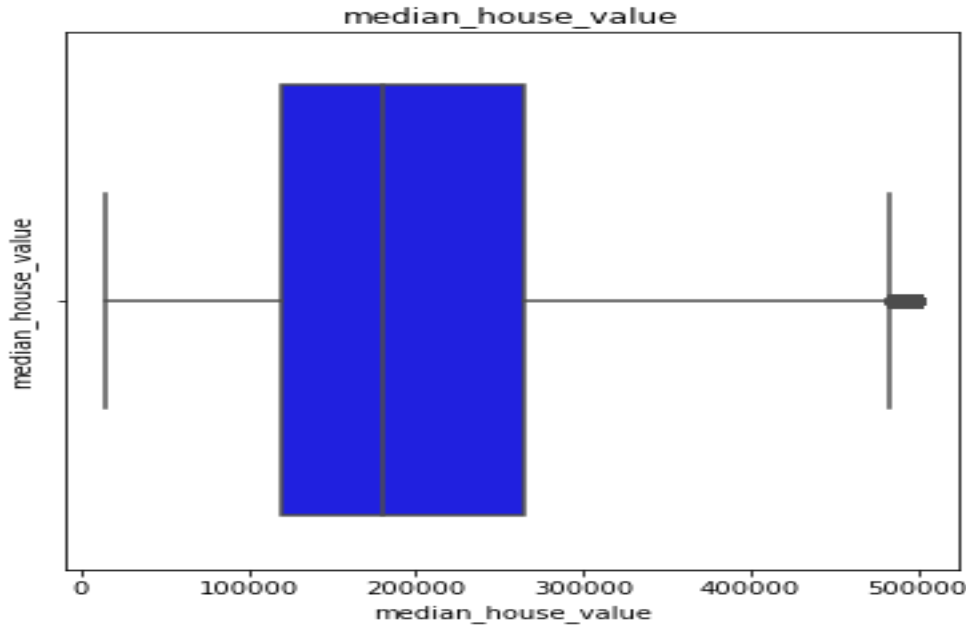
12. Most areas in the dataset have a relatively low population. However, there are some highly populated areas, as evidenced by the long tail. These may represent urban centers.
13. Median House Value:
14. The sharp peak at the end of the histogram suggests that house prices in the dataset are capped at a maximum value, which could limit the variability in predictions.

```
for col in Numerical:  
    plt.figure(figsize=(6, 6))  
  
    sns.boxplot(df[col], color='blue')  
    plt.title(col)  
    plt.ylabel(col)  
  
    plt.show()
```









#### Outlier Analysis for Each Feature:

1. Total Rooms: There are numerous data points above the upper whisker, indicating a significant number of outliers.
2. Total Bedrooms: Numerous data points above the upper whisker indicate a significant presence of outliers with very high total\_bedrooms values.
3. Population: There are numerous outliers above the upper whisker, with extreme population values reaching beyond 35,000.
4. Households There is a significant number of outliers above the upper whisker. These values represent areas with an unusually high number of households.
5. Median Income: There are numerous data points above the upper whisker, marked as circles. These are considered potential outliers.
6. Median House Value: A small cluster of outliers is visible near the maximum value of 500,000.

#### General Actions for Outlier Handling:

1. Transformation: Apply log or square root transformations to reduce skewness for features like total rooms, population, and median income.
2. Removal: If outliers are due to data errors or are not relevant, consider removing them.

## 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.

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

# Load California Housing dataset
data = fetch_california_housing()

# Convert to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target # Adding the target variable (median house value)

# Table of Meaning of Each Variable
variable_meaning = {
    "MedInc": "Median income in block group",
    "HouseAge": "Median house age in block group",
    "AveRooms": "Average number of rooms per household",
    "AveBedrms": "Average number of bedrooms per household",
    "Population": "Population of block group",
    "AveOccup": "Average number of household members",
    "Latitude": "Latitude of block group",
    "Longitude": "Longitude of block group",
    "Target": "Median house value (in $100,000s)"
}

variable_df = pd.DataFrame(list(variable_meaning.items()),
                           columns=["Feature", "Description"])
print("\nVariable Meaning Table:")
print(variable_df)
```

## Variable Meaning Table:

	Feature	Description
0	MedInc	Median income in block group
1	HouseAge	Median house age in block group
2	AveRooms	Average number of rooms per household
3	AveBedrms	Average number of bedrooms per household
4	Population	Population of block group
5	AveOccup	Average number of household members
6	Latitude	Latitude of block group

```

7 Longitude Longitude of block group
8 Target Median house value (in $100,000s)

```

### # Basic Data Exploration

```

print("\nBasic Information about Dataset:")
print(df.info()) # Overview of dataset
print("\nFirst Five Rows of Dataset:")
print(df.head()) # Display first few rows

```

Basic Information about Dataset:

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 20640 entries, 0 to 20639

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	MedInc	20640 non-null	float64
1	HouseAge	20640 non-null	float64
2	AveRooms	20640 non-null	float64
3	AveBedrms	20640 non-null	float64
4	Population	20640 non-null	float64
5	AveOccup	20640 non-null	float64
6	Latitude	20640 non-null	float64
7	Longitude	20640 non-null	float64
8	Target	20640 non-null	float64

dtypes: float64(9)

memory usage: 1.4 MB

None

First Five Rows of Dataset:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Longitude Target

0	-122.23	4.526
1	-122.22	3.585
2	-122.24	3.521
3	-122.25	3.413
4	-122.25	3.422

### # Summary Statistics

```

print("\nSummary Statistics:")
print(df.describe()) # Summary statistics of dataset

```

Summary Statistics:



	MedInc	HouseAge	AveRooms	AveBedrms	Population
\					
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744
std	1.899822	12.585558	2.474173	0.473911	1132.462122
min	0.499900	1.000000	0.846154	0.333333	3.000000
25%	2.563400	18.000000	4.440716	1.006079	787.000000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000
max	15.000100	52.000000	141.909091	34.066667	35682.000000

	AveOccup	Latitude	Longitude	Target
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.070655	35.631861	-119.569704	2.068558
std	10.386050	2.135952	2.003532	1.153956
min	0.692308	32.540000	-124.350000	0.149990
25%	2.429741	33.930000	-121.800000	1.196000
50%	2.818116	34.260000	-118.490000	1.797000
75%	3.282261	37.710000	-118.010000	2.647250
max	1243.333333	41.950000	-114.310000	5.000010

#### # Explanation of Summary Statistics

```
summary_explanation = """
```

The summary statistics table provides key percentiles and other descriptive metrics for each numerical feature:

- **\*\*25% (First Quartile - Q1):\*\*** This represents the value below which 25% of the data falls. It helps in understanding the lower bound of typical data values.
- **\*\*50% (Median - Q2):\*\*** This is the middle value when the data is sorted. It provides the central tendency of the dataset.
- **\*\*75% (Third Quartile - Q3):\*\*** This represents the value below which 75% of the data falls. It helps in identifying the upper bound of typical values in the dataset.
- These percentiles are useful for detecting skewness, data distribution, and identifying potential outliers (values beyond  $Q1 - 1.5 \times IQR$  or  $Q3 + 1.5 \times IQR$ ).

```
print("\nSummary Statistics Explanation:")
```

```
print(summary_explanation)
```

#### Summary Statistics Explanation:

The summary statistics table provides key percentiles and other descriptive metrics for each numerical feature:

- **\*\*25% (First Quartile - Q1):\*\*** This represents the value below which 25% of the data falls. It helps in understanding the lower bound of typical data values.
- **\*\*50% (Median - Q2):\*\*** This is the middle value when the data is sorted. It provides the central tendency of the dataset.
- **\*\*75% (Third Quartile - Q3):\*\*** This represents the value below which 75% of the data falls. It helps in identifying the upper bound of typical values in

the dataset.

- These percentiles are useful for detecting skewness, data distribution, and identifying potential outliers (values beyond  $Q1 - 1.5 \cdot IQR$  or  $Q3 + 1.5 \cdot IQR$ ).

*# Check for missing values*

```
print("\nMissing Values in Each Column:")  
print(df.isnull().sum()) # Count of missing values
```

Missing Values in Each Column:

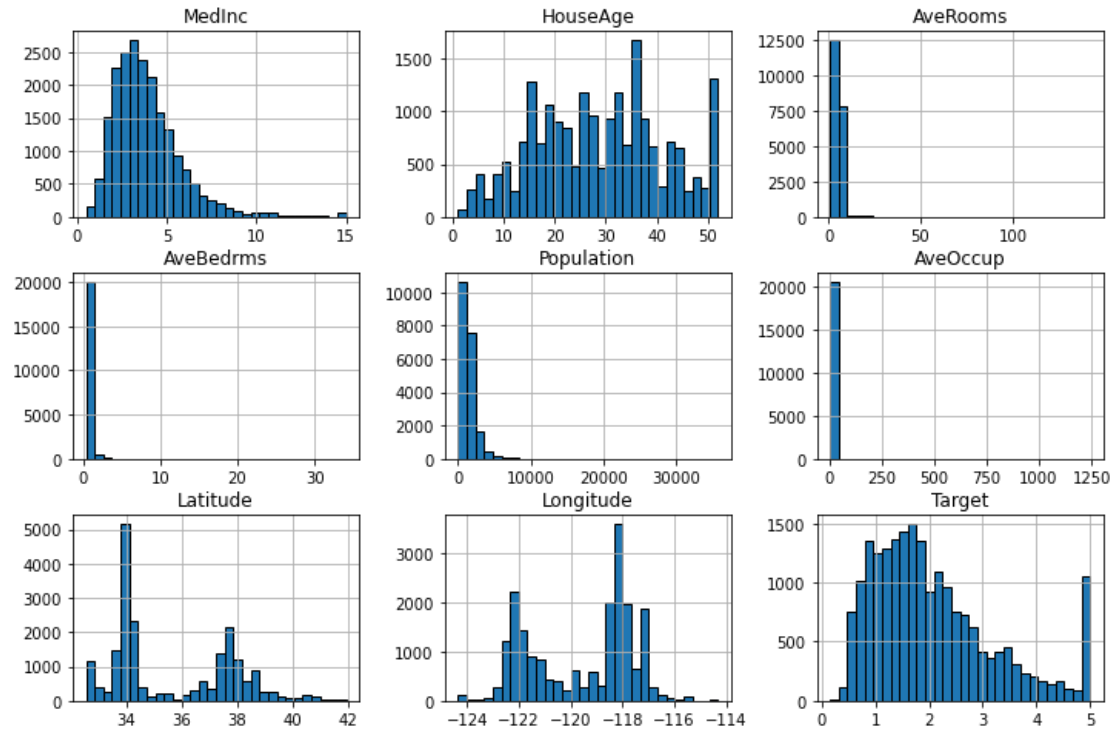
```
MedInc      0  
HouseAge    0  
AveRooms    0  
AveBedrms   0  
Population  0  
AveOccup    0  
Latitude    0  
Longitude   0  
Target      0  
dtype: int64
```

*# Histograms for distribution of features*

```
plt.figure(figsize=(12, 8))  
df.hist(figsize=(12, 8), bins=30, edgecolor='black')  
plt.suptitle("Feature Distributions", fontsize=16)  
plt.show()
```

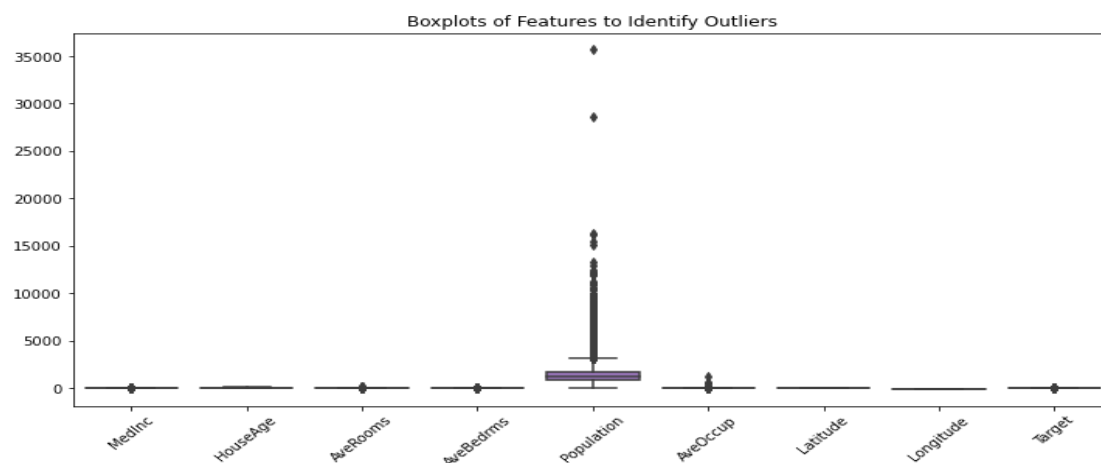
<Figure size 864x576 with 0 Axes>

Feature Distributions



*# Boxplots for outlier detection*

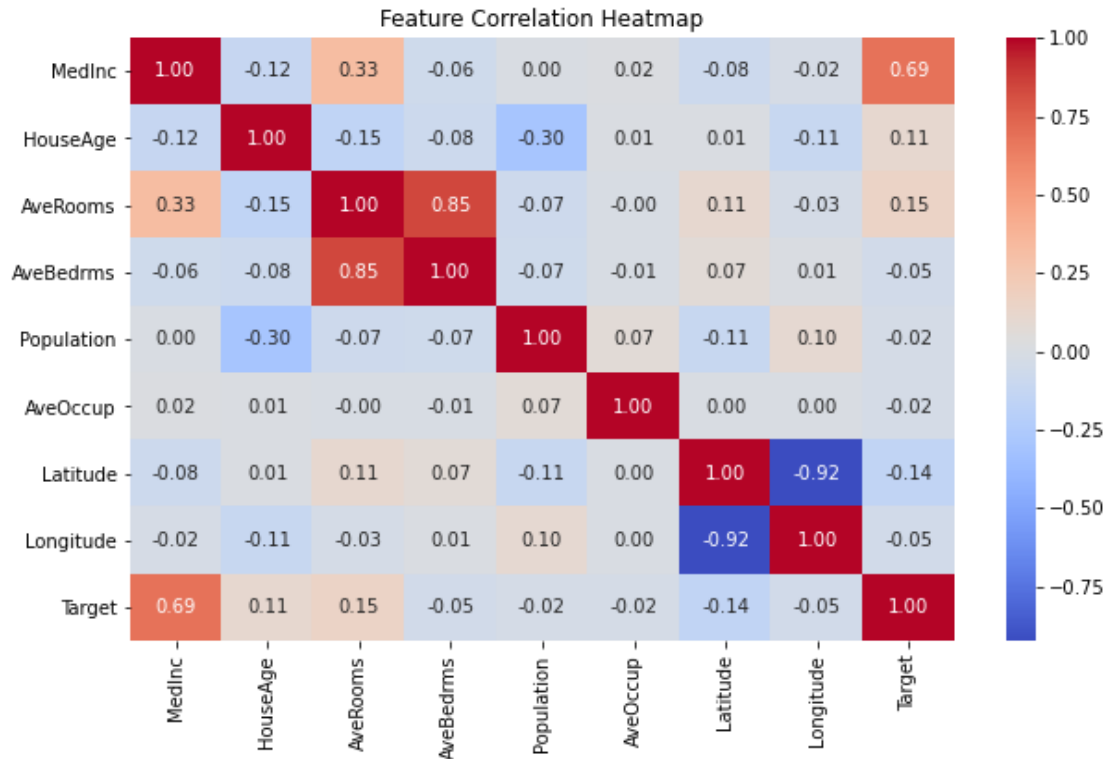
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplots of Features to Identify Outliers")
plt.show()
```



*# Correlation Matrix*

```
plt.figure(figsize=(10, 6))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
```

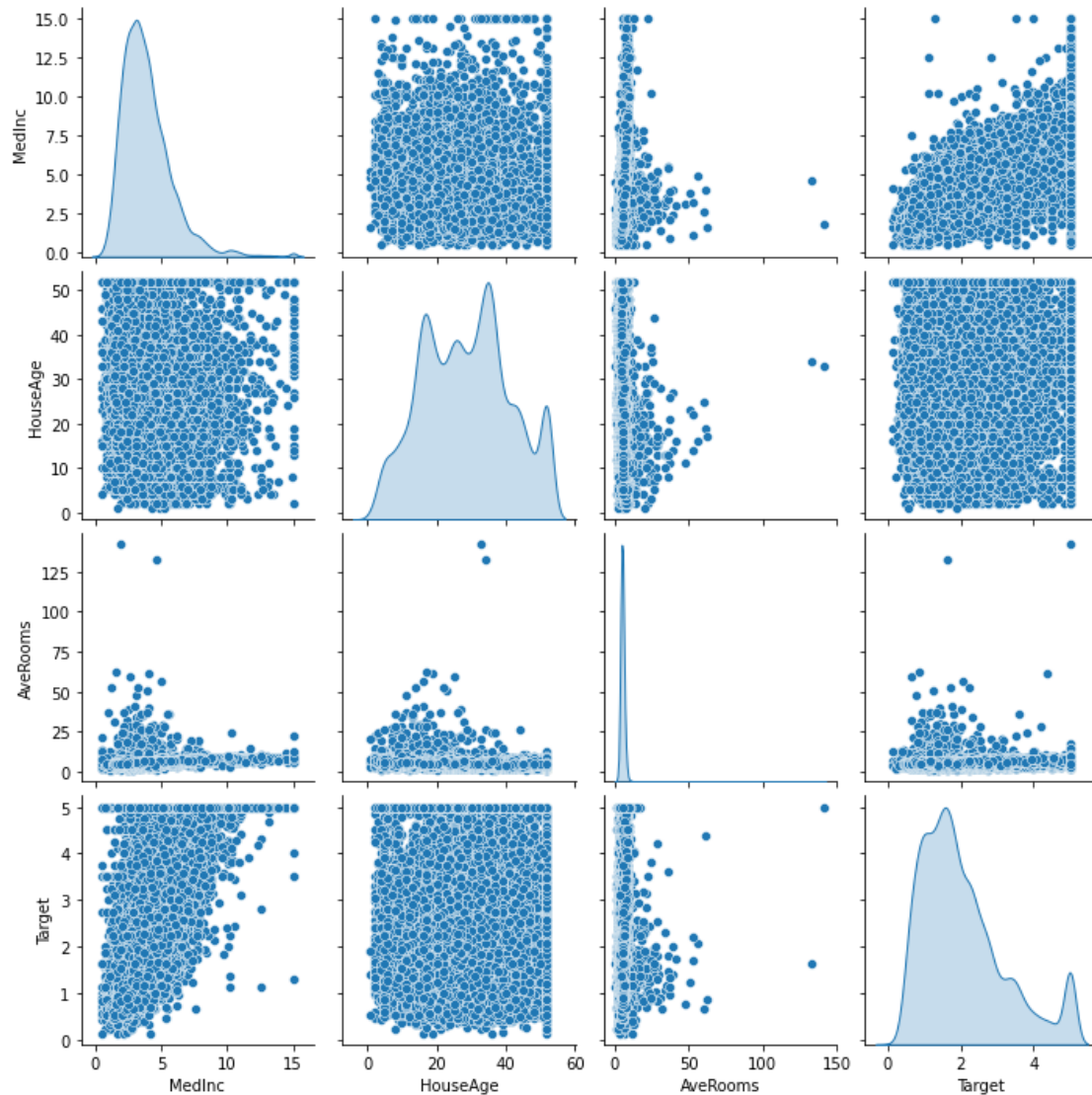
```
plt.title("Feature Correlation Heatmap")
plt.show()
```



```
# Pairplot to analyze feature relationships (only a subset for clarity)
sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'Target']],
diag_kind='kde')
plt.show()
```

```
# Insights from Data Exploration
```

```
print("\nKey Insights:")
print("1. The dataset has", df.shape[0], "rows and", df.shape[1], "columns.")
print("2. No missing values were found in the dataset.")
print("3. Histograms show skewed distributions in some features like 'MedInc'.")
print("4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.")
print("5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.")
```



#### Key Insights:

1. The dataset has 20640 rows and 9 columns.
2. No missing values were found in the dataset.
3. Histograms show skewed distributions in some features like 'MedInc'.
4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.
5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.

### Program 3

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

*the Iris dataset from 4 features to 2.*

```
# Introduction to the Iris Dataset
# The Iris dataset is one of the most well-known datasets in machine learning
and statistics.
# It contains 150 samples of iris flowers categorized into three species:
Setosa, Versicolor, and Virginica.

#
# The goal of using PCA in this exercise is to reduce these four features
into two principal components.
# This will help in visualizing the data better and understanding its
underlying structure.
#
# Since humans struggle to visualize data in more than three dimensions,
reducing the data to 2D allows us to
# retain the most important patterns while making it easier to interpret. PCA
helps us achieve this while
# preserving as much variance as possible.
```

### Explanation of Features in the Iris Dataset

The Iris dataset consists of 4 features, which represent different physical characteristics of iris flowers:

Sepal Length (cm)  
Sepal Width (cm)  
Petal Length (cm)  
Petal Width (cm)

These features were chosen because they effectively differentiate between the three iris species (Setosa, Versicolor, and Virginica).

In the 3D visualizations, we select three features for plotting, which are:

Feature 1 → Sepal Length  
Feature 2 → Sepal Width  
Feature 3 → Petal Length

These features are chosen arbitrarily for visualization, but all four features are used in the PCA computation. Why is the Iris Dataset Important?

The Iris dataset is a benchmark dataset in machine learning because:

It is small yet diverse, making it easy to analyze.  
It has clearly separable classes, which makes it ideal for classification tasks.  
It is preloaded in Scikit-learn, making it accessible for learning and experimentation.

Since the dataset contains three classes (Setosa, Versicolor, and Virginica), PCA helps visualize how well the classes can be separated in a lower-dimensional space.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load the Iris Dataset
iris = datasets.load_iris()
X = iris.data # Extracting feature matrix (4D data)
y = iris.target # Extracting Labels (0, 1, 2 representing three iris species)

# Step 2: Standardizing the Data
# PCA works best when data is standardized (mean = 0, variance = 1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Calculating Covariance Matrix and Eigenvalues/Eigenvectors
# The foundation of PCA is eigen decomposition of the covariance matrix
cov_matrix = np.cov(X_scaled.T)
print(cov_matrix)
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)

# Step 4: Visualizing Data in 3D before PCA
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
colors = ['red', 'green', 'blue']
labels = iris.target_names
for i in range(len(colors)):
    ax.scatter(X_scaled[y == i, 0], X_scaled[y == i, 1], X_scaled[y == i, 2],
              color=colors[i], label=labels[i])
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
ax.set_title('3D Visualization of Iris Data Before PCA')
```

```

plt.legend()
plt.show()

# Step 5: Applying PCA using SVD (Singular Value Decomposition)
# PCA internally relies on SVD, which decomposes a matrix into three parts:
# U, S, and V
U, S, Vt = np.linalg.svd(X_scaled, full_matrices=False)
print("Singular Values:", S)

# Step 6: Applying PCA to Reduce Dimensionality to 2D
# We reduce 4D data to 2D for visualization while retaining maximum variance
pca = PCA(n_components=2) # We choose 2 components because we want to
visualize
X_pca = pca.fit_transform(X_scaled) # Transform data into principal
components

# Step 7: Understanding Variance Explained
# PCA provides the percentage of variance retained in each principal
component
explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance by PC1: {explained_variance[0]:.2f}")
print(f"Explained Variance by PC2: {explained_variance[1]:.2f}")

# Step 8: Visualizing the Transformed Data
# We plot the 2D representation of the Iris dataset after PCA transformation
plt.figure(figsize=(8, 6))
for i in range(len(colors)):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], color=colors[i],
label=labels[i])

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Iris Dataset (Dimensionality Reduction)')
plt.legend()
plt.grid()
plt.show()

# Step 9: Visualizing Eigenvectors Superimposed on 3D Data
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
for i in range(len(colors)):
    ax.scatter(X_scaled[y == i, 0], X_scaled[y == i, 1], X_scaled[y == i, 2],
color=colors[i], label=labels[i])
for i in range(3): # Plot first three eigenvectors
    ax.quiver(0, 0, 0, eigenvectors[i, 0], eigenvectors[i, 1],
eigenvectors[i, 2], color='black', length=1)
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
ax.set_title('3D Data with Eigenvectors')

```



```
plt.legend()  
plt.show()
```

*# Recap:*

*# - The Iris dataset is historically important for testing classification models.*

*# - We standardized the data to ensure fair comparison across features.*

*# - We calculated the covariance matrix, eigenvalues, and eigenvectors.*

*# - PCA is built on SVD, which decomposes data into important components.*

*# - We visualized the original 3D data and superimposed eigenvectors.*

*# - We applied PCA to reduce the dimensionality from 4D to 2D.*

*# - Finally, we visualized the transformed data in 2D space.*

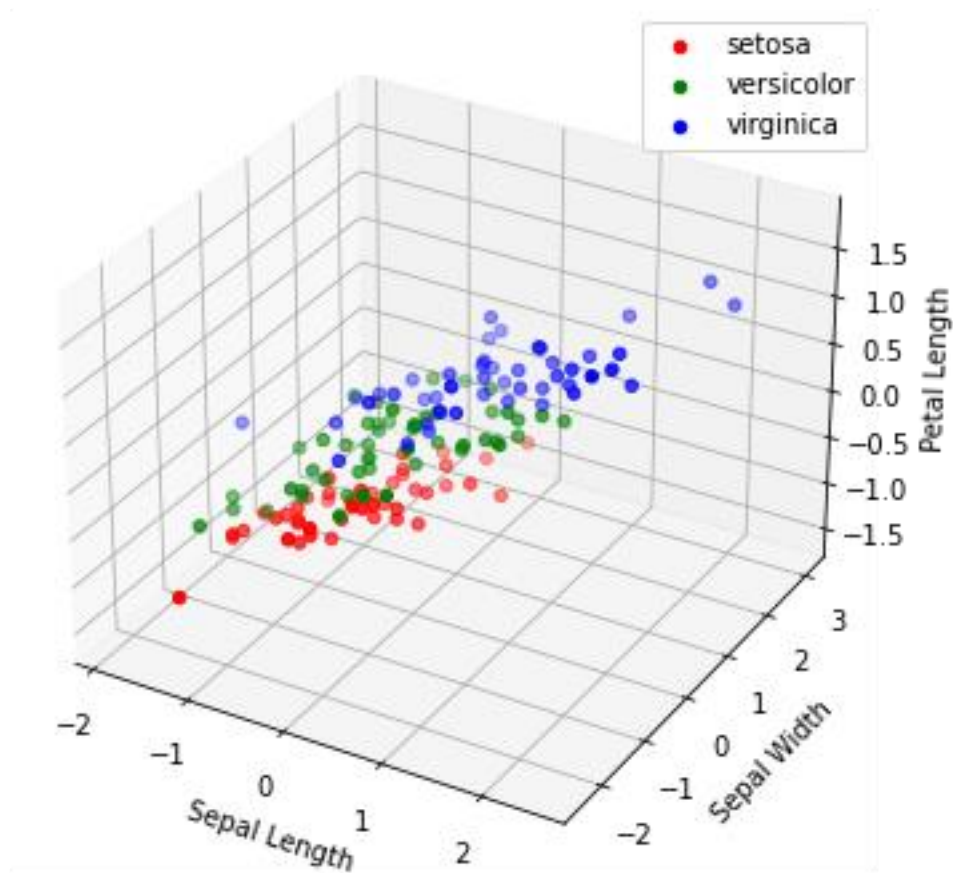
```
[[ 1.00671141 -0.11835884  0.87760447  0.82343066]  
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]  
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]  
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]
```

```
Eigenvalues: [2.93808505 0.9201649  0.14774182 0.02085386]
```

```
Eigenvectors:
```

```
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]  
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]  
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]  
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]
```

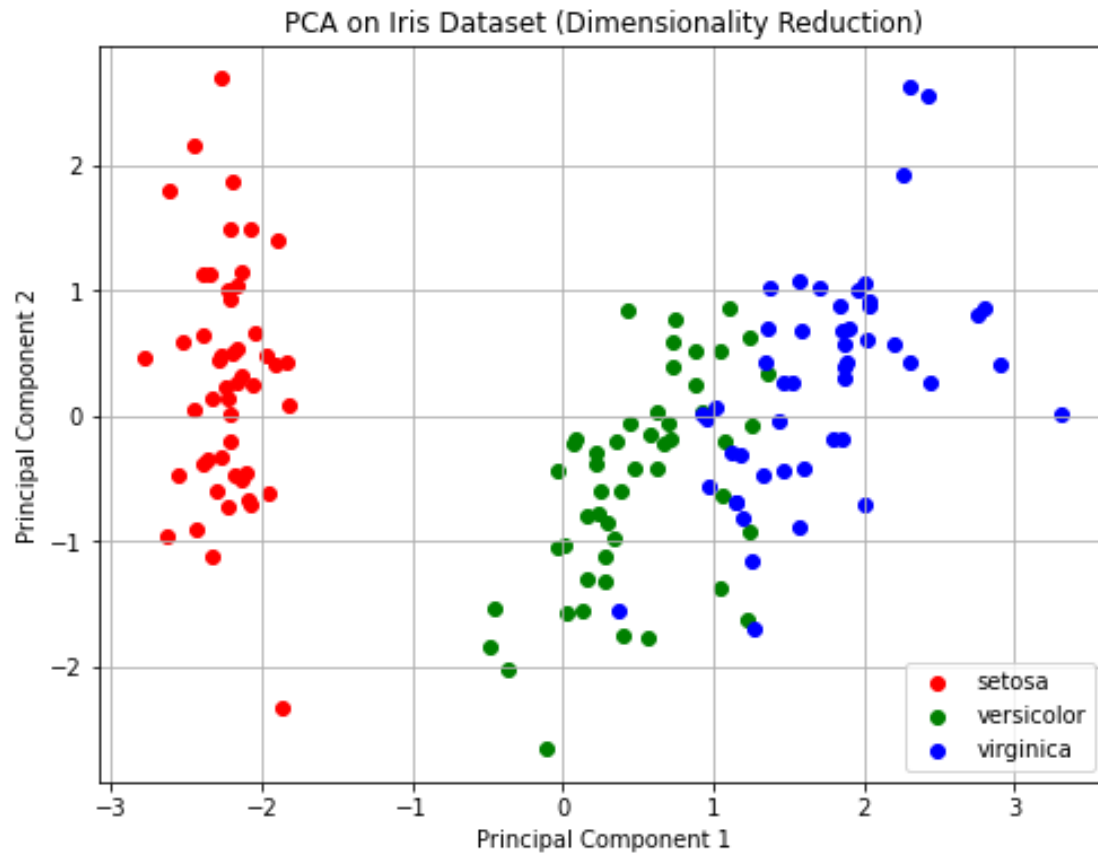
3D Visualization of Iris Data Before PCA



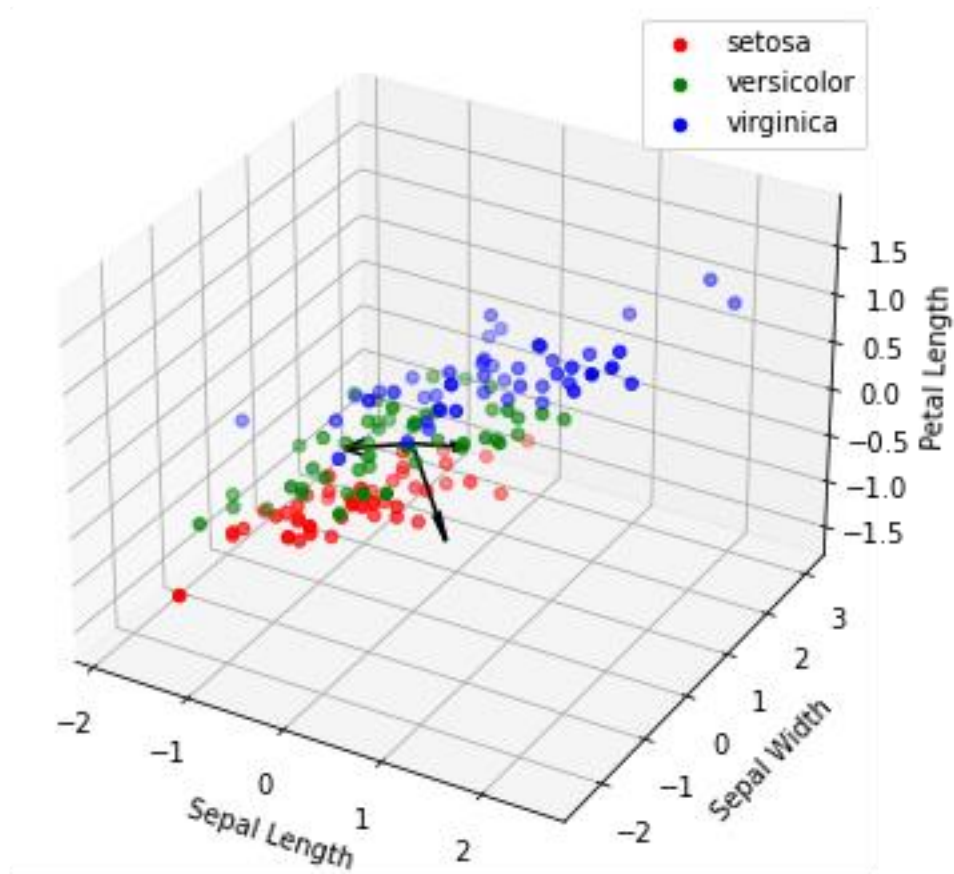
Singular Values: [20.92306556 11.7091661 4.69185798 1.76273239]

Explained Variance by PC1: 0.73

Explained Variance by PC2: 0.23



3D Data with Eigenvectors



## 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.

## 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.

### Understanding Find-S Algorithm and Hypothesis Concept

The Find-S algorithm is a simple machine-learning algorithm used in concept learning. It finds the most specific hypothesis that is consistent with all positive examples in a given training dataset. The algorithm assumes:

The target concept is represented in a binary classification (yes/no, true/false, etc.).

The hypothesis space uses conjunctive attributes (each attribute in a hypothesis must match exactly).

There is at least one positive example in the dataset.

```
import pandas as pd
```

```
data = pd.read_csv(r"C:\Users\vijay\Desktop\Machine Learning Course
Batches\FDP_ML_6thSem_VTU\Experiment_4_FindS\training_data.csv")
```

```
print(data)
```

	Experience	Qualification	Skill	Age	Hired
0	Yes	Masters	Python	30	Yes
1	Yes	Bachelors	Python	25	Yes
2	No	Bachelors	Java	28	No
3	Yes	Masters	Java	40	Yes
4	No	Masters	Python	35	No

```
def find_s_algorithm(data):
```

```
    """Implements the Find-S algorithm to find the most specific
    hypothesis."""
```

```
    # Extract feature columns and target column
```

```
    attributes = data.iloc[:, :-1].values # All columns except last
```

```
    target = data.iloc[:, -1].values # Last column (class labels)
```

```
    # Step 1: Initialize hypothesis with first positive example
```

```
    for i in range(len(target)):
```

```
        if target[i] == "Yes": # Consider only positive examples
```

```
            hypothesis = attributes[i].copy()
```

```
            break
```

```
# Step 2: Update hypothesis based on other positive examples
for i in range(len(target)):
    if target[i] == "Yes":
        for j in range(len(hypothesis)):
            if hypothesis[j] != attributes[i][j]:
                hypothesis[j] = '?' # Generalize inconsistent attributes

return hypothesis

# Run Find-S Algorithm
final_hypothesis = find_s_algorithm(data)

# Print the Learned hypothesis
print("Most Specific Hypothesis:", final_hypothesis)

Most Specific Hypothesis: ['Yes' '?' '?' '?']
```

## 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.

3. Label the first 50 points {x<sub>1</sub>,.....,x<sub>50</sub>} as follows: if (x<sub>i</sub> ≤ 0.5), then x<sub>i</sub> ∈ Class1, else x<sub>i</sub> ∈ Class1
4. Classify the remaining points, x<sub>51</sub>,.....,x<sub>100</sub> using KNN. Perform this for k=1,2,3,4,5,20,30

```
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.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')

# Step 1: Generate dataset
np.random.seed(42)
values = np.random.rand(100)

labels = []

for i in values[:50]:
    if i <=0.5:
        labels.append('Class1')
    else:
        labels.append('Class2')

labels += [None] * 50

print(labels)

['Class1', 'Class2', 'Class2', 'Class2', 'Class1', 'Class1', 'Class1',
'Class2', 'Class2', 'Class2', 'Class1', 'Class2', 'Class2', 'Class1',
'Class1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2',
'Class1', 'Class1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class2',
'Class2', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2', 'Class2',
'Class2', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', 'Class1',
'Class1', 'Class2', 'Class1', 'Class2', 'Class1', 'Class2', 'Class2',
'Class1', None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None]

data = {
    "Point": [f"x{i+1}" for i in range(100)],
```

```

    "Value": values,
    "Label": labels
}

print(data)
type(data)

{'Point': ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10',
'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21',
'x22', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28', 'x29', 'x30', 'x31', 'x32',
'x33', 'x34', 'x35', 'x36', 'x37', 'x38', 'x39', 'x40', 'x41', 'x42', 'x43',
'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'x50', 'x51', 'x52', 'x53', 'x54',
'x55', 'x56', 'x57', 'x58', 'x59', 'x60', 'x61', 'x62', 'x63', 'x64', 'x65',
'x66', 'x67', 'x68', 'x69', 'x70', 'x71', 'x72', 'x73', 'x74', 'x75', 'x76',
'x77', 'x78', 'x79', 'x80', 'x81', 'x82', 'x83', 'x84', 'x85', 'x86', 'x87',
'x88', 'x89', 'x90', 'x91', 'x92', 'x93', 'x94', 'x95', 'x96', 'x97', 'x98',
'x99', 'x100'], 'Value': array([0.37454012, 0.95071431, 0.73199394,
0.59865848, 0.15601864,
    0.15599452, 0.05808361, 0.86617615, 0.60111501, 0.70807258,
    0.02058449, 0.96990985, 0.83244264, 0.21233911, 0.18182497,
    0.18340451, 0.30424224, 0.52475643, 0.43194502, 0.29122914,
    0.61185289, 0.13949386, 0.29214465, 0.36636184, 0.45606998,
    0.78517596, 0.19967378, 0.51423444, 0.59241457, 0.04645041,
    0.60754485, 0.17052412, 0.06505159, 0.94888554, 0.96563203,
    0.80839735, 0.30461377, 0.09767211, 0.68423303, 0.44015249,
    0.12203823, 0.49517691, 0.03438852, 0.9093204 , 0.25877998,
    0.66252228, 0.31171108, 0.52006802, 0.54671028, 0.18485446,
    0.96958463, 0.77513282, 0.93949894, 0.89482735, 0.59789998,
    0.92187424, 0.0884925 , 0.19598286, 0.04522729, 0.32533033,
    0.38867729, 0.27134903, 0.82873751, 0.35675333, 0.28093451,
    0.54269608, 0.14092422, 0.80219698, 0.07455064, 0.98688694,
    0.77224477, 0.19871568, 0.00552212, 0.81546143, 0.70685734,
    0.72900717, 0.77127035, 0.07404465, 0.35846573, 0.11586906,
    0.86310343, 0.62329813, 0.33089802, 0.06355835, 0.31098232,
    0.32518332, 0.72960618, 0.63755747, 0.88721274, 0.47221493,
    0.11959425, 0.71324479, 0.76078505, 0.5612772 , 0.77096718,
    0.4937956 , 0.52273283, 0.42754102, 0.02541913, 0.10789143]), 'Label':
['Class1', 'Class2', 'Class2', 'Class2', 'Class1', 'Class1', 'Class1',
'Class2', 'Class2', 'Class2', 'Class1', 'Class2', 'Class2', 'Class1',
'Class1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2',
'Class1', 'Class1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class2',
'Class2', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2', 'Class2',
'Class2', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', 'Class1',
'Class1', 'Class2', 'Class1', 'Class2', 'Class1', 'Class2', 'Class2',
'Class1', None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None, None, None,
None]]

dict

```



```

df = pd.DataFrame(data)
df.head()

   Point  Value  Label
0    x1  0.374540  Class1
1    x2  0.950714  Class2
2    x3  0.731994  Class2
3    x4  0.598658  Class2
4    x5  0.156019  Class1

df.nunique()

Point      100
Value      100
Label        2
dtype: int64

df.shape

(100, 3)

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   Point   100 non-null      object
 1   Value   100 non-null      float64
 2   Label   50 non-null       object
dtypes: float64(1), object(2)
memory usage: 2.5+ KB

df.describe().T

      count      mean      std      min      25%      50%      75%  \
Value  100.0  0.470181  0.297489  0.005522  0.193201  0.464142  0.730203

      max
Value  0.986887

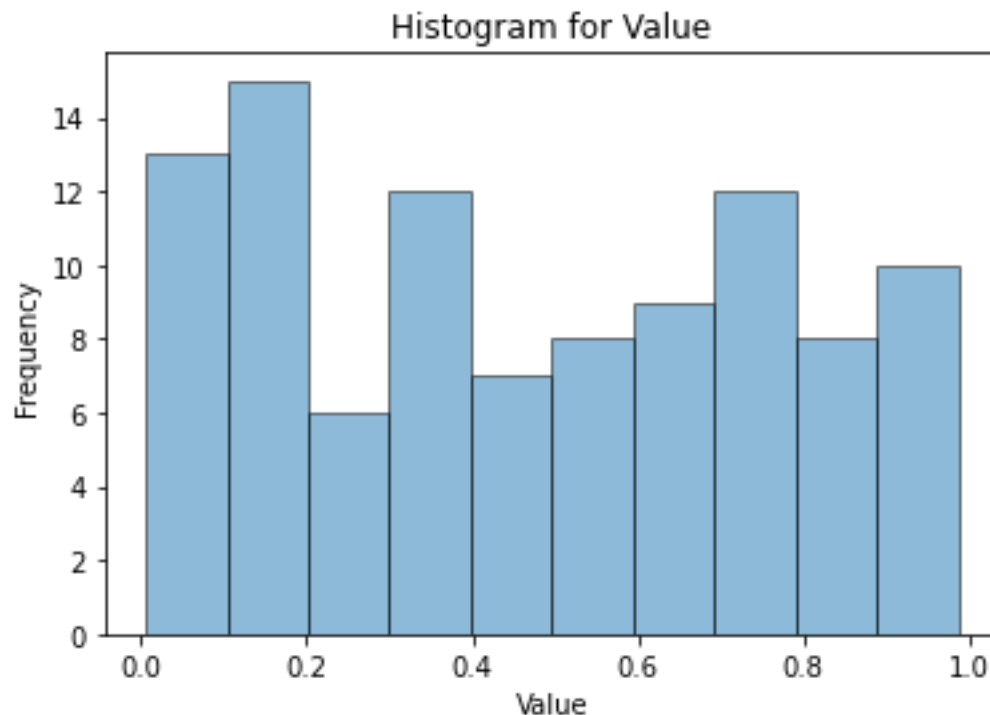
df.isnull().sum()

Point      0
Value      0
Label     50
dtype: int64

num_col = df.select_dtypes(include=['int', 'float']).columns
for col in num_col:
    df[col].hist(bins=10, alpha=0.5, edgecolor='black', grid=False)
    plt.title(f'Histogram for {col}')

```

```
plt.xlabel(col)
plt.ylabel('Frequency')
plt.show()
```



```
# Split data into labeled and unlabeled
labeled_df = df[df["Label"].notna()]
X_train = labeled_df[["Value"]]
y_train = labeled_df["Label"]

unlabeled_df = df[df["Label"].isna()]
X_test = unlabeled_df[["Value"]]

# Generate true labels for testing (for accuracy calculation)
true_labels = ["Class1" if x <= 0.5 else "Class2" for x in values[50:]]

# Step 2: Perform KNN classification for different values of k
k_values = [1, 2, 3, 4, 5, 20, 30]
results = {}
accuracies = {}

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    predictions = knn.predict(X_test)
    results[k] = predictions

# Calculate accuracy
accuracy = accuracy_score(true_labels, predictions) * 100
accuracies[k] = accuracy
```

```

print(f"Accuracy for k={k}: {accuracy:.2f}%")

# Assign predictions back to the DataFrame for this k
unlabeled_df[f"Label_k{k}"] = predictions

Accuracy for k=1: 100.00%
Accuracy for k=2: 100.00%
Accuracy for k=3: 98.00%
Accuracy for k=4: 98.00%
Accuracy for k=5: 98.00%
Accuracy for k=20: 98.00%
Accuracy for k=30: 100.00%

print(predictions)

['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class1' 'Class1']

df1 = unlabeled_df.drop(columns=['Label'], axis=1)
df1

```

	Point	Value	Label_k1	Label_k2	Label_k3	Label_k4	Label_k5	Label_k20	\
50	x51	0.969585	Class2	Class2	Class2	Class2	Class2	Class2	
51	x52	0.775133	Class2	Class2	Class2	Class2	Class2	Class2	
52	x53	0.939499	Class2	Class2	Class2	Class2	Class2	Class2	
53	x54	0.894827	Class2	Class2	Class2	Class2	Class2	Class2	
54	x55	0.597900	Class2	Class2	Class2	Class2	Class2	Class2	
55	x56	0.921874	Class2	Class2	Class2	Class2	Class2	Class2	
56	x57	0.088493	Class1	Class1	Class1	Class1	Class1	Class1	
57	x58	0.195983	Class1	Class1	Class1	Class1	Class1	Class1	
58	x59	0.045227	Class1	Class1	Class1	Class1	Class1	Class1	
59	x60	0.325330	Class1	Class1	Class1	Class1	Class1	Class1	
60	x61	0.388677	Class1	Class1	Class1	Class1	Class1	Class1	
61	x62	0.271349	Class1	Class1	Class1	Class1	Class1	Class1	
62	x63	0.828738	Class2	Class2	Class2	Class2	Class2	Class2	
63	x64	0.356753	Class1	Class1	Class1	Class1	Class1	Class1	
64	x65	0.280935	Class1	Class1	Class1	Class1	Class1	Class1	
65	x66	0.542696	Class2	Class2	Class2	Class2	Class2	Class2	
66	x67	0.140924	Class1	Class1	Class1	Class1	Class1	Class1	
67	x68	0.802197	Class2	Class2	Class2	Class2	Class2	Class2	
68	x69	0.074551	Class1	Class1	Class1	Class1	Class1	Class1	
69	x70	0.986887	Class2	Class2	Class2	Class2	Class2	Class2	
70	x71	0.772245	Class2	Class2	Class2	Class2	Class2	Class2	
71	x72	0.198716	Class1	Class1	Class1	Class1	Class1	Class1	
72	x73	0.005522	Class1	Class1	Class1	Class1	Class1	Class1	
73	x74	0.815461	Class2	Class2	Class2	Class2	Class2	Class2	

74	x75	0.706857	Class2	Class2	Class2	Class2	Class2	Class2
75	x76	0.729007	Class2	Class2	Class2	Class2	Class2	Class2
76	x77	0.771270	Class2	Class2	Class2	Class2	Class2	Class2
77	x78	0.074045	Class1	Class1	Class1	Class1	Class1	Class1
78	x79	0.358466	Class1	Class1	Class1	Class1	Class1	Class1
79	x80	0.115869	Class1	Class1	Class1	Class1	Class1	Class1
80	x81	0.863103	Class2	Class2	Class2	Class2	Class2	Class2
81	x82	0.623298	Class2	Class2	Class2	Class2	Class2	Class2
82	x83	0.330898	Class1	Class1	Class1	Class1	Class1	Class1
83	x84	0.063558	Class1	Class1	Class1	Class1	Class1	Class1
84	x85	0.310982	Class1	Class1	Class1	Class1	Class1	Class1
85	x86	0.325183	Class1	Class1	Class1	Class1	Class1	Class1
86	x87	0.729606	Class2	Class2	Class2	Class2	Class2	Class2
87	x88	0.637557	Class2	Class2	Class2	Class2	Class2	Class2
88	x89	0.887213	Class2	Class2	Class2	Class2	Class2	Class2
89	x90	0.472215	Class1	Class1	Class1	Class1	Class1	Class1
90	x91	0.119594	Class1	Class1	Class1	Class1	Class1	Class1
91	x92	0.713245	Class2	Class2	Class2	Class2	Class2	Class2
92	x93	0.760785	Class2	Class2	Class2	Class2	Class2	Class2
93	x94	0.561277	Class2	Class2	Class2	Class2	Class2	Class2
94	x95	0.770967	Class2	Class2	Class2	Class2	Class2	Class2
95	x96	0.493796	Class1	Class1	Class2	Class2	Class2	Class2
96	x97	0.522733	Class2	Class2	Class2	Class2	Class2	Class2
97	x98	0.427541	Class1	Class1	Class1	Class1	Class1	Class1
98	x99	0.025419	Class1	Class1	Class1	Class1	Class1	Class1
99	x100	0.107891	Class1	Class1	Class1	Class1	Class1	Class1

Label\_k30

50	Class2
51	Class2
52	Class2
53	Class2
54	Class2
55	Class2
56	Class1
57	Class1
58	Class1
59	Class1
60	Class1
61	Class1
62	Class2
63	Class1
64	Class1
65	Class2
66	Class1
67	Class2
68	Class1
69	Class2
70	Class2
71	Class1

```
72     Class1
73     Class2
74     Class2
75     Class2
76     Class2
77     Class1
78     Class1
79     Class1
80     Class2
81     Class2
82     Class1
83     Class1
84     Class1
85     Class1
86     Class2
87     Class2
88     Class2
89     Class1
90     Class1
91     Class2
92     Class2
93     Class2
94     Class2
95     Class1
96     Class2
97     Class1
98     Class1
99     Class1

# Display accuracies
print("\nAccuracies for different k values:")
for k, acc in accuracies.items():
    print(f"k={k}: {acc:.2f}%")
```

Accuracies for different k values:

```
k=1: 100.00%
k=2: 100.00%
k=3: 98.00%
k=4: 98.00%
k=5: 98.00%
k=20: 98.00%
k=30: 100.00%
```

## Program 6

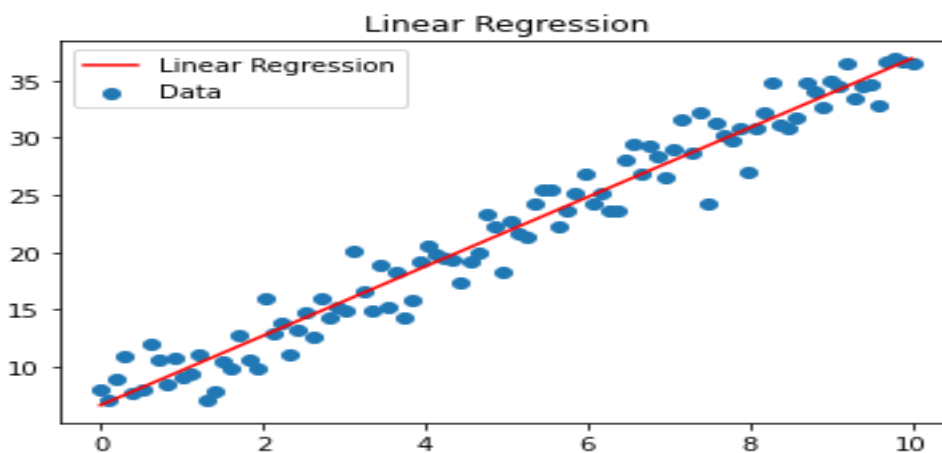
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from scipy.spatial.distance import cdist

# Load datasets
df_linear = pd.read_csv("linear_dataset.csv")
df_lwr = pd.read_csv("lwr_dataset.csv")
df_poly = pd.read_csv("polynomial_dataset.csv")

# Linear Regression
def linear_regression(df):
    X, y = df[['X']], df['Y']
    model = LinearRegression()
    model.fit(X, y)
    y_pred = model.predict(X)
    plt.scatter(X, y, label='Data')
    plt.plot(X, y_pred, color='red', label='Linear Regression')
    plt.legend()
    plt.title("Linear Regression")
    plt.show()

linear_regression(df_linear)
```



```

# Locally Weighted Regression (LWR)
def gaussian_kernel(x, X, tau):
    return np.exp(-cdist([[x]], X, 'sqeuclidean') / (2 * tau**2))

def locally_weighted_regression(X_train, y_train, tau=0.5):
    X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train]) # Add
    intercept
    X_range = np.linspace(X_train[:, 1].min(), X_train[:, 1].max(), 100)
    y_pred = []

    for x in X_range:
        x_vec = np.array([1, x]) # Intercept term
        weights = gaussian_kernel(x, X_train[:, 1:], tau).flatten()
        W = np.diag(weights)

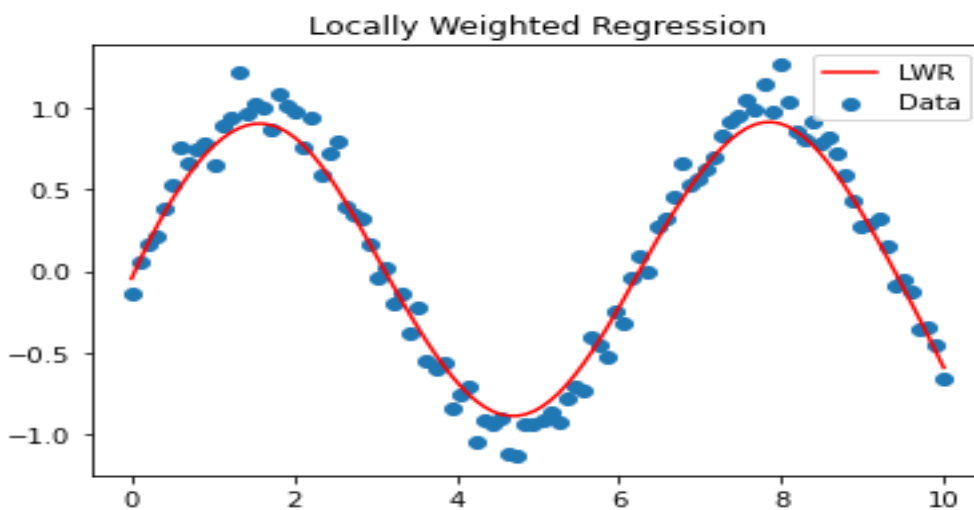
        theta = np.linalg.pinv(X_train.T @ W @ X_train) @ (X_train.T @ W @
y_train)
        y_pred.append(x_vec @ theta) # Use dot product for prediction

    plt.scatter(X_train[:, 1], y_train, label='Data')
    plt.plot(X_range, y_pred, color='red', label='LWR')
    plt.legend()
    plt.title("Locally Weighted Regression")
    plt.show()

# Run the models

locally_weighted_regression(df_lwr[['X']].values, df_lwr['Y'].values)

```



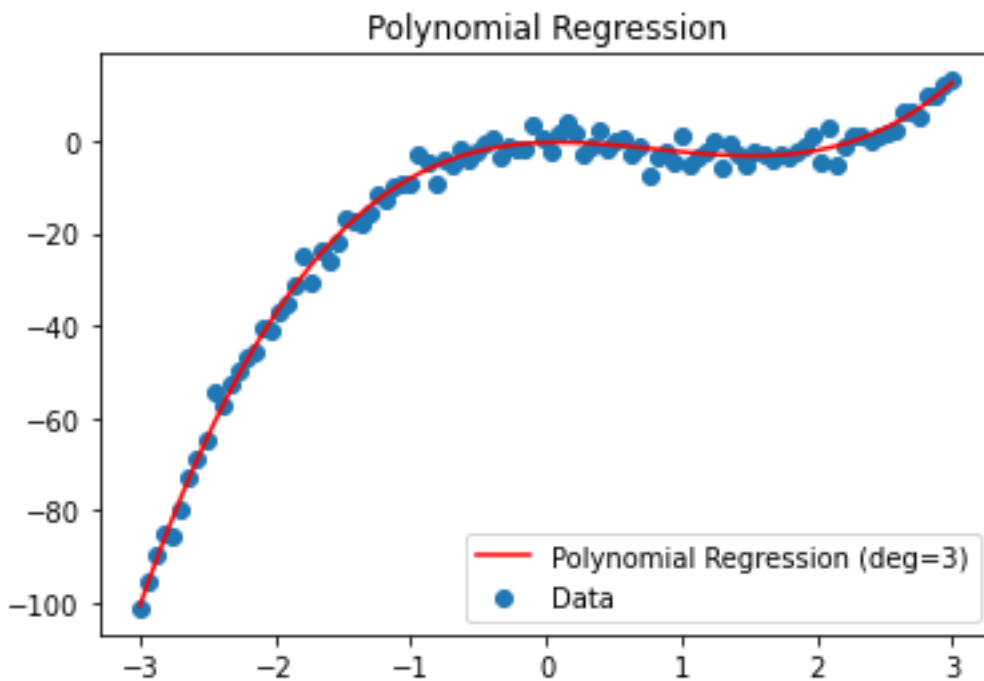
```

# Polynomial Regression
def polynomial_regression(df, degree=3):
    X, y = df[['X']], df['Y']

```

```
model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
model.fit(X, y)
y_pred = model.predict(X)
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='red', label=f'Polynomial Regression
(deg={degree})')
plt.legend()
plt.title("Polynomial Regression")
plt.show()

polynomial_regression(df_poly, degree=3)
```





## 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 pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv(r"C:\Users\Admin\OneDrive\Documents\Machine Learning
Lab\Datasets\Boston housing dataset.csv")
```

5. CRIM: Per capita crime rate by town.
6. ZN: Proportion of residential land zoned for lots over 25,000 square feet.
7. INDUS: Proportion of non-retail business acres per town.
8. CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise).
9. NOX: Nitric oxide concentration (parts per 10 million).
10. RM: Average number of rooms per dwelling.
11. AGE: Proportion of owner-occupied units built before 1940.
12. DIS: Weighted distances to five Boston employment centers.
13. RAD: Index of accessibility to radial highways.
14. TAX: Full-value property-tax rate per \$10,000.
15. PTRATIO: Pupil-teacher ratio by town.
16. B:  $1000(Bk - 0.63)^2$ , where  $Bk$  is the proportion of Black residents by town.
17. LSTAT: Percentage of the lower status of the population.
18. MEDV: Median value of owner-occupied homes in \$1000s.

```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7

	B	LSTAT	MEDV
0	396.90	4.98	24.0
1	396.90	9.14	21.6

```

2  392.83    4.03    34.7
3  394.63    2.94    33.4
4  396.90     NaN    36.2

```

```
data.shape
```

```
(506, 14)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 506 entries, 0 to 505
```

```
Data columns (total 14 columns):
```

#	Column	Non-Null Count	Dtype
0	CRIM	486 non-null	float64
1	ZN	486 non-null	float64
2	INDUS	486 non-null	float64
3	CHAS	486 non-null	float64
4	NOX	506 non-null	float64
5	RM	506 non-null	float64
6	AGE	486 non-null	float64
7	DIS	506 non-null	float64
8	RAD	506 non-null	int64
9	TAX	506 non-null	int64
10	PTRATIO	506 non-null	float64
11	B	506 non-null	float64
12	LSTAT	486 non-null	float64
13	MEDV	506 non-null	float64

```
dtypes: float64(12), int64(2)
```

```
memory usage: 55.5 KB
```

19. The dataset contains 506 entries and 14 columns, with 6 columns (CRIM, ZN, INDUS, CHAS, AGE, LSTAT) having 20 missing values each.
20. Most columns are continuous (float64), while RAD and TAX are discrete (int64).
21. MEDV (median home value) is the target variable, likely influenced by features like RM (average rooms) and LSTAT (lower-status population).
22. Missing values need to be addressed through imputation or by dropping rows with missing data.
23. Exploratory analysis and modeling can help understand feature relationships and predict MEDV.

```
data.nunique()
```

CRIM	484
ZN	26
INDUS	76
CHAS	2
NOX	81
RM	446
AGE	348

```
DIS      412
RAD       9
TAX      66
PTRATIO  46
B       357
LSTAT   438
MEDV    229
dtype: int64
```

```
data.CHAS.unique()
```

```
array([ 0., nan,  1.])
```

```
data.ZN.unique()
```

```
array([ 18. ,  0. , 12.5, 75. , 21. , 90. , 85. , 100. , 25. ,
        17.5, 80. , nan, 28. , 45. , 60. , 95. , 82.5, 30. ,
        22. , 20. , 40. , 55. , 52.5, 70. , 34. , 33. , 35. ])
```

## Data Cleaning

### Checking Null values

`data.isnull()` - Returns a DataFrame of the same shape as data, where each element is True if it's NaN and False otherwise.

`.sum()` - Sums up the True values (which are treated as 1 in Python) column-wise, giving the total count of missing values for each column.

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

```
CRIM      20
ZN        20
INDUS     20
CHAS      20
NOX       0
RM        0
AGE       20
DIS       0
RAD       0
TAX       0
PTRATIO   0
B         0
LSTAT     20
MEDV      0
dtype: int64
```

```
data.duplicated().sum()
```

```
np.int64(0)
```

```
df = data.copy()
```

```

df['CRIM'].fillna(df['CRIM'].mean(), inplace=True)
df['ZN'].fillna(df['ZN'].mean(), inplace=True)
df['CHAS'].fillna(df['CHAS'].mode()[0], inplace=True)
df['INDUS'].fillna(df['INDUS'].mean(), inplace=True)
df['AGE'].fillna(df['AGE'].median(), inplace=True) # Median is often
preferred for skewed distributions
df['LSTAT'].fillna(df['LSTAT'].median(), inplace=True)

```

```
df.isnull().sum()
```

```

CRIM      0
ZN        0
INDUS     0
CHAS      0
NOX       0
RM        0
AGE       0
DIS       0
RAD       0
TAX       0
PTRATIO   0
B         0
LSTAT     0
MEDV     0
dtype: int64

```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7

	B	LSTAT	MEDV
0	396.90	4.98	24.0
1	396.90	9.14	21.6
2	392.83	4.03	34.7
3	394.63	2.94	33.4
4	396.90	11.43	36.2

```
df['CHAS'] = df['CHAS'].astype('int')
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	\
CRIM	506.0	3.611874	8.545770	0.00632	0.083235	0.29025	
ZN	506.0	11.211934	22.921051	0.00000	0.000000	0.00000	
INDUS	506.0	11.083992	6.699165	0.46000	5.190000	9.90000	
CHAS	506.0	0.067194	0.250605	0.00000	0.000000	0.00000	

NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850
AGE	506.0	68.845850	27.486962	2.90000	45.925000	76.80000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000
LSTAT	506.0	12.664625	7.017219	1.73000	7.230000	11.43000
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000

	75%	max
CRIM	3.611874	88.9762
ZN	11.211934	100.0000
INDUS	18.100000	27.7400
CHAS	0.000000	1.0000
NOX	0.624000	0.8710
RM	6.623500	8.7800
AGE	93.575000	100.0000
DIS	5.188425	12.1265
RAD	24.000000	24.0000
TAX	666.000000	711.0000
PTRATIO	20.200000	22.0000
B	396.225000	396.9000
LSTAT	16.570000	37.9700
MEDV	25.000000	50.0000

```

for i in df.columns:
    plt.figure(figsize=(6,3))

    plt.subplot(1, 2, 1)
    df[i].hist(bins=20, alpha=0.5, color='b',edgecolor='black')
    plt.title(f'Histogram of {i}')
    plt.xlabel(i)
    plt.ylabel('Frequency')

    plt.subplot(1, 2, 2)
    plt.boxplot(df[i], vert=False)
    plt.title(f'Boxplot of {i}')

plt.show()

```

```

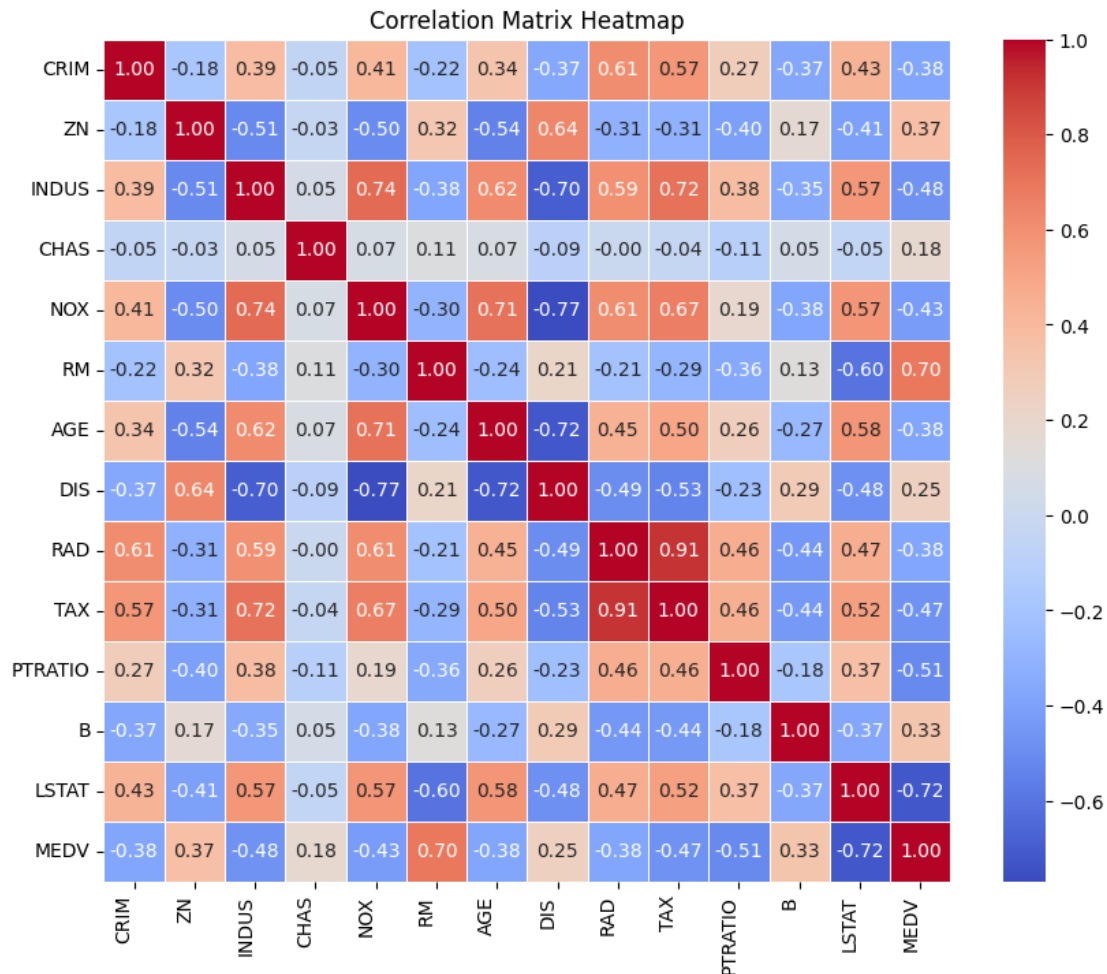
-----
NameError                                Traceback (most recent call last)
<ipython-input-1-72aa2831224b> in <module>
----> 1 for i in df.columns:
      2     plt.figure(figsize=(6,3))
      3
      4     plt.subplot(1, 2, 1)
      5     df[i].hist(bins=20, alpha=0.5, color='b',edgecolor='black')

```

NameError: name 'df' is not defined

```
corr = df.corr(method='pearson')
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.xticks(rotation=90, ha='right')
plt.yticks(rotation=0)
plt.title("Correlation Matrix Heatmap")
plt.show()
```



```
X = df.drop('MEDV', axis=1) # ALL columns except 'MEDV'
y = df['MEDV'] # Target variable
```

### Why Use StandardScaler?

24. Improved model performance: Linear models assume that features are normally distributed around the mean. Scaling the data can make the algorithm converge faster and produce more accurate predictions.
25. Prevents bias due to feature magnitude: Features with larger numeric ranges (like TAX or CRIM) may dominate the model if not scaled properly, especially in

regularized models. While standard linear regression may not be heavily affected, scaling ensures more consistent results.

```
# Scale the features
scale = StandardScaler()
X_scaled = scale.fit_transform(X)

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

# Initialize the linear regression model
model = LinearRegression()

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

LinearRegression()

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

array([28.99719439, 36.56606809, 14.51022803, 25.02572187, 18.42885474,
      23.02785726, 17.95437605, 14.5769479 , 22.14430832, 20.84584632,
      25.15283588, 18.55925182, -5.69168071, 21.71242445, 19.06845707,
      25.94275348, 19.70991322,  5.85916505, 40.9608103 , 17.21528576,
      25.36124981, 30.26007975, 11.78589412, 23.48106943, 17.35338161,
      15.13896898, 21.61919056, 14.51459386, 23.17246824, 19.40914754,
      22.56164985, 25.21208496, 25.88782605, 16.68297496, 16.44747174,
      16.65894826, 31.10314158, 20.25199803, 24.38567686, 23.09800032,
      14.47721796, 32.36053979, 43.01157914, 17.61473728, 27.60723089,
      16.43366912, 14.25719607, 26.0854729 , 19.75853278, 30.15142187,
      21.01932313, 33.72128781, 16.39180467, 26.36438908, 39.75793372,
      22.02419633, 18.39453126, 32.81854401, 25.370573 , 12.82224665,
      22.76128341, 30.73955199, 31.34386371, 16.27681305, 20.36945226,
      17.23156773, 20.15406451, 26.15613066, 30.92791361, 11.42177654,
      20.89590447, 26.58633798, 11.01176073, 12.76831709, 23.73870867,
       6.37180464, 21.6922679 , 41.74800223, 18.64423785,  8.82325704,
      20.96406016, 13.20179007, 20.99146149,  9.17404063, 23.0011185 ,
      32.41062673, 18.99778065, 25.56204885, 28.67383635, 19.76918944,
      25.94842754,  5.77674362, 19.514431 , 15.22571165, 10.87671123,
      20.08359505, 23.77725749,  0.05985008, 13.56333825, 16.1215622 ,
      22.74200442, 24.36218289])

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Calculate R-squared value
```

```
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
print(f'R-squared: {r2}')
```

Mean Squared Error: 24.944071172175562  
Root Mean Squared Error: 4.994403985679929  
R-squared: 0.6598556613717499



## 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 pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

```
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
data = pd.read_csv(r'C:\Users\Admin\OneDrive\Documents\Machine Learning
Lab\Datasets\Breast Cancer Dataset.csv')
```

```
pd.set_option('display.max_columns', None)
```

```
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
0	842302	M	17.99	10.38	122.80	1001.0
1	842517	M	20.57	17.77	132.90	1326.0
2	84300903	M	19.69	21.25	130.00	1203.0
3	84348301	M	11.42	20.38	77.58	386.1
4	84358402	M	20.29	14.34	135.10	1297.0

	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean
0	0.11840	0.27760	0.3001	0.14710
1	0.08474	0.07864	0.0869	0.07017
2	0.10960	0.15990	0.1974	0.12790
3	0.14250	0.28390	0.2414	0.10520
4	0.10030	0.13280	0.1980	0.10430

	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se
0	0.2419	0.07871	1.0950	0.9053	8.589
1	0.1812	0.05667	0.5435	0.7339	3.398
2	0.2069	0.05999	0.7456	0.7869	4.585

## Machine Learning Lab (BCSL606)

3	0.2597	0.09744	0.4956	1.1560	3.445
4	0.1809	0.05883	0.7572	0.7813	5.438

	area_se	smoothness_se	compactness_se	concavity_se	concave_points_se	\
0	153.40	0.006399	0.04904	0.05373	0.01587	
1	74.08	0.005225	0.01308	0.01860	0.01340	
2	94.03	0.006150	0.04006	0.03832	0.02058	
3	27.23	0.009110	0.07458	0.05661	0.01867	
4	94.44	0.011490	0.02461	0.05688	0.01885	

	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	\
0	0.03003	0.006193	25.38	17.33	
1	0.01389	0.003532	24.99	23.41	
2	0.02250	0.004571	23.57	25.53	
3	0.05963	0.009208	14.91	26.50	
4	0.01756	0.005115	22.54	16.67	

	perimeter_worst	area_worst	smoothness_worst	compactness_worst	\
0	184.60	2019.0	0.1622	0.6656	
1	158.80	1956.0	0.1238	0.1866	
2	152.50	1709.0	0.1444	0.4245	
3	98.87	567.7	0.2098	0.8663	
4	152.20	1575.0	0.1374	0.2050	

	concavity_worst	concave_points_worst	symmetry_worst	\
0	0.7119	0.2654	0.4601	
1	0.2416	0.1860	0.2750	
2	0.4504	0.2430	0.3613	
3	0.6869	0.2575	0.6638	
4	0.4000	0.1625	0.2364	

	fractal_dimension_worst
0	0.11890
1	0.08902
2	0.08758
3	0.17300
4	0.07678

data.shape

(569, 32)

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 569 entries, 0 to 568

Data columns (total 32 columns):

#	Column	Non-Null Count	Dtype
---	-----	-----	-----
0	id	569 non-null	int64
1	diagnosis	569 non-null	object

```

2   radius_mean          569 non-null    float64
3   texture_mean         569 non-null    float64
4   perimeter_mean       569 non-null    float64
5   area_mean            569 non-null    float64
6   smoothness_mean      569 non-null    float64
7   compactness_mean     569 non-null    float64
8   concavity_mean       569 non-null    float64
9   concave_points_mean  569 non-null    float64
10  symmetry_mean        569 non-null    float64
11  fractal_dimension_mean 569 non-null    float64
12  radius_se            569 non-null    float64
13  texture_se           569 non-null    float64
14  perimeter_se         569 non-null    float64
15  area_se              569 non-null    float64
16  smoothness_se        569 non-null    float64
17  compactness_se       569 non-null    float64
18  concavity_se         569 non-null    float64
19  concave_points_se    569 non-null    float64
20  symmetry_se          569 non-null    float64
21  fractal_dimension_se  569 non-null    float64
22  radius_worst         569 non-null    float64
23  texture_worst        569 non-null    float64
24  perimeter_worst      569 non-null    float64
25  area_worst           569 non-null    float64
26  smoothness_worst     569 non-null    float64
27  compactness_worst    569 non-null    float64
28  concavity_worst      569 non-null    float64
29  concave_points_worst 569 non-null    float64
30  symmetry_worst       569 non-null    float64
31  fractal_dimension_worst 569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB

```

```
data.diagnosis.unique()
```

```
array(['M', 'B'], dtype=object)
```

## Data Preprocessing

### *Data Cleaning*

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

```

id                0
diagnosis         0
radius_mean       0
texture_mean      0
perimeter_mean    0
area_mean         0
smoothness_mean   0
compactness_mean  0
concavity_mean    0

```

```

concave_points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave_points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst        0
compactness_worst        0
concavity_worst          0
concave_points_worst     0
symmetry_worst           0
fractal_dimension_worst  0
dtype: int64

```

```
data.duplicated().sum()
```

```
np.int64(0)
```

```
df = data.drop(['id'], axis=1)
```

```
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0}) # Malignant:1, Benign:0
```

## Discriptive Statistics

```
df.describe().T
```

	count	mean	std	min \
diagnosis	569.0	0.372583	0.483918	0.000000
radius_mean	569.0	14.127292	3.524049	6.981000
texture_mean	569.0	19.289649	4.301036	9.710000
perimeter_mean	569.0	91.969033	24.298981	43.790000
area_mean	569.0	654.889104	351.914129	143.500000
smoothness_mean	569.0	0.096360	0.014064	0.052630
compactness_mean	569.0	0.104341	0.052813	0.019380
concavity_mean	569.0	0.088799	0.079720	0.000000
concave_points_mean	569.0	0.048919	0.038803	0.000000
symmetry_mean	569.0	0.181162	0.027414	0.106000
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960
radius_se	569.0	0.405172	0.277313	0.111500
texture_se	569.0	1.216853	0.551648	0.360200
perimeter_se	569.0	2.866059	2.021855	0.757000

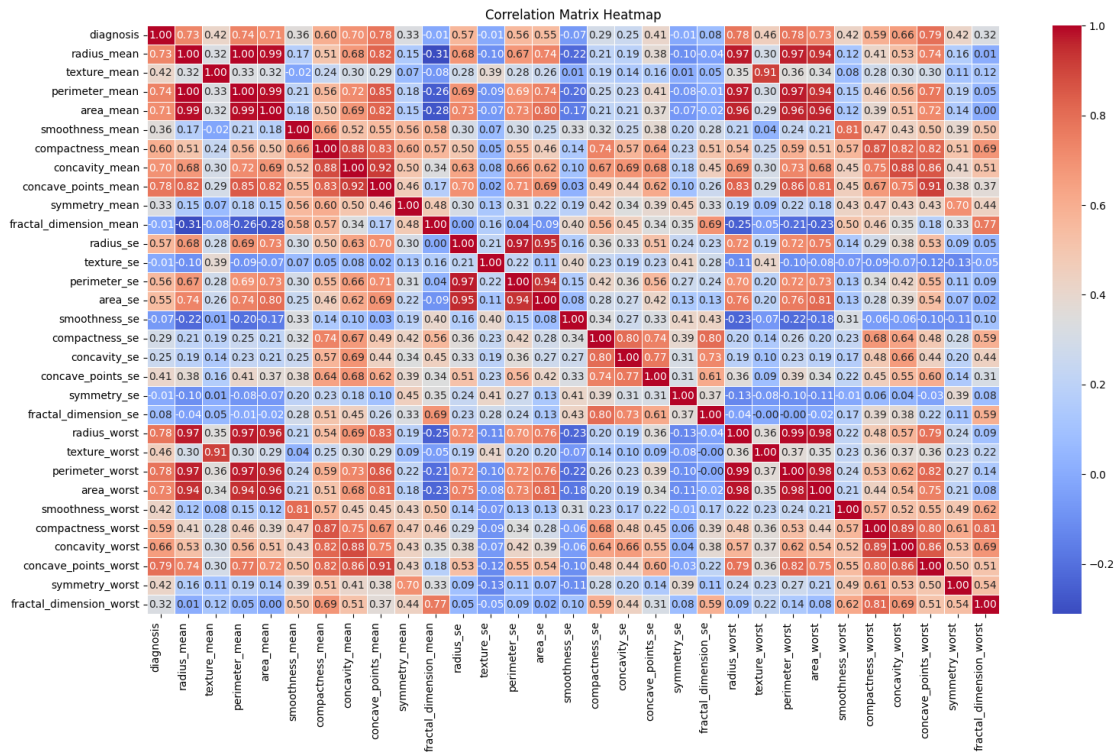
## Machine Learning Lab (BCSL606)

area_se	569.0	40.337079	45.491006	6.802000
smoothness_se	569.0	0.007041	0.003003	0.001713
compactness_se	569.0	0.025478	0.017908	0.002252
concavity_se	569.0	0.031894	0.030186	0.000000
concave_points_se	569.0	0.011796	0.006170	0.000000
symmetry_se	569.0	0.020542	0.008266	0.007882
fractal_dimension_se	569.0	0.003795	0.002646	0.000895
radius_worst	569.0	16.269190	4.833242	7.930000
texture_worst	569.0	25.677223	6.146258	12.020000
perimeter_worst	569.0	107.261213	33.602542	50.410000
area_worst	569.0	880.583128	569.356993	185.200000
smoothness_worst	569.0	0.132369	0.022832	0.071170
compactness_worst	569.0	0.254265	0.157336	0.027290
concavity_worst	569.0	0.272188	0.208624	0.000000
concave_points_worst	569.0	0.114606	0.065732	0.000000
symmetry_worst	569.0	0.290076	0.061867	0.156500
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040

	25%	50%	75%	max
diagnosis	0.000000	0.000000	1.000000	1.000000
radius_mean	11.700000	13.370000	15.780000	28.11000
texture_mean	16.170000	18.840000	21.800000	39.28000
perimeter_mean	75.170000	86.240000	104.100000	188.50000
area_mean	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	0.086370	0.095870	0.105300	0.16340
compactness_mean	0.064920	0.092630	0.130400	0.34540
concavity_mean	0.029560	0.061540	0.130700	0.42680
concave_points_mean	0.020310	0.033500	0.074000	0.20120
symmetry_mean	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	0.057700	0.061540	0.066120	0.09744
radius_se	0.232400	0.324200	0.478900	2.87300
texture_se	0.833900	1.108000	1.474000	4.88500
perimeter_se	1.606000	2.287000	3.357000	21.98000
area_se	17.850000	24.530000	45.190000	542.20000
smoothness_se	0.005169	0.006380	0.008146	0.03113
compactness_se	0.013080	0.020450	0.032450	0.13540
concavity_se	0.015090	0.025890	0.042050	0.39600
concave_points_se	0.007638	0.010930	0.014710	0.05279
symmetry_se	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	0.002248	0.003187	0.004558	0.02984
radius_worst	13.010000	14.970000	18.790000	36.04000
texture_worst	21.080000	25.410000	29.720000	49.54000
perimeter_worst	84.110000	97.660000	125.400000	251.20000
area_worst	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	0.116600	0.131300	0.146000	0.22260
compactness_worst	0.147200	0.211900	0.339100	1.05800
concavity_worst	0.114500	0.226700	0.382900	1.25200
concave_points_worst	0.064930	0.099930	0.161400	0.29100
symmetry_worst	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	0.071460	0.080040	0.092080	0.20750

```
corr = df.corr(method='pearson')
```

```
plt.figure(figsize=(18, 10))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.xticks(rotation=90, ha='right')
plt.yticks(rotation=0)
plt.title("Correlation Matrix Heatmap")
plt.show()
```



```
X = df.drop('diagnosis', axis=1) # Drop the 'diagnosis' column (target)
y = df['diagnosis']
```

```
# Split the dataset 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)
```

```
# Fit the decision tree model
```

```
model = DecisionTreeClassifier(criterion='entropy') #criteria = gini, entropy
```

```
model.fit(X_train, y_train)
```

```
model
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
import math
```

```
# Function to calculate entropy
```

```
def entropy(column):
    counts = column.value_counts()
    probabilities = counts / len(column)
```

```

    return -sum(probabilities * probabilities.apply(math.log2))

# Function to calculate conditional entropy
def conditional_entropy(data, X, target):
    feature_values = data[X].unique() # Corrected: use .unique() on the
    series
    weighted_entropy = 0
    for value in feature_values:
        subset = data[data[X] == value]
        weighted_entropy += (len(subset) / len(data)) *
        entropy(subset[target])
    return weighted_entropy

# Function to calculate information gain
def information_gain(data, X, target):
    total_entropy = entropy(data[target])
    feature_conditional_entropy = conditional_entropy(data, X, target)
    return total_entropy - feature_conditional_entropy

# Calculate information gain for each feature

for feature in X:
    ig = information_gain(df, feature, 'diagnosis')
    print(f"Information Gain for {feature}: {ig}")

Information Gain for radius_mean: 0.8607815854835991
Information Gain for texture_mean: 0.8357118798482908
Information Gain for perimeter_mean: 0.9267038614138748
Information Gain for area_mean: 0.9280305529818247
Information Gain for smoothness_mean: 0.7761788341876101
Information Gain for compactness_mean: 0.9091291689709926
Information Gain for concavity_mean: 0.9350604299589776
Information Gain for concave_points_mean: 0.9420903069361305
Information Gain for symmetry_mean: 0.735036638169654
Information Gain for fractal_dimension_mean: 0.8361770160635639
Information Gain for radius_se: 0.9337337383910278
Information Gain for texture_se: 0.8642965239721755
Information Gain for perimeter_se: 0.9315454914704012
Information Gain for area_se: 0.925377169845925
Information Gain for smoothness_se: 0.9350604299589776
Information Gain for compactness_se: 0.9231889229252984
Information Gain for concavity_se: 0.9280305529818247
Information Gain for concave_points_se: 0.8585933385629725
Information Gain for symmetry_se: 0.8181371874054084
Information Gain for fractal_dimension_se: 0.9174857375160954
Information Gain for radius_worst: 0.9003074642106167
Information Gain for texture_worst: 0.8634349686194988
Information Gain for perimeter_worst: 0.8985843535052632
Information Gain for area_worst: 0.9350604299589776

```

Information Gain for smoothness\_worst: 0.7197189097252679  
 Information Gain for compactness\_worst: 0.9183472928687721  
 Information Gain for concavity\_worst: 0.9302187999024514  
 Information Gain for concave\_points\_worst: 0.9148323543801957  
 Information Gain for symmetry\_worst: 0.8453951399613433  
 Information Gain for fractal\_dimension\_worst: 0.8915544765281104

# Export the tree to DOT format

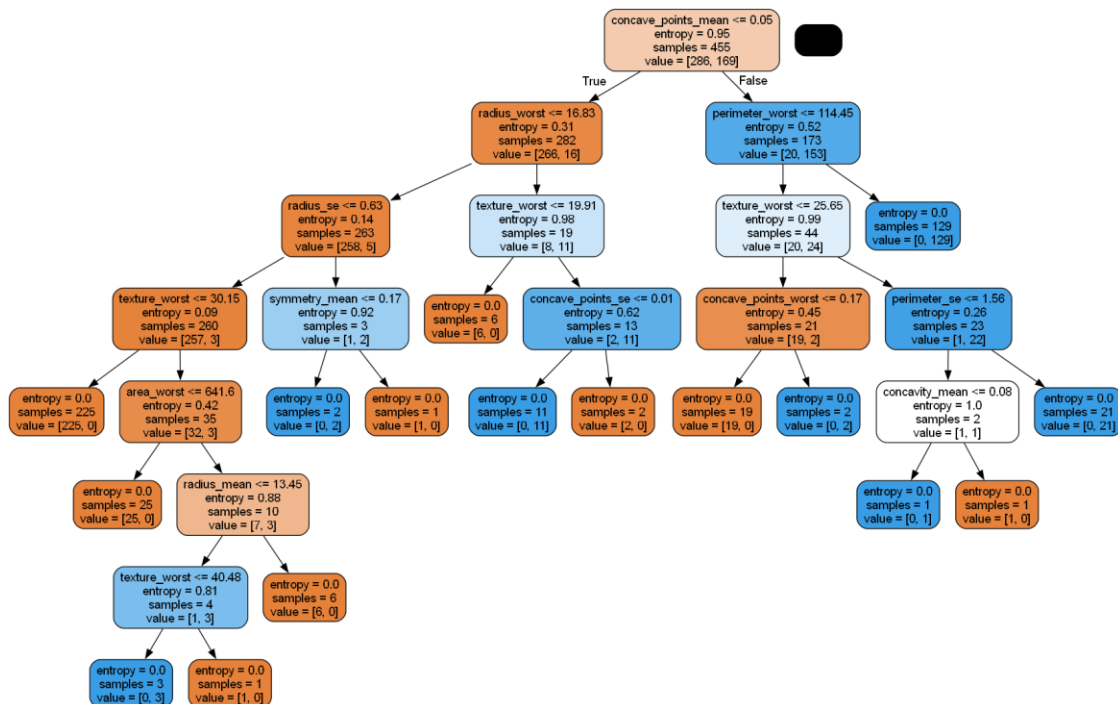
```
dot_data = export_graphviz(model, out_file=None,
                           feature_names=X_train.columns,
                           rounded=True, proportion=False,
                           precision=2, filled=True)
```

# Convert DOT data to a graph

```
graph = pydotplus.graph_from_dot_data(dot_data)
```

# Display the graph

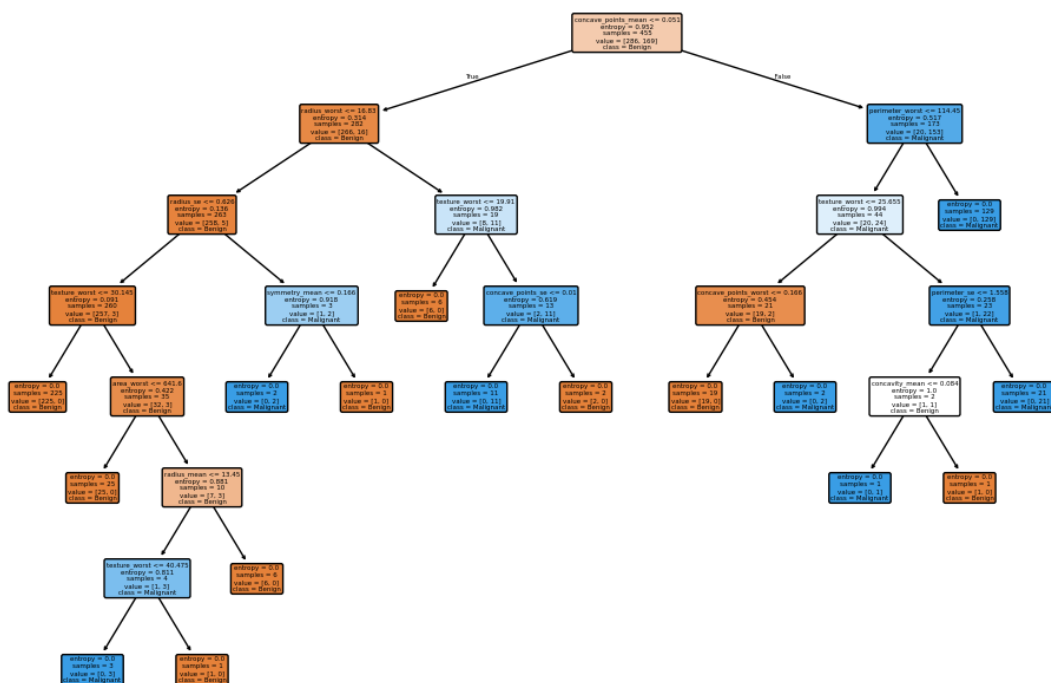
```
Image(graph.create_png())
```



# Visualize the Decision Tree (optional)

```
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, feature_names=X.columns, class_names=['Benign',
                              'Malignant'], rounded=True)
plt.show()
```





```
y_pred = model.predict(X_test)
```

y\_pred

```
array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1])
```

## # Evaluate the model

```
accuracy = accuracy_score(y_test, y_pred) * 100
```

```
classification_rep = classification_report(y_test, y_pred)
```

```
# Print the results
```

```
print("Accuracy:", accuracy)
```

```
print("Classification Report:\n", classification_rep)
```

Accuracy: 94.73684210526315

## Classification Report:

	precision	recall	f1-score	support
0	0.93	0.99	0.96	71
1	0.97	0.88	0.93	43
accuracy			0.95	114
macro avg	0.95	0.93	0.94	114

```
weighted avg      0.95      0.95      0.95      114
```

```
df.head(1)
```

```

  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99        10.38         122.8      1001.0

  smoothness_mean  compactness_mean  concavity_mean  concave_points_mean  \
0          0.1184          0.2776          0.3001          0.1471

  symmetry_mean  fractal_dimension_mean  radius_se  texture_se  perimeter_se
\
0          0.2419          0.07871      1.095      0.9053          8.589

  area_se  smoothness_se  compactness_se  concavity_se  concave_points_se  \
0    153.4      0.006399      0.04904      0.05373          0.01587

  symmetry_se  fractal_dimension_se  radius_worst  texture_worst  \
0    0.03003          0.006193          25.38          17.33

  perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0          184.6      2019.0          0.1622          0.6656

  concavity_worst  concave_points_worst  symmetry_worst  \
0          0.7119          0.2654          0.4601

  fractal_dimension_worst
0          0.1189
```

```

new = [[12.5, 19.2, 80.0, 500.0, 0.085, 0.1, 0.05, 0.02, 0.17, 0.06,
        0.4, 1.0, 2.5, 40.0, 0.006, 0.02, 0.03, 0.01, 0.02, 0.003,
        16.0, 25.0, 105.0, 900.0, 0.13, 0.25, 0.28, 0.12, 0.29, 0.08]]
y_pred = model.predict(new)
```

```
# Output the prediction (0 = Benign, 1 = Malignant)
```

```

if y_pred[0] == 0:
    print("Prediction: Benign")
else:
    print("Prediction: Malignant")
```

```
Prediction: 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 set.**

The Olivetti Face Dataset is a collection of images of faces, used primarily for face recognition tasks. The dataset contains 400 images of 40 different individuals, with 10 images per person. The dataset was created for research in machine learning and pattern recognition, especially in the context of facial recognition.

The Olivetti dataset provides the following key features:

\*400 Images: Each image is a grayscale photo of a person's face.

*40 People: The dataset contains 40 different individuals, and each individual Has 10 different images.*

\*Image Size: Each image is 64x64 pixels, resulting in 4096 features (flattened vector) per image.

\*Target Labels: Each image is associated with a label representing the individual (0 to 39).

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

from sklearn.datasets import fetch_olivetti_faces
data = fetch_olivetti_faces()

data.keys()

dict_keys(['data', 'images', 'target', 'DESCR'])

print("Data Shape:", data.data.shape)
print("Target Shape:", data.target.shape)
print("There are {} unique persons in the dataset".format(len(np.unique(data.target))))
print("Size of each image is {}x{}".format(data.images.shape[1], data.images.shape[1]))

Data Shape: (400, 4096)
Target Shape: (400,)
There are 40 unique persons in the dataset
Size of each image is 64x64

def print_faces(images, target, top_n):
    # Ensure the number of images does not exceed available data
    top_n = min(top_n, len(images))

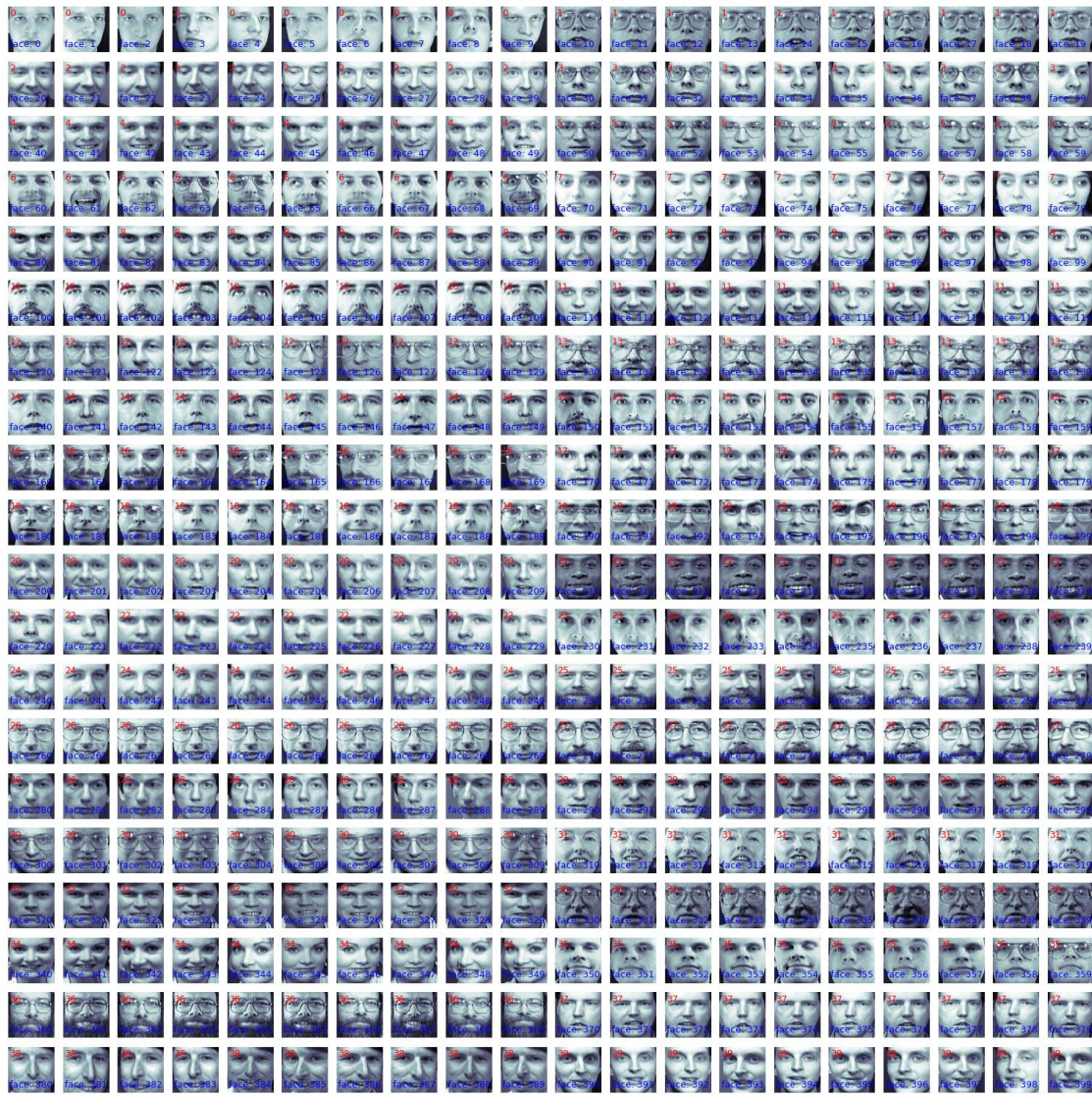
    # Set up figure size based on the number of images
    grid_size = int(np.ceil(np.sqrt(top_n)))
    fig, axes = plt.subplots(grid_size, grid_size, figsize=(15, 15))
```

```
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.2,
wspace=0.2)
```

```
for i, ax in enumerate(axes.ravel()):
    if i < top_n:
        ax.imshow(images[i], cmap='bone')
        ax.axis('off')
        ax.text(2, 12, str(target[i]), fontsize=9, color='red')
        ax.text(2, 55, f"face: {i}", fontsize=9, color='blue')
    else:
        ax.axis('off')
```

```
plt.show()
```

```
print_faces(data.images,data.target,400)
```





```

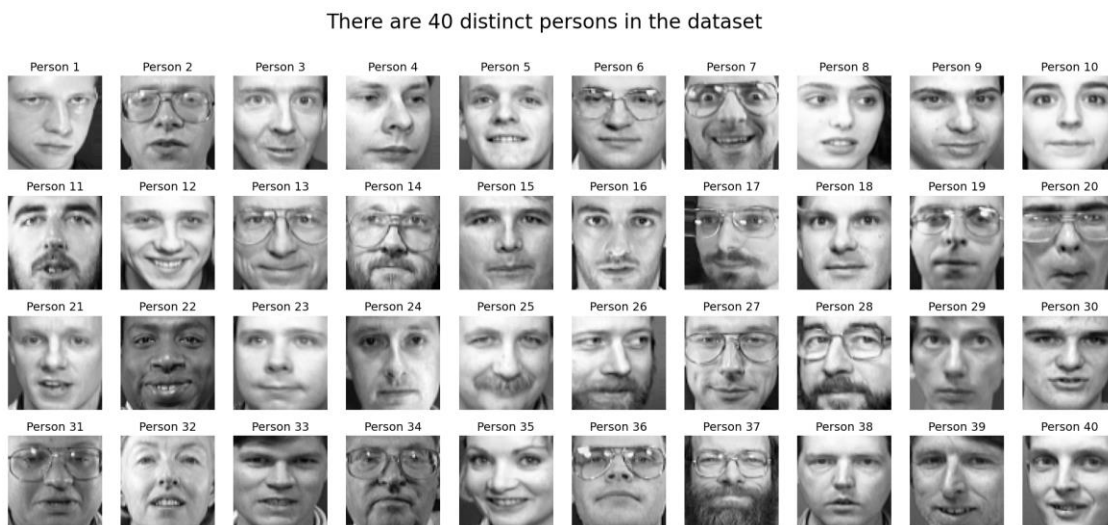
#Let us extract unique charaters present in dataset
def display_unique_faces(pics):
    fig = plt.figure(figsize=(24, 10)) # Set figure size
    columns, rows = 10, 4 # Define grid dimensions

    # Loop through grid positions and plot each image
    for i in range(1, columns * rows + 1):
        img_index = 10 * i - 1 # Calculate the image index
        if img_index < pics.shape[0]: # Check for valid image index
            img = pics[img_index, :, :]
            ax = fig.add_subplot(rows, columns, i)
            ax.imshow(img, cmap='gray')
            ax.set_title(f"Person {i}", fontsize=14)
            ax.axis('off')

    plt.suptitle("There are 40 distinct persons in the dataset", fontsize=24)
    plt.show()

display_unique_faces(data.images)

```



```

from sklearn.model_selection import train_test_split
X = data.data
Y = data.target
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,
random_state=46)

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)

x_train: (280, 4096)
x_test: (120, 4096)

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

```

```

# Train the model
nb = GaussianNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)

# Calculate accuracy
nb_accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)

# Display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Display accuracy result
print(f"Naive Bayes Accuracy: {nb_accuracy}%")

Confusion Matrix:
[[3 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 3 0]
 [1 0 0 ... 0 0 1]]
Naive Bayes Accuracy: 73.33%

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report

# Initialize and fit Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)

# Calculate accuracy
accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(f"Multinomial Naive Bayes Accuracy: {accuracy}%")

Multinomial Naive Bayes Accuracy: 85.83%

# Calculate the number of misclassified images
misclassified_idx = np.where(y_pred != y_test)[0]
num_misclassified = len(misclassified_idx)

```

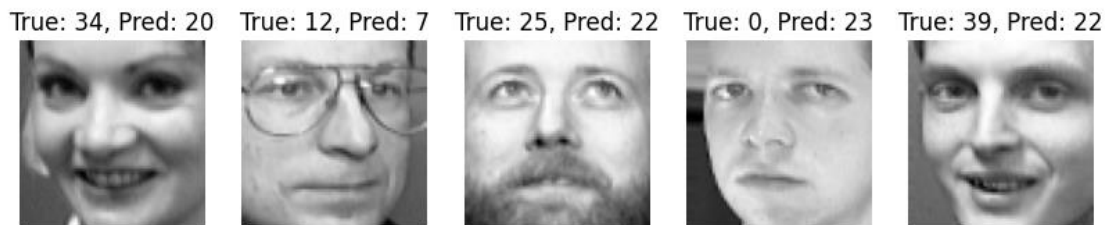
```

# Print the number of misclassified images and accuracy
print(f"Number of misclassified images: {num_misclassified}")
print(f"Total images in test set: {len(y_test)}")
print(f"Accuracy: {round((1 - num_misclassified / len(y_test)) * 100, 2)}%")

# Visualize some of the misclassified images
n_misclassified_to_show = min(num_misclassified, 5) # Show up to 5
misclassified_images
plt.figure(figsize=(10, 5))
for i in range(n_misclassified_to_show):
    idx = misclassified_idx[i]
    plt.subplot(1, n_misclassified_to_show, i + 1)
    plt.imshow(x_test[idx].reshape(64, 64), cmap='gray')
    plt.title(f"True: {y_test[idx]}, Pred: {y_pred[idx]}")
    plt.axis('off')
plt.show()

```

Number of misclassified images: 17  
 Total images in test set: 120  
 Accuracy: 85.83%



```

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score

```

```

# Binarize the test labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Get predicted probabilities for each class
y_pred_prob = nb.predict_proba(x_test)

# Calculate and print AUC for each class
for i in range(y_test_bin.shape[1]):
    roc_auc = roc_auc_score(y_test_bin[:, i], y_pred_prob[:, i])
    print(f"Class {i} AUC: {roc_auc:.2f}")

Class 0 AUC: 0.92
Class 1 AUC: 1.00
Class 2 AUC: 1.00
Class 3 AUC: 1.00
Class 4 AUC: 1.00
Class 5 AUC: 1.00
Class 6 AUC: 1.00

```

Class 7 AUC: 1.00  
Class 8 AUC: 1.00  
Class 9 AUC: 1.00  
Class 10 AUC: 1.00  
Class 11 AUC: 1.00  
Class 12 AUC: 0.87  
Class 13 AUC: 1.00  
Class 14 AUC: 1.00  
Class 15 AUC: 1.00  
Class 16 AUC: 0.65  
Class 17 AUC: 0.16  
Class 18 AUC: 0.36  
Class 19 AUC: 0.89  
Class 20 AUC: 0.52  
Class 21 AUC: 0.81  
Class 22 AUC: 0.13  
Class 23 AUC: 0.34  
Class 24 AUC: 0.64  
Class 25 AUC: 0.55  
Class 26 AUC: 0.48  
Class 27 AUC: 0.38  
Class 28 AUC: 0.62  
Class 29 AUC: 0.73  
Class 30 AUC: 0.55  
Class 31 AUC: 0.17  
Class 32 AUC: 0.47  
Class 33 AUC: 0.67  
Class 34 AUC: 0.31  
Class 35 AUC: 0.03  
Class 36 AUC: 0.91  
Class 37 AUC: 0.87  
Class 38 AUC: 0.47



Program 10:

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

### Steps to Implement K-Means Clustering

1. **Load the dataset** – Use sklearn.datasets to fetch the Wisconsin Breast Cancer dataset.
2. **Preprocess the data** – Normalize features for better clustering.
3. **Apply K-Means algorithm** – Use KMeans from sklearn.cluster.
4. **Evaluate clustering performance** – Compare with actual labels using ARI or silhouette score.
5. **Visualize clusters** – Use PCA for dimensionality reduction and plot clusters.

### 1. Install and Import Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, adjusted_rand_score
```

### 2. Load the Wisconsin Breast Cancer Dataset

```
# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target # Ground truth labels (0 - Malignant, 1 - Benign)
feature_names = data.feature_names

# Convert to DataFrame for better visualization
```

```
df = pd.DataFrame(X, columns=feature_names)
df['target'] = y # Add target labels
df.head()
```

### **3. Normalize the Data**

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # Standardize features
```

### **4. Apply K-Means Clustering**

```
# Choose number of clusters (K=2 as we have benign/malignant labels)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
y_pred = kmeans.fit_predict(X_scaled)

# Add cluster labels to DataFrame
df['cluster'] = y_pred
```

### **5. Evaluate Clustering Performance**

```
# Compare predicted clusters with actual labels
ari_score = adjusted_rand_score(y, y_pred)
silhouette_avg = silhouette_score(X_scaled, y_pred)

print(f"Adjusted Rand Index: {ari_score:.3f}")
print(f"Silhouette Score: {silhouette_avg:.3f}")
```

### **6. Visualize Clusters using PCA (2D Projection)**

```
# Reduce data to 2D using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

```
# Plot the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y_pred, palette='coolwarm', alpha=0.7)
plt.title("K-Means Clustering on Breast Cancer Dataset (PCA Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.show()
```

## Results & Insights

- 1.The **Adjusted Rand Index (ARI)** measures how well the clustering matches the actual labels (closer to 1 means better clustering).
- 2.The **Silhouette Score** measures how well-separated the clusters are.
- 3.The **PCA scatter plot** provides a 2D visualization of the clusters.