ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

(DATA SCIENCE)

(ACADEMIC YEAR 2024-25)

LABORATORY MANUAL

SUBJECT: MACHINE LEARNING LABORATORY

SUB CODE: BCSL606

Prepared By

Dr.Anitha D B

Associate professor & HoD

Ms. Bhoomika A L

Instructor

Approved By

Dr. Anitha D B

HOD, CSE-DSE

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow’s society.
- To strive to attain ever-higher benchmarks of educational excellence.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND ENGINEERING

(DATA SCIENCE &ENGINEERING)

Vision of The Department

- To impart technical education in the field of data science of excellent quality with a high level of professional competence, social responsibility, and global awareness among the students

Mission

- To impart technical education that is up to date, relevant and makes students competitive and employable at global level
- To provide technical education with a high sense of discipline, social relevance in an intellectually, ethically and socially challenging environment for better tomorrow
- Educate to the global standards with a benchmark of excellence and to kindle the spirit of innovation.

Program Outcomes(PO)

- **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

- **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

- **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.
- **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.

- **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.

- **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
- **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.

- **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **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.

- **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.
- **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 (PSOs)

- PSO1: Develop relevant programming skills to become a successful data scientist
- PSO2: Apply data science concepts and algorithms to solve real world problems of the society
- PSO3: Apply data science techniques in the various domains like agriculture, education healthcare for better society

Program Educational Objectives (PEOs):

PEO1: Develop cutting-edge skills in data science and its related technologies, such as machine learning, predictive analytic, and data engineering.

PEO2: Design and develop data-driven solutions to real-world problems in a business, research, or social environment.

PEO3: Apply data engineering and data visualization techniques to discover, investigate, and interpret data.

PEO4: Demonstrate ethical and responsible data practices in problem solving

PEO5: Integrate fields within computer science, optimization, and statistics to develop better solutions

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:

1. To become familiar with data and visualize univariate, bivariate, and multivariate data using statistical techniques and dimensionality reduction.
2. To understand various machine learning algorithms such as similarity-based learning, regression, decision trees, and clustering.
3. 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. Label the first 50 points {*x1*,……,*x50*} as follows: if (xi ≤ 0.5), then *xi* ∊ Class1, else *xi* ∊ Class1 Classify the remaining points, *x51*,……,*x100* 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**

Experiment 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.

Introduction

Data visualization is a crucial step in exploratory data analysis (EDA), enabling data scientists to understand the distribution and spread of numerical features. Two widely used visualization techniques for analyzing numerical data are histograms and box plots. These plots help identify patterns, trends, and potential anomalies in datasets, making them valuable tools for data preprocessing and feature engineering.

Distribution

In statistics, distribution refers to how data values are spread across a range. Understanding the distribution of numerical features in a dataset helps in identifying patterns, detecting outliers, and making informed decisions. The two primary ways to visualize distribution are histograms and box plots.

1. Histograms

A histogram is a graphical representation of the distribution of a numerical feature. It divides the data into bins (intervals) and counts the number of observations in each bin.

Importance of Histograms:

- **Detecting Skewness:** A histogram can reveal whether a distribution is symmetric, left-skewed, right Skewed.
- **Identifying Modal Patterns:** Some distributions are unimodal (single peak), while others may be bimodal or multimodal.
- **Assessing Normality:** If the histogram resembles a bell curve, the data may be normally distributed.
- **Understanding Data Spread:** Helps in detecting whether data is evenly distributed or concentrated in certain regions.

2. Box Plots (Box-and-Whisker Plots)

A box plot provides a summary of the distribution of numerical data using five key statistics:

- Minimum: The smallest value (excluding outliers).
- First Quartile (Q1): 25th percentile.
- Median (Q2): 50th percentile (middle value).
- Maximum: The largest value (excluding outliers).
- Outliers are detected using the Interquartile Range (IQR) rule: Outliers = Values outside Q1 - 1.5 * IQR or Q3 + 1.5 * IQR.

Importance of Box Plots

- **Identifying Outliers:** Points lying outside the whiskers indicate potential outliers.
- **Comparing Distributions:** Box plots allow easy comparison of multiple features or groups.
- **Understanding Skewness:** If the median is closer to one end, the distribution may be skewed.
- **Measuring Data Spread:** The length of the box and whiskers provides insight into data variability.

Outlier

An outlier is an observation or data point that significantly differs from the rest of the data in a dataset. Outliers can skew statistical analyses and distort the interpretation of results, making it important to identify and understand them.

Key Characteristics of Outliers:

- Deviation from the Norm:
  - Outliers exhibit values that deviate substantially from the typical or expected range of values in a dataset.
- Impact on Statistical Measures:
  - Outliers can heavily influence summary statistics such as the mean and standard deviation, leading to misleading representations of central tendency and dispersion.
- Identification:
  - Outliers are often identified through statistical methods or visual inspection of graphs; such as box plots or scatter plots.
- Causes of Outliers:
  - Outliers can arise from measurement errors, data entry mistakes, natural variability, or genuine extreme observations in the population.

Ways to Identify Outliers:

- Visual Inspection:

  Plotting the data using graphs like box plots, scatter plots, or histograms can reveal observations that stand out from the majority.

- Statistical Methods:

  Z-Score: Identifying data points with z-scores beyond a certain threshold (e.g., $|z| > 3$) as potential outliers.

  $Z = (x-\mu)/\sigma$

- Interquartile Range (IQR): Using the IQR to identify observations outside a defined range.

  $IQR = Q3 - Q1$

  $LF = Q1 - (1.5*IQR)$

  $UF = Q3 + (1.5*IQR)$

Dealing with Outliers:

Retaining Outliers:

- In some cases, it may be appropriate to retain outliers, especially if they represent genuine extreme values in the data.
- Retaining outliers allows for an inclusive analysis, considering the full range of variability in the dataset.

Removing Outliers:

- Removing outliers involves excluding extreme values from the dataset before analysis.
- Common methods include using statistical criteria (e.g., Z-scores, IQR) to identify and exclude observations beyond a certain threshold.
- Reduces the impact of extreme values on summary statistics and model results
- Loss of information: Excluding outliers may discard meaningful data points.

Transformation:

- Transformation involves applying mathematical functions to the data to modify its distribution and reduce the impact of outliers.
- Common transformations include logarithmic, square root, or Cube root transformations.

Application in Data Analysis

- Histograms and box plots play a crucial role in:
- Data Cleaning: Detecting anomalies and erroneous values.
- Feature Engineering: Identifying transformations needed for better model performance.
- Understanding Dataset Characteristics: Providing insight into feature distributions,which informs modeling decisions.

About Datasets

Context

This is the dataset used in the second chapter of Aurélien Géron's recent book 'Hands-On Machine learning with Scikit-Learn and TensorFlow'. It serves as an excellent introduction to implementing machine learning algorithms because it requires rudimentary data cleaning, has an easily understandable list of variables and sits at an optimal size between being to toyish and too cumbersome.

The data contains information from the 1990 California census. So although it may not help you with predicting current housing prices like the Zillow Zestimate dataset, it does provide an accessible introductory dataset for teaching people about the basics of machine learning.

Content

The data pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. Be warned the data aren't cleaned so there are some preprocessing steps required! The columns are as follows, their names are pretty self explanitory:

longitude latitude

housing_median_age

total_rooms total_bedrooms

population households

median_income

median_house_value(Target)

ocean_proximity

Import Necessary Libraries

Import all libraries which are required for our analysis, such as Data Loading, Statistical analysis, Visualizations, Data Transformations, Merge and Joins, etc.

Longitude: 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.

Latitude: Similar to longitude, the latitude distribution shows houses concentrated in particular zones. This suggests geographic clustering, possibly around major cities.

Housing Median Age: 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.

Total Rooms: 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).

Median Income: 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.

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.

Median House Value: 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.

Population: 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.

Median House Value: 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.

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 Rooms: There are numerous data points above the upper whisker, indicating a significant number of outliers.

3. Total Bedrooms: Numerous data points above the upper whisker indicate a significant presence of outliers with very high total_bedrooms values.

1. Population: There are numerous outliers above the upper whisker, with extreme population values reaching beyond 35,000.

2. Households There is a significant number of outliers above the upper whisker. These values represent areas with an unusually high number of households.

3. Median Income: There are numerous data points above the upper whisker, marked as circles. These are considered potential outliers.

4. Median House Value: A small cluster of outliers is visible near the maximum value of 500,000.

General Actions for Outlier Handling:

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

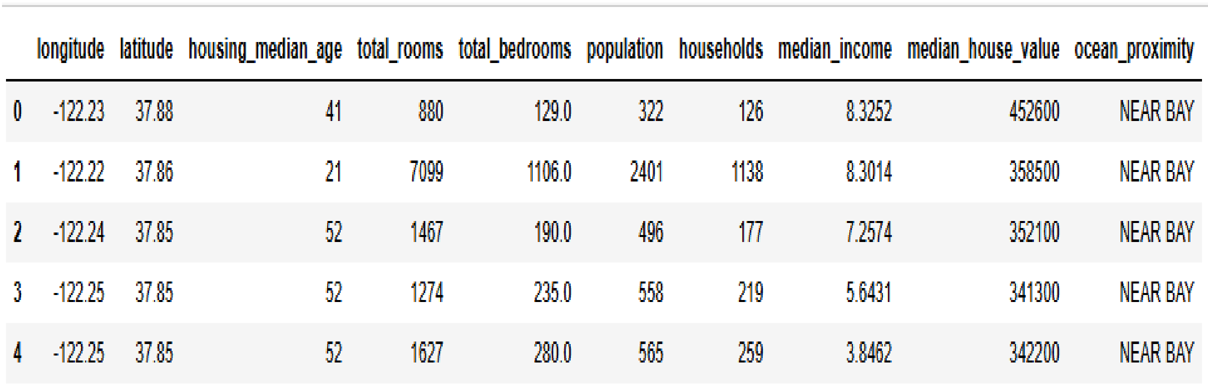
Program

#Import Necessary libraries :

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

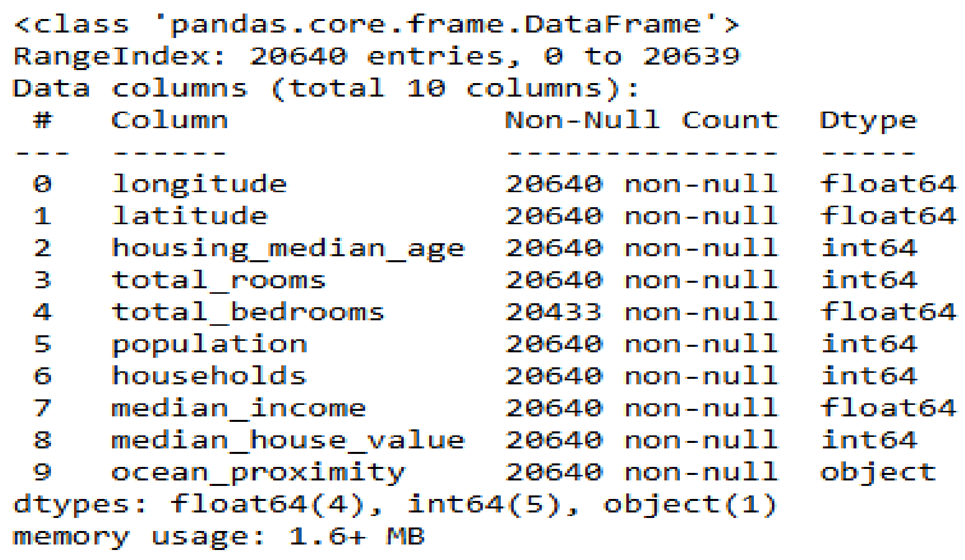
#import warnings

```
warnings.filterwarnings('ignore')
df = pd.read_csv("C:/Users/student/Desktop/4AD22CD042/housing.csv")
df.head()
```



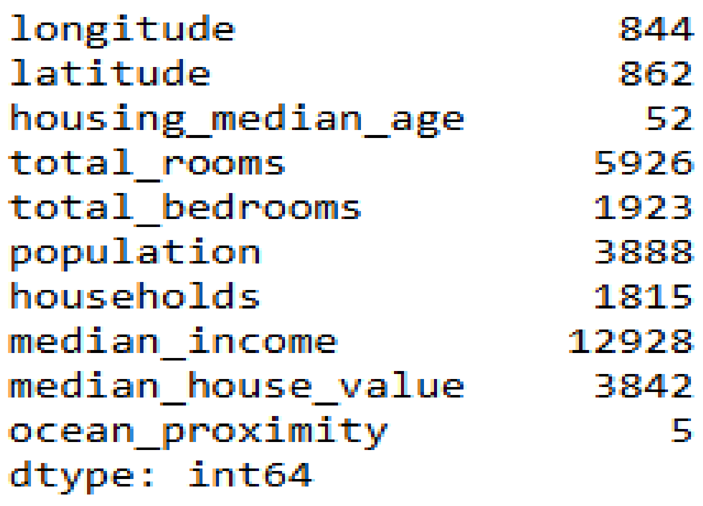
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BAY

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  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  int64  
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(4), int64(5), 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	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY

#Discriptive Statistics
df.describe().T

	count	mean	std	min	25%	50%	75%	max
longitude	20640.0	-119.569704	2.003532	-124.3500	-121.8000	-118.4900	-118.01000	-114.3100
latitude	20640.0	35.631861	2.135952	32.5400	33.9300	34.2600	37.71000	41.9500
housing_median_age	20640.0	28.639486	12.585558	1.0000	18.0000	29.0000	37.00000	52.0000
total_rooms	20640.0	2635.763081	2181.615252	2.0000	1447.7500	2127.0000	3148.00000	39320.0000
total_bedrooms	20640.0	536.838857	419.391878	1.0000	297.0000	435.0000	643.25000	6445.0000
population	20640.0	1425.476744	1132.462122	3.0000	787.0000	1166.0000	1725.00000	35682.0000
households	20640.0	499.539680	382.329753	1.0000	280.0000	409.0000	605.00000	6082.0000
median_income	20640.0	3.870671	1.899822	0.4999	2.5634	3.5348	4.74325	15.0001
median_house_value	20640.0	206855.816909	115395.615874	14999.0000	119600.0000	179700.0000	264725.00000	500001.0000

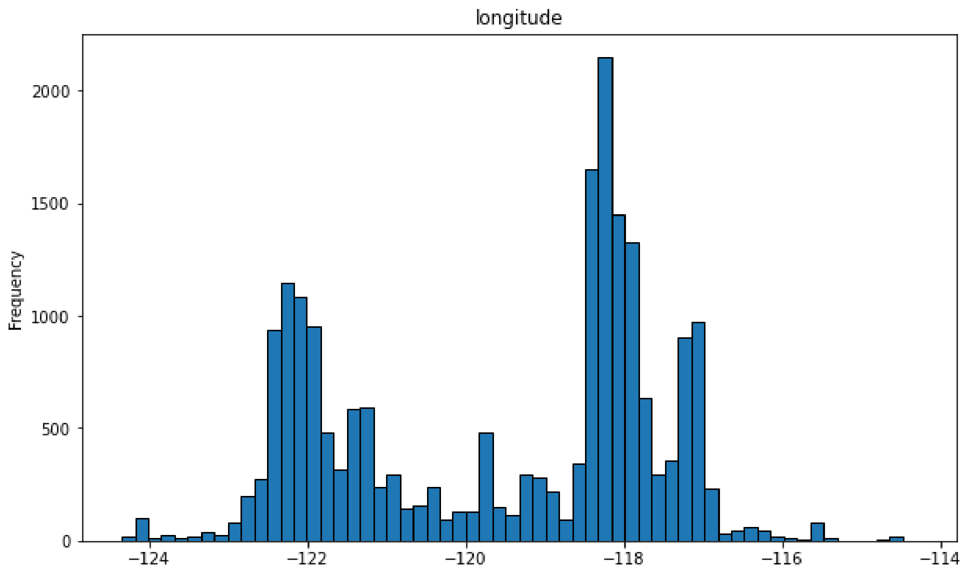
Numerical = df.select_dtypes(include=[np.number]).columns

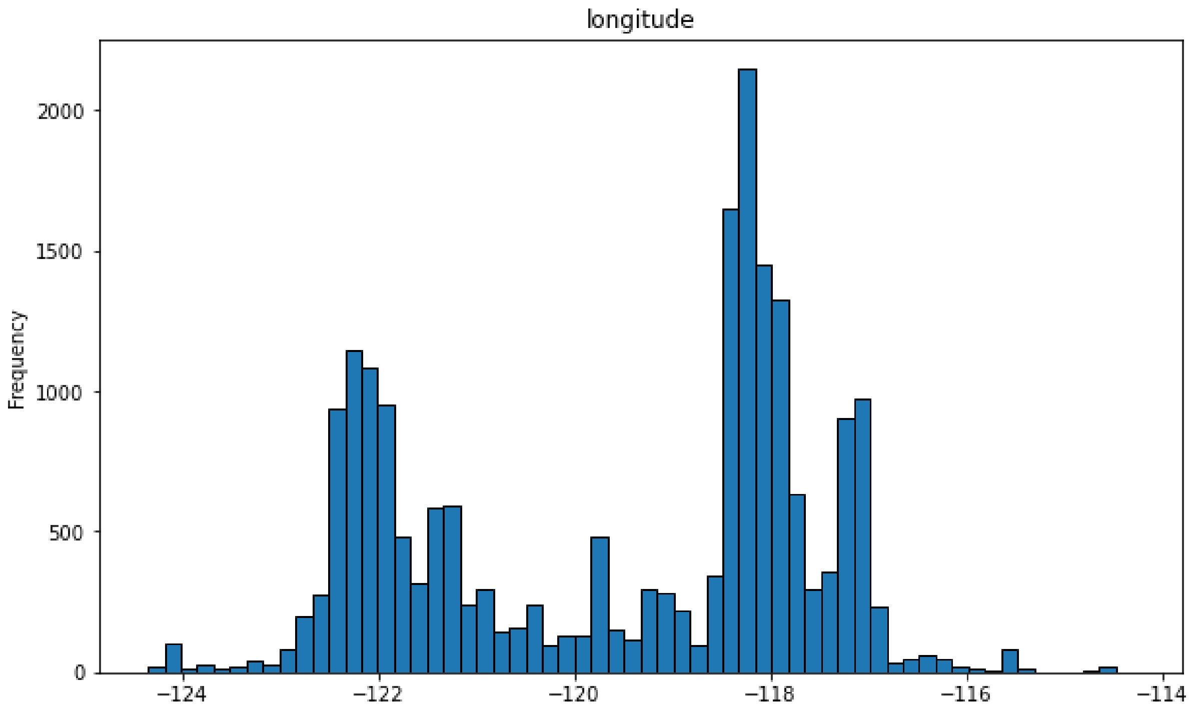
print(Numerical)

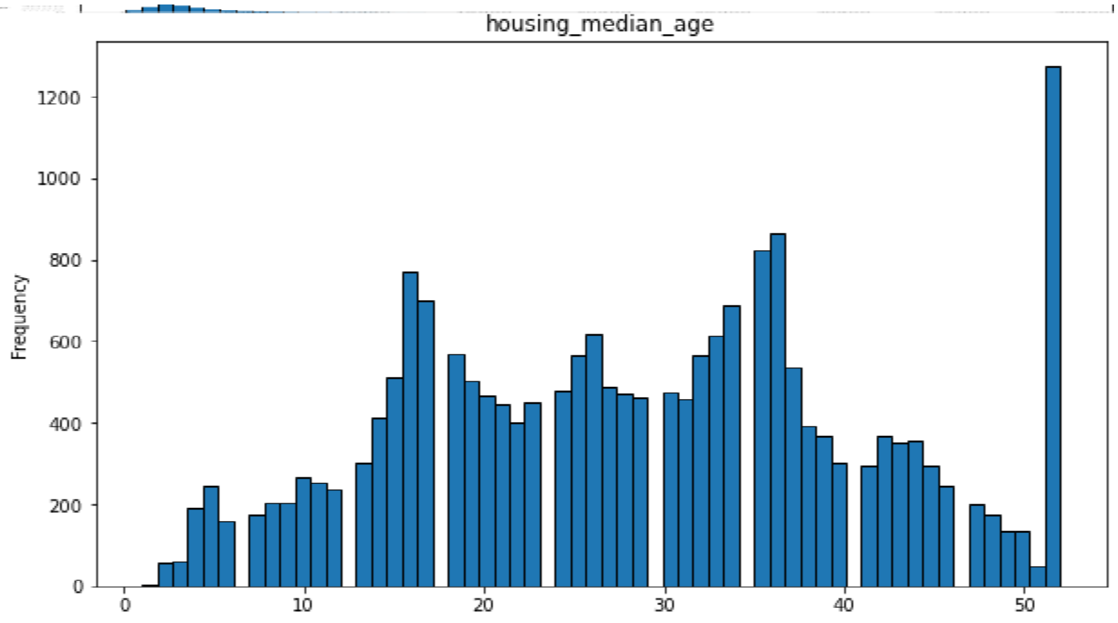
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

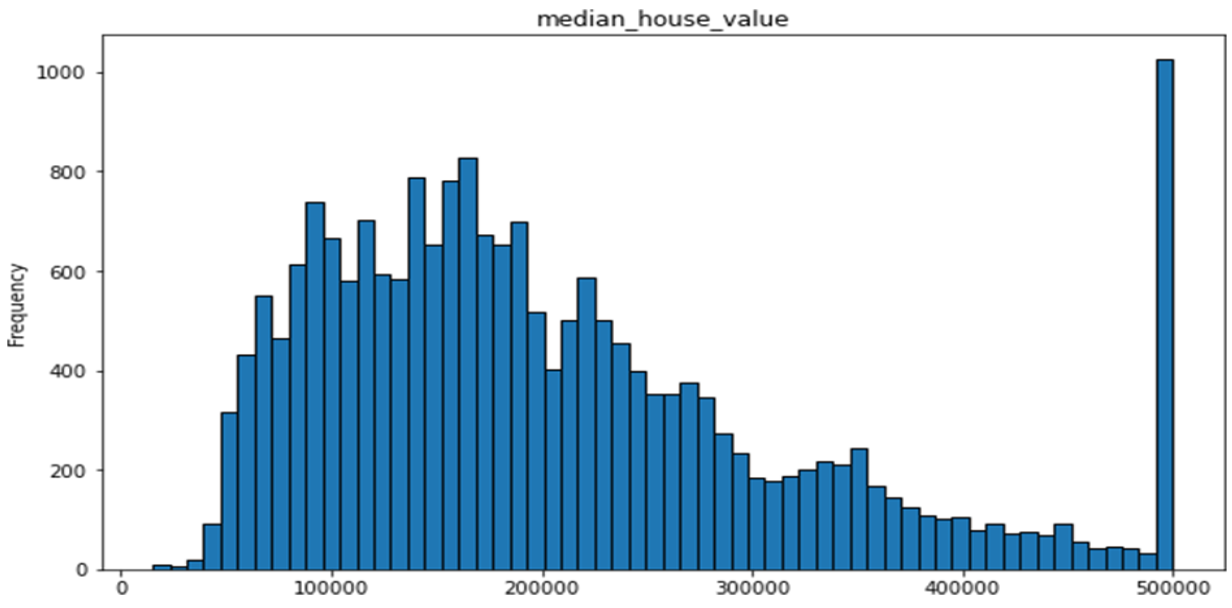
#Histogram of all Numerical Features

```
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()
```



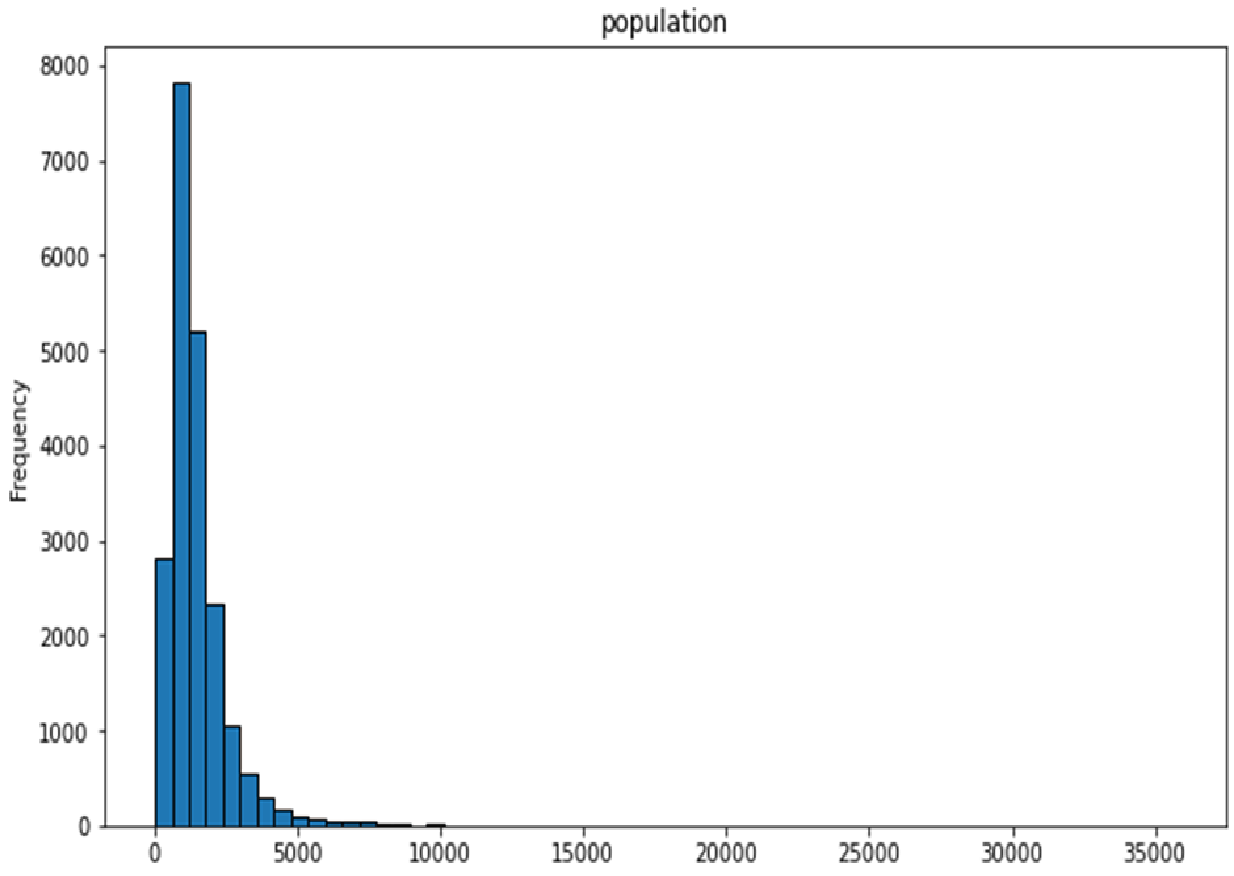

longitude
Frequency
2000
1500
1000
500
0
−124
−122
−120
−118
−116
−114


housing_median_age
Frequency
1200
1000
800
600
400
200
0
0
10
20
30
40
50

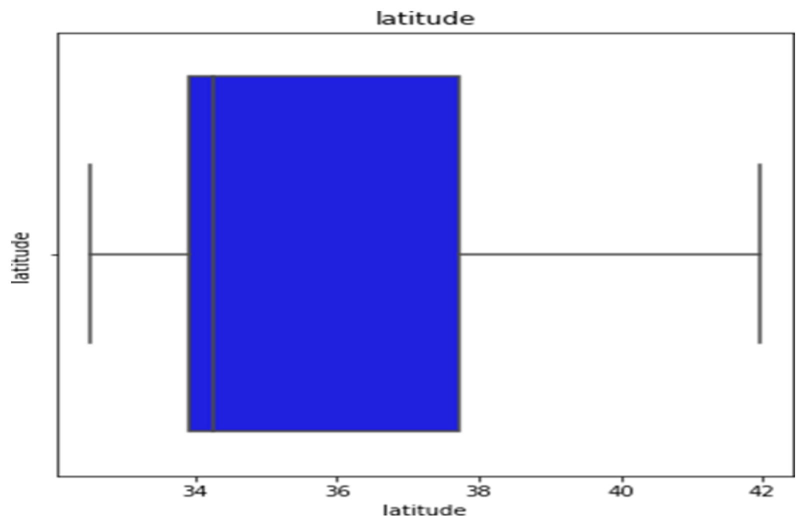


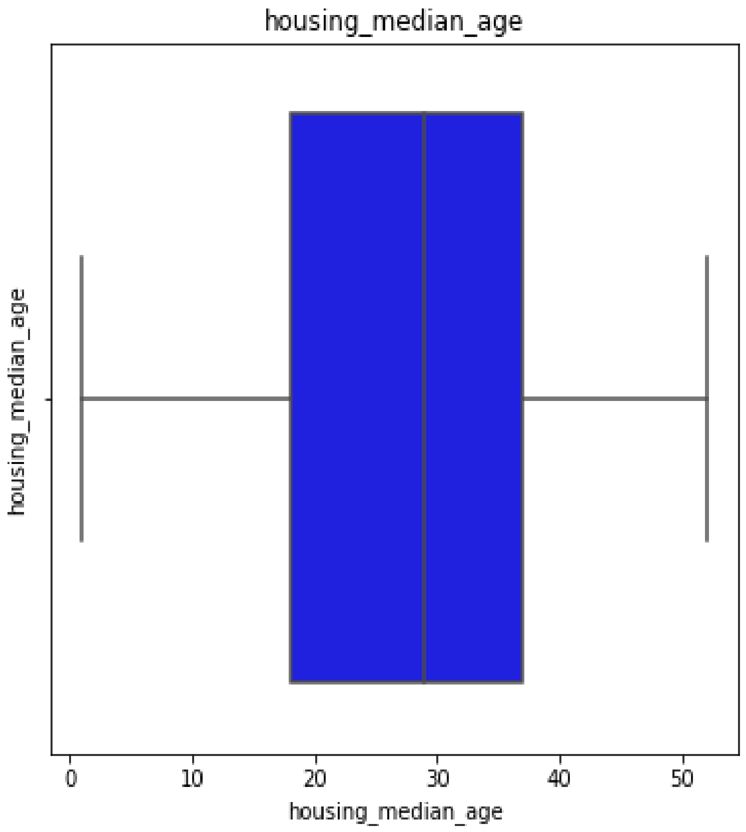
#Box plot of all numerical Features

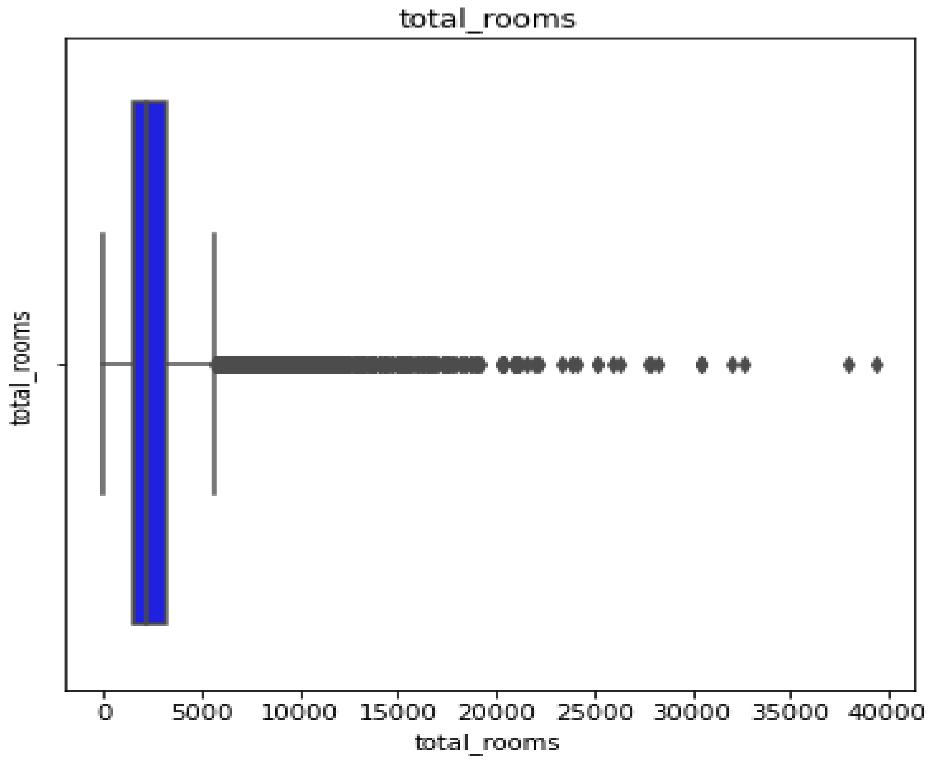
for col in Numerical:

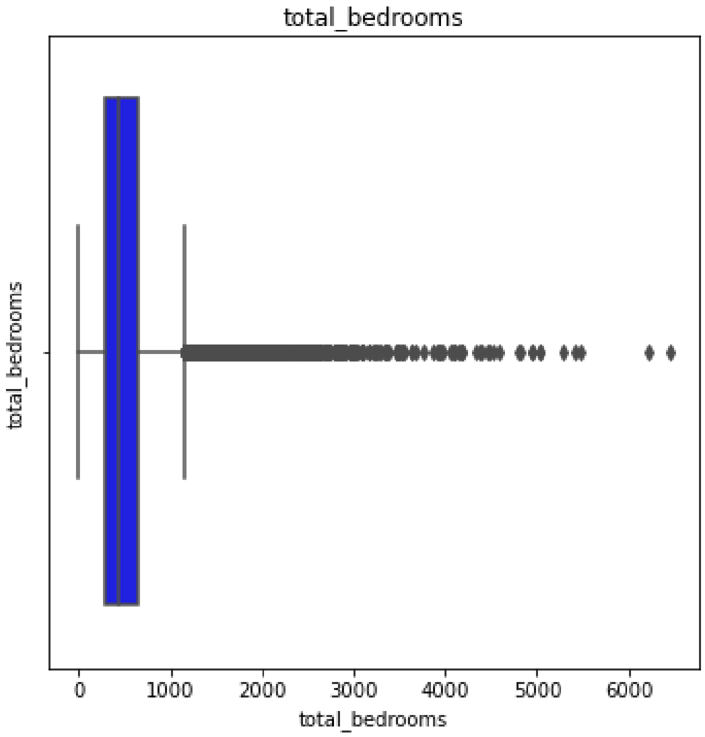


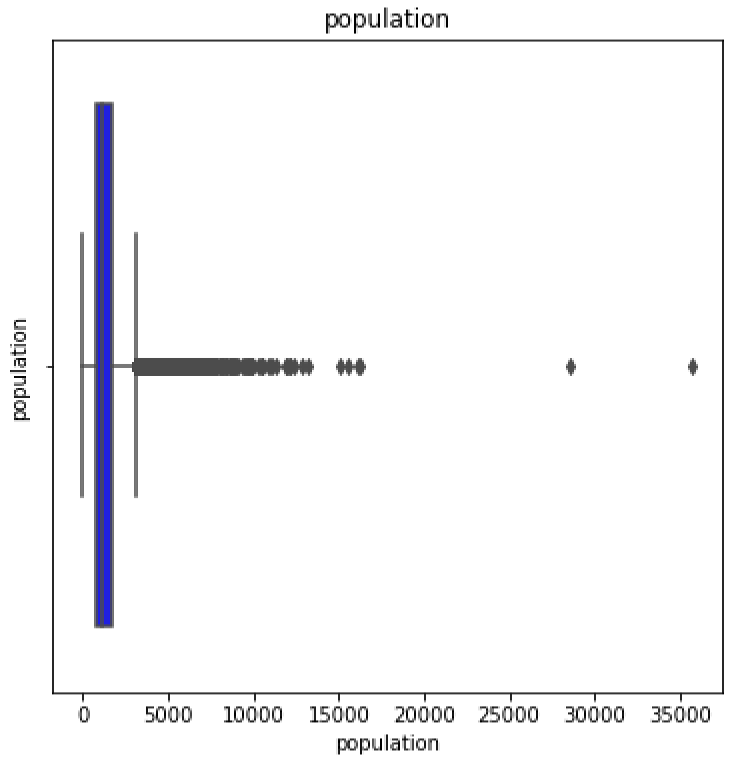
```
plt.figure(figsize=(6, 6))
  sns.boxplot(df[col], color='blue')
  plt.title(col)
  plt.ylabel(col)
  plt.show()
```

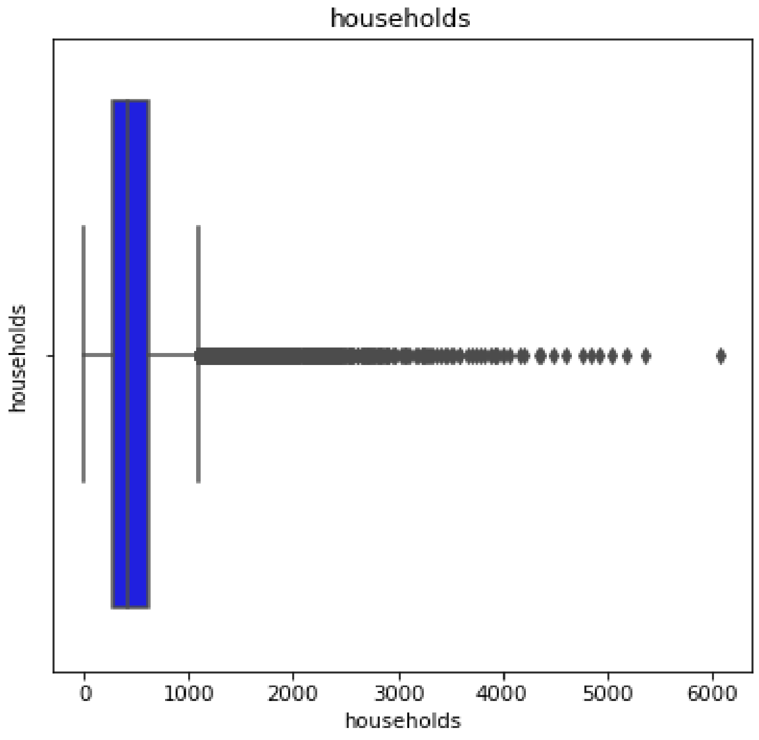


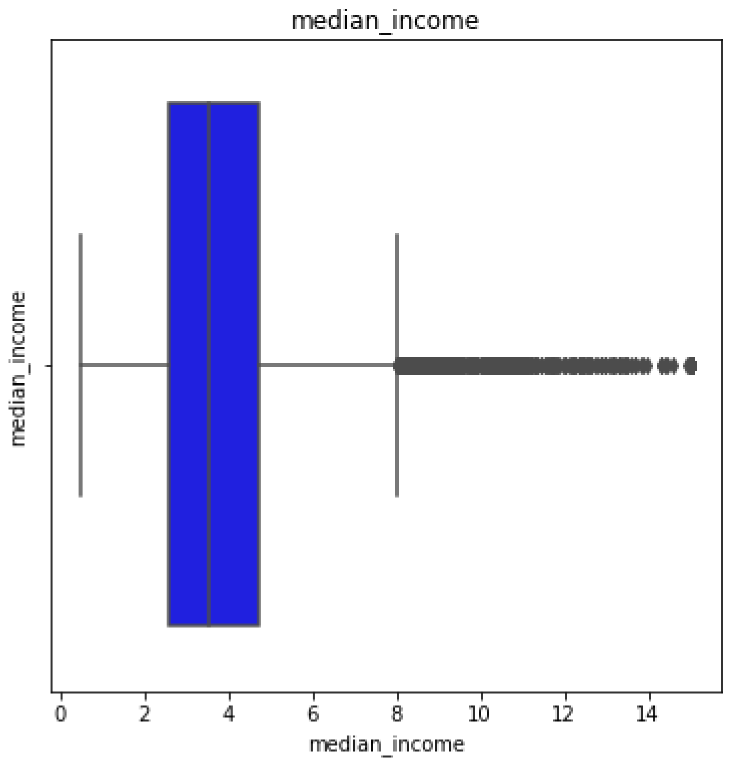


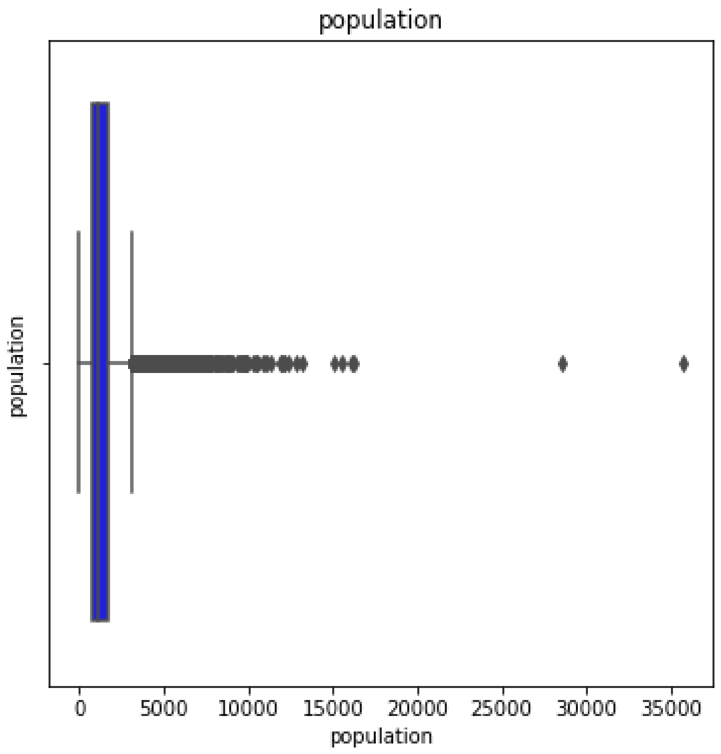

total_rooms
total_rooms
0
5000
10000
15000
20000
25000
30000
35000
40000
total_rooms


total_bedrooms
total_bedrooms
0
1000
2000
3000
4000
5000
6000
total_bedrooms


population
population
0
5000
10000
15000
20000
25000
30000
35000
population


households
households
0
1000
2000
3000
4000
5000
6000
households


median_income
median_income
0
2
4
6
8
10
12
14
median_income


population
population
0
5000
10000
15000
20000
25000
30000
35000
population

Experiment 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.

Introduction

In data analysis and machine learning, understanding the relationships between features is crucial for feature selection, multicollinearity detection, and data interpretation. Correlation and pair plots are two essential techniques to analyze these relationships.

Correlation Matrix

A **correlation matrix** is a table showing correlation coefficients between variables. It helps in understanding how strongly features are related to each other.

Types of Correlation

- **Positive Correlation (+1 to 0)**: As one feature increases, the other also increases.
- **Negative Correlation (0 to -1)**: As one feature increases, the other decreases.
- **No Correlation (0)**: No linear relationship between the variables.

Why Should You Use a Correlation Matrix?

- Identifies relationships between features.
- Helps in detecting multicollinearity in machine learning models.
- Highlights redundant features that may not add value to the model.

In data analysis and machine learning, understanding the relationships between features is crucial for feature selection, multicollinearity detection, and data interpretation. Correlation and pair plots are two essential techniques to analyze these relationships.

1. Correlation Matrix

A **correlation matrix** is a table showing correlation coefficients between variables. It helps in understanding how strongly features are related to each other.

Types of Correlation

- **Positive Correlation (+1 to 0)**: As one feature increases, the other also increases.

- **Negative Correlation (0 to -1)**: As one feature increases, the other decreases.
- **No Correlation (0)**: No linear relationship between the variables.

Why Should You Use a Correlation Matrix?

- Identifies relationships between features.
- Helps in detecting multicollinearity in machine learning models.
- Highlights redundant features that may not add value to the model.

Heatmap for Correlation Matrix:

A **heatmap** is a visual representation of the correlation matrix. It uses color coding to indicate the strength of relationships between variables.

Benefits of Using a Heatmap

- Easy to interpret relationships between features.
- Quickly identifies highly correlated variables.
- Helps in feature selection and data preprocessing

3.Pair Plot

A **pair plot** (also known as a scatterplot matrix) is a collection of scatter plots for every pair of numerical variables in the dataset. It helps in visualizing relationships between variables.

Why Use a Pair Plot?

- Shows the distribution of individual features along the diagonal.
- Displays relationships between features using scatter plots.
- Helps in identifying clusters, trends, and potential outliers.

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*IQR or Q3 + 1.5*IQR).

Program

#import necessary Libraries

import numpy as np

import pandas as pd

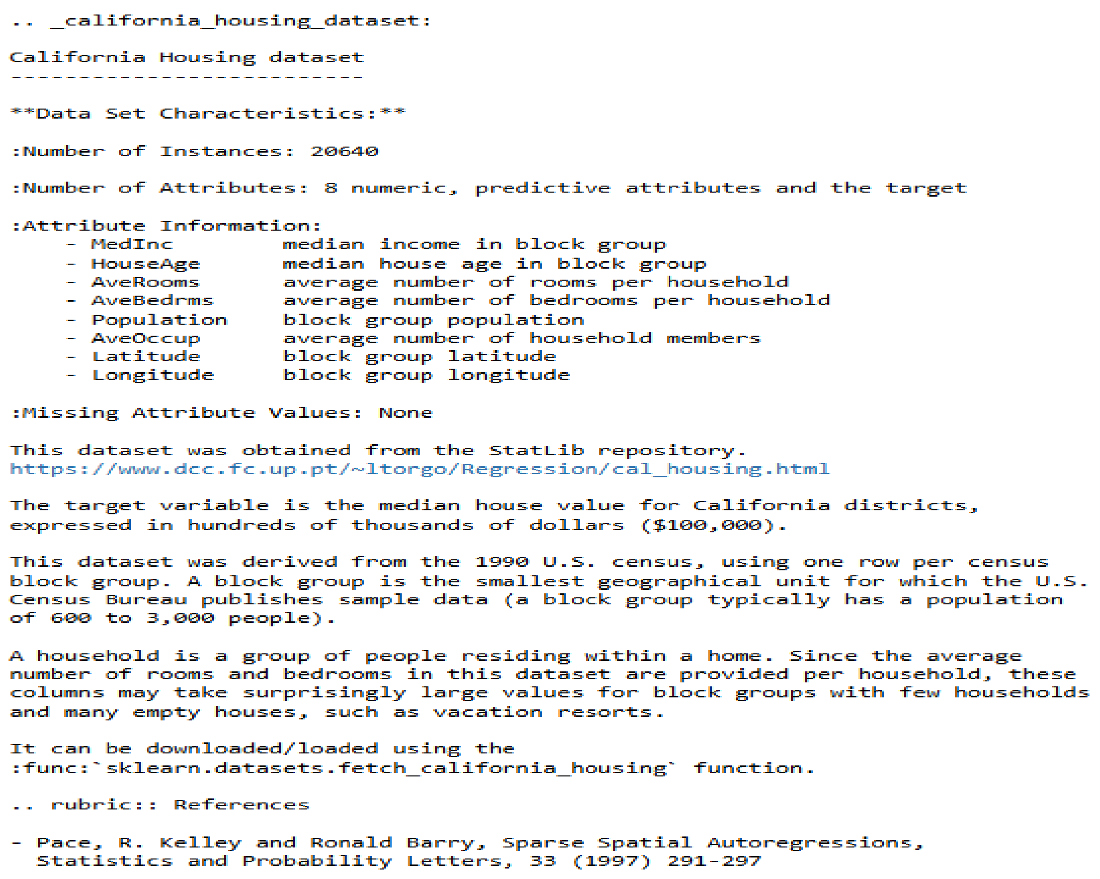
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch_california_housing

#load California Housing dataset

df = fetch_california_housing()



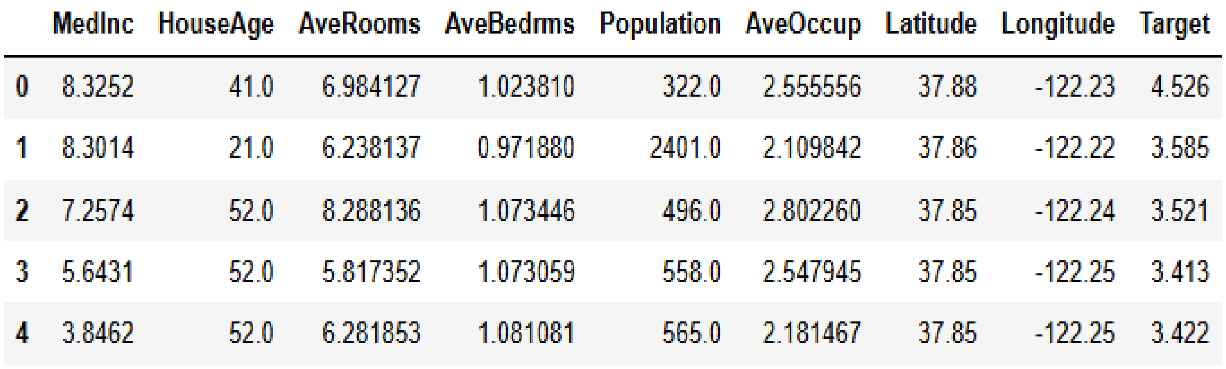
```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - 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    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude

:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297
```

#convert to Dataframe

df = pd.DataFrame(data.data, columns=data.feature_names)

df['Target'] = data.target

df.head()



	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

# Correlation Matrix

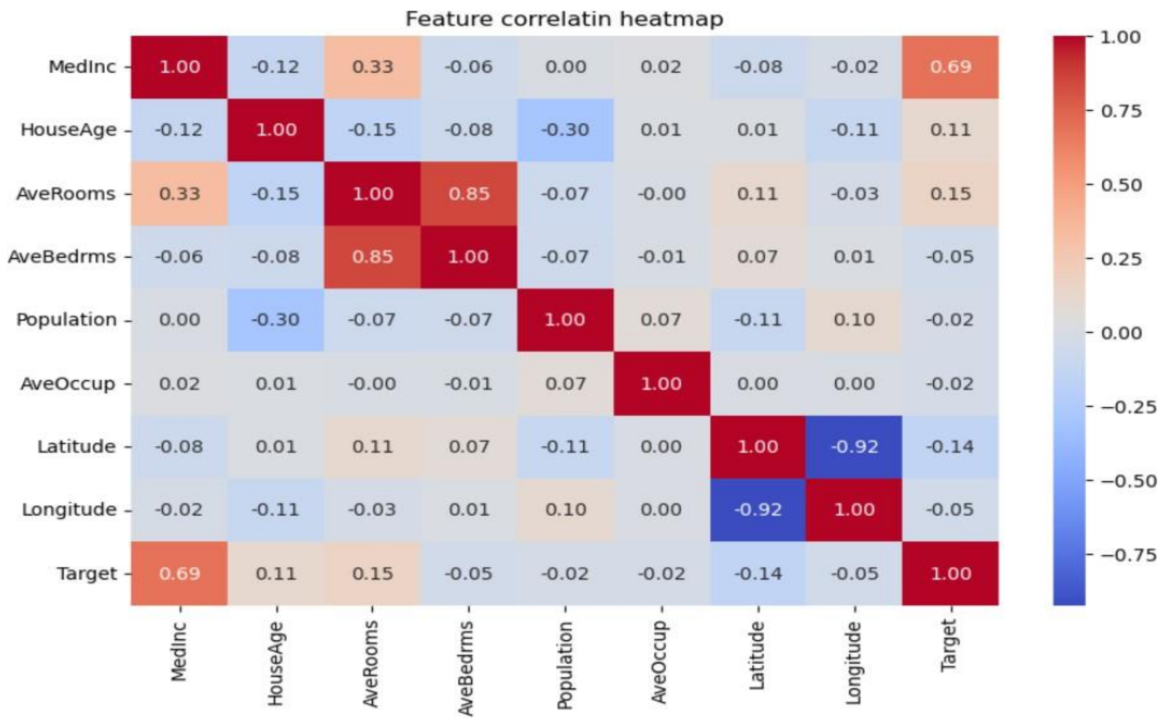
plt.figure(figsize=(10, 6))

corr_matrix = df.corr()

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title("Feature correlatin heatmap")

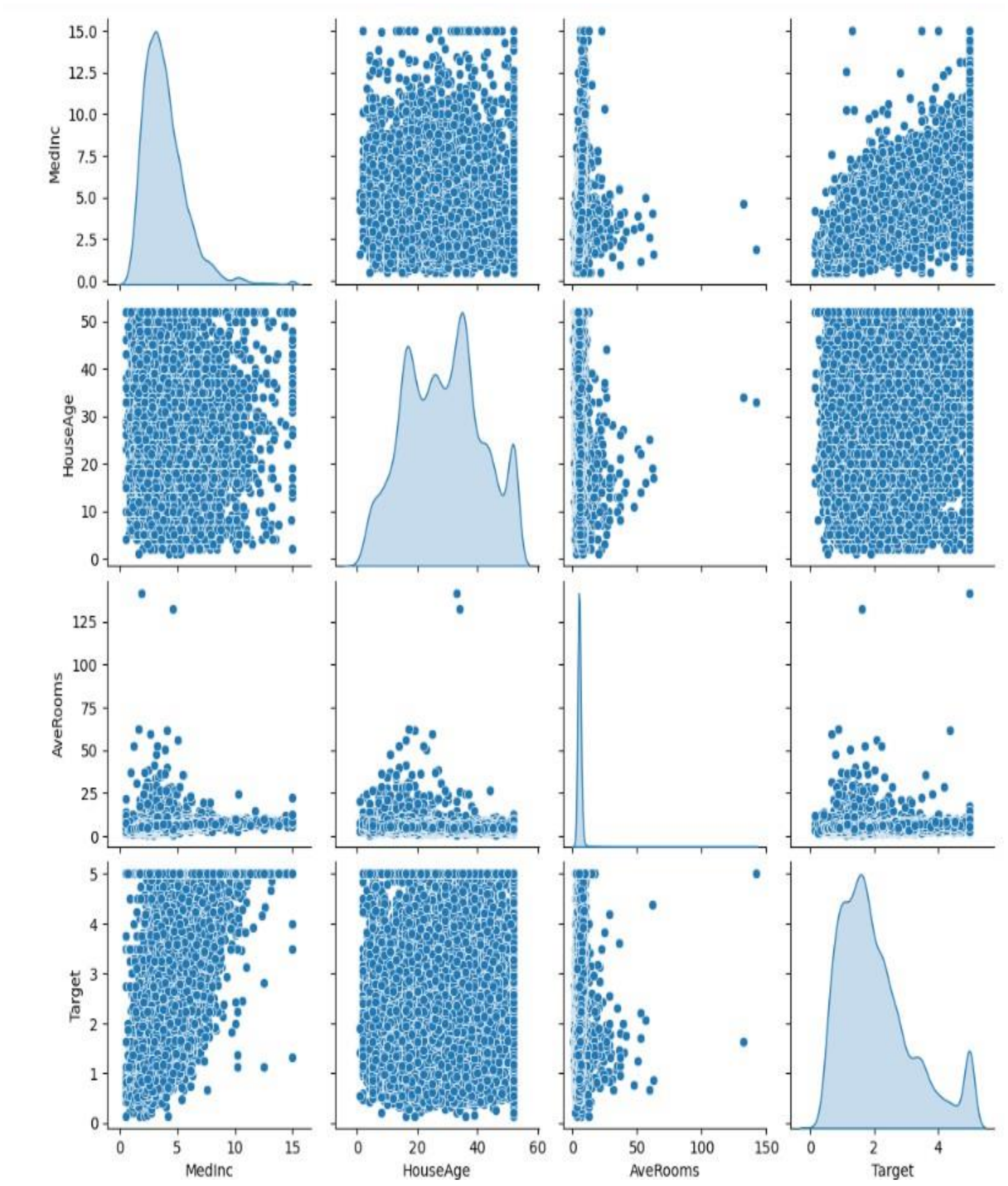
plt.show()



	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

#pairplot to analyze feature relationships

```
sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'Target']], diag_kind='kde')

plt.show()
```



Experiment 3

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

Introduction to Principal Component Analysis (PCA)

What is PCA?

Principal Component Analysis (PCA) is a **dimensionality reduction technique** used to transform a high-dimensional dataset into a lower-dimensional space while retaining as much variance as possible. It is an unsupervised learning method commonly used in machine learning and data visualization.

Importance of PCA

- ➢ Reduces computational complexity by lowering the number of features.
- ➢ Helps in visualizing high-dimensional data.
- ➢ Removes redundant or correlated features, improving model performance. Reduces overfitting by eliminating noise in the data.

How Does PCA Work?

PCA follows these key steps:

1. **Standardization:** The data is normalized so that all features have a mean of zero and a standard deviation of one.
2. **Compute the Covariance Matrix:** This step helps in understanding how different features relate to each other.
3. **Eigenvalue & Eigenvector Calculation:** Eigenvectors represent the direction of the new feature axes, and eigenvalues determine the importance of these axes.
4. **Selecting Principal Components:** The eigenvectors corresponding to the highest eigenvalues are chosen to form the new feature space.
5. **Transforming Data:** The original dataset is projected onto the new feature space with reduced dimensions.

Applying PCA to the Iris Dataset

The **Iris dataset** consists of 4 numerical features (**sepal length, sepal width, petal length, petal width**) used to classify flowers into 3 species (**Setosa, Versicolor, and Virginica**).

- ➢ **Goal:** Reduce the **4-dimensional feature space** to **2 principal components** while retaining most of the variance.

- **Benefit:** Enables 2D visualization of the dataset, making it easier to interpret classification results.

Understanding PCA Output

1. Variance Explained by Each Principal Component

PCA provides **explained variance ratios**, which indicate how much information each principal component retains.

- If **PC1 explains 70%** and **PC2 explains 20%**, then the first two principal components capture **90% of the variance** in the dataset.

Scatter Plot of PCA-Reduced Data

A 2D scatter plot of PCA-transformed features allows us to visualize how well PCA separates different species in the Iris dataset.

Impact of PCA on Classification

- If PCA preserves most of the variance, classification algorithms (e.g., k-NN, SVM) can achieve similar performance with fewer features.
- If too much information is lost, classification accuracy may decrease.

Benefits of PCA

- **Feature Reduction:** Reduces the number of variables without significant loss of information.
- **Noise Reduction:** Removes redundant or less informative features.
- **Improved Visualization:** Enables easier interpretation of high-dimensional data.

Better Model Performance: Enhances efficiency in training machine learning models.

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.

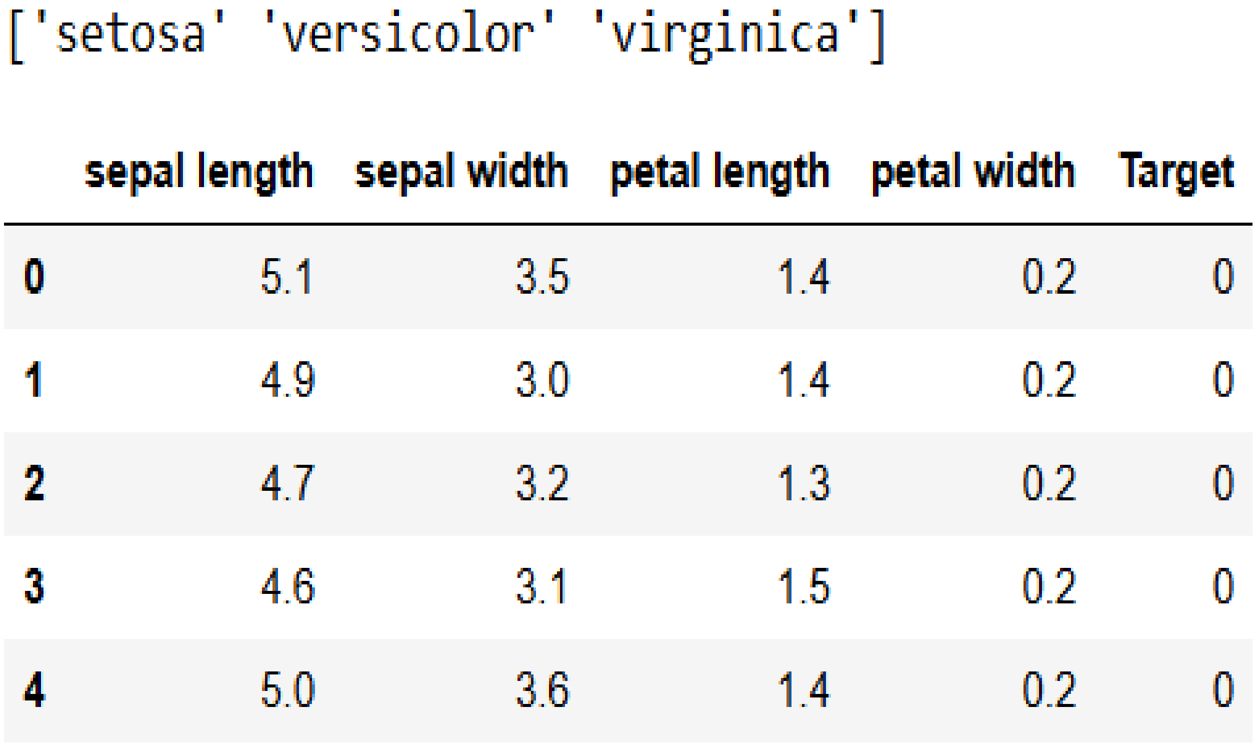
It has clearly separable classes, which makes it ideal for classification tasks.
It is preloaded in Scikit-learn, making it accessible for learning and experimentation.

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.

Program

```
#import necessary libraries:
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt

#load the iris dataset:
iris = load_iris()
features = iris.data
target = iris.target
print(iris.target_names)
iris_df = pd.DataFrame(data=features,columns=['sepal length','sepal width','petal length','petal
width'])
iris_df['Target']=target
```

iris_df.head()



```
['setosa' 'versicolor' 'virginica']
```

	sepal length	sepal width	petal length	petal width	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

#Standardize the features

scaler=StandardScaler()

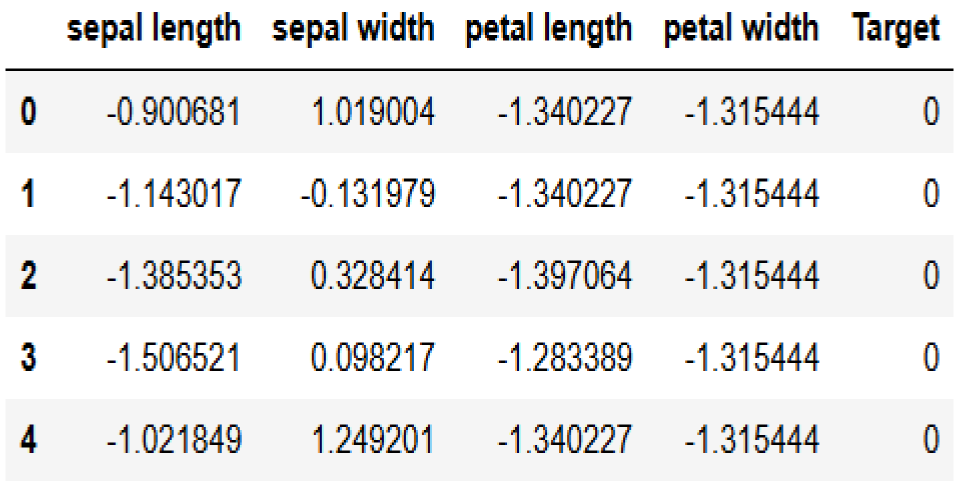
features_standardized=scaler.fit_transform(features)

features_standardized=StandardScaler().fit_transform(features)

iris_std_df=pd.DataFrame(data=features_standardized,columns=['sepal length','sepal width','petal length','petal width'])

iris_std_df['Target']=target

iris_std_df.head()



	sepal length	sepal width	petal length	petal width	Target
0	-0.900681	1.019004	-1.340227	-1.315444	0
1	-1.143017	-0.131979	-1.340227	-1.315444	0
2	-1.385353	0.328414	-1.397064	-1.315444	0
3	-1.506521	0.098217	-1.283389	-1.315444	0
4	-1.021849	1.249201	-1.340227	-1.315444	0

#create a dataframe for the reduced data

pca=PCA(n_components=2)

features_pca=pca.fit_transform(features_standardized)

pca_df=pd.DataFrame(data=features_pca,columns=["Principal Component1","Principal Component2"])

pca_df["Target"]=target

print(pca_df)

```
     Principal Component1  Principal Component2  Target
0               -2.264703              0.480027       0
1               -2.080961             -0.674134       0
2               -2.364229             -0.341908       0
3               -2.299384             -0.597395       0
4               -2.389842              0.646835       0
..                    ...                   ...     ...
145              1.870503              0.386966       2
146              1.564580             -0.896687       2
147              1.521170              0.269069       2
148              1.372788              1.011254       2
149              0.960656             -0.024332       2

[150 rows x 3 columns]
```

#Visualize the results plt.scatter (x,y,label,,,)

scaler=StandardScaler()

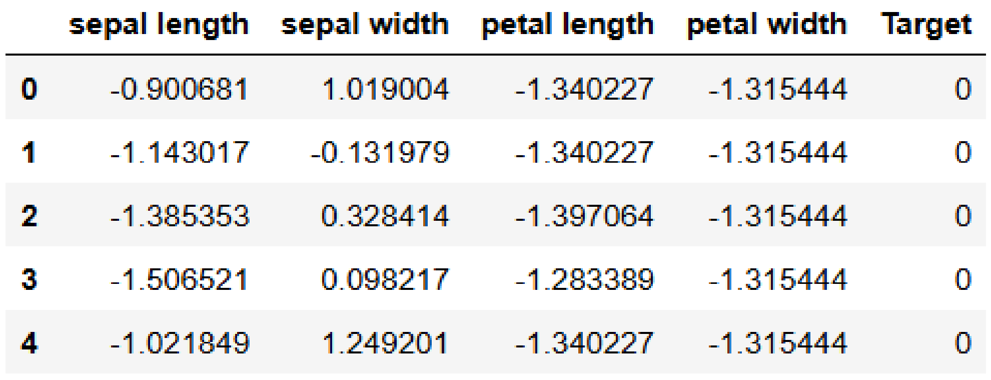
features_standardized=scaler.fit_transform(features)

features_standardized=StandardScaler().fit_transform(features)

iris_std_df=pd.DataFrame(data=features_standardized,columns=['sepal length','sepal width','petal length','petal width'])

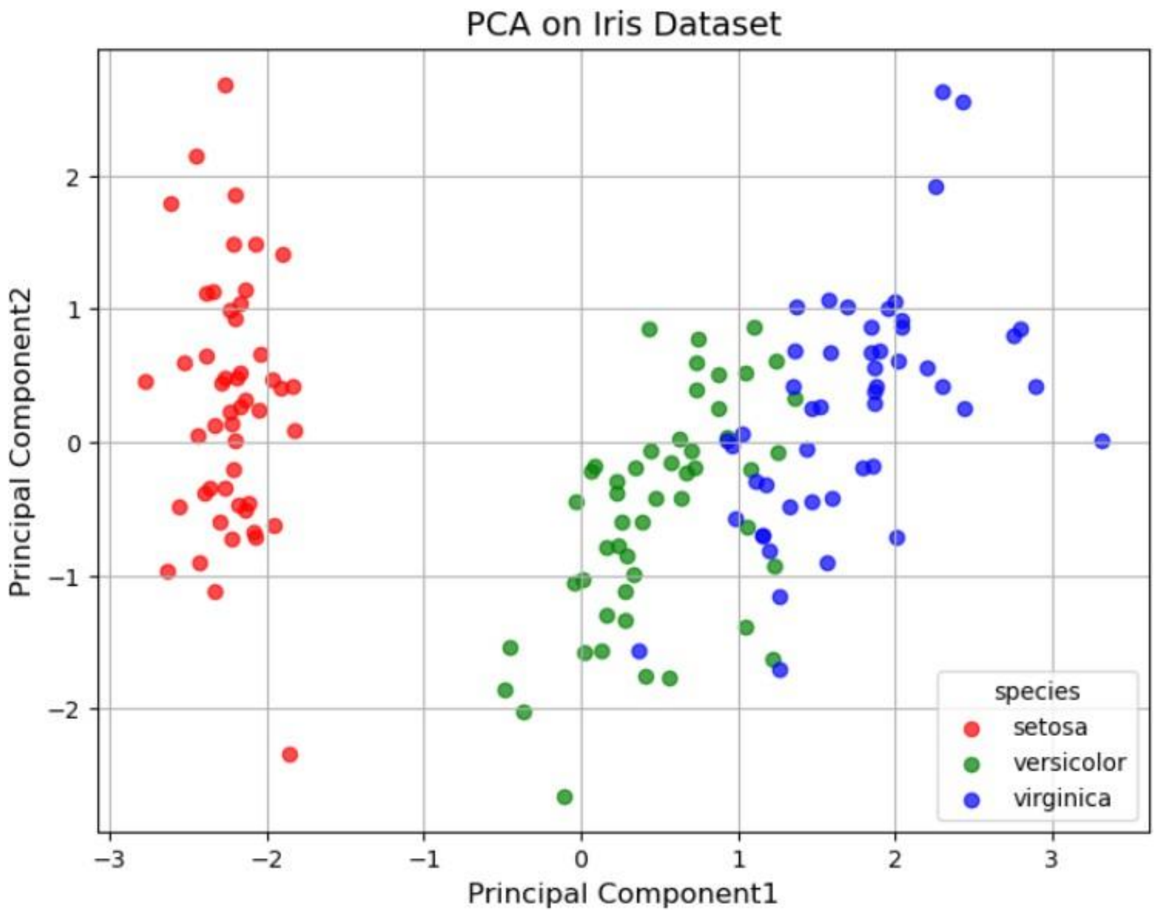
iris_std_df['Target']=target

iris_std_df.head()

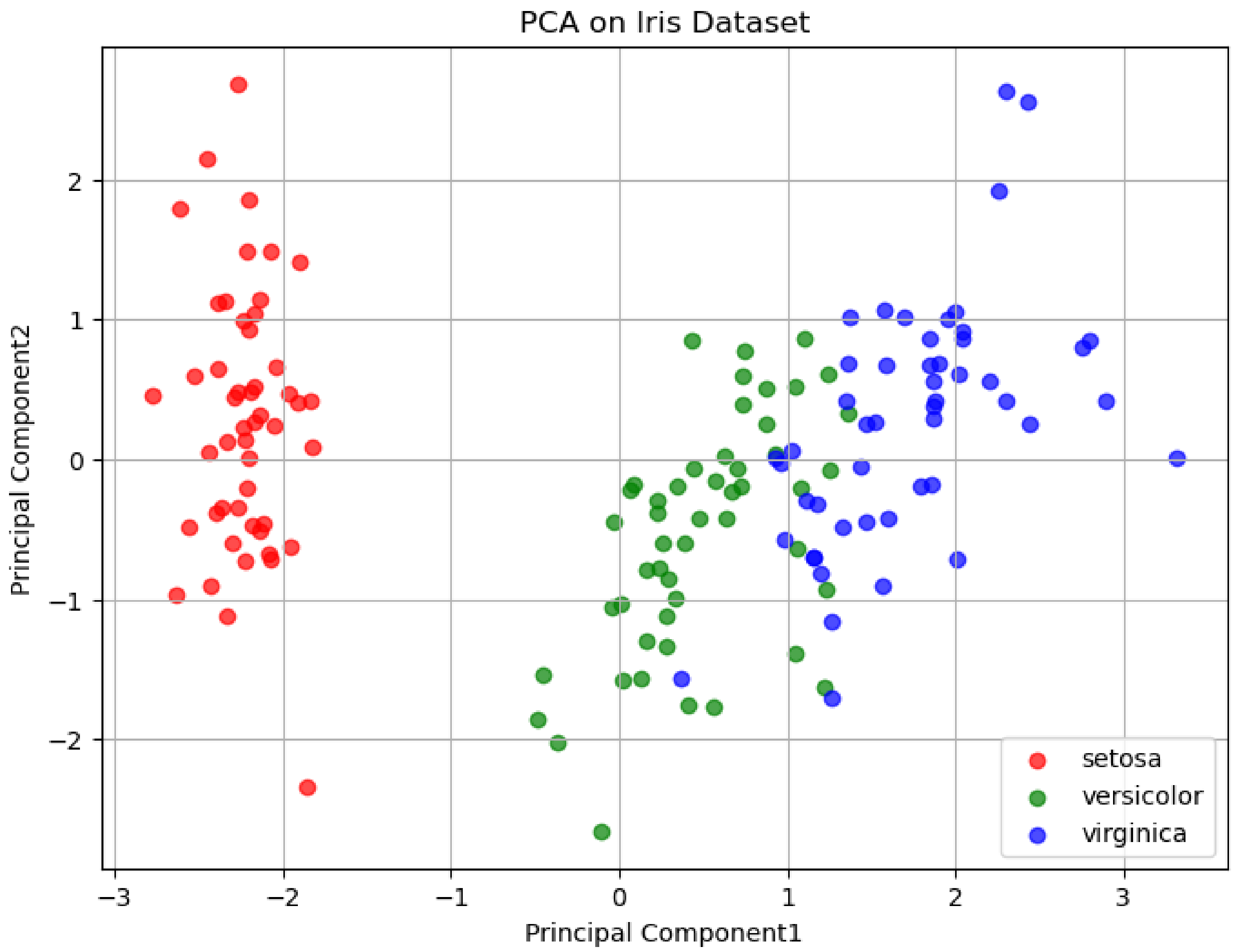


	sepal length	sepal width	petal length	petal width	Target
0	-0.900681	1.019004	-1.340227	-1.315444	0
1	-1.143017	-0.131979	-1.340227	-1.315444	0
2	-1.385353	0.328414	-1.397064	-1.315444	0
3	-1.506521	0.098217	-1.283389	-1.315444	0
4	-1.021849	1.249201	-1.340227	-1.315444	0

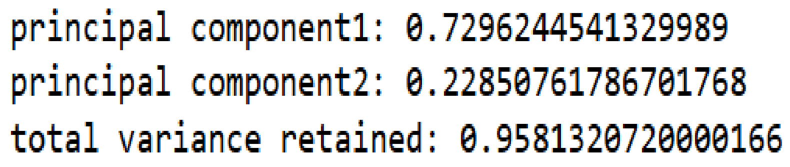
```
plt.figure(figsize=(8,6))
for label,color in zip(iris.target_names,["red","green","blue"]):
    plt.scatter(
    pca_df.loc[pca_df["Target"]==list(iris.target_names).index(label),"Principal Component1"],
    pca_df.loc[pca_df["Target"]==list(iris.target_names).index(label),"Principal Component2"],
    label=label,color=color,alpha=0.7
    )
plt.xlabel("Principal Component1",fontsize=12)
plt.ylabel("Principal Component2",fontsize=12)
plt.title('PCA on Iris Dataset',fontsize=14)
plt.legend(title="species")
plt.grid()
plt.show()
```



```
colors=['red','green','blue']

labels=iris.target_names

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

for i in range (len(colors)):

plt.scatter(features_pca[target==i,0],features_pca[target==i,1],color=colors[i],label=labels[i],alph
a=0.7)

plt.xlabel("Principal Component1")

plt.ylabel("Principal Component2")

plt.title('PCA on Iris Dataset')

plt.legend()

plt.grid()

plt.show()
```



```
#Explaines Variance by each principal component
explained_variance = pca.explained_variance_ratio_
print("principal component1:", explained_variance[0])
print("principal component2:", explained_variance[1])
print("total variance retained:",sum(explained_variance))
```



```
principal component1: 0.7296244541329989
principal component2: 0.22850761786701768
total variance retained: 0.9581320720000166
```

Experiment 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.

Introduction to the Find-S Algorithm

What is the Find-S Algorithm?

The **Find-S algorithm** is a **supervised learning algorithm** used in **concept learning** to find the most specific hypothesis that is consistent with a given set of positive training examples. It is one of the simplest algorithms for learning from examples in a hypothesis space.

Importance of Find-S Algorithm

- Helps in understanding how hypotheses are learned from training data.
- Provides a structured way to **generalize from specific instances**.
- Forms the foundation for more advanced **machine learning algorithms**.

Working of the Find-S Algorithm

The Find-S algorithm follows these steps:

Initialize the Hypothesis: Start with the most specific hypothesis (i.e., all attributes set to the most restrictive value).

Iterate Through Each Training Example:

- If the example is **positive** (output = "Yes"), update the hypothesis:
- Replace any attribute value in the hypothesis that is **not consistent** with the example with a more general value (?).

If the example is **negative** (output = "No"), ignore it.

1. Final Hypothesis:

- After processing all positive examples, the **final hypothesis** represents the most specific generalization of the training data.

Understanding the Output Hypothesis

1. Initial Hypothesis

• The algorithm starts with the most specific hypothesis:

h = ("Ø", "Ø", "Ø", "Ø") (empty hypothesis).

1. *Iterative Learning Process*

- It generalizes step by step based on the positive training examples.
- Attributes that differ among positive examples are replaced with ? (wildcard).

Final Hypothesis

- The final hypothesis is the most specific generalization covering all positive examples.
- It represents a **logical rule** derived from the dataset.

The algorithm will generate the most specific hypothesis that covers all positive instances.

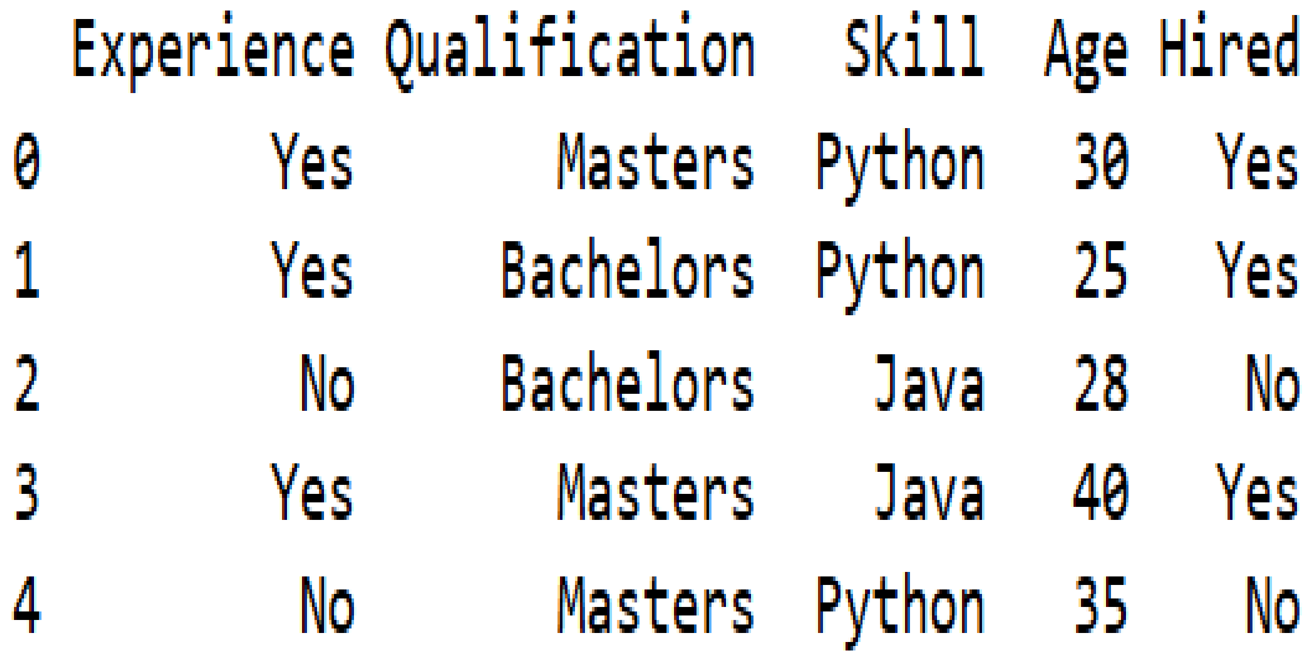
Limitations of Find-S

- **Only considers positive examples**: It ignores negative examples, which may lead to an incomplete hypothesis.
- **Cannot handle noise or missing data**: Works only when training data is perfect.

Finds only one hypothesis:

- Does not provide alternative consistent hypotheses.
- 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.

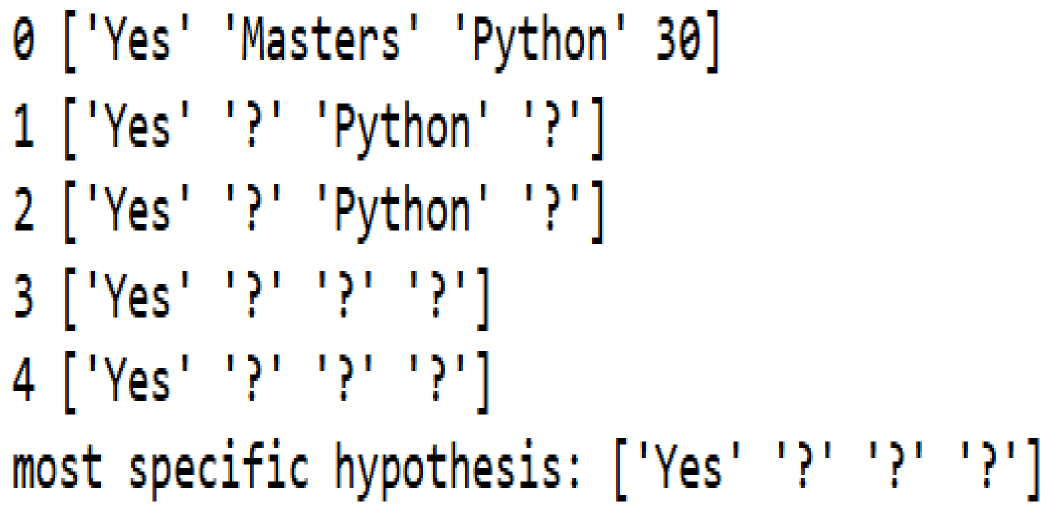
```
#import necessary libraries
import pandas as pd
#Read the csv file
data = pd.read_csv(r'training_data.csv')
print(data)
```



```
  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
```

```
#write/create user defined function for find S algorithm
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
    target=data.iloc[:,-1].values
#Initialize hypothesis with first positive example
    for i in range (len(target)):
        if target[i]=="Yes":
            hypothesis=attributes[i].copy()
            break
```

```
#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]='?'
        print(i,hypothesis)
    return hypothesis
#Run/Call Find-S-Algorithm
final_hypothesis = find_s_algorithm(data)
#Print the learned hypothesis
print("most specific hypothesis:",final_hypothesis)
```



```
0 ['Yes' 'Masters' 'Python' 30]
1 ['Yes' '?' 'Python' '?']
2 ['Yes' '?' 'Python' '?']
3 ['Yes' '?' '?' '?']
4 ['Yes' '?' '?' '?']
most specific hypothesis: ['Yes' '?' '?' '?']
```

Experiment 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.

1. **Label the first 50 points {x1,……,x50} as follows: if (xi ≤ 0.5), then xi ε Class1, else xi ε Class2**
2. **Classify the remaining points, x51,……,x100 using KNN. Perform this for k=1,2,3,4,5,20,30**

Introduction to k-Nearest Neighbors (k-NN)

What is k-NN?

- The **k-Nearest Neighbors (k-NN) algorithm** is a **supervised learning algorithm** used for both classification and regression. It classifies a data point based on the majority class among its nearest neighbors.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

Importance of k-NN

- **Simple and effective** for classification tasks.
- **Non-parametric** (makes no assumptions about the data distribution).
- **Handles multi-class classification** with ease.

How k-NN Works?

The k-NN algorithm follows these steps:

1. **Choose the value of k** (number of nearest neighbors).
2. **Compute the distance** between the test sample and all training samples using a distance metric (e.g., Euclidean distance).
3. **Select the k nearest neighbors** (data points with the smallest distance to the test sample).
4. **Assign the majority class** among the k neighbors to the test sample.

Working of the k-NN Algorithm Choose a Value for k:

- A **small k (e.g., k=1)** makes the model sensitive to noise and results in **high variance**.
- A **large k (e.g., k=30)** smooths the decision boundary but may lead to **high bias**.
- The optimal k is usually found by **cross-validation**.

Compute Distance Between Data Points: The algorithm relies on a distance metric to determine similarity between data points. Common distance measures include:

`- **Euclidean Distance** (Most commonly used)

Manhattan Distance

Minkowski Distance

- `Cosine Similarity** (Used in text-based applications)

- The most common method is **Euclidean Distance**:

$d = \sqrt{[(x2 - x1)2 + (y2 - y1)2]}$

3. Decision Rule for Classification

- **Majority Voting:** The most common class among the k neighbors determines the predicted class.

Weighted Voting: Closer neighbors have higher influence on the prediction than farther neighbors.

Dataset Generation and Classification Task

Step 1: Generate 100 Random Points in the Range [0,1]

- The dataset consists of 100 random values of x uniformly distributed between **0 and 1**.

Step 2: Assign Labels to the First 50 Points

- The first **50 points ($x_1, x_2, ..., x_{50}$)** are labeled as:

- **Class 1** if x_i <= 0.5
- **Class 2** if x_i > 0.5

Step 3: Classify Remaining Points (x_{51}, ..., x_{100}) using k-NN

- The k-NN algorithm is used to classify the next **50 points** based on the first **50 labeled points**.

Step 4: Experiment with Different k Values

- Classification is performed for multiple values of k:
- k = 1, 2, 3, 4, 5, 20, 30
- Observing how different values of kaffect classification accuracy and decision boundaries.

Bias-Variance Tradeoff in k-NN

- **Smaller k values (e.g., k=1)** → Low bias, high variance (more flexible but prone to noise).
- **Larger k values (e.g., k=20, 30)** → High bias, low variance (less flexible but smoother decision boundary).

Advantages of k-NN

- Simple and easy to implement.
- **No training phase**—all computation happens during prediction.
- Works well for multi-class classification problems.
- **Can model complex decision boundaries** when k is appropriately chosen.

Limitations of k-NN

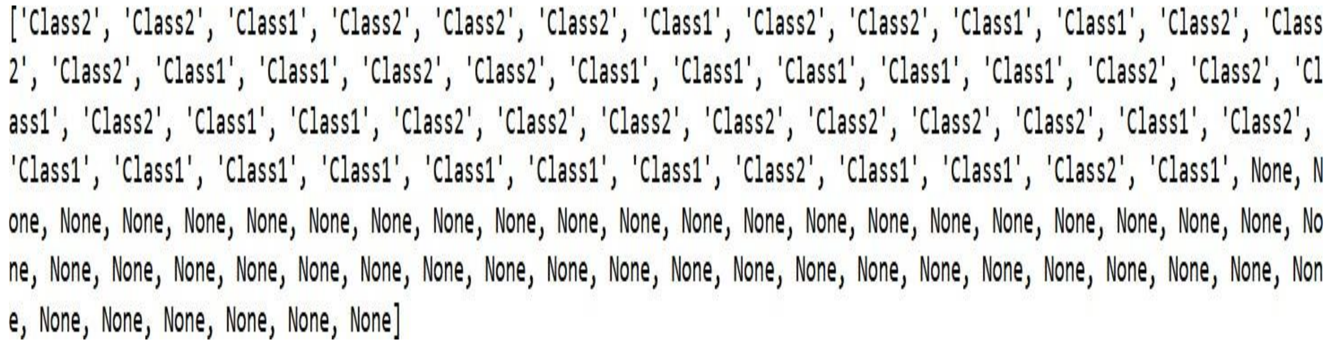
- **Computationally expensive** for large datasets.
- Performance depends on the choice of k.
- **Sensitive to irrelevant or redundant features**.
- **Memory-intensive** since all training data needs to be stored.

#import Necessary Libraries

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

Generate dataset

```
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)
```



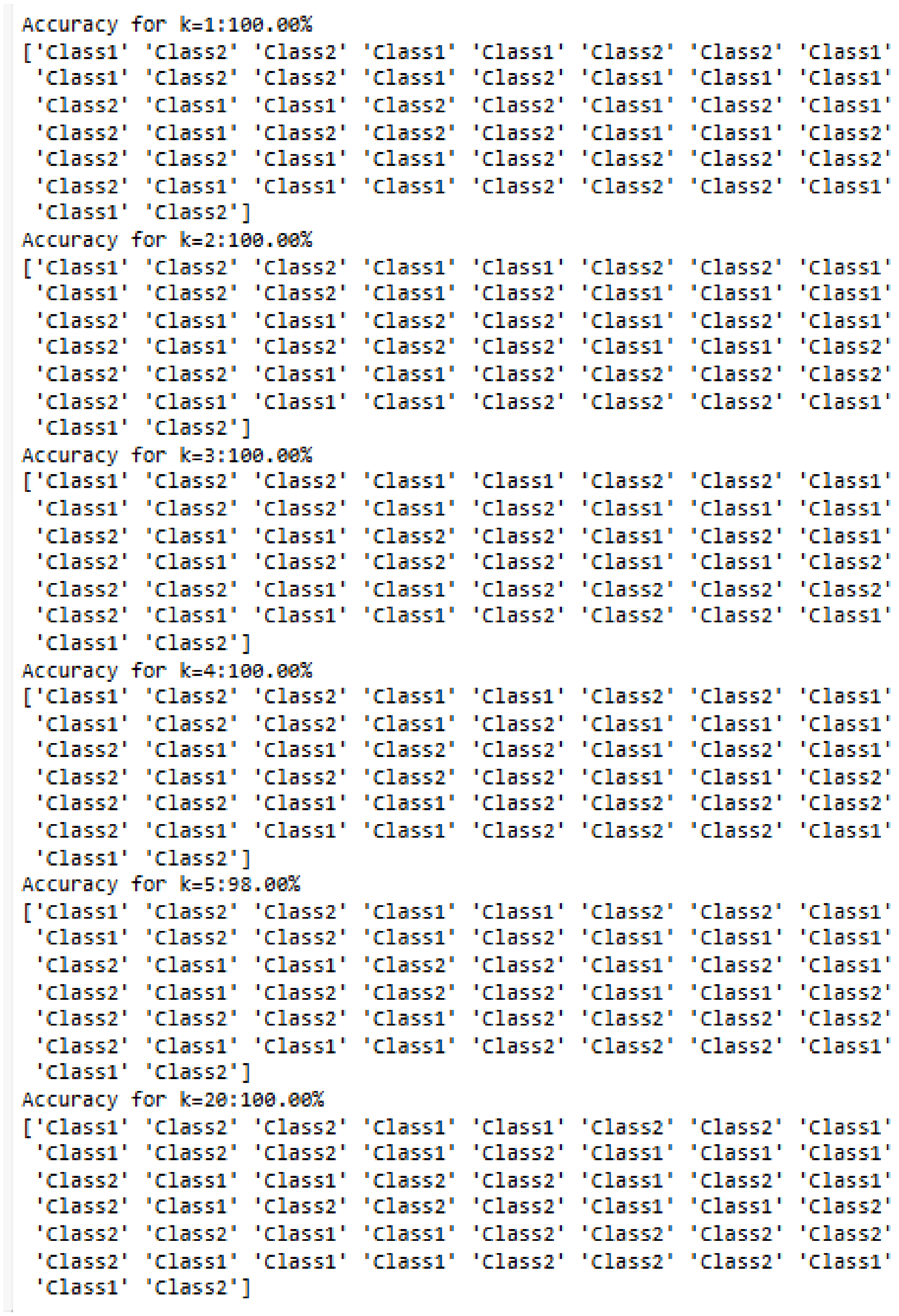
```
['Class2', 'Class2', 'Class1', 'Class2', 'Class2', 'Class2', 'Class1', 'Class2', 'Class2', 'Class1', 'Class1', 'Class2', 'Class
2', 'Class2', 'Class1', 'Class1', 'Class2', 'Class2', 'Class1', 'Class1', 'Class1', 'Class1', 'Class1', 'Class2', 'Class2', 'Cl
ass1', 'Class2', 'Class1', 'Class1', 'Class2', 'Class2', 'Class2', 'Class2', 'Class2', 'Class2', 'Class2', 'Class1', 'Class2',
'Class1', 'Class1', 'Class1', 'Class1', 'Class1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2', 'Class1', None, N
one, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e, 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)
df=pd.DataFrame(data)
df.head()
```

```
{'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([0.72095621, 0.76615021, 0.06775014, 0.5
1103614, 0.6869719 ,
       0.55393421, 0.46737394, 0.72969114, 0.68647285, 0.16084033,
       0.08046086, 0.57909295, 0.70095687, 0.88929224, 0.24161365,
       0.04981832, 0.66752434, 0.65973086, 0.13324933, 0.36112965,
       0.41325153, 0.0716705 , 0.20320352, 0.8544208 , 0.51807002,
       0.17992753, 0.69345369, 0.30810718, 0.08342543, 0.67243429,
       0.50718056, 0.66407811, 0.87049266, 0.87840566, 0.85327485,
       0.92391467, 0.45196583, 0.74977731, 0.22934881, 0.08863437,
       0.21351729, 0.39945073, 0.21389974, 0.32312058, 0.4938461 ,
       0.92438473, 0.36924282, 0.31012793, 0.73257046, 0.06952616,
       0.08232899, 0.71605679, 0.72409503, 0.45003625, 0.42243751,
       0.82754646, 0.54356809, 0.3118151 , 0.04734741, 0.63677823,
       0.58242715, 0.35379241, 0.50949749, 0.43020334, 0.28334047,
       0.2556375 , 0.83250701, 0.27517161, 0.45476971, 0.73040117,
       0.96502963, 0.47991932, 0.76448356, 0.20762745, 0.9830655 ,
       0.38486831, 0.78643136, 0.71738101, 0.86987852, 0.03176016,
       0.22518964, 0.97803035, 0.65801176, 0.66928888, 0.48700725,
       0.29401596, 0.93905041, 0.57636956, 0.50551851, 0.63391406,
       0.82254585, 0.20896943, 0.45948065, 0.38469067, 0.85974426,
       0.72865672, 0.75936767, 0.35329706, 0.27421068, 0.84959246]), 'Label': ['Class2', 'Class2', 'Class1', 'Class2', 'Class
2', 'Class2', 'Class1', 'Class2', 'Class2', 'Class1', 'Class1', 'Class2', 'Class2', 'Class2', 'Class1', 'Class1', 'Class2', 'Cl
ass2', 'Class1', 'Class1', 'Class1', 'Class1', 'Class1', 'Class2', 'Class2', 'Class1', 'Class2', 'Class1', 'Class1', 'Class2',
'Class2', 'Class2', 'Class2', 'Class2', 'Class2', 'Class2', 'Class1', 'Class2', 'Class1', 'Class1', 'Class1', 'Class1', 'Class
1', 'Class1', 'Class1', 'Class2', 'Class1', 'Class1', '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, N
one, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]}
```

	Point	Value	Label
0	x1	0.720956	Class2
1	x2	0.766150	Class2
2	x3	0.067750	Class1
3	x4	0.511036	Class2
4	x5	0.686972	Class2

```
#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 lsbels for testing (for accuracy calculation)
true_labels=["Class1" if x<=0.5 else "Class2" for x in values[50:]]
#Perform KNN classification for different values of K
k_values=[1,2,3,4,5,20]
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

#accuracy calculation
  accuracy=accuracy_score(true_labels,predictions)*100
  accuracies[k]=accuracy
    print(f"Accuracy for k={k}:{accuracy:.2f}%")
    print(predictions)
```



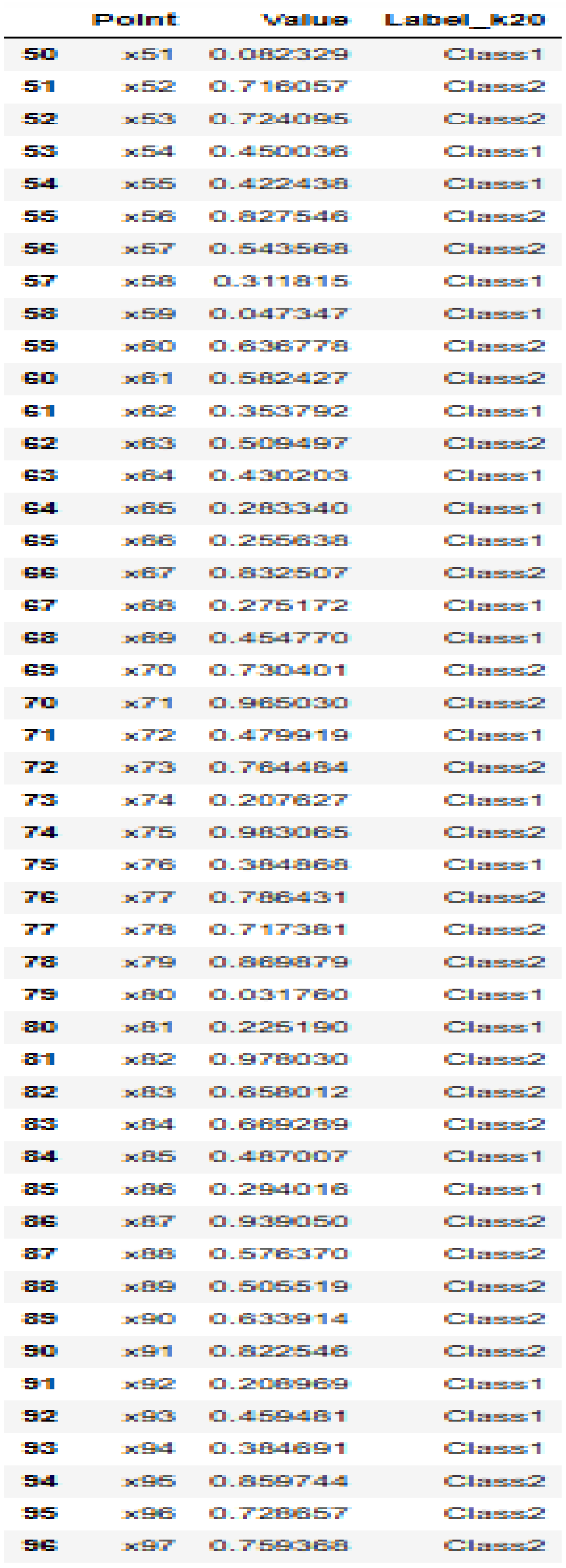
```
Accuracy for k=1:100.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
Accuracy for k=2:100.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
Accuracy for k=3:100.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
Accuracy for k=4:100.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
Accuracy for k=5:98.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
Accuracy for k=20:100.00%
['Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1'
 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class2' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2'
 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2']
```

#assign predictions back to the dataframe for this k

unlabeled_df[f"Label_k{k}"]=predictions

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

df1



	Point	Value	Label_k20
50	x51	0.082329	Class1
51	x52	0.716057	Class2
52	x53	0.724095	Class2
53	x54	0.450036	Class1
54	x55	0.422438	Class1
55	x56	0.827546	Class2
56	x57	0.543568	Class2
57	x58	0.311815	Class1
58	x59	0.047347	Class1
59	x60	0.636778	Class2
60	x61	0.582427	Class2
61	x62	0.353792	Class1
62	x63	0.509497	Class2
63	x64	0.430203	Class1
64	x65	0.283340	Class1
65	x66	0.255638	Class1
66	x67	0.832507	Class2
67	x68	0.275172	Class1
68	x69	0.454770	Class1
69	x70	0.730401	Class2
70	x71	0.965030	Class2
71	x72	0.479919	Class1
72	x73	0.764484	Class2
73	x74	0.207627	Class1
74	x75	0.983065	Class2
75	x76	0.384868	Class1
76	x77	0.786431	Class2
77	x78	0.717381	Class2
78	x79	0.869879	Class2
79	x80	0.031760	Class1
80	x81	0.225190	Class1
81	x82	0.978030	Class2
82	x83	0.658012	Class2
83	x84	0.669289	Class2
84	x85	0.487007	Class1
85	x86	0.294016	Class1
86	x87	0.939050	Class2
87	x88	0.576370	Class2
88	x89	0.505519	Class2
89	x90	0.633914	Class2
90	x91	0.822546	Class2
91	x92	0.208969	Class1
92	x93	0.459481	Class1
93	x94	0.384691	Class1
94	x95	0.859744	Class2
95	x96	0.728657	Class2
96	x97	0.759368	Class2

Experiment 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

Introduction to Locally Weighted Regression (LWR)

What is Locally Weighted Regression?

Locally Weighted Regression (LWR) is a **non-parametric** machine learning algorithm that fits a regression model to a local subset of data points. Unlike traditional regression techniques, LWR does not assume a fixed set of parameters for the entire dataset but instead assigns different weights to data points based on their distance from the target point.

Importance of Locally Weighted Regression

- Handles **non-linearity** effectively.
- Provides **better flexibility** compared to global regression models.
- More **robust to outliers** due to localized weighting.
- Suitable for datasets where relationships between variables **vary locally**.

How Locally Weighted Regression Works

1. Define the Weighting Function

- A kernel function (e.g., **Gaussian kernel**) is used to assign weights to data points:

$$w_i = e^{-\frac{(x-x_i)^2}{2\tau^2}}$$

- Here, τ (tau) is the **bandwidth parameter** that controls the locality of weighting.

Compute Localized Weights

For a given query point x, assign weights to training points based on proximity.

Fit a Local Model

- Solve a **weighted least squares** problem using the locally weighted dataset.

1. Make Predictions: Compute the predicted value at x using the locally trained model.

Dataset Selection:

For this experiment, we need a dataset with a **clear non-linear relationship** between independent and dependent variables. Some possible datasets include:

Synthetic Data: Randomly generated non-linear data points. •

Real-World Data:

- **Auto MPG Dataset:** Predict fuel efficiency based on engine displacement, horsepower, etc.
- **California Housing Dataset:** Predict house prices based on features like location and area.
- **Temperature vs. Time Series Data:** Forecast weather trends.

Steps for Implementing Locally Weighted Regression

1. Load the Dataset

- Choose a dataset with **one independent variable (x) and one dependent variable (y)**.

2. Apply the Locally Weighted Regression Algorithm

- Assign weights to each data point using a Gaussian kernel.
- Solve the weighted linear regression equation.

3. Experiment with Different Bandwidth Parameters (τ)

- **Small τ:** Model focuses on very close neighbors → **More variance, less bias** (risk of overfitting).
- **Large τ:** Model considers a broader range of points → **More bias, less variance** (risk of underfitting).

Visualize the Results:

- **Scatter Plot of Data Points** to observe the actual distribution.
- **Fitted Curve from LWR** with different values of τ to compare model performance.

Advantages of Locally Weighted Regression

✓ **Captures complex relationships** between input and output variables.

✓ **Works well with small datasets** where global linear regression may not be suitable.

✓ **Does not assume a fixed functional form**, making it highly flexible.

Limitations of Locally Weighted Regression

+ **Computationally expensive**: Must compute a separate model for each query point.

+ **Sensitive to bandwidth parameter** ((\tau)): Choosing the wrong value can lead to overfitting or underfitting.

+ **Not suitable for large datasets**: As the dataset size increases, the algorithm becomes impractical due to high computation time.

The tau (τ) parameter in your code is the bandwidth for the Gaussian kernel, which controls how much influence nearby points have in the Locally Weighted Regression (LWR). Here's what it does:

Determines the Weight Decay:

If τ is small, only very nearby points contribute significantly, making LWR behave like a very local model (more sensitive to noise). If τ is large, more distant points contribute significantly,

making LWR behave more like global linear regression.

Controls the Model Complexity:

A small τ → Highly flexible model, more prone to overfitting.

A large τ → More smoothing, leading to a simpler model (can underfit if too large).

Example Effect of Tau

τ = 0.1 → LWR behaves almost like a nearest-neighbor model (highly local, very wiggly curve).

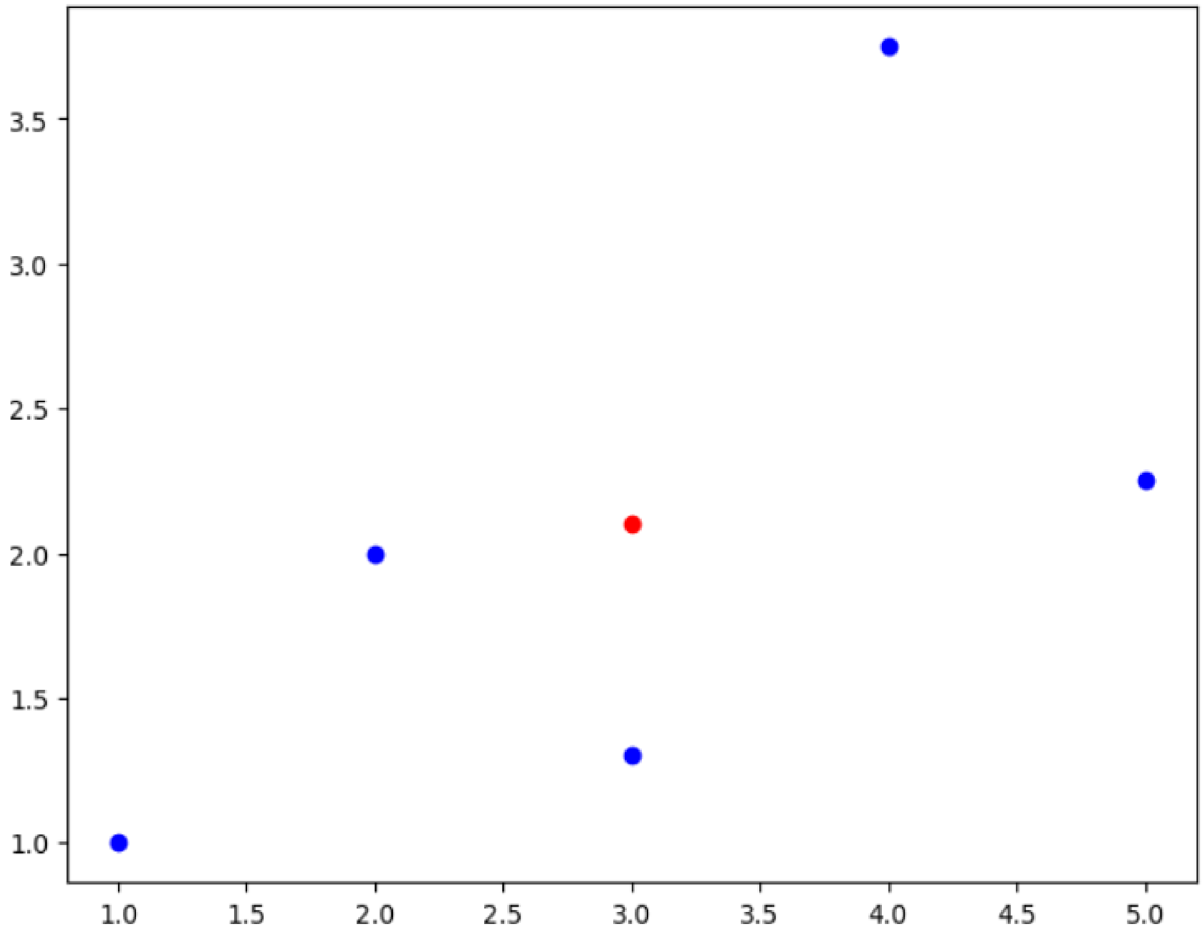
τ = 1.0 → Moderate smoothing, a good balance between flexibility and generalization.

τ = 10 → LWR behaves like ordinary least squares regression (all points are weighted almost equally).

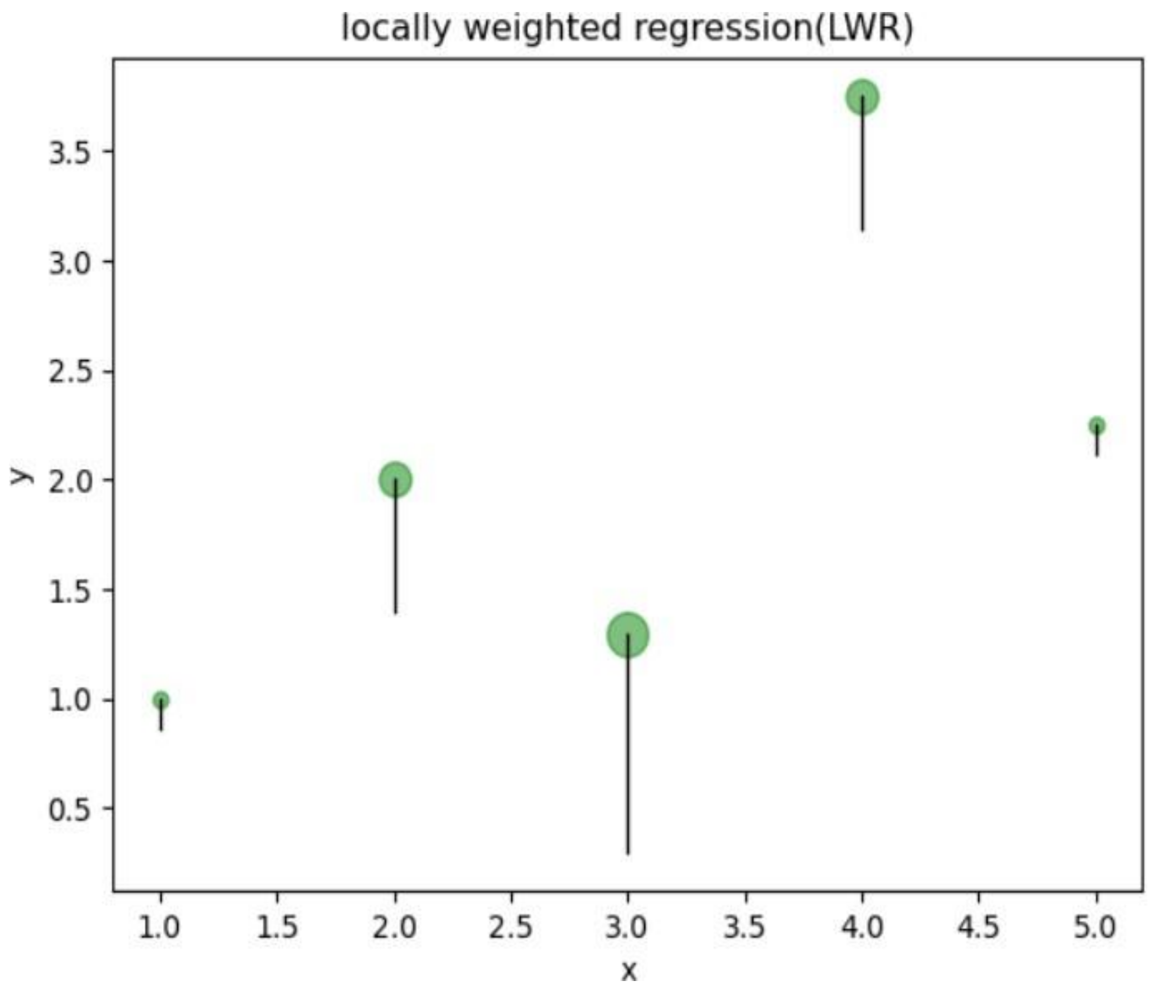
Program

```
import numpy as np
import matplotlib.pyplot as plt
def  gk(x,xq,tau):
    return np.exp(-(x-xq)**2/(2*tau**2))
def lwr(x,y,xq,tau):
    xb = np.c_[np.ones(len(x)), x]
    xqb = np.array([1,xq])
    w = np.diag(gk(x,xq,tau))
    theta = np.linalg.inv(xb.T @ w @ xb) @ xb.T @ w @ y
    return xqb @ theta
x = np.array([1,2,3,4,5])
y = np.array([1,2,1.3,3.75,2.25])
xq = 3
tau = 1.0
yp = lwr(x,y,xq,tau)

plt.figure(figsize=(8,6))
plt.scatter(x,y,color = 'blue',label = 'data points')
plt.scatter(xq,yp,color = 'red',label = f'prediction at x = {xq}')
```

```
<matplotlib.collections.PathCollection at 0x1eee0844510>
```

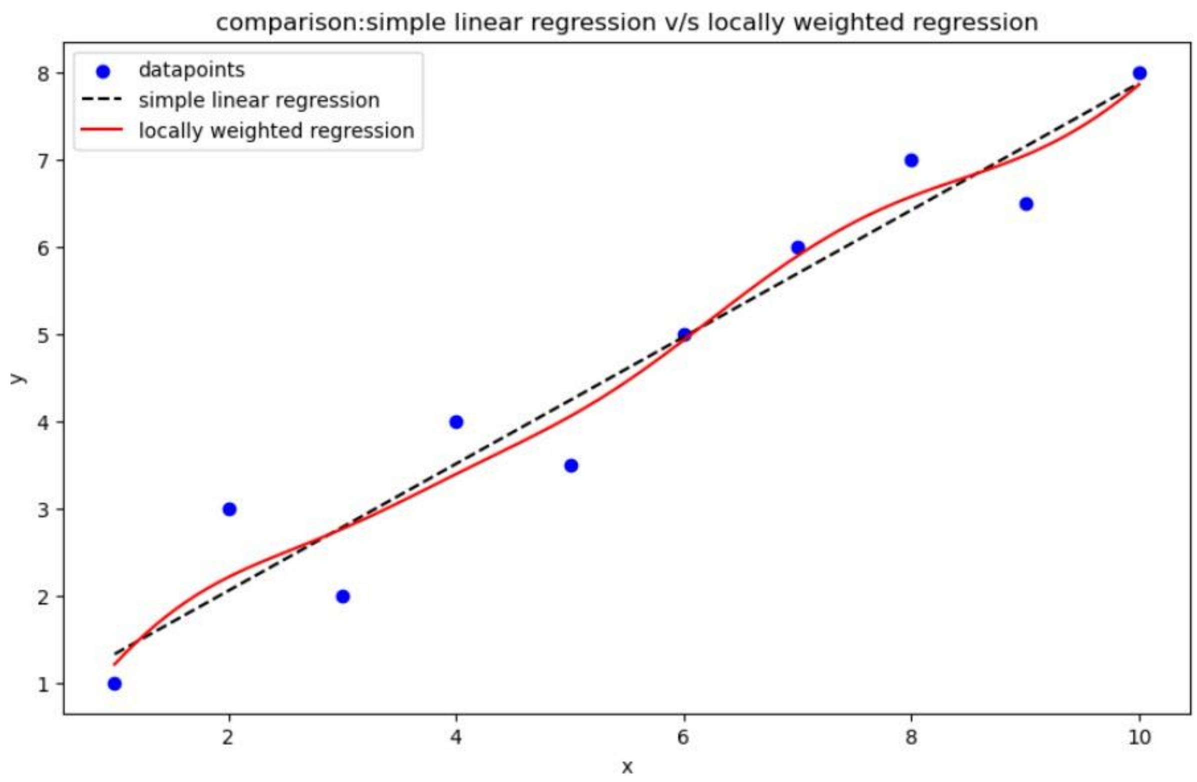


```
weights = gk(x,xq,tau)
for i in range(len(x)):
    plt.plot([x[i],x[i]],[y[i],y[i]-weights[i]],'k-',lw =1)
    plt.scatter(x[i],y[i],s=weights[i]*200,color = 'green',alpha =0.5)
plt.title("locally weighted regression(LWR)")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression


def  gk(x,xq,tau):
    return np.exp(-(x-xq)**2/(2*tau**2))
```

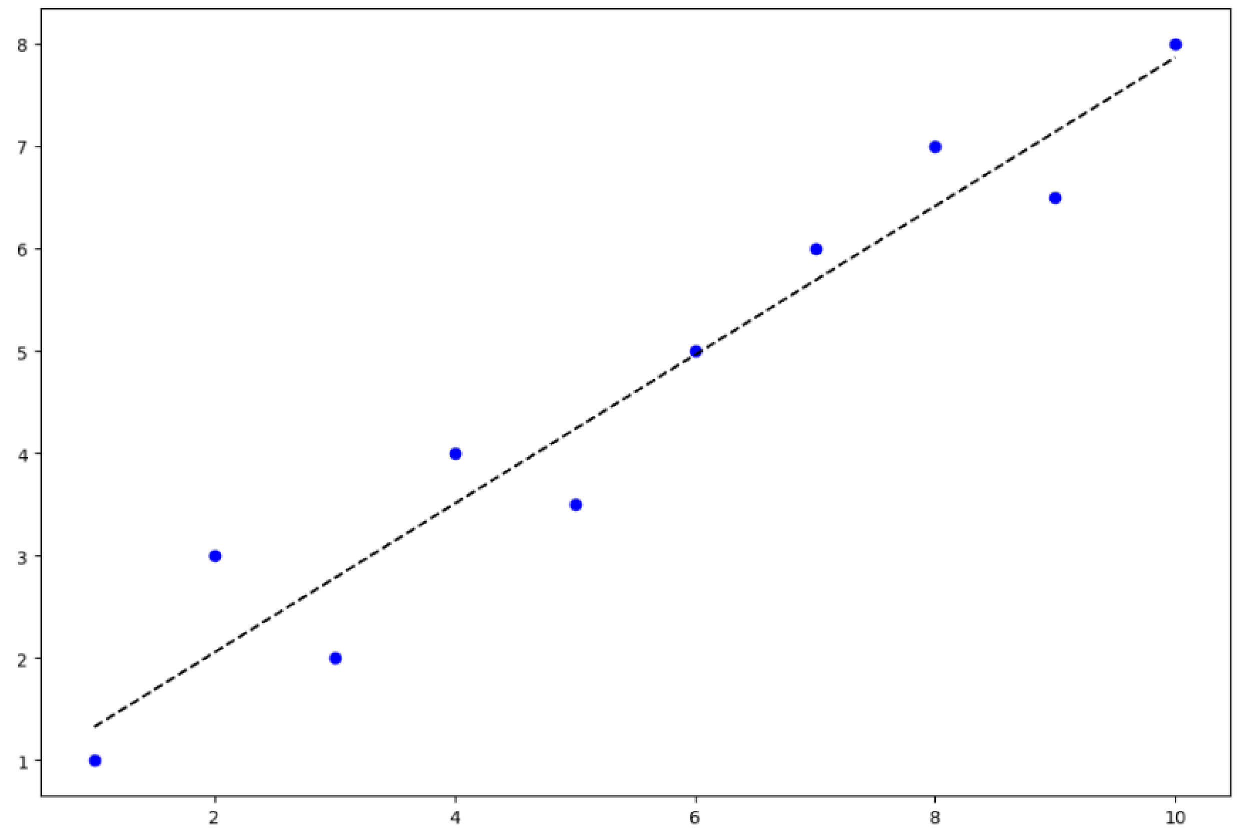
```
def lwr(x,y,xq,tau):
    xb = np.c_[np.ones(len(x)), x]
    xqb = np.array([1,xq])
    w = np.diag(gk(x,xq,tau))
    theta = np.linalg.inv(xb.T @ w @ xb) @ xb.T @ w @ y
    return xqb @ theta

x = np.array([1,2,3,4,5,6,7,8,9,10])
y = np.array([1,3,2,4,3.5,5,6,7,6.5,8])

xq = np.linspace(1,10,100)
tau = 1.0
y_lwr = np.array([lwr(x,y,xq,tau) for xq in xq])
lr = LinearRegression()
xr = x.reshape(-1,1)
lr.fit(xr,y)
yl = lr.predict(xq.reshape(-1,1))

plt.figure(figsize=(10,6))
plt.scatter(x,y,color = 'blue',label='datapoints')
plt.plot(xq,yl,color='black',linestyle='dashed',label='simple linear regression')
plt.plot(xq,y_lwr,color='red',label='locally weighted regression')
plt.title("comparison:simple linear regression v/s locally weighted regression")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



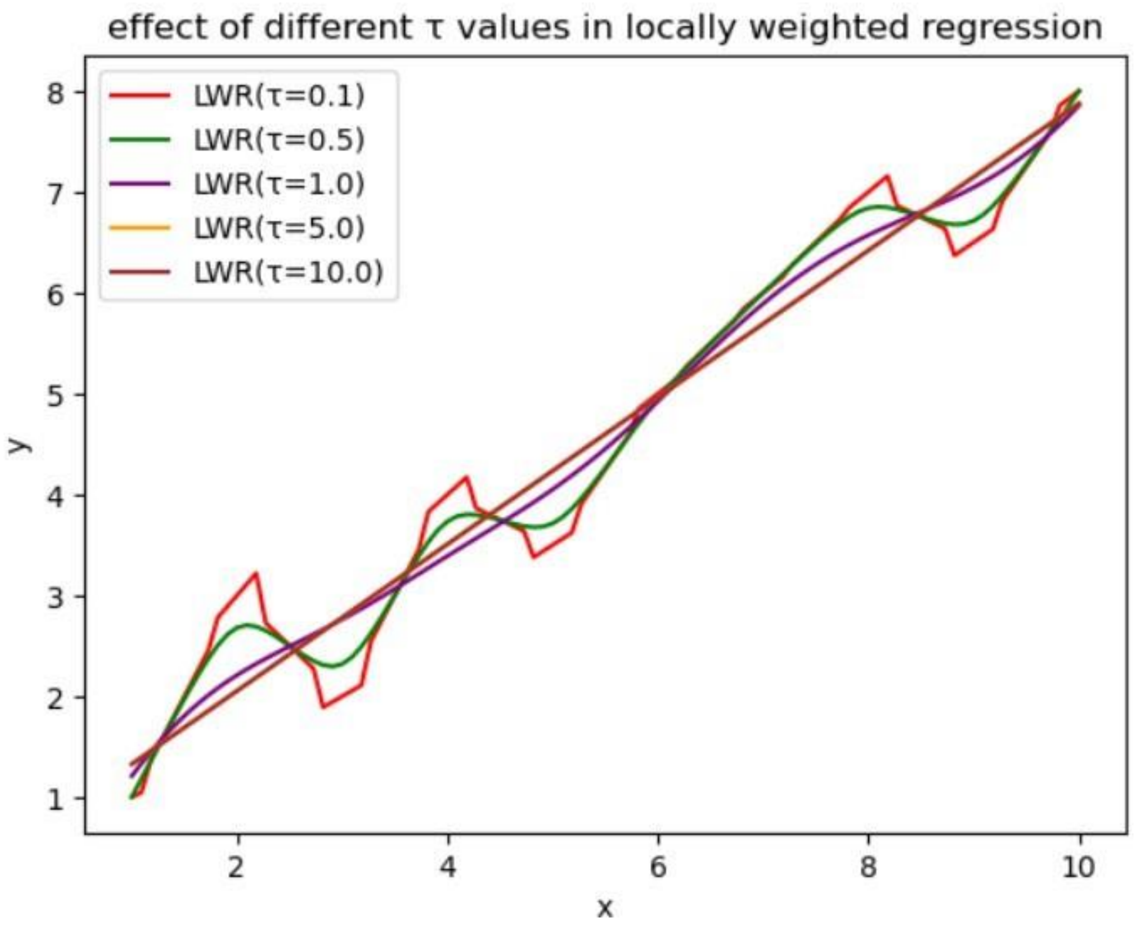
```
def  gk(x,xq,tau):
    return np.exp(-(x-xq)**2/(2*tau**2))
def lwr(x,y,xq,tau):
    xb = np.c_[np.ones(len(x)), x]
    xqb = np.array([1,xq])
    w = np.diag(gk(x,xq,tau))
    theta = np.linalg.pinv(xb.T @ w @ xb) @ xb.T @ w @ y
    return xqb @ theta


x = np.array([1,2,3,4,5,6,7,8,9,10])
y = np.array([1,3,2,4,3.5,5,6,7,6.5,8])


xq = np.linspace(1,10,100)
tau_v = [0.1,0.5,1.0,5.0,10.0]


lr = LinearRegression()
```

```
xr = x.reshape(-1,1)

lr.fit(xr,y)

yl = lr.predict(xq.reshape(-1,1))


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

plt.scatter(x,y,color = 'blue',label='datapoints')

plt.plot(xq,yl,color='black',linestyle='dashed',label='simple linear regression')
```

```
[<matplotlib.lines.Line2D at 0x1ee85537c10>]
```



```
colors = ['red','green','purple','orange','brown']

for tau,color in zip(tau_v,colors):

    y_lwr = np.array([lwr(x,y,xq,tau)for xq in xq])

    plt.plot(xq,y_lwr,color=color,label = f'LWR(τ={tau})')

plt.title("effect of different τ values in locally weighted regression")

plt.xlabel("x")

plt.ylabel("y")

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


effect of different τ values in locally weighted regression
LWR(τ=0.1)
LWR(τ=0.5)
LWR(τ=1.0)
LWR(τ=5.0)
LWR(τ=10.0)
y
x

Experiment 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.

7a. Develop a program to demonstrate the working of Linear Regression. Use Boston Housing Dataset.

Introduction to Regression Analysis What is Regression?

Regression is a fundamental statistical and machine learning technique used to model relationships between variables. It helps in predicting a dependent variable (target) based on one or more independent variables (features).

Types of Regression Models:

1. Linear Regression – Assumes a linear relationship between independent and dependent variables.
2. Polynomial Regression – Extends linear regression by introducing polynomial terms to capture non-linearity.

Terms to capture non-linearity. Linear Regression

Definition:

Linear Regression models the relationship between an independent variable (x) and a dependent variable (y) using a straight-line equation:

$y = mx + c$

m is the slope (coefficient) of the line,

c is the intercept,

x is the independent variable,

y is the dependent variable (predicted value).

Working of Linear Regression

1. Identify the best-fitting line: Uses the least squares method to minimize the error between actual and predicted values.

Compute the cost function: Measures how well the model fits the data using Mean Squared Error (MSE)

Optimize the model parameters: Uses Gradient Descent or other optimization techniques to find the best m and c .

Applications of Linear Regression

- ➢ Predicting sales revenue based on advertising spend.
- ➢ Estimating house prices based on size and location.
- ➢ Forecasting demand in supply chain management.

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,

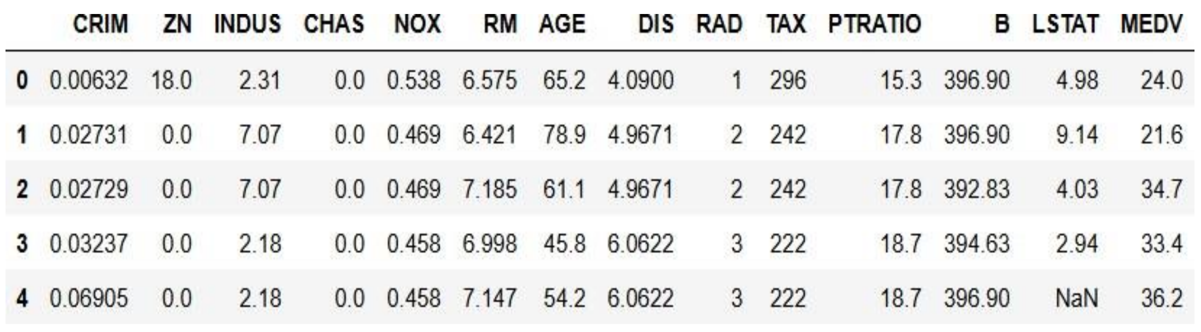
giving the total count of missing values for each column.

#import 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.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.preprocessing import StandardScaler
```

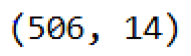
#import Boston housing dataset

```
data = pd.read_csv(r"Boston housing dataset.csv")
data.head()
```



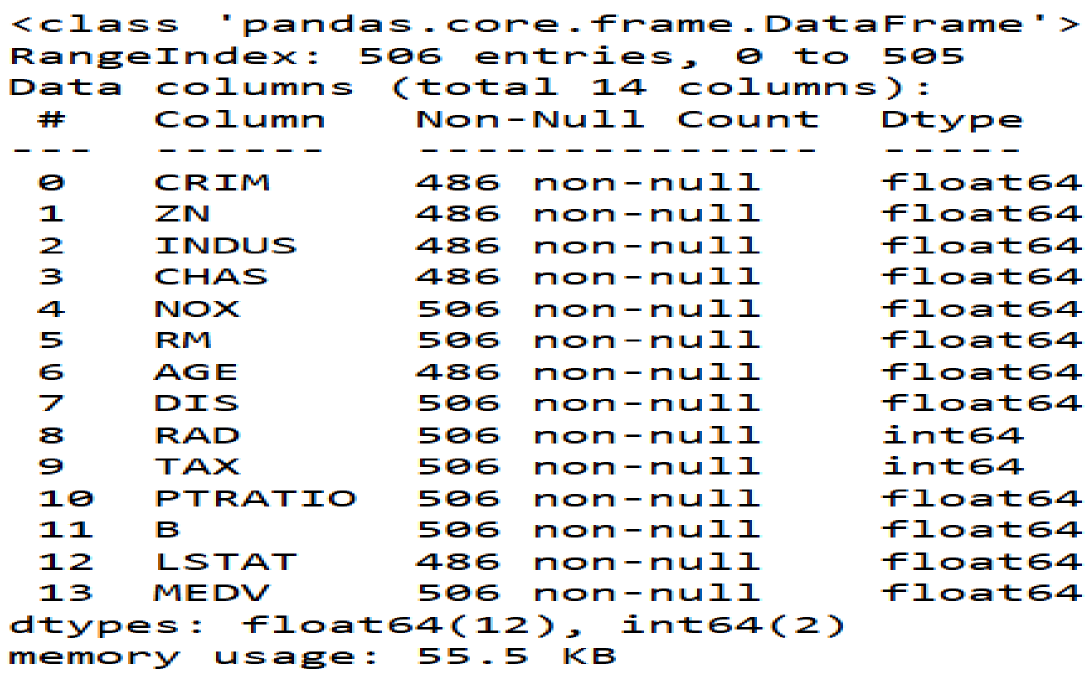
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

data.shape



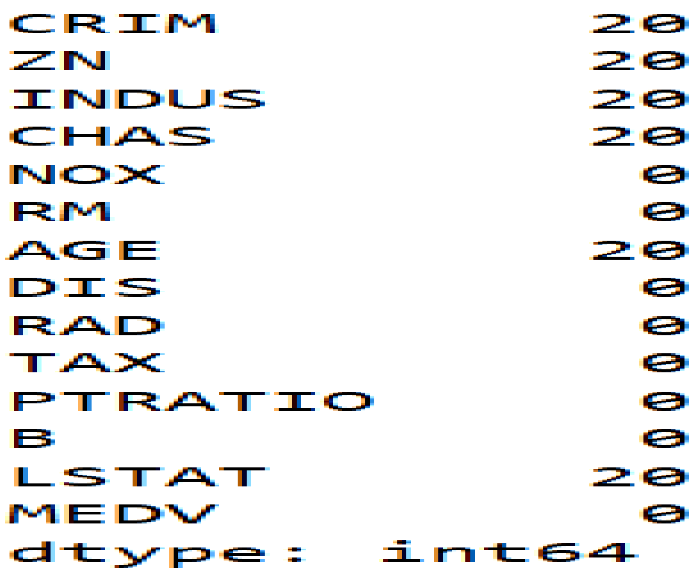
```
(506, 14)
```

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
```

data.isnull().sum()



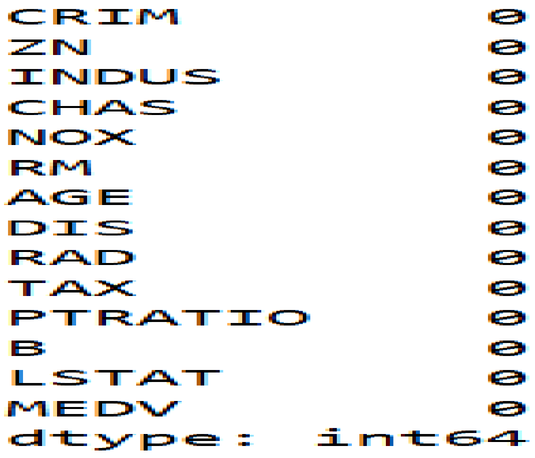
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

df=data.copy()

df['CRIM'].fillna(df['CRIM'].mean(),inplace=True)

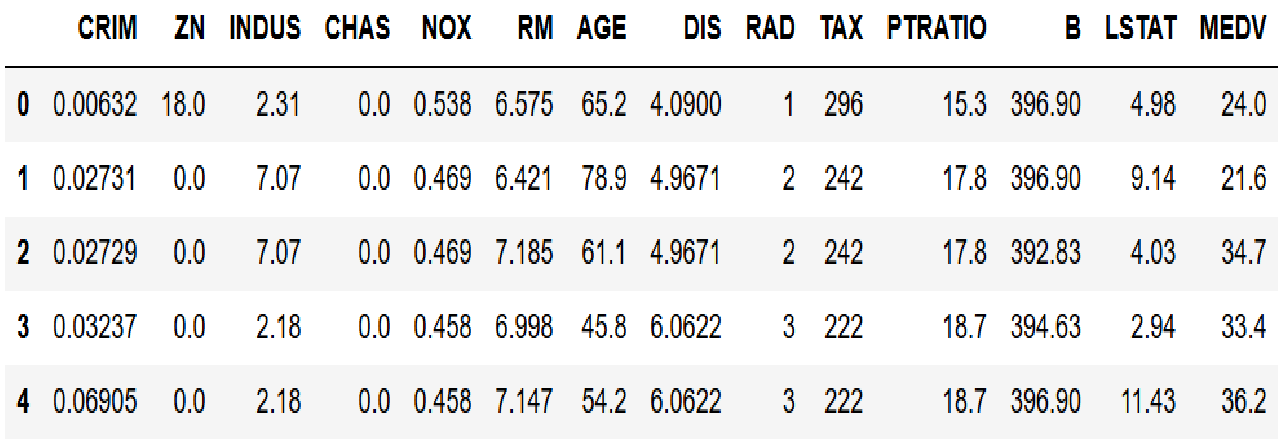
df['ZN'].fillna(df['ZN'].mean(),inplace=True)

```
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)

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
```

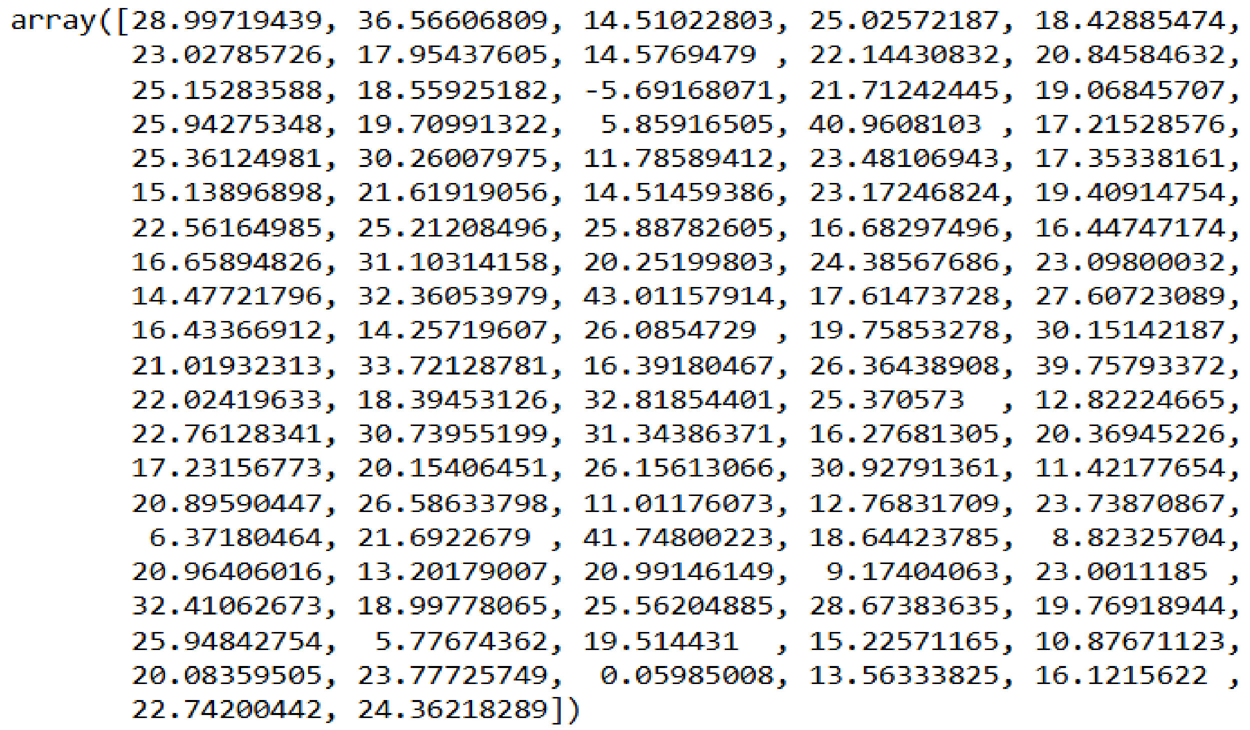
```
df.head()
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	11.43	36.2

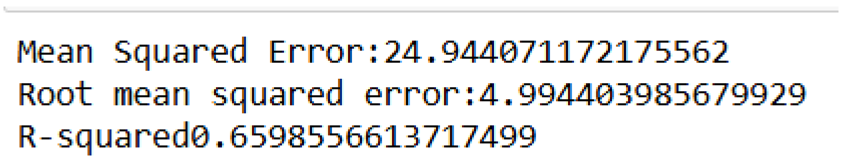
```
x = df.drop('MEDV',axis=1)

y = df['MEDV']

sc = StandardScaler()

x_s = sc.fit_transform(x)

x_train,x_test,y_train,y_test = train_test_split(x_s, y, test_size=0.2,random_state=42)
```

```
m = LinearRegression()

m.fit(x_train,y_train)

y_p = m.predict(x_test)

y_p
```



```
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])
```

```
mse = mean_squared_error(y_test,y_p)

print(f'Mean Squared Error:{mse}')


rmse = np.sqrt(mse)

print(f'Root mean squared error:{rmse}')


r2 = r2_score(y_test,y_p)

print(f'R-squared{r2}')
```



```
Mean Squared Error:24.944071172175562
Root mean squared error:4.994403985679929
R-squared0.6598556613717499
```

7b. Develop a program to demonstrate the working of Polynomial Regression. Use Auto MPG Dataset (for vehicle fuel efficiency prediction)

Polynomial Regression

Definition

Polynomial regression is a type of regression analysis used in statistics and machine learning when the relationship between the independent variable (input) and the dependent variable (output) is not linear. While simple linear regression models the relationship as a straight line, polynomial regression allows for more flexibility by fitting a polynomial equation to the data.

Polynomial Regression is an extension of Linear Regression where the relationship between variables is modeled using a polynomial equation: where n represents the degree of the polynomial. Importance of Polynomial Regression. When the relationship between variables is non-linear and a straight line does not fit well. Captures curved patterns in data by introducing higher-degree polynomial terms.

Working of Polynomial Regression

1. Transform the input features by introducing polynomial terms.
2. Apply Linear Regression to fit the transformed dataset.
3. Choose the optimal polynomial degree to balance underfitting and overfitting.

   Choosing the Right Degree (n)

   Degree 1: Equivalent to Linear Regression.

   Degree 2-3: Captures slight curves in data while preventing overfitting.

   Degree >3: More flexible but risks overfitting (too much complexity).71

Applications of Polynomial Regression

- Predicting fuel efficiency based on vehicle characteristics.
- Modeling economic growth trends over time.
- Analyzing the effect of temperature on crop yields.

```
#import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
```

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split

import warnings

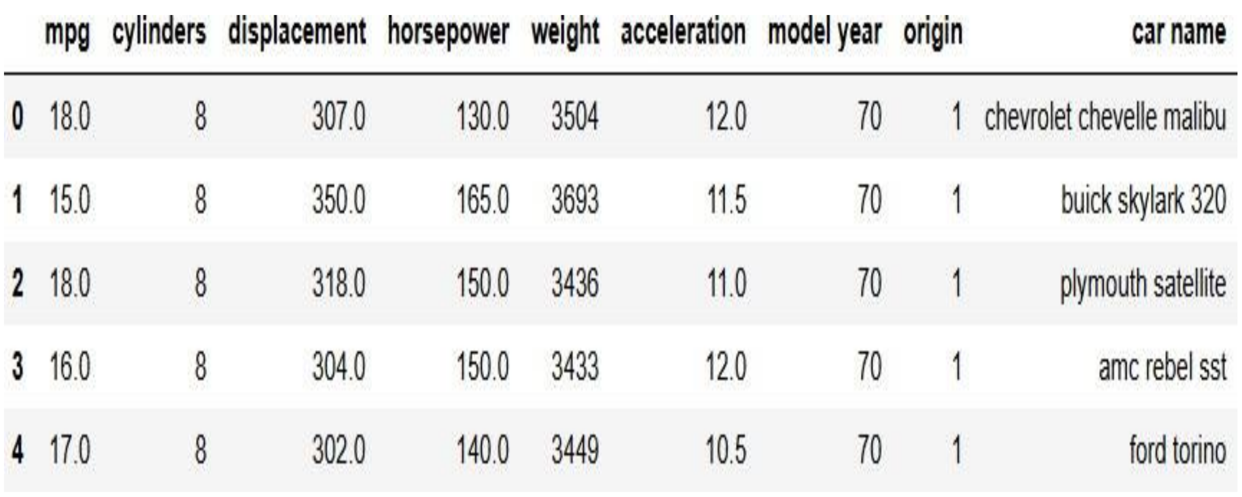
warnings.filterwarnings("ignore")

data = pd.read_csv(r'auto-mpg.csv')

data.shape

```
(398, 9)
```

data.head()



	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

data.isnull().sum()

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

df = data.copy()

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

X = df[['horsepower']]

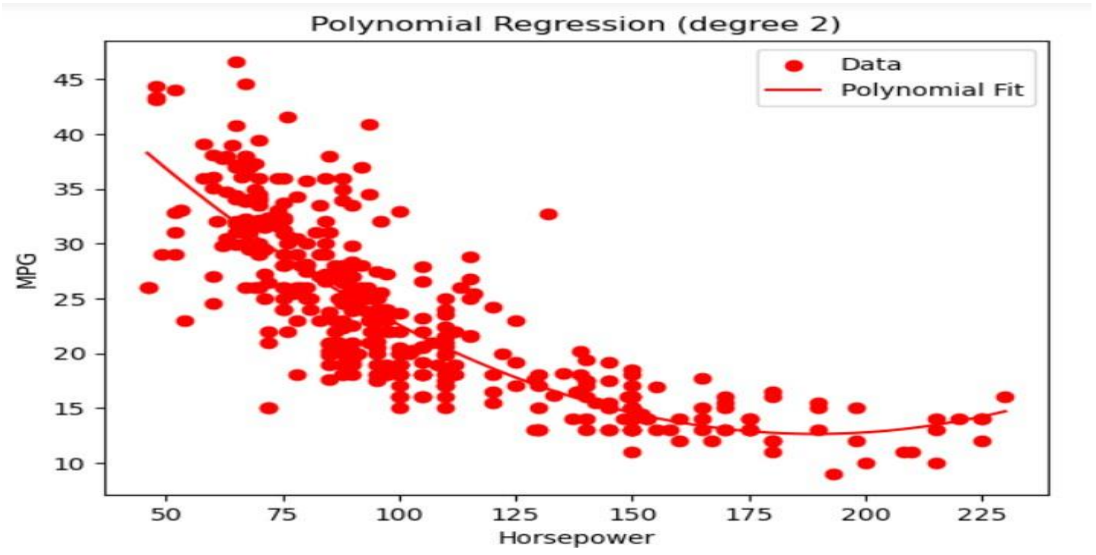
y = df['mpg']

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

```
degree = 2
poly = PolynomialFeatures(degree)
X_poly_train = poly.fit_transform(X_train)
model = LinearRegression()
model.fit(X_poly_train, y_train)
```

```
LinearRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
plt.scatter(X, y, color='red', label='Data')
X_range = np.linspace(X.min(), X.max(), 100)
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Fit')
plt.xlabel('Horsepower')

plt.ylabel('MPG')
plt.legend()
plt.title(f'Polynomial Regression (degree {degree})')
plt.show()
```



Experiment 8

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

Introduction to Decision Trees

What is a Decision Tree?

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. It models decisions using a tree-like structure where:

Nodes represent decision points based on feature values.

Edges represent possible outcomes (branches).

Leaves represent the final decision or classification.

Decision trees work by recursively splitting data into subsets based on the most significant feature, ensuring maximum information gain at each step.

Working of the Decision Tree Algorithm

1. **Selecting the Best Feature for Splitting**

At each step, the algorithm selects the feature that best separates the data. Common methods for choosing the best feature include:

Gini Impurity

Gini = 1- ∑Pi2

Measures how often a randomly chosen element would be incorrectly classified. Entropy (Information Gain)

Entropy = ∑p(X)log p(X)

Measures the uncertainty in a dataset and selects splits that maximize information gain.

Chi-Square Test Evaluates the statistical significance of the feature split.

1. **Splitting the Data:**
   - The dataset is divided into subsets based on the selected feature.
   - The process continues recursively until:
   - A stopping condition is met (e.g., pure classification, max depth).

- ➢ The tree reaches a predefined depth.

2. Making Predictions

For a new sample, traverse the tree from the root to a leaf node. The leaf node contains the predicted class label.

Advantages of Decision Trees

- ➢ Easy to interpret – Mimics human decision-making.
- ➢ Handles both numerical & categorical data.
- ➢ Requires little data preprocessing – No need for feature scaling.
- ➢ Works well with missing values.

Challenges of Decision Trees

- ➢ Overfitting – Deep trees may memorize noise instead of patterns.
- ➢ Bias towards dominant features – Features with more categories can lead to biased splits.
- ➢ Instability – Small data variations can lead to different trees.

Optimizing Decision Trees

1. Pruning

Pre-Pruning: Stop the tree early using conditions (e.g., min samples per split).

Post-Pruning: Remove unnecessary branches after the tree is built.

2. Setting Tree Depth

Limiting maximum depth prevents overfitting.

3. Using Ensemble Methods

Random Forest: Combines multiple trees for better generalization. Gradient

Boosting: Sequentially improves predictions.

Applications of Decision Trees

- ➢ Medical Diagnosis – Classifying diseases based on symptoms.
- ➢ Fraud Detection – Identifying fraudulent transactions.
- ➢ Customer Segmentation – Categorizing users based on behavior.

Program

# 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

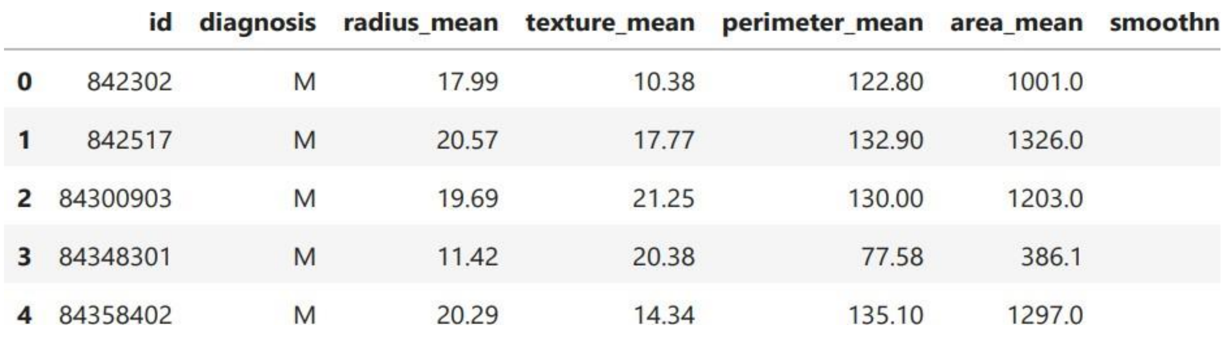
warnings.filterwarnings('ignore')

#Load Dataset

data = pd.read_csv(r'Breast Cancer Datset')

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

data.head()

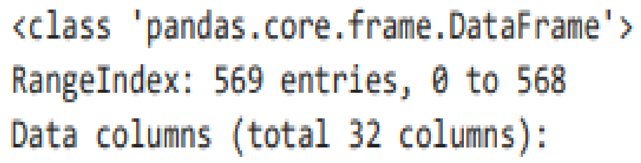


	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

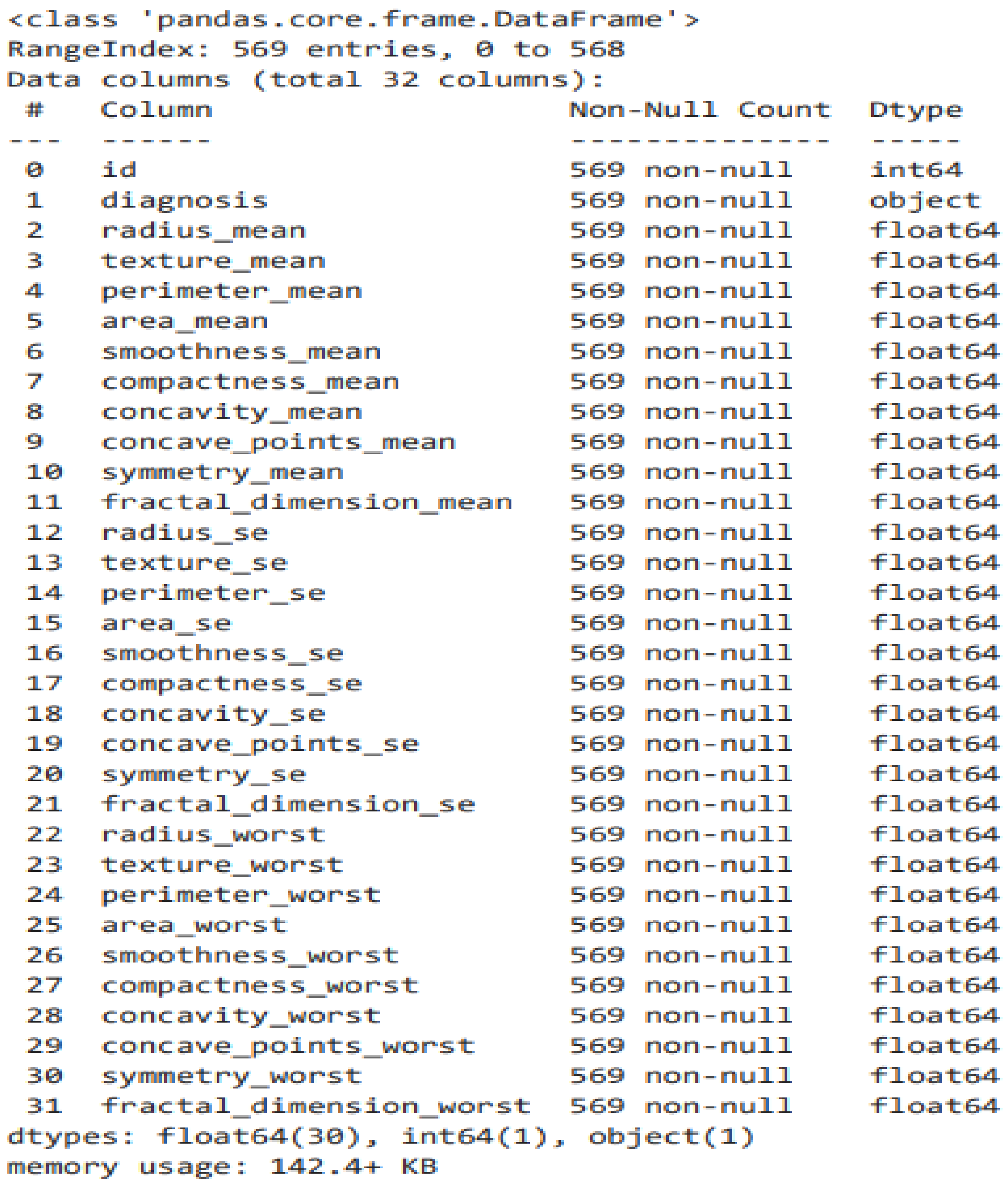
data.shape()

(569, 32)

data.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
```



```
<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.duplicated().sum()
```

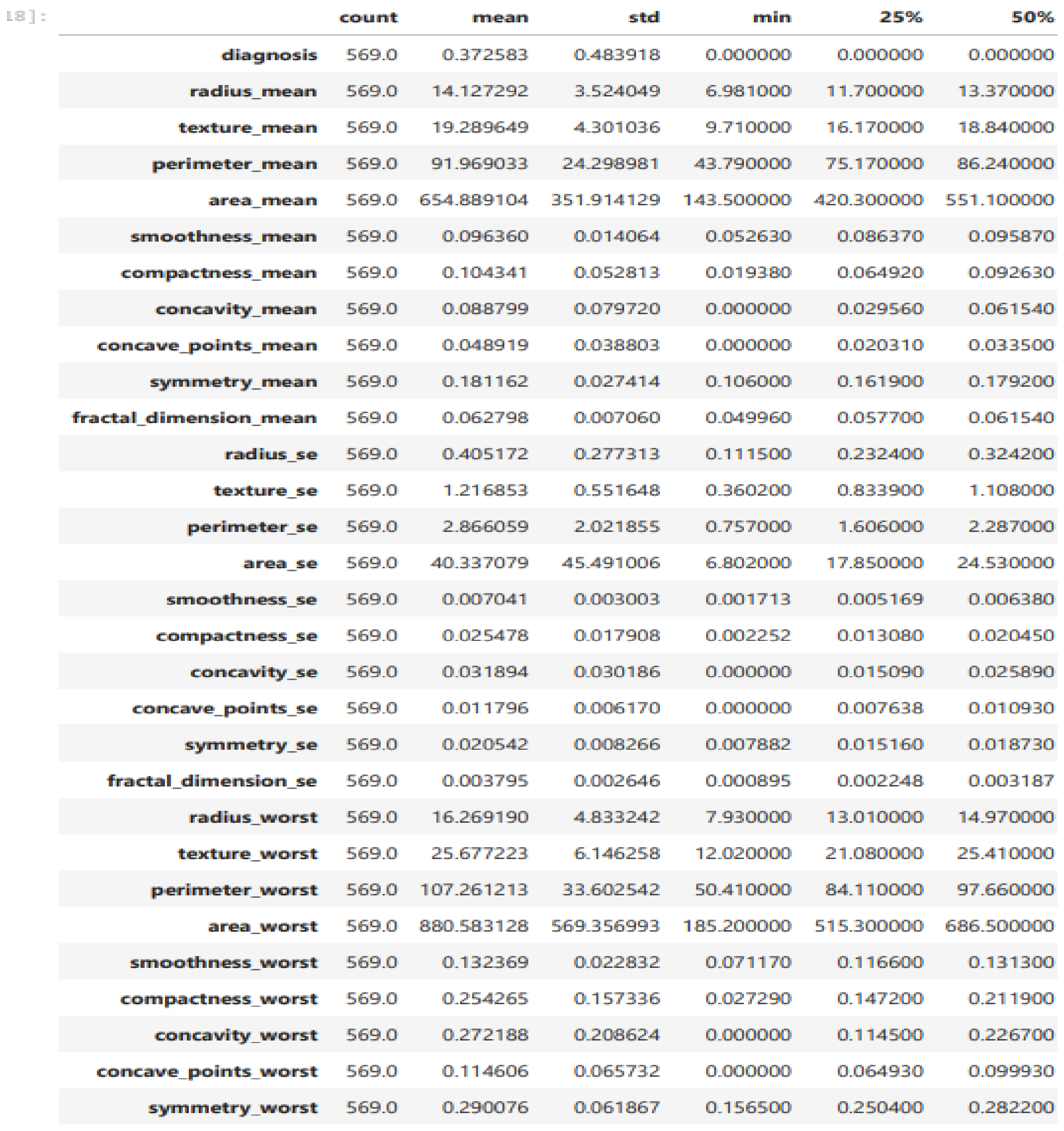
```
np.int64(0)
```

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

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

Discriptive Statistics

df.describe().T



18]:

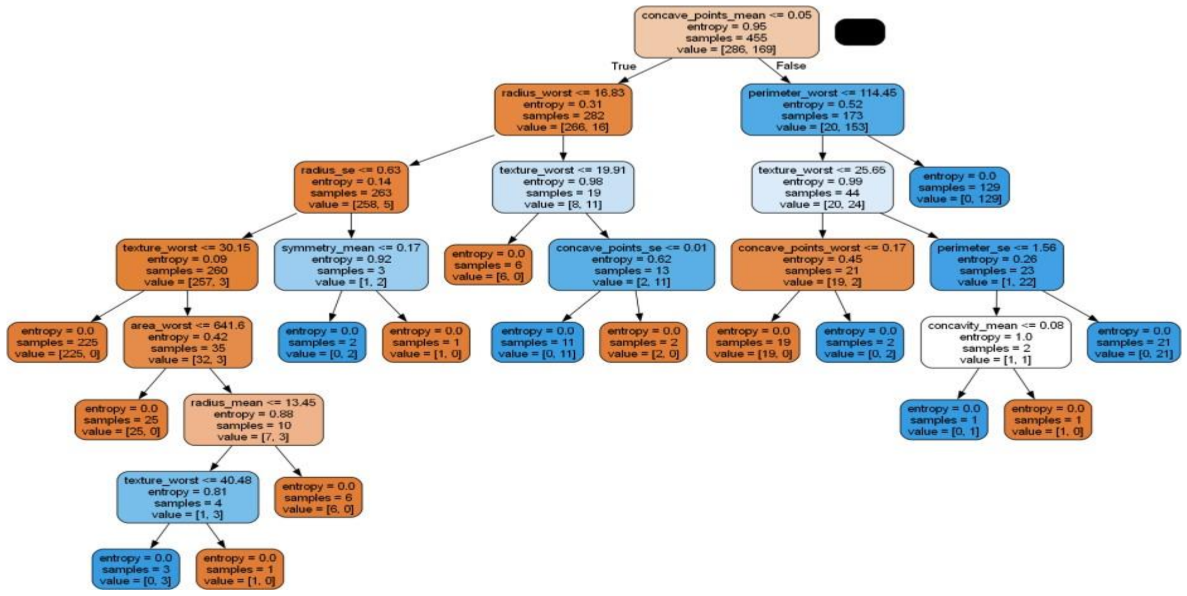
	count	mean	std	min	25%	50%
diagnosis	569.0	0.372583	0.483918	0.000000	0.000000	0.000000
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540
concave_points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890
concave_points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700
concave_points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200

# Export the tree to DOT format

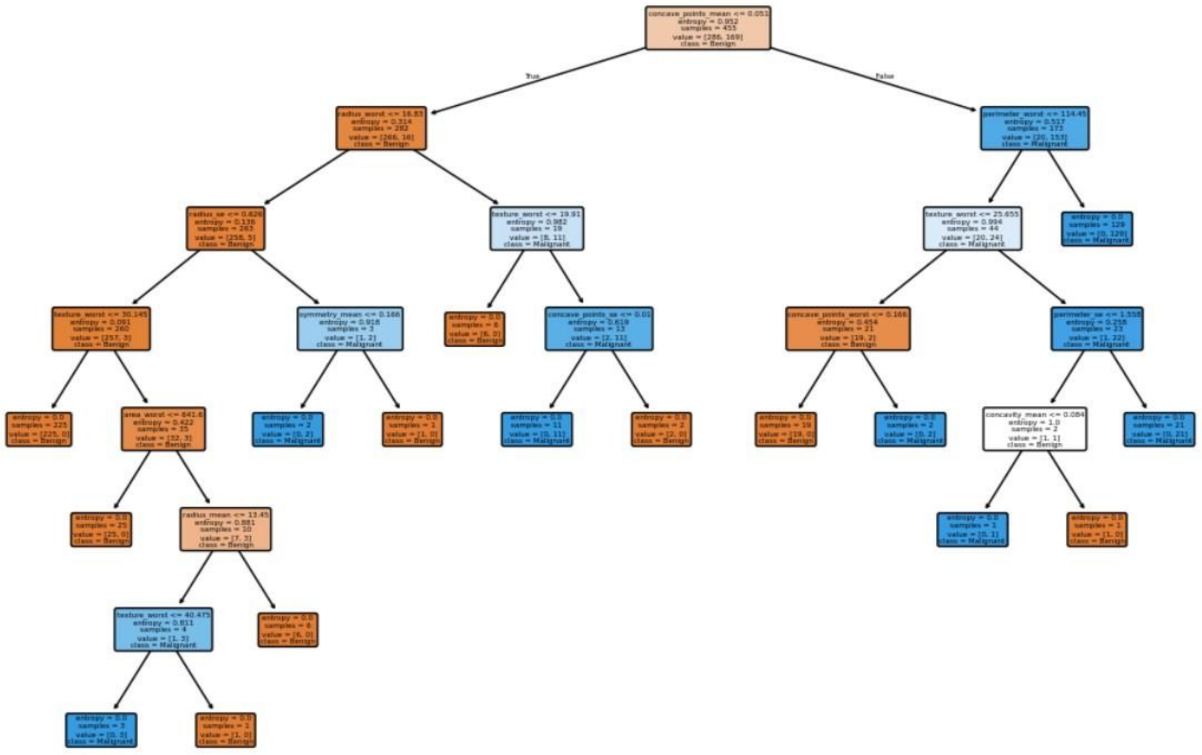
```
dot_data = export_graphviz(model, out_file=None,
 feature_names=X_train.columns,
 rounded=True, proportion=False,
 precision=2, filled=True)
# Convert DOT data to a graph
graph = pydotplus.graph_from_dot_data(dot_data)
```

# Display the graph

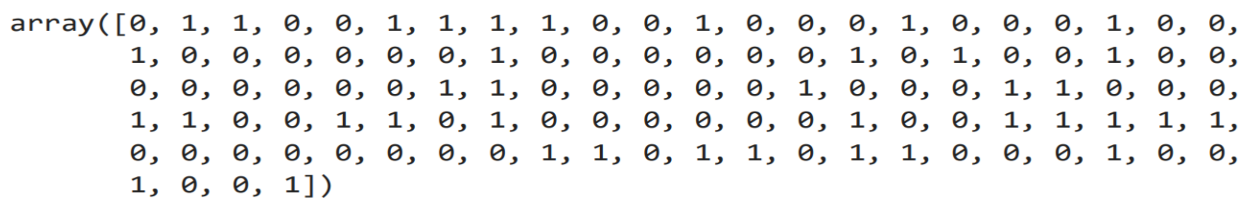
Image(graph.create_png())



# Visualize the Decision Tree (optional)
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, feature_names=X.columns, class_names=['Benign', 'Mali
plt.show()

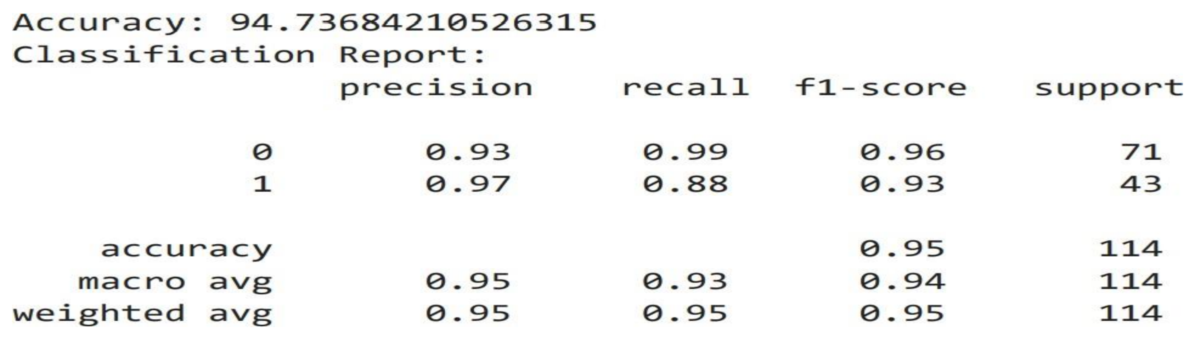


```
y_pred = model.predict(X_test)
y_pred
```



```
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, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 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
```

```
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
```

Experiment 9

Develop a program to implement the Naive Bayesian classifier, considering the 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)

Introduction to Naive Bayes Classification

What is Naive Bayes?

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem with the naïve assumption that features are independent of each other. Despite this strong assumption, it performs well in many real-world scenarios.

It is widely used for text classification, spam detection, medical diagnosis, and facial recognition.

Program

```
#import necessary libraries:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
#Load the dataset locally
data=np.load('Olivetti_faces_offline.npz')
```

```
images=data['images']

targets=data['target']
```

#Display first t faces with 1Ds

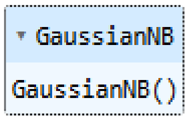
```
plt.figure(figsize=(10,8))

for i in range(20):

   plt.subplot(4,5,i+1)

   plt.imshow(images[i],cmap='gray')

   plt.title(f"ID:{targets[i]}")

   plt.axis('off')

   plt.tight_layout()

   plt.show()
```

#Flatten the images for training (convert 3D to 2D)

```
x=images.reshape((images.shape[0],-1))

y=targets
```

#Train _test Split

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,stratify=y,random_state=42)

model=GaussianNB()

model.fit(x_train,y_train)
```



```
GaussianNB
GaussianNB()
```

#Predict and Evalute

```
y_pred=model.predict(x_test)

accuracy=accuracy_score(y_test,y_pred)

print(f"Accuracy:{accuracy*100:.2f}%")
```

```
Accuracy:92.00%
```

Experiment 10

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

What is Clustering?

Clustering is an unsupervised machine learning technique used to group data points into clusters based on their similarity. The goal is to identify hidden patterns or natural groupings in the data.

One of the most widely used clustering algorithms is K-Means Clustering, which divides the dataset into K clusters, where each data point belongs to the nearest cluster center.

What is K-Means Clustering?

K-Means is a centroid-based clustering algorithm that partitions data into K clusters by minimizing the variance within each cluster.

Working of K-Means Algorithm

1. Choose the number of clusters (K).
2. Randomly initialize K cluster centroids.
3. Assign each data point to the nearest centroid based on distance (e.g., Euclidean distance).
4. Update the centroids by computing the mean of all points assigned to each cluster.
5. Repeat Steps 3 and 4 until convergence (when centroids no longer change significantly).

Mathematical Representation

The objective is to minimize the sum of squared distances (SSD) between data points and their assigned cluster centroid: where:
K = Number of clusters

xj = Data point

μi = Centroid of cluster Ci

Choosing the Optimal Number of Clusters (K)
Selecting the right value of K is crucial. Some common methods include:
Elbow Method:

Plots the within-cluster sum of squares (WCSS) for different K values.

The "elbow point" where WCSS stops decreasing significantly is chosen as the optimal K.

1. Silhouette Score: Measures how well-separated the clusters are. A higher score indicates better clustering.
2. Gap Statistics:

Compares clustering performance to randomly generated reference data. Distance Metrics in K-Means typically uses Euclidean Distance to measure how close a data point is to a centroid:

Other distance metrics include:

Manhattan

Distance Cosine

Similarity

Mahalanobis

Distance

Advantages of K-Means Clustering

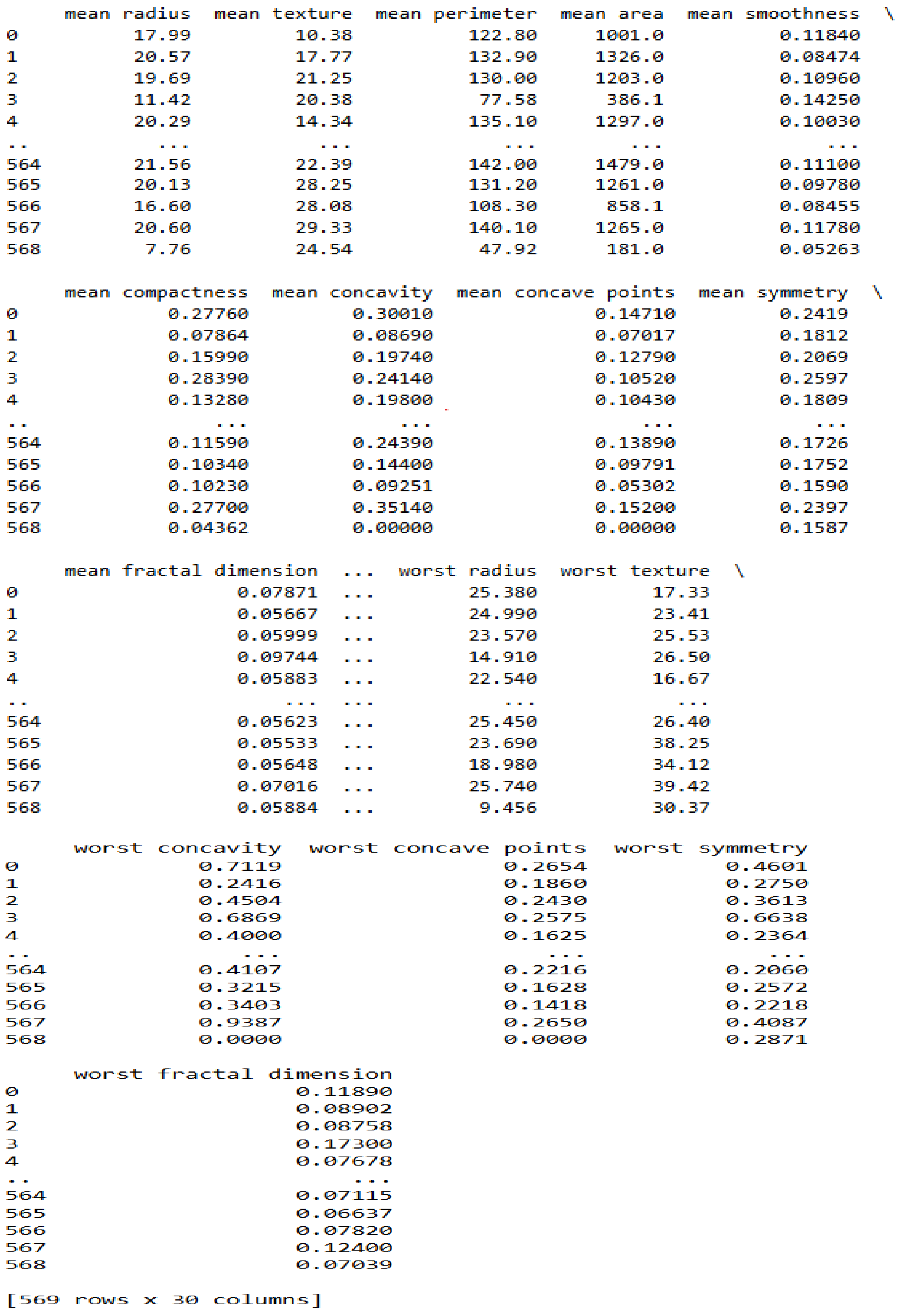
✓ Efficient and Scalable – Works well with large datasets.

✓ Easy to Implement – Simple and interpretable.

✓ Handles High-Dimensional Data – Can work on complex datasets.

Program

```
#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
#load the dataset
data=load_breast_cancer()
df=pd.DataFrame(data.data,columns=data.feature_names)
print(df)
```



```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           ...           ...             ...        ...              ...
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                ...             ...                  ...            ...
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst radius  worst texture  \
0                   0.07871  ...        25.380          17.33
1                   0.05667  ...        24.990          23.41
2                   0.05999  ...        23.570          25.53
3                   0.09744  ...        14.910          26.50
4                   0.05883  ...        22.540          16.67
..                      ...  ...           ...            ...
564                 0.05623  ...        25.450          26.40
565                 0.05533  ...        23.690          38.25
566                 0.05648  ...        18.980          34.12
567                 0.07016  ...        25.740          39.42
568                 0.05884  ...         9.456          30.37

     worst concavity  worst concave points  worst symmetry
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
..               ...                   ...             ...
564           0.4107                0.2216          0.2060
565           0.3215                0.1628          0.2572
566           0.3403                0.1418          0.2218
567           0.9387                0.2650          0.4087
568           0.0000                0.0000          0.2871

     worst fractal dimension
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.17300
4                    0.07678
..                       ...
564                  0.07115
565                  0.06637
566                  0.07820
567                  0.12400
568                  0.07039

[569 rows x 30 columns]
```

#Standardize the dataset

scaler=StandardScaler()

df_scaled=scaler.fit_transform(df)

#Apply K-means clustering

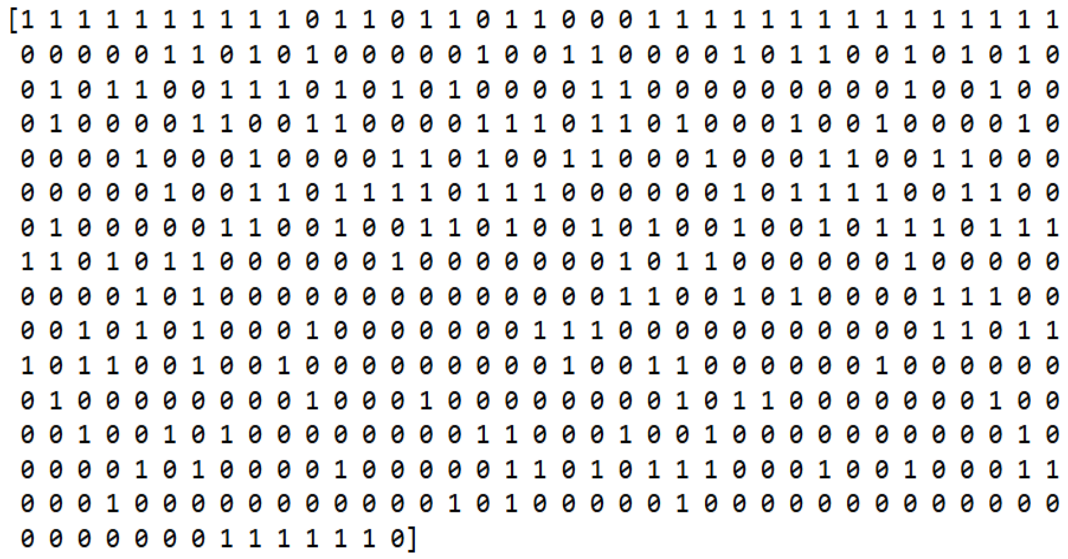
k=2

kmeans=KMeans(n_clusters=k, random_state=42,n_init=10)

kmeans.fit(df_scaled)

labels=kmeans.labels_

print(labels)



```
[1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0
 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0
 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 1 1 0 1 1 1
 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

#Reduce dimensions using PCA for visualization

pca=PCA(n_components=2)

df_pca=pca.fit_transform(df_scaled)

#Create a dataframe for visualization

df_visual=pd.DataFrame(df_pca,columns=['PC1','PC2'])

df_visual['cluster']=labels

#plot the clusters

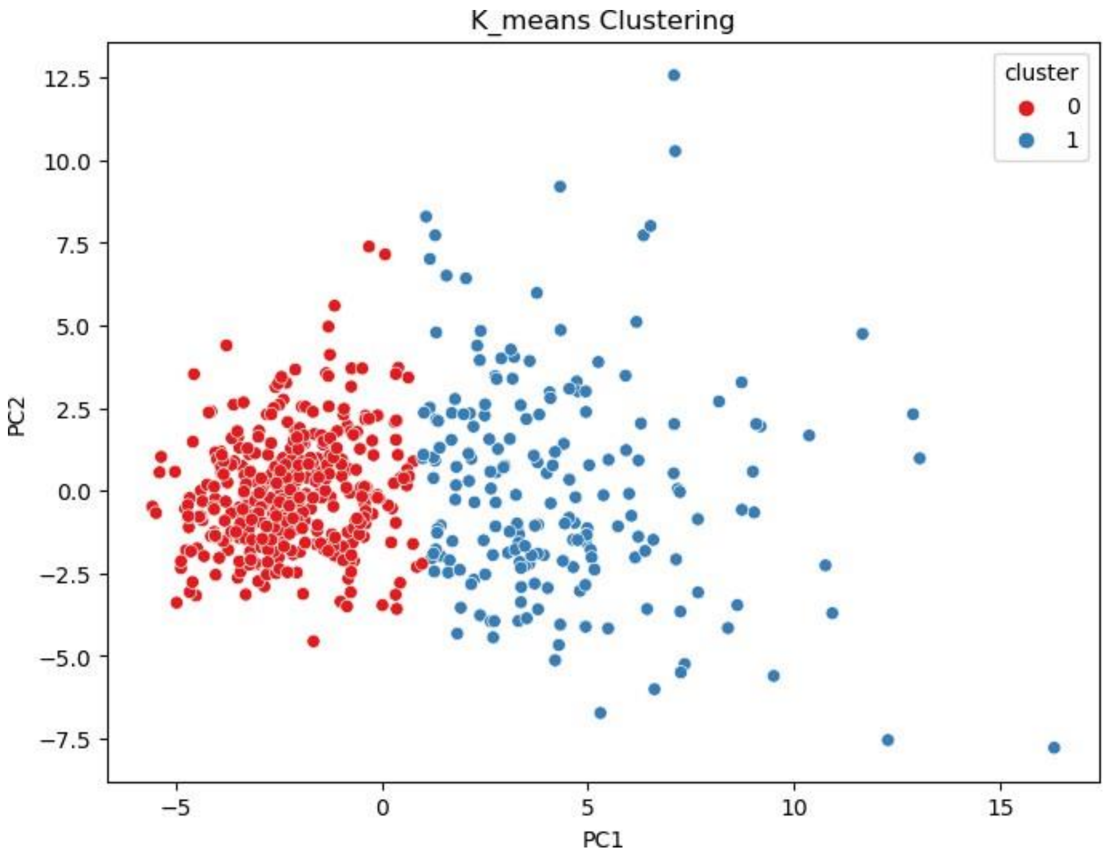
plt.figure(figsize=(8,6))

sns.scatterplot(x='PC1',y='PC2',hue=df_visual['cluster'],palette='Set1',data=df_visual)

plt.title("K_means Clustering")

plt.xlabel('PC1')

```
plt.ylabel('PC2')

plt.legend(title='cluster')

plt.show()
```


K_means Clustering
cluster
PC1
PC2