RAJEEV INSTITUTE OF TECHNOLOGY

HASSAN-573201

Machine Learning Laboratory

(BCSL606)

As per VTU Syllabus/scheme for 6th Semester

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

VISION & MISSION OF THE INSTITUTE

VISION:

- To be an academic institution in vibrant social & economic environment, striving continuously for excellence in education, research and technological service to the society.

MISSION:

- To achieve academic excellence in engineering and management through dedication to duty, offering state of the art education and faith in human values.
- To create and endure a community of learning among students, develop outstanding professionals with high ethical standards.
- To provide academic ambience conducive to the development, needs and growth of society and the industry.

VISION & MISSION OF THE DEPARTMENT

VISION:

- To become an exemplary center for engineering education and research in the frontier areas of Computer Science and Engineering.

MISSION:

- Render quality education through best teaching learning process and modern educational tools to enable students for good careers.
- Impart essential skills to make students industry ready and future professional with social concern.
- Provide facilities and expertise in recent computer technology to promote research and innovations.

PROGRAM SPECIFIC OUTCOMES (PSO'S)

The graduates of Computer Science & engineering program of Rajeev Institute of Technology should be able to attain the following at the time of graduation.

PSO1: To apply software engineering principles and practices to design, develop and implement software solutions.

PSO2: To use recent software tools to solve and analyze a given problem.

PROGRAM EDUCATIONAL OBJECTIVES (PEO'S)

The program educational objectives are the statements that describe the expected achievements of graduates within first few years of their graduation from the program. The program educational objectives of Bachelor of Computer Science & Engineering at Rajeev Institute of Technology can be broadly defined as,

PEO1: Graduates will become software developers in diverse domains and/or pursue higher studies.

PEO2: Graduates will process an agility to adapt to challenges in their profession.

PEO3: Graduates exhibit leadership qualities and will be a team player.

PEO4: Graduates will be able to communicate effectively and cultivate ethics.

PROGRAM OUTCOMES

Graduation students of Bachelor of Computer Science and Engineering program at Rajeev Institute of Technology will attain the following program outcomes in the field of Computer Science and Engineering.

PO1-Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2-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.

PO3-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.

PO4-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.

PO5-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.

PO6-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.

PO7-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.

PO8-Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9-Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10-Communication: Communicate effectively on complex engineering activities with the engineering community and with the 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.

PO11-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 environment.

PO12-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.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MODULES OF THE SUBJECT TAUGHT FOR THE SESSION AY:2024-25

Course Syllabus with CO's

Academic Year: 2024 – 2025							
Department: **COMPUTER SCIENCE AND ENGINEERING**							
Course Code	Course Title	Core/ Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
BCSL606	MACHINE LEARNING LAB	Core	**Statistics, Programming concepts**	**-**	**-**	**2**	**12**
Objectives	Course Learning Objectives CLO 1. To become familiar with data and visualize univariate, bivariate, and multivariate data using statistical techniques and dimensionality reduction. CLO 2. To understand various machine learning algorithms such as similarity-based learning, regression, decision trees, and clustering. CLO 3. To familiarize with learning theories, probability-based models and developing the skills required for decision-making in dynamic environments.						

Programs Covered as per Syllabus

Program 1: Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

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

Program 3: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

Program 4: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

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

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

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

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

Program 9: Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

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

List of URL s, Text Books, Notes, Multimedia Content

Suggested Learning Resources:

1. S Sridhar and M Vijayalakshmi, "Machine Learning", Oxford University Press, 2021.
2. M N Murty and Ananthanarayana V S, "Machine Learning: Theory and Practice", Universities Press (India) Pvt. Limited, 2024.

Web links and Video Lectures (e-Resources):

- https://www.drssridhar.com/?page_id=1053
- https://www.universitiespress.com/resources?id=9789393330697
- https://onlinecourses.nptel.ac.in/noc23_cs18/preview

Course Outcomes	At the end of the course, the student will be able to: CO 1. Illustrate the principles of multivariate data and apply dimensionality reduction techniques. CO 2. Demonstrate similarity-based learning methods and perform regression analysis. CO 3. Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning. CO 4. Implement the clustering algorithms to share computing resources.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum

total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are 50 Marks.
The split-up of CIE marks for record/ journal and test are in the ratio 60:40.
• Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
• Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
• Total marks scored by the students are scaled down to 30 marks (60% of maximum marks).
• Weightage to be given for neatness and submission of record/write-up on time.
• Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
• In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
• The suitable rubrics can be designed to evaluate each student's performance and learning ability.
• The marks scored shall be scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.
The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

• SEE marks for the practical course are 50 Marks.
• SEE shall be conducted jointly by the two examiners of the same institute; examiners are appointed by the Head of the Institute.
• The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
• All laboratory experiments are to be included for practical examination.
• (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.
• Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
• Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)
Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.
The minimum duration of SEE is 02 hours

The Correlation of Course Outcomes (CO's) and Program Outcomes (PO's)

Subject Code: BCSL606												
List of Course Outcomes	**Program Outcomes**											
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO 10**	**PO 11**	**PO 12**
CO-1	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**2**
CO-2	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CO-3	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CO-4	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
Average	**3**	**3**	**3**	**2**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**

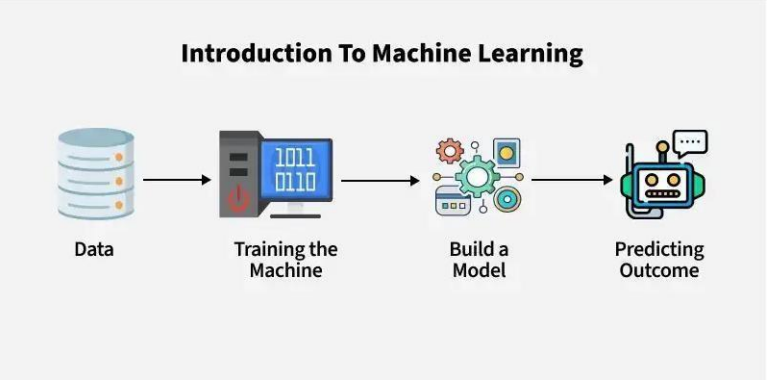
Note: 3 = Strong Contribution 2 = Average Contribution 1 = Weak Contribution 0 = No Contribution

The Correlation of Course Outcomes (CO's) and Program Specific Outcomes (PSO's)

Subject Code: BCSL606		
List of Course Outcomes	**PSO1**	**PSO2**
CO-1	**2**	**2**
CO-2	**2**	**2**
CO-3	**2**	**2**
CO-4	**2**	**2**
Average	**2**	**2**

Introduction to Machine Learning

Machine learning (ML) allows computers to learn and make decisions without being explicitly programmed. It involves feeding data into algorithms to identify patterns and make predictions on new data. Machine learning is used in various applications, including image and speech recognition, natural language processing, and recommender systems.



Why do we need Machine Learning?

Machine Learning algorithm learns from data, train on patterns, and solve or predict complex problems beyond the scope of traditional programming. It drives better decision-making and tackles intricate challenges efficiently.

1. Solving Complex Business Problems

Traditional programming struggles with tasks like image recognition, natural language processing (NLP), and medical diagnosis. ML, however, thrives by learning from examples and making predictions without relying on predefined rules.

Example Applications:

- Image and speech recognition in healthcare.
- Language translation and sentiment analysis.

2. Handling Large Volumes of Data

With the internet's growth, the data generated daily is immense. ML effectively processes and analyzes this data, extracting valuable insights and enabling real-time predictions.

Use Cases:

- Fraud detection in financial transactions.
- Social media platforms like Facebook and Instagram predicting personalized feed recommendations from billions of interactions.

3. Automate Repetitive Tasks

ML automates time-intensive and repetitive tasks with precision, reducing manual effort and error-prone systems.

Examples:

- **Email Filtering**: Gmail uses ML to keep your inbox spam-free.
- **Chatbots**: ML-powered chatbots resolve common issues like order tracking and password resets.
- **Data Processing**: Automating large-scale invoice analysis for key insights.

4. Personalized User Experience

ML enhances user experience by tailoring recommendations to individual preferences. Its algorithms analyze user behavior to deliver highly relevant content.

Real-World Applications:

- **Netflix**: Suggests movies and TV shows based on viewing history.
- **E-Commerce**: Recommends products you're likely to purchase.

5. Self-Improvement in Performance

ML models evolve and improve with more data, making them smarter over time. They adapt to user behavior and refine their performance.

Examples:

- **Voice Assistants** (e.g., Siri, Alexa): Learn user preferences, improve voice recognition, and handle diverse accents.
- **Search Engines**: Refine ranking algorithms based on user interactions.
- **Self-Driving Cars**: Enhance decision-making using millions of miles of data from simulations and real-world driving.

What Makes a Machine "Learn"?

A machine "learns" by recognizing patterns and improving its performance on a task based on data, without being explicitly programmed.

The process involves:

1. **Data Input:** Machines require data (e.g., text, images, numbers) to analyze.
2. **Algorithms:** Algorithms process the data, finding patterns or relationships.
3. **Model Training:** Machines learn by adjusting their parameters based on the input data using mathematical models.
4. **Feedback Loop:** The machine compares predictions to actual outcomes and corrects errors (via optimization methods like gradient descent).

5. **Experience and Iteration:** Repeating this process with more data improves the machine's accuracy over time.
6. **Evaluation and Generalization:** The model is tested on unseen data to ensure it performs well on real-world tasks.

In essence, machines "learn" by continuously refining their understanding through data-driven iterations, much like humans learn from experience.

Importance of Data in Machine Learning

Data is the foundation of machine learning (ML). Without quality data, ML models cannot learn, perform, or make accurate predictions.

- Data provides the examples from which models learn patterns and relationships.
- High-quality and diverse data improves model accuracy and generalization.
- Data ensures models understand real-world scenarios and adapt to practical applications.
- Features derived from data are critical for training models.
- Separate datasets for validation and testing assess how well the model performs on unseen data.
- Data fuels iterative improvements in ML models through feedback loops.

Types of Machine Learning

1. Supervised learning

Supervised learning is a type of machine learning where a model is trained on labeled data—meaning each input is paired with the correct output. The model learns by comparing its predictions with the actual answers provided in the training data.

Both classification and regression problems are supervised learning problems.

Example: Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labeled as "healthy" or "sick"

Gender	Age	Label
M	48	sick
M	67	sick

Gender	Age	Label
F	53	healthy
M	49	sick
F	32	healthy
M	34	healthy
M	21	healthy

In this example, supervised learning is to use this labeled data to train a model that can predict the label ("healthy" or "sick") for new patients based on their gender and age. For instance, if a new patient (e.g., Male, 50 years old) visits the clinic, the model can classify whether the patient is "healthy" or "sick" based on the patterns it learned during training.

2. Unsupervised learning:

Unsupervised learning algorithms draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, classification or categorization is not included in the observations.

Example: Consider the following data regarding patients entering a clinic. The dataset includes **unlabeled data**, where only the gender and age of the patients are available, with no health status labels.

Gender	Age
M	48
M	67
F	53
M	49

Gender	Age
F	34
M	21

Here, unsupervised learning technique will be used to find patterns or groupings in the data such as clustering patients by age or gender. For example, the algorithm might group patients into clusters, such as "younger healthy patients" or "older patients," without prior knowledge of their health status.

3. Reinforcement Learning

Reinforcement Learning (RL) trains an agent to act in an environment by maximizing rewards through trial and error. Unlike other machine learning types, RL doesn't provide explicit instructions.

Instead, the agent learns by:

- **Exploring Actions**: Trying different actions.
- **Receiving Feedback**: Rewards for correct actions, punishments for incorrect ones.
- **Improving Performance**: Refining strategies over time.

Example: Identifying a Fruit

The system receives an input (e.g., an apple) and initially makes an incorrect prediction ("It's a mango"). Feedback is provided to correct the error ("Wrong! It's an apple"), and the system updates its model based on this feedback.

Over time, it learns to respond correctly ("It's an apple") when encountering similar inputs, improving accuracy through trial, error, and feedback.

Benefits of Machine Learning

- **Enhanced Efficiency and Automation:** ML automates repetitive tasks, freeing up human resources for more complex work. It also streamlines processes, leading to increased efficiency and productivity.
- **Data-Driven Insights:** ML can analyze vast amounts of data to identify patterns and trends that humans might miss. This allows for better decision-making based on real-world data.
- **Improved Personalization:** ML personalizes user experiences across various platforms. From recommendation systems to targeted advertising, ML tailors content and services to individual preferences.
- **Advanced Automation and Robotics:** ML empowers robots and machines to perform complex tasks with greater accuracy and adaptability. This is revolutionizing fields like manufacturing and logistics

Introduction to data Visualization

Data visualization is the graphical representation of data to help people understand the patterns, trends, and insights within the information. It involves using visual elements such as charts, graphs, and maps to convey complex data in an accessible and understandable manner. The primary goal of data visualization is to make data more interpretable, allowing users to extract meaningful information efficiently.

Key Concepts in Data Visualization:

Data Representation: Data visualization involves transforming raw data into visual forms, making it easier to grasp patterns, correlations, and outliers.

Visual Elements: Common visual elements include points, lines, bars, shapes, colors, and labels. Each element is chosen to effectively represent different types of data and relationships.

Types of Visualizations: Different types of visualizations are suitable for different types of data. Common types include:

Line Charts: Show trends and patterns over time.

Bar Charts: Compare values across categories.

Scatter Plots: Display relationships between two variables.

Pie Charts: Represent parts of a whole.

Interactivity: Interactive visualizations allow users to explore and interact with data dynamically. This can enhance the depth of understanding and enable users to focus on specific aspects.

Storytelling: Effective data visualization often tells a story. It guides viewers through a narrative, helping them uncover insights and draw conclusions.

Color and Design: Color choices and design elements play a crucial role. They should enhance understanding, avoid confusion, and highlight important information.

Data Preparation: Before creating visualizations, it's important to clean and prepare the data. This includes handling missing values, outliers, and ensuring that the data is in a suitable format for visualization.

Importance of Data Visualization:

- **Clarity and Understanding:** Visualizations simplify complex data, making it easier for both experts and non-experts to understand the information.

- **Identification of Patterns:** Visualizations reveal patterns and trends in data that may not be apparent in raw datasets.
- **Communication:** Effective visualizations facilitate communication of findings, insights, and trends to a broader audience.
- **Decision-Making:** Visualizations aid decision-making processes by providing a clear and concise representation of relevant information.
- **Exploration and Analysis:** Interactive visualizations allow users to explore data from different perspectives, enabling deeper analysis and discovery.
- **Memorability:** Well-designed visualizations are more memorable than raw data, helping people retain and recall information.

Tools for Data Visualization:

- **matplotlib:** A versatile 2D plotting library for Python.
- **seaborn:** Built on Matplotlib, Seaborn provides a high-level interface for statistical data visualization.
- **pandas:** Pandas is a powerful Python library used for data manipulation and analysis.
- **numpy:** NumPy (Numerical Python) is a powerful library for numerical computing in Python. It provides efficient handling of large arrays, matrices, and mathematical functions.
- **sklearn**: scikit-Learn (sklearn) is one of the most popular Python libraries for **machine learning**. It provides efficient tools for data preprocessing, model selection, training, and evaluation.

In summary, data visualization is a powerful tool for making data more accessible, understandable, and actionable. It plays a crucial role in the data analysis process, aiding in exploration, communication, and decision-making.

Introduction to Python

Python is a high-level, general-purpose programming language known for its readability, simplicity, and versatility. Developed by Guido van Rossum, Python has become one of the most popular programming languages worldwide. Here's a comprehensive introduction to Python:

Key Features of Python:

- **Readability:** Python emphasizes code readability and uses English-like syntax. This makes it easier for developers to write and maintain code.
- **Versatility:** Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It can be used for a wide range of applications, from web development to scientific computing.
- **Large Standard Library:** Python comes with a vast standard library that includes modules and packages for various tasks, making it a powerful and resourceful language.
- **Interpreted and Interactive:** Python is an interpreted language, allowing for quick development and testing. It also supports an interactive mode where code can be executed line by line.
- **Dynamic Typing:** Python uses dynamic typing, meaning you don't need to declare variable types explicitly. This allows for more flexibility and faster development.
- **Community and Documentation:** Python has a large and active community, contributing to its rich ecosystem of libraries and frameworks. Additionally, Python documentation is comprehensive and user-friendly.

Basic Syntax:

```
# Hello, World! program

print("Hello, World!")

# Variables and types

x = 10

y = 3.14

name = "Python"


# Lists

my_list = [1, 2, 3, 4, 5]

# Conditional statements

if x > 5:
```

```
        print("x is greater than 5")
    elif x =5:
        print("x is equal to 5")
    else:
        print("x is less than 5") #
```

Loops

```
  for i in range(5):
      print(i)
```

 # Functions

```
def greet(name):
    print("Hello, " + name + "!") #
```

Function call

```
greet("Alice")
```

Data Structures:

- **Lists:** Ordered, mutable collections.

  my_list = [1, 2, 3, 4, 5]

- **Tuples:** Ordered, immutable collections.

  my_tuple = (1, 2, 3)

- **Dictionaries:** Unordered key-value pairs.

  my_dict = {"name": "John", "age":

- **Sets:** Unordered, unique elements.

  my_set = {1, 2, 3, 4, 5}

Program 1: Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

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

- **import pandas as pd** Imports the pandas library and assigns it the alias pd. pandas is used for data manipulation and analysis (like loading and processing datasets).
- **import numpy as np** Imports the numpy library with the alias np. numpy is a library for numerical computing, especially with arrays and mathematical functions.
- **import seaborn as sns** Imports the seaborn library (with the alias sns), which is a data visualization library built on top of matplotlib. It simplifies plotting and adds more advanced visualizations.
- **import matplotlib.pyplot as plt** Imports the matplotlib library for creating static, animated, and interactive plots in Python. It's the main tool used here to create plots.
- **from sklearn.datasets import fetch_california_housing** Imports the fetch_california_housing function from sklearn.datasets, which is used to load the California Housing dataset.

Step 1: Load the California Housing dataset

```
data = fetch_california_housing(as_frame=True)
housing_df = data.frame
```

- **data = fetch_california_housing(as_frame=True)** Loads the California Housing dataset using fetch_california_housing and stores the data in the variable data. The as_frame=True argument returns the data as a pandas DataFrame rather than as a dictionary of arrays.
- **housing_df = data.frame** Extracts the DataFrame (which is stored in data.frame) from the data object and assigns it to housing_df for easier reference.

Step 2: Create histograms for numerical features

```
numerical_features = housing_df.select_dtypes(include=[np.number]).columns
```

- **housing_df.select_dtypes(include=[np.number])** selects all columns from housing_df that have numerical data types (e.g., int64, float64).
- **columns-** retrieves the column names of the selected numerical features and stores them in numerical_features.

Plot histograms

```
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
      plt.subplot(3, 3, i + 1)
      sns.histplot(housing_df[feature], kde=True, bins=30, color='blue')
      plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```

Step 3: Generate box plots for numerical features

```
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
      plt.subplot(3, 3, i + 1)
      sns.boxplot(x=housing_df[feature], color='orange')
      plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()
```

- **plt.figure(figsize=(15, 10))** Creates a figure (a blank canvas) with a specified size of 15 inches by 10 inches. This is where the plots will be drawn.
- **for i, feature in enumerate(numerical_features):** Loops through the list of numerical features, where i is the index and feature is the column name for each numerical feature.
- **plt.subplot(3, 3, i + 1)** Creates a subplot in a grid layout with 3 rows and 3 columns. The i + 1 ensures each plot is placed in a different subplot. As there are 9 numerical features, this will create a 3x3 grid.
- **sns.histplot(housing_df[feature], kde=True, bins=30, color='blue')**
- **sns.histplot()** creates a histogram plot for the current numerical feature.
- **housing_df[feature]** selects the current feature's data from the DataFrame.

- **kde=True** adds a Kernel Density Estimate (a smooth curve representing the distribution).
- **bins=30** sets the number of bins (bars) in the histogram to 30.
- **color='blue'** sets the color of the histogram bars to blue.
- **plt.title(f'Distribution of {feature}')** Sets the title of the subplot to indicate which feature's distribution is being plotted.
- **plt.tight_layout()** Adjusts the layout so the plots fit within the figure without overlapping.
- **plt.show()** Displays all the plots.
- **sns.boxplot()** creates a box plot, which is useful for showing the distribution of the data and identifying outliers.

Step 4: Identify outliers using the IQR method

```
print("Outliers Detection:")
outliers_summary = {}
for feature in numerical_features:
      Q1 = housing_df[feature].quantile(0.25)
      Q3 = housing_df[feature].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR
      outliers = housing_df[(housing_df[feature] < lower_bound) |
      housing_df[feature] > upper_bound)]
      outliers_summary[feature] = len(outliers)
      print(f"{feature}: {len(outliers)} outliers")
```

- **print("Outliers Detection:")** Prints a message indicating that outliers are being detected.
- **outliers_summary = {}** Initializes an empty dictionary to store the number of outliers for each feature.
- **for feature in numerical_features:** Loops through each numerical feature in the dataset.
- **Q1 = housing_df[feature].quantile(0.25)** Calculates the first quartile (25th percentile) of the data for the current feature.
- **Q3 = housing_df[feature].quantile(0.75)** Calculates the third quartile (75th percentile) of the data for the current feature.
- **IQR = Q3 - Q1** Computes the Interquartile Range (IQR) by subtracting Q1 from Q3.
- **lower_bound = Q1 - 1.5 * IQR** Calculates the lower bound for outliers, which is 1.5 times the IQR below Q1.
- **upper_bound = Q3 + 1.5 * IQR** Calculates the upper bound for outliers, which is 1.5 times the IQR above Q3.
- **outliers = housing_df[(housing_df[feature] < lower_bound) | (housing_df[feature] > upper_bound)]** Identifies the outliers by selecting the rows where the feature's values are either below the lower bound or above the upper bound.
- **outliers_summary[feature] = len(outliers)** Records the number of outliers for the current feature in the outliers_summary dictionary.
- **print(f"{feature}: {len(outliers)} outliers")** Prints the number of outliers detected for each feature.

Optional: Print a summary of the dataset

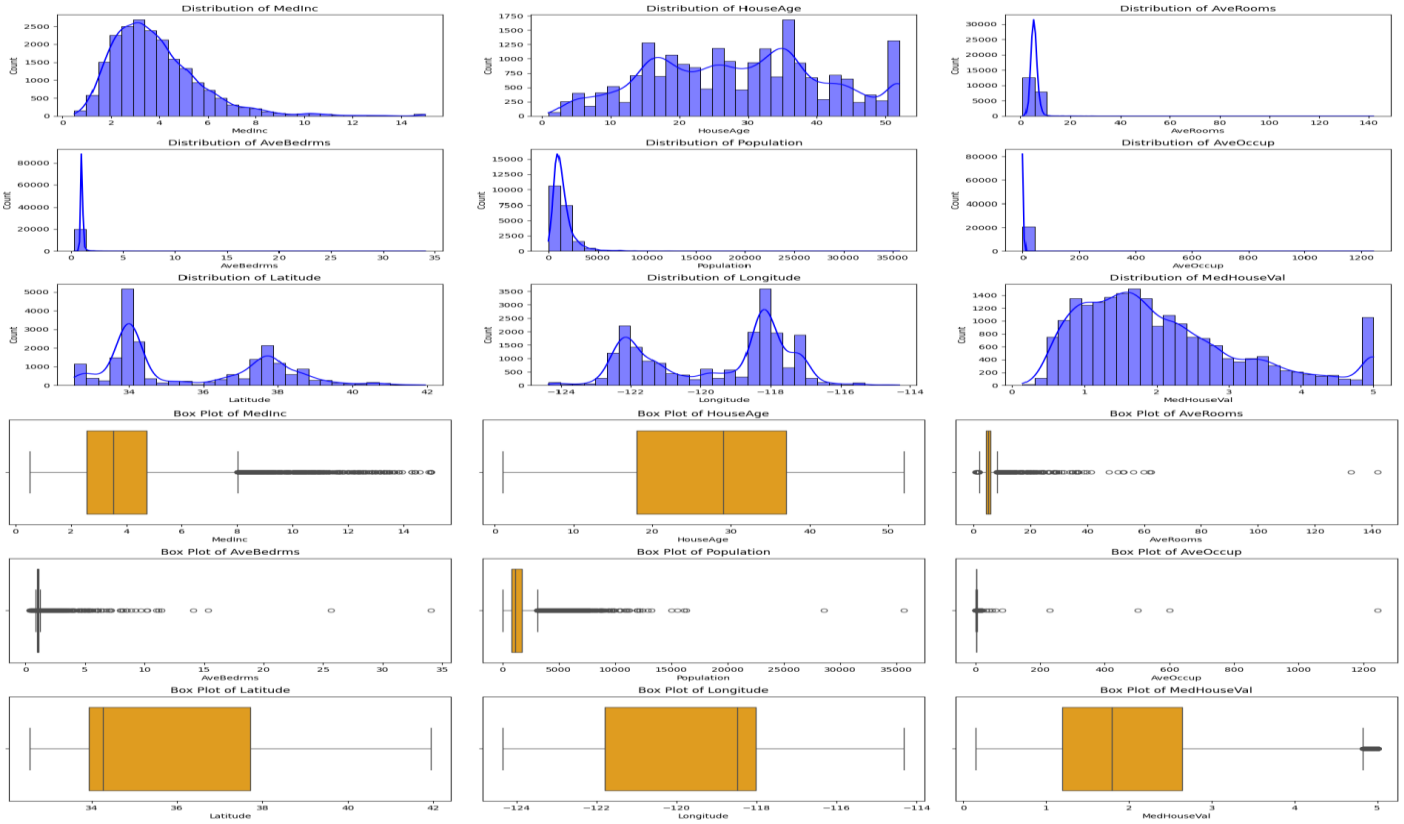
```
print("\nDataset Summary:")
print(housing_df.describe())
```

- **print("\nDataset Summary:")** Prints a message indicating that the dataset summary is being displayed.
- **print(housing_df.describe())** Displays a summary of the dataset using describe(). This provides statistics like the count, mean, standard deviation, minimum, and maximum values for each numerical feature.

Explanation:

This script performs exploratory data analysis (EDA) on the California Housing dataset. It first loads the dataset into a pandas Data Frame, then visualizes the distribution of numerical features using histograms and box plots. Histograms help understand how values are spread, while box plots highlight outliers. To further detect outliers, the script uses the Interquartile Range (IQR) method, identifying extreme values that may require attention. Finally, it prints a summary of the dataset, including statistical measures like mean, standard deviation, and quartiles, providing insights into the data's characteristics.

Output:




```
Outliers Detection:
MedInc: 681 outliers
HouseAge: 0 outliers
AveRooms: 511 outliers
AveBedrms: 1424 outliers
Population: 1196 outliers
AveOccup: 711 outliers
Latitude: 0 outliers
Longitude: 0 outliers
MedHouseVal: 1071 outliers

Dataset Summary:
             MedInc      HouseAge  ...     Longitude   MedHouseVal
count  20640.000000  20640.000000  ...  20640.000000  20640.000000
mean       3.870671     28.639486  ...   -119.569704      2.068558
std        1.899822     12.585558  ...      2.003532      1.153956
min        0.499900      1.000000  ...   -124.350000      0.149990
25%        2.563400     18.000000  ...   -121.800000      1.196000
50%        3.534800     29.000000  ...   -118.490000      1.797000
75%        4.743250     37.000000  ...   -118.010000      2.647250
max       15.000100     52.000000  ...   -114.310000      5.000010
```

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

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

- **import pandas as pd** Imports the pandas library and assigns it the alias pd. pandas is used for data manipulation and analysis (like loading and processing datasets).
- **import seaborn as sns** Imports the seaborn library (with the alias sns), which is a data visualization library built on top of matplotlib. It simplifies plotting and adds more advanced visualizations.
- **import matplotlib.pyplot as plt** Imports the matplotlib library for creating static, animated, and interactive plots in Python. It's the main tool used here to create plots.
- **from sklearn.datasets import fetch_california_housing** Imports the fetch_california_housing function from sklearn.datasets, which is used to load the California Housing dataset.

```
# Step 1: Load the California Housing Dataset
california_data = fetch_california_housing(as_frame=True)
data = california_data.frame
```

- **fetch_california_housing** function is used to retrieve the dataset, which contains information on housing prices and related features in California.
- **as_frame=True** This parameter returns the data as a Pandas DataFrame, which makes it easier to manipulate and analyze.
- **data** The dataset is stored in a variable named data, which contains the features (e.g., average house value, house age, population, etc.) and the target variable (i.e., the house value).

```
# Step 2: Compute the correlation matrix
correlation_matrix = data.corr()
```

- This calculates the correlation matrix for all the numerical features in the dataset using the **corr()** method from Pandas. The correlation matrix tells you how strongly each pair of features is related.
- Correlation values range from -1 (strong negative correlation) to +1 (strong positive correlation).
- A value of 0 indicates no correlation.
- **correlation_matrix** The result is a DataFrame where each value represents the correlation between two features.

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

- **plt.figure(figsize=(10, 8))** Specifies the size of the figure (10 by 8 inches).
- **sns.heatmap()** This function creates the heatmap.
- **annot=True** Annotates each cell with the correlation values.
- **cmap='coolwarm'** Specifies the color palette used for the heatmap, with warm colors indicating high correlation and cool colors indicating low correlation.
- **fmt='.2f'** Formats the correlation values to two decimal places.
- **linewidths=0.5** Adds a small line between the cells of the heatmap for better readability.
- **plt.title()** Adds a title to the plot.
- **plt.show()** Displays the plot.

```
# Step 4: Create a pair plot to visualize pairwise relationships
sns.pairplot(data, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pair Plot of California Housing Features', y=1.02)
plt.show()
```

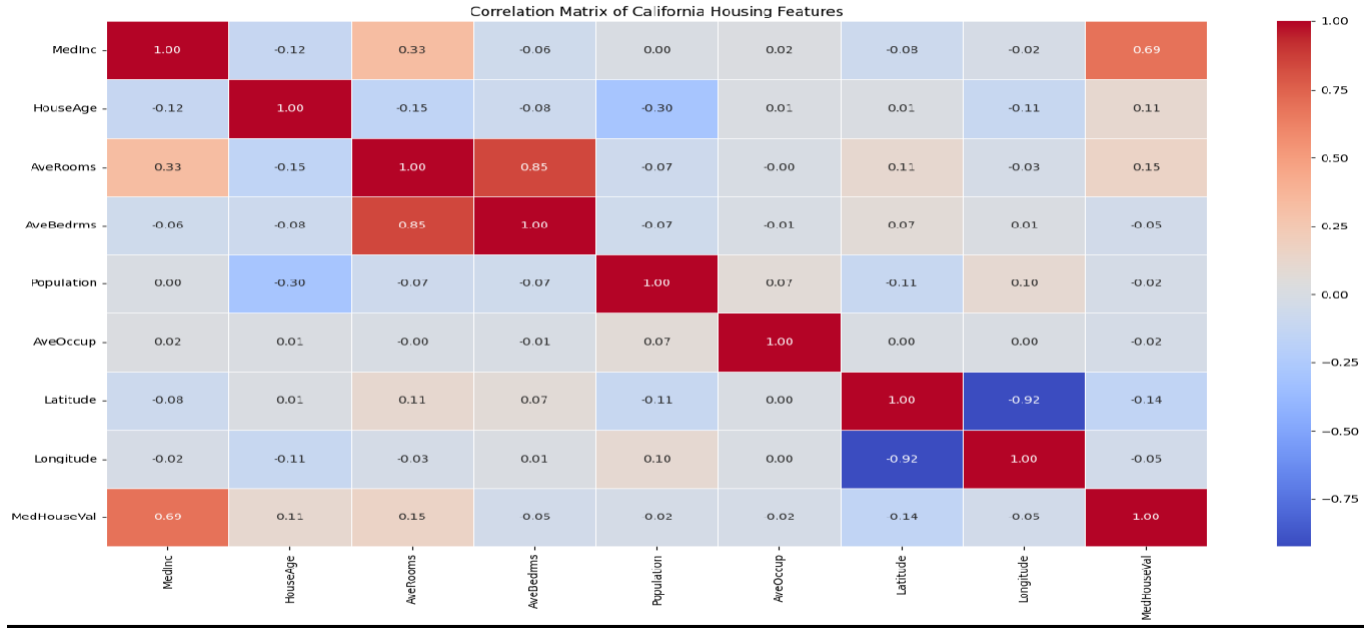
- **sns.pairplot()** This function generates a grid of scatter plots (or other types of plots) to show relationships between features.
- **diag_kind='kde'** Specifies that the diagonal should contain Kernel Density Estimation (KDE) plots to show the distribution of each feature.

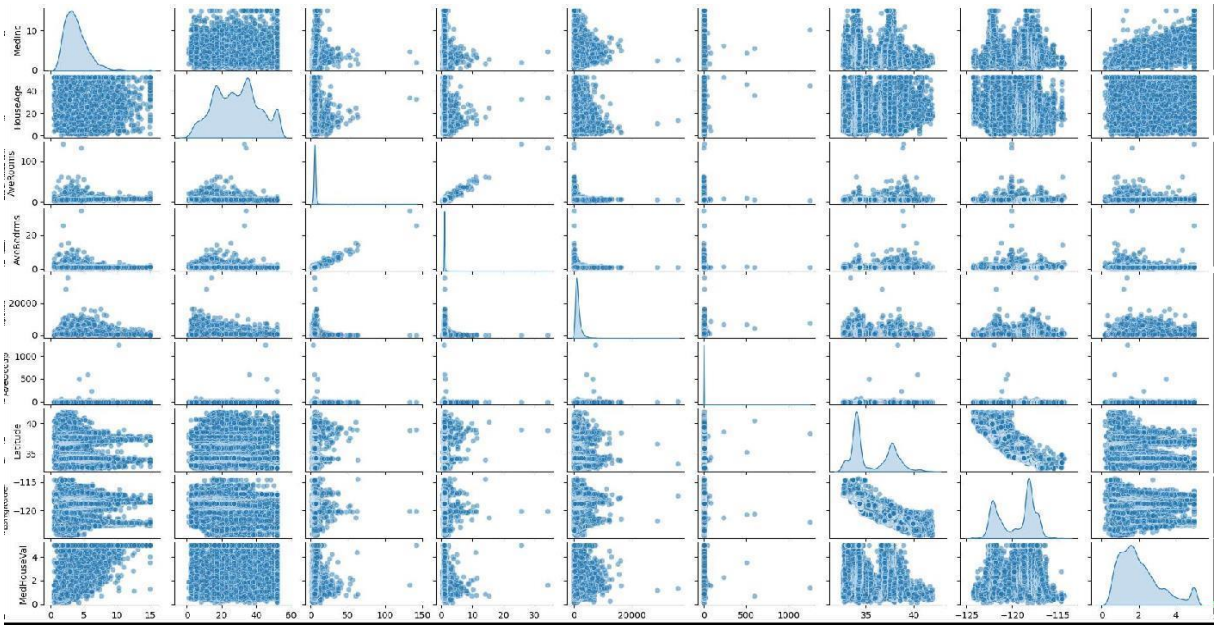
- **plot_kws={'alpha': 0.5}** Sets the transparency of the scatter plots to 50% (alpha=0.5) to make overlapping points more visible.
- **plt.suptitle()** Adds a title to the pair plot with a slight adjustment (y=1.02) to move it higher.
- **plt.show()** Displays the pair plot.

Explanation:

This script performs an exploratory data analysis on the California Housing dataset using Pandas, Seaborn, and Matplotlib. First, it loads the dataset, which contains various housing-related features, and converts it into a Data Frame. Then, it calculates the correlation matrix to measure relationships between numerical features. To visualize these relationships, a heatmap is created using Seaborn, displaying correlations with color gradients and numerical values for better interpretability. Additionally, a pair plot is generated to illustrate pairwise relationships between features, using scatter plots for bivariate distributions and KDE plots for univariate distributions. These visualizations help in understanding feature dependencies and potential patterns in the dataset.

Output:


Correlation Matrix of California Housing Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal
1.00 -0.12 0.33 -0.06 0.00 0.02 -0.08 -0.02 0.69
-0.12 1.00 -0.15 -0.08 -0.30 0.01 0.01 -0.11 0.11
0.33 -0.15 1.00 0.85 -0.07 -0.00 0.11 -0.03 0.15
-0.06 -0.08 0.85 1.00 -0.07 -0.01 0.07 0.01 -0.05
0.00 -0.30 -0.07 -0.07 1.00 0.07 -0.11 0.10 -0.02
0.02 0.01 -0.00 -0.01 0.07 1.00 0.00 0.00 -0.02
-0.08 0.01 0.11 0.07 -0.11 0.00 1.00 -0.92 -0.14
-0.02 -0.11 -0.03 0.01 0.10 0.00 -0.92 1.00 -0.05
0.69 0.11 0.15 0.05 0.02 0.02 0.14 0.05 1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

Program 3: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

- **numpy** This library is used for numerical operations, especially handling arrays and mathematical functions.
- **pandas** A powerful data manipulation and analysis library, often used for working with data in tabular form (like Excel sheets).
- **load_iris** A function from the sklearn.datasets module that loads the Iris dataset, a classic dataset for classification problems.
- **PCA** A technique from the sklearn.decomposition module for Principal Component Analysis, which reduces the dimensionality of data while retaining as much variance as possible.
- **matplotlib.pyplot** A plotting library used for creating static, interactive, and animated visualizations in Python.

```
# Load the Iris dataset
iris = load_iris()
data = iris.data
labels = iris.target
label_names = iris.target_names
```

- **iris** This loads the Iris dataset, which is a dataset containing 150 samples of iris flowers, with 4 features (sepal length, sepal width, petal length, petal width) for each sample.
- **data** The actual features of the dataset (4 features for each sample).
- **labels** The target labels (the species of each sample), which are numeric (0, 1, 2).
- **label_names** The corresponding names of the labels, which are the three species of Iris: Setosa, Versicolor, and Virginica.

```
# Convert to a DataFrame for better visualization
iris_df = pd.DataFrame(data, columns=iris.feature_names)
```

- This converts the dataset into a pandas DataFrame, which is a more convenient format for exploration and visualization. It labels the columns according to the names of the features (sepal length, sepal width, etc.).

```
# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(data)
```

- **PCA(n_components=2)** This initializes a PCA model with the goal of reducing the dataset's dimensionality to 2 principal components (i.e., 2D data).
- **pca.fit_transform(data)** This fits the PCA model to the data and transforms it into the reduced 2D space. data_reduced now contains the projection of the original data onto the two principal components.

```
# Create a DataFrame for the reduced data
reduced_df = pd.DataFrame(data_reduced, columns=['Principal Component 1', 'Principal
Component 2'])
reduced_df['Label'] = labels
```

- This converts the reduced data (which now has 2 principal components) into a DataFrame for easier handling and plotting.
- Label column is added to the DataFrame so we can color the data points according to their species when plotting.

```
# Plot the reduced data
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b']
for i, label in enumerate(np.unique(labels)):
      plt.scatter(reduced_df[reduced_df['Label'] == label]['Principal Component 1'],
      reduced_df[reduced_df['Label'] == label]['Principal Component 2'],
      label=label_names[label],color=colors[i])
plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
```

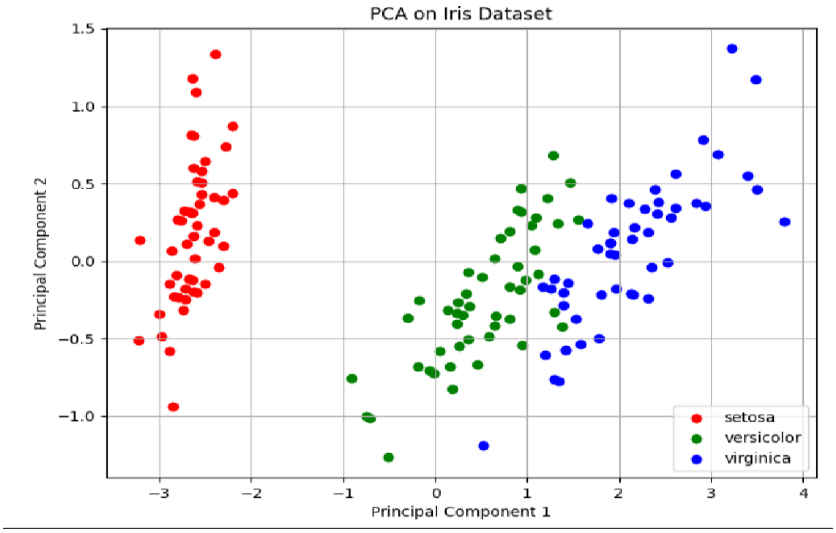
```
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

- **plt.figure(figsize=(8, 6))** Creates a figure with a specified size for the plot.
- colors: A list of colors (red, green, blue) that will be used to differentiate the species in the plot.
- **np.unique(labels)** This finds the unique species labels (0, 1, 2).
- **plt.scatter(...)** This creates a scatter plot for each species. For each unique label (species), the corresponding points in the 2D PCA space are plotted, with a color representing the species.
- **plt.title(), plt.xlabel(), plt.ylabel()** These functions add a title and labels to the plot.
- **plt.legend()** Displays the legend with species names (Setosa, Versicolor, Virginica).
- **plt.grid()** Adds a grid to the plot for better visualization.
- **plt.show()** Displays the plot.

Explanation:

This Python script demonstrates how to apply Principal Component Analysis (PCA) on the Iris dataset to reduce its dimensionality from four features to two, making it easier to visualize. The script first loads the Iris dataset using sklearn.datasets.load_iris(), extracts the feature data and labels, and converts it into a Pandas Data Frame for better readability. It then applies PCA with n_components=2 to transform the dataset into a two-dimensional space. The transformed data is stored in another Data Frame, with the corresponding class labels. Finally, the script uses matplotlib to create a scatter plot, where different species of flowers are represented in distinct colors. This visualization helps in understanding how well PCA separates the three species in lower-dimensional space.

Output:



Program 4: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
# pandas is used to read and process the dataset (CSV file).
import pandas as pd

def find_s_algorithm(file_path):
      data =  pd.read_csv(file_path)
      print("Training data:")
      print(data)
      attributes = data.columns[:-1] class_label =
      data.columns[-1]
      hypothesis = ['?' for _ in attributes]
```

- This function takes the file path of the training dataset as input.
- Reads the CSV file into a **pandas DataFrame**.
- Prints the training data.
- Assumes that the **last column** in the dataset is the **class label (Yes/No)**.
- All other columns are treated as **attributes**.
- The hypothesis is initialized as ['?', '?', ..., '?'], meaning it **accepts all values** initially.

```
      for index, row in data.iterrows():
            if row[class_label] == 'Yes':
                  for i, value in enumerate(row[attributes]):
                        if  hypothesis[i] == '?' or hypothesis[i] == value:
                              hypothesis[i] = value
                        else:
                              hypothesis[i] = '?'
                  return hypothes
```

- Loops through each row in the dataset.
- The algorithm ignores negative examples (No) and updates the hypothesis only for positive examples (Yes).
- For each attribute in the positive example:
  - If the current hypothesis is '?', assign it the value from the training example.
  - If the attribute value matches the hypothesis, keep it unchanged.
  - If the attribute value differs, generalize it to '?' (since it must work for all positive examples).
- After looping through all examples, the function returns the most specific hypothesis.

```
isfile_path = 'C:\\Users\\Admin\\Downloads\\training_data.csv'
hypothesis = find_s_algorithm(file_path)
print("\nThe final hypothesis is:", hypothesis)
```

- Calls the function with a CSV file as input.
- Prints the final hypothesis

Explanation:

The given Python script implements the Find-S algorithm using the pandas library for reading and processing a CSV dataset. The Find-S algorithm is a simple machine learning approach used to find the most specific hypothesis that fits all positive examples in a dataset. The script reads a CSV file containing training data, where the last column represents the class label (e.g., "Yes" or "No"). It initializes a hypothesis with the most general values ('?' for each attribute) and then iterates through the dataset, updating the hypothesis whenever a positive example ('Yes' in the class label) is encountered. If an attribute in a positive example matches the current hypothesis, it remains unchanged; otherwise, it is generalized to '?'. The final hypothesis is returned and printed, representing the most specific description that fits all positive examples.

Output:

```
Training data:
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3      Rain        Cold     High  False        Yes
4      Rain        Cold     High   True         No
5  Overcast         Hot     High   True        Yes
6     Sunny         Hot     High  False         No

The final hypothesis is: ['Overcast', 'Hot', 'High', '?']
```

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

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
```

- numpy (np) → Used for generating random numbers and handling arrays.
- matplotlib.pyplot (plt) → Used for visualizing classification results.
- Counter from collections → Used to count occurrences of labels and find the most common one.

```
# Creates an array of 100 random numbers between 0 and 1.
data = np.random.rand(100)

labels = ["Class1" if x <= 0.5 else "Class2" for x in data[:50]]
```

- The first 50 numbers (training data) are labeled based on a threshold of 0.5:
  - If x ≤ 0.5, it's labeled "Class1".
  - If x > 0.5, it's labeled "Class2".

```
# Computes the Euclidean distance between two 1D points x1 and x2.
def euclidean_distance(x1, x2):
      return abs(x1 - x2)

def knn_classifier(train_data, train_labels, test_point, k):
      distances = [(euclidean_distance(test_point, train_data[i]), train_labels[i])
for i in range(len(train_data))]
      distances.sort(key=lambda x: x[0])
      k_nearest_neighbors = distances[:k]
      k_nearest_labels = [label for _, label in k_nearest_neighbors]
      return Counter(k_nearest_labels).most_common(1)[0][0]
```

- Calculate distances between the test point and all training points.
- Sort the distances in ascending order.
- Select the k nearest neighbors.
- Count the labels of the k neighbors and return the most common label.

```
train_data = data[:50]
train_labels = labels
test_data = data[50:]
```

- The first 50 points are used for training.
- The last 50 points are used for testing.

```
#A list of different k values to experiment with.
k_values = [1, 2, 3, 4, 5, 20, 30]

print("--- k-Nearest Neighbors Classification ---")
print("Training dataset: First 50 points labeled based on the rule (x <= 0.5 -> Class1, x >
0.5 -> Class2)")
print("Testing dataset: Remaining 50 points to be classified\n")
results = {}
for k in k_values:
      print(f"Results for k = {k}:")
```

```
    classified_labels = [knn_classifier(train_data, train_labels, test_point, k) for
test_point in test_data]
    results[k] = classified_labels
    for i, label in enumerate(classified_labels, start=51):
        print(f"Point x{i} (value: {test_data[i - 51]:.4f}) is classified as
    {label}")
    print("\n")
```

- Loops through each k-value.
- Classifies each test point using the knn_classifier function.
- Stores the results in a dictionary results.
- Prints the classification results for each test point.

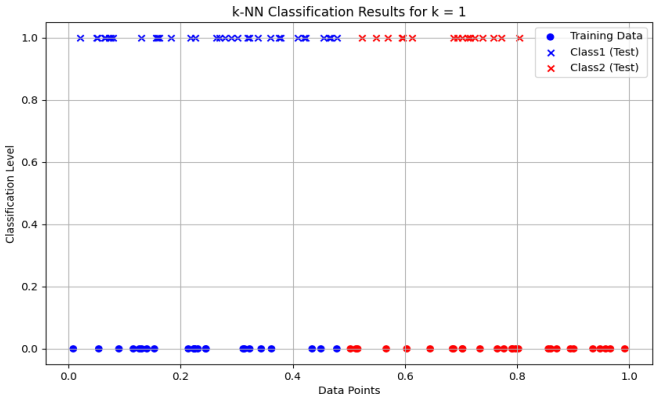
```
print("Classification complete.\n")

for k in k_values:
    classified_labels = results[k]
    class1_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] ==
"Class1"]
    class2_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] ==
"Class2"]
    plt.figure(figsize=(10, 6))
    plt.scatter(train_data, [0] * len(train_data), c=["blue" if label == "Class1" else
"red" for label in train_labels],label="Training Data", marker="o")
    plt.scatter(class1_points, [1] * len(class1_points), c="blue", label="Class1 (Test)",
marker="x")
    plt.scatter(class2_points, [1] * len(class2_points), c="red", label="Class2 (Test)",
marker="x")
    plt.title(f"k-NN Classification Results for k = {k}")
    plt.xlabel("Data Points")
    plt.ylabel("Classification Level")
    plt.legend()
    plt.grid(True)
    plt.show()
```

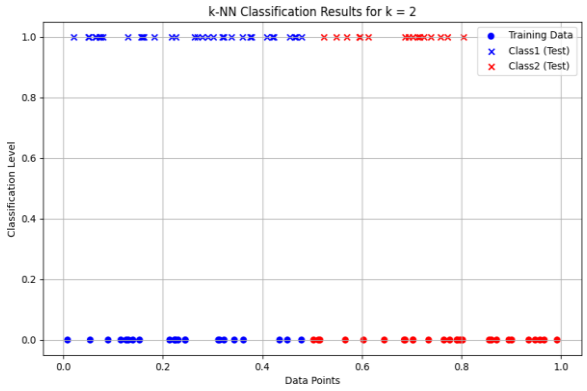
- Loops through different k-values.
- Extracts classified test points for each class.
- Plots a scatter plot:
- Training data is marked as o.
- Test data is marked as x:
- Blue x → Classified as Class1.
- Red x → Classified as Class2.
- Adds labels and a legend to the graph.
- Displays the plot.

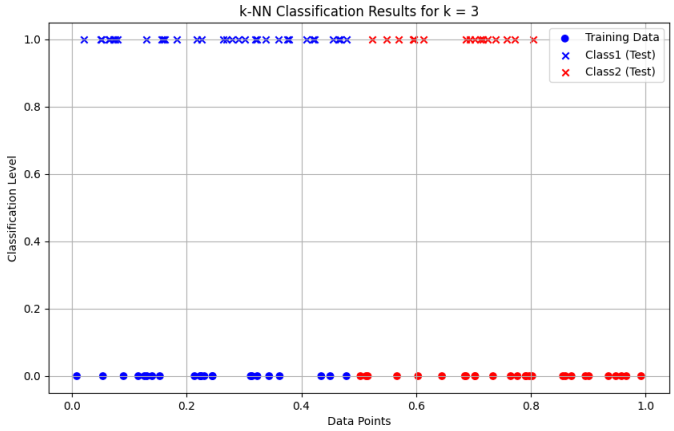
Explanation:

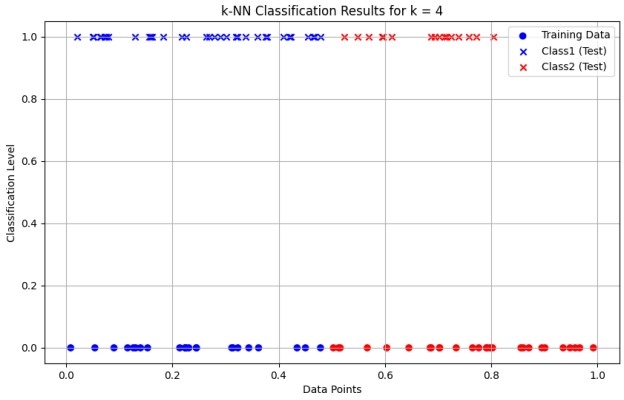
This Python script implements a simple k-Nearest Neighbors (k-NN) classifier using Euclidean distance for a 1D dataset generated randomly. It starts by creating 100 random values between 0 and 1, labeling the first 50 points as "Class1" if ≤ 0.5 and "Class2" otherwise. The script defines a function to calculate Euclidean distance and another to classify test points based on their nearest neighbors. It then iterates over different k-values (1, 2, 3, etc.), classifying the remaining 50 data points. The classification results are printed and visualized using Matplotlib, where training points are plotted as circles and test points as Class1 (blue crosses) or Class2 (red crosses). The visualizations help analyze how different k-values affect classification performance.

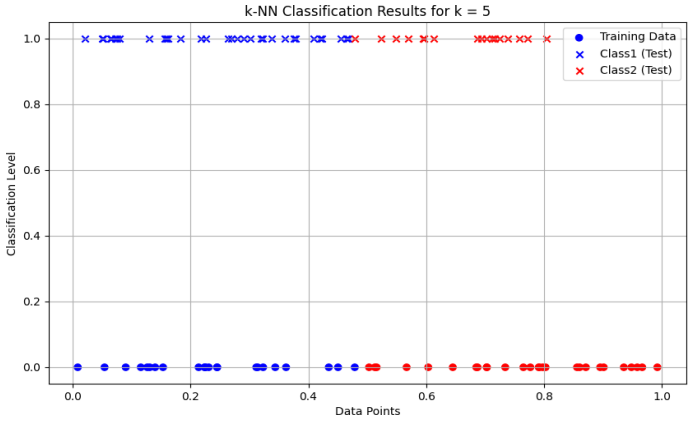
Output:

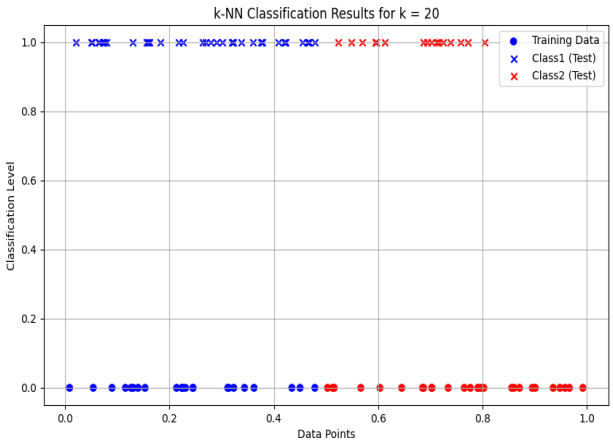


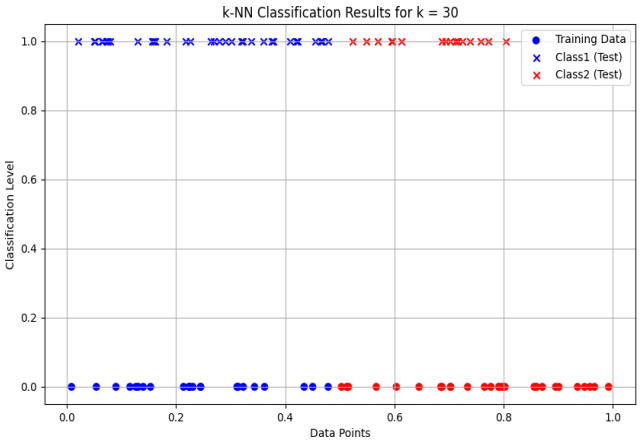



k-NN Classification Results for k = 3
Training Data
Class1 (Test)
Class2 (Test)
Classification Level
Data Points


k-NN Classification Results for k = 4
Training Data
Class1 (Test)
Class2 (Test)
Classification Level
Data Points


k-NN Classification Results for k = 5
Training Data
Class1 (Test)
Class2 (Test)
Classification Level
Data Points


k-NN Classification Results for k = 20
Training Data
Class1 (Test)
Class2 (Test)
Classification Level
Data Points


k-NN Classification Results for k = 30
Training Data
Class1 (Test)
Class2 (Test)
Classification Level
Data Points

--- k-Nearest Neighbors Classification ---
Training dataset: First 50 points labeled based on the rule (x <= 0.5 -> Class1, x > 0.5 -> Class2)
Testing dataset: Remaining 50 points to be classified

Results for k = 1:
Point x51 (value: 0.5702) is classified as Class2
Point x52 (value: 0.4654) is classified as Class1
Point x53 (value: 0.7016) is classified as Class2
Point x54 (value: 0.5964) is classified as Class2
Point x55 (value: 0.0643) is classified as Class1
Point x56 (value: 0.2698) is classified as Class1
Point x57 (value: 0.7124) is classified as Class2
Point x58 (value: 0.3219) is classified as Class1

Point x59 (value: 0.2637) is classified as Class1
Point x60 (value: 0.5483) is classified as Class2
Point x61 (value: 0.1561) is classified as Class1
Point x62 (value: 0.1592) is classified as Class1
Point x63 (value: 0.3752) is classified as Class1
Point x64 (value: 0.1299) is classified as Class1
Point x65 (value: 0.6934) is classified as Class2
Point x66 (value: 0.5240) is classified as Class2
Point x67 (value: 0.0203) is classified as Class1
Point x68 (value: 0.3789) is classified as Class1
Point x69 (value: 0.6866) is classified as Class2
Point x70 (value: 0.1834) is classified as Class1
Point x71 (value: 0.4197) is classified as Class1
Point x72 (value: 0.3608) is classified as Class1
Point x73 (value: 0.7579) is classified as Class2
Point x74 (value: 0.1624) is classified as Class1
Point x75 (value: 0.5943) is classified as Class2
Point x76 (value: 0.4097) is classified as Class1
Point x77 (value: 0.6124) is classified as Class2
Point x78 (value: 0.2794) is classified as Class1
Point x79 (value: 0.3193) is classified as Class1
Point x80 (value: 0.0503) is classified as Class1
Point x81 (value: 0.8038) is classified as Class2
Point x82 (value: 0.0792) is classified as Class1
Point x83 (value: 0.4230) is classified as Class1
Point x84 (value: 0.7250) is classified as Class2
Point x85 (value: 0.7162) is classified as Class2
Point x86 (value: 0.0725) is classified as Class1
Point x87 (value: 0.0752) is classified as Class1
Point x88 (value: 0.4676) is classified as Class1
Point x89 (value: 0.2256) is classified as Class1
Point x90 (value: 0.4552) is classified as Class1
Point x91 (value: 0.4787) is classified as Class1
Point x92 (value: 0.7390) is classified as Class2
Point x93 (value: 0.0649) is classified as Class1
Point x94 (value: 0.3373) is classified as Class1
Point x95 (value: 0.7719) is classified as Class2
Point x96 (value: 0.0512) is classified as Class1
Point x97 (value: 0.3012) is classified as Class1
Point x98 (value: 0.5966) is classified as Class2
Point x99 (value: 0.2897) is classified as Class1
Point x100 (value: 0.2176) is classified as Class1

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

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

- numpy is used for numerical computations, including matrix operations.
- matplotlib.pyplot is used for plotting the data and the locally weighted regression curve.

```
def gaussian_kernel(x, xi, tau):
    return np.exp(-np.sum((x - xi) ** 2) / (2 * tau ** 2))
```

- This function calculates the weight of a training point x_i based on how far it is from the query point x.
- The weight follows a Gaussian distribution:

$$w_i = \exp\left(-\frac{(x - x_i)^2}{2\tau^2}\right)$$

- τ\tauτ (tau) is the bandwidth parameter that controls how much influence neighboring points have. Smaller τ\tauτ means a narrower weighting function, focusing on fewer points.

```
def locally_weighted_regression(x, X, y, tau):
    m = X.shape[0]
    weights = np.array([gaussian_kernel(x, X[i], tau) for i in range(m)])
    W = np.diag(weights)
    X_transpose_W = X.T @ W
    theta = np.linalg.inv(X_transpose_W @ X) @ X_transpose_W @ y
    return x @ theta
```

- Compute the weights for each training example relative to the test point x.
- Construct a weight matrix W, where higher weight is assigned to training points close to x.
- Solve the weighted normal equation:

$$\theta = (X^T W X)^{-1} X^T W y$$

  This is similar to the ordinary least squares (OLS) formula but with weighting.
- Predict the output y for x using the computed θ.

```
np.random.seed(42)
X = np.linspace(0, 2 * np.pi, 100)
y = np.sin(X) + 0.1 * np.random.randn(100)
X_bias = np.c_[np.ones(X.shape), X]
```

- np.linspace(0, 2 * np.pi, 100): Creates 100 points evenly spaced between 0 and 2π.
- np.sin(X) + 0.1 * np.random.randn(100): Generates noisy sine wave data by adding random Gaussian noise.
- X_bias = np.c_[np.ones(X.shape), X]: Adds a bias term (a column of ones) to enable linear regression.

```
x_test = np.linspace(0, 2 * np.pi, 200)
x_test_bias = np.c_[np.ones(x_test.shape), x_test]
tau = 0.5
y_pred = np.array([locally_weighted_regression(xi, X_bias, y, tau) for xi in x_test_bias])
```

- Generates test points x_test from 0 to 2π (more points for a smoother curve).
- Adds the bias term for consistency with X_bias.
- Applies LWR for each test point, using tau = 0.5 as the bandwidth parameter.

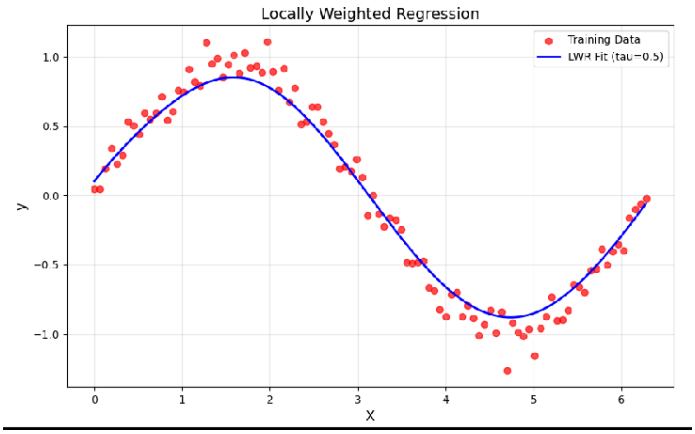
```
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', label='Training Data', alpha=0.7)
plt.plot(x_test, y_pred, color='blue', label=f'LWR Fit (tau={tau})', linewidth=2)
plt.xlabel('X', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Locally Weighted Regression', fontsize=14)
plt.legend(fontsize=10)
plt.grid(alpha=0.3)
plt.show()
```

- Plots training data (red scatter points).
- Plots LWR-fitted curve (blue line).
- Labels axis and adds a title.
- Enables a grid for better readability.

Explanation:

This program implements **Locally Weighted Regression (LWR)** using a Gaussian kernel to assign weights to training points based on their distance from a given query point. The function gaussian_kernel computes the weight for each training point relative to the query point using a Gaussian function with bandwidth tau. The locally_weighted_regression function then performs weighted linear regression by computing the weighted least squares estimate for the regression coefficients. The dataset consists of noisy sine wave samples, and LWR is applied to estimate the function's trend. The predictions are visualized, showing the fitted curve along with the training data, demonstrating how LWR captures local patterns in the data.

Output:



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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
```

- numpy and pandas: Used for numerical and data manipulation.
- matplotlib.pyplot: Used for visualization.
- sklearn.datasets: Fetches the California Housing Dataset.
- sklearn.model_selection.train_test_split: Splits data into training and testing sets.
- sklearn.linear_model.LinearRegression: Implements simple linear regression.
- sklearn.preprocessing.PolynomialFeatures, StandardScaler: Used for polynomial regression and feature scaling.
- sklearn.pipeline.make_pipeline: Creates a pipeline for polynomial regression.
- sklearn.metrics.mean_squared_error, r2_score: Evaluates model performance.

```
def linear_regression_california():
    housing = fetch_california_housing(as_frame=True)
    X = housing.data[["AveRooms"]]
    y = housing.target
```

- Fetches the California Housing dataset.
- Selects AveRooms (average number of rooms) as the only feature for prediction.
- y is the target variable: Median house price.

#80% of data is used for training, 20% for testing.

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

#A Linear Regression model is created and trained.

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

    y_pred = model.predict(X_test)

    plt.scatter(X_test, y_test, color="blue", label="Actual")
    plt.plot(X_test, y_pred, color="red", label="Predicted")
```

- Blue dots represent actual house prices.
- Red line represents the predicted trend.

```
    plt.xlabel("Average number of rooms (AveRooms)")
    plt.ylabel("Median value of homes ($100,000)")
    plt.title("Linear Regression - California Housing Dataset")
    plt.legend()
    plt.show()

    print("Linear Regression - California Housing Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))
```

- Mean Squared Error (MSE): Measures the average squared difference between actual and predicted values.
- R^2 Score: Measures how well the model explains the variability of the target variable (1 is best, 0 is worst).

```
def polynomial_regression_auto_mpg():
      url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-
mpg.data"
      column_names = ["mpg", "cylinders", "displacement", "horsepower", "weight",
"acceleration", "model_year", "origin"]
      data = pd.read_csv(url, sep='\s+', names=column_names, na_values="?")
      data = data.dropna()
```

- Downloads the Auto MPG dataset from UCI Machine Learning Repository.
- Renames columns for readability.
- Handles missing values by dropping rows with NaN values.

```
      X = data["displacement"].values.reshape(-1, 1)
      y = data["mpg"].values
```

- X: Displacement (size of engine, in cubic inches).
- y: Miles per gallon (mpg).

#80% training, 20% testing.

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

      poly_model = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(),
LinearRegression())
      poly_model.fit(X_train, y_train)
```

- **PolynomialFeatures(degree=2)** Converts X into polynomial features (quadratic terms).
- **StandardScaler()** Standardizes features for better performance.
- **LinearRegression()** Fits a polynomial model

```
      y_pred = poly_model.predict(X_test)

      plt.scatter(X_test, y_test, color="blue", label="Actual")
      plt.scatter(X_test, y_pred,   color="red", label="Predicted")
```

- Blue dots: Actual mpg values.
- Red dots: Predicted mpg values.

```
      plt.xlabel("Displacement")
      plt.ylabel("Miles per gallon (mpg)")
      plt.title("Polynomial Regression - Auto MPG Dataset")
      plt.legend()
      plt.show()

      print("Polynomial Regression - Auto MPG Dataset")
      print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
      print("R^2 Score:", r2_score(y_test, y_pred))
```

- MSE: Measures prediction error.
- R^2 Score: Measures model accuracy.

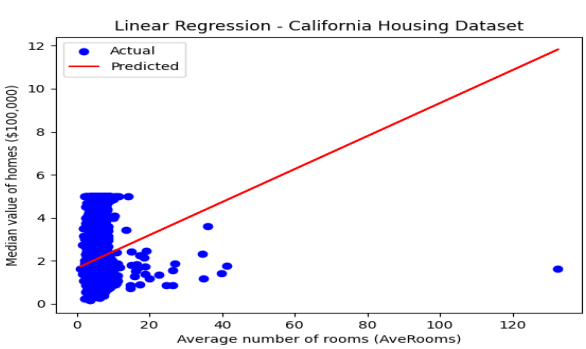
```
if __name__ == "__main__":
      print("Demonstrating Linear Regression and Polynomial Regression\n")
      linear_regression_california()
      polynomial_regression_auto_mpg()
```

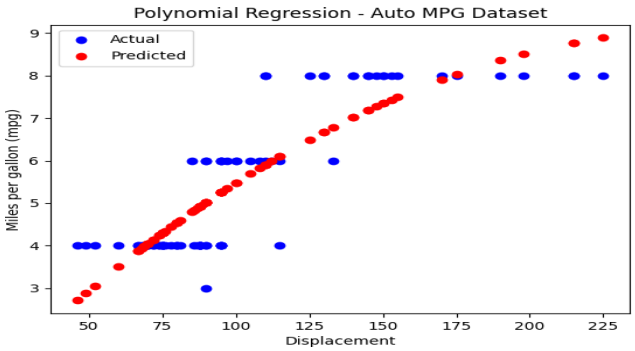
- Ensures the script runs only when executed directly.
- Calls both functions to demonstrate linear and polynomial regression.

Explanation:

The provided code demonstrates linear and polynomial regression using two datasets. First, it performs linear regression on the California Housing dataset, focusing on predicting the median home value based on the average number of rooms. The data is split into training and test sets, and a LinearRegression model is trained and evaluated, displaying a scatter plot comparing actual and predicted values along with the Mean Squared Error (MSE) and R^2 score. In the second part, polynomial regression is applied to the Auto MPG dataset, predicting fuel efficiency (mpg) based on engine displacement. A pipeline is used, consisting of polynomial feature transformation, standard scaling, and linear regression. The model's performance is visualized similarly, and metrics are printed. Both visualizations help in understanding model fit and performance.

Output:






```
Demonstrating Linear Regression and Polynomial Regression

Linear Regression - California Housing Dataset
Mean Squared Error: 1.2923314440807299
R^2 Score: 0.013795337532284901
Polynomial Regression - Auto MPG Dataset
Mean Squared Error: 0.743149055720586
R^2 Score: 0.7505650609469626
```

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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

- **numpy as np** Used for handling numerical data efficiently.
- **matplotlib.pyplot as plt** Used for plotting and visualizing the decision tree.
- **load_breast_cancer** Loads the breast cancer dataset from sklearn.datasets.
- **train_test_split** Splits the dataset into training and testing sets.
- **DecisionTreeClassifier** Implements a Decision Tree classifier.
- **accuracy_score** Measures the performance of the model.
- **Tree** Contains functions to visualize the decision tree.

```
data = load_breast_cancer()
X = data.data
y = data.target
```

- **data = load_breast_cancer()** Loads the dataset, which contains:
  - **data.data** The feature matrix (various attributes of the tumor).
  - **data.target** The target labels (0 = Malignant, 1 = Benign).
- **X = data.data** Assigns feature data to X.
- **y = data.target** Assigns target labels to y.

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

- **train_test_split** Splits data into training (80%) and testing (20%) subsets.
- **test_size=0.2** Allocates 20% of data for testing.
- **random_state=42** Ensures reproducibility (same split every time).

```
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

- **clf = DecisionTreeClassifier(random_state=42**) Creates a decision tree classifier.
- **clf.fit(X_train, y_train)** Trains the model using training data.

```
# Uses the trained classifier to predict labels for the test set.
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

- **accuracy_score(y_test, y_pred)** Computes the accuracy of predictions.
- The result is printed as a percentage.

```
new_sample = np.array([X_test[0]])
prediction = clf.predict(new_sample)
prediction_class = "Benign" if prediction == 1 else "Malignant"
print(f"Predicted Class for the new sample: {prediction_class}")
```

- Selects the first test sample.
- Predicts its class (Benign or Malignant).

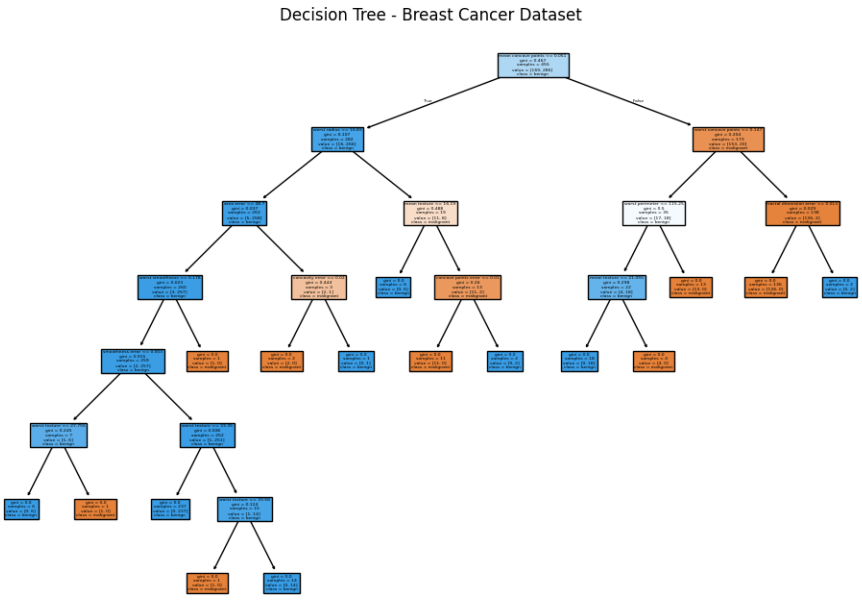
```
plt.figure(figsize=(12,8))
tree.plot_tree(clf,filled=True,feature_names=data.feature_names,class_names=data.target_name
s)
plt.title("Decision Tree - Breast Cancer Dataset")
plt.show()
```

- **plt.figure(figsize=(12,8))** Sets the figure size.
- **tree.plot_tree()** Plots the decision tree.
- **filled=True** Colors the nodes for better visualization.
- **feature_names=data.feature_names** Labels the features.
- **class_names=data.target_names** Labels the classes (Malignant, Benign).
- **plt.show()** Displays the plot.

Explanation:

This Python script implements a Decision Tree Classifier to classify breast cancer tumors as malignant or benign using the Breast Cancer dataset from sklearn.datasets. It begins by importing necessary libraries, then loads the dataset and separates it into features (X) and target labels (y). The dataset is split into training (80%) and testing (20%) subsets using train_test_split. A DecisionTreeClassifier model is created and trained on the training data. After training, predictions are made on the test set, and the model's accuracy is evaluated using accuracy_score. The script then predicts the class of a new sample (first test sample) and prints whether it is malignant or benign. Finally, it visualizes the decision tree using matplotlib and tree.plot_tree, displaying how the classifier makes decisions based on feature splits.

Output:



Program 9: Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

```
import numpy as np
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
```

- numpy → Used for numerical operations.
- fetch_olivetti_faces → Loads the Olivetti Faces dataset (grayscale 64×64 face images of 40 individuals).
- train_test_split → Splits the dataset into training and testing sets.
- cross_val_score → Evaluates the model using cross-validation.
- GaussianNB → Implements the Gaussian Naïve Bayes classifier.
- accuracy_score, classification_report, confusion_matrix → Evaluate the model.
- matplotlib.pyplot → Used for visualizing the results.

```
data = fetch_olivetti_faces(shuffle=True, random_state=42)
X = data.data
y = data.target
```

- **fetch_olivetti_faces()** loads the dataset.
- **shuffle=True** ensures the dataset is randomized.
- **random_state=42** ensures reproducibility.
- **X = data.data** contains images, where each 64×64 grayscale image is flattened into a 1D array of 4096 pixels.
- **y = data.target** contains the labels (person ID, ranging from 0 to 39).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- train_test_split() splits X and y into:
  - X_train, y_train → 70% training data.
  - X_test, y_test → 30% testing data.
- random_state=42 ensures consistency.

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

- Gaussian Naïve Bayes (GNB) assumes that features follow a normal distribution.
- **.fit(X_train, y_train)** trains the model using the training data.

```
# predict(X_test) generates predictions for the test set.
y_pred = gnb.predict(X_test)
```

accuracy_score(y_test, y_pred) computes the percentage of correctly classified images.

```
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy: {accuracy * 100:.2f}%')
print("\nClassification Report:")

print(classification_report(y_test, y_pred, zero_division=1))
```

- Classification report provides:
  - Precision (correct positive predictions out of all positive predictions).
  - Recall (correct positive predictions out of actual positives).
  - F1-score (harmonic mean of precision and recall).
    The **confusion matrix** shows how many images were classified correctly and where misclassifications occurred.

#The confusion matrix shows how many images were classified correctly and where misclassifications occurred.

```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

cross_val_accuracy = cross_val_score(gnb, X, y, cv=5, scoring='accuracy')
print(f'\nCross-validation accuracy: {cross_val_accuracy.mean() * 100:.2f}%')
```

- 5-fold cross-validation is performed to evaluate the model's robustness.
- The mean accuracy over 5 different splits is printed.

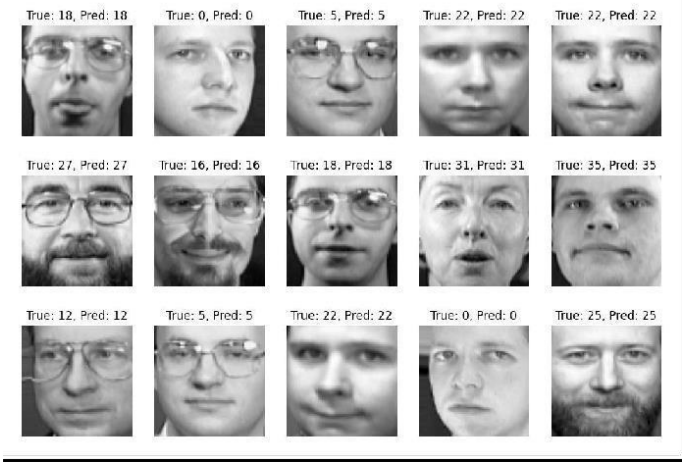
```
fig, axes = plt.subplots(3, 5, figsize=(12, 8))
for ax, image, label, prediction in zip(axes.ravel(), X_test, y_test, y_pred):
      ax.imshow(image.reshape(64, 64), cmap=plt.cm.gray)
      ax.set_title(f"True: {label}, Pred: {prediction}")
      ax.axis('off')
plt.show()
```

- Displays 15 images (3 rows × 5 columns).
- Each subplot shows:
  - The actual label (True: label).
  - The predicted label (Pred: prediction).
- **reshape(64, 64)** converts the 1D array back to a 2D image.
- **plt.show()** renders the visualization.

Explanation:

This Python script uses machine learning to classify human face images from the Olivetti Faces dataset. It begins by importing necessary libraries like NumPy, scikit-learn, and Matplotlib. The dataset is loaded using fetch_olivetti_faces(), which provides grayscale 64x64 face images and corresponding labels. The data is split into training and testing sets using train_test_split(). A Gaussian Naïve Bayes (GNB) classifier is then trained on the training set and used to predict labels for the test set. The model's performance is evaluated using accuracy_score, classification_report, and confusion_matrix. Additionally, cross-validation is performed to assess generalization. Finally, a visualization is created using Matplotlib, displaying a subset of test images with their true and predicted labels.

Output:



Accuracy: 80.83%

Classification Report:

	precision	recall	f1-score	support
0	0.67	1.00	0.80	2
1	1.00	1.00	1.00	2
2	0.33	0.67	0.44	3
3	1.00	0.00	0.00	5
4	1.00	0.50	0.67	4
5	1.00	1.00	1.00	2
7	1.00	0.75	0.86	4
8	1.00	0.67	0.80	3
9	1.00	0.75	0.86	4
10	1.00	1.00	1.00	3
11	1.00	1.00	1.00	1
12	0.40	1.00	0.57	4
13	1.00	0.80	0.89	5
14	1.00	0.40	0.57	5
15	0.67	1.00	0.80	2
16	1.00	0.67	0.80	3
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	3
19	0.67	1.00	0.80	2
20	1.00	1.00	1.00	3
21	1.00	0.67	0.80	3
22	1.00	0.60	0.75	5
23	1.00	0.75	0.86	4
24	1.00	1.00	1.00	3
25	1.00	0.75	0.86	4
26	1.00	1.00	1.00	2
27	1.00	1.00	1.00	5
28	0.50	1.00	0.67	2
29	1.00	1.00	1.00	2
30	1.00	1.00	1.00	2
31	1.00	0.75	0.86	4
32	1.00	1.00	1.00	2
34	0.25	1.00	0.40	1
35	1.00	1.00	1.00	5

36	1.00	1.00	1.00	3
37	1.00	1.00	1.00	1
38	1.00	0.75	0.86	4
39	0.50	1.00	0.67	5
accuracy			0.81	120
macro avg	0.89	0.85	0.83	120
weighted avg	0.91	0.81	0.81	120

Confusion Matrix:

```
[[2 0 0 ... 0 0 0]
 [0 2 0 ... 0 0 0]
 [0 0 2 ... 0 0 1]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 5]]
```

Cross-validation accuracy: 87.25%

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

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

- **numpy (np)** Used for numerical computations.
- **pandas (pd)** Used for handling tabular data.
- **matplotlib.pyplot (plt)** Used for visualization.
- **seaborn (sns)** Used for making attractive statistical plots.
- **sklearn.datasets** Loads the Breast Cancer dataset.
- **sklearn.cluster.KMeans** Implements K-Means clustering.
- **sklearn.preprocessing.StandardScaler** Standardizes features by removing the mean and scaling to unit variance.
- **sklearn.decomposition.PCA** Performs Principal Component Analysis (PCA) for dimensionality reduction.
- **sklearn.metrics** Computes evaluation metrics like confusion matrix and classification report.

```
data = load_breast_cancer()
X = data.data
y = data.target
```

- The Breast Cancer dataset is loaded.
- X: Features (e.g., mean radius, mean texture, etc.).
- y: Labels (0 = malignant, 1 = benign).

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Standardization ensures that all features contribute equally by scaling them to have mean 0 and standard deviation 1.
- This is important for K-Means, which is distance-based

```
kmeans = KMeans(n_clusters=2, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)
```

- n_clusters=2 We assume two clusters (malignant vs. benign).
- fit_predict Computes cluster assignments.

```
print("Confusion Matrix:")
print(confusion_matrix(y, y_kmeans))
print("\nClassification Report:")
print(classification_report(y, y_kmeans))
```

- Since K-Means is unsupervised, it doesn't have true labels.
- However, we compare the cluster assignments (y_kmeans) to actual labels (y).
- The confusion matrix and classification report show how well K-Means matches the true classification.

```
# PCA reduces the 30-dimensional data to 2 principal components for easy visualization.
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Stores the PCA-reduced features, cluster assignments, and true labels
df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df['Cluster'] = y_kmeans
df['True Label'] = y

# Scatter plot of clusters identified by K-Means.
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100,
edgecolor='black', alpha=0.7)
plt.title('K-Means Clustering of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="Cluster")
plt.show()
```

#Scatter plot of true cancer labels

```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='True Label', palette='coolwarm', s=100,
edgecolor='black', alpha=0.7)
plt.title('True Labels of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="True Label")
plt.show()
```

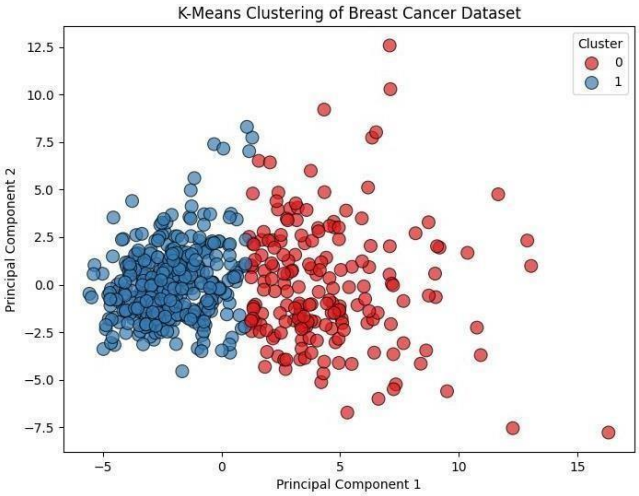
Adds centroids (red Xs) to show cluster centers.

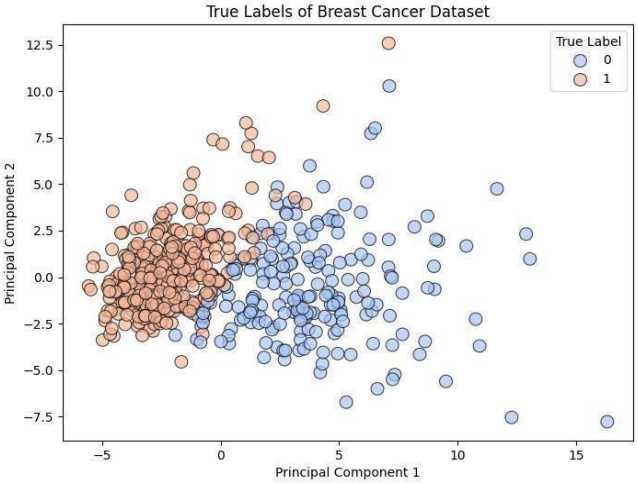
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100,
edgecolor='black', alpha=0.7)
centers = pca.transform(kmeans.cluster_centers_) plt.scatter(centers[:, 0], centers[:, 1],
s=200, c='red', marker='X', label='Centroids')
plt.title('K-Means Clustering with Centroids')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="Cluster")
plt.show()
```

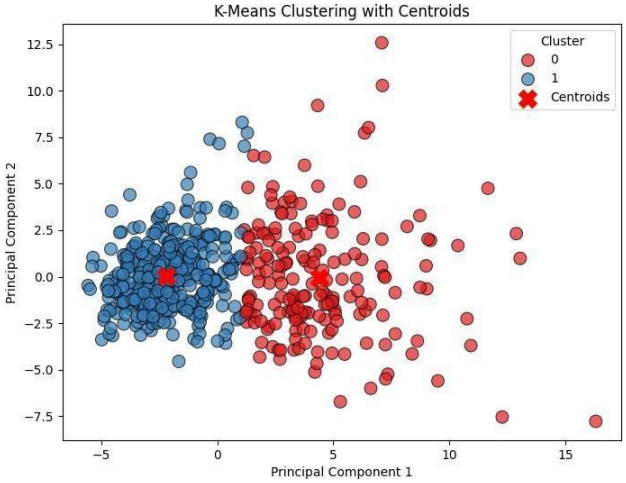
Explanation:

This code performs K-Means clustering on the Breast Cancer dataset from sklearn. It first loads the dataset and standardizes the features using StandardScaler to ensure all features have equal importance. Then, it applies K-Means clustering with two clusters, as the dataset has two target classes (malignant and benign). The clustering results are evaluated using a confusion matrix and a classification report. To visualize the data, Principal Component Analysis (PCA) is used to reduce the dimensions to two principal components, making it easier to plot. The results are displayed using scatter plots, comparing predicted clusters and true labels. Finally, the cluster centroids are plotted on the PCA-transformed data, helping to understand the separation of clusters.

Output:








```
Confusion Matrix:
[[175  37]
 [ 13 344]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.83      0.88       212
           1       0.90      0.96      0.93       357

    accuracy                           0.91       569
   macro avg       0.92      0.89      0.90       569
weighted avg       0.91      0.91      0.91       569
```

Viva Questions:

1. How does machine learning differ from traditional programming?
   In traditional programming, rules and logic are explicitly coded by developers. In machine learning, the system learns patterns and rules from data, allowing it to adapt to new inputs without being reprogrammed.

2. What is machine learning?
   Machine learning is a subset of artificial intelligence that involves the development of algorithms and statistical models that allow computers to learn and make predictions or decisions based on data.

3. Explain the concept of "training" in machine learning?
   Training in machine learning refers to the process of feeding data to a model so it can learn patterns and relationships in the data to make accurate predictions or decisions.

4. How does machine learning differ from artificial intelligence?
   Machine learning is a subset of artificial intelligence. AI encompasses a broader scope, including rule-based systems, robotics, and natural language processing, while machine learning specifically focuses on learning from data.

5. What are the main types of machine learning?
   Supervised Learning: Learning from labeled data.
   Unsupervised Learning: Learning from unlabeled data.
   Reinforcement Learning: Learning through interaction with an environment to maximize rewards.

6. Give an example of supervised learning.
   Predicting house prices based on features like size, location, and number of bedrooms using historical labeled data.

7. What is the difference between supervised and unsupervised learning?
   Supervised learning uses labeled data to train the model, while unsupervised learning works with unlabeled data to find hidden patterns or groupings.

8. How do you handle overfitting in a machine learning model?
   Using regularization techniques.
   Pruning decision trees.
   Increasing training data.
   Using cross-validation.

9. What are the steps involved in the machine learning process?
   Problem definition.
   Data collection.
   Data preprocessing (cleaning, normalization).
   Feature selection and engineering.
   Model selection and training.

Model evaluation.
Deployment and monitoring.

10. Why is data preprocessing important in machine learning?
    Data preprocessing ensures that the data is clean, consistent, and suitable for analysis, improving the model's accuracy and performance.

11. What are some real-world applications of machine learning?

    Healthcare: Disease diagnosis and drug discovery.
    Finance: Fraud detection and algorithmic trading.
    Retail: Recommendation systems and inventory management.
    Autonomous vehicles: Object detection and navigation.
    Natural language processing: Chatbots and language translation.

12. What are structured and unstructured data?
    Structured Data: Organized in a fixed format, like tables (e.g., relational databases).
    Unstructured Data: Lacks a predefined format (e.g., text, images, videos).

13. What is Big Data?
    Big Data refers to extremely large and complex datasets that cannot be processed using traditional methods due to their volume, velocity, and variety.

14. What are the 5 V's of Big Data?
    Volume: The amount of data.
    Velocity: The speed at which data is generated.
    Variety: Different types of data (structured, unstructured, semi-structured).
    Veracity: The quality or accuracy of the data.
    Value: The usefulness of the data.

15. What are the key components of a Big Data framework?
    Data Sources: Origin of data (e.g., sensors, social media).
    Data Storage: Systems like Hadoop HDFS, cloud storage.
    Data Processing: Tools like Spark, MapReduce.
    Data Analysis: Techniques like machine learning, statistical analysis.
    Visualization: Presenting results through tools like Tableau or Power BI.

16. What is Hadoop, and why is it important?
    Hadoop is an open-source framework for distributed storage and processing of Big Data. It allows scalability and fault tolerance.

17. What are descriptive statistics?
    Descriptive statistics summarizes and organizes data to describe its main features using measures like

mean, median, mode, and standard deviation.

18. What are the measures of central tendency?

Mean: The average value.
Median: The middle value in a sorted dataset.
Mode: The most frequently occurring value.

19. What are measures of dispersion?

Range: Difference between the highest and lowest values.
Variance: The average squared deviation from the mean.
Standard Deviation: The square root of variance.

20. What is univariate data analysis?

Univariate data analysis involves analyzing a single variable to understand its distribution, central tendency, and variability.

21. What are some common methods of univariate data visualization?

Bar Charts: For categorical data.
Histograms: For continuous data.
Box Plots: To show distribution and detect outliers.
Pie Charts: To represent proportions.

22. How do you identify outliers in univariate data?

Box Plot: Data points outside the whiskers.
Z-Score: Data points with a Z-score > ±3.
IQR Method: Values below Q1 - 1.5×IQR or above Q3 + 1.5×IQR.

23. What is bivariate data?

Data involving two variables, often analyzed to find relationships or correlations between them.

24. What is multivariate data?

Data involving three or more variables, analyzed to study complex relationships.

25. Give an example of bivariate and multivariate data.

Bivariate: Height vs. weight of individuals.
Multivariate: Height, weight, age, and income of individuals.

26. What are common techniques in multivariate statistics?

PCA (Principal Component Analysis), Factor Analysis, MANOVA (Multivariate Analysis of Variance), and Cluster Analysis.

27. What is feature engineering?
    The process of selecting, modifying, or creating features to improve the performance of a model.

28. What is dimensionality reduction?
    Reducing the number of features in a dataset while preserving as much information as possible.

29. Name common dimensionality reduction techniques.
    PCA, t-SNE, LDA (Linear Discriminant Analysis), and autoencoders.

30. What is nearest-neighbor learning?
    A non-parametric method where predictions are based on the closest data points in the feature space.

31. What is the K-Nearest-Neighbor algorithm?
    A classification or regression algorithm that considers the 'k' closest data points to make predictions.

32. What is weighted KNN?
    A variant of KNN where closer neighbors are given more weight in predictions.

33. What is a nearest centroid classifier?
    A classification method where each class is represented by the mean of its data points.

34. What is Locally Weighted Regression (LWR)?
    A regression method that assigns weights to data points based on their distance from the query point.

35. What is regression?
    A statistical method for modeling the relationship between a dependent variable and one or more independent variables.

36. Differentiate between linear and logistic regression.
    Linear regression predicts continuous values, while logistic regression predicts probabilities for classification.

37. What is multiple linear regression?
    A regression model with multiple independent variables.

38. What is polynomial regression?
    A regression model that fits a polynomial equation to the data.

39. What is a decision tree?
    A tree-like model used for classification and regression tasks, where decisions are made at nodes based on feature values.

40. What is decision tree induction?
The process of building a decision tree from training data.

41. What is probability-based learning?
A method of learning based on probabilistic models, such as Bayes' theorem.

42. What is Bayes' theorem?
A formula that describes the probability of an event based on prior knowledge of related events.

43. What is the Naïve Bayes algorithm?
A probabilistic classifier based on Bayes' theorem, assuming independence between features.

44. What is an artificial neuron?
A computational model inspired by biological neurons, consisting of weights, bias, and an activation function.

45. What are the types of ANNs?
Feedforward, convolutional, recurrent, and generative adversarial networks.

46. What is clustering?
An unsupervised learning method to group similar data points.

47. What is hierarchical clustering?
A method that builds a hierarchy of clusters using agglomerative or divisive approaches.

48. What is density-based clustering?
A method that groups data points based on density, such as DBSCAN.

49. What is reinforcement learning?
A learning paradigm where agents learn by interacting with an environment to maximize cumulative rewards.

50. What is a Markov Decision Process (MDP)?
A mathematical framework for modeling decision-making in reinforcement learning.

51. What is Q-Learning?
A model-free RL algorithm that learns a value function to estimate the quality of actions.

52. What is SARSA?
An on-policy RL algorithm that updates its Q-values based on the current state-action pair and the next state-action pair.