PIMPRI CHINCHWAD EDUCATION TRUST'S

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

(An ISO 21001:2018 Certified institute) Accredited with 'A' Grade by NAAC
Sector No.26, Pradhikaran, Nigdi, Pune-411044

PRACTICAL / TERM WORK JOURNAL DEPT.

OF COMPUTER ENGINEERING

Name: Satyendra Kumar Sah

Year and course: SY Comp 2024-25 **Semester-III**

Division/Branch: D / Comp Engineering **PRN/Roll no.-123B1B259**

Subject: Data Exploration and Visualization

Academic Year: 2024-25

PIMPRI CHINCHWAD EDUCATION TRUST'S

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

Sector No. 26, Pradhikaran, Nigdi, Pune - 411044

Department Of Computer Engineering

Year and Course 2nd year (2024-2025) / Computer Engineering Roll No. : 123B1B259

INDEX

Sr. No.	Title	Page No.	Date	Signature	Remarks
1 (a)	Create a dictionary to store		5/08/024	CM	
	student names and their Corresp-		6/08/024		
	onding scores in a test. write function				
	to add new students name, update				
	scores, delete students and find the				
	student with the highest score.				
(b)	Using python perform list operations		12/08/24		
	and list slicing in detail with				
	necessary programs.				
(c)	perform matrix addition subtraction		20/08/024		
	and multiplication using Numpy				
	library.				
2 (a)	write a python program to create		27/08/024		
	a dataframe by importing csv,		02/09/24		
	JSON files and perform various operation.				
(b)	perform web scraping for any		03/09/024		
	website (using scarpy / beautiful		10/09/024		
	soap / selenium)				

This is certify that Shri. Satyendra Kumar Sah has carried out the above mentioned Data Exploration and Visualization Practicals / Term Work in the Computer Engineering Department of Pimpri Chinchwad College of Engineering, Pune-44.

Date : 15-11-2024

Subject Incharge CM

Head of Department

Pimpri Chinchwad College of Engineering, Pune - 44

PIMPRI CHINCHWAD EDUCATION TRUST'S

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

Sector No. 26, Pradhikaran, Nigdi, Pune - 411044

Department Of Computer Engineering

Year and Course 2nd year (2024-2025) / Computer Engg Roll No. : 123B1B259

INDEX

Sr. No.	Title	Page No.	Date	Signature	Remarks
(C)	write a python program to		23/09/024		
	import any dataset from UCI		24/09/024		
	Kaggle, perform data cleaning				
	and remove the outliers.				
3-(a)	Download or Create any dataset		30/09/024		
	and perform Exploratory data		01/10/24		
	Analysis for understanding the				
	data and finding the hidden pattern				
	and perform the following tasks.				
	• print summary statistics and write				
	the observation if any.				
	• Univariate analysis and write				
	the observation if any (Eg dataset:				
	automobile/car dataset, Loan prediction)				
(b)	Data Visualization with matplotlib		07/10/024		
	and seaborn. Dataset: use iris dataset		08/10/024		
	perform the following task:				
	• Data Loading and summary • Visualization with matplotlib				
	• visualization with seaborn.				

This is certify that Shri. Satyendra Kumar Sah has carried out the above mentioned Data Exploration and visualization Practicals / Term Work in the Computer Engineering Department of Pimpri Chinchwad College of Engineering, Pune-44.

Date: 15-11-2024

Subject Incharge

Head of Department

Pimpri Chinchwad College of Engineering, Pune - 44

PIMPRI CHINCHWAD EDUCATION TRUST'S

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

Sector No. 26, Pradhikaran, Nigdi, Pune - 411044

Department Of Computer Engineering

Year and Course 2nd year (2024-2025) / Comp. Engineering Roll No. : 123B1B259

INDEX

Sr. No.	Title	Page No.	Date	Signature	Remarks
(c)	Exploratory data Analysis using any dataset from UCI/kaggle.		14/10/024		
	and perform the following tasks:		15/10/024		
	• Data Loading and summary statistics.				
	• Data Distribution Analysis				
	• Bivariate/multivariate analysis				
	• Correlation analysis and write observation.				
4(a)	Download suitable Housing data		21/10/24		
	from Kaggle/UCI/any other source.		22/10/024		
	predict Houseprice using linear				
	regression and Evaluate the model				
	using RMSE or any other suitable				
	Metric. Example Dataset: (https://www.kaggle.com/datasets/yasserh/housing-prices-dataset) (Note: Use				
	Sci-kit learn library)				
(b)	Download Diabetic data from		28/10/024		
	Kaggle/UCI/any other source.		29/10/024		
	Build classification model for				
	diabetic prediction and Evaluate				

This is certify that Shri. Satyendra Kumar Sah has carried out the above mentioned Data Exploration and Visualization Practicals / Term Work in the Computer Engineering Department of Pimpri Chinchwad College of Engineering, Pune-44.

Date : 15-11-2024

Subject Incharge

Head of Department

Pimpri Chinchwad College of Engineering, Pune - 44

PIMPRI CHINCHWAD EDUCATION TRUST'S

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

Sector No. 26, Pradhikaran, Nigdi, Pune - 411044

Department Of Computer Engineering

Year and Course 2nd year (2024-2025) / Comp. Engg. Roll No. : 123B1B259

INDEX

Sr. No.	Title	Page No.	Date	Signature	Remarks
	the model using accuracy, precision, Recall etc., -- Example Dataset: ~~kaggle - pima~~ (https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.) (Note:- Use Sci-Kit learn Library)				
(C)	Hypothesis test: Assume our business has two units that make pizzas. Check if there is any significant difference in the average diameter of pizzas between the two making units.		11/11/024		
(5)	Mini project		15/11/024		

This is certify that Shri. Satyendra Kumar sah has carried out the above mentioned Data Exploration and Visualization Practicals / Term Work in the Computer Engineering Department of Pimpri Chinchwad College of Engineering, Pune-44.

Date : 15-11-2024

Subject Incharge

Head of Department
Pimpri Chinchwad College of Engineering, Pune - 44

Assignment No:1 A

Title:

Create a dictionary to store student names and their corresponding scores in a test. Write functions to add new students, update scores, delete students, and find the student with the highest score.

Aim:

The aim of this assignment is to create a Python dictionary to manage student scores. You will write functions to perform various operations such as adding new students, updating scores, deleting students, and finding the student with the highest score.

Theory:

A dictionary in Python is a collection of key-value pairs where each key is unique and maps to a value. Dictionaries are versatile and can be used to store various types of data. For this assignment, the dictionary will store student names as keys and their scores as values.

Algorithm:

1. Start
2. Declare a dictionary
3. For adding a student call add function by passing parameters.
4. To update mark of the student
    a) Check if name in the dictionary then
    b) Add the student mark to the dictionary
    c) Otherwise print does not exist.
5. To delete the student name check if name in dictionary then delete the name, otherwise it does not exist.
6. To find highest mark in the dictionary
    a) Check if not record in the dictionary
    b) Then print no any record in the dictionary
    c) Otherwise call max function and assign to the variable
    d) Return the mark and name
7. In the main function, passing value to each function and calling them individually.
8. Stop

Mathematical Background:

The operations involved in this assignment primarily use basic dictionary operations and traversal techniques. Finding the maximum value in a dictionary can be done using linear search, which has a time complexity of O(n)O(n)O(n), where nnn is the number of students.

Code:

[1]:
```
student_scores = {}

def add_student(name, score):
    """Add a new student with their score."""
    student_scores[name] = score
    print(f"Added student {name} with score {score}.")

def update_score(name, score):
    """Update the score for an existing student."""
    if name in student_scores:
        student_scores[name] = score
        print(f"Updated {name}'s score to {score}.")
    else:
        print(f"Student {name} does not exist. Use add_student to add them first.")

def delete_student(name):
    """Delete a student's score from the dictionary."""
    if name in student_scores:
        del student_scores[name]
        print(f"Deleted student {name}.")
    else:
        print(f"Student {name} does not exist.")

def topper():
    """Return the student with the highest score."""
    if not student_scores:
        print("No students in the list.")
        return None
    top_student = max(student_scores, key=student_scores.get)
    top_score = student_scores[top_student]
    print(f"Top student is {top_student} with a score of {top_score}.")
    return top_student, top_score

# Example Usage
add_student("Satyendra", 85)
add_student("Sidhu", 92)
update_score("Satyendra", 90)
delete_student("Sidhu")
topper()
```

```
Added student Satyendra with score 85.
Added student Sidhu with score 92.
Updated Satyendra's score to 90.
Deleted student Sidhu.
Top student is Satyendra with a score of 90.
```

[1]:
```
('Satyendra', 90)
```

Conclusion:

This program allows managing a list of students and their scores efficiently. It provides basic functionalities like adding, updating, and deleting students, as well as identifying the top-performing student based on their score. The dictionary data structure ensures that all operations, such as updating or deleting, are performed efficiently in constant time.

Assignment: 1 B

Title:

Using python perform List Operations and list slicing in detail with necessary programs.

Aim:

To understand and implement list operations and list slicing in Python, using various methods and techniques. This includes performing tasks such as adding, removing, and modifying elements within a list, as well as extracting subsets of lists using slicing.

Theory:

Lists are one of the most versatile data structures in Python, capable of storing an ordered collection of items that can be of different data types. Lists are mutable, meaning their content can be changed after creation. Python provides a wide range of built-in functions and methods to perform operations on lists.

Algorithm:

1- Create a list with some initial elements.

2- Perform basic list operations:

- A- Add an element at the end using append().
- B- Insert an element at a specific position using insert().
- C- Remove an element using remove().
- D- Remove an element from a specific position using pop().
- E- Modify an element by accessing it via its index.

3- Perform list slicing:

- A- Slice the list to get a specific range of elements.
- B- Slice the list to get elements from the beginning to a specific position.
- C- Slice the list to get elements from a specific position to the end.
- D- Slice the list with a step value to skip elements.

4- Print the list after each operation to observe the changes

Mathematical Background:

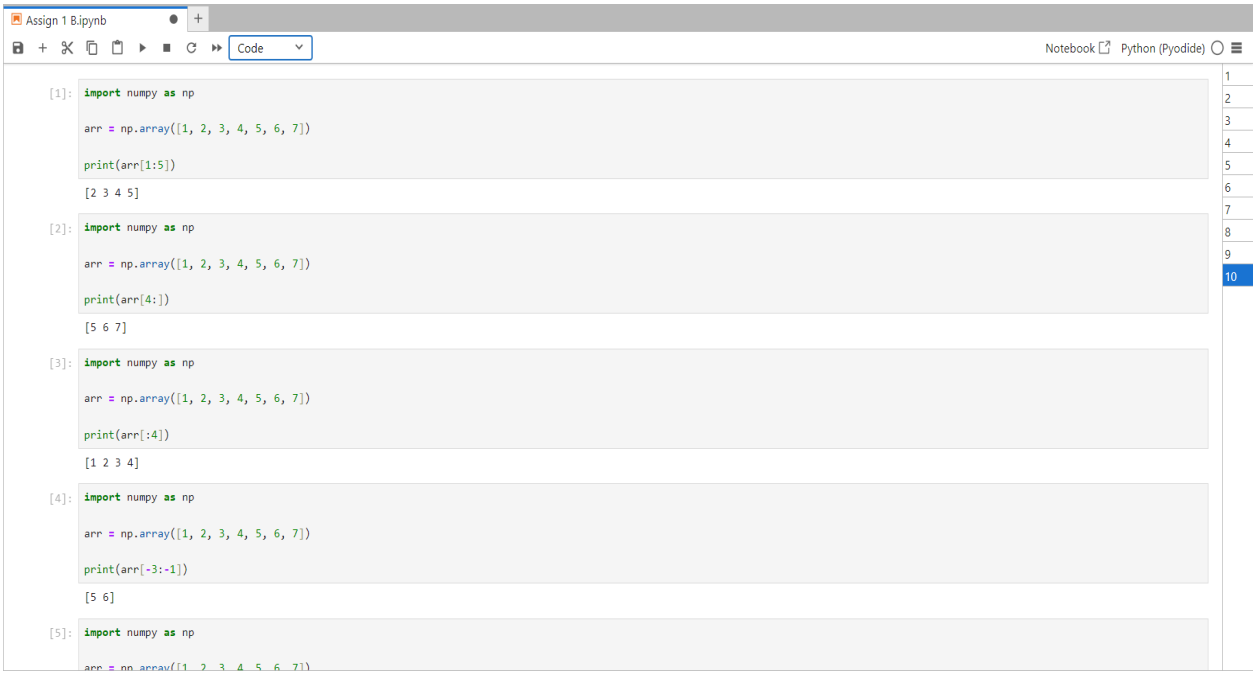
A- Understanding indexing in Python (starting from 0).

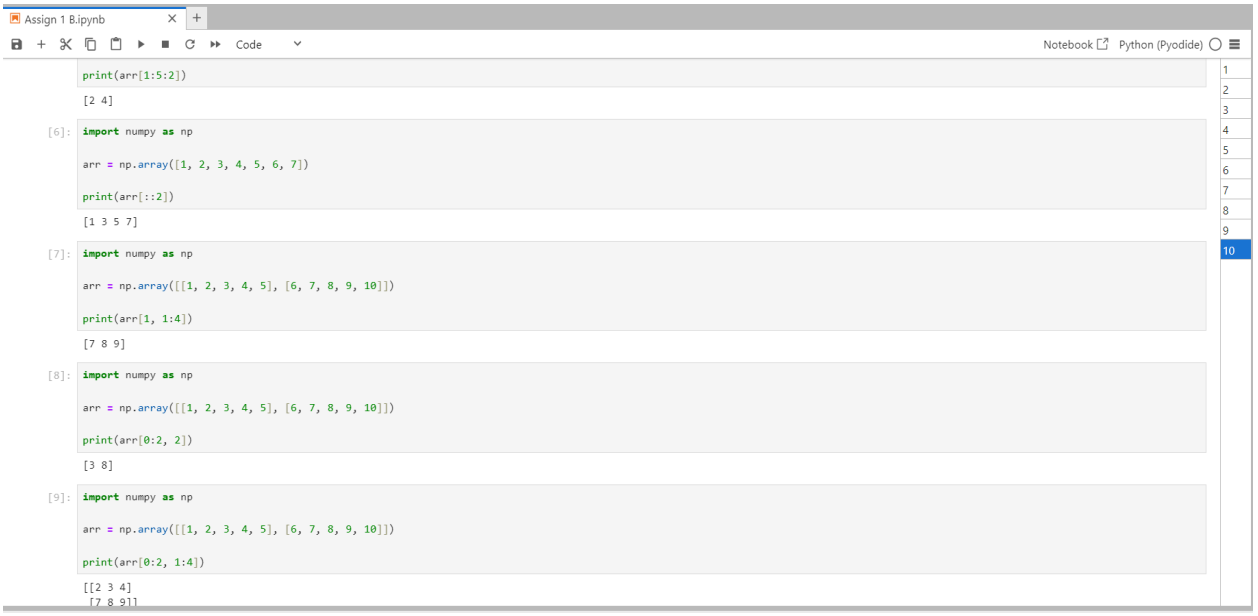
B- Understanding range and step values in slicing.

DataSet:

For this assignment, no external dataset is required. We will create sample lists within the program to demonstrate various operations and slicing techniques.

Code:





Conclusion: In this assignment, we explored how to perform basic list operations and list slicing in Python. These concepts are fundamental when dealing with list data structures and have wide-ranging applications in programming tasks.

Assignment: 1 C

Title:

Perform matrix addition subtraction and multiplication using NumPy library

Aim:

To perform matrix addition, subtraction, and multiplication using the NumPy library in Python, and to understand the mathematical operations and their computational implementation.

Theory:

A matrix is a two-dimensional array of numbers arranged in rows and columns. Operations like addition, subtraction, and multiplication are fundamental in linear algebra and have wide applications in fields like physics, computer graphics, machine learning, and more.

Algorithm:

1. Take two matrices A and B of the same dimensions.
2. Initialize the matrices A and B.
3. Add corresponding elements of A and B.
4. Store the result in matrix C.
5. Subtract corresponding elements of B from A.
6. Store the result in matrix C.
7. Be sure that the number of columns in A equals the number of rows in B.
8. Perform dot product between rows of A and columns of B.
9. Store the result in matrix C.

Mathematical Background:

For Addition:

A= [aij] and B= [bij] of size m*n, the result C= A+B is computed as:

$$C_{ij} = a_{ij} + b_{ij}$$

For Subtraction:

A= [aij] and B= [bij] of size m*n, the result C= A-B is computed as:

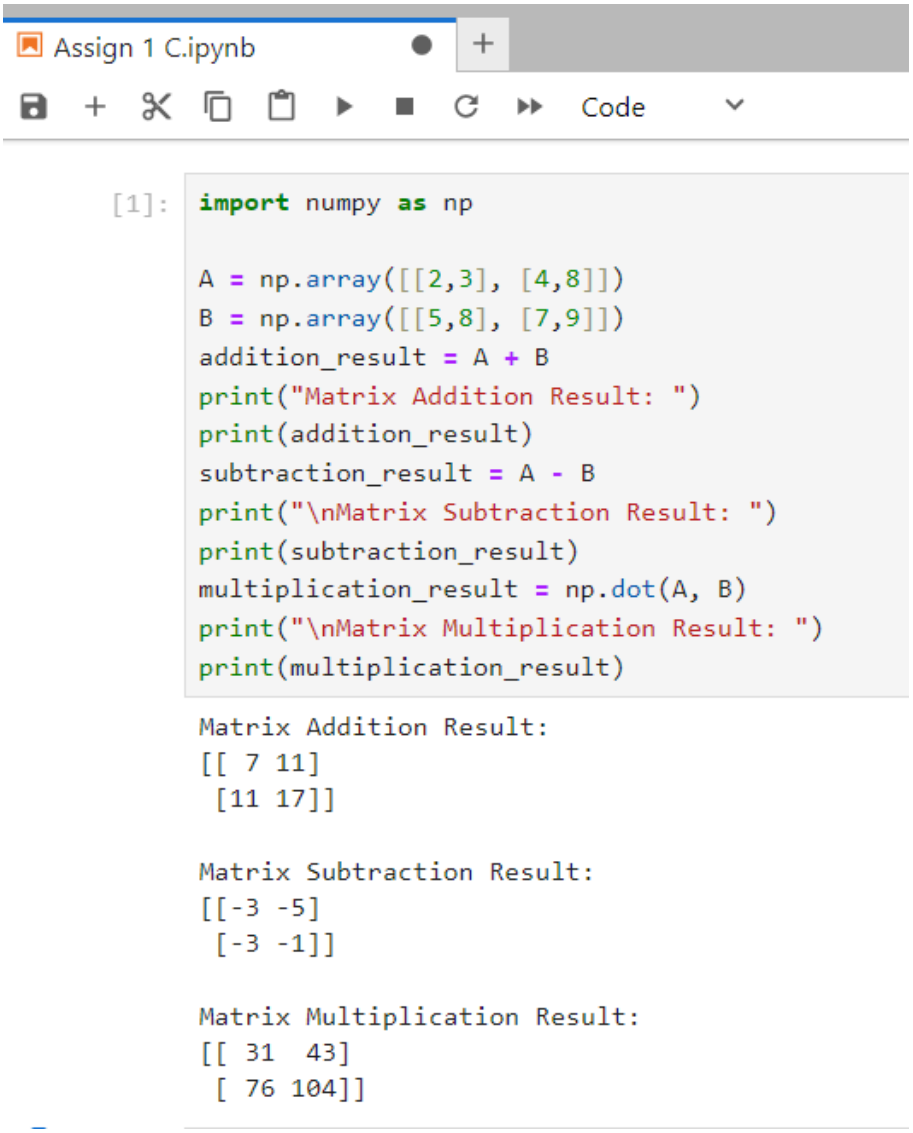
Cij= aij - bij

For Multiplication:

Given matrices A of size m*n and B of size n*p, the product C= A*B is:

cij= k=1∑n (aik×bkj)

Code:



Conclusion: The operations of matrix addition, subtraction, and multiplication have been successfully implemented using the NumPy library. These operations are essential in various scientific computations, and NumPy provides an efficient way to handle large-scale matrix manipulations.

Assignment: 2 A

Title:

Write a Python program to create a dataframe by importing CSV, JSON files and performing various operations.

Aim:

The aim of this assignment is to write a Python program to create a DataFrame by importing data from CSV and JSON files. The program will then perform various operations on the DataFrame, such as data manipulation, filtering, sorting, aggregation, and data visualization.

Theory:

DataFrames are one of the most commonly used data structures for data manipulation and analysis, provided by the pandas library in Python. A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).

The importation of CSV (Comma-Separated Values) and JSON (JavaScript Object Notation) files into a DataFrame is a crucial step in handling structured data. CSV files store tabular data in plain text, where each line of the file represents a data record. JSON, on the other hand, stores data in a human-readable format that consists of key-value pairs, making it useful for exchanging data between servers and web applications.

Algorithm:

Step 1: Install and import necessary libraries.
Step 2: Read data from a CSV file.
Step 3: Read data from a JSON file.
Step 4: Display the DataFrames.
Step 5: Perform basic data operations on the DataFrame.

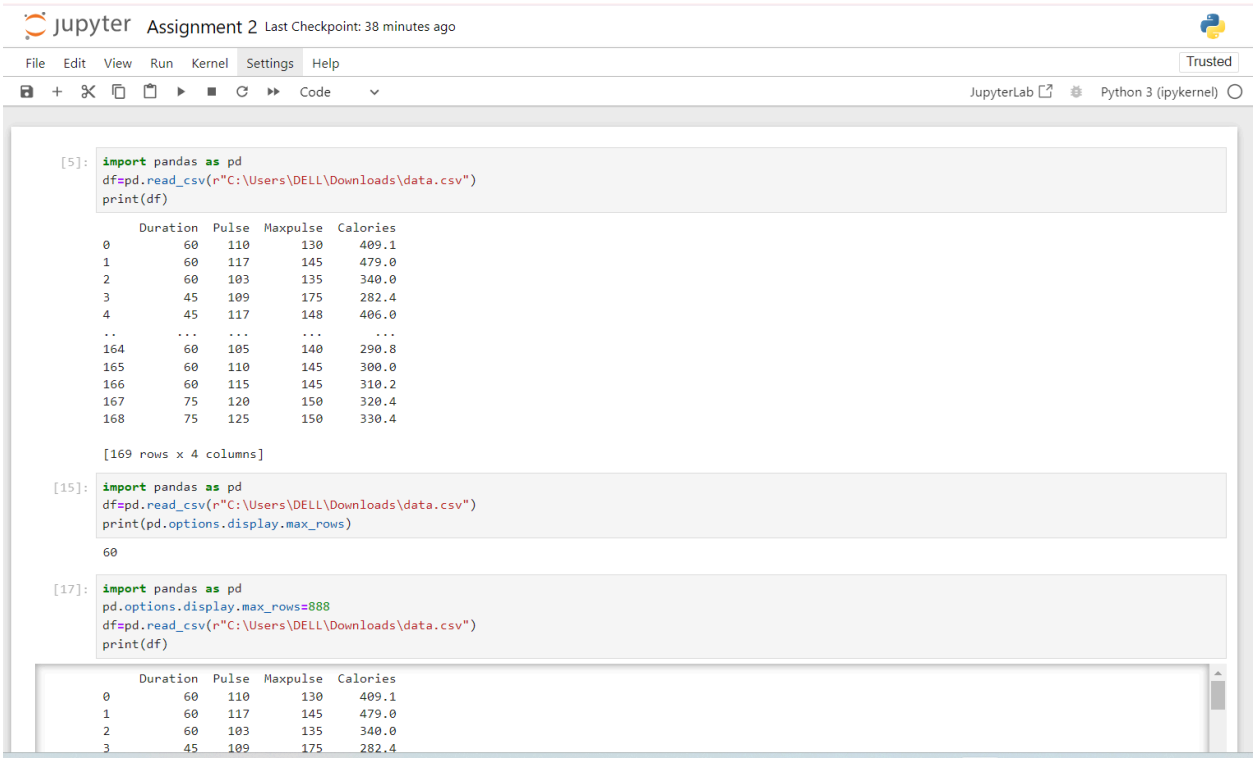
Mathematical Background:

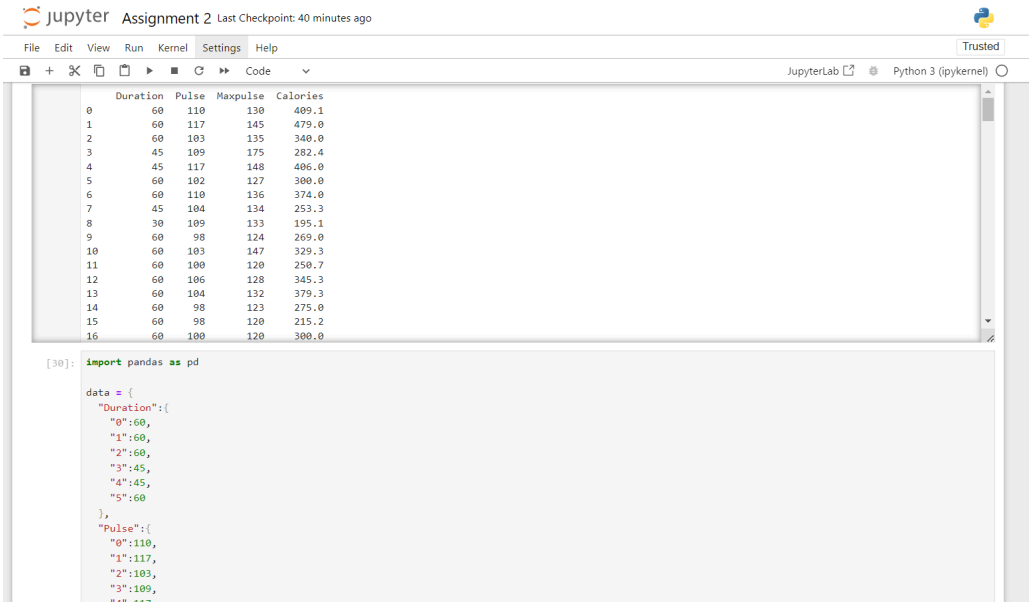
DataFrames are similar to matrices in mathematics, where data is organized into rows and columns. Many operations performed on DataFrames (e.g., addition, multiplication) are similar to matrix operations. For instance, calculating the mean, variance, and sum are basic mathematical functions applied column-wise or row-wise in the context of a DataFrame.

Dataset:

Data uploaded on w3 schools website as data.csv,data.json.

Code & Output





jupyter Assignment 2 Last Checkpoint: 40 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130       409
1        60    117       145       479
2        60    103       135       340
3        45    109       175       282
4        45    117       148       406
5        60    102       127       300
```

jupyter Assignment 2 Last Checkpoint: 41 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[32]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
print(df.head(10))
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.0
6        60    110       136     374.0
7        45    104       134     253.3
8        30    109       133     195.1
9        60     98       124     269.0
```

[34]:
```
print(df.tail())
```

```
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

[36]:
```
print (df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

Assignment: 2 B

Title:

Perform web scraping for any website (using scrapy/beautifulsoup/selenium)

Aim:

To develop a Python program that performs web scraping to extract data from a website using web scraping libraries such as **Scrapy**, **BeautifulSoup**, or **Selenium**. The extracted data will be processed and stored for analysis or future use.

Theory:

Web scraping is the automated process of extracting data from websites. It is used in data mining, information retrieval, and analysis. Web scraping allows programmers to collect large datasets from the internet, which can then be structured and stored in formats like CSV, JSON, or databases.

Python offers several libraries to perform web scraping, including:

1. **BeautifulSoup**: A Python library used to parse HTML and XML documents. It creates a parse tree that can be used to extract data from HTML tags.
2. **Scrapy**: An open-source and collaborative web crawling framework for extracting the data structured using XPath or CSS selectors.
3. **Selenium**: A web automation framework that allows you to simulate user interaction with web pages. It is useful for scraping dynamic pages that use JavaScript to load data.

Algorithm:

1. Choose a Target Website: Identify a website from which to scrape data.
2. Inspect the Web Page: Use browser developer tools to inspect the HTML structure and identify the data to be extracted.
3. Set Up the Environment:
   A-Install necessary libraries (Scrapy, Beautiful Soup, or Selenium).
   B-Set up a new project.
4. Write the Scraping Code:
   A-For Scrapy: Define spiders and parse methods.
   B-For Beautiful Soup: Use requests to fetch the web page and parse it.
   C-For Selenium: Use WebDriver to automate browser actions and scrape the data.
5. Run The Scraper: Execute the scraping script to collect the data.

6. Store the Data: Save the extracted data in a structured format (CSV, JSON, or database).

7. Handle Errors: Implement error handling for common issues (e.g., network errors, changes in HTML structure).

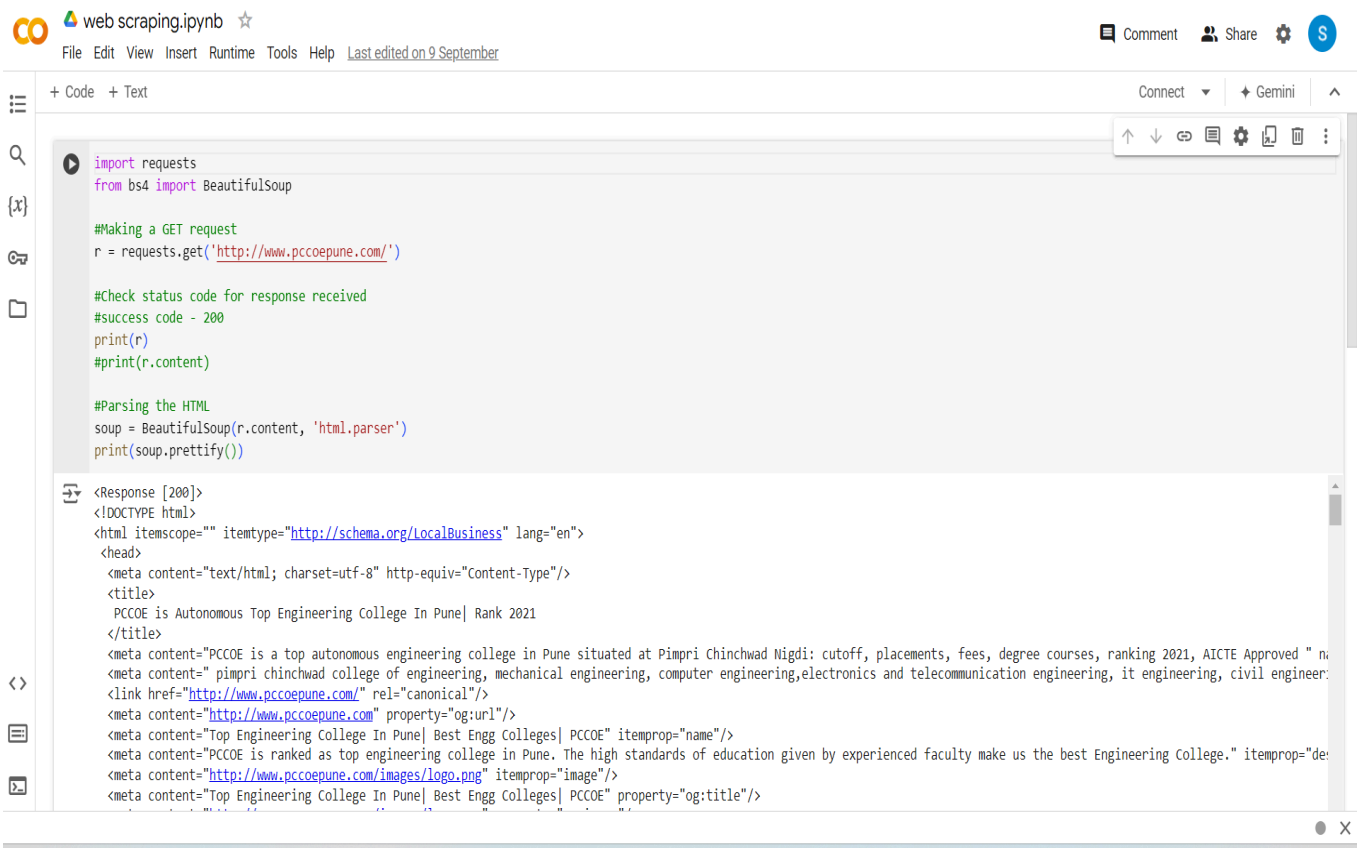
Mathematical Background:

Web scraping mainly involves string manipulation and HTML parsing rather than heavy mathematics.

Dataset:

For this assignment, a suitable website (e.g., an e-commerce site or a news website) can be used as the target for web scraping.

Code and Output:



Assignment: 2 C

Title:

Write a Python program to import any dataset from UCI/Kaggle, perform data cleaning and remove the outliers.

Aim:

The aim of this assignment is to develop a Python program that imports a dataset from UCI Machine Learning Repository or Kaggle, performs data cleaning to handle missing values and duplicates, and removes outliers using statistical methods.

Theory:

1. Data Importing: Data importing is the process of reading datasets from external sources into a Python environment. Kaggle and UCI are popular repositories for machine learning datasets.

2. Data Cleaning: Data cleaning is a critical preprocessing step to ensure data quality. It involves handling missing values, eliminating duplicates, and transforming data into a format suitable for analysis. Techniques include:

- Handling missing data using methods such as imputation (mean, median, mode) or dropping records.
- Removing duplicates to avoid redundant data.
- Converting data types when necessary (e.g., from strings to numerics).

3. Outlier Detection and Removal: Outliers are data points that deviate significantly from the other observations in the dataset. They can distort statistical analysis and machine learning model performance. Common methods for detecting outliers include:

- **Z-score**: Measures how many standard deviations a data point is from the mean. A Z-score greater than 3 or less than -3 is often considered an outlier.
- **IQR (Interquartile Range)**: This method uses quartiles to measure the spread of data. Outliers are considered those points lying below $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$

Algorithm:

Step 1: Import Libraries

- Import required libraries such as pandas, numpy, matplotlib, and seaborn for data handling and visualization.

Step 2: Load Dataset

- Load a dataset from either UCI or Kaggle. If the dataset is from Kaggle, use the Kaggle API or manually download the CSV file.

Step 3: Data Exploration

- Explore the dataset by checking its structure (e.g., columns, data types, missing values, duplicates).

Step 4: Data Cleaning

- Handling Missing Values: Use imputation techniques (mean/median/mode) or remove records with missing data.

Step 5: Outlier Detection and Removal

- Use statistical techniques to detect outliers, such as the Z-score method or IQR method.

Step 6: Verify Data Cleaning

- After removing outliers, check the data for consistency and correctness by visualizing it using box plots, histograms, etc.

Step 7: Save and Export Cleaned Data

- Save the cleaned dataset to a new CSV file for future use.

Dataset

For this assignment, you can choose a website relevant to your interests. Some suggested websites include:A-W3 schools.

Code & Output:

jupyter Assignment 2 C Last Checkpoint: 18 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[1]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
new_df = df.dropna()
print(new_df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
```

[3]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
df.dropna(inplace = True)
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
```

jupyter Assignment 2 C Last Checkpoint: 19 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[9]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
df.fillna(300, inplace = True)
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
```

[7]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
df["Calories"].fillna(130, inplace = True)
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
```

jupyter Assignment 2 C Last Checkpoint: 20 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
```

[11]:

```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
x = df["Calories"].median()
df["Calories"].fillna(x, inplace = True)
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
```

jupyter Assignment 2 C Last Checkpoint: 21 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
15         60     98       120     215.2
16         60    100       120     300.0
```

[13]:

```
import pandas as pd
df = pd.read_csv(r"C:\Users\DELL\Downloads\data.csv")
x = df["Calories"].mode()[0]
df["Calories"].fillna(x, inplace = True)
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.0
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
```

Assignment: 3 A

Title:

Download or create any dataset and perform Exploratory data analysis for understanding the data and finding the hidden patterns and perform the following tasks

- Print summary statistics and write the observations if any
- Univariate analysis and write the observations if any

(Example datasets: automobile/ car dataset , Loan prediction data set or any other)

Aim:

To perform Exploratory Data Analysis on a Car Details Dataset to understand the underlying structure of the data, identify patterns, and summarize key insights through descriptive statistics and visualization techniques.

Theory:

Exploratory Data Analysis is a crucial step in the data analysis process. It focuses on summarizing the main characteristics of the data, often using visual methods. This process helps to uncover hidden patterns, spot anomalies, test assumptions, and determine relationships within the dataset. It involves:

1. Summary Statistics: Descriptive statistics such as mean, median, variance, etc., provide an overview of the distribution and central tendency of numerical data.

2. Univariate Analysis: Examining individual variables to understand their distribution, central tendency, and spread.

3. Visualizations: Visual tools like histograms, pie charts, bar plots, etc., help in understanding the distribution of variables and the relationships between them.

Algorithm:

1. Data Loading: The first step involves loading the dataset into a pandas DataFrame from a CSV file.

2. Data Cleaning: Handling missing values and invalid data entries.

3. Summary Statistics: Using descriptive statistics like mean, median, standard deviation,etc., to summarize the dataset.

4. Univariate Analysis: Plotting histograms and pie charts to understand the distribution of individual variables.

5. Observations: Based on the analysis, we extract key insights and patterns from the data.

Mathematical Background:

1. Mean: The average value of a dataset.

2. Median: The middle value of a dataset when sorted in ascending order.

3. Standard Deviation: Measures the amount of variation or dispersion in a set of values.

4. Variance: Measures the spread of the data around the mean.

Data Visualizations: Visual representation of data distributions and relationships using histograms, pie charts, etc.

Dataset:

https://www.kaggle.com/datasets/akshaydattatraykhare/car-details-dataset

Observations:

Summary Statistics:

● **Selling Price**: The dataset showcases a broad range in selling prices, highlighting the diversity in car conditions and types. The mean selling price is lower than the maximum, indicating a right-skewed distribution where a small portion of cars are significantly more expensive than the average.

● **Kilometers Driven (km_driven)**: The average number of kilometers driven by the cars in the dataset is approximately 70,000 km. This suggests that the cars listed are mostly moderately used, with some outliers showing higher usage (well over 100,000 km) and a few with minimal usage.

● **Year**: The cars in the dataset vary in age, with most vehicles being between 5 to 15

years old. The age range indicates that the dataset predominantly consists of used cars from various years of manufacture, but newer cars tend to be more common.

Univariate Analysis:

● **Year**: The majority of the cars in the dataset were manufactured between 2005 and 2015. This indicates that the dataset has a focus on relatively newer used cars, with the number of cars peaking in more recent years, which could be attributed to a higher resale demand for these models.

● **Selling Price**: Most of the selling prices are concentrated at the lower end of the spectrum. The histogram reveals a significant number of cars with lower prices, suggesting that the dataset primarily consists of budget-friendly used cars. However, a few higher-priced cars exist, but they are outliers.

● **Kilometers Driven**: The kilometers driven are widely distributed, but most cars fall within the range of 50,000 to 100,000 kilometers. This indicates that a substantial proportion of the vehicles are moderately used. A smaller proportion of the vehicles have either very high or very low mileage.

Fuel Type Distribution:

● **Pie Chart Analysis**: The fuel type distribution is dominated by petrol and diesel vehicles, which together account for the vast majority of the cars. Petrol cars constitute the largest percentage, followed by diesel cars. There is a smaller proportion of cars using alternative fuel sources, such as electric or hybrid cars. This reflects the general trends in fuel preferences in the car market, where petrol and diesel still dominate.

Code With Output:

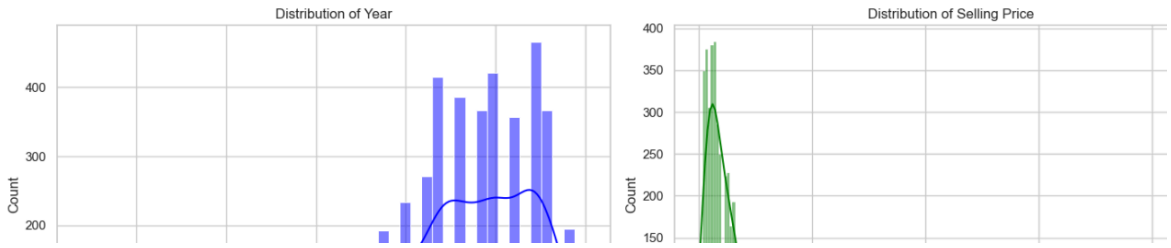
[3]:
```
import pandas as pd
CAR_df = pd.read_csv("C:/Users/DELL/Downloads/CAR DETAILS FROM CAR DEKHO.csv")
print(CAR_df)
```

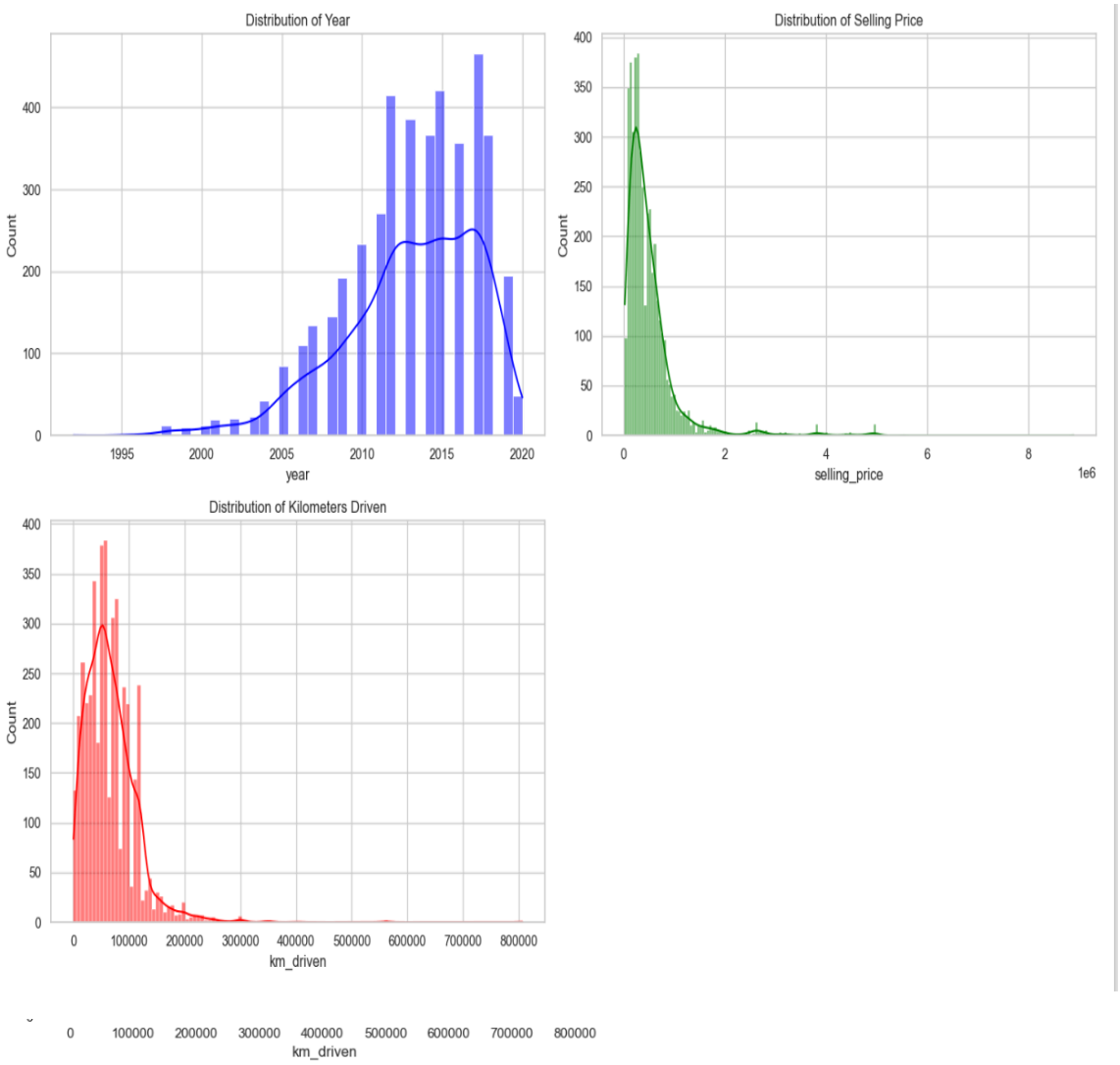
```
                                  name  year  selling_price  km_driven  \
0                        Maruti 800 AC  2007          60000      70000
1                Maruti Wagon R LXI Minor  2007         135000      50000
2                    Hyundai Verna 1.6 SX  2012         600000     100000
3                 Datsun RediGO T Option  2017         250000      46000
4                  Honda Amaze VX i-DTEC  2014         450000     141000
...                                  ...   ...            ...        ...
4335  Hyundai i20 Magna 1.4 CRDi (Diesel)  2014         409999      80000
4336           Hyundai i20 Magna 1.4 CRDi  2014         409999      80000
4337                 Maruti 800 AC BSIII  2009         110000      83000
4338     Hyundai Creta 1.6 CRDi SX Option  2016         865000      90000
4339                   Renault KWID RXT  2016         225000      40000

        fuel seller_type transmission         owner
0     Petrol  Individual       Manual   First Owner
1     Petrol  Individual       Manual   First Owner
2     Diesel  Individual       Manual   First Owner
3     Petrol  Individual       Manual   First Owner
4     Diesel  Individual       Manual  Second Owner
...      ...         ...          ...           ...
4335  Diesel  Individual       Manual  Second Owner
4336  Diesel  Individual       Manual  Second Owner
4337  Petrol  Individual       Manual  Second Owner
4338  Diesel  Individual       Manual   First Owner
4339  Petrol  Individual       Manual   First Owner

[4340 rows x 8 columns]
```

[5]:
```
# Summary statistics for numerical columns in the dataset
summary_stats = CAR_df.describe()
print(summary_stats)
```

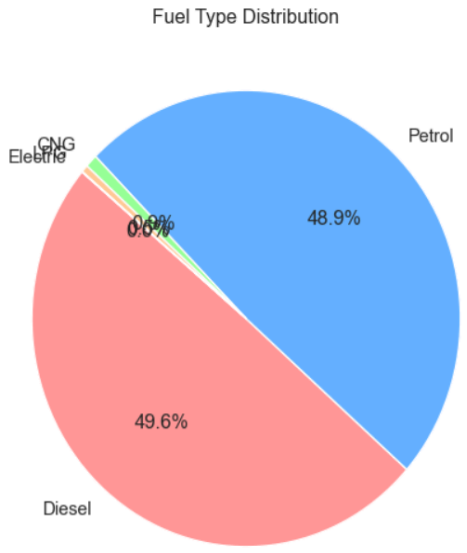
```
              year  selling_price      km_driven
count  4340.000000   4.340000e+03    4340.000000
mean   2013.090783   5.041273e+05   66215.777419
std       4.215344   5.785487e+05   46644.102194
min    1992.000000   2.000000e+04       1.000000
25%    2011.000000   2.087498e+05   35000.000000
50%    2014.000000   3.500000e+05   60000.000000
75%    2016.000000   6.000000e+05   90000.000000
max    2020.000000   8.900000e+06  806599.000000
```

[7]:
```
#Univariate analysis and observations
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid")
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
sns.histplot(CAR_df['year'], kde=True, color='blue')
plt.title('Distribution of Year')
plt.subplot(2, 2, 2)
sns.histplot(CAR_df['selling_price'], kde=True, color='green')
plt.title('Distribution of Selling Price')
plt.subplot(2, 2, 3)
sns.histplot(CAR_df['km_driven'], kde=True, color='red')
plt.title('Distribution of Kilometers Driven')
plt.tight_layout()
plt.show()
```





[33]:
```
# Plotting the pie chart for fuel type distribution
import matplotlib.pyplot as plt
fuel_counts = CAR_df['fuel'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(fuel_counts, labels=fuel_counts.index, autopct='%1.1f%%', startangle=140, colors=['#ff9999','#66b3ff','#99ff99','#ffcc99'])
plt.title('Fuel Type Distribution')
plt.show()
```



[35]:
```
# Checking for missing values
missing_values = CAR_df.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
dtype: int64
```

[37]:
```
# Checking the distribution of categorical variables
fuel_count = CAR_df['fuel'].value_counts()
seller_type_count = CAR_df['seller_type'].value_counts()
transmission_count = CAR_df['transmission'].value_counts()
owner_count = CAR_df['owner'].value_counts()
print("Fuel Type Count:\n", fuel_count)
print("\nSeller Type Count:\n", seller_type_count)
print("\nTransmission Type Count:\n", transmission_count)
print("\nOwner Count:\n", owner_count)
```

```
Fuel Type Count:
 fuel
Diesel      2153
Petrol      2123
CNG           40
LPG           23
Electric       1
Name: count, dtype: int64

Seller Type Count:
 seller_type
Individual          3244
Dealer               994
Trustmark Dealer     102
Name: count, dtype: int64

Transmission Type Count:
 transmission
Manual       3892
Automatic     448
Name: count, dtype: int64

Owner Count:
 owner
First Owner             2832
Second Owner            1106
Third Owner              304
Fourth & Above Owner      81
Test Drive Car            17
Name: count, dtype: int64
```

[39]:
```
#Plotting the histogram for 'selling_price'
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
plt.hist(CAR_df['selling_price'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Selling Price')
plt.xlabel('Selling Price')
plt.ylabel('Frequency')
plt.show()
```

[ ]:

Assignment: 3 B

Title:

Data Visualization with Matplotlib and Seaborn.

Dataset: Use the iris dataset or any available datasets. Perform the following tasks:

- Data Loading and Summary
- Visualizations with Matplotlib
- Visualizations with Seaborn

Aim:

To load and visualize the Iris dataset using data visualization techniques with the help of Matplotlib and Seaborn libraries in Python. The goal is to generate visual representations such as histograms, scatter plots, box plots, and heatmaps to understand the distribution and relationships of the dataset features.

Theory:

Introduction to Data Visualization:

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

Iris Dataset:

The Iris dataset is a popular dataset used in pattern recognition and machine learning literature. It includes the measurements of sepal length, sepal width, petal length, and petal width for 150 iris flowers belonging to three species:

- Iris-setosa
- Iris-versicolor
- Iris-virginica

The goal of visualizing this dataset is to understand the relationships between different features, how the features differ across species, and to explore patterns in the data.

Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used in the Python ecosystem for creating charts such as line plots, scatter plots, histograms, and more.

Seaborn:

Seaborn is a Python data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It is particularly useful for creating complex visualizations like pair plots and heatmaps with fewer lines of code.

Histograms:

A histogram is a graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable and shows the frequency of values falling within certain ranges.

Box Plot:

A box plot displays the distribution of data based on a five-number summary:

- Minimum
- First quartile (Q1)
- Median (Q2)
- Third quartile (Q3)
- Maximum

Scatter Plot:

A scatter plot uses dots to represent values for two different variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point.

Heatmap:

A heatmap is a graphical representation of data where individual values are represented as colors. It is commonly used to visualize the strength of relationships between different features, with color intensity indicating correlation magnitude.

Algorithm :

Step 1: Data Loading

- Import the required libraries such as pandas, matplotlib.pyplot, and seaborn.
- Load the Iris dataset from a CSV file into a pandas DataFrame.

Step 2: Data Preprocessing

- Examine the dataset to ensure it is loaded correctly.
- Check for missing values and handle them accordingly (if necessary).

❖ Understand the dataset structure by checking data types, descriptive statistics, and correlation between features.

Step 3: Data Visualization

❖ Use Matplotlib to create basic visualizations such as histograms and scatter plots to understand the distribution of data and relationships between features.

❖ Use Seaborn for more advanced visualizations such as box plots, pair plots, and heatmaps to gain deeper insights into the dataset.

Step 4: Analysis of Visualization

❖ Analyze the visualizations to identify patterns, trends, and relationships between features. For instance, explore how petal and sepal dimensions differ across species and how features are correlated.

Step 5: Conclusion

❖ Summarize the findings from the visualizations and their implications on understanding the dataset.

Mathematical Background:

The Pearson's correlation coefficient (rrr) is a statistical measure that calculates the strength and direction of a linear relationship between two variables. It is a value between -1 and 1, where:

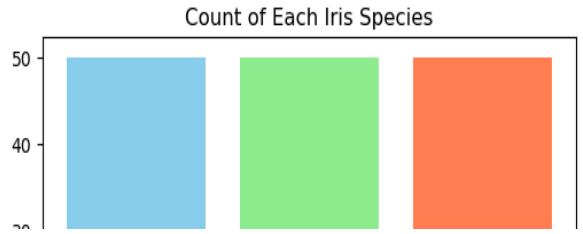
- ❖ r =-1 indicates a perfect positive linear relationship (as one variable increases, the other increases proportionally).
- ❖ r = -1 indicates a perfect negative linear relationship (as one variable increases, the other decreases proportionally).
- ❖ r=0 indicates no linear relationship between the variables.

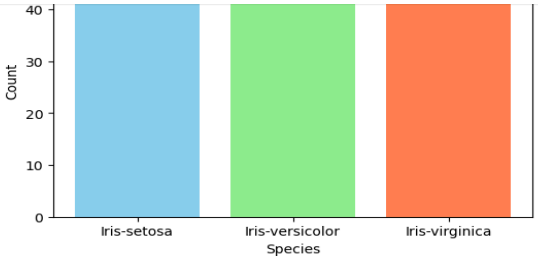
Code and Output:

[3]:
```
#Data Loading and Summary
import pandas as pd
data = pd.read_csv("C:/Users/DELL/Downloads/Iris.csv")
print(data.head())
print(data.info())
print(data.describe())
```

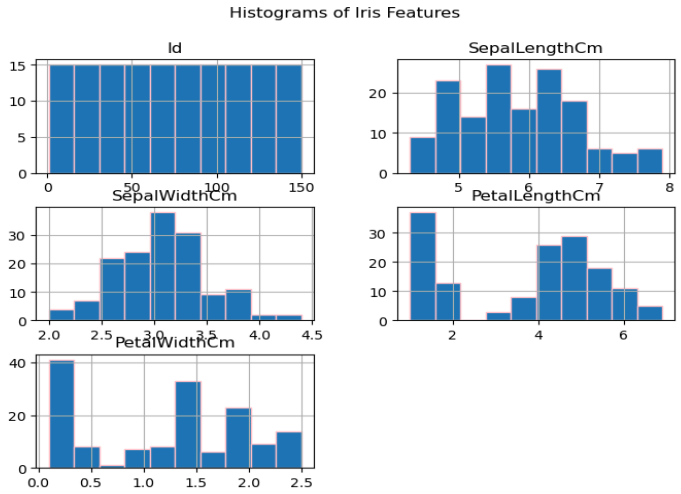
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.000000     150.000000    150.000000     150.000000    150.000000
mean    75.500000       5.843333      3.054000       3.758667      1.198667
std     43.445368       0.828066      0.433594       1.764420      0.763161
min      1.000000       4.300000      2.000000       1.000000      0.100000
25%     38.250000       5.100000      2.800000       1.600000      0.300000
50%     75.500000       5.800000      3.000000       4.350000      1.300000
75%    112.750000       6.400000      3.300000       5.100000      1.800000
max    150.000000       7.900000      4.400000       6.900000      2.500000
```

[9]:
```
# A bar plot for the count of species
import pandas as pd
import matplotlib.pyplot as plt
species_counts = data['Species'].value_counts()
plt.figure(figsize=(6, 4))
plt.bar(species_counts.index, species_counts.values, color=['skyblue', 'lightgreen', 'coral'])
plt.title('Count of Each Iris Species')
plt.ylabel('Count')
plt.xlabel('Species')
plt.show()
```

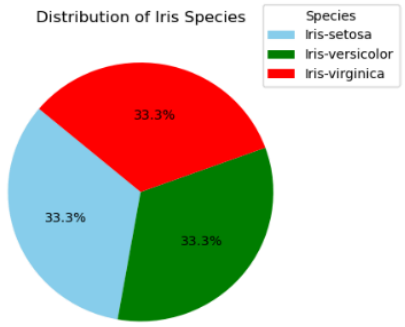




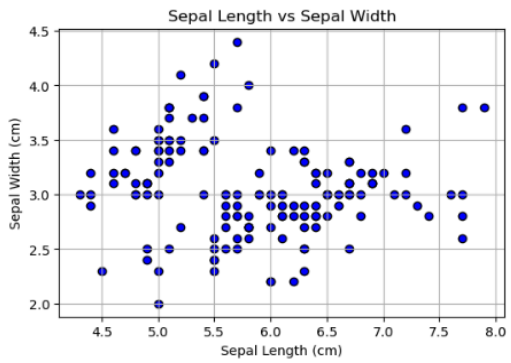
[11]:
```
# Create histograms for each feature
import matplotlib.pyplot as plt
data.hist(figsize=(8, 6), bins=10, edgecolor='pink')
plt.suptitle('Histograms of Iris Features')
plt.show()
```



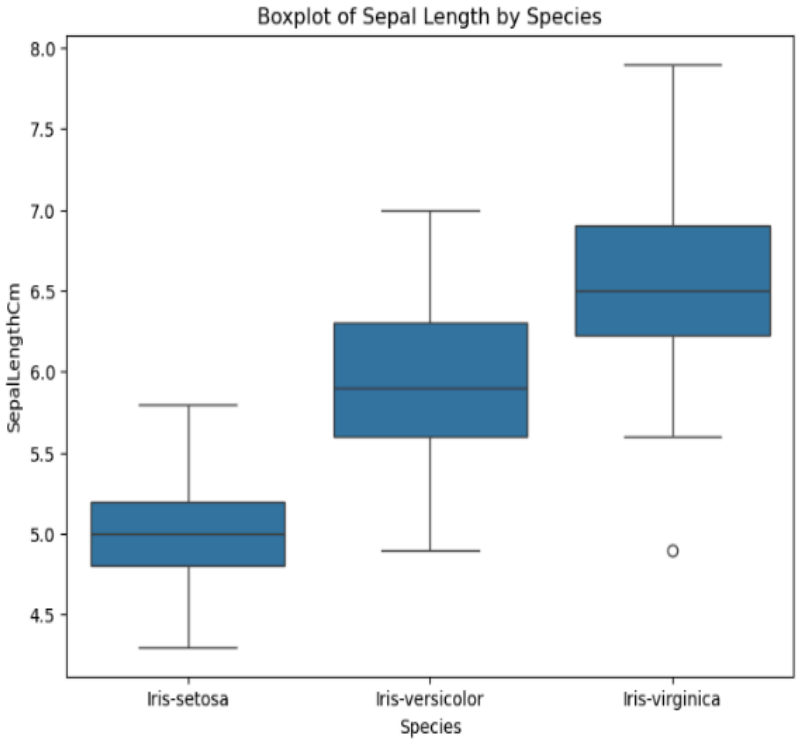
[13]:
```
#pie presentation for species
species_counts = data['Species'].value_counts()
plt.figure(figsize=(6.5, 4.5))
wedges, texts, autotexts = plt.pie(species_counts, autopct='%1.1f%%', colors=['skyblue', 'green', 'red'], startangle=140)
plt.legend(wedges, species_counts.index, title="Species", loc="upper right", bbox_to_anchor=(1.3, 1.1))
plt.title('Distribution of Iris Species')
plt.show()
```



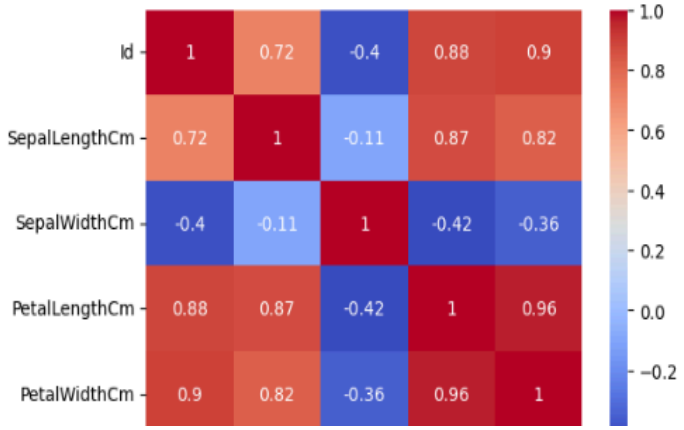
[15]:
```
#Scatter Plot: Visualizing the relationship between two features (e.g., Sepal Length vs. Sepal Width).
plt.figure(figsize=(6, 4))
plt.scatter(data['SepalLengthCm'], data['SepalWidthCm'], c='blue', edgecolor='k')
plt.title('Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.grid(True)
plt.show()
```



[17]:
```
# Boxplot of Sepal Length grouped by species
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.boxplot(x='Species', y='SepalLengthCm', data=data)
plt.title('Boxplot of Sepal Length by Species')
plt.show()
```

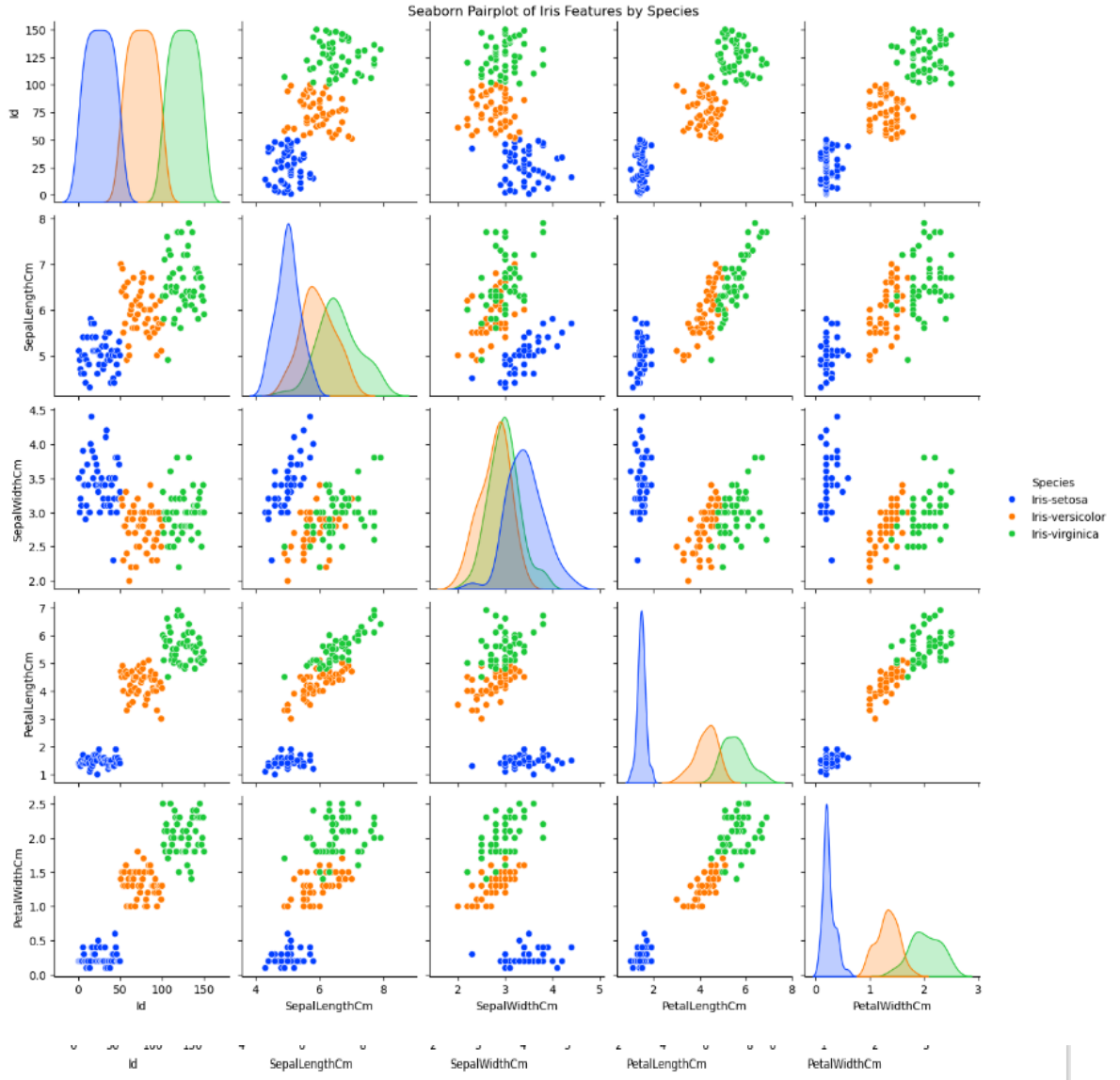


[21]:
```
# Heatmap plot
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("C:/Users/DELL/Downloads/Iris.csv")
numeric_data = data.select_dtypes(include=['number'])
corr = numeric_data.corr()
plt.figure(figsize=(6, 4))
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.show()
```

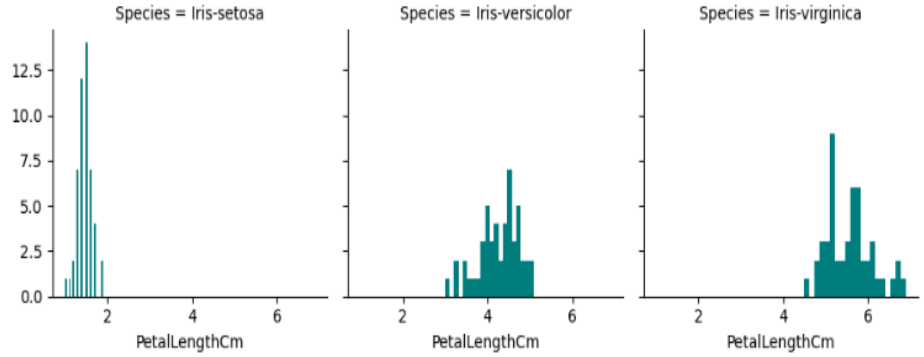


Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm

[23]:
```
# Using Seaborn's pairplot function to visualize relationships between variables
sns.pairplot(data, hue='Species', palette='bright')
plt.suptitle('Seaborn Pairplot of Iris Features by Species', y=1.0)
plt.show()
```



[25]:
```
# Create a FacetGrid to visualize distribution of Petal Length
g = sns.FacetGrid(data, col='Species', height=3)
g.map(plt.hist, 'PetalLengthCm', bins=20, color='teal')
plt.show()
```



Assignment: 3 C

Title:

Exploratory Data Analysis using any datasets from UCI/Kaggle and Perform the following tasks:

- Data Loading and Summary Statistics
- Data Distribution Analysis
- Bivariate / multivariate analysis
- Correlation analysis and write observation

Aim:

To perform Exploratory Data Analysis (EDA) on the Heart Disease dataset from UCI, identifying patterns, relationships between variables, and key features that contribute to the presence of heart disease.

Theory:

Exploratory Data Analysis (EDA) is a critical step in data analysis and machine learning, allowing us to understand the underlying patterns in data, detect anomalies, check assumptions, and identify relationships between variables before applying predictive models.

EDA includes:

1. Data Loading & Summary Statistics**:** Understanding the structure of the dataset and the characteristics of individual features.

2. Data Distribution Analysis: Analyzing the spread and distribution of data to detect outliers, skewness, or multimodality.

3. Bivariate/Multivariate Analysis: Investigating relationships between two or more variables to detect interactions or trends.

4. Correlation Analysis: Quantifying the strength of linear relationships between numerical features, helping to assess the interdependence of variables. The goal of EDA is to summarize data, generate hypotheses, and prepare for model building by handling missing data, outliers, and creating relevant features.

Algorithm:

1. Data Loading and Summary Statistics:

● Objective: To get an initial understanding of the dataset structure (rows, columns, data types) and summary statistics (mean, standard deviation, quartiles).

● Operations:

- ○ Load the dataset using tools like pandas.
- ○ Check the shape (number of rows and columns).
- ○ Display the first few rows to understand the data layout.
- ○ Calculate basic descriptive statistics for numerical columns (mean, median,max, min, etc.).

2. Data Distribution Analysis:

● Objective: To explore the distribution of individual numerical and categorical variables.

● Operations:

- ○ Plot histograms for continuous variables to visualize the frequency distribution.
- ○ Boxplots to identify outliers and analyze the spread of the data.
- ○ Count plots for categorical variables to visualize the frequency of each category.

3. Bivariate and Multivariate Analysis:

● Objective: To investigate relationships between two or more variables and detect any trends, patterns, or correlations.

● Operations:

- ○ Scatter plots and pair plots to examine relationships between continuous variables.
- ○ Grouped bar plots for categorical data.
- ○ Heatmaps to visualize interactions between numerical features.

4. Correlation Analysis:

● Objective: To quantify the linear relationships between variables, identifying which variables are strongly related.

● Operations:

- ○ Compute the correlation matrix to measure the correlation coefficient (ranging from -1 to 1).

○ Use heatmaps to visualize the correlation matrix.

Mathematical Background:

1. Summary Statistics:
    - ○ Mean: Measures the central tendency of the data.
    - ○ Median: The middle value when the data is sorted.
    - ○ Standard Deviation: Measures the spread or variability of data.
    - ○ Quartiles: Divide the data into four equal parts.

2. Correlation Coefficient:
    - ○ Pearson Correlation Coefficient: Measures the linear relationship between two variables.

      Where:
        - ■ r is the correlation coefficient.
        - ■ x and $\bar{y}$ are the means of variables x and y, respectively.
    - ○ The value of rrr lies between -1 and 1:
        - ■ +1: Perfect positive correlation.
        - ■ -1: Perfect negative correlation.
        - ■ 0: No correlation.

3. Data Visualization:
    - ○ Histograms: Represent the frequency distribution of numerical data. It helps detect skewness, modality, and outliers.
    - ○ Box Plots: Show the distribution of data based on five summary statistics (minimum, first quartile, median, third quartile, and maximum) and help detect outliers.
    - ○ Scatter Plots: Useful for visualizing the relationship between two continuous variables.
    - ○ Heatmaps: Visualize the strength of correlation between variables using colors.

Dataset:

https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset?resource=download

Conclusion:

The EDA reveals key patterns in the Heart Disease dataset:

- ● Variables like age, thalach, oldpeak, and chest pain type show strong relationships with the presence of heart disease.

- ● Certain variables display potential multicollinearity, which may require feature selection or regularization before model building.

- Correlation Analysis highlighted the importance of variables like thalach and oldpeak in predicting heart disease, suggesting their importance in predictive modeling.

Code and Output:

[1]:
```
# Required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset into a pandas dataframe
df = pd.read_csv("C:/Users/DELL/Downloads/heart.csv")
print(df)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
...   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
...     ...  ..   ...     ...
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

[3]:
```
# Check for missing or invalid data
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
```

[5]:
```
# Display the first few rows of the dataset
print(df.head())
```

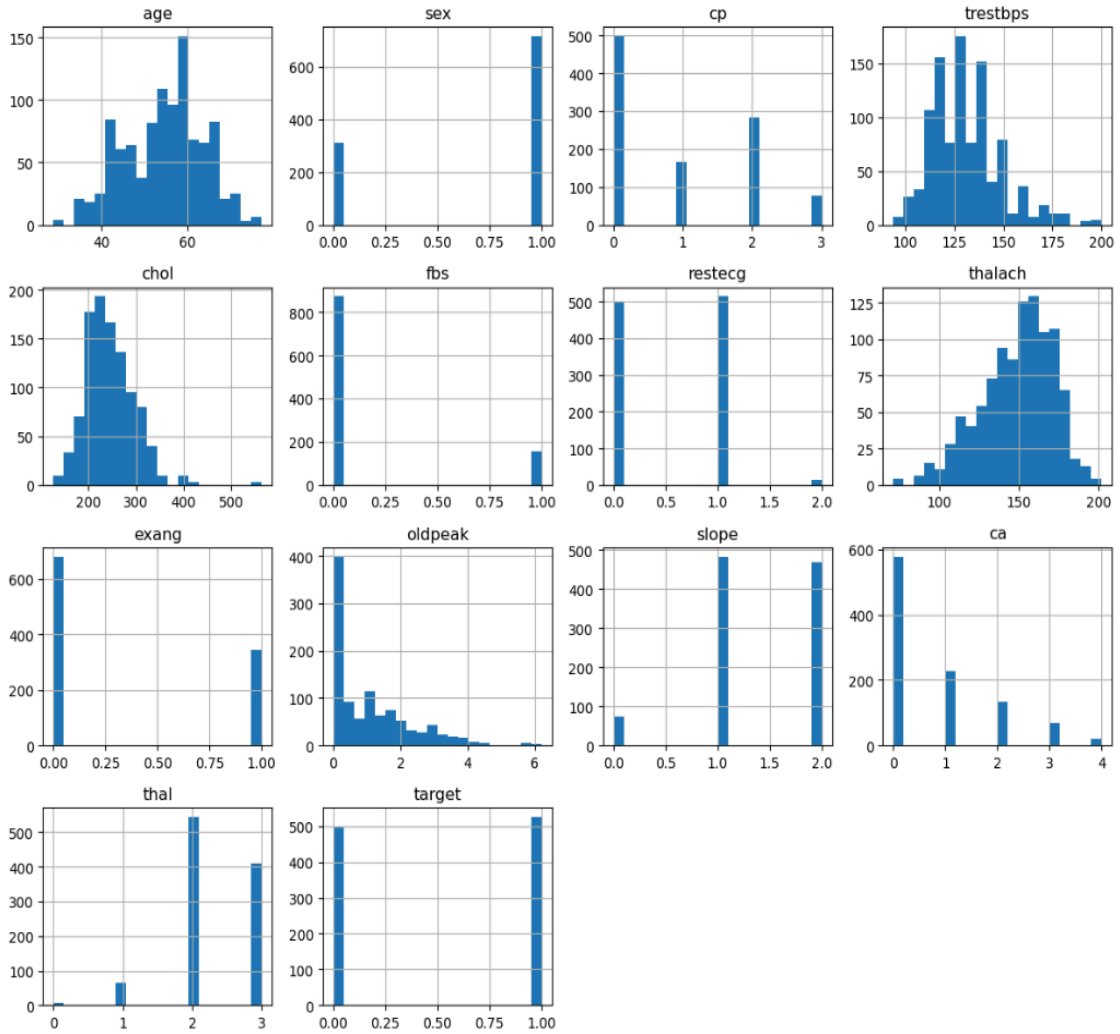
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

[7]:
```
df.describe()
```

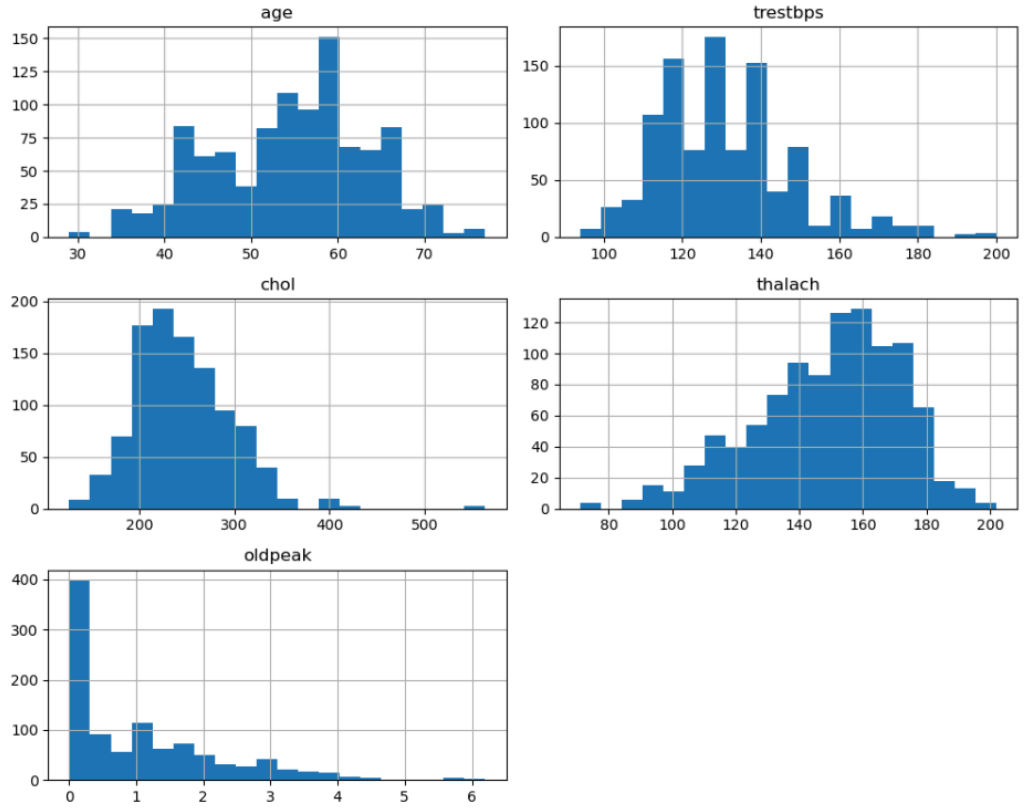
[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	10
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	

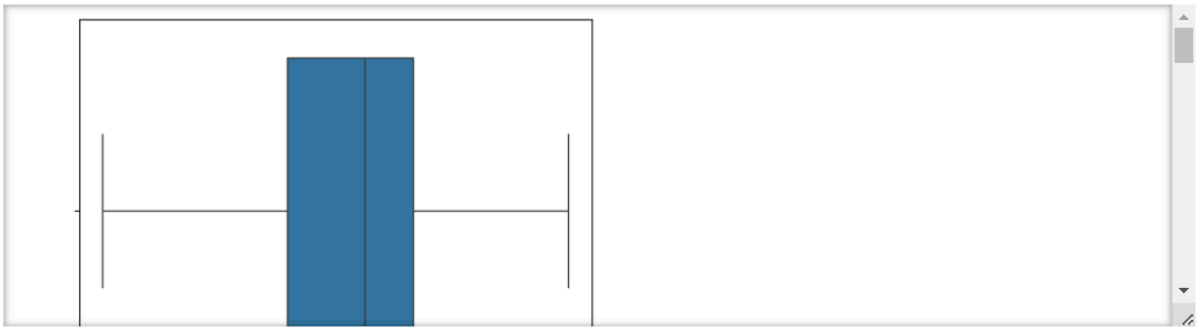
[9]:
```
# Visualize the distribution of each numerical column
df.hist(figsize=(12, 10), bins=20)
plt.tight_layout()
plt.show()
```



[13]:
```
# 1. Histogram for continuous variables
df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']].hist(figsize=(10, 8), bins=20)
plt.tight_layout()
plt.show()
```

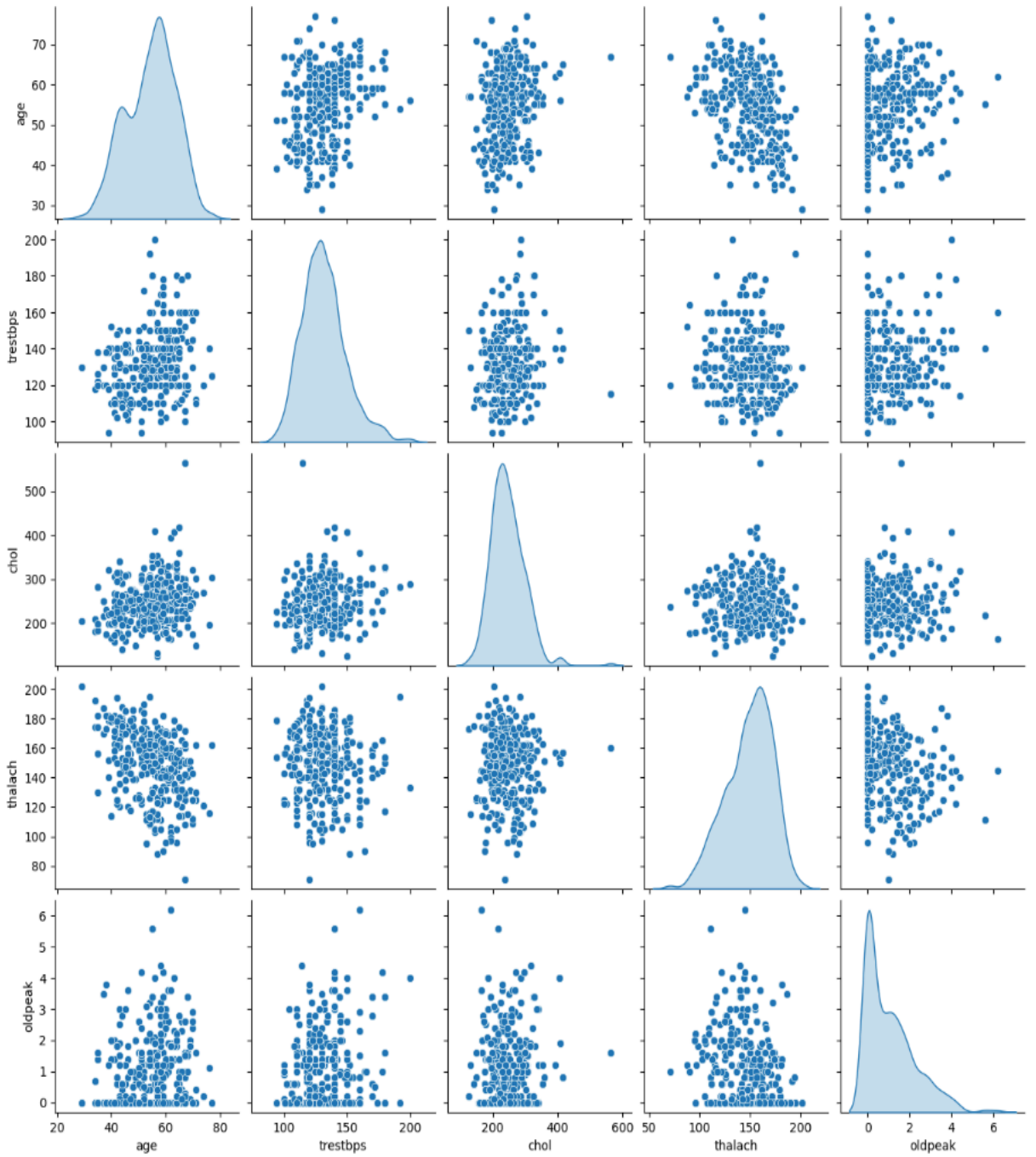


[15]:
```
#3. Boxplots to check for outliers in continuous variables
for col in ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']:
    sns.boxplot(x=col, data=df)
    plt.show()
```

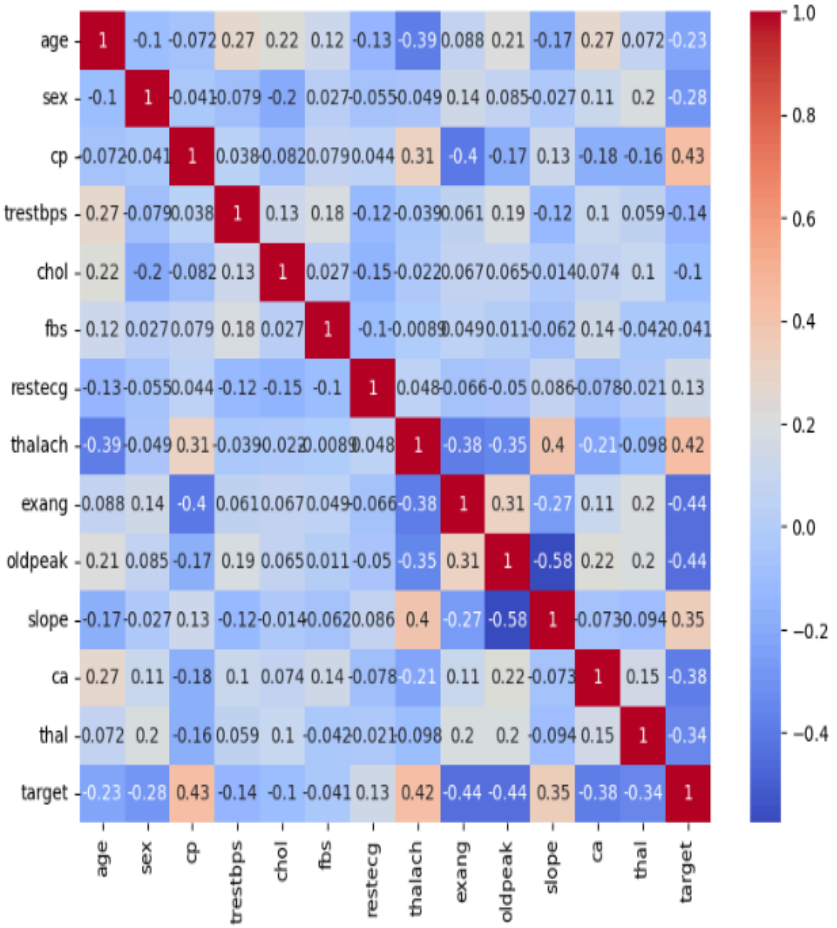


[17]:
```
# 2. Count plot for categorical variables (e.g., 'sex', 'cp', 'fbs', 'restecg', 'exang', 'num')
categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'num']
for col in categorical_columns:
    sns.countplot(x=col, data=df)
    plt.show()
```

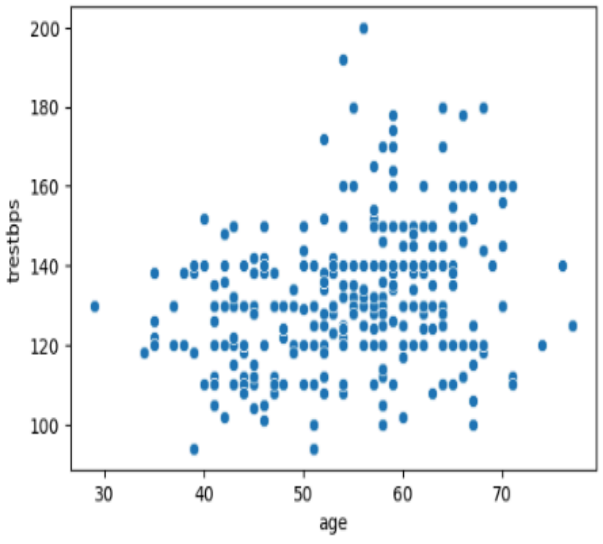
[19]:
```
# Pairplot: Visualizing relationships between multiple variables
sns.pairplot(df[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']],diag_kind='kde')
plt.show()
```



[21]:
```
# Correlation heatmap
plt.figure(figsize=(9, 7))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

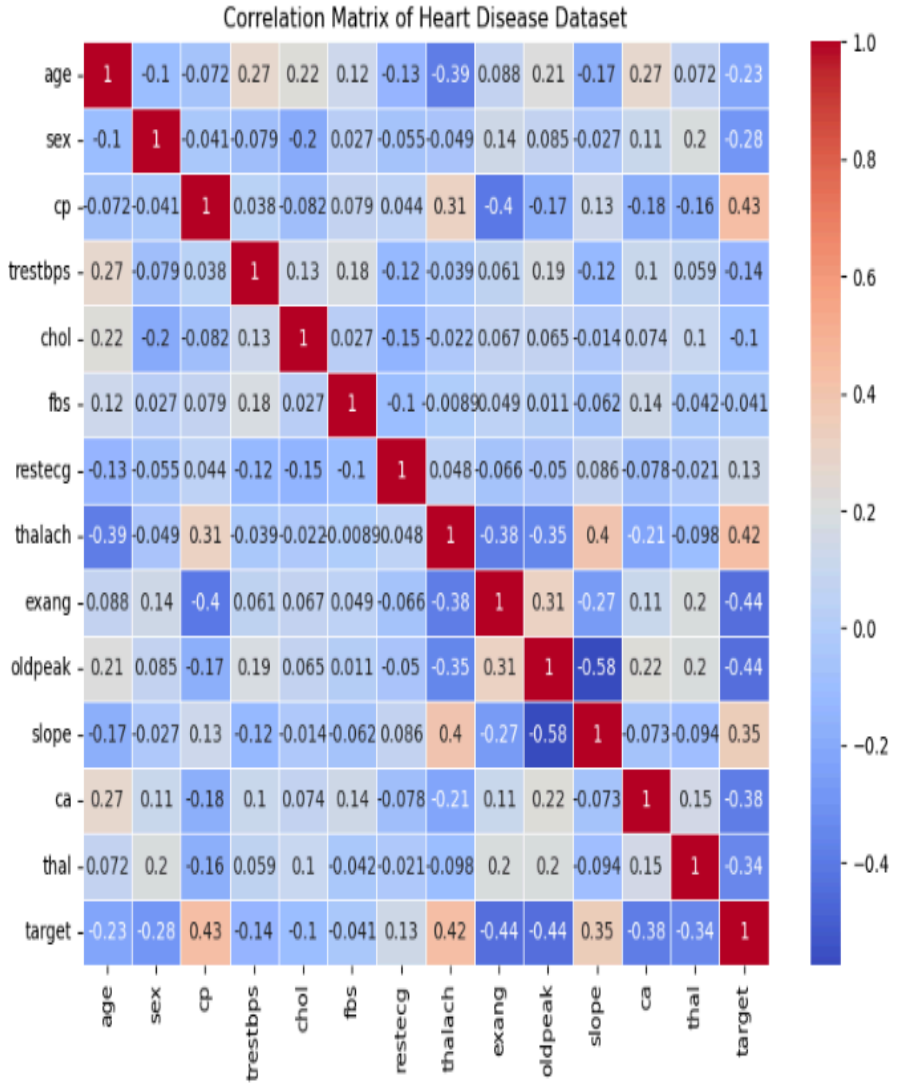


[23]:
```
# Scatter plots for key features
plt.figure(figsize=(6,4))
sns.scatterplot(x='age', y='trestbps', data=df)
plt.show()
```



[25]:
```
# Convert invalid entries (like '?') to NaN and convert columns to numeric
df.replace('?', pd.NA, inplace=True)
df = df.apply(pd.to_numeric, errors='coerce')

# Drop rows with missing values (optional, depends on how you want to handle missing data)
df = df.dropna()

# Compute correlation matrix
corr_matrix = df.corr()

# Display the correlation matrix
print(corr_matrix)

# Plot the heatmap for better visualization
plt.figure(figsize=(10, 7))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Heart Disease Dataset')
plt.show()
```

```
              age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.103240 -0.071966  0.271121  0.219823  0.121243
sex      -0.103240  1.000000 -0.041119 -0.078974 -0.198258  0.027200
cp       -0.071966 -0.041119  1.000000  0.038177 -0.081641  0.079294
trestbps  0.271121 -0.078974  0.038177  1.000000  0.127977  0.181767
chol      0.219823 -0.198258 -0.081641  0.127977  1.000000  0.026917
fbs       0.121243  0.027200  0.079294  0.181767  0.026917  1.000000
restecg  -0.132696 -0.055117  0.043581 -0.123794 -0.147410 -0.104051
thalach  -0.390227 -0.049365  0.306839 -0.039264 -0.021772 -0.008866
exang     0.088163  0.139157 -0.401513  0.061197  0.067382  0.049261
oldpeak   0.208137  0.084687 -0.174733  0.187434  0.064880  0.010859
slope    -0.169105 -0.026666  0.131633 -0.120445 -0.014248 -0.061902
ca        0.271551  0.111729 -0.176206  0.104554  0.074259  0.137156
thal      0.072297  0.198424 -0.163341  0.059276  0.100244 -0.042177
target   -0.229324 -0.279501  0.434854 -0.138772 -0.099966 -0.041164

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.132696 -0.390227  0.088163  0.208137 -0.169105  0.271551
sex      -0.055117 -0.049365  0.139157  0.084687 -0.026666  0.111729
cp        0.043581  0.306839 -0.401513 -0.174733  0.131633 -0.176206
trestbps -0.123794 -0.039264  0.061197  0.187434 -0.120445  0.104554
chol     -0.147410 -0.021772  0.067382  0.064880 -0.014248  0.074259
fbs      -0.104051 -0.008866  0.049261  0.010859 -0.061902  0.137156
restecg   1.000000  0.048411 -0.065606 -0.050114  0.086086 -0.078072
thalach   0.048411  1.000000 -0.380281 -0.349796  0.395308 -0.207888
exang    -0.065606 -0.380281  1.000000  0.310844 -0.267335  0.107849
oldpeak  -0.050114 -0.349796  0.310844  1.000000 -0.575189  0.221816
slope     0.086086  0.395308 -0.267335 -0.575189  1.000000 -0.073440
ca       -0.078072 -0.207888  0.107849  0.221816 -0.073440  1.000000
thal     -0.020504 -0.098068  0.197201  0.202672 -0.094090  0.149014
target    0.134468  0.422895 -0.438029 -0.438441  0.345512 -0.382085

             thal    target
age      0.072297 -0.229324
sex      0.198424 -0.279501
cp      -0.163341  0.434854
trestbps 0.059276 -0.138772
chol     0.100244 -0.099966
fbs     -0.042177 -0.041164
restecg -0.020504  0.134468
thalach -0.098068  0.422895
exang    0.197201 -0.438029
oldpeak  0.202672 -0.438441
slope   -0.094090  0.345512
ca       0.149014 -0.382085
thal     1.000000 -0.337838
target  -0.337838  1.000000
```


Correlation Matrix of Heart Disease Dataset
age 1 -0.1 -0.072 0.27 0.22 0.12 -0.13 -0.39 0.088 0.21 -0.17 0.27 0.072 -0.23
sex -0.1 1 -0.041 -0.079 -0.2 0.027 -0.055 -0.049 0.14 0.085 -0.027 0.11 0.2 -0.28
cp -0.072 -0.041 1 0.038 -0.082 0.079 0.044 0.31 -0.4 -0.17 0.13 -0.18 -0.16 0.43
trestbps 0.27 -0.079 0.038 1 0.13 0.18 -0.12 -0.039 0.061 0.19 -0.12 0.1 0.059 -0.14
chol 0.22 -0.2 -0.082 0.13 1 0.027 -0.15 -0.022 0.067 0.065 -0.014 0.074 0.1 -0.1
fbs 0.12 0.027 0.079 0.18 0.027 1 -0.1 -0.0089 0.049 0.011 -0.062 0.14 -0.042 -0.041
restecg -0.13 -0.055 0.044 -0.12 -0.15 -0.1 1 0.048 -0.066 -0.05 0.086 -0.078 -0.021 0.13
thalach -0.39 -0.049 0.31 -0.039 -0.022 0.0089 0.048 1 -0.38 -0.35 0.4 -0.21 -0.098 0.42
exang 0.088 0.14 -0.4 0.061 0.067 0.049 -0.066 -0.38 1 0.31 -0.27 0.11 0.2 -0.44
oldpeak 0.21 0.085 -0.17 0.19 0.065 0.011 -0.05 -0.35 0.31 1 -0.58 0.22 0.2 -0.44
slope -0.17 -0.027 0.13 -0.12 -0.014 -0.062 0.086 0.4 -0.27 -0.58 1 -0.073 -0.094 0.35
ca 0.27 0.11 -0.18 0.1 0.074 0.14 -0.078 -0.21 0.11 0.22 -0.073 1 0.15 -0.38
thal 0.072 0.2 -0.16 0.059 0.1 -0.042 -0.021 -0.098 0.2 0.2 -0.094 0.15 1 -0.34
target -0.23 -0.28 0.43 -0.14 -0.1 -0.041 0.13 0.42 -0.44 -0.44 0.35 -0.38 -0.34 1
age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4

[ ]:

Assignment: 4 A

Title:

Download suitable Housing data from kaggle/ UCI / any other source, Predict House prices using Linear Regression and Evaluate the model using RMSE or any other suitable metric. Example Dataset: (https://www.kaggle.com/datasets/yasserh/housing-prices-dataset)
(Note: Use sci-kit learn Library)

Aim:

To develop a predictive model that can estimate house prices based on various property characteristics by implementing a linear regression model and evaluating It uses Root Mean Squared Error (RMSE) as the metric.

Theory:

Linear regression is a fundamental statistical technique for predictive modeling. It involves establishing a linear relationship between a dependent variable (in this case, house prices) and one or more independent variables (such as the size of the house, the number of bedrooms, and location features). The underlying assumption of linear regression is that there exists a linear relationship between the input features and the target output.

In this assignment, we use a simple linear regression model. Given a set of independent variables (X1,X2,...,Xn)(X_1, X_2, ..., X_n)(X1,X2,...,Xn), linear regression attempts to estimate the target variable YYY (house price) as a weighted sum of the input variables. This relationship can be represented mathematically as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon$$

where:

- YYY is the predicted house price,
- β0\beta_0β0 is the intercept (constant term),
- βi\beta_iβi are the coefficients of the independent variables XiX_iXi,
- ε\epsilonε is the error term.

Algorithm

1. Data Collection and Preprocessing:

- Load the dataset and inspect the data structure.

○ Handle missing values by either imputing or removing them.

○ Encode categorical variables using techniques like one-hot encoding.

○ Standardize or normalize numerical features as needed to ensure all features are on a similar scale.

2. Split the Dataset:

○ Divide the dataset into training and testing sets. Commonly, a 70:30 or 80:20 split is used.

3. Model Training:

○ Use the training set to train a linear regression model. Scikit-learn's LinearRegression function can be used to fit the model to the data.

4. Prediction:

○ Use the trained model to predict house prices on the test data.

5. Evaluation:

○ Calculate the Root Mean Squared Error (RMSE) on the test data to evaluate model performance.

Mathematical Background:

Linear regression estimates the coefficients by minimizing the residual sum of squares between the observed target values and the values predicted by the model. The cost function, Mean Squared Error (MSE), for the model is:

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- yiy_iyi is the actual house price,
- y^i\hat{y}_iy^i is the predicted house price, and
- nnn is the number of observations.

The RMSE is the square root of MSE, providing a more interpretable metric by keeping the unit consistent with the predicted variable:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Dataset

- Dataset Name: Housing Prices Dataset
- Source: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset
- Description: This dataset contains information on house features such as area, number of rooms, location, etc. It's commonly used for predictive modeling in the real estate domain.

Code and Output:

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
df = pd.read_csv("C:/Users/DELL/Downloads/Housing.csv")
df.head(10)
```

[1]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
5	10850000	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished
6	10150000	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished
7	10150000	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished
8	9870000	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished
9	9800000	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

[ ]:
```
#encode categorical variables
df=pd.get_dummies(df,columns=['mainroad','guestroom','basement','hotwaterheating','airconditioning','prefarea','furnishingstatus'],drop_first=True)
```

[14]:
```
#define features x and target y
X=df.drop('price',axis=1)
y=df['price']
```

[16]:
```
X = df.drop('price', axis=1)
y = df['price']

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

model = LinearRegression()

model.fit(X_train, y_train)
```

[16]:
```
LinearRegression
LinearRegression()
```

[18]:
```
y_pred = model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 1324506.9600914388
```

[20]:
```
comparison = pd.DataFrame({'Actual Price': y_test, 'Predicted Price': y_pred})
print(comparison)
```

```
     Actual Price  Predicted Price
316       4060000     5.164654e+06
77        6650000     7.224722e+06
360       3710000     3.109863e+06
90        6440000     4.612075e+06
493       2800000     3.294646e+06
..            ...              ...
15        9100000     4.973331e+06
357       3773000     4.336651e+06
39        7910000     7.059063e+06
54        7350000     6.398941e+06
155       5530000     6.363890e+06

[109 rows x 2 columns]
```

Assignment: 4 B

Title:

Download Diabetic data from kaggle/ UCI / any other source, Build classification model for diabetic prediction and Evaluate the model using accuracy, precision, Recall etc.,.

Example Dataset: (https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database)

(Note: Use sci-kit learn Library)

Aim:

To develop a classification model to predict the likelihood of diabetes in patients based on medical and demographic features, and evaluate the model's performance using accuracy, precision, recall, and F1-score.

Theory:

Classification models aim to predict categorical outcomes based on input features. For this assignment, we'll use supervised classification to predict a binary outcome: whether a person is diabetic (positive class) or non-diabetic (negative class). This classification task is essential in medical diagnosis, where accurate predictions can significantly impact patient care.

Logistic Regression, Decision Trees, and other classification algorithms can be applied here. Each classifier operates differently:

- Logistic Regression uses a sigmoid function to estimate probabilities.
- Decision Trees split data points based on feature thresholds to classify outcomes.

Algorithm:

1. Data Collection and Preprocessing:

- Load and inspect the dataset.
- Handle missing values by imputation.
- Standardize or normalize continuous features (e.g., age, glucose levels) as needed.

2. Dataset Splitting:

- Split the data into training and testing sets.

3. Model Selection and Training:

○ Train a classification model (e.g., Logistic Regression or Decision Tree) on the training set using scikit-learn's library.

4. Prediction:

○ Use the trained model to make predictions on the test data.

5. Evaluation Metrics:

○ Accuracy: Proportion of correctly predicted instances.

○ Precision: Measures how many of the predicted positive cases are actually positive.

○ Recall: Measures how many of the actual positive cases are predicted correctly.

○ F1 Score: Harmonic mean of precision and recall, useful for imbalanced datasets.

Mathematical Background:

- **Accuracy**: $\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}$
- **Precision**: $\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$
- **Recall**: $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$
- **F1 Score**: $\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Dataset

● Dataset Name: Pima Indians Diabetes Database

● Source: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

● Description: Contains medical data like glucose levels, BMI, and blood pressure of patients, which are used as features for diabetes prediction.

Code and Output:

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
data = pd.read_csv("C:/Users/DELL/Downloads/diabetes.csv")
```

[3]:
```
data.head()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[5]:
```
# Check for missing values
print(data.isnull().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[7]:
```
data.dropna(axis=0,how='all',inplace=True)
```

[9]:
```
data.dtypes
```

[9]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

[11]:
```
X = data.drop(columns='Outcome')  # Features
y = data['Outcome']  # Target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[17]:
```
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

[17]:
```
RandomForestClassifier(random_state=42)
```

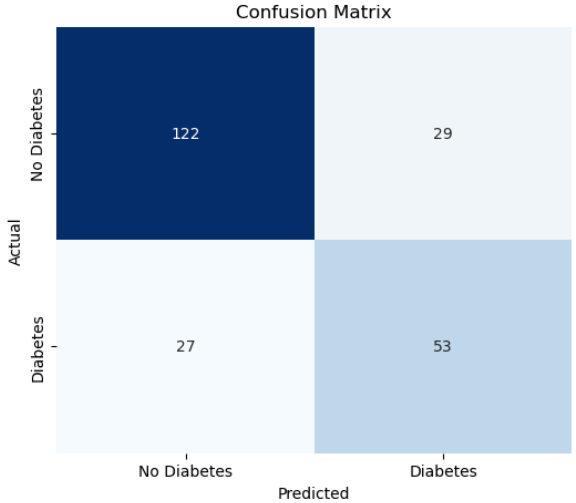
[15]:
```
y_pred = model.predict(X_test)
```

[19]:
```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Precision
precision = precision_score(y_test, y_pred)

# Recall
recall = recall_score(y_test, y_pred)

# F1 Score
f1 = f1_score(y_test, y_pred)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.76
Precision: 0.65
Recall: 0.66
F1 Score: 0.65
Confusion Matrix:
[[122  29]
 [ 27  53]]
```

[27 53]]

[21]:
```
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Diabetes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Assignment: 4 C

Title:

Hypothesis test: Assume our business has two units that make pizzas. Check if there is any significant difference in the average diameter of pizzas between the two making units.

Aim:

The primary aim of this analysis is to determine if there exists a statistically significant difference in the average diameter of pizzas produced by two different units within our business. Understanding this difference is crucial for maintaining product consistency, ensuring customer satisfaction, and implementing quality control measures across production facilities.

Theory:

This analysis is rooted in the principles of statistical hypothesis testing. Hypothesis testing is a statistical method used to make inferences about population parameters based on sample data. In this context, we focus on comparing the means of two independent samples—the pizza diameters from Unit A and Unit B.

The statistical test employed for this analysis is the **independent two-sample t-test**. This test is appropriate when comparing means from two different groups that are independent of one another. The underlying assumptions of this test include:

- ❖ Independence: The samples must be collected independently from each other.
- ❖ Normality: The data in each group should be approximately normally distributed, especially for smaller sample sizes.
- ❖ Equal Variances: The variances of the two populations should be equal, although the t-test is robust to violations of this assumption in larger samples.

Algorithm:

The analysis follows a systematic approach:

➔ Formulate Hypotheses:

◆ **Null Hypothesis (H0):** There is no significant difference in the average diameter of pizzas between Unit A and Unit B (μA=μB\mu_A = \mu_BμA=μB).

◆ **Alternative Hypothesis (Ha):** There is a significant difference in the average diameter of pizzas between Unit A and Unit B (μA≠μB\mu_A \neq\mu_BμA=μB).

➔ **Collect Data:** Gather the diameter measurements of pizzas produced by both units. This data should be recorded accurately to reflect true production quality.

➔ **Conduct t-test:** Perform the independent two-sample t-test on the collected data to determine the t-statistic and corresponding p-value.

➔ **Decision Rule:**

- ◆ Set a significance level (α\alphaα), typically 0.05.
- ◆ Compare the computed p-value from the t-test to the significance level.
- ◆ If p<αp < \alphap<α, reject the null hypothesis, suggesting a significant difference; if p≥αp \geq \alphap≥α, fail to reject the null hypothesis.

➔ **Interpret Results:** Based on the statistical findings, provide insights regarding the average diameters of pizzas from both units.

Mathematical Background:

$$t = \frac{\bar{X}_A - \bar{X}_B}{s_p\sqrt{\frac{1}{n_A} + \frac{1}{n_B}}}$$

Where:

- $\bar{X}_A$ and $\bar{X}_B$ are the sample means of the two groups.
- s_p is the pooled standard deviation, calculated as:

$$s_p = \sqrt{\frac{(n_A - 1)s_A^2 + (n_B - 1)s_B^2}{n_A + n_B - 2}}$$

- n_A and n_B are the sample sizes for units A and B, respectively.
- s_A and s_B are the sample standard deviations for each unit.

The degrees of freedom (df) for the t-test can be calculated as:

$$df = n_A + n_B - 2$$

The independent two-sample t-test assesses whether the means of two groups are statistically different from each other. The t-statistic is calculated using the following formula:

After calculating the t-statistic, we can find the p-value using the t-distribution with the computed degrees of freedom. The p-value indicates the probability of observing the data, or something more extreme, assuming the null hypothesis is true.

Conclusion:

This comparative analysis aims to provide insights into the production quality of our pizza-making units. By employing the independent two-sample t-test, we can quantitatively assess whether any observed differences in average pizza diameters are statistically significant.

Understanding the results of this analysis will enable our business to maintain consistency in product quality across different units, ultimately enhancing customer satisfaction and trust in our brand. Furthermore, if significant differences are found, it may prompt an investigation into production processes, quality control measures, and equipment standards between the two units.

Code and Output:

*[1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
```

*[2]:
```
df=pd.read_csv(r"C:\Users\DELL\Downloads\pizzas - pizzas.csv")
```

[3]:
```
df.head(4)
```

[3]:

	Making Unit 1	Making Unit 2
0	6.8090	6.7703
1	6.4376	7.5093
2	6.9157	6.7300
3	7.3012	6.7878

[4]:
```
df.shape
```

[4]:
```
(35, 2)
```

[5]:
```
#Access each column directly since each column corresponds to a different unit.
unit1 = df['Making Unit 1']
unit2 = df['Making Unit 2']
```

[6]:
```
#Define the significance level (alpha), commonly 0.05.
alpha = 0.05
```

[7]:
```
#Use stats.ttest_ind() to compare the two columns.
t_statistic, p_value = stats.ttest_ind(unit1, unit2)
```

[8]:
```
#Compare the p-value with alpha to determine
#if there's a significant difference in pizza diameters.
if p_value < alpha:
    print("Reject the null hypothesis: Significant difference in pizza diameter.")
else:
    print("Fail to reject the null hypothesis: No significant difference in pizza diameter.")
```

```
Fail to reject the null hypothesis: No significant difference in pizza diameter.
```

*[9]:
```
#Print the t-statistic and p-value for additional context.
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")
```

```
T-statistic: 0.7228688704678063
P-value: 0.47223947245995
```

•[10]:
```
import pandas as pd
import numpy as np
from scipy.stats import shapiro, levene, ttest_ind

df = pd.read_csv(r"C:\Users\DELL\Downloads\pizzas - pizzas.csv")

# Display the first few rows of the dataset and shape to understand structure
print("Data Preview:\n", df.head(4))
print("Data Shape:", df.shape)
```

```
Data Preview:
    Making Unit 1  Making Unit 2
0         6.8090         6.7703
1         6.4376         7.5093
2         6.9157         6.7300
3         7.3012         6.7878
Data Shape: (35, 2)
```

•[11]:
```
# Define hypotheses for normality test
H0_normality = "Data is normal"
Ha_normality = "Data is not normal"
alpha = 0.05  # Significance level
```

[12]:
```
# Perform Shapiro-Wilk Test for normality on both units' data
p1 = round(shapiro(df['Making Unit 1'])[1], 2)  # P-value for Making Unit 1
p2 = round(shapiro(df['Making Unit 2'])[1], 2)  # P-value for Making Unit 2

# Print p-values for normality
print(f"\nNormality Test Results:")
print(f"P-value for Making Unit 1: {p1}")
print(f"P-value for Making Unit 2: {p2}")
```

```
Normality Test Results:
P-value for Making Unit 1: 0.32
P-value for Making Unit 2: 0.52
```

[13]:
```
# Conclude based on the p-value for normality test
if p1 > alpha:
    print(f"{p1} > {alpha}. We fail to reject the Null Hypothesis for Making Unit 1. {H0_normality}")
else:
    print(f"{p1} <= {alpha}. We reject the Null Hypothesis for Making Unit 1. {Ha_normality}")

if p2 > alpha:
    print(f"{p2} > {alpha}. We fail to reject the Null Hypothesis for Making Unit 2. {H0_normality}")
else:
    print(f"{p2} <= {alpha}. We reject the Null Hypothesis for Making Unit 2. {Ha_normality}")
```

```
0.32 > 0.05. We fail to reject the Null Hypothesis for Making Unit 1. Data is normal
0.52 > 0.05. We fail to reject the Null Hypothesis for Making Unit 2. Data is normal
```

[14]:
```
# Define hypotheses for the variance test
H0_variance = "Variances of Unit 1 and Unit 2 are approximately equal"
Ha_variance = "Variances of Unit 1 and Unit 2 are not equal"
```

[15]:
```
# Perform Levene's Test for equality of variances between both units
p_levene = round(levene(df['Making Unit 1'], df['Making Unit 2'])[1], 2)

# Print p-value for variance equality
print(f"\nVariance Test Results:")
print(f"P-value for Levene's Test: {p_levene}")
```

```
Variance Test Results:
P-value for Levene's Test: 0.42
```

[16]:
```
# Conclude based on the p-value from Levene's test
if p_levene > alpha:
    print(f"{p_levene} > {alpha}. We fail to reject the Null Hypothesis. {H0_variance}")
else:
    print(f"{p_levene} <= {alpha}. We reject the Null Hypothesis. {Ha_variance}")
```

```
0.42 > 0.05. We fail to reject the Null Hypothesis. Variances of Unit 1 and Unit 2 are approximately equal
```

[17]:
```
# Define hypotheses for the two-sample t-test
H0_ttest = "There is no significant difference in the average diameter."
Ha_ttest = "There is a significant difference in the average diameter."
```

[18]:
```
# Conduct a two-sample t-test for independent samples assuming equal variances
t_stat, p_value_ttest = ttest_ind(df['Making Unit 1'], df['Making Unit 2'], equal_var=(p_levene > alpha))

# Round and print the p-value from the t-test
p_value_ttest = round(p_value_ttest, 2)
print(f"\nT-test Results:")
print(f"P-value for T-test: {p_value_ttest}")
```

```
T-test Results:
P-value for T-test: 0.47
```

[19]:
```
# Conclude based on the p-value from the t-test
if p_value_ttest > alpha:
    print(f"{p_value_ttest} > {alpha}. We fail to reject the Null Hypothesis. {H0_ttest}")
else:
    print(f"{p_value_ttest} <= {alpha}. We reject the Null Hypothesis. {Ha_ttest}")
```

```
0.47 > 0.05. We fail to reject the Null Hypothesis. There is no significant difference in the average diameter.
```

Assignment: 5

Title: Mini Project

CERTIFICATION: