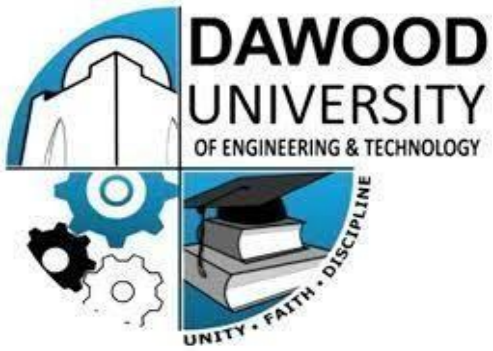
# Programming of Artificial Intelligence

## (Practical Manual)




**4th Semester, 2nd Year**

**BATCH -2023**

# BS ARTIFICIAL INTELLIGENCE

**DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY, KARACHI**

Dawood University Of Engineering and Technology, Karachi


# CERTIFICATE

This is to certify that **Abdul Rehman Waseem** with Roll # **23-AI-27** of **Batch 2023** has successfully completed all the labs prescribed for the course "**Programming of Artificial Intelligence**".

**Engr. Hamza Farooqui**
Lecturer
Department of AI

| S. NO. | TITLE OF EXPERIMENT |
|---|---|
| 1 | Introduction to Programming in Python |
| 2 | Object-Oriented Programming (OOP) in Python |
| 3 | Working with NumPy Arrays |
| 4 | Data Manipulation Using Pandas |
| 5 | R Programming using RStudio – Data Manipulation |
| 6 | Open Ended Lab - 1 |
| 7 | Data Visualization using Matplotlib |
| 8 | Data Visualization using Seaborn |
| 9 | Descriptive and Inferential Statistics using Python and R |
| 10 | Solving Ordinary Differential Equations (ODEs) using Python (SciPy) |
| 11 | Open Ended Lab – 2 |

# Lab No: 1

**Objective:** To introduce students to **Python programming** and develop their ability to write, understand, and execute basic Python code for data handling and problem solving.

**Why Python?**

- Python is a high-level, interpreted language widely used in AI, data science, and software development.
- It is known for its simple syntax, large community, and rich set of libraries.

**Core Concepts: -**

| Concept | Description |
|---|---|
| Variables & Data Types | int, float, str, bool, list, tuple, dict |
| Operators | Arithmetic (+, -, *, /), Comparison (==, !=) |
| Control Structures | if, elif, else, for, while |
| Functions | Using def to define reusable code blocks |
| Input/Output | input(), print() |
| Basic Libraries | math, random, datetime, etc. |

**Simple Example Code**

```
name = input("Enter your name: ")
print("Hello,", name)
num = int(input("Enter a number: "))
print("Square is:", num ** 2)
```

**Why It Matters in AI:**

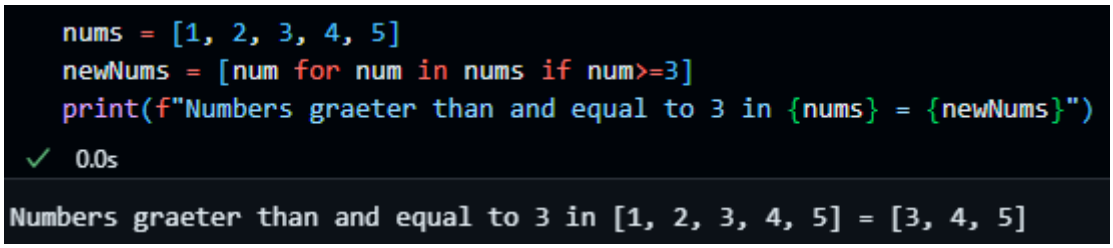
- Python is the primary language for AI frameworks like TensorFlow, PyTorch, and scikit-learn.
- Understanding Python is essential for implementing AI algorithms, preprocessing data, and building models.

**Tasks:**

a) Execute the following code in terms of ternary operator:

```
nums = [1,2,3,4,5]
newNums = []
for num in nums:
     if num >= 3:
          newNums.append(num)
print(newNums)
```

**SOLUTION:**




b) We define the usage of capitals in a word to be right when one of the following cases holds:

- All letters in this word are capitals, like "USA".
- All letters in this word are not capitals, like "leetcode".
- Only the first letter in this word is capital, like "Google".

Given a string word, return true if the usage of capitals in it is right.

**Example 1:**

**Input:** word = "USA"

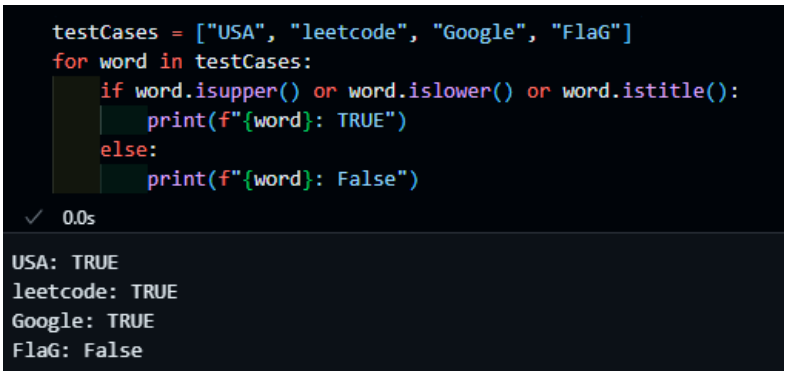
**Output:** true

**Example 2:**

**Input:** word = "FlaG"

**Output:** false

**SOLUTION:**



# Lab No: 2

**Objective:** To enable students to understand and apply **object-oriented programming** concepts in Python by defining classes, creating objects, using constructors, and accessing attributes and methods.

Object-Oriented Programming (OOP**)** is a programming paradigm centered around objects and classes, enabling code reuse, modularity, and real-world modeling.

**Key Concepts:**

- **Class**: A blueprint for creating objects. It defines attributes (variables) and methods (functions).
- **Object**: An instance of a class.
- **Constructor (__init__method)**: Automatically called when an object is created. It initializes attributes.
- **self keyword**: Refers to the current instance of the class.
- **Methods**: Functions defined inside a class that operate on the object's attributes.

**Benefits of Using Classes and Objects in Python**

- **Modularity**: Code is organized into objects with clear structure.
- **Reusability**: Classes can be reused and extended for multiple objects.
- **Maintainability**: Easier to update and manage object behavior.
- **Real-World Modeling**: Classes mirror real-world entities, making code intuitive and meaningful.

**Procedural vs. Object-Oriented Approach**

| **Feature** | **Procedural Programming** | **Object-Oriented Programming** |
|---|---|---|
| Structure | Organized around functions | Organized around objects |
| Reusability | Limited | High (through class reuse) |
| Data & Functions | Separate | Encapsulated together in objects |
| Flexibility | Low | High (supports inheritance, polymorphism) |

**Use Cases of OOP in Real Life:**

- **Student Management System**: Each student is an object with details and methods.
- **Banking System**: Accounts, users, and transactions are modeled as objects.
- **Game Development**: Characters, environments, and weapons as classes.

- **AI/ML Models**: Models are treated as objects with training, testing, and evaluation behaviors.

**Tasks:**

**Create a Simple Class**

- Define a class Student with attributes:
    - name, roll_no, marks
- Create a method display_info() to print student details.

SOLUTION

```
class Student:
    Tabnine | Edit | Test | Explain | Document
    def displayInfo(self):
        print(f"Name: {self.name},\nRoll No: {self.rollNo},\nMarks: {self.marks}")

s1 = Student()
s1.name = 'Sir Hamza Farooqui'
s1.rollNo = 1
s1.marks = 100
s1.displayInfo()
```

✓ 0.0s

```
Name: Sir Hamza Farooqui,
Roll No: 1,
Marks: 100
```

**Use Constructor to Initialize Objects**

- Use the __init__() method to initialize values while creating objects.

```
class Student:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, name, rollNo, marks):
        self.name = name
        self.rollNo = rollNo
        self.marks = marks

    Tabnine | Edit | Test | Explain | Document
    def displayInfo(self):
        print(f"Name: {self.name},\nRoll No: {self.rollNo},\nMarks: {self.marks}")

s2 = Student('Alice', 2, 99)
s2.displayInfo()
```

✓ 0.0s

```
Name: Alice,
Roll No: 2,
Marks: 99
```

**Create and Use Objects**

- Create at least two Student objects.
- Call the display_info() method for each object

```
class Student:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, name, rollNo, marks):
        self.name = name
        self.rollNo = rollNo
        self.marks = marks

    Tabnine | Edit | Test | Explain | Document
    def displayInfo(self):
        print(f"Name: {self.name},\nRoll No: {self.rollNo},\nMarks: {self.marks}\n")

s1 = Student('Ben Tenyson', 3, 89)
s2 = Student('Kevin 11', 4, 75)

print("Student 1 Info:")
s1.displayInfo()

print("Student 2 Info:")
s2.displayInfo()
```

✓ 0.0s

```
Student 1 Info:
Name: Ben Tenyson,
Roll No: 3,
Marks: 89

Student 2 Info:
Name: Kevin 11,
Roll No: 4,
```

# Lab No: 3

**Objective:** Write Python program to demonstrate useof **Numpy**

**Practical Significance: -**

Though Python is simple to learn language but it also very strong with its features. As mentioned earlier Python supports various built-in packages. Apart from built-in package user can also make their own packages i.e. User Defined Packages. **Numpy** is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. This practical will allow students to writea code.

**Minimum Theoretical Background: -**

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Steps for Installing numpy in windows OS

1. goto Command prompt
2. run command pip install numpy
3. open IDLE Python Interpreter
4. Check numpy is working or not

```
>>> import numpy
>>> import numpy as np
>>> a=np.array([10,20,30,40,50])
>>> print(a)
[10 20 30 40 50]
```

**Example: -**

```
>>>  student=np.dtype([('name','S20'),(' age' ,'i1'),(' marks','f4')])
>>>  a=np.array([(' Hamza',43,90),('Asad',38,80)],dtype= student)
>>> print(a)
[('Hamza', 43, 90.) ('Asad', 38, 80.)]
```

**Example: -**

```
>>> print(a)
[10 20 30 40 50 60]
>>> a.shape=(2,3)
>>> print(a) [[10 20 30]
 [40 50 60]]
>>> a.shape=(3,2)
>>> print(a) [[10 20]
 [30 40]
 [50 60]]
```

## Tasks: -

We'll use the **Iris Dataset**

- **Dataset Link**

Iris Dataset (CSV)

(Or get the CSV version from Kaggle)

It contains 150 rows and 5 columns:

- SepalLength
- SepalWidth
- PetalLength
- PetalWidth
- Class (species name)

---

### Task 1: Load the Dataset

1. Load the CSV file using np.genfromtxt() or np.loadtxt() (skip the header if needed).
2. Slice out the numerical columns into a separate NumPy array (4 features only).
3. Print the shape of the resulting NumPy array.

```
import numpy as np
dataSet = np.loadtxt('Iris.csv', delimiter=',', skiprows=1, usecols=(1, 2, 3, 4))
dataSet
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
...
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

```
print(f"Shape of the resulting NumPy array is {dataSet.shape}")
```

```
Shape of the resulting NumPy array is (150, 4)
```

## Task 2: Basic Array Operations

1. Compute the **mean**, **max**, and **min** for each column.
2. Calculate the **standard deviation** and **variance** for the dataset.
3. Normalize the data using Z-score normalization:

$z = x - \mu \sigma$ z = \frac{x - \mu}{\sigma}

```
print("mean per column:",np.mean(dataSet,axis=0))
print("max per column:",np.max(dataSet,axis=0))
print("min per column:",np.min(dataSet,axis=0))
```

```
mean per column: [5.84333333 3.054      3.75866667 1.19866667]
max per column: [7.9 4.4 6.9 2.5]
min per column: [4.3 2.  1.  0.1]
```

```
print(f"Standard Deviation per Column: {np.round(np.std(dataSet, axis=0),3)}")
print(f"Variance per Column: {np.round(np.var(dataSet, axis=0), 3)}")
```

✓ 0.0s

```
Standard Deviation per Column: [0.825 0.432 1.759 0.761]
Variance per Column: [0.681 0.187 3.092 0.579]
```

```
normalized_dataSet = (dataSet - np.mean(dataSet)) / np.std(dataSet)
normalized_dataSet.round(2)
```

✓ 0.0s

```
array([[ 0.83,  0.02, -1.05, -1.65],
       [ 0.73, -0.23, -1.05, -1.65],
       [ 0.63, -0.13, -1.1 , -1.65],
       [ 0.58, -0.18, -0.99, -1.65],
       [ 0.78,  0.07, -1.05, -1.65],
       [ 0.98,  0.22, -0.89, -1.55],
       [ 0.58, -0.03, -1.05, -1.6 ],
       [ 0.78, -0.03, -0.99, -1.65],
       [ 0.47, -0.29, -1.05, -1.65],
       [ 0.73, -0.18, -0.99, -1.7 ],
       [ 0.98,  0.12, -0.99, -1.65],
       [ 0.68, -0.03, -0.94, -1.65],
       [ 0.68, -0.23, -1.05, -1.7 ],
       [ 0.42, -0.23, -1.2 , -1.7 ],
       [ 1.18,  0.27, -1.15, -1.65],
       [ 1.13,  0.47, -0.99, -1.55],
       [ 0.98,  0.22, -1.1 , -1.55],
       [ 0.83,  0.02, -1.05, -1.6 ],
       [ 1.13,  0.17, -0.89, -1.6 ],
       [ 0.83,  0.17, -0.99, -1.6 ],
       [ 0.98, -0.03, -0.89, -1.65],
       [ 0.83,  0.12, -0.99, -1.55],
       [ 0.58,  0.07, -1.25, -1.65],
       [ 0.83, -0.08, -0.89, -1.5 ],
       [ 0.68, -0.03, -0.79, -1.65],
...
       [ 1.64, -0.23,  0.88, -0.59],
       [ 1.44, -0.49,  0.78, -0.79],
       [ 1.54, -0.23,  0.88, -0.74],
       [ 1.39, -0.03,  0.98, -0.59],
       [ 1.23, -0.23,  0.83, -0.84]])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

## Task 3: Indexing and Slicing

1. Extract only the **Sepal Length** column.
2. Get the values for the first 10 flowers.
3. Extract flowers where **Petal Length > 1.5**.

```
sepal_length = dataSet[:, 0]
print(sepal_length.reshape(150, 1))
```

✓ 0.0s

```
[[5.1]
 [4.9]
 [4.7]
 [4.6]
 [5. ]
 [5.4]
 [4.6]
 [5. ]
 [4.4]
 [4.9]
 [5.4]
 [4.8]
 [4.8]
 [4.3]
 [5.8]
 [5.7]
 [5.4]
 [5.1]
 [5.7]
 [5.1]
 [5.4]
 [5.1]
 [4.6]
 [5.1]
 [4.8]
...
 [6.3]
 [6.5]
 [6.2]
 [5.9]]
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

```
first_10 = dataSet[:10]
print(first_10)
```

✓ 0.0s

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

```
filtered_flowers = dataSet[dataSet[:, 2] > 1.5]
print(filtered_flowers)
```

✓ 0.0s

```
[[5.4 3.9 1.7 0.4]
 [4.8 3.4 1.6 0.2]
 [5.7 3.8 1.7 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [5.1 3.8 1.6 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
...
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

## # Task 4: Advanced Operations

1. Find the **Euclidean distance** between the first two rows.
2. Count how many flowers have **Sepal Width greater than the mean**.
3. Multiply two columns element-wise (e.g., SepalLength * PetalLength).

```
row1 = dataSet[0]
row2 = dataSet[1]

euclidean_distance = np.linalg.norm(row1 - row2)
print("Euclidean Distance:", euclidean_distance)
```

✓ 0.0s

```
Euclidean Distance: 0.5385164807134502
```

```
mean_sepal_width = np.mean(dataSet[:, 1])
count = np.sum(dataSet[:, 1] > mean_sepal_width)
print("Flowers with Sepal Width > Mean:", count)
```

✓ 0.0s

```
Flowers with Sepal Width > Mean: 67
```

```
result = dataSet[:, 0] * dataSet[:, 2]
print("SepalLength x PetalLength:\n", result.reshape(15, 10))
```

✓ 0.0s

```
SepalLength x PetalLength:
 [[ 7.14  6.86  6.11  6.9   7.    9.18  6.44  7.5   6.16  7.35]
 [ 8.1   7.68  6.72  4.73  6.96  8.55  7.02  7.14  9.69  7.65]
 [ 9.18  7.65  4.6   8.67  9.12  8.    8.    7.8   7.28  7.52]
 [ 7.68  8.1   7.8   7.7   7.35  6.    7.15  7.35  5.72  7.65]
 [ 6.5   5.85  5.72  8.    9.69  6.72  8.16  6.44  7.95  7.  ]
 [32.9  28.8  33.81 22.   29.9  25.65 29.61 16.17 30.36 20.28]
 [17.5  24.78 24.   28.67 20.16 29.48 25.2  23.78 27.9  21.84]
 [28.32 24.4  30.87 28.67 27.52 29.04 32.64 33.5  27.   19.95]
 [20.9  20.35 22.62 30.6  24.3  27.   31.49 27.72 22.96 22.  ]
 [24.2  28.06 23.2  16.5  23.52 23.94 23.94 26.66 15.3  23.37]
 [37.8  29.58 41.89 35.28 37.7  50.16 22.05 45.99 38.86 43.92]
 [33.15 33.92 37.4  28.5  29.58 33.92 35.75 51.59 53.13 30.  ]
 [39.33 27.44 51.59 30.87 38.19 43.2  29.76 29.89 35.84 41.76]
 [45.14 50.56 35.84 32.13 34.16 46.97 35.28 35.2  28.8  37.26]
 [37.52 35.19 29.58 40.12 38.19 34.84 31.5  33.8  33.48 30.09]]
```

## Task 5: Array Reshaping and Stacking

1. Reshape the array to simulate batches of size 30.
2. Stack two feature columns horizontally.
3. Create a boolean mask to filter rows with **Petal Width < 0.5**.

```
batches = dataSet.reshape(5, 30, dataSet.shape[1])
print("Batches shape:", batches.shape)
```

✓ 0.0s

```
Batches shape: (5, 30, 4)
```

```
stacked = np.column_stack((dataSet[:, 0], dataSet[:, 2]))
print("Horizontally stacked shape:", stacked.shape)
print(stacked[:5])
```

✓ 0.0s

```
Horizontally stacked shape: (150, 2)
[[5.1 1.4]
 [4.9 1.4]
 [4.7 1.3]
 [4.6 1.5]
 [5.  1.4]]
```

```
mask = dataSet[:, 3] < 0.5
filtered_rows = dataSet[mask]
print("Filtered rows where Petal Width < 0.5:\n", filtered_rows)
```

✓ 0.0s

```
Filtered rows where Petal Width < 0.5:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [4.8 3.4 1.9 0.2]
...
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]]
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

# Lab No: 4

**Objective:** To equip students with the skills to manipulate, clean, analyze, and preprocess structured datasets using the **Pandas library** in Python, preparing data for use in AI models and algorithms.

**Introduction to Pandas: -**

**Pandas** is a powerful Python library used for data manipulation and analysis. It provides two main data structures:

1. **Series** – One-dimensional labeled array.
2. **DataFrame** – Two-dimensional labeled data structure, similar to a table in a database or an Excel sheet.

Pandas is widely used in **AI and Machine Learning** pipelines for preprocessing, analyzing, and cleaning data before feeding it into models.

**Loading Data: -**

You can read structured data from various file formats:

```
# Load CSV file
df = pd.read_csv('data.csv')

# Load Excel file
df_excel = pd.read_excel('data.xlsx')

# Load from dictionary
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df_dict = pd.DataFrame(data)
```

---

**Exploring Data: -**

```
df.head()          # First 5 rows
df.tail()          # Last 5 rows
df.info()          # Data types and non-null values
df.describe()      # Statistical summary of numeric columns
```

---

**Data Selection: -**

```
df['Age']              # Select a single column
df[['Name', 'Age']]    # Select multiple columns
df.iloc[0]             # Row by index position
df.loc[0]              # Row by index label
```

---

**Filtering Data: -**

```
# Filter rows where Age > 25
df[df['Age'] > 25]

# Filter rows with multiple conditions
df[(df['Age'] > 25) & (df['Gender'] == 'Male')]
```

---

### Adding/Modifying Columns: -

```
# Add a new column
df['Is_Adult'] = df['Age'] >= 18

# Modify an existing column
df['Age'] = df['Age'] + 1
```

---

### Handling Missing Values: -

```
df.isnull().sum()         # Count missing values
df.dropna()               # Drop rows with any missing values
df.fillna(0)              # Fill missing values with 0
df.fillna(df.mean())      # Fill with mean of the column
```

---

### Grouping and Aggregation: -

```
# Group by Gender and calculate mean age
df.groupby('Gender')['Age'].mean()

# Count entries per category
df['Gender'].value_counts()
```

---

### Sorting and Reordering: -

```
df.sort_values(by='Age', ascending=False)  # Sort by Age descending
df.reset_index(drop=True, inplace=True)      # Reset index after sorting
```

---

### Dropping Columns and Rows: -

```
df.drop(columns=['Is_Adult'], inplace=True)  # Drop a column
df.drop(index=[0], inplace=True)               # Drop a row
```

---

### Merging and Joining DataFrames: -

```
# Merge on a common column
merged_df = pd.merge(df1, df2, on='ID')

# Concatenate along rows or columns
pd.concat([df1, df2], axis=0)  # Row-wise
pd.concat([df1, df2], axis=1)  # Column-wise
```

---

### Saving Data: -

```
df.to_csv('cleaned_data.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

---

### Why Pandas is Important in AI: -

- Prepares raw data for ML models.
- Enables feature engineering.
- Helps detect and handle missing or inconsistent values.
- Supports exploratory data analysis (EDA) and data cleaning.

**Tasks: -**

**Kaggle** IMDb Top 1000 Movies dataset

**Task 1: Load and Explore the Dataset**

1. Load the dataset.
2. Display the first 5 rows.
3. Check the data types of each column.
4. Find the number of rows and columns.
5. Check for missing values.

```
import pandas as pd

dataSet = pd.read_csv("imdb_top_1000.csv")

dataSet
```

Python

| | Poster_Link | Series_Title | Released_Year | Certificate | Runtime | Genre | IMDB_Rating | Overview | Meta_score | Director | Star1 | Star2 | Star3 | Star4 | No_of_Vot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | https://m.media-amazon.com/images/M/MV5BMDFkYT... | The Shawshank Redemption | 1994 | A | 142 min | Drama | 9.3 | Two imprisoned men bond over a number of years... | 80.0 | Frank Darabont | Tim Robbins | Morgan Freeman | Bob Gunton | William Sadler | 23431 |

```
first_five_rows = dataSet.head()
first_five_rows
```

Python

| | Poster_Link | Series_Title | Released_Year | Certificate | Runtime | Genre | IMDB_Rating | Overview | Meta_score | Director | Star1 | Star2 | Star3 | Star4 | No_of_Votes | Gr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | https://m.media-amazon.com/images/M/MV5BMDFkYT... | The Shawshank Redemption | 1994 | A | 142 min | Drama | 9.3 | Two imprisoned men bond over a number of years... | 80.0 | Frank Darabont | Tim Robbins | Morgan Freeman | Bob Gunton | William Sadler | 2343110 | 28,341,4 |
| 1 | https://m.media-amazon.com/images/M/MV5BM2MyNj... | The Godfather | 1972 | A | 175 min | Crime, Drama | 9.2 | An organized crime dynasty's aging patriarch t... | 100.0 | Francis Ford Coppola | Marlon Brando | Al Pacino | James Caan | Diane Keaton | 1620367 | 134,966,4 |
| 2 | https://m.media-amazon.com/images/M/MV5BMTMxNT... | The Dark Knight | 2008 | UA | 152 min | Action, Crime, Drama | 9.0 | When the menace known as the Joker wreaks havo... | 84.0 | Christopher Nolan | Christian Bale | Heath Ledger | Aaron Eckhart | Michael Caine | 2303232 | 534,858,4 |
| 3 | https://m.media-amazon.com/images/M/MV5BMWMwMG... | The Godfather: Part II | 1974 | A | 202 min | Crime, Drama | 9.0 | The early life and career of Vito Corleone in ... | 90.0 | Francis Ford Coppola | Al Pacino | Robert De Niro | Robert Duvall | Diane Keaton | 1129952 | 57,300,0 |
| 4 | https://m.media-amazon.com/images/M/MV5BMWU4N2... | 12 Angry Men | 1957 | U | 96 min | Crime, Drama | 9.0 | A jury holdout attempts to prevent a | 96.0 | Sidney Lumet | Henry Fonda | Lee J. Cobb | Martin Balsam | John Fiedler | 689845 | 4,360,0 |

```
print("DataTypes of Each Column:")
dataSet.dtypes
```

```
DataTypes of Each Column:

Poster_Link       object
Series_Title      object
Released_Year     object
Certificate       object
Runtime           object
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross             object
dtype: object
```

```
rows, columns = dataSet.shape
print(f"There are {rows} Rows and {columns} Columns")
```

```
There are 1000 Rows and 16 Columns
```

```
missingValues = dataSet.isnull().sum()
print(f"Coulmns with number of missing Values is given by:\n{missingValues}")
```

```
Coulmns with number of missing Values is given by:
Poster_Link         0
Series_Title        0
Released_Year       0
Certificate       101
Runtime             0
Genre               0
IMDB_Rating         0
Overview            0
Meta_score        157
Director            0
Star1               0
Star2               0
Star3               0
Star4               0
No_of_Votes         0
Gross             169
dtype: int64
```

**Task 2: Data Cleaning**

1. Remove any duplicate rows if present.
2. Fill missing values in the dataset (e.g., replace missing ratings with the mean rating).
3. Convert the Runtime column (which is in minutes as a string, e.g., "120 min") to an integer.

```
no_duplicate_row = dataSet.drop_duplicates()
no_duplicate_row
```

```
dataSet["IMDB_Rating"] = dataSet["IMDB_Rating"].fillna(dataSet["IMDB_Rating"].mean())
dataSet
```

```
dataSet["Runtime in Int"] = dataSet["Runtime"].str.replace(" min", "", regex=False)
dataSet["Runtime in Int"] = dataSet["Runtime in Int"].astype(float).fillna(0).astype(int)
```

**Task 3: Data Filtering & Sorting**

1. Find all movies with an **IMDb rating greater than 8.5**.
2. List movies that belong to the **Action or Sci-Fi genre**.
3. Find movies that were released **between 2000 and 2015**.
4. Sort the dataset based on **IMDb rating in descending order**.

```
movies_with_rating_greaterThan_8_5 = dataSet[dataSet["IMDB_Rating"]>8.5]
movies_with_rating_greaterThan_8_5
```

```
movies_belongs_Action_Sci_fi = dataSet[dataSet["Genre"].str.contains("Action|Sci-Fi", na=False)]
movies_belongs_Action_Sci_fi
```

```
dataSet["Released_Year"] = pd.to_numeric(dataSet["Released_Year"], errors='coerce')
movies_bw_2000_2015 = dataSet[(dataSet["Released_Year"] >= 2000) & (dataSet["Released_Year"] <= 2015)]
movies_bw_2000_2015
```

```
sorted_Dataset = dataSet.sort_values(by="IMDB_Rating",ascending=False)
sorted_Dataset
```

**Data Aggregation & Grouping**

1. Find the **average IMDb rating** for each genre.
2. Determine which **year had the most movies released**.
3. Find the **top 5 directors** who have directed the most movies in the dataset.

```
average_IMDb_ratings_for_each_gener = dataSet.groupby('Genre')['IMDB_Rating'].mean().round(3)
average_IMDb_ratings_for_each_gener
```

```
Genre
Action, Adventure                8.180
Action, Adventure, Biography     7.900
Action, Adventure, Comedy        7.910
Action, Adventure, Crime         7.600
Action, Adventure, Drama         8.150
                                 ...
Mystery, Romance, Thriller       8.300
Mystery, Sci-Fi, Thriller        7.800
Mystery, Thriller                7.978
Thriller                         7.800
Western                          8.350
Name: IMDB_Rating, Length: 202, dtype: float64
```

```
most_released_movie = dataSet.groupby('Released_Year')['IMDB_Rating'].count()
top_year = most_released_movie.sort_values(ascending=False).idxmax()
print(f"In {top_year}, Most movies had released.")
```

```
In 2014.0, Most movies had released.
```

```
top_five_directors = dataSet['Director'].value_counts().head()
top_five_directors
```

```
Director
Alfred Hitchcock    14
Steven Spielberg    13
Hayao Miyazaki      11
Akira Kurosawa      10
Martin Scorsese     10
Name: count, dtype: int64
```

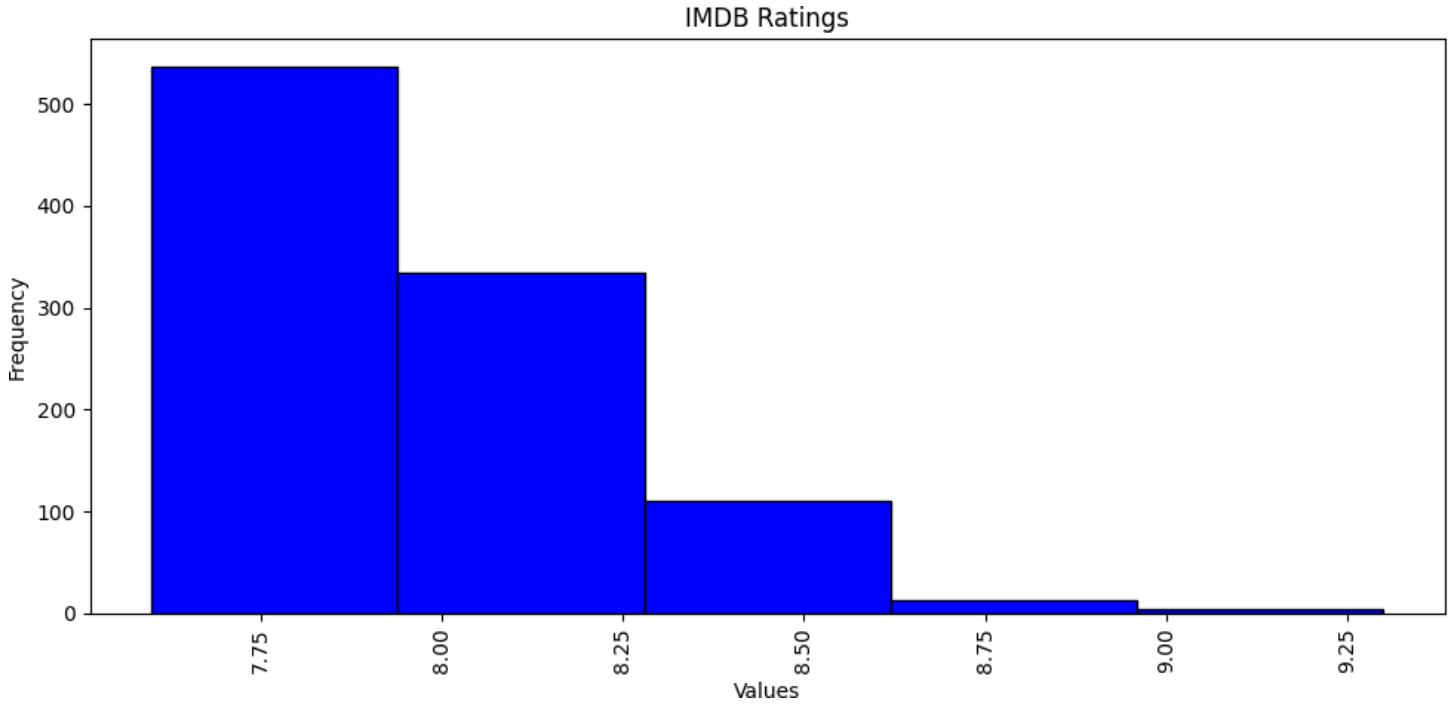
**Visualization (Optional)**

**Matplotlib** or **Seaborn**

1. Plot a histogram of IMDb ratings.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
plt.hist(dataSet['IMDB_Rating'], bins = 5, color='blue', edgecolor='black')
plt.title("IMDB Ratings")
plt.xlabel("Values")
plt.ylabel("Frequency")
plt.xticks(rotation=90)
plt.tight_layout()
```

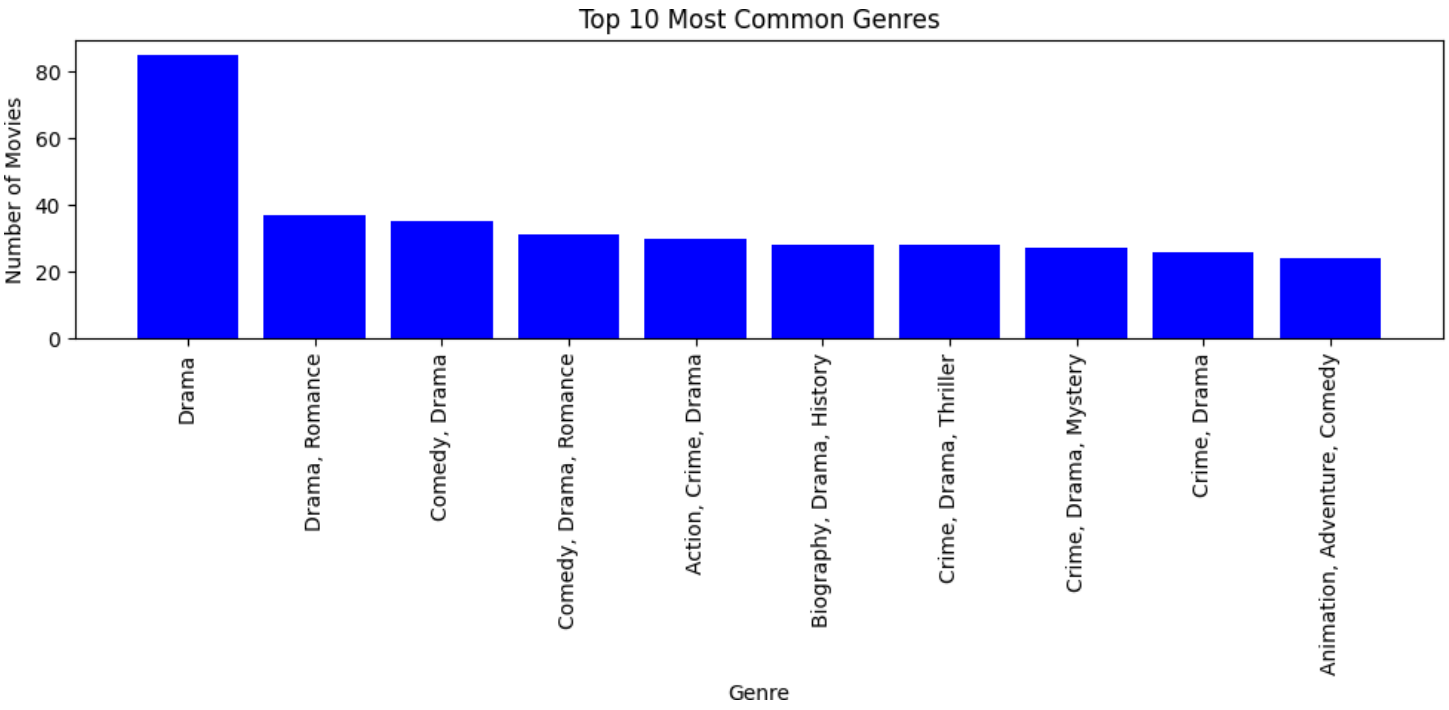
✓ 0.8s



2. Create a bar chart showing the **top 10 genres** with the most movies.

```
top_genre = dataSet['Genre'].value_counts().head(10)

plt.figure(figsize=(10, 5))
plt.bar(top_genre.index, top_genre.values, color='blue')
plt.xticks(rotation=90)
plt.xlabel("Genre")
plt.ylabel("Number of Movies")
plt.title("Top 10 Most Common Genres")
plt.tight_layout()
plt.show()
```

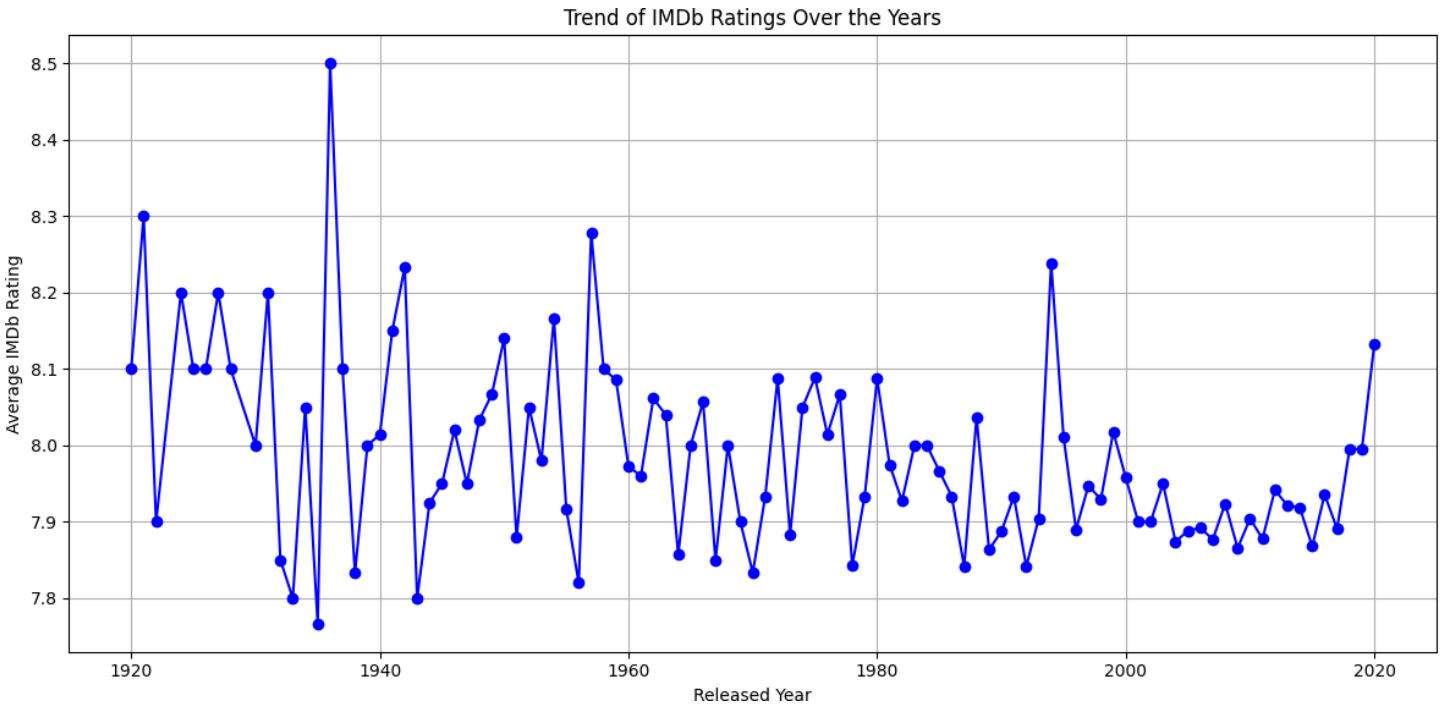
✓ 0.3s



3. Visualize the **trend of IMDb ratings over the years**.

```
dataSet['Released_Year'] = pd.to_numeric(dataSet['Released_Year'], errors='coerce')

trend_data = dataSet.dropna(subset=['IMDB_Rating', 'Released_Year'])

avg_rating_by_year = trend_data.groupby('Released_Year')['IMDB_Rating'].mean().sort_index()

plt.figure(figsize=(12, 6))
plt.plot(avg_rating_by_year.index, avg_rating_by_year.values, marker='o', linestyle='-', color='blue')
plt.xlabel("Released Year")
plt.ylabel("Average IMDb Rating")
plt.title("Trend of IMDb Ratings Over the Years")
plt.grid(True)
plt.tight_layout()
plt.show()
```

✓ 0.3s



# Lab No: 5

**Objective:** To enable students to understand and implement fundamental data manipulation operations in **R using RStudio** and tidyverse.

**1. Difference between R and Python**

| | R | Python |
|---|---|---|
| **Primary Use:** | Statistical computing, Data analysis | General-purpose, Machine Learning |
| **Libraries:** | tidyverse, dplyr, ggplot2 | pandas, NumPy, scikit-learn |
| **IDE:** | RStudio | Jupyter Notebook, VS Code |
| **Syntax:** | More functional | More object-oriented |
| **Learning Curve:** | Steeper for beginners | Beginner-friendly |

---

**2. Packages and tidyverse**

- Packages are collections of R functions and datasets.
- tidyverse is a suite of packages (like dplyr, ggplot2, readr, etc.) for data science tasks.

```
install.packages("tidyverse")  # Install
library(tidyverse)              # Load
```

---

**3. Vectors in R**

- A vector is a basic data structure that contains elements of the same type.

```
numeric_vec <- c(1, 2, 3)
char_vec <- c("a", "b", "c")
```

---

**4. Importing Datasets**

```
data <- read.csv("data.csv")
```

---

**5. Data Manipulation Functions**

| Function | Description |
|---|---|
| filter() | Select rows based on condition |
| select() | Choose specific columns |
| mutate() | Create or transform columns |
| na.omit() | Remove rows with missing values |
| mean() | Calculate average |
| median() | Calculate median value |

**Example:**

```
data_clean <- data %>%
  filter(age > 20) %>%
  select(name, age, salary) %>%
  mutate(salary_k = salary / 1000) %>%
```

```
  na.omit()
```

---

**6. The Pipe Operator %>%**

- The pipe operator passes the result of one function to the next.

```
data %>%
  filter(gender == "Female") %>%
  select(name, score) %>%
  summarise(avg_score = mean(score))
```

**Tasks: -**

**Part A: Data Preparation**

1. Load the student_performance.csv dataset and remove rows with missing Remarks or Passed values.

```
student_performance <- read_csv("student_performance.csv")

view(student_performance)
cleanData <- subset(student_performance, !is.na(Remarks) & !is.na(Passed))
view(cleanData)
```

2. Create a new column Total_Score (sum of the three subject scores).

```
sumOfMarks <- mutate(cleanData, TotalScore = Math_Score + English_Score + Science_Score)
view(sumOfMarks)
```

3. Create a new column Performance_Level:
    - "Excellent" if Total_Score ≥ 240
    - "Good" if 200 ≤ Total_Score < 240
    - "Average" if 150 ≤ Total_Score < 200
    - "Poor" otherwise

```
sumOfMarks <- mutate(cleanData, TotalScore = Math_Score + English_Score + Science_Score)
view(sumOfMarks)

performanceLevel <- mutate(sumOfMarks, performance_level = ifelse(
  TotalScore >= 240, "Excellent",
  ifelse(TotalScore >= 200 & TotalScore < 240, "Good",
         ifelse(TotalScore >= 150 & TotalScore < 200, "Average", "Poor")
  )
))

View(performanceLevel)
```

---

**Part B: Data Analysis**

1. Count how many students fall into each Performance_Level.

```
eachPerformanceLevel <- performanceLevel %>% select(Student_ID, performance_level) %>%
  group_by(performance_level) %>% summarise(Category = n())
view(eachPerformanceLevel)
```

2. Compute the average Study_Hours_Per_Week and Attendance_Percentage for each performance level.

```
Study_Hours_Per_Week <- performanceLevel %>%
                        select(Student_ID, performance_level, Study_Hours_Per_Week, Attendance_Percentage) %>%
                        group_by(performance_level) %>%
                        summarise(Average_Study_Hours_Per_Week = mean(Study_Hours_Per_Week, na.rm = TRUE),
                                  Average_Attendance_Percentage = mean(Attendance_Percentage))
view((Study_Hours_Per_Week))
```

3. Group by Gender and School_Type, and compute:
   - Mean Total_Score
   - Pass percentage (Passed == "Yes")

```
gender_school_type <- sumOfMarks %>% select(School_Type, Gender, TotalScore) %>%
                      group_by(School_Type, Gender) %>%
                      summarise(Average_Total_Score = mean(TotalScore))
view(gender_school_type)


Passed_Percentage <-  sumOfMarks %>%
                      select(School_Type, Gender, TotalScore, Passed) %>%
                      group_by(School_Type, Gender) %>%
                      summarise(
                        Average_Total_Score = mean(TotalScore, na.rm = TRUE),
                        Passed_Percentage = mean(Passed == "Yes") * 100
                      )
View(Passed_Percentage)
```

4. Find the top 5 students with the highest Study_Hours_Per_Week **who did not pass**.

```
topNotPassedStudent = sumOfMarks %>% filter(Passed != 'Yes') %>%
                      arrange(desc(Study_Hours_Per_Week)) %>% slice_head(n=5)
view(topNotPassedStudent)
```

**Part C: Conditional and Logical Operations**

1. Create a new column Study_Efficiency:

   Study_Efficiency = Total_Score / Study_Hours_Per_Week

   - Filter students with Study_Efficiency < 10 and Passed == "Yes".

```
Study_Efficiency <- sumOfMarks %>% mutate(Study_Efficiency = TotalScore / Study_Hours_Per_Week)
view(Study_Efficiency)

lessThanTen = Study_Efficiency %>% filter(Study_Efficiency < 10) %>% filter(Passed == "Yes")
view(lessThanTen)
```

2. Add a column Eligible_for_Scholarship:

   - TRUE if Total_Score ≥ 230 and Attendance_Percentage > 90, else FALSE

```
Eligible_for_Scholarship = sumOfMarks %>%
                           mutate(Eligible_for_Scholarship = ifelse(TotalScore >= 230 & Attendance_Percentage > 90,
                                                                    TRUE, FALSE))
view(Eligible_for_Scholarship)
```

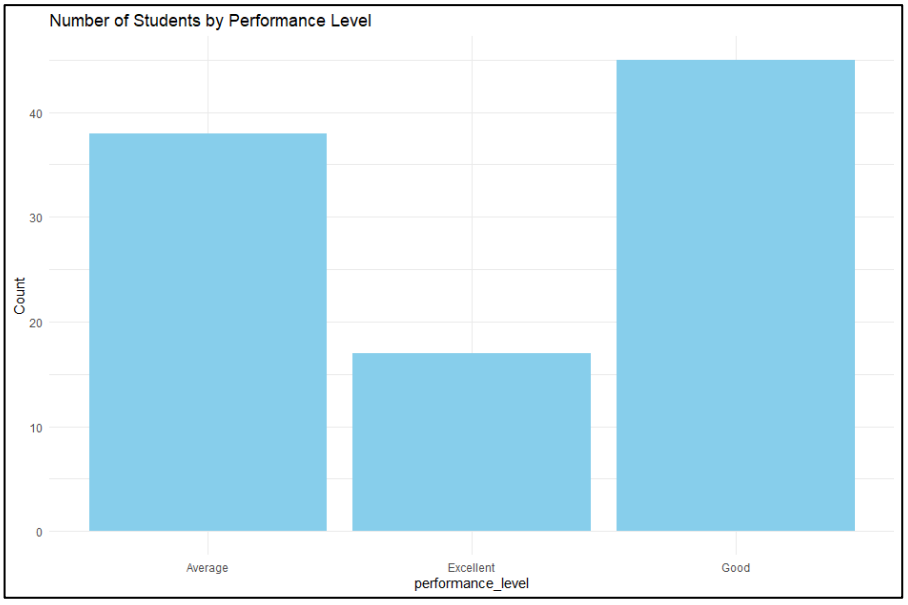
---

**Part D: Data Visualization**

Using ggplot2:

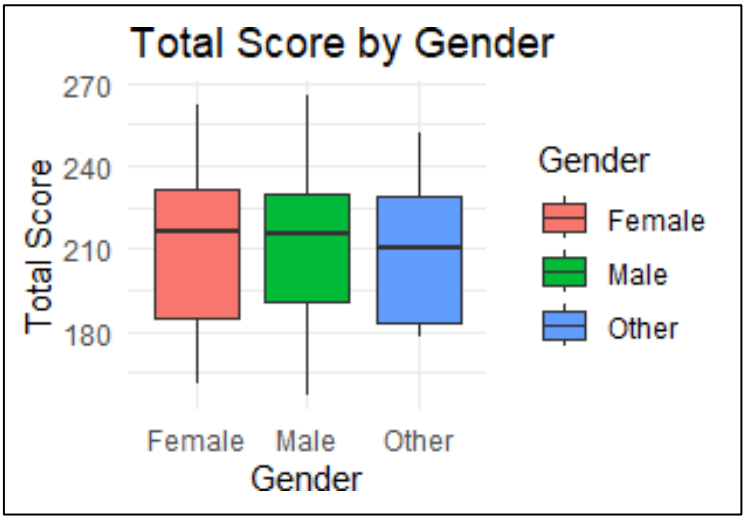
1. Bar chart showing number of students in each Performance_Level

```
install.packages("labeling")

install.packages("ggplot2")
library(ggplot2)


ggplot(performanceLevel, aes(x = performance_level)) +
  geom_bar(fill = "skyblue") +
  labs(title = "Number of Students by Performance Level",
       x = "performance_level", y = "Count") +
  theme_minimal()
```



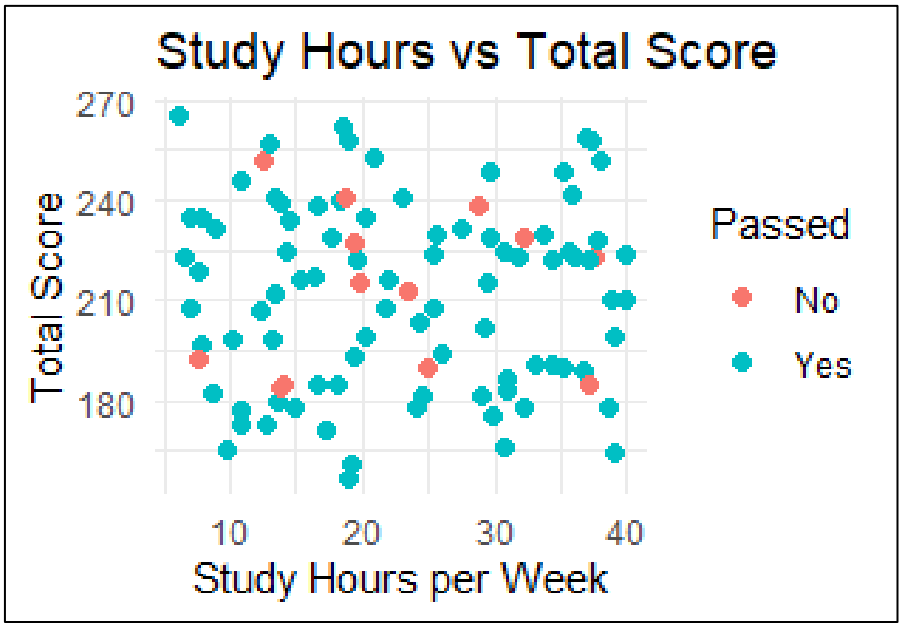
2. Boxplot comparing Total_Score across Gender

```
ggplot(performanceLevel, aes(x = Gender, y = TotalScore, fill = Gender)) +
  geom_boxplot() +
  labs(title = "Total Score by Gender",
       x = "Gender", y = "Total Score") +
  theme_minimal()
```



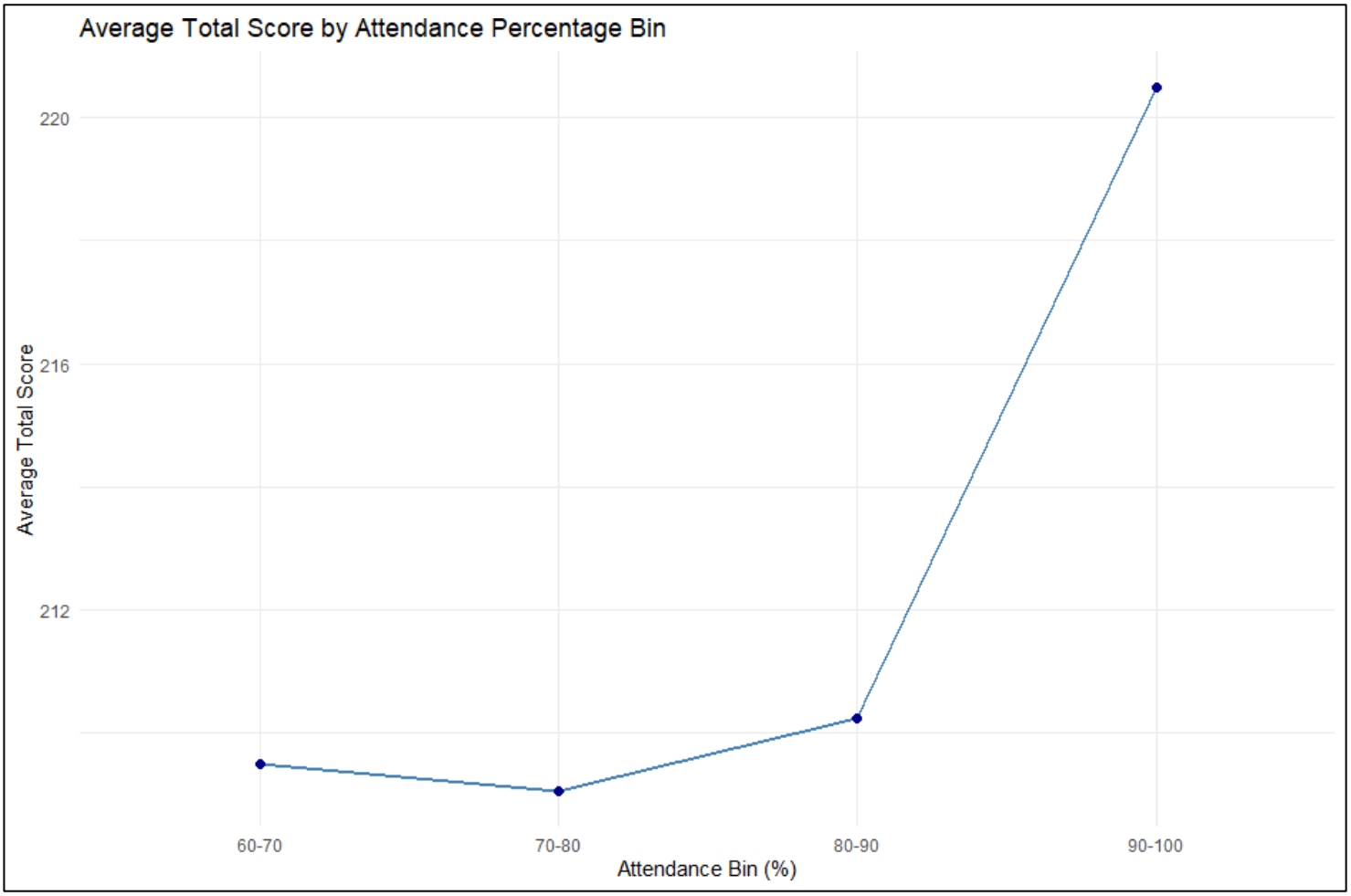

3. Scatter plot of Study_Hours_Per_Week vs Total_Score, colored by Passed

```
ggplot(performanceLevel, aes(x = Study_Hours_Per_Week, y = TotalScore, color = Passed)) +
  geom_point(size = 2) +
  labs(title = "Study Hours vs Total Score",
       x = "Study Hours per Week", y = "Total Score") +
  theme_minimal()
```



4. Line chart showing average Total_Score by Attendance_Percentage bins (use cut() to bin attendance)

```r
performanceLevel$Attendance_Bin <- cut(performanceLevel$Attendance_Percentage,
                        breaks = seq(0, 100, by = 10),
                        labels = paste0(seq(0, 90, by = 10), "-", seq(10, 100, by = 10)))


library(dplyr)
avg_score_by_bin <- performanceLevel %>%
  group_by(Attendance_Bin) %>%
  summarise(Average_Score = mean(TotalScore, na.rm = TRUE))

ggplot(avg_score_by_bin, aes(x = Attendance_Bin, y = Average_Score, group = 1)) +
  geom_line(color = "steelblue", size = 1) +
  geom_point(color = "darkblue", size = 2) +
  labs(title = "Average Total Score by Attendance Percentage Bin",
       x = "Attendance Bin (%)", y = "Average Total Score") +
  theme_minimal()
```



## QUESTIONS:

- **Are high study hours always linked to passing?**

  **Ans**: Not always.
  While mean and median study hours are usually higher among students who passed, there is overlap**,** some students with low study hours passed**,** and some with high hours still failed.

- **Do school type and gender impact overall performance?**

  **Ans:** Yes, to some extent.

  - **Gender** may show slight differences in mean Total_Score**,** but not always statistically significant.
  - **School type** (e.g., private vs public) often shows a stronger difference**,** with private school students sometimes having higher average scores**.**

- **Which factors best predict scholarship eligibility?**

  **Ans:** The best predictors are usually:

  - Total_Score (strongest predictor)
  - Attendance_Percentage
  - Study_Hours_Per_Week**.**

**Lab No: 6**

```
install.packages("tidyverse")

library(dplyr)

library(readr)


data <- read_csv("C:/Users/pc148/OneDrive/Desktop/customer_churn.csv")


names(data)


# 1. Import the dataset and display the number of customers who have churned (Churn ==

"Yes")


churned_count <- data %>%

 filter(Churn == "Yes") %>%

nrow()


print(paste("Number of customers who have churned:", churned_count))


# 2. Create a new column ChargeGap as TotalCharges - (MonthlyCharges * Tenure). Handle

#any NAs appropriately.
```

```
data <- data %>%

mutate(

ChargeGap = TotalCharges - (MonthlyCharges * Tenure)

) %>%

filter(!is.na(ChargeGap))


View(data)


# 3. Filter customers who have been with the company for more than 2 years (Tenure
> 24)

#and are still active (Churn == "No").


active_2plus_years <- data %>%

filter(Tenure > 24, Churn == "No")


print("Active customers with > 2 years tenure:")

print(active_2plus_years)


# 4. Find the average MonthlyCharges for each ContractType (e.g., "Month-to-Month",
"One

#year", "Two year").
```

```
avg_monthly_by_contract <- data %>%
group_by(ContractType) %>%
summarise(avg_monthly = mean(MonthlyCharges, na.rm = TRUE))

print("Average Monthly Charges by Contract Type:")
print(avg_monthly_by_contract)

# 5. Categorize customers based on Age:

data <- data %>%
mutate(AgeGroup = case_when(
Age < 25 ~ "Youth",
Age >= 25 & Age <= 55 ~ "Adult",
Age > 55 ~ "Senior",
TRUE ~ "Unknown"
))

View(data)

# 6. Find the top 5 cities with the highest number of churned customers.

top_5_churn_cities <- data %>%
```

```
filter(Churn == "Yes") %>%
count(City, sort = TRUE) %>%
head(5)

print("Top 5 cities with highest number of churned customers:")
print(top_5_churn_cities)

# 7. Extract only the names and cities of customers whose TotalCharges > 3000, are on a
#Month-to-Month contract, and have churned.

high_value_churners <- data %>%
filter(
TotalCharges > 3000,
ContractType == "Month-to-month",
Churn == "Yes"
) %>%
select(Name, City)

print("High value churned customers on month-to-month:")
print(high_value_churners)
```

**# 8. Create a table that shows the average tenure and total revenue (sum(TotalCharges)) for**

**#each ContractType.**

**contract_summary <- data %>%**

**group_by(ContractType) %>%**

**summarise(**

**avg_tenure = mean(Tenure, na.rm = TRUE),**

**total_revenue = sum(TotalCharges, na.rm = TRUE)**

**)**

**print("Contract Summary: Average Tenure and Total Revenue")**

**print(contract_summary)**

## Lab No: 7

**Objective:** To enable students to understand and implement basic to intermediate data visualization techniques using **Matplotlib in Python**

Matplotlib is a powerful 2D plotting library in Python. The module matplotlib.pyplot provides a MATLAB-like interface for creating visualizations.

```
import matplotlib.pyplot as plt
```

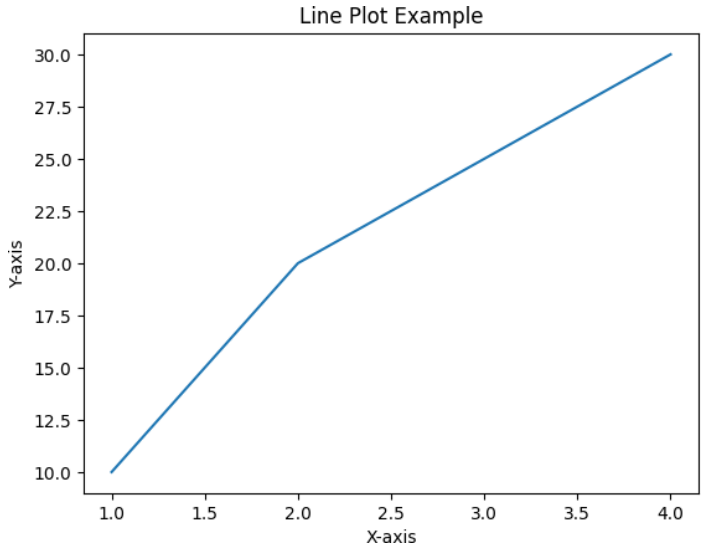
---

**1. Line Plot**

- Used to display information as a series of data points connected by straight lines.

```
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.title("Line Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
```

```
plt.show()
```

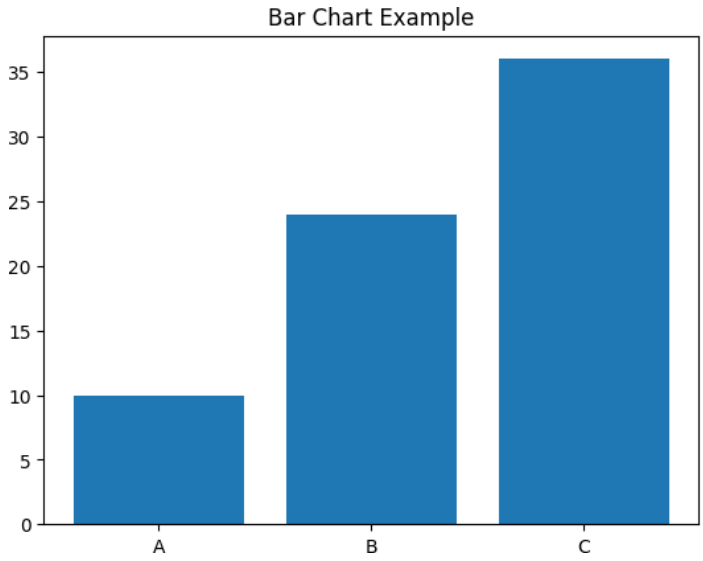



---

### 3. Bar Chart

- Represents categorical data with rectangular bars.

```
categories = ['A', 'B', 'C']
values = [10, 24, 36]
plt.bar(categories, values)
plt.title("Bar Chart Example")
plt.show()
```

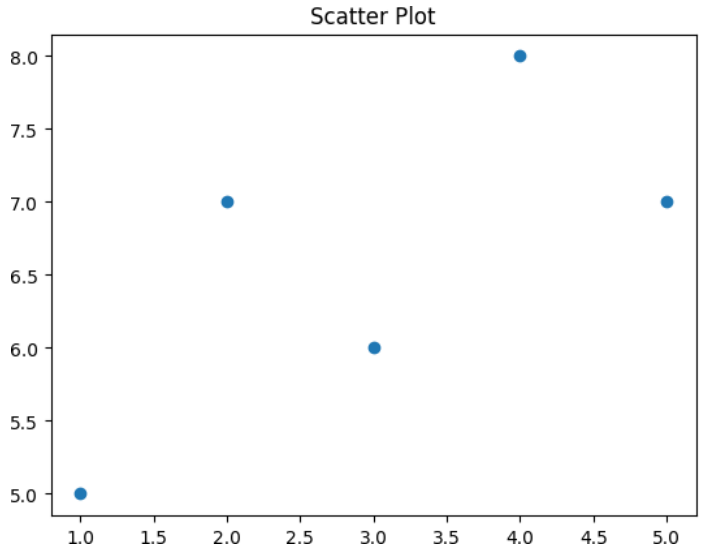



---

**4. Scatter Plot**

- Used to show relationships between two numerical variables.

```
x = [1, 2, 3, 4, 5]
y = [5, 7, 6, 8, 7]
plt.scatter(x, y)
plt.title("Scatter Plot")
plt.show()
```

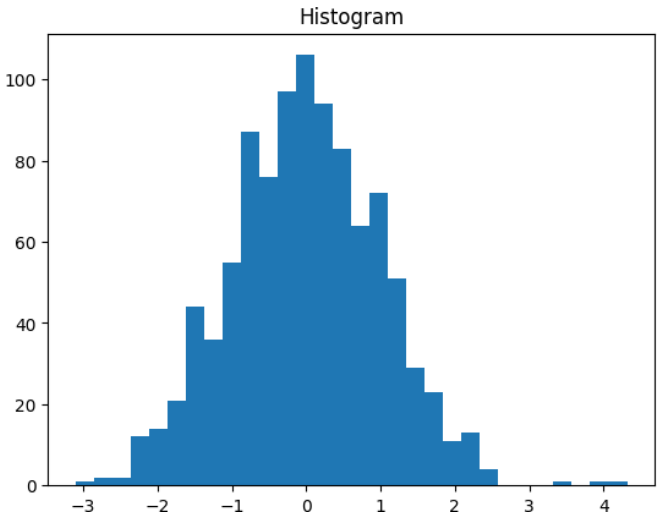



---

**5. Histogram**

- Used to show the distribution of a dataset.

```
import numpy as np
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.title("Histogram")
plt.show()
```

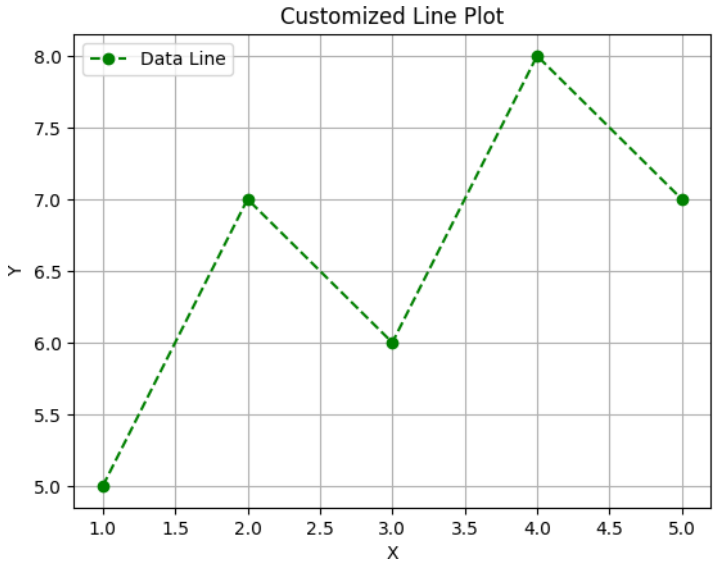



---

**6. Customizing Plots**

- Add **labels**, **titles**, **legends**, **grid**, and change **line styles** or **colors** to enhance visualization.

```
plt.plot(x, y, color='green', linestyle='--', marker='o', label='Data
Line')
plt.title("Customized Line Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid(True)
plt.show()
```




---

**7. Saving Plots**

- You can save the figure as an image using:

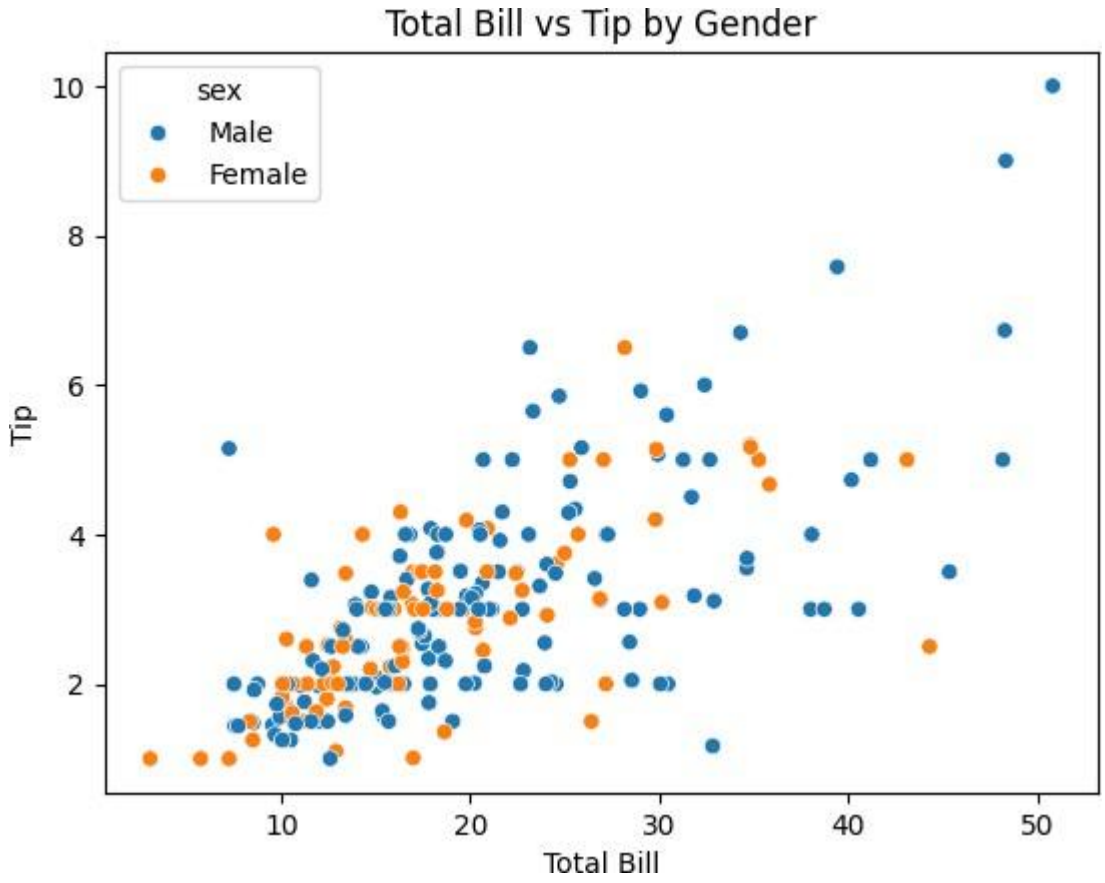
```
plt.savefig("plot.png")
```

**Tasks: -**

Use the built-in **tips** dataset from Seaborn or load another dataset like **Iris** or a **custom CSV file** with numerical and categorical variables.

1. **Scatter Plot**
    - Plot total_bill vs tip with appropriate axis labels and title.
    - Add color to points based on sex.

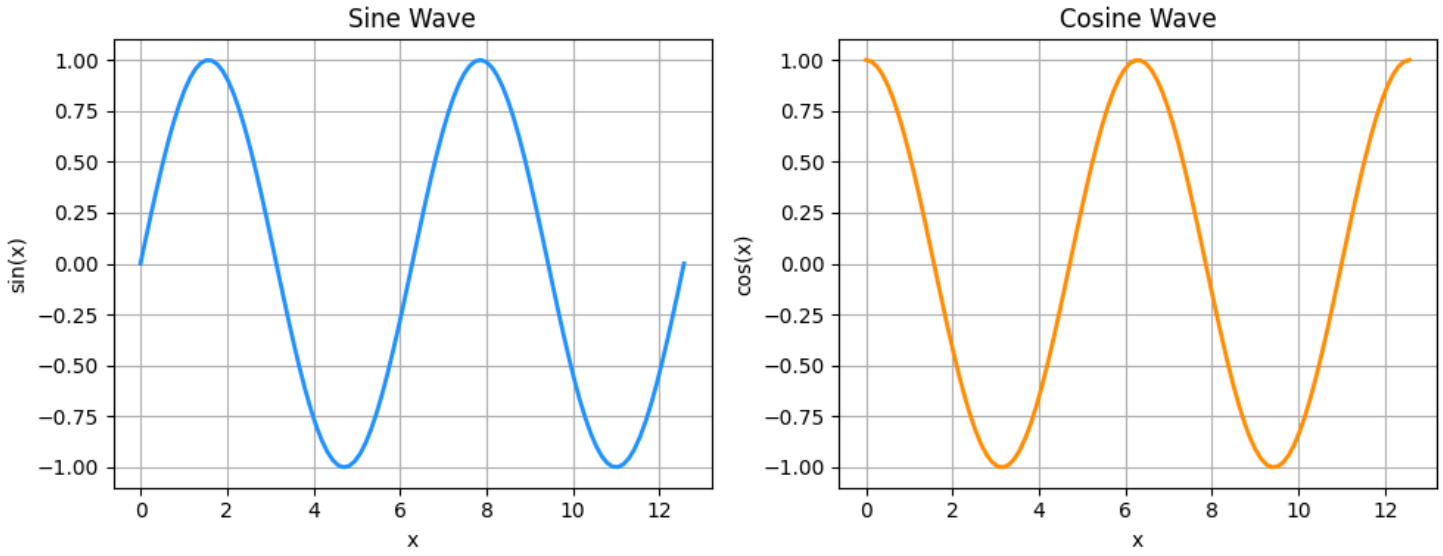
```
import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset("tips")

sns.scatterplot(data=tips, x="total_bill", y="tip", hue="sex")
plt.title("Total Bill vs Tip by Gender")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.show()
```




z

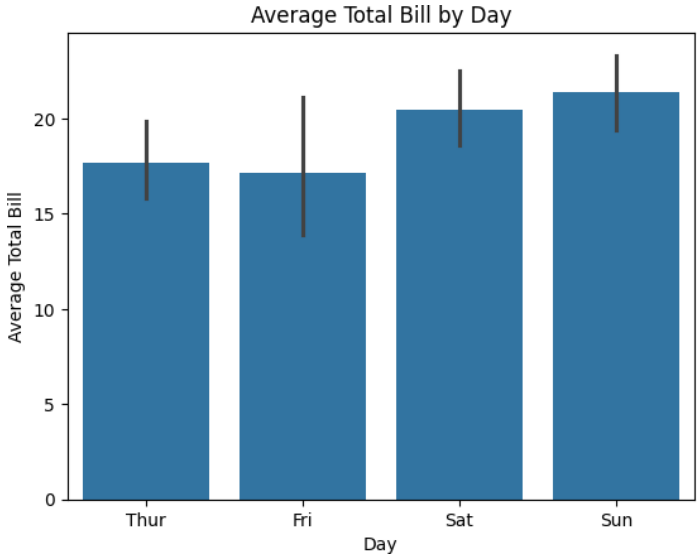
2. **Subplots**
    - Create **two subplots** side by side:
        - First plot: Line plot of sine wave.
        - Second plot: Line plot of cosine wave.
        - Use numpy to generate x-values from 0 to 2π.

```
import numpy as np

pi = 4 * np.pi
x = np.linspace(0, pi, 100)
y = np.sin(x)
y1 = np.cos(x)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(x, y, color='dodgerblue', linewidth=2)
plt.title("Sine Wave")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(x, y1, color='darkorange', linewidth=2)
plt.title("Cosine Wave")
plt.xlabel("x")
plt.ylabel("cos(x)")
plt.grid(True)

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



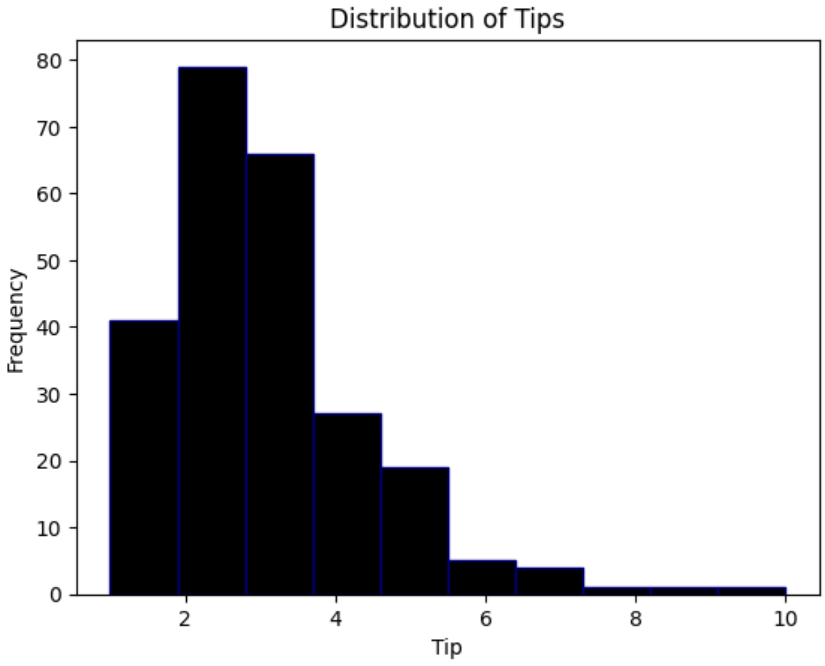
3. **Bar Plot**
    - Plot average total_bill for each day using a bar plot.

```
sns.barplot(data=tips, x="day", y="total_bill", estimator='mean')
plt.title("Average Total Bill by Day")
plt.xlabel("Day")
plt.ylabel("Average Total Bill")
plt.show()
```



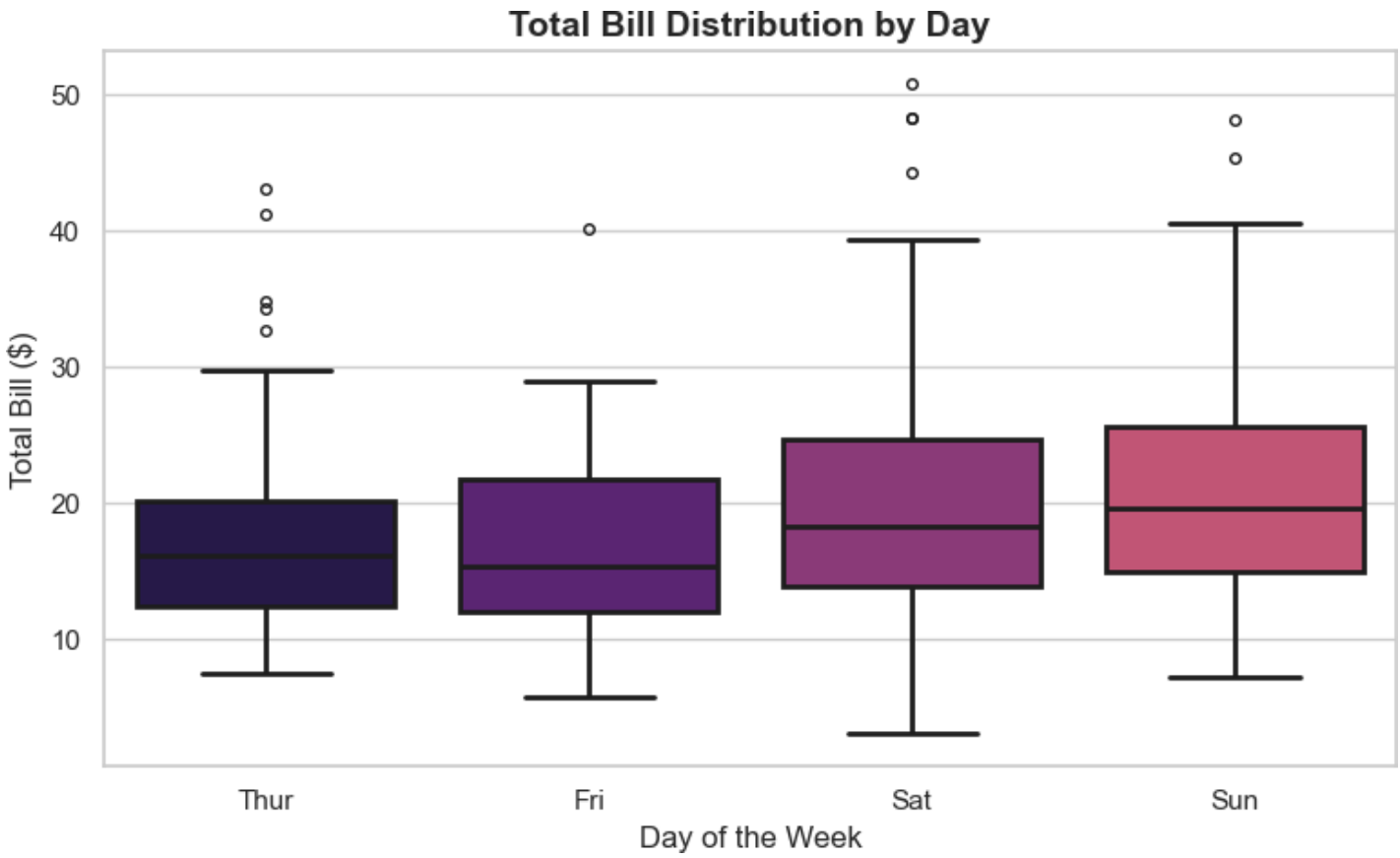

4. **Histogram**
    - Create a histogram of tip values with bins=10 and appropriate labels.

```
plt.hist(tips["tip"], bins=10, color='black', edgecolor='navy')
plt.title("Distribution of Tips")
plt.xlabel("Tip")
plt.ylabel("Frequency")
plt.show()
```



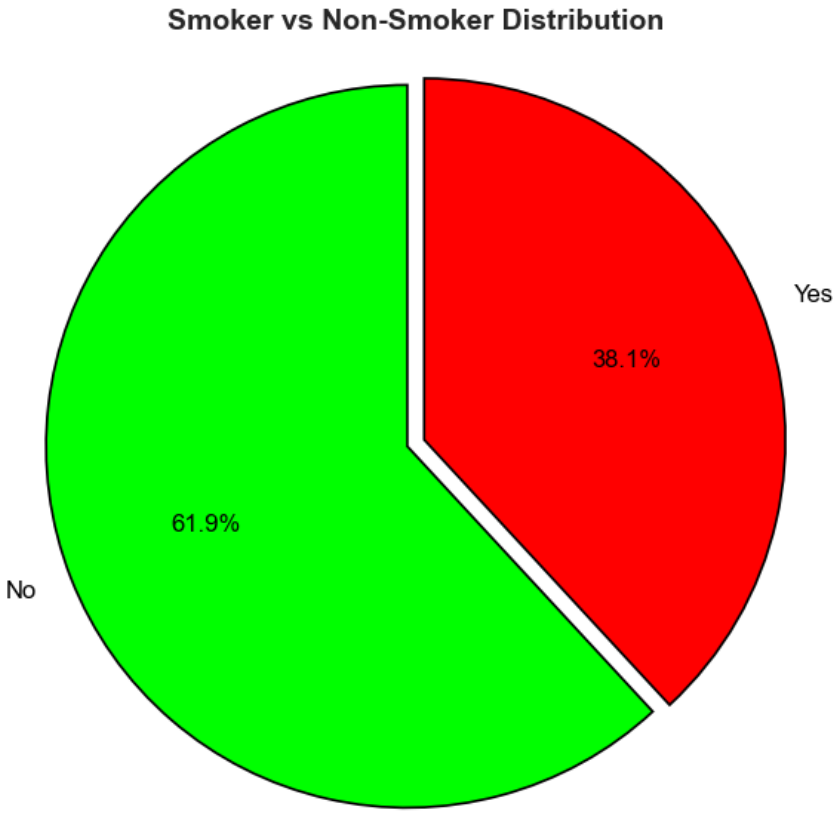
5. **Boxplot**
    - Create a boxplot of total_bill grouped by day.

```
sns.set_style("whitegrid")
plt.figure(figsize=(8, 5))

palette = sns.color_palette("magma")

sns.boxplot(data=tips, x="day", y="total_bill", palette=palette, linewidth=2, fliersize=4)

plt.title("Total Bill Distribution by Day", fontsize=14, fontweight='bold')
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Total Bill ($)", fontsize=12)

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



6. **Pie Chart**
    - Show pie chart of smoker vs non-smoker counts.

```
smoker_counts = tips["smoker"].value_counts()

labels = smoker_counts.index
sizes = smoker_counts.values
colors = ['#00FF00', '#FF0000']
explode = [0.05, 0]

plt.figure(figsize=(6, 6))
plt.pie(
    sizes,
    labels=labels,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    explode=explode,
    wedgeprops={'edgecolor': 'black', 'linewidth': 1.2},
    textprops={'fontsize': 12, 'color': 'black'}
)

plt.title("Smoker vs Non-Smoker Distribution", fontsize=14, fontweight='bold')
plt.axis('equal')
plt.tight_layout()
plt.show()
```



# Lab No: 8

**Objective:** To enable students to understand and implement statistical data visualizations using the **Seaborn library in Python**.

Seaborn is a Python visualization library based on Matplotlib, integrated with Pandas for ease of use with DataFrames. It provides high-level functions for attractive and informative statistical graphics.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

---

**1. Loading Built-in Datasets**

- Seaborn comes with built-in datasets like tips, iris, penguins, etc.

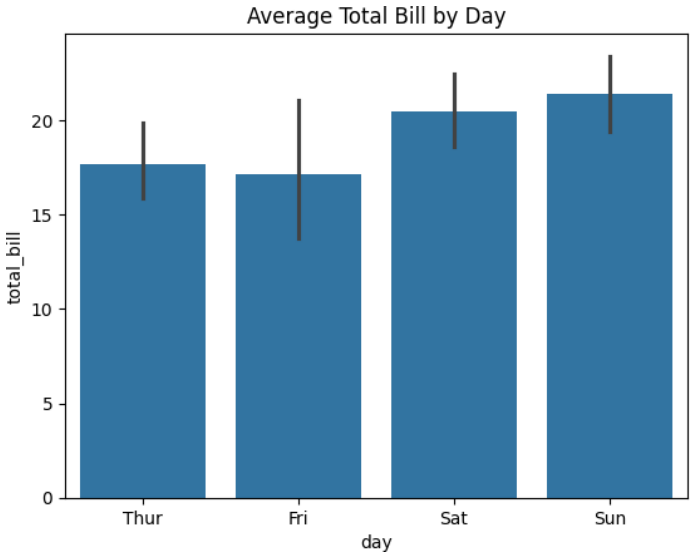
```
df = sns.load_dataset("tips")
```

---

**3. Bar Plot**

- Shows the relationship between a categorical variable and a numeric variable.

```
sns.barplot(x="day", y="total_bill", data=df)
plt.title("Average Total Bill by Day")
plt.show()
```

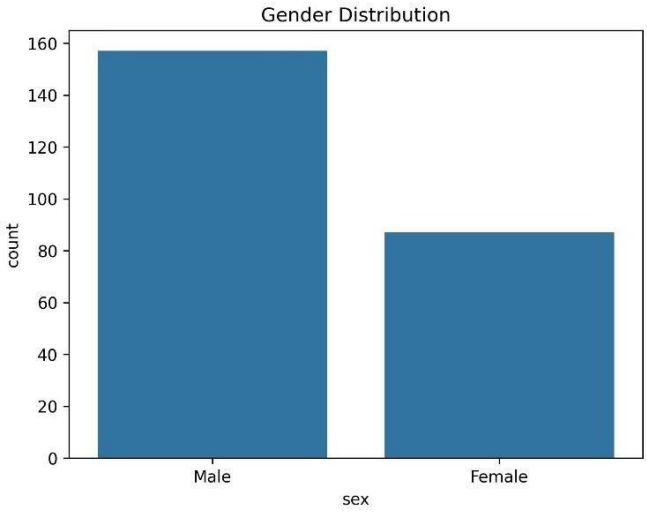



---

**4. Count Plot**

- Displays the count of observations in each categorical bin using bars.

```
sns.countplot(x="sex", data=df)
plt.title("Gender Distribution")
plt.show()
```

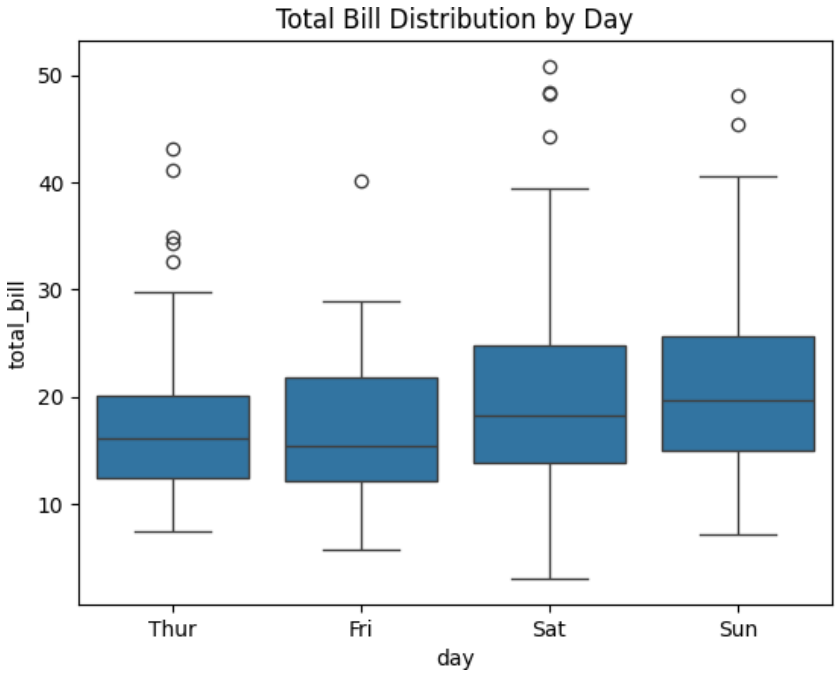



---

**5. Box Plot**

- Visualizes the distribution, median, and outliers of a numeric variable.

```
sns.boxplot(x="day", y="total_bill", data=df)
plt.title("Total Bill Distribution by Day")
plt.show()
```

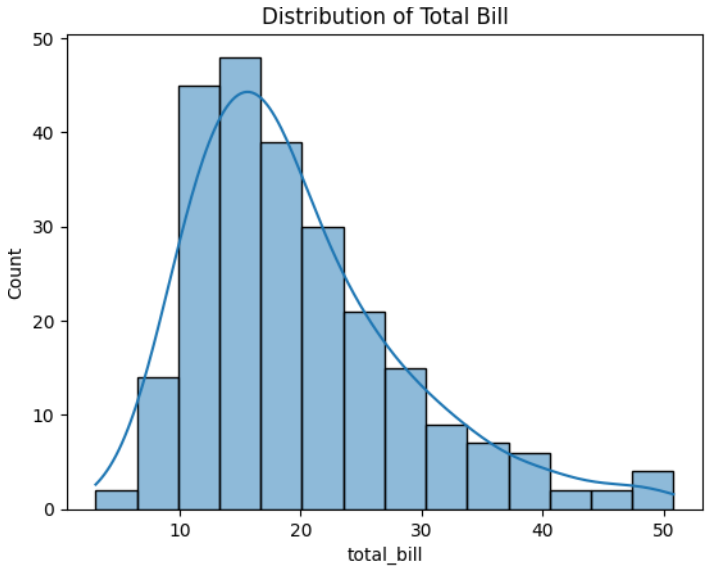



---

**6. Histogram / Distribution Plot**

- Used to show the distribution of a numeric variable.

```
sns.histplot(df["total_bill"], kde=True)
plt.title("Distribution of Total Bill")
plt.show()
```

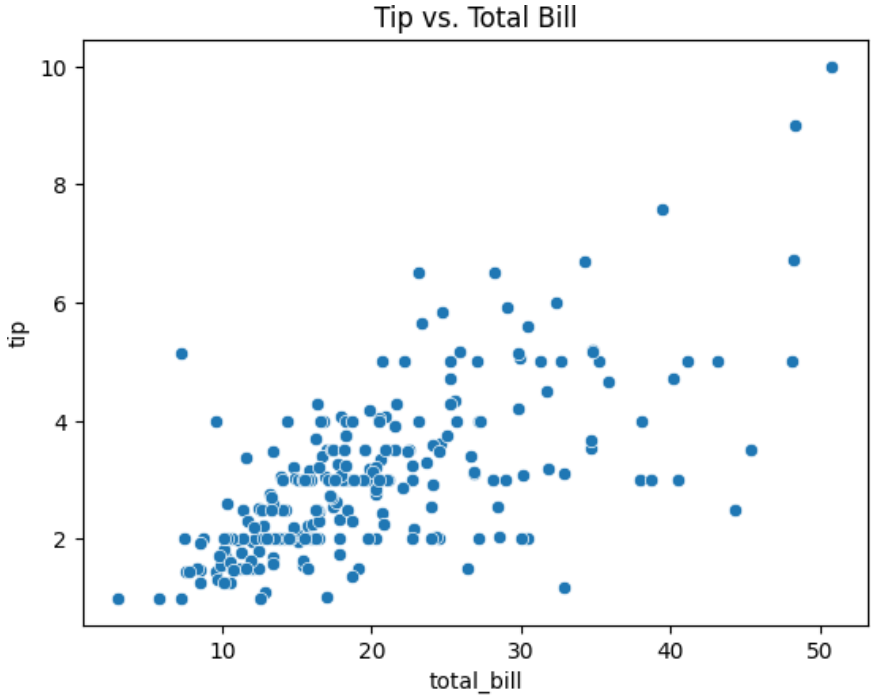



---

**7. Scatter Plot**

- Shows the relationship between two numeric variables.

```
sns.scatterplot(x="total_bill", y="tip", data=df)
plt.title("Tip vs. Total Bill")
plt.show()
```

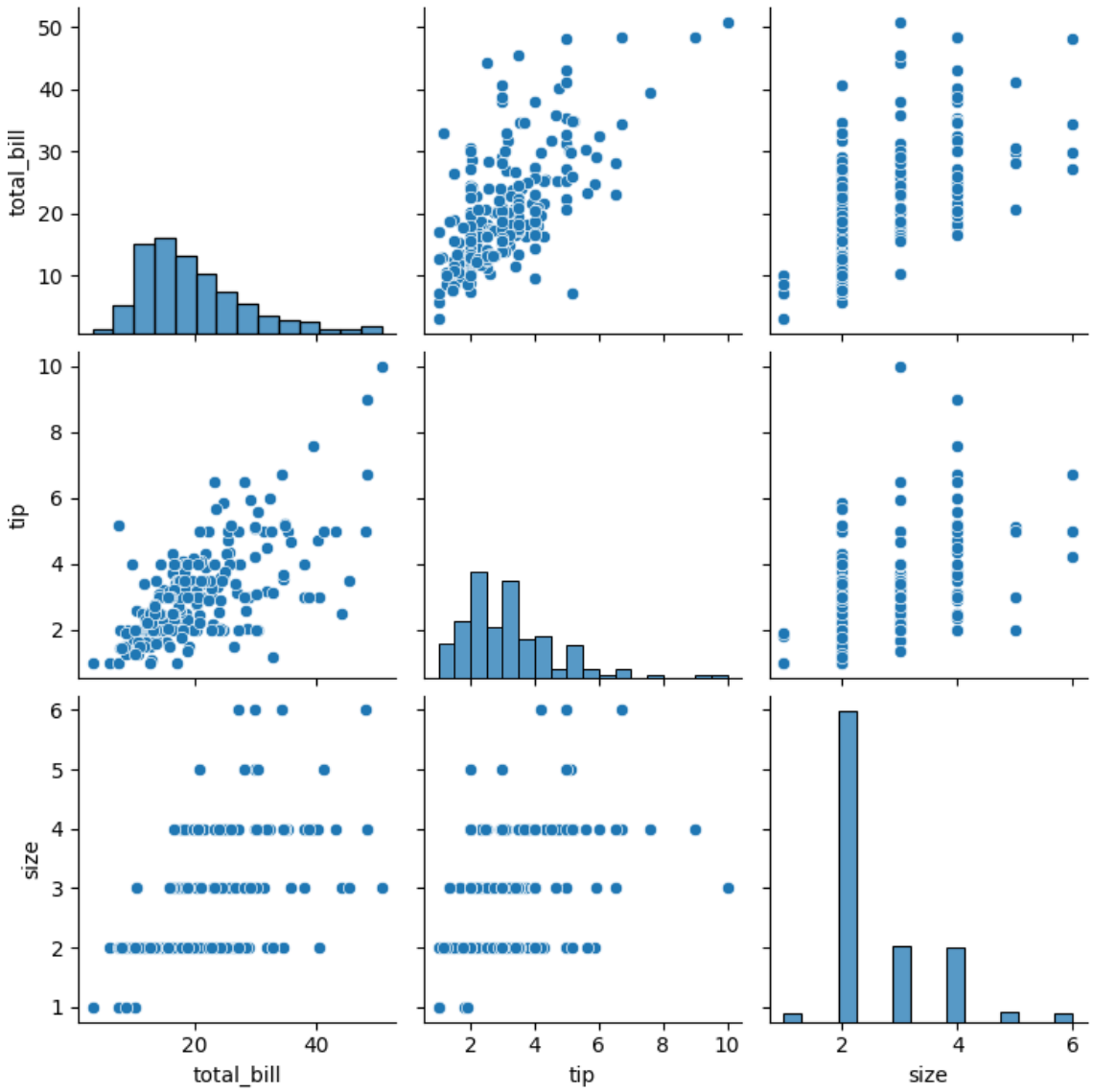



---

**8. Pair Plot**

- Displays pairwise relationships across the entire dataset.

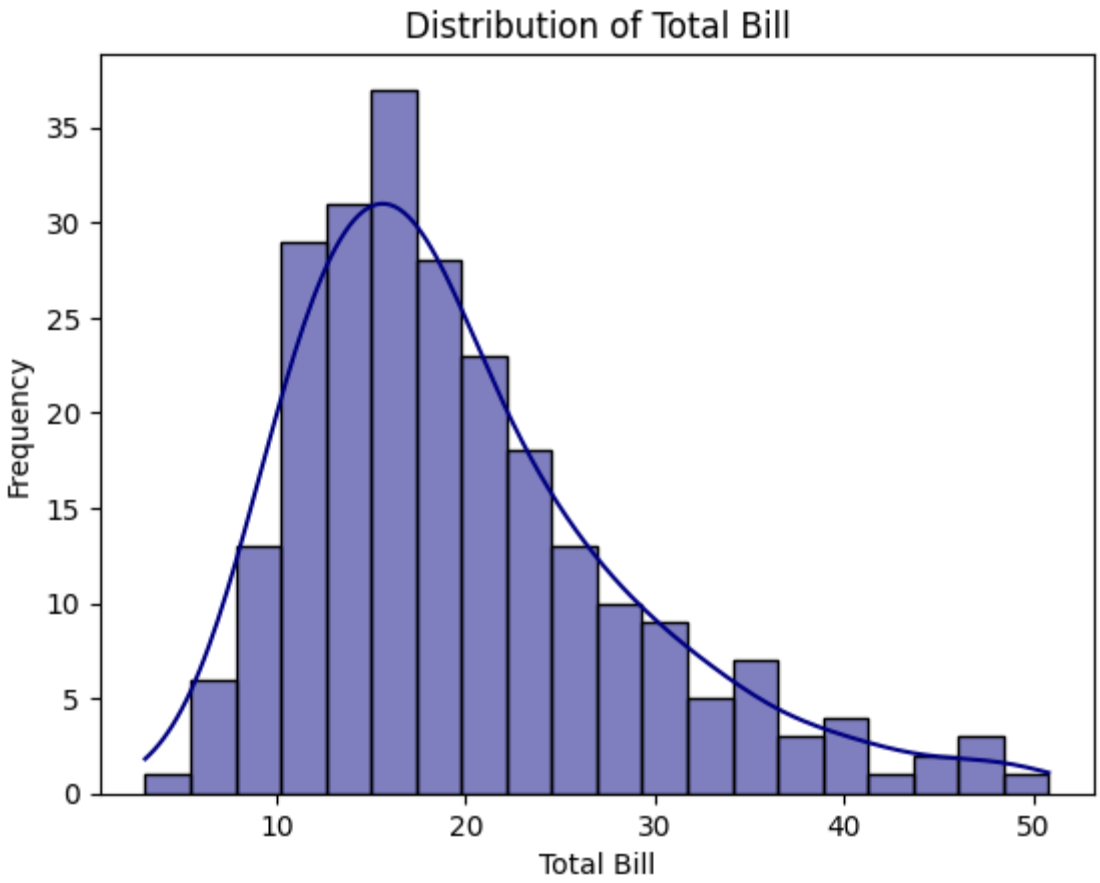
```
sns.pairplot(df)
plt.show()
```


total_bill
tip
size
50
40
30
20
10
10
8
6
4
2
6
5
4
3
2
1
20
40
2
4
6
8
10
2
4
6

**Task:**

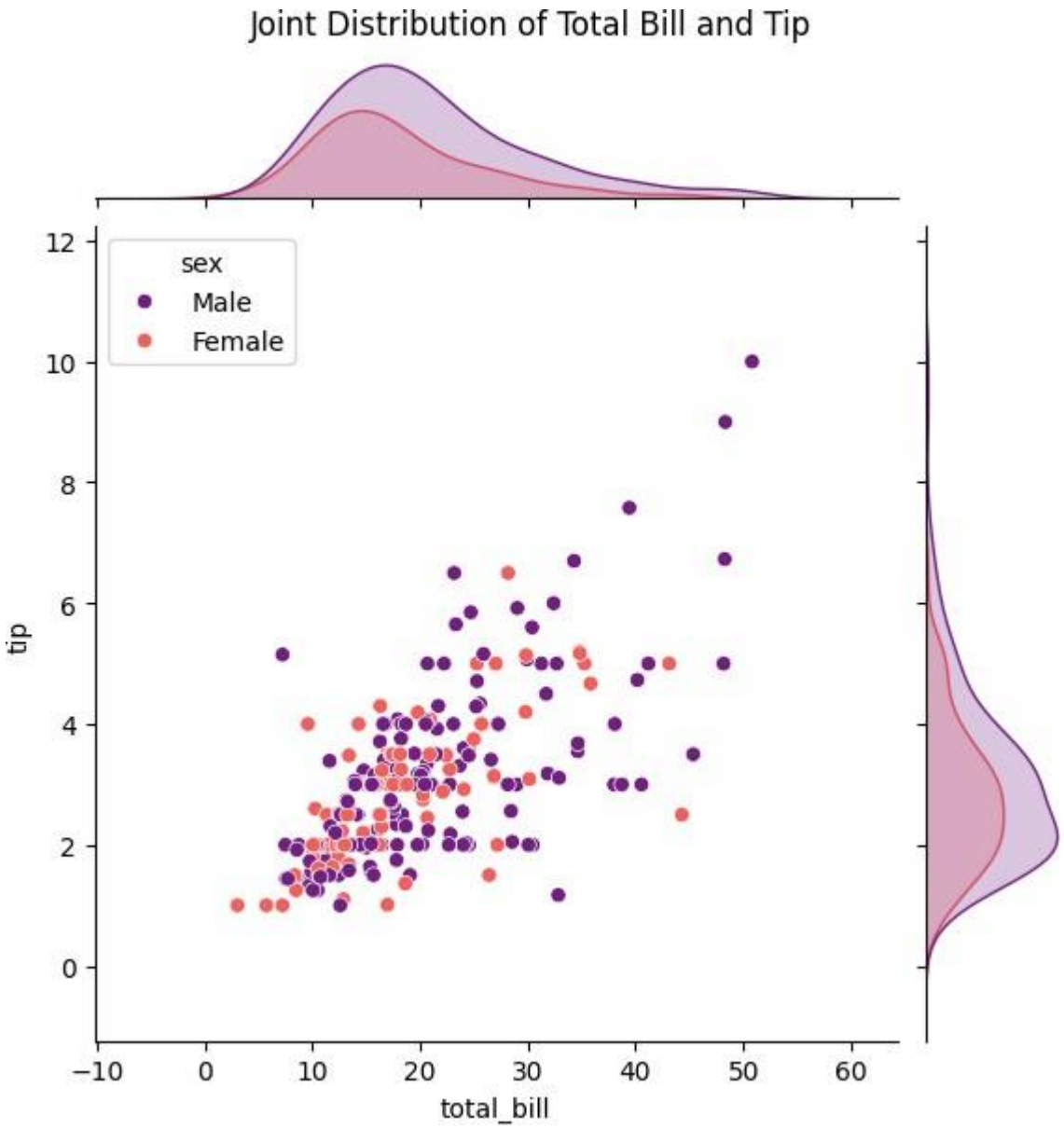
1. **Distplot**
    - Create a distribution plot of total_bill.

```
sns.histplot(tips["total_bill"], kde=True, color="navy", bins=20)
plt.title("Distribution of Total Bill")
plt.xlabel("Total Bill")
plt.ylabel("Frequency")
plt.show()
```



2. **Jointplot**
    - Plot a joint distribution of total_bill and tip.

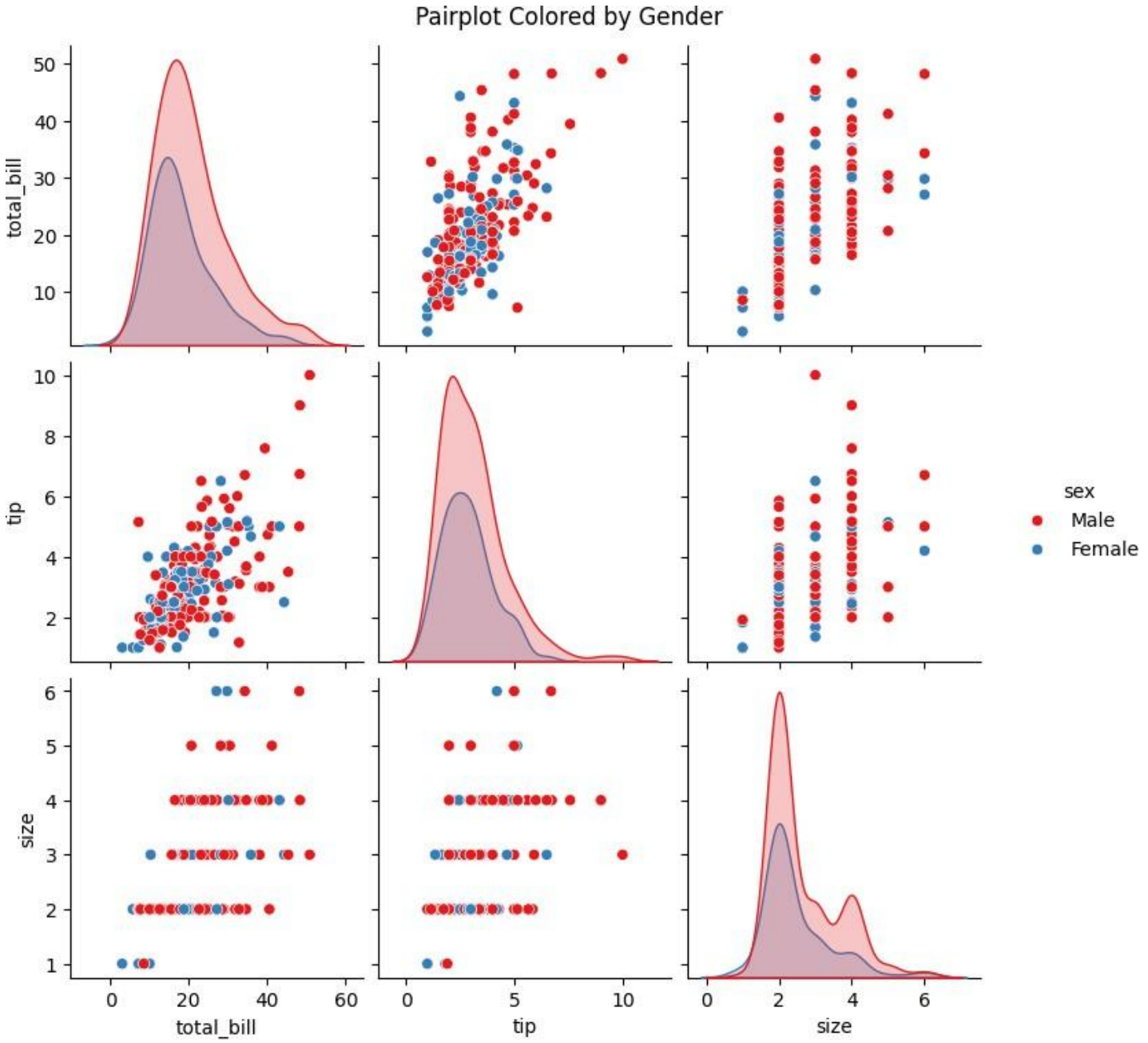
```
sns.jointplot(data=tips, x="total_bill", y="tip", kind="scatter", hue="sex", palette="magma")
plt.suptitle("Joint Distribution of Total Bill and Tip", y=1.02)
plt.show()
```



3. **Pairplot**

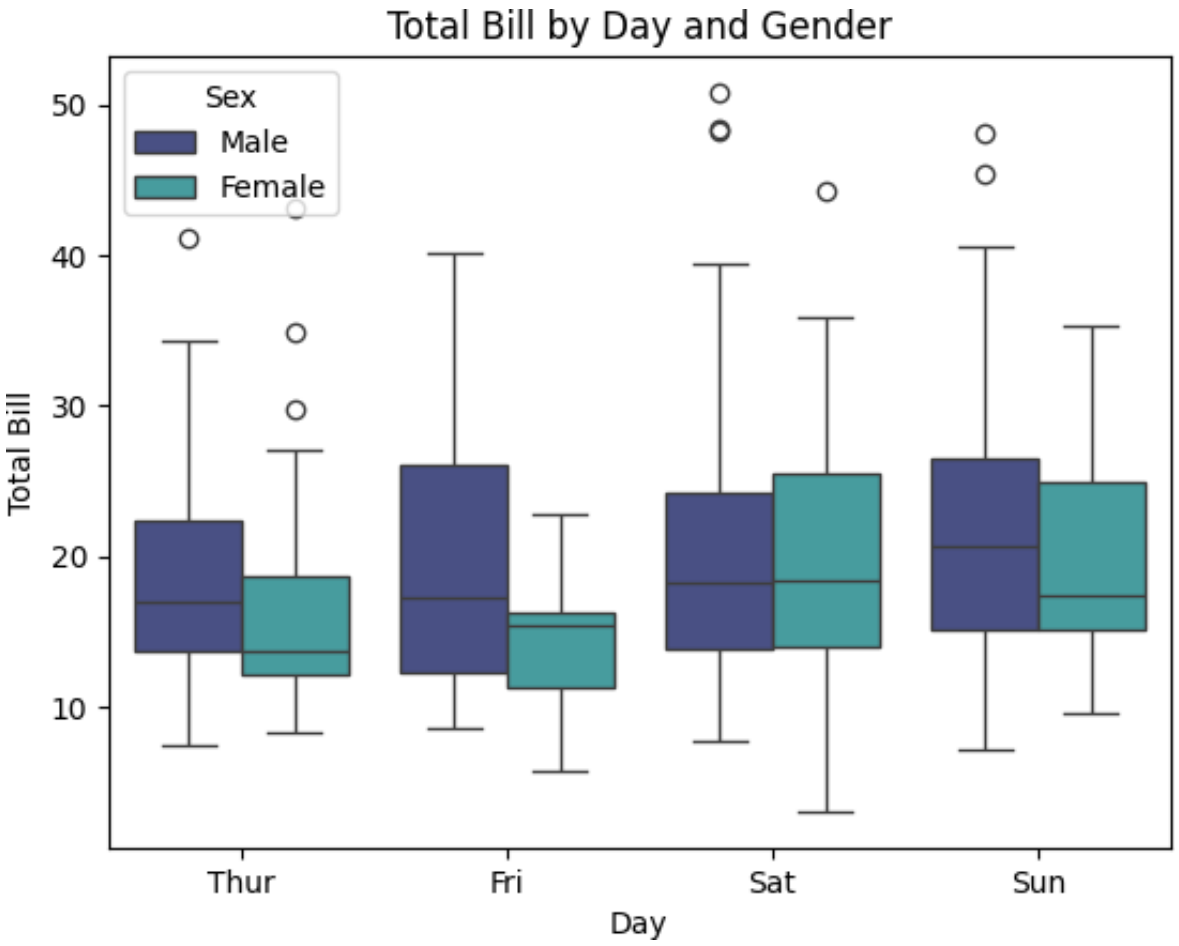
   - Create a pairplot of the numerical columns in the dataset colored by sex.

```
sns.pairplot(tips, hue="sex", palette="Set1")
plt.suptitle("Pairplot Colored by Gender", y=1.02)
plt.show()
```



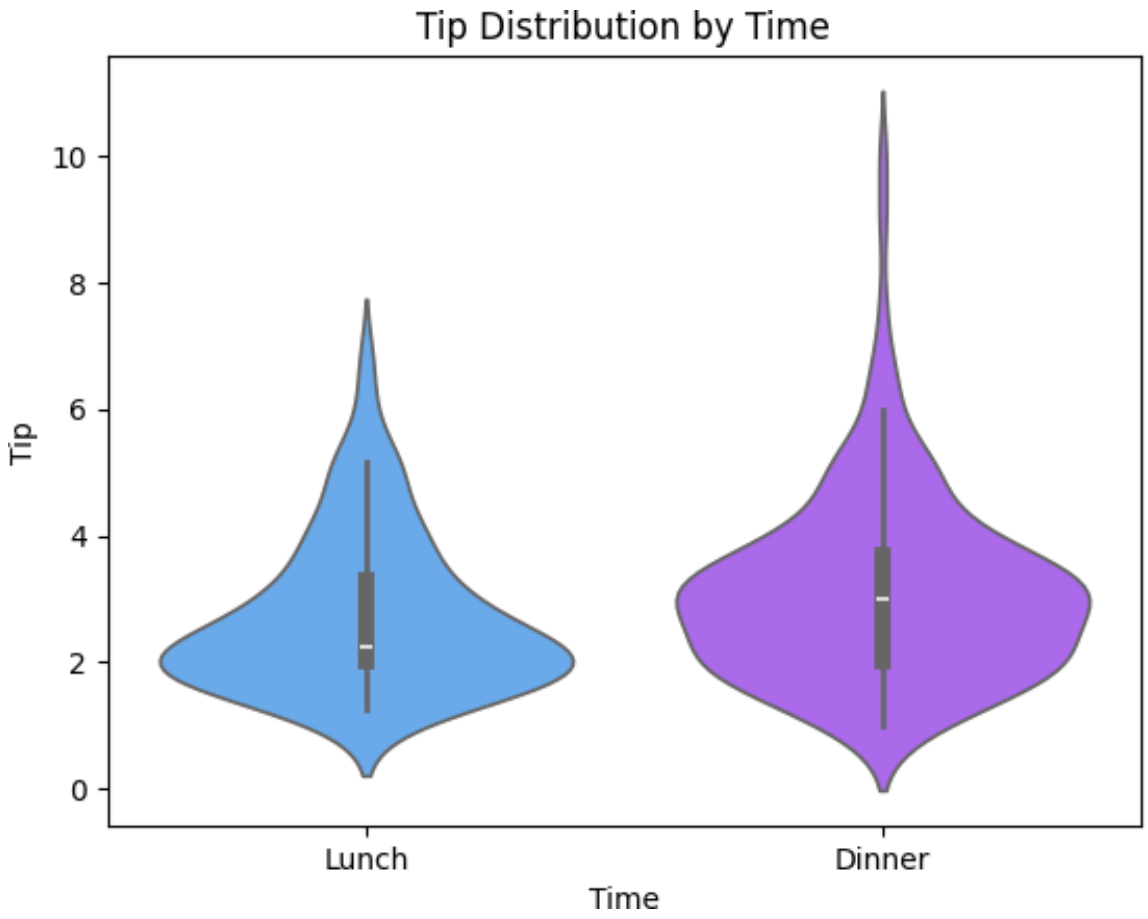
4. **Boxplot**
    - Create a boxplot showing total_bill for each day and further grouped by sex.

```
sns.boxplot(data=tips, x="day", y="total_bill", hue="sex", palette="mako")
plt.title("Total Bill by Day and Gender")
plt.xlabel("Day")
plt.ylabel("Total Bill")
plt.legend(title="Sex")
plt.show()
```



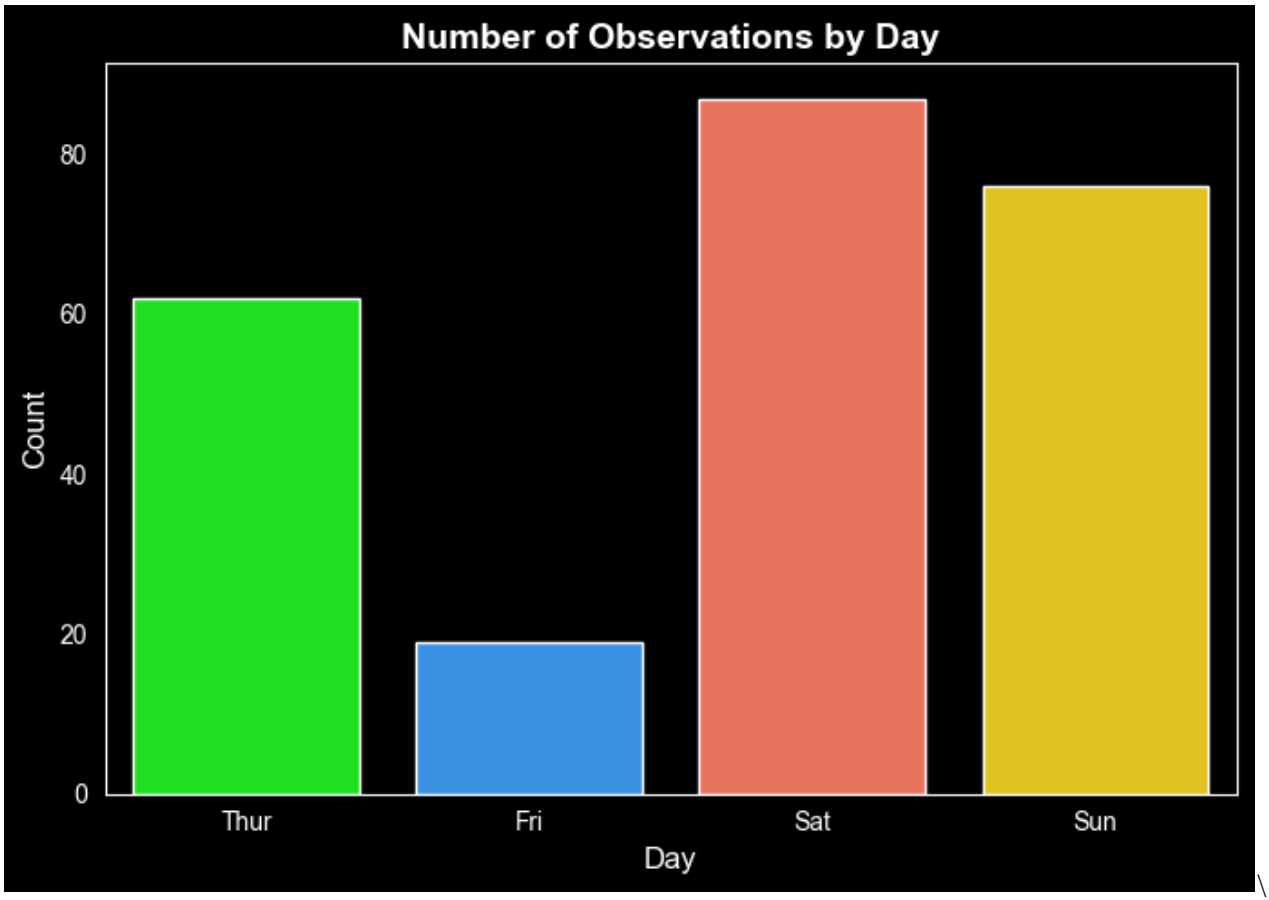
5. **Violinplot**
    - Create a violin plot comparing tip across different times (Lunch, Dinner).

```
sns.violinplot(data=tips, x="time", y="tip", palette="cool")
plt.title("Tip Distribution by Time")
plt.xlabel("Time")
plt.ylabel("Tip")
plt.show()
```



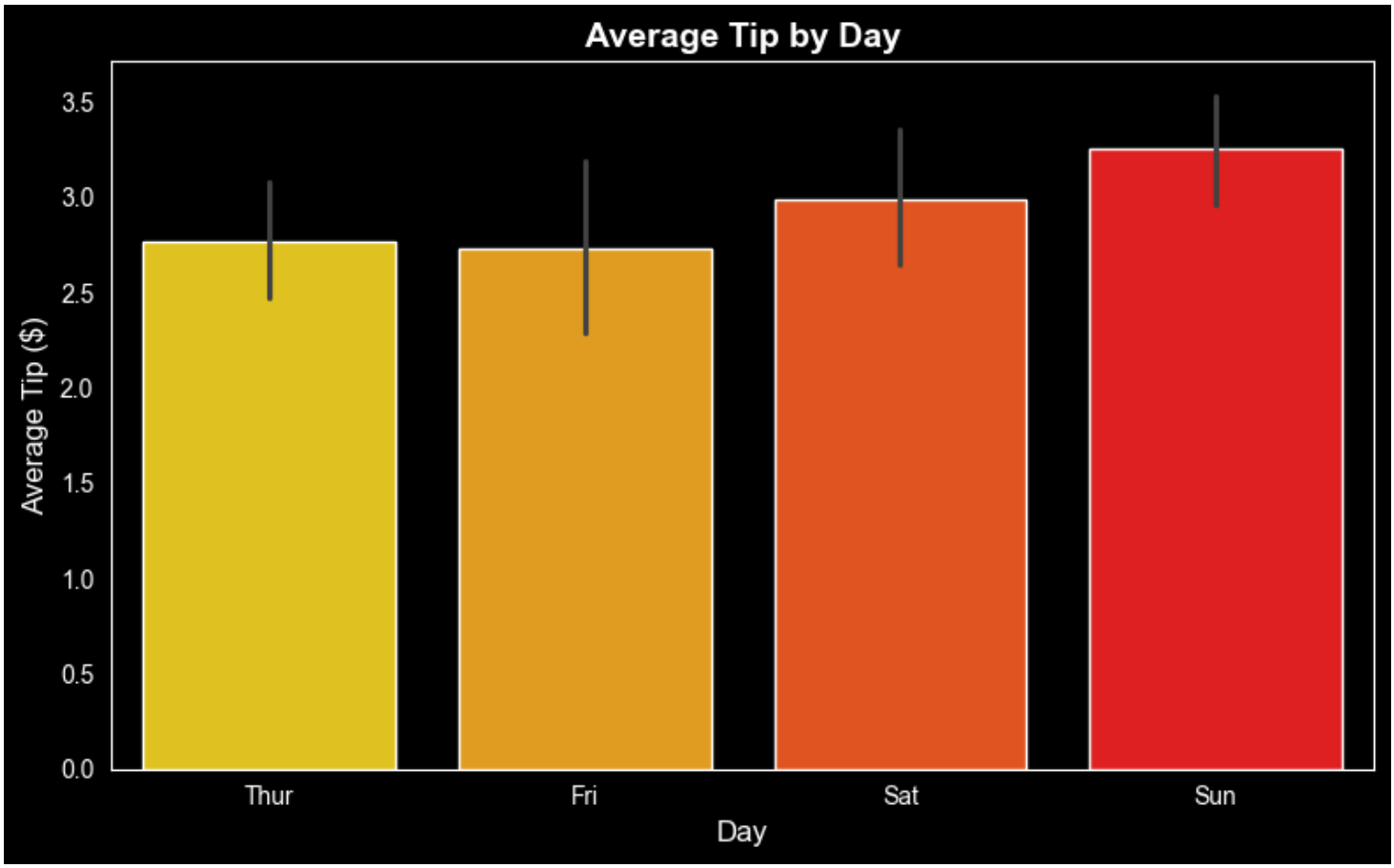
6. **Countplot**
    - Create a countplot showing the number of observations for each day.

```
sns.set_style("dark")
plt.style.use("dark_background")

dark_colors = ['#00FF00', '#1E90FF', '#FF6347', '#FFD700']

plt.figure(figsize=(7, 5))
sns.countplot(data=tips, x="day", palette=dark_colors)

plt.title("Number of Observations by Day", fontsize=14, fontweight='bold', color='white')
plt.xlabel("Day", fontsize=12, color='white')
plt.ylabel("Count", fontsize=12, color='white')
plt.xticks(color='white')
plt.yticks(color='white')

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



7. **Bar Plot**
    - Use sns.barplot() to show average tip for each day.

```
sns.set_style("dark")
plt.style.use("dark_background")

fire_palette = ['#FFD700', '#FFA500', '#FF4500', '#FF0000']

plt.figure(figsize=(8, 5))
sns.barplot(data=tips, x="day", y="tip", estimator='mean', palette=fire_palette)

plt.title("Average Tip by Day", fontsize=14, fontweight='bold', color='white')
plt.xlabel("Day", fontsize=12, color='white')
plt.ylabel("Average Tip ($)", fontsize=12, color='white')
plt.xticks(color='white')
plt.yticks(color='white')

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



# Lab No: 9

**Objective:** To perform **descriptive and inferential statistical** analyses using both Python and R, and to interpret the outcomes for real-world data sets. This lab will reinforce the role of statistics in AI tasks like data understanding, feature engineering, and modeling.

Statistical analysis forms the foundation of data-driven decision-making and is a critical component of artificial intelligence (AI), particularly in tasks such as data understanding, feature selection, model evaluation, and performance interpretation. This lab introduces both descriptive and inferential statistical techniques and demonstrates their implementation using Python and R.

---

**1. Descriptive Statistics**

Descriptive statistics are used to summarize and describe the main features of a dataset quantitatively. These statistics help provide a clear understanding of the distribution, central tendency, and variability within data.

**Key Concepts:**

- **Mean**: The average value of the dataset.
- **Median**: The middle value when data is sorted.
- **Mode**: The most frequently occurring value.
- **Variance**: The measure of spread in the data (how far each value is from the mean).
- **Standard Deviation (SD)**: The square root of variance; shows the dispersion in the same units as the data.
- **Skewness**: Indicates the asymmetry of the data distribution. Positive skew means a long right tail; negative skew means a long-left tail.
- **Kurtosis**: Measures the "tailedness" of the data distribution. High kurtosis indicates heavy tails; low kurtosis indicates light tails.

These metrics help in understanding the shape, spread, and central behaviour of the dataset.

---

**2. Inferential Statistics**

Inferential statistics involve **drawing conclusions or making predictions** about a population based on a sample of data. These techniques allow us to test hypotheses and understand relationships between variables.

**Key Concepts:**

- **Correlation**:
  - Measures the **strength and direction of a linear relationship** between two numeric variables.
  - Values range from -1 (perfect negative) to +1 (perfect positive).
  - Zero correlation implies no linear relationship.
- **T-Test**:
  - A statistical hypothesis test used to compare the **means of two groups**.
  - Commonly used t-tests:
    - **Independent t-test**: Compares means of two independent groups.
    - **Paired t-test**: Compares means of the same group at two different times.
  - Assumes normally distributed data and equal variances (in standard forms).
- **P-Value**:
  - The **probability** of obtaining results at least as extreme as the observed ones, assuming the null hypothesis is true.
  - A small p-value (typically $< 0.05$) indicates **strong evidence against the null hypothesis**.

- **Confidence Interval (CI)**:
  - A **range of values** within which the true population parameter is expected to lie, with a certain level of confidence (commonly 95%).

---

**3. Role of Statistics in AI and Data Science**

In AI workflows, statistical techniques are used to:

- Understand the dataset before modelling (exploratory data analysis).
- Engineer features by identifying important variables and handling data distributions.
- Validate models using hypothesis tests and statistical performance metrics.
- Explain model predictions in interpretable ways (especially important in ethical AI and explainable AI frameworks).

By performing both descriptive and inferential statistics in **Python** and **R**, students gain practical fluency in interpreting real-world datasets and applying statistical thinking in AI applications.

**Tasks:**

**Part A – Descriptive Statistics in Python**

1. Load a dataset using pandas (e.g., Titanic, Iris, or custom CSV).

```
import pandas as pd
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("The Titanic Dataset.csv")
```

**2. Calculate:**

**- Mean, median, mode**
**- Variance and standard deviation**
**- Skewness and kurtosis**

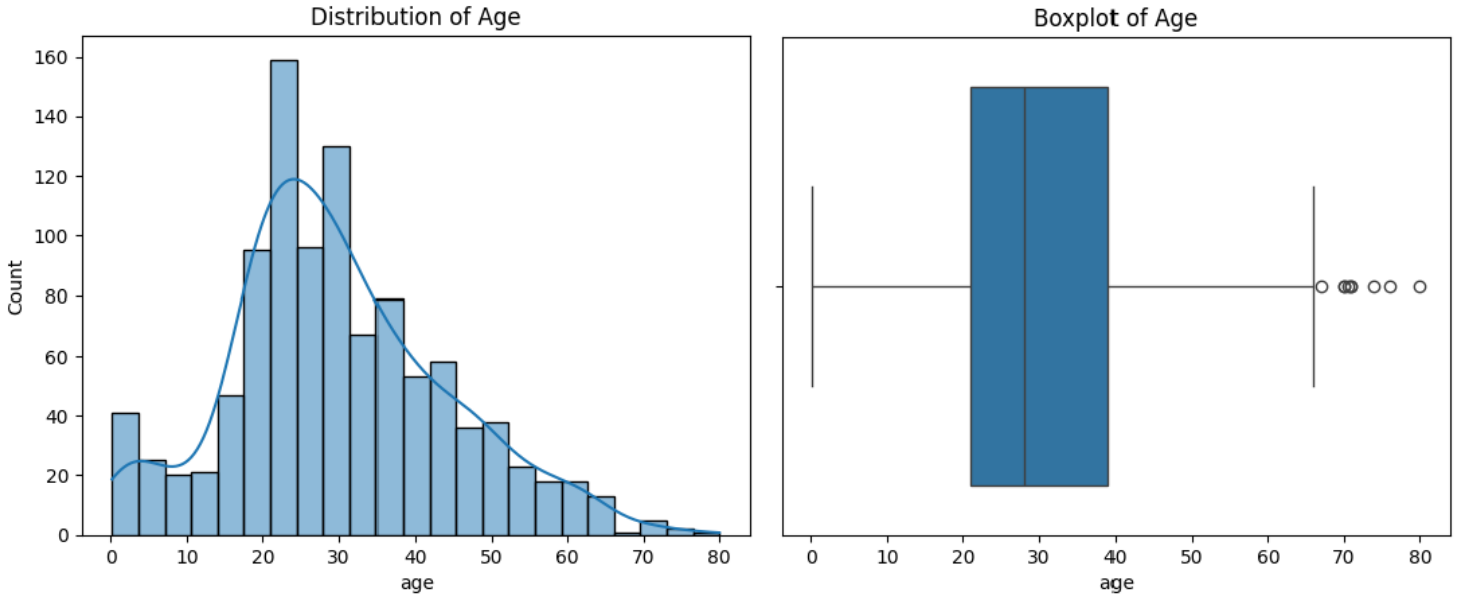
```
mean_age = df['age'].mean()
median_age = df['age'].median()
mode_age = df['age'].mode()[0]

variance_age = df['age'].var()
std_dev_age = df['age'].std()

skew_age = df['age'].skew()
kurt_age = df['age'].kurt()

print("Mean:", mean_age)
print("Median:", median_age)
print("Mode:", mode_age)
print("Variance:", variance_age)
print("Std Dev:", std_dev_age)
print("Skewness:", skew_age)
print("Kurtosis:", kurt_age)
```

**3. Display summary using df.describe()**

```
print(df.describe())
```

**4. Plot distributions using seaborn.histplot() or boxplot()**

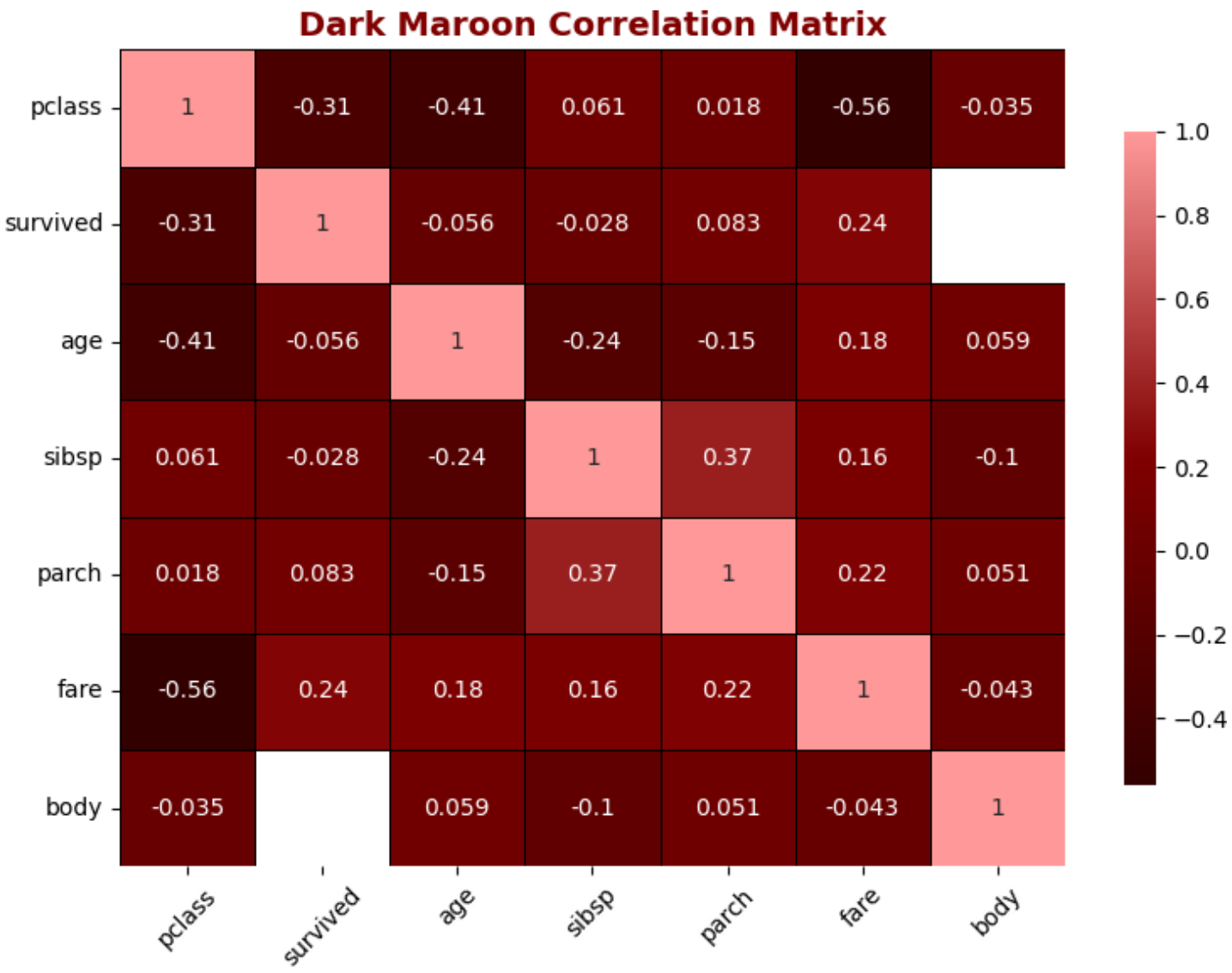
```
# Histogram
sns.histplot(df['age'].dropna(), kde=True)
plt.title("Distribution of Age")
plt.show()

# Boxplot
sns.boxplot(x=df['age'])
plt.title("Boxplot of Age")
plt.show()
```




**Part B – Inferential Statistics in Python**

1. Compute correlation matrix using df.corr().

```
from matplotlib.colors import LinearSegmentedColormap
maroon_cmap = LinearSegmentedColormap.from_list(
    "dark_maroon", ["#330000", "#800000", "#FF9999"]
)

corr_matrix = df.corr(numeric_only=True)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap=maroon_cmap, linewidths=0.5, linecolor='black',
cbar_kws={"shrink": 0.8})

plt.title("Dark Maroon Correlation Matrix", fontsize=14, fontweight='bold', color='maroon')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```




**2. Conduct a t-test using scipy.stats.ttest_ind() to compare two sample means.**

**3. Interpret p-values and confidence intervals.**

```
survived = df[df["survived"] == 1]["age"].dropna()
not_survived = df[df["survived"] == 0]["age"].dropna()

t_stat, p_val = stats.ttest_ind(survived, not_survived, equal_var=False)

print("T-Statistic:", t_stat)
print("P-Value:", p_val)
```

# Lab No: 10

**Objective:** To introduce students to solving **ordinary differential equations (ODEs)** using Python's SciPy library.

Differential equations are mathematical equations that relate a function to its derivatives. They are essential in modeling a wide range of real-world phenomena such as population dynamics, chemical reactions, mechanical vibrations, electrical circuits, and many systems in artificial intelligence, robotics, and control theory.

This lab focuses on Ordinary Differential Equations (ODEs) and demonstrates how to solve them numerically using Python's SciPy library.

---

### 1. Ordinary Differential Equations (ODEs)

An Ordinary Differential Equation (ODE) is an equation that involves one or more derivatives of a function with respect to a single independent variable (usually time t).

**Types of ODEs:**

- **First-Order ODE**: Involves the first derivative of the function.
  - Example:

$$\frac{dy}{dt} = -2y + 1$$

- **Second-Order ODE**: Involves the second derivative.
  - Example:

$$\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 2y = 0$$

---

### 2. Initial Value Problems (IVPs)

In practical applications, ODEs are often solved as initial value problems, where the value of the unknown function is given at a specific point.

- For the equation:

$$\frac{dy}{dt} = f(y, t)$$

an initial value problem specifies:

$$y(t_0) = y_0$$

The solution is then computed for y(t)y(t) over a given range starting from t0t_0, using numerical methods.

---

### 3. Solving ODEs in Python using scipy.integrate.odeint

Python's SciPy library provides powerful tools for solving ODEs. The most commonly used function is:

`scipy.integrate.odeint(func, y0, t)`

- `func`: a user-defined function that returns dy/dt
- `y0`: the initial condition
- `t`: array of time points for which the solution is to be computed

**Example:**

To solve:

$$\frac{dy}{dt} = -2y + 1, \quad y(0) = 0$$

You define the function in Python:

```
def model(y, t):
    return -2*y + 1
```

And solve it using odeint.

---

### 4. Visualizing the Solution

To understand the behavior of the solution, the result is plotted using Matplotlib, which provides tools for creating line graphs to represent the change of `y(t)` over time.

**Basic Plot:**

```
import matplotlib.pyplot as plt
plt.plot(t, y)
plt.xlabel("Time")
plt.ylabel("y(t)")
plt.title("Solution of dy/dt = -2y + 1")
```

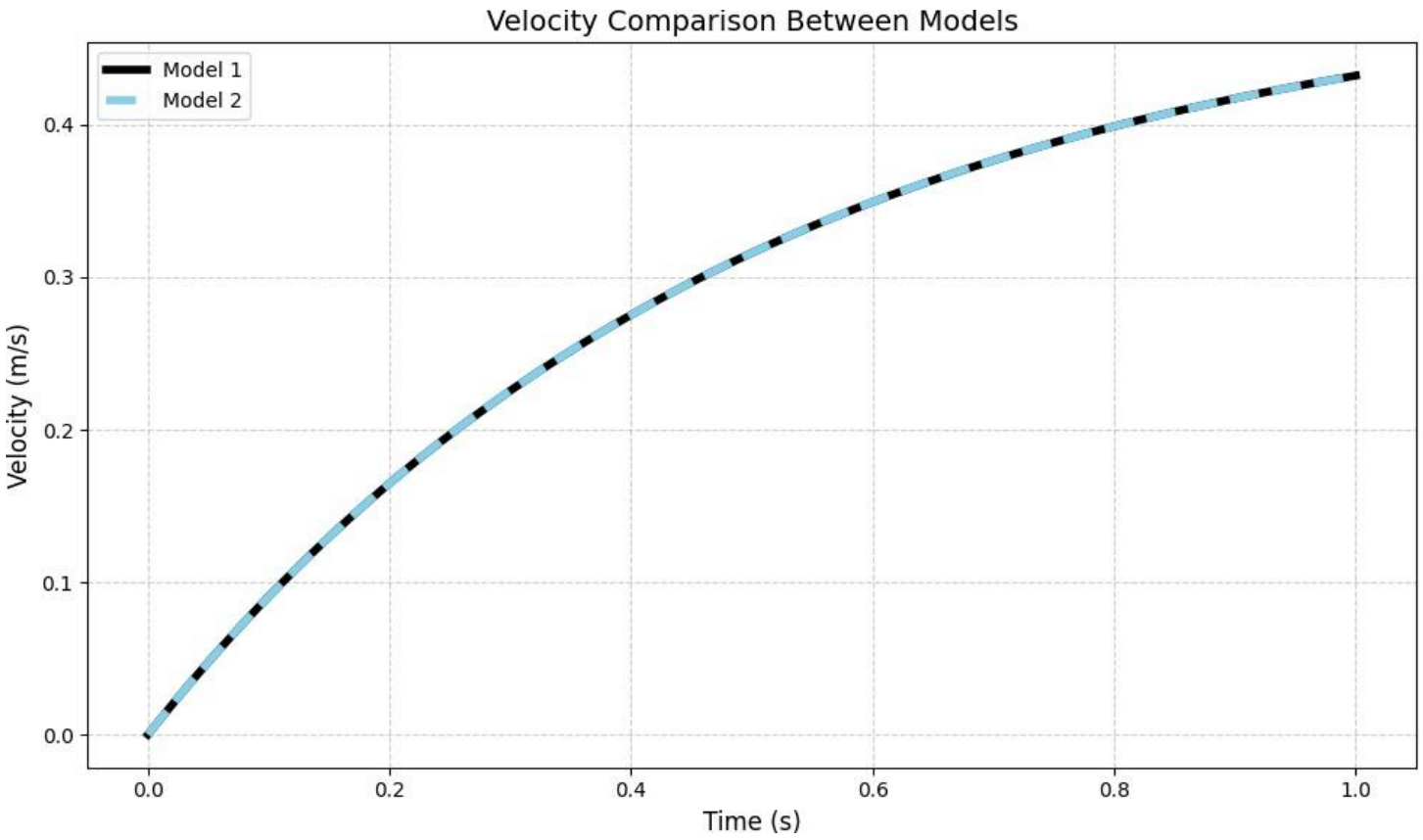
Visualization helps interpret the long-term behavior, stability, and trends of the dynamic system modeled by the differential equation.

**Tasks:**

1. Import required libraries.
2. Define a simple first-order ODE:
   dy/dt = -2y + 1
3. Solve the ODE with initial condition y(0) = 0.
4. Plot the result using Matplotlib.

**SOLUTION**

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.integrate import odeint
from scipy.integrate import solve_ivp

def dvdt(t, y):
    return (-2*y + 1)
y0 = 0
t = np.linspace(0, 1, 100)

solve_m1 = odeint(dvdt, y0=0, t=t, tfirst=True)
solve_m2 = solve_ivp(dvdt, y0=[y0], t_span=(min(t), max(t)), t_eval=t)

v_solve_m1 = solve_m1.T[0]
v_solve_m2 = solve_m2.y[0]

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

plt.plot(t, v_solve_m1, label='Model 1', color='black', linewidth=4)
plt.plot(t, v_solve_m2, '--', label='Model 2', color='skyblue', linewidth=4)

plt.title('Velocity Comparison Between Models', fontsize=14)
plt.xlabel('Time (s)', fontsize=12)
plt.ylabel('Velocity (m/s)', fontsize=12)

plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()

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




# Lab No: 12

Github Repo Link: https://github.com/Rehman810/AI-OEL-2/blob/main/23-AI-27.ipynb

```
graph = {
   'A': [('B', 6), ('C', 2)],
   'B': [('D', 5), ('E', 3)],
   'C': [('F', 4)],
   'D': [('G', 2)],
   'E': [('G', 6)],
   'F': [('G', 1)],
   'G': []
}

def dfs(graph, start, goal, path=None, visited=None):
   if visited is None:
      visited = set()
   if path is None:
      path = []

   visited.add(start)
   path = path + [start]
```

```
    if start == goal:
        return path

    for neighbor, _ in graph[start]:
        if neighbor not in visited:
            new_path = dfs(graph, neighbor, goal, path, visited)
            if new_path:
                return new_path
    return None

dfs_path = dfs(graph, 'A', 'G')
print("DFS Path:", dfs_path)
```

```
DFS Path: ['A', 'B', 'D', 'G']
```

**2. Implement A* Search, assuming the following heuristic values:**

```
heuristic = {
    'A': 7, 'B': 6, 'C': 4, 'D': 3,
    'E': 5, 'F': 2, 'G': 0
}
In [32]:
from heapq import heappush, heappop

def a_star(graph, start, goal, h):
    open_set = []
    heappush(open_set, (h[start], 0, start, [start]))

    visited = {}

    while open_set:
        f, g, current, path = heappop(open_set)

        if current in visited and visited[current] <= g:
            continue

        visited[current] = g

        if current == goal:
            return path, g

        for neighbor, cost in graph[current]:
            new_g = g + cost
            new_f = new_g + h[neighbor]
            heappush(open_set, (new_f, new_g, neighbor, path + [neighbor]))

    return None, float('inf')

a_star_path, a_star_distance = a_star(graph, 'A', 'G', heuristic)
print("A* Path:", a_star_path)
```

```
A* Path: ['A', 'C', 'F', 'G']
```

**3. Show the path and total distance covered for both algorithms**

```
def calculate_distance(path, graph):
    distance = 0
    for i in range(len(path) - 1):
        for neighbor, weight in graph[path[i]]:
            if neighbor == path[i + 1]:
                distance += weight
                break
    return distance

dfs_distance = calculate_distance(dfs_path, graph)
print("DFS Total Distance:", dfs_distance)

a_star_path, a_star_distance = a_star(graph, 'A', 'G', heuristic)
print("A* Total Distance:", a_star_distance)
```

```
DFS Total Distance: 13
A* Total Distance: 7
```