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

DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY

UNITY • FAITH • DISCIPLINE

CERTIFICATE

This is to certify that Mr./Ms. ARSALAN AHMED with Roll # 23-AI-66 of Batch 2023 has successfully completed all the labs prescribed for the course "Programming of Artificial Intelligence".

Engr. Hamza Farooqui
Lecturer
Department of AI

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 interms of ternary operator:

```
nums = [1,2,3,4,5]

newNums = []

for num in nums:

        if num >= 3:

                newNums.append(num)

print(newNums)
```

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

-: LAB 01 :-

Q) Rewrite The Given Code In List Comprehension Form.

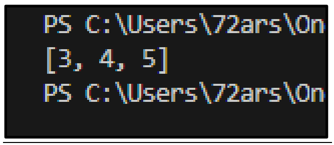
CODE:

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

LIST COMPREHENSION FORM:

```
nums = [1, 2, 3, 4, 5] new_num = [num for num in nums if num >= 3]
print(new_num)
```

OUTPUT:



```
PS C:\Users\72ars\On
[3, 4, 5]
PS C:\Users\72ars\On
```

LEET CODE

DETECT CAPITAL:

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

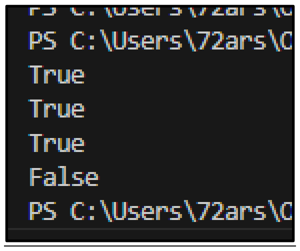
- All letters in this word are capitals, like "USA".
- All letters in this word are not capitals, like "leetcode".
- Only the first letter in this word is capital, like "Google".

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

CODE:

```
def detect_capital_use(word: str) -> bool:
    if not word:
        return False
    if all('A' <= char <= 'Z' for char in word):
        return True
    if all('a' <= char <= 'z' for char in word):
        return True
    if 'A' <= word[0] <= 'Z' and all('a' <= char <= 'z' for char in word[1:]):
        return True
    return False
print(detect_capital_use("PAK"))
print(detect_capital_use("pak"))
print(detect_capital_use("Pak"))
print(detect_capital_use("PaK"))
```

OUTPUT:



```
PS C:\Users\72ars\C
PS C:\Users\72ars\C
True
True
True
False
PS C:\Users\72ars\C
```

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.

Use Constructor to Initialize Objects

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

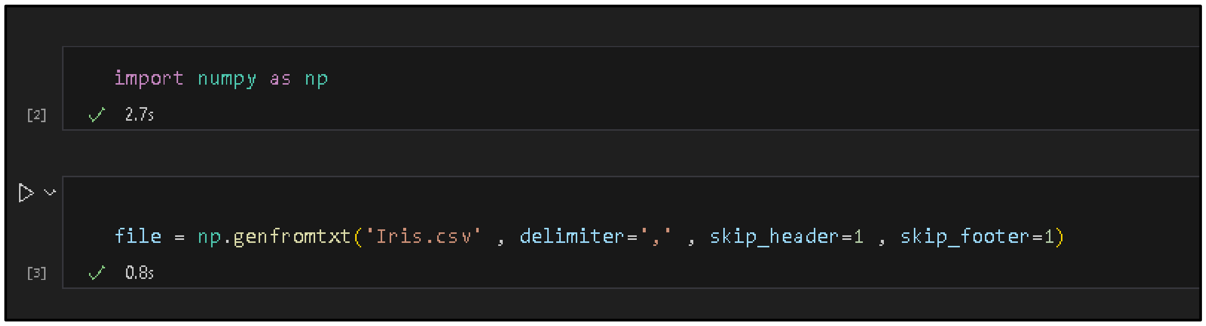
Create and Use Objects

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

-: LAB 02 :-

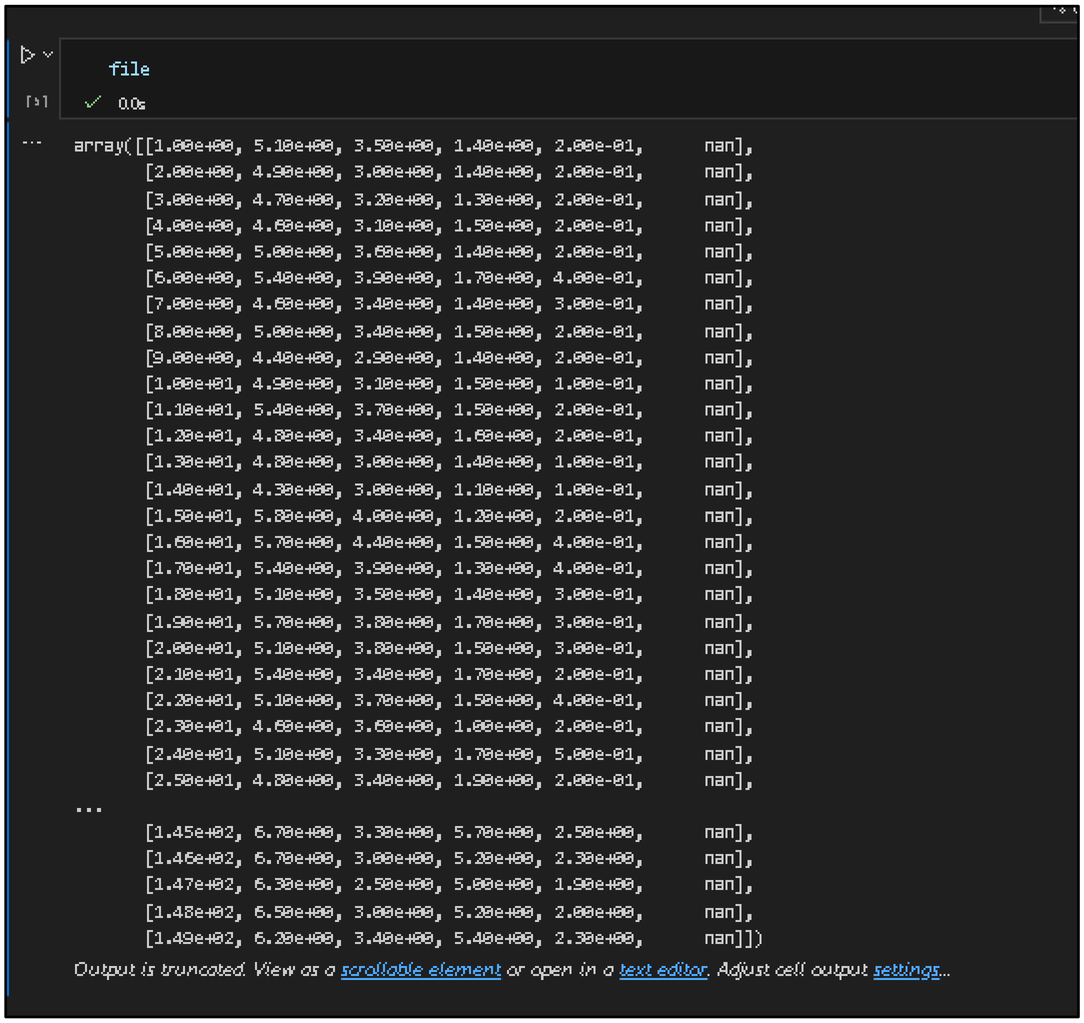
TASK:01:

1)



```
import numpy as np
[2] ✓ 2.7s

file = np.genfromtxt('Iris.csv' , delimiter=',' , skip_header=1 , skip_footer=1)
[3] ✓ 0.8s
```

2)

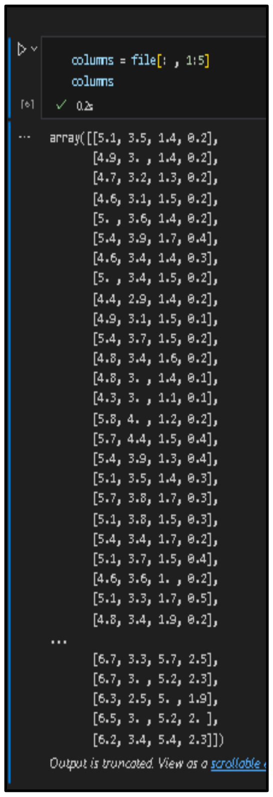


```
file
```

```
array([[1.00e+00, 5.10e+00, 3.50e+00, 1.40e+00, 2.00e-01,      nan],
       [2.00e+00, 4.90e+00, 3.00e+00, 1.40e+00, 2.00e-01,      nan],
       [3.00e+00, 4.70e+00, 3.20e+00, 1.30e+00, 2.00e-01,      nan],
       [4.00e+00, 4.60e+00, 3.10e+00, 1.50e+00, 2.00e-01,      nan],
       [5.00e+00, 5.00e+00, 3.60e+00, 1.40e+00, 2.00e-01,      nan],
       [6.00e+00, 5.40e+00, 3.90e+00, 1.70e+00, 4.00e-01,      nan],
       [7.00e+00, 4.60e+00, 3.40e+00, 1.40e+00, 3.00e-01,      nan],
       [8.00e+00, 5.00e+00, 3.40e+00, 1.50e+00, 2.00e-01,      nan],
       [9.00e+00, 4.40e+00, 2.90e+00, 1.40e+00, 2.00e-01,      nan],
       [1.00e+01, 4.90e+00, 3.10e+00, 1.50e+00, 1.00e-01,      nan],
       [1.10e+01, 5.40e+00, 3.70e+00, 1.50e+00, 2.00e-01,      nan],
       [1.20e+01, 4.80e+00, 3.40e+00, 1.60e+00, 2.00e-01,      nan],
       [1.30e+01, 4.80e+00, 3.00e+00, 1.40e+00, 1.00e-01,      nan],
       [1.40e+01, 4.30e+00, 3.00e+00, 1.10e+00, 1.00e-01,      nan],
       [1.50e+01, 5.80e+00, 4.00e+00, 1.20e+00, 2.00e-01,      nan],
       [1.60e+01, 5.70e+00, 4.40e+00, 1.50e+00, 4.00e-01,      nan],
       [1.70e+01, 5.40e+00, 3.90e+00, 1.30e+00, 4.00e-01,      nan],
       [1.80e+01, 5.10e+00, 3.50e+00, 1.40e+00, 3.00e-01,      nan],
       [1.90e+01, 5.70e+00, 3.80e+00, 1.70e+00, 3.00e-01,      nan],
       [2.00e+01, 5.10e+00, 3.80e+00, 1.50e+00, 3.00e-01,      nan],
       [2.10e+01, 5.40e+00, 3.40e+00, 1.70e+00, 2.00e-01,      nan],
       [2.20e+01, 5.10e+00, 3.70e+00, 1.50e+00, 4.00e-01,      nan],
       [2.30e+01, 4.60e+00, 3.60e+00, 1.00e+00, 2.00e-01,      nan],
       [2.40e+01, 5.10e+00, 3.30e+00, 1.70e+00, 5.00e-01,      nan],
       [2.50e+01, 4.80e+00, 3.40e+00, 1.90e+00, 2.00e-01,      nan],
...
       [1.45e+02, 6.70e+00, 3.30e+00, 5.70e+00, 2.50e+00,      nan],
       [1.46e+02, 6.70e+00, 3.00e+00, 5.20e+00, 2.30e+00,      nan],
       [1.47e+02, 6.30e+00, 2.50e+00, 5.00e+00, 1.90e+00,      nan],
       [1.48e+02, 6.50e+00, 3.00e+00, 5.20e+00, 2.00e+00,      nan],
       [1.49e+02, 6.20e+00, 3.40e+00, 5.40e+00, 2.30e+00,      nan]])
```

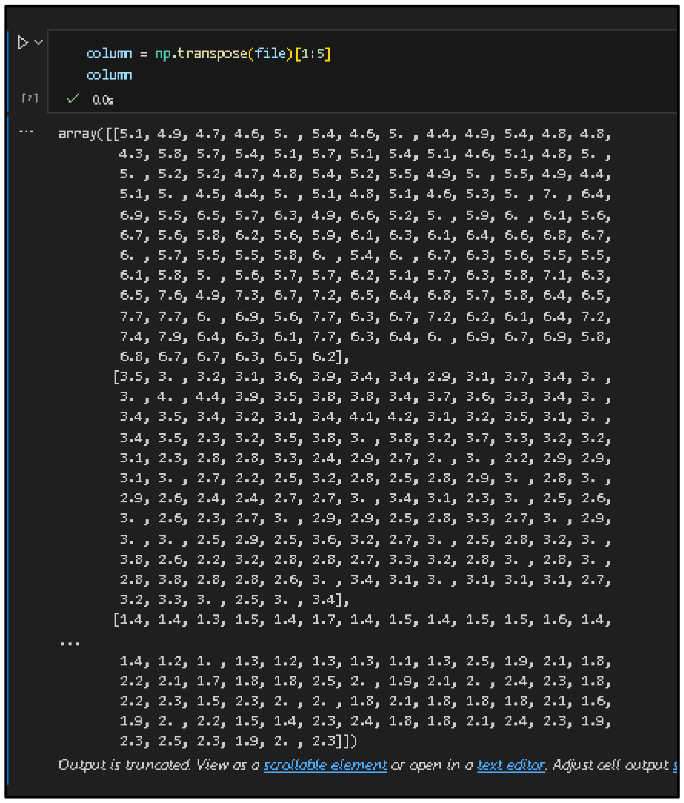
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

3)



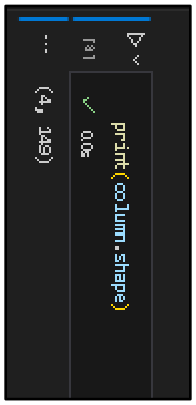
```
columns = file[: , 1:5]
columns
```

```
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, 3.3, 5.7, 2.5],
       [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]])
Output is truncated. View as a scrollable
```



```
column = np.transpose(file)[1:5]
column
```

```
array([[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, 5. ,
        5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
        5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
        6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
        6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
        6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
        6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
        6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
        7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
        7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
        6.8, 6.7, 6.7, 6.3, 6.5, 6.2],
       [3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3. ,
        3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
        3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. ,
        3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3, 3.2, 3.2,
        3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. , 2.2, 2.9, 2.9,
        3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. , 2.8, 3. ,
        2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5, 2.6,
        3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3. , 2.9,
        3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3. , 2.5, 2.8, 3.2, 3. ,
        3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3. , 2.8, 3. ,
        2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1, 3.1, 3.1, 2.7,
        3.2, 3.3, 3. , 2.5, 3. , 3.4],
       [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.4,
...
        1.4, 1.2, 1. , 1.3, 1.2, 1.3, 1.3, 1.1, 1.3, 2.5, 1.9, 2.1, 1.8,
        2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2. , 1.9, 2.1, 2. , 2.4, 2.3, 1.8,
        2.2, 2.3, 1.5, 2.3, 2. , 2. , 1.8, 2.1, 1.8, 1.8, 1.8, 2.1, 1.6,
        1.9, 2. , 2.2, 1.5, 1.4, 2.3, 2.4, 1.8, 1.8, 2.1, 2.4, 2.3, 1.9,
        2.3, 2.5, 2.3, 1.9, 2. , 2.3]])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output
```

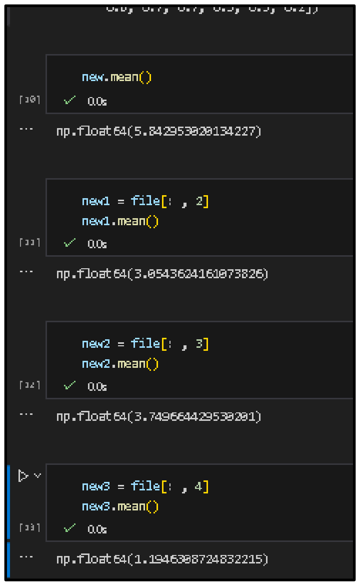


```
print(column.shape)
```

```
(4, 149)
```

<u>TASK:02</u>

1) <u>MEAN:</u>



```
new.mean()
```

```
np.float64(5.842953020134227)
```

```
new1 = file[: , 2]
new1.mean()
```

```
np.float64(3.0543624161073826)
```

```
new2 = file[: , 3]
new2.mean()
```

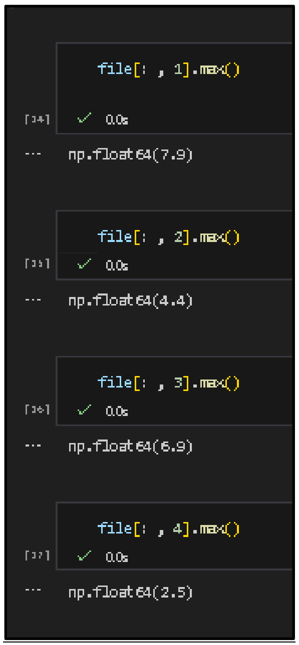
```
np.float64(3.749664429530201)
```

```
new3 = file[: , 4]
new3.mean()
```

```
np.float64(1.1946308724832215)
```

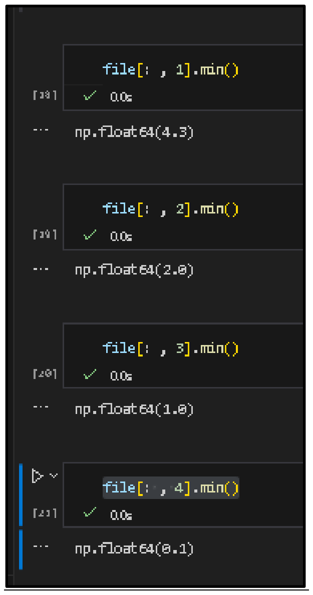
<u>MAX:</u>

3)



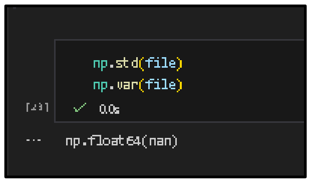
```
file[: , 1].max()
✓ 0.0s
np.float64(7.9)

file[: , 2].max()
✓ 0.0s
np.float64(4.4)

file[: , 3].max()
✓ 0.0s
np.float64(6.9)

file[: , 4].max()
✓ 0.0s
np.float64(2.5)
```

MIN:



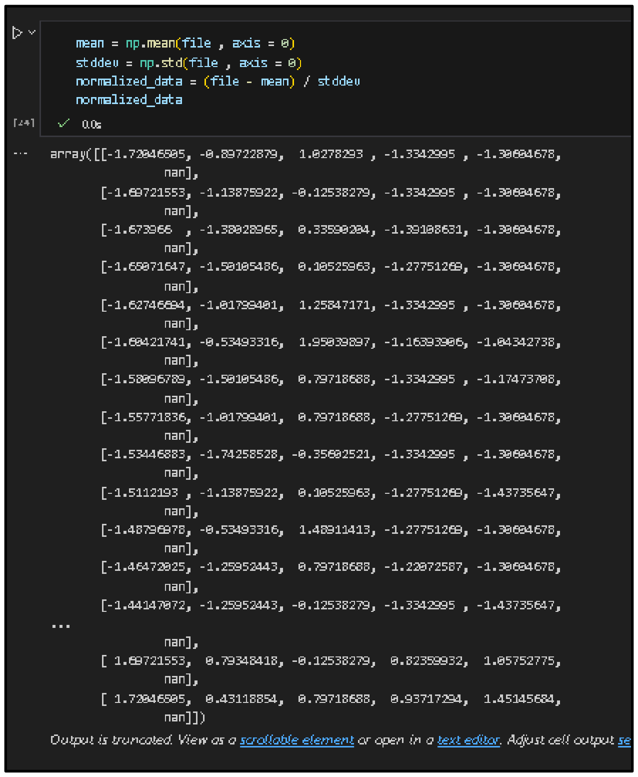
```
file[: , 1].min()
✓ 0.0s
np.float64(4.3)

file[: , 2].min()
✓ 0.0s
np.float64(2.0)

file[: , 3].min()
✓ 0.0s
np.float64(1.0)

file[: , 4].min()
✓ 0.0s
np.float64(0.1)
```

2)



```
np.std(file)
np.var(file)
✓ 0.0s
np.float64(nan)
```

3)



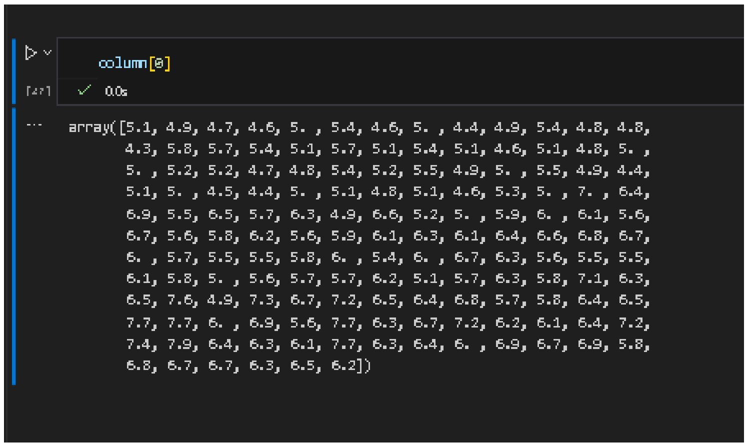
```
mean = np.mean(file , axis = 0)
stddev = np.std(file , axis = 0)
normalized_data = (file - mean) / stddev
normalized_data
```

```
[24]  ✓ 0.0s
```

```
array([[-1.72046505, -0.89722879,  1.0278293 , -1.3342995 , -1.30604678,
               nan],
       [-1.69721553, -1.13875922, -0.12538279, -1.3342995 , -1.30604678,
               nan],
       [-1.673966  , -1.38028965,  0.33590204, -1.39108631, -1.30604678,
               nan],
       [-1.65071647, -1.50105486,  0.10525963, -1.27751269, -1.30604678,
               nan],
       [-1.62746694, -1.01799401,  1.25847171, -1.3342995 , -1.30604678,
               nan],
       [-1.60421741, -0.53493316,  1.95039897, -1.16393906, -1.04342738,
               nan],
       [-1.58096789, -1.50105486,  0.79718688, -1.3342995 , -1.17473708,
               nan],
       [-1.55771836, -1.01799401,  0.79718688, -1.27751269, -1.30604678,
               nan],
       [-1.53446883, -1.74258528, -0.35602521, -1.3342995 , -1.30604678,
               nan],
       [-1.5112193 , -1.13875922,  0.10525963, -1.27751269, -1.43735647,
               nan],
       [-1.48796978, -0.53493316,  1.48911413, -1.27751269, -1.30604678,
               nan],
       [-1.46472025, -1.25952443,  0.79718688, -1.22072587, -1.30604678,
               nan],
       [-1.44147072, -1.25952443, -0.12538279, -1.3342995 , -1.43735647,
...
               nan],
       [ 1.69721553,  0.79348418, -0.12538279,  0.82359932,  1.05752775,
               nan],
       [ 1.72046505,  0.43118854,  0.79718688,  0.93717294,  1.45145684,
               nan]])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output se
```

TASK:03

1)

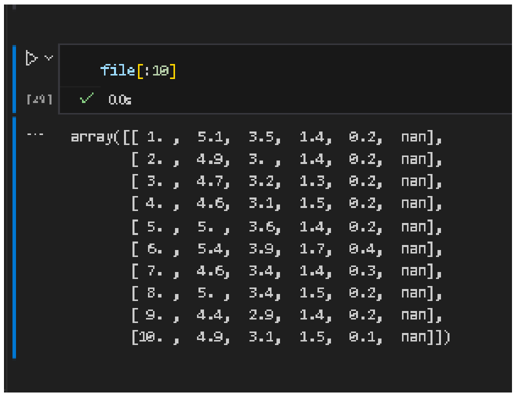


```
column[0]
```

```
[27]  ✓ 0.0s
```

```
array([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, 5. ,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
       6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
       6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
       6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
       6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
       6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
       7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
       7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
       6.8, 6.7, 6.7, 6.3, 6.5, 6.2])
```

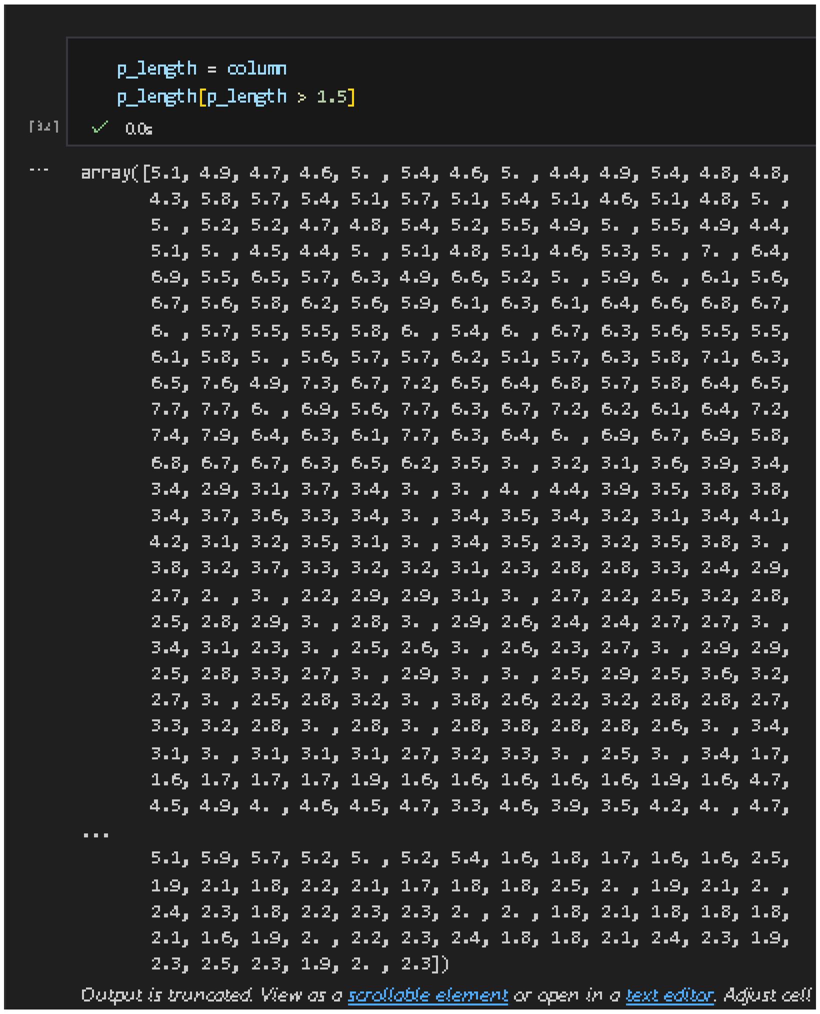
2)



```
file[:10]
```

```
array([[ 1. ,  5.1,  3.5,  1.4,  0.2,  nan],
       [ 2. ,  4.9,  3. ,  1.4,  0.2,  nan],
       [ 3. ,  4.7,  3.2,  1.3,  0.2,  nan],
       [ 4. ,  4.6,  3.1,  1.5,  0.2,  nan],
       [ 5. ,  5. ,  3.6,  1.4,  0.2,  nan],
       [ 6. ,  5.4,  3.9,  1.7,  0.4,  nan],
       [ 7. ,  4.6,  3.4,  1.4,  0.3,  nan],
       [ 8. ,  5. ,  3.4,  1.5,  0.2,  nan],
       [ 9. ,  4.4,  2.9,  1.4,  0.2,  nan],
       [10. ,  4.9,  3.1,  1.5,  0.1,  nan]])
```

3)

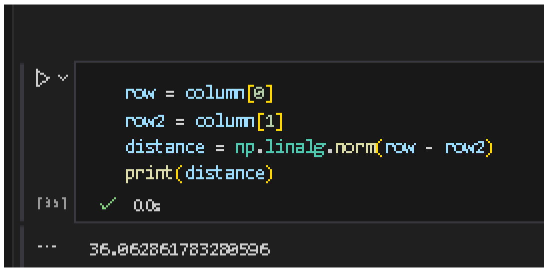


```
p_length = column
p_length[p_length > 1.5]
```

```
array([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, 5. ,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
       6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
       6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
       6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
       6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
       6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
       7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
       7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
       6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4,
       3.4, 2.9, 3.1, 3.7, 3.4, 3. , 3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8,
       3.4, 3.7, 3.6, 3.3, 3.4, 3. , 3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1,
       4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. ,
       3.8, 3.2, 3.7, 3.3, 3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9,
       2.7, 2. , 3. , 2.2, 2.9, 2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8,
       2.5, 2.8, 2.9, 3. , 2.8, 3. , 2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. ,
       3.4, 3.1, 2.3, 3. , 2.5, 2.6, 3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9,
       2.5, 2.8, 3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2,
       2.7, 3. , 2.5, 2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7,
       3.3, 3.2, 2.8, 3. , 2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4,
       3.1, 3. , 3.1, 3.1, 3.1, 2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4, 1.7,
       1.6, 1.7, 1.7, 1.7, 1.9, 1.6, 1.6, 1.6, 1.6, 1.6, 1.9, 1.6, 4.7,
       4.5, 4.9, 4. , 4.6, 4.5, 4.7, 3.3, 4.6, 3.9, 3.5, 4.2, 4. , 4.7,
...
       5.1, 5.9, 5.7, 5.2, 5. , 5.2, 5.4, 1.6, 1.8, 1.7, 1.6, 1.6, 2.5,
       1.9, 2.1, 1.8, 2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2. , 1.9, 2.1, 2. ,
       2.4, 2.3, 1.8, 2.2, 2.3, 2.3, 2. , 2. , 1.8, 2.1, 1.8, 1.8, 1.8,
       2.1, 1.6, 1.9, 2. , 2.2, 2.3, 2.4, 1.8, 1.8, 2.1, 2.4, 2.3, 1.9,
       2.3, 2.5, 2.3, 1.9, 2. , 2.3])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell
```

TASK:04

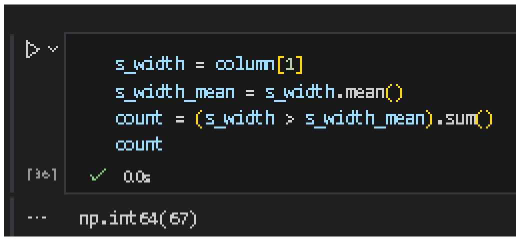
1)



```
row = column[0]
row2 = column[1]
distance = np.linalg.norm(row - row2)
print(distance)
```

```
36.062861783280596
```

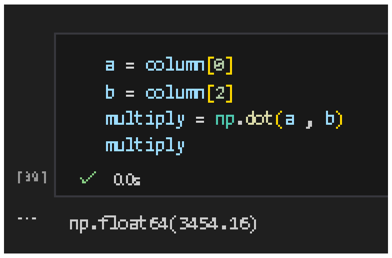
2)



```
s_width = column[1]
s_width_mean = s_width.mean()
count = (s_width > s_width_mean).sum()
count
```

```
np.int64(67)
```

3)

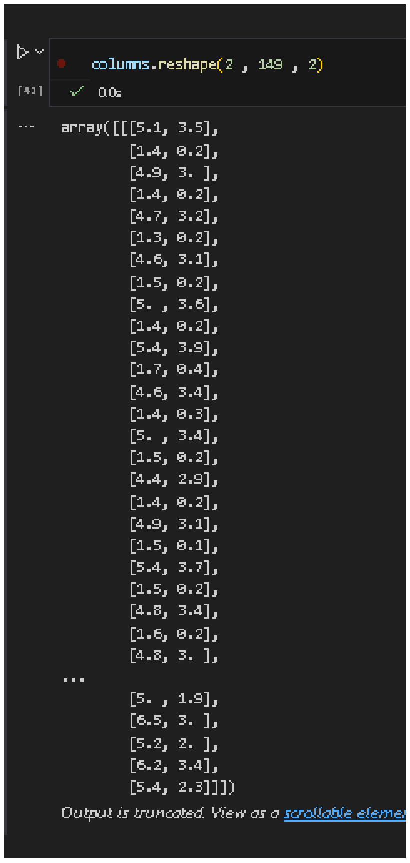


```
a = column[0]
b = column[2]
multiply = np.dot(a , b)
multiply
```

```
np.float64(3454.16)
```

TASK:05

1)

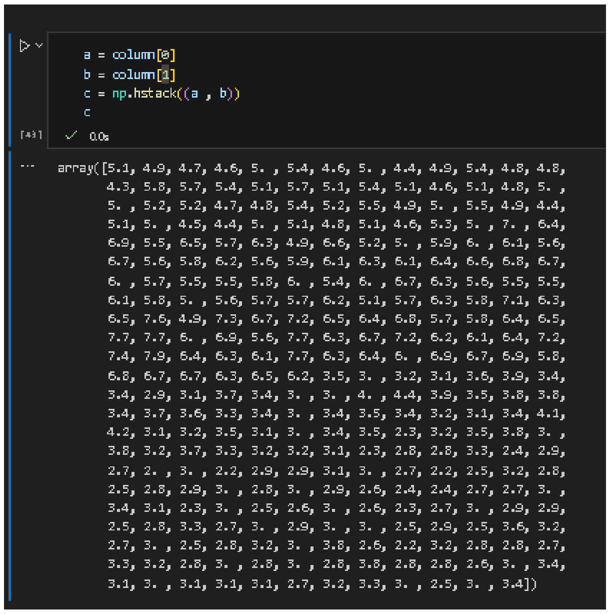


```
columns.reshape(2 , 149 , 2)
```

```
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. ],
 ...
        [5. , 1.9],
        [6.5, 3. ],
        [5.2, 2. ],
        [6.2, 3.4],
        [5.4, 2.3]]])
```

Output is truncated. View as a scrollable elemen

2)



```
a = column[0]
b = column[1]
c = np.hstack((a , b))
c
```

```
array([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, 5. ,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
       6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
       6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
       6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
       6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
       6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
       7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
       7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
       6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4,
       3.4, 2.9, 3.1, 3.7, 3.4, 3. , 3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8,
       3.4, 3.7, 3.6, 3.3, 3.4, 3. , 3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1,
       4.2, 3.1, 3.2, 3.5, 3.1, 3. , 3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. ,
       3.8, 3.2, 3.7, 3.3, 3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9,
       2.7, 2. , 3. , 2.2, 2.9, 2.9, 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8,
       2.5, 2.8, 2.9, 3. , 2.8, 3. , 2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. ,
       3.4, 3.1, 2.3, 3. , 2.5, 2.6, 3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9,
       2.5, 2.8, 3.3, 2.7, 3. , 2.9, 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2,
       2.7, 3. , 2.5, 2.8, 3.2, 3. , 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7,
       3.3, 3.2, 2.8, 3. , 2.8, 3. , 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4,
       3.1, 3. , 3.1, 3.1, 3.1, 2.7, 3.2, 3.3, 3. , 2.5, 3. , 3.4])
```

Lab No: 3

Objective: Write Python program to demonstrate use of 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 write a 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.

🔄 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−μσz = \frac{x - \mu}{\sigma}

🔢 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.

📊 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).

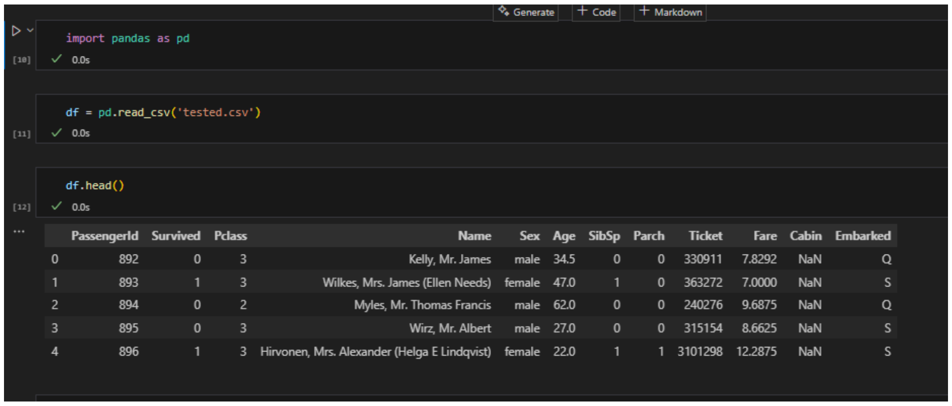
🔄 Task 5: Array Reshaping and Stacking

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.

<u>LAB:03</u>

<u>TASK:01</u>

1)



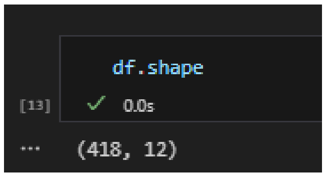
```
import pandas as pd
```

```
df = pd.read_csv('tested.csv')
```

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

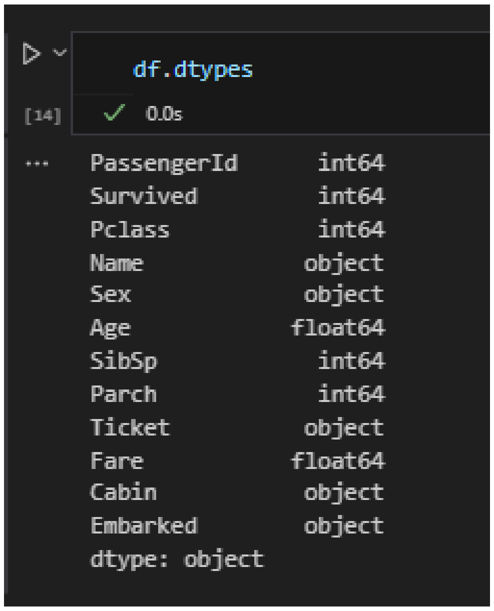
2)



```
df.shape
```

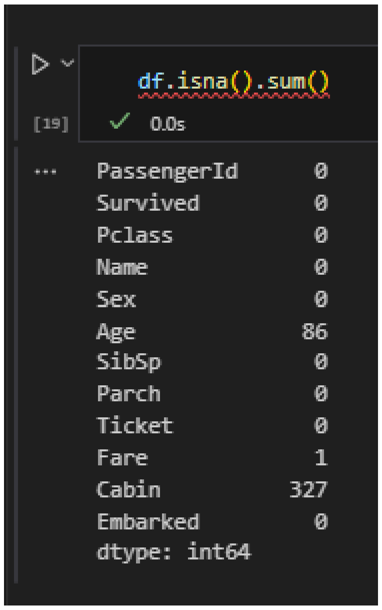
```
(418, 12)
```

3) & 4)



```
df.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

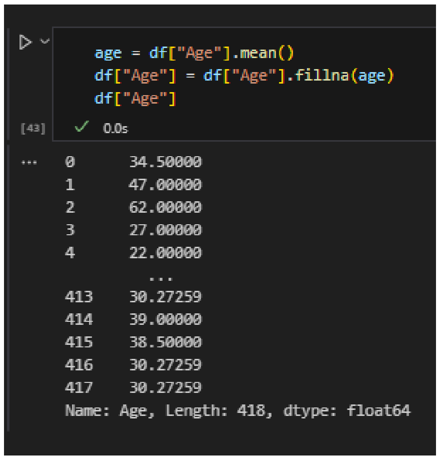


```
df.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

TASK:02

1)

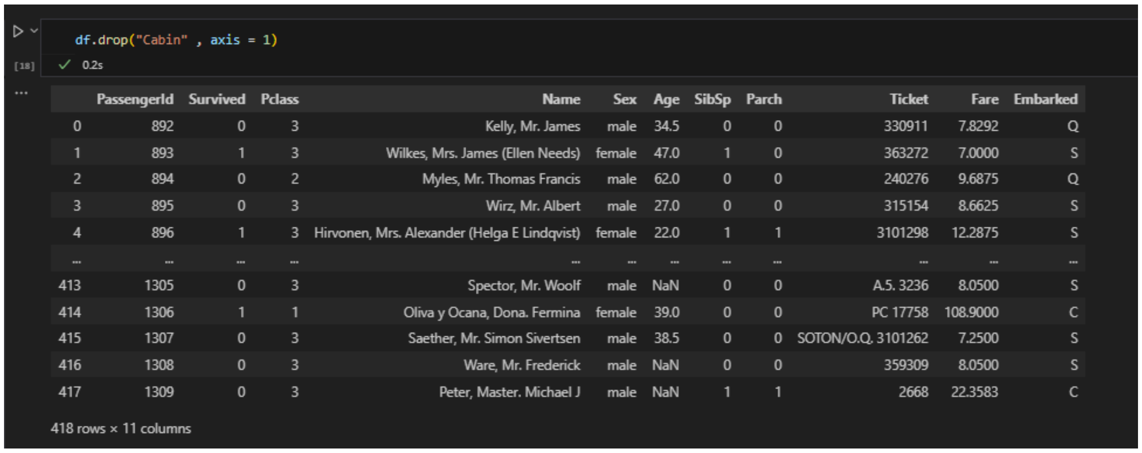


```
age = df["Age"].mean()
df["Age"] = df["Age"].fillna(age)
df["Age"]
```

[43] ✓ 0.0s

```
0      34.50000
1      47.00000
2      62.00000
3      27.00000
4      22.00000
         ...
413    30.27259
414    39.00000
415    38.50000
416    30.27259
417    30.27259
Name: Age, Length: 418, dtype: float64
```

2)



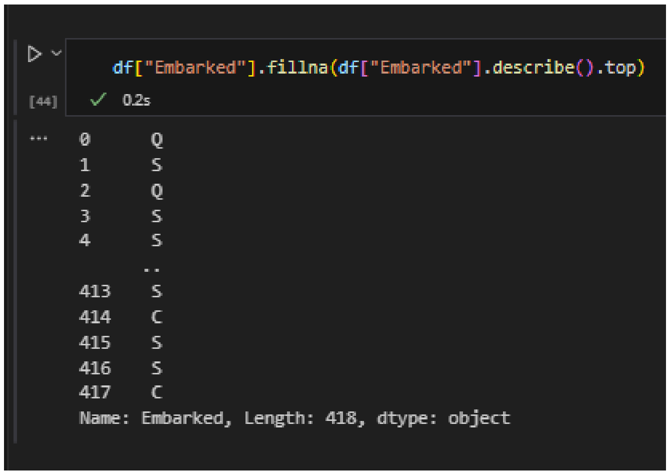
```
df.drop("Cabin" , axis = 1)
```

[18] ✓ 0.2s

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	Q
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	S
...	...	...	...	...	...	...	...	...	...	...	...
413	1305	0	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	S
414	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C
415	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	S
416	1308	0	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	S
417	1309	0	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	C

418 rows × 11 columns

3)



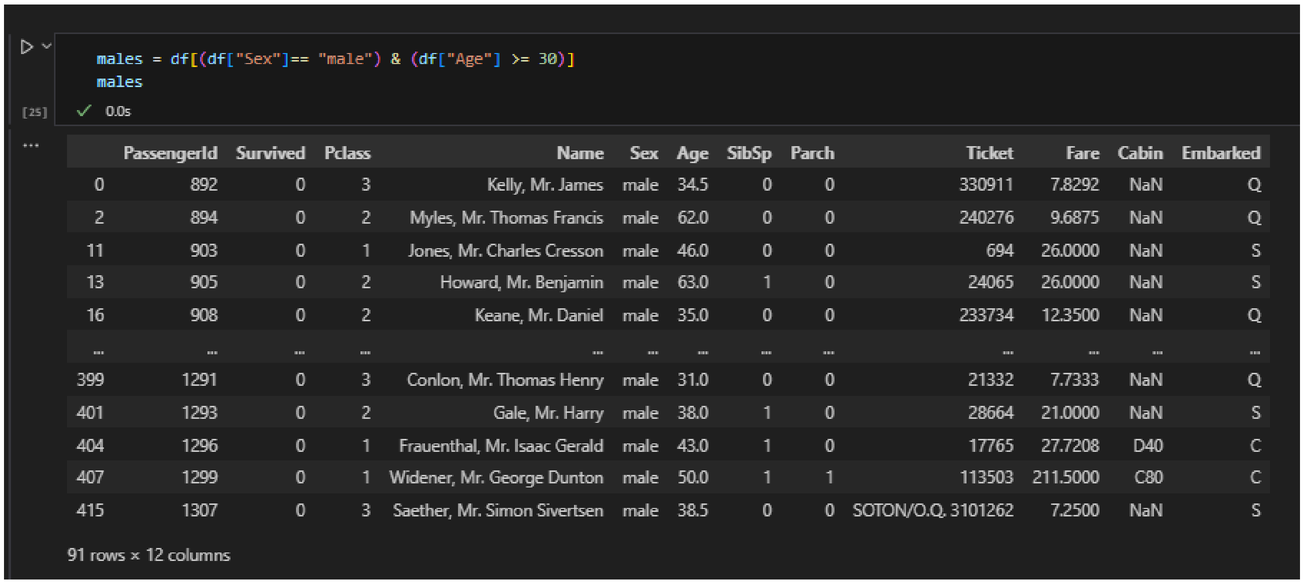
```
df["Embarked"].fillna(df["Embarked"].describe().top)
```

[44] ✓ 0.2s

```
0      Q
1      S
2      Q
3      S
4      S
      ..
413    S
414    C
415    S
416    S
417    C
Name: Embarked, Length: 418, dtype: object
```

TASK:03

1)

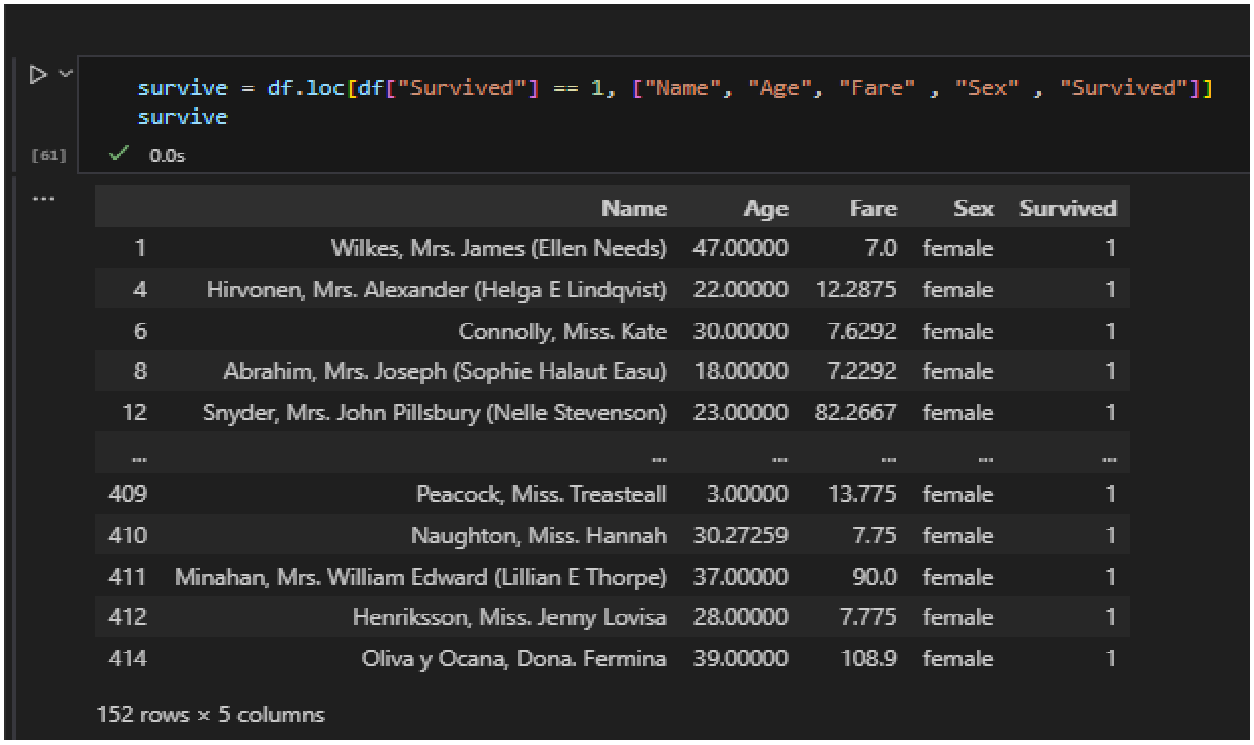


```
males = df[(df["Sex"]== "male") & (df["Age"] >= 30)]
males
```

```
[25]  ✓ 0.0s
```

```
     PassengerId  Survived  Pclass                         Name   Sex   Age  SibSp  Parch               Ticket      Fare  Cabin  Embarked
0            892         0       3             Kelly, Mr. James  male  34.5      0      0               330911    7.8292    NaN         Q
2            894         0       2    Myles, Mr. Thomas Francis  male  62.0      0      0               240276    9.6875    NaN         Q
11           903         0       1    Jones, Mr. Charles Cresson  male  46.0      0      0                  694   26.0000    NaN         S
13           905         0       2         Howard, Mr. Benjamin  male  63.0      1      0                24065   26.0000    NaN         S
16           908         0       2            Keane, Mr. Daniel  male  35.0      0      0               233734   12.3500    NaN         Q
...          ...       ...     ...                          ...   ...   ...    ...    ...                  ...       ...    ...       ...
399         1291         0       3    Conlon, Mr. Thomas Henry  male  31.0      0      0                21332    7.7333    NaN         Q
401         1293         0       2             Gale, Mr. Harry  male  38.0      1      0                28664   21.0000    NaN         S
404         1296         0       1  Frauenthal, Mr. Isaac Gerald  male  43.0      1      0                17765   27.7208    D40         C
407         1299         0       1   Widener, Mr. George Dunton  male  50.0      1      1               113503  211.5000    C80         C
415         1307         0       3  Saether, Mr. Simon Sivertsen  male  38.5      0      0  SOTON/O.Q. 3101262    7.2500    NaN         S

91 rows × 12 columns
```

2)

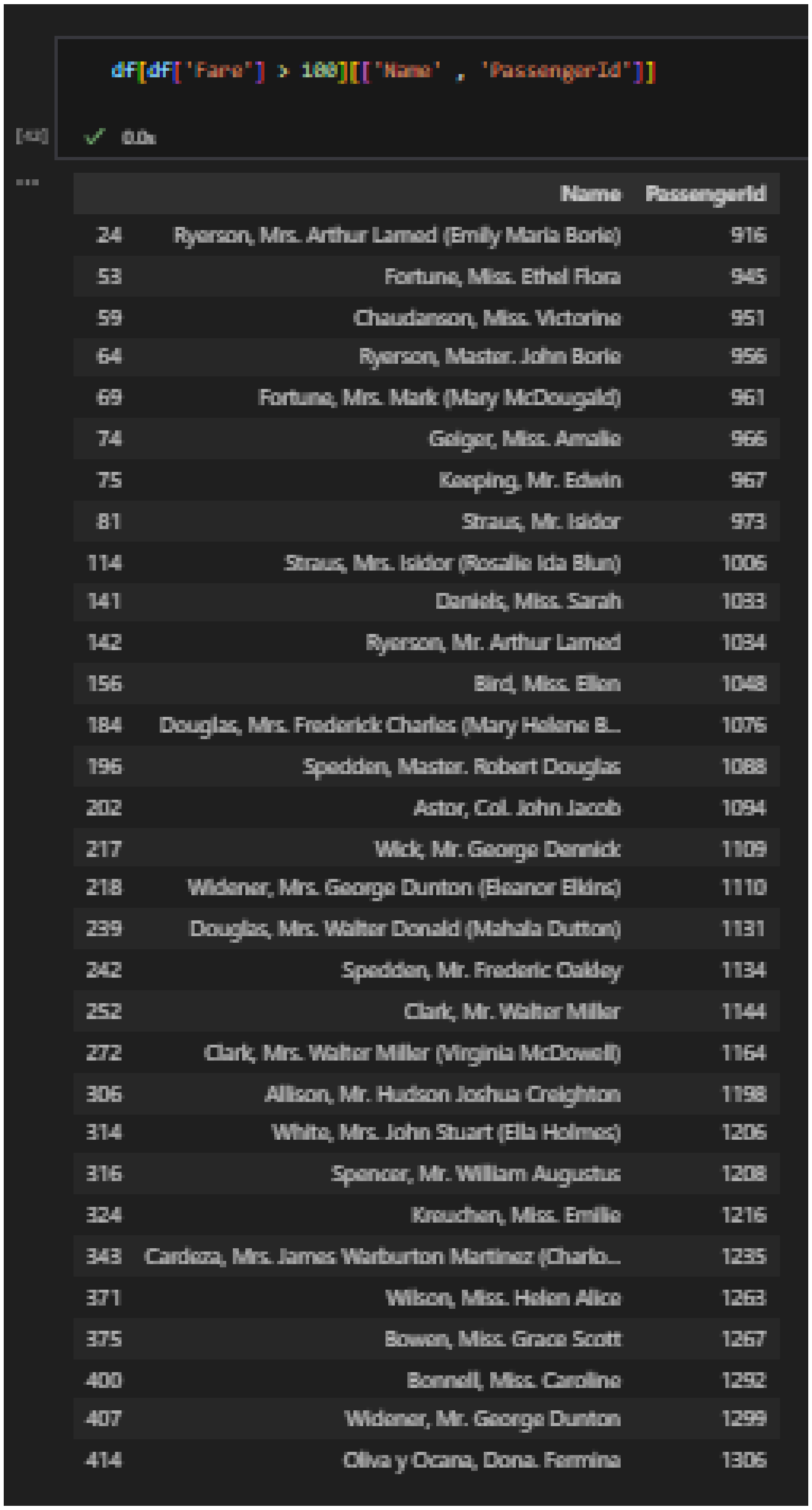


```
survive = df.loc[df["Survived"] == 1, ["Name", "Age", "Fare" , "Sex" , "Survived"]]
survive
```

```
[61]  ✓ 0.0s
```

```
                                          Name       Age     Fare     Sex  Survived
1                   Wilkes, Mrs. James (Ellen Needs)  47.00000      7.0  female         1
4      Hirvonen, Mrs. Alexander (Helga E Lindqvist)  22.00000  12.2875  female         1
6                              Connolly, Miss. Kate  30.00000   7.6292  female         1
8          Abrahim, Mrs. Joseph (Sophie Halaut Easu)  18.00000   7.2292  female         1
12       Snyder, Mrs. John Pillsbury (Nelle Stevenson)  23.00000  82.2667  female         1
...                                              ...       ...      ...     ...       ...
409                        Peacock, Miss. Treasteall   3.00000   13.775  female         1
410                          Naughton, Miss. Hannah  30.27259     7.75  female         1
411  Minahan, Mrs. William Edward (Lillian E Thorpe)  37.00000     90.0  female         1
412                    Henriksson, Miss. Jenny Lovisa  28.00000    7.775  female         1
414                      Oliva y Ocana, Dona. Fermina  39.00000    108.9  female         1

152 rows × 5 columns
```

3)



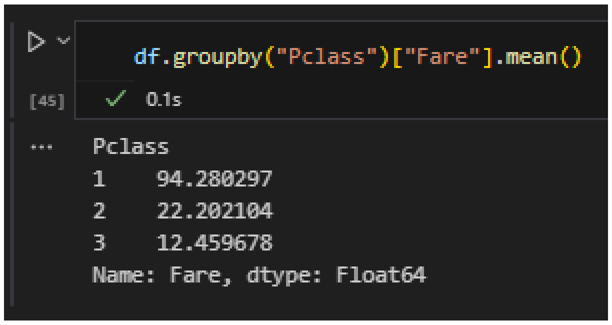
```
df[df['Fare'] > 100][['Name' , 'PassengerId']]
```

```
[43]   ✓ 0.0s
```

	Name	PassengerId
24	Ryerson, Mrs. Arthur Larned (Emily Maria Borie)	916
53	Fortune, Miss. Ethel Flora	945
59	Chaudanson, Miss. Victorine	951
64	Ryerson, Master. John Borie	956
69	Fortune, Mrs. Mark (Mary McDougald)	961
74	Geiger, Miss. Amalie	966
75	Keeping, Mr. Edwin	967
81	Straus, Mr. Isidor	973
114	Straus, Mrs. Isidor (Rosalie Ida Blun)	1006
141	Daniels, Miss. Sarah	1033
142	Ryerson, Mr. Arthur Larned	1034
156	Bird, Miss. Ellen	1048
184	Douglas, Mrs. Frederick Charles (Mary Helene B...	1076
196	Spedden, Master. Robert Douglas	1088
202	Astor, Col. John Jacob	1094
217	Wick, Mr. George Dennick	1109
218	Widener, Mrs. George Dunton (Eleanor Elkins)	1110
239	Douglas, Mrs. Walter Donald (Mahala Dutton)	1131
242	Spedden, Mr. Frederic Oakley	1134
252	Clark, Mr. Walter Miller	1144
272	Clark, Mrs. Walter Miller (Virginia McDowell)	1164
306	Allison, Mr. Hudson Joshua Creighton	1198
314	White, Mrs. John Stuart (Ella Holmes)	1206
316	Spencer, Mr. William Augustus	1208
324	Kreuchen, Miss. Emilie	1216
343	Cardeza, Mrs. James Warburton Martinez (Charlo...	1235
371	Wilson, Miss. Helen Alice	1263
375	Bowen, Miss. Grace Scott	1267
400	Bonnell, Miss. Caroline	1292
407	Widener, Mr. George Dunton	1299
414	Oliva y Ocana, Dona. Fermina	1306

TASK:04

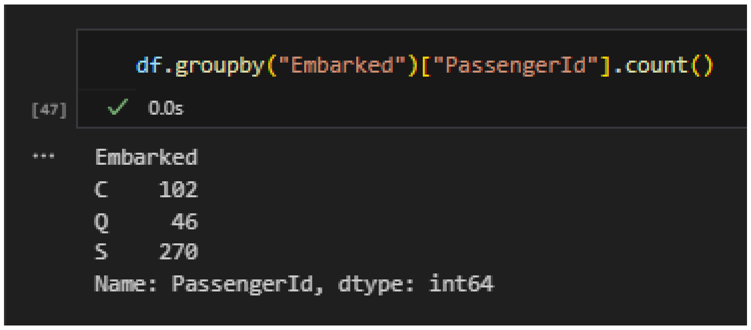
1)



```
[45]
df.groupby("Pclass")["Fare"].mean()
✓ 0.1s
```

```
Pclass
1    94.280297
2    22.202104
3    12.459678
Name: Fare, dtype: float64
```

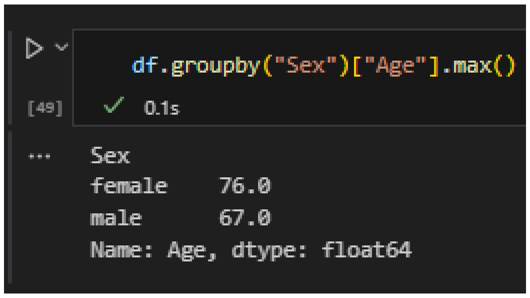
2)



```
[47]
df.groupby("Embarked")["PassengerId"].count()
✓ 0.0s
```

```
Embarked
C    102
Q     46
S    270
Name: PassengerId, dtype: int64
```

3)

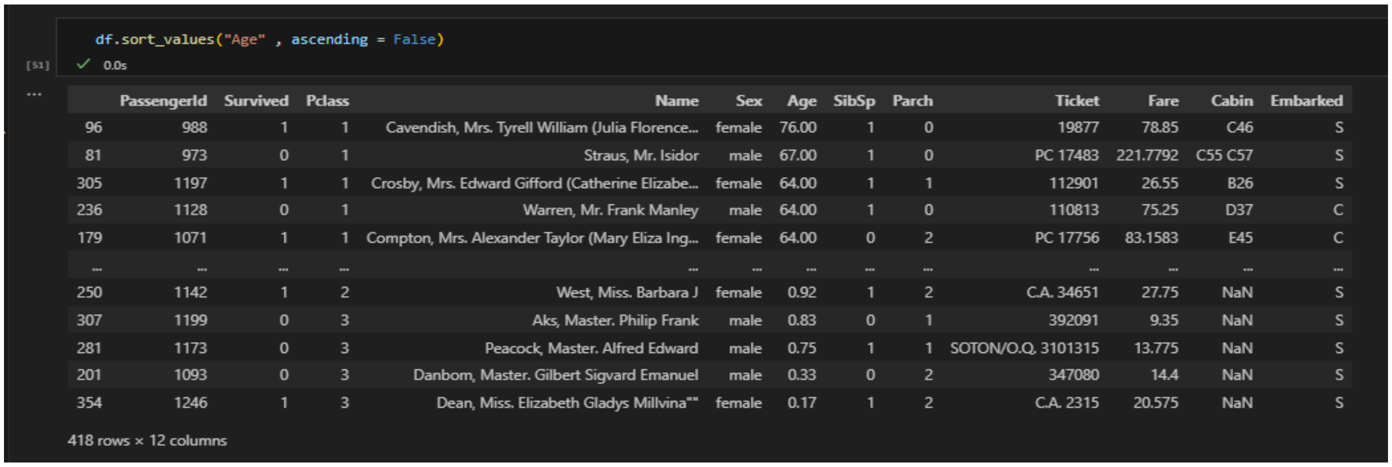


```
[49]
df.groupby("Sex")["Age"].max()
✓ 0.1s
```

```
Sex
female    76.0
male      67.0
Name: Age, dtype: float64
```

TASK:05

1)

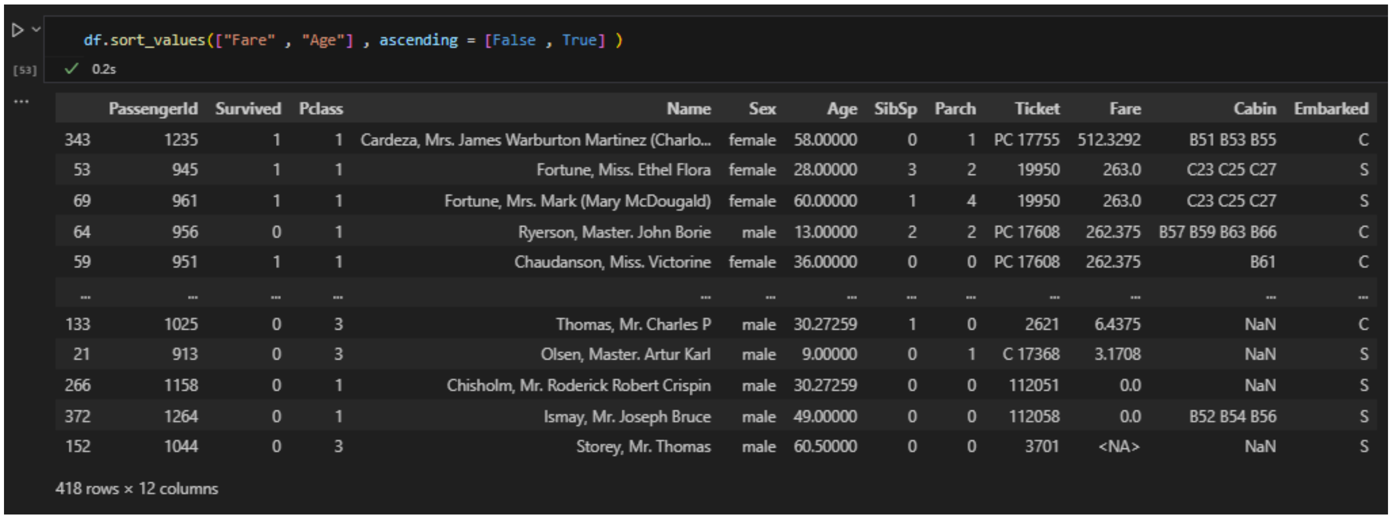


```
df.sort_values("Age" , ascending = False)
```

```
[51]  ✓ 0.0s
```

```
     PassengerId  Survived  Pclass                                             Name     Sex    Age  SibSp  Parch              Ticket      Fare    Cabin  Embarked
96           988         1       1       Cavendish, Mrs. Tyrell William (Julia Florence...  female  76.00      1      0               19877     78.85      C46         S
81           973         0       1                                     Straus, Mr. Isidor    male  67.00      1      0            PC 17483  221.7792  C55 C57         S
305         1197         1       1      Crosby, Mrs. Edward Gifford (Catherine Elizabe...  female  64.00      1      1              112901     26.55      B26         S
236         1128         0       1                                Warren, Mr. Frank Manley    male  64.00      1      0              110813     75.25      D37         C
179         1071         1       1   Compton, Mrs. Alexander Taylor (Mary Eliza Ing...  female  64.00      0      2            PC 17756   83.1583      E45         C
...          ...       ...     ...                                                    ...     ...    ...    ...    ...                 ...       ...      ...       ...
250         1142         1       2                                   West, Miss. Barbara J  female   0.92      1      2           C.A. 34651     27.75      NaN         S
307         1199         0       3                                 Aks, Master. Philip Frank    male   0.83      0      1              392091      9.35      NaN         S
281         1173         0       3                             Peacock, Master. Alfred Edward    male   0.75      1      1  SOTON/O.Q. 3101315    13.775      NaN         S
201         1093         0       3                      Danbom, Master. Gilbert Sigvard Emanuel    male   0.33      0      2              347080      14.4      NaN         S
354         1246         1       3                   Dean, Miss. Elizabeth Gladys Millvina""  female   0.17      1      2            C.A. 2315    20.575      NaN         S

418 rows × 12 columns
```

2)



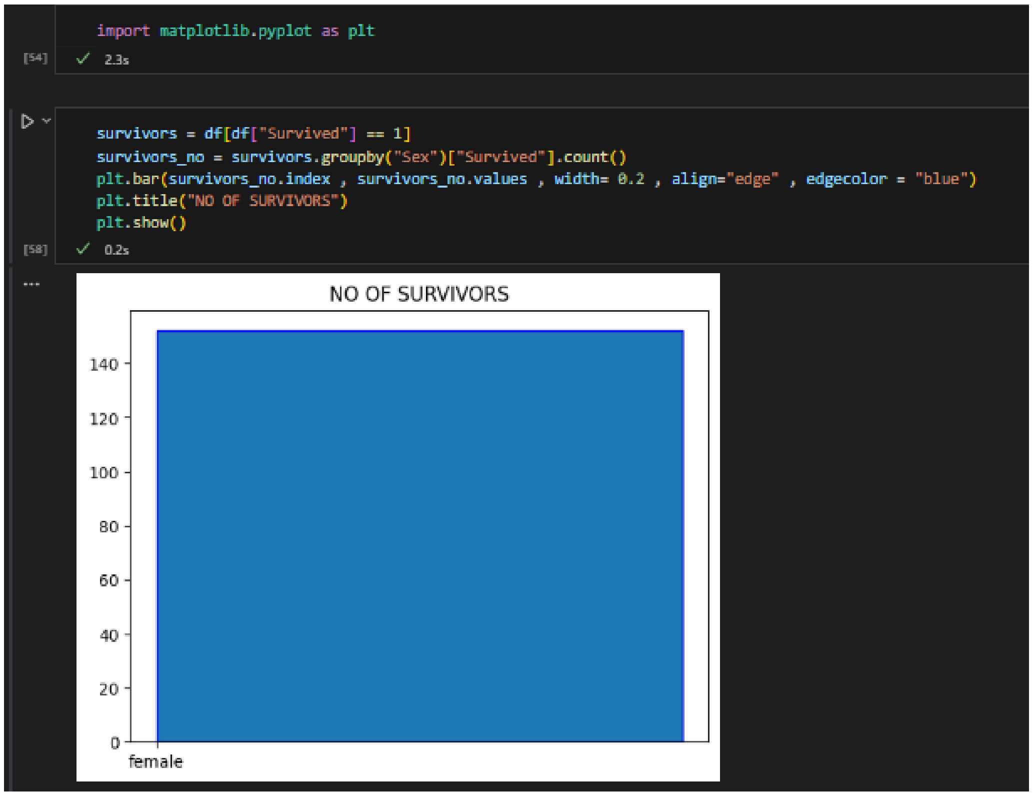
```
df.sort_values(["Fare" , "Age"] , ascending = [False , True] )
```

```
[53]  ✓ 0.2s
```

```
     PassengerId  Survived  Pclass                                          Name     Sex       Age  SibSp  Parch     Ticket      Fare            Cabin  Embarked
343         1235         1       1  Cardeza, Mrs. James Warburton Martinez (Charlo...  female  58.00000      0      1   PC 17755  512.3292      B51 B53 B55         C
53           945         1       1                          Fortune, Miss. Ethel Flora  female  28.00000      3      2      19950     263.0      C23 C25 C27         S
69           961         1       1                 Fortune, Mrs. Mark (Mary McDougald)  female  60.00000      1      4      19950     263.0      C23 C25 C27         S
64           956         0       1                          Ryerson, Master. John Borie    male  13.00000      2      2   PC 17608   262.375  B57 B59 B63 B66         C
59           951         1       1                        Chaudanson, Miss. Victorine  female  36.00000      0      0   PC 17608   262.375              B61         C
...          ...       ...     ...                                               ...     ...       ...    ...    ...        ...       ...              ...       ...
133         1025         0       3                             Thomas, Mr. Charles P    male  30.27259      1      0       2621    6.4375              NaN         C
21           913         0       3                            Olsen, Master. Artur Karl    male   9.00000      0      1    C 17368    3.1708              NaN         S
266         1158         0       1                  Chisholm, Mr. Roderick Robert Crispin    male  30.27259      0      0     112051       0.0              NaN         S
372         1264         0       1                             Ismay, Mr. Joseph Bruce    male  49.00000      0      0     112058       0.0      B52 B54 B56         S
152         1044         0       3                                 Storey, Mr. Thomas    male  60.50000      0      0       3701      <NA>              NaN         S

418 rows × 12 columns
```

TASK:06

1)



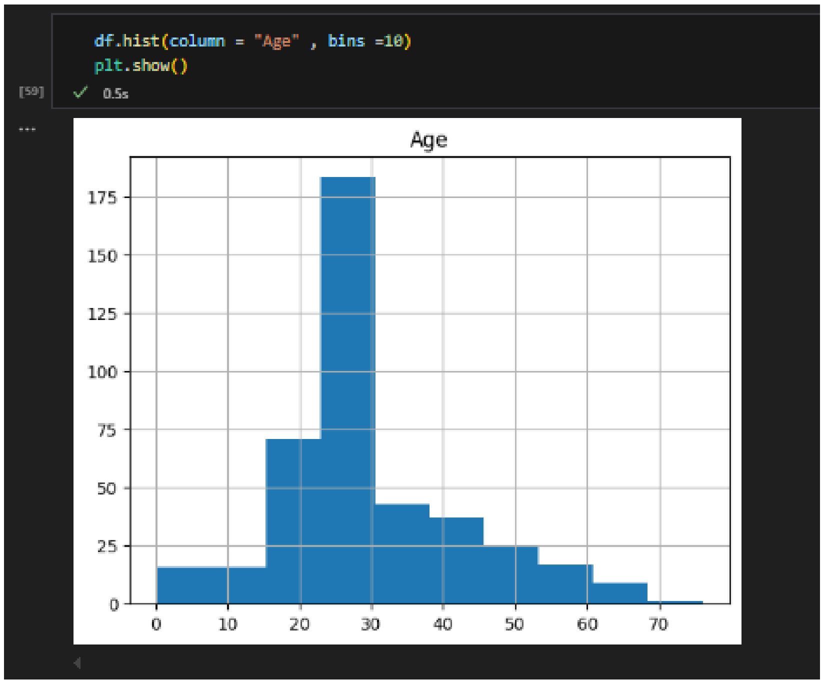
```
import matplotlib.pyplot as plt
```

```
[54] ✓ 2.3s
```

```
survivors = df[df["Survived"] == 1]
survivors_no = survivors.groupby("Sex")["Survived"].count()
plt.bar(survivors_no.index , survivors_no.values , width= 0.2 , align="edge" , edgecolor = "blue")
plt.title("NO OF SURVIVORS")
plt.show()
```

```
[58] ✓ 0.2s
```

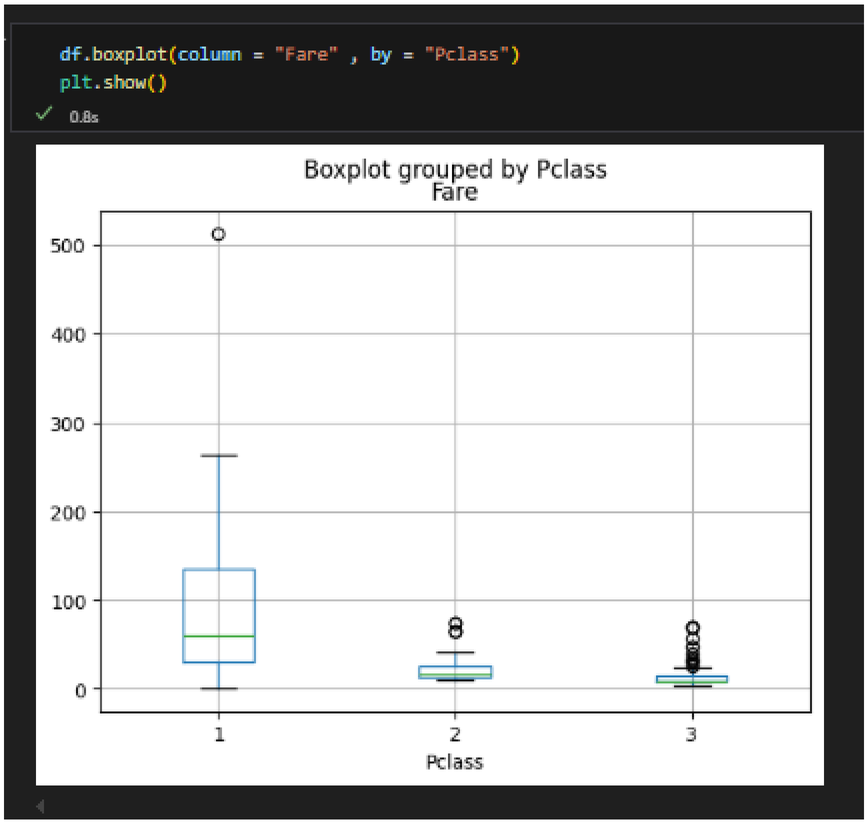
2)



```
df.hist(column = "Age" , bins =10)
plt.show()
```

```
[59] ✓ 0.5s
```

3)



```
df.boxplot(column = "Fare" , by = "Pclass")
plt.show()
```

```
0.8s
```

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

Part B: Data Analysis

1. Count how many students fall into each Performance_Level.
2. Compute the average Study_Hours_Per_Week and Attendance_Percentage for each performance level.
3. Group by Gender and School_Type, and compute:
   - Mean Total_Score
   - Pass percentage (Passed == "Yes")
4. Find the top 5 students with the highest Study_Hours_Per_Week who did not pass.

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".

2. Add a column Eligible_for_Scholarship:
    - TRUE if Total_Score ≥ 230 and Attendance_Percentage > 90, else FALSE

Part D: Data Visualization

Using ggplot2:

1. Bar chart showing number of students in each Performance_Level
2. Boxplot comparing Total_Score across Gender
3. Scatter plot of Study_Hours_Per_Week vs Total_Score, colored by Passed
4. Line chart showing average Total_Score by Attendance_Percentage bins (use cut() to bin attendance)

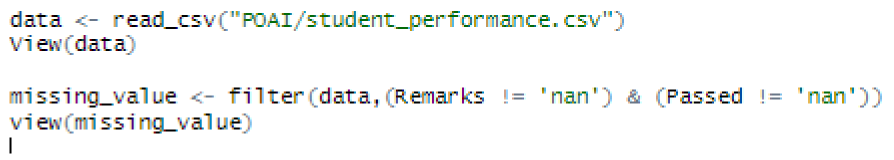
Questions:

- Are high study hours always linked to passing?
- Do school type and gender impact overall performance?
- Which factors best predict scholarship eligibility?

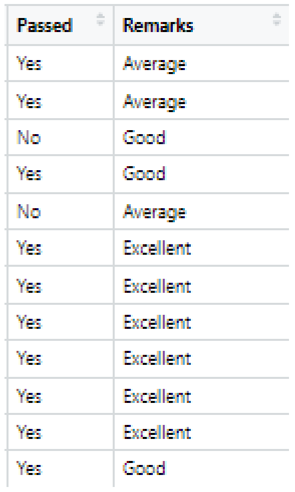
LAB:05

Part A:

1)

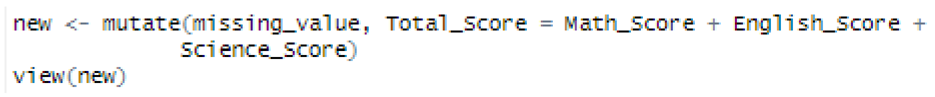


```
data <- read_csv("POAI/student_performance.csv")
View(data)

missing_value <- filter(data,(Remarks != 'nan') & (Passed != 'nan'))
view(missing_value)
```

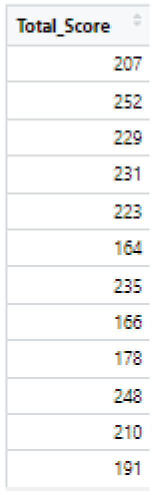


Passed	Remarks
Yes	Average
Yes	Average
No	Good
Yes	Good
No	Average
Yes	Excellent
Yes	Excellent
Yes	Excellent
Yes	Excellent
Yes	Excellent
Yes	Excellent
Yes	Good

2)



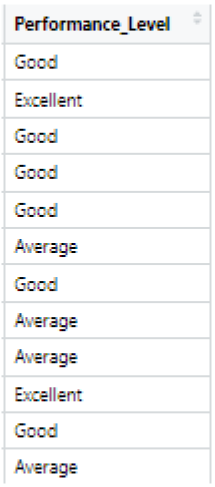
```
new <- mutate(missing_value, Total_Score = Math_Score + English_Score +
                Science_Score)
view(new)
```



Total_Score
207
252
229
231
223
164
235
166
178
248
210
191

3)

```
pl <- new %>%
          mutate(Performance_Level = ifelse(Total_Score >= 240, 'Excellent',
          ifelse(Total_Score >= 200 & Total_Score < 240, 'Good',
          ifelse(Total_Score >= 150 & Total_Score < 200, 'Average','Poor'))))

View(pl)
```

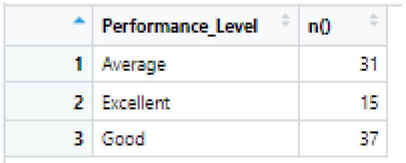


Performance_Level
Good
Excellent
Good
Good
Good
Average
Good
Average
Average
Excellent
Good
Average

Part B:

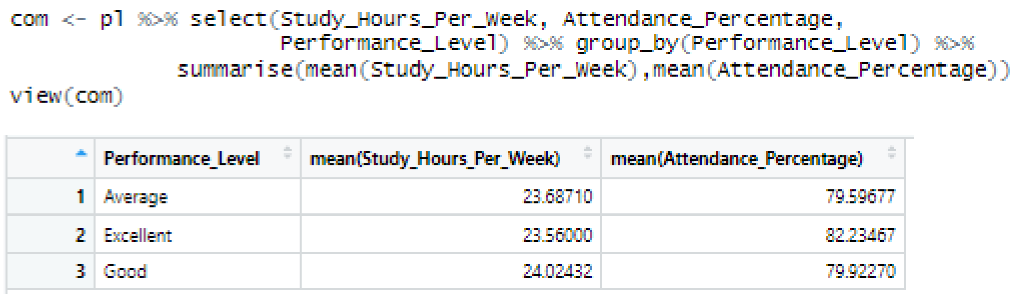
1)

```
count <- pl %>% group_by(Performance_Level) %>% summarise(n())
view(count)
```



	Performance_Level	n()
1	Average	31
2	Excellent	15
3	Good	37

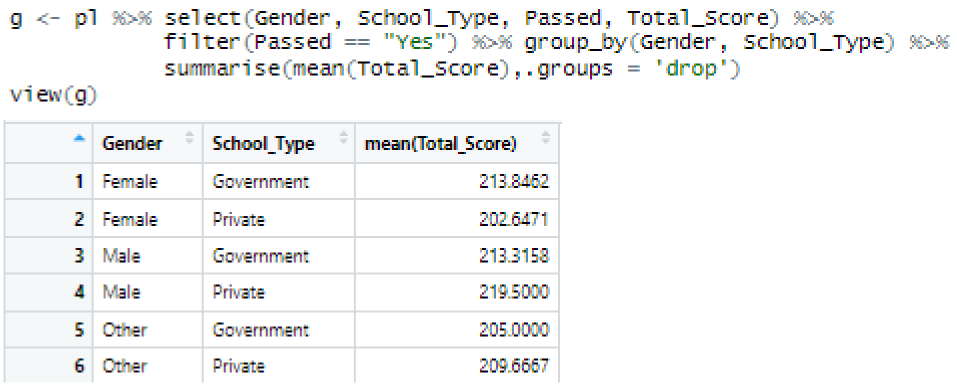
2)



```
com <- pl %>% select(Study_Hours_Per_Week, Attendance_Percentage,
                     Performance_Level) %>% group_by(Performance_Level) %>%
             summarise(mean(Study_Hours_Per_Week),mean(Attendance_Percentage))
view(com)
```

	Performance_Level	mean(Study_Hours_Per_Week)	mean(Attendance_Percentage)
1	Average	23.68710	79.59677
2	Excellent	23.56000	82.23467
3	Good	24.02432	79.92270

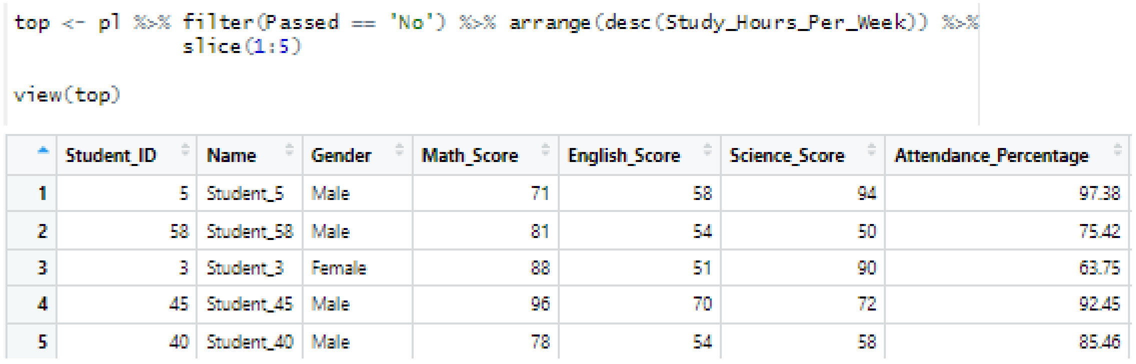
3)



```
g <- pl %>% select(Gender, School_Type, Passed, Total_Score) %>%
           filter(Passed == "Yes") %>% group_by(Gender, School_Type) %>%
           summarise(mean(Total_Score),.groups = 'drop')
view(g)
```

	Gender	School_Type	mean(Total_Score)
1	Female	Government	213.8462
2	Female	Private	202.6471
3	Male	Government	213.3158
4	Male	Private	219.5000
5	Other	Government	205.0000
6	Other	Private	209.6667

4)

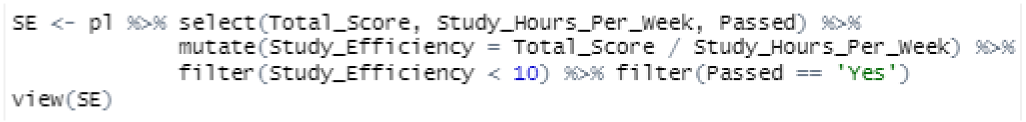


```
top <- pl %>% filter(Passed == 'No') %>% arrange(desc(Study_Hours_Per_Week)) %>%
              slice(1:5)

view(top)
```

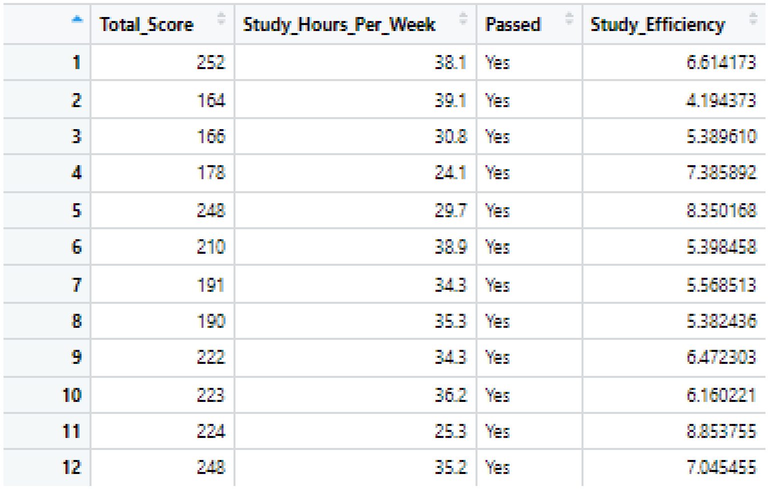
	Student_ID	Name	Gender	Math_Score	English_Score	Science_Score	Attendance_Percentage
1	5	Student_5	Male	71	58	94	97.38
2	58	Student_58	Male	81	54	50	75.42
3	3	Student_3	Female	88	51	90	63.75
4	45	Student_45	Male	96	70	72	92.45
5	40	Student_40	Male	78	54	58	85.46

Part C:

1)

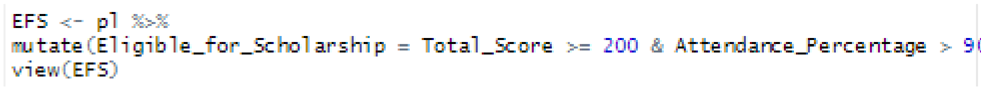


```
SE <- pl %>% select(Total_Score, Study_Hours_Per_Week, Passed) %>%
             mutate(Study_Efficiency = Total_Score / Study_Hours_Per_Week) %>%
             filter(Study_Efficiency < 10) %>% filter(Passed == 'Yes')
view(SE)
```

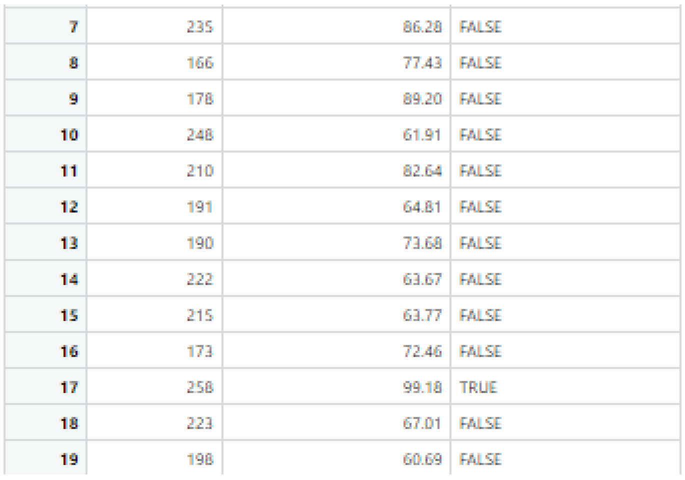


	Total_Score	Study_Hours_Per_Week	Passed	Study_Efficiency
1	252	38.1	Yes	6.614173
2	164	39.1	Yes	4.194373
3	166	30.8	Yes	5.389610
4	178	24.1	Yes	7.385892
5	248	29.7	Yes	8.350168
6	210	38.9	Yes	5.398458
7	191	34.3	Yes	5.568513
8	190	35.3	Yes	5.382436
9	222	34.3	Yes	6.472303
10	223	36.2	Yes	6.160221
11	224	25.3	Yes	8.853755
12	248	35.2	Yes	7.045455

2)



```
EFS <- pl %>%
mutate(Eligible_for_Scholarship = Total_Score >= 200 & Attendance_Percentage > 90)
view(EFS)
```

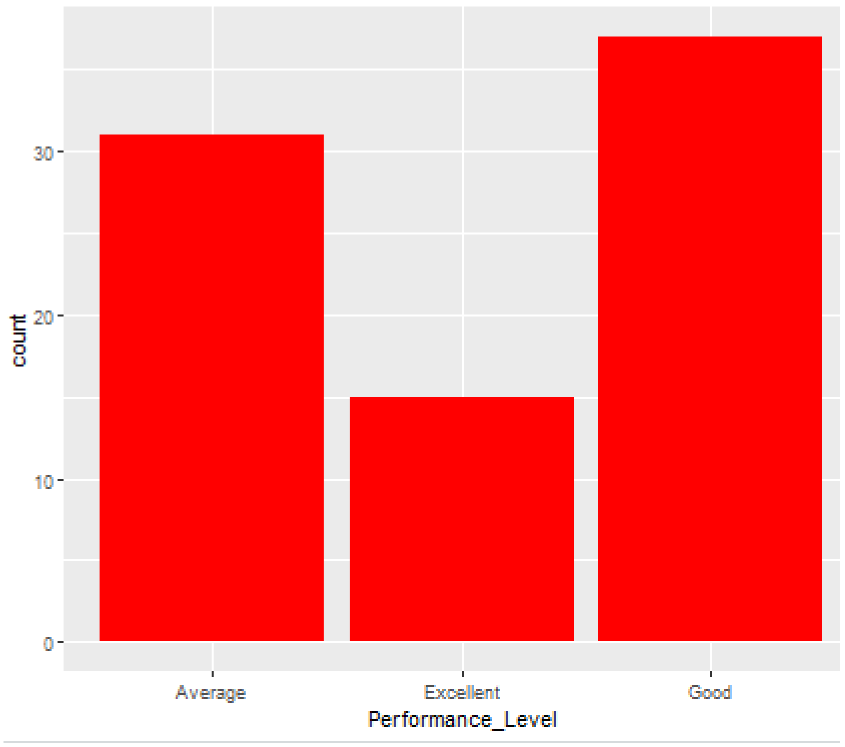


7	235	86.28	FALSE
8	166	77.43	FALSE
9	178	89.20	FALSE
10	248	61.91	FALSE
11	210	82.64	FALSE
12	191	64.81	FALSE
13	190	73.68	FALSE
14	222	63.67	FALSE
15	215	63.77	FALSE
16	173	72.46	FALSE
17	258	99.18	TRUE
18	223	67.01	FALSE
19	198	60.69	FALSE

Part D:

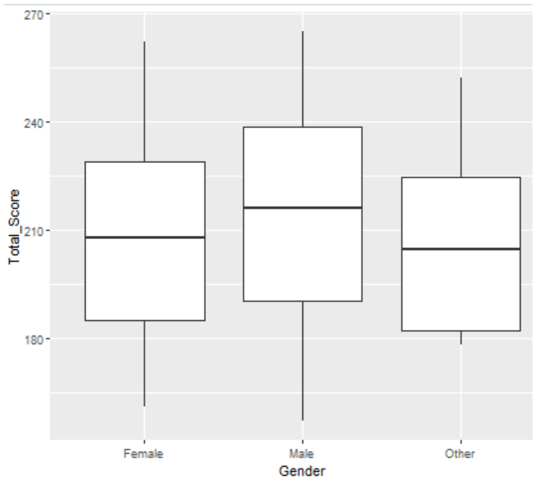
1)

```
bar_plot <- ggplot(pl, aes(Performance_Level)) + geom_bar(fill = "Red")
bar_plot
```



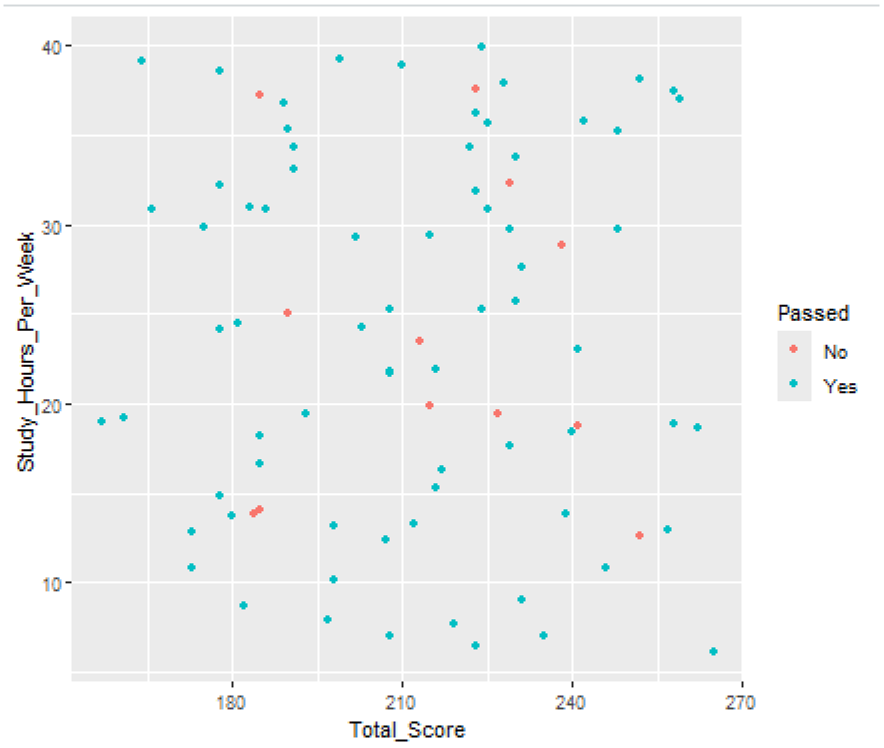
2)

```
bplt <- ggplot(pl, aes(Gender, Total_Score)) + geom_boxplot()
bplt
```



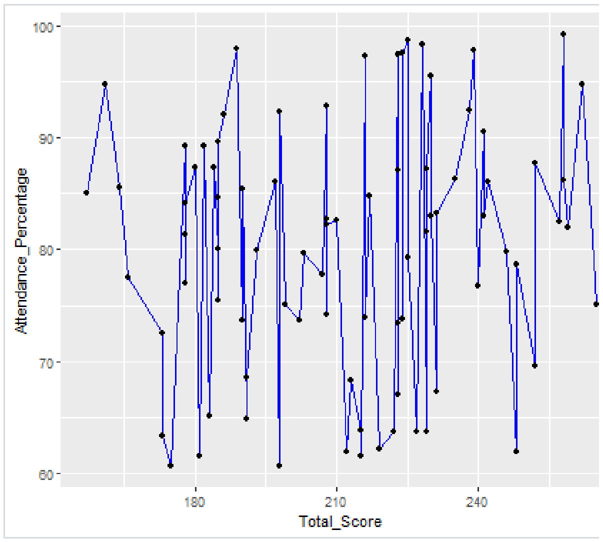
3)

```
splt <- ggplot(pl, aes(Total_Score, Study_Hours_Per_Week , col = Passed)) +
        geom_point()
splt
```



4)

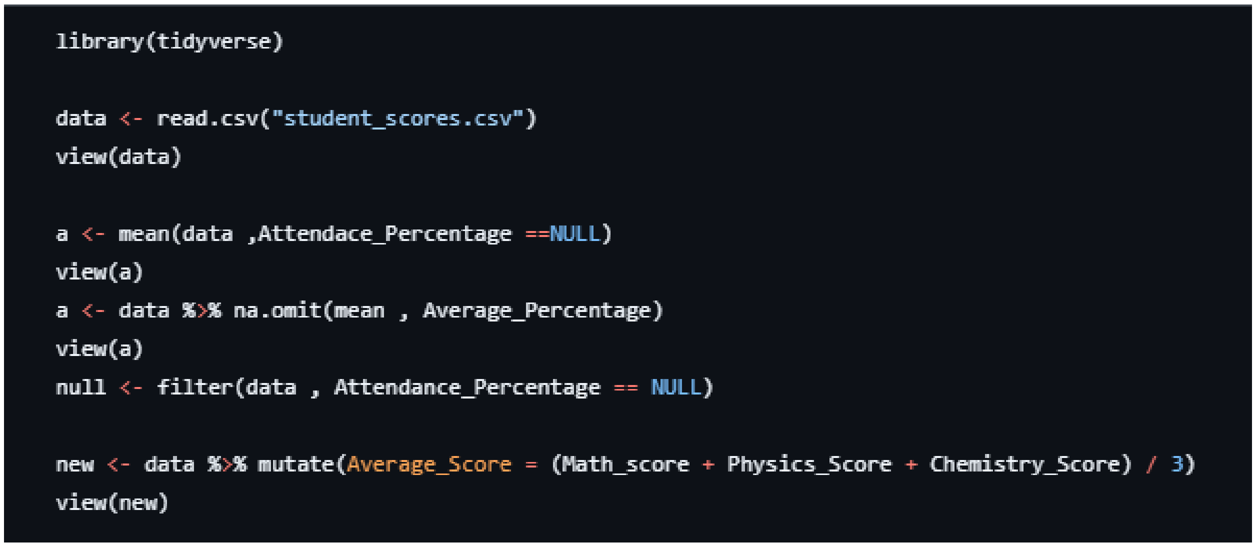
```
lchrt <- ggplot(pl, aes(Total_Score, Attendance_Percentage) ,
                cut(Attendance_Percentage)) + geom_line(col = "Blue") + geom_point()
lchrt
```



Questions:

1) High study hours can’t be always linked with passing because some students with high study hours are not pass.
2) Both school type and gender can influence academic performance the quality of teaching and available resources often play a more significant role than school type or gender alone.
3) Total Score and Attendance Percentage are the strongest predictors of scholarship eligibility, with other factors like study hours and school type.

OEL 1:



```r
library(tidyverse)

data <- read.csv("student_scores.csv")
view(data)

a <- mean(data ,Attendace_Percentage ==NULL)
view(a)
a <- data %>% na.omit(mean , Average_Percentage)
view(a)
null <- filter(data , Attendance_Percentage == NULL)

new <- data %>% mutate(Average_Score = (Math_score + Physics_Score + Chemistry_Score) / 3)
view(new)
```

Lab No: 6

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

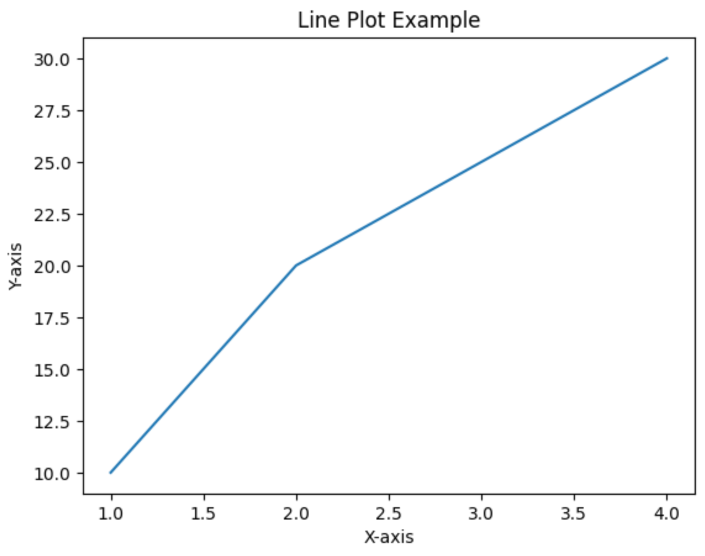
Matplotlib is a powerful 2D plotting library in Python. The module matplotlib.pyplot provides a MATLAB-like interface for creating visualizations.

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

---

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

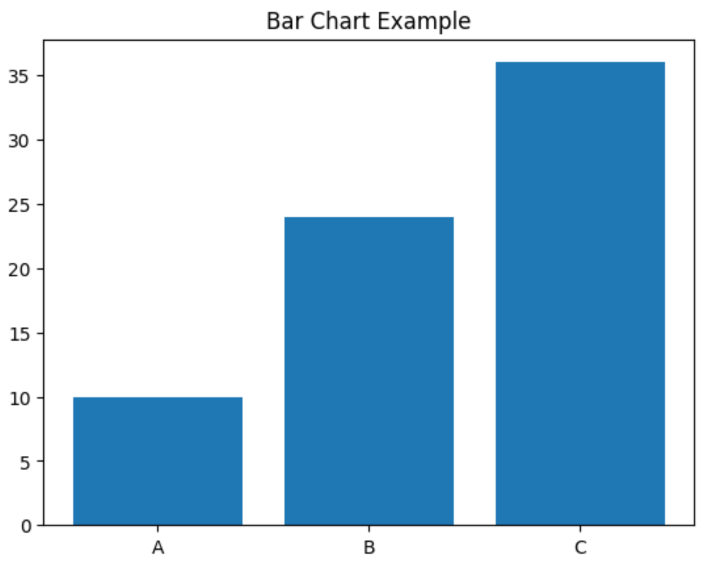
```
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.title("Line Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```



---

3. Bar Chart
    - Represents categorical data with rectangular bars.

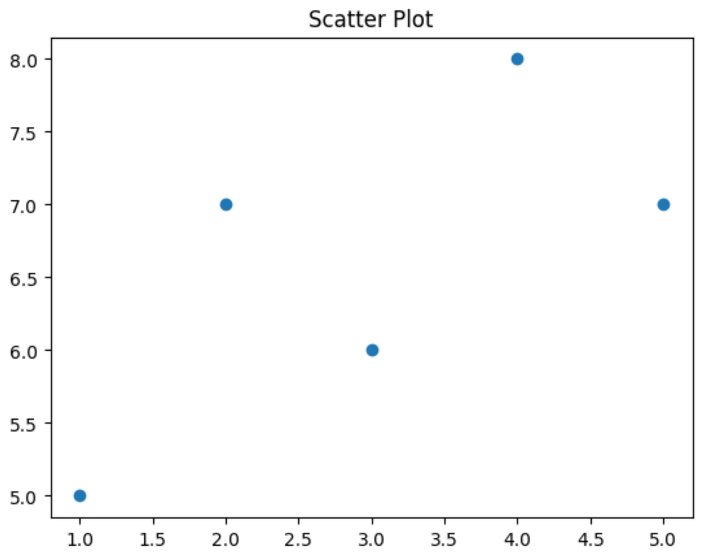
```
categories = ['A', 'B', 'C']
values = [10, 24, 36]
plt.bar(categories, values)
plt.title("Bar Chart Example")
plt.show()
```



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()
```

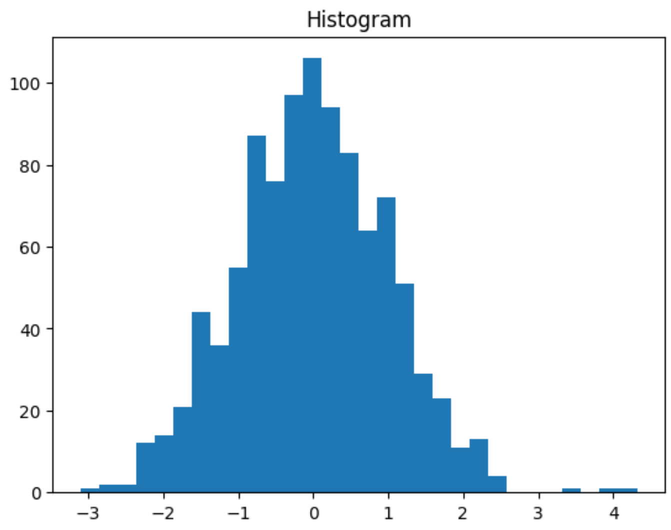


5. Histogram

- 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()
```

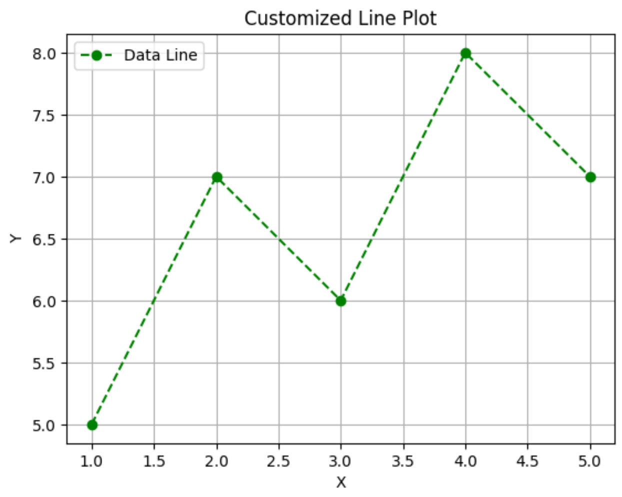


---

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")
```

<u>Tasks: -</u>

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.
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.
3. Bar Plot
    - Plot average total_bill for each day using a bar plot.
4. Histogram
    - Create a histogram of tip values with bins=10 and appropriate labels.
5. Boxplot
    - Create a boxplot of total_bill grouped by day.
6. Pie Chart
    - Show pie chart of smoker vs non-smoker counts.

Lab No: 7

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.

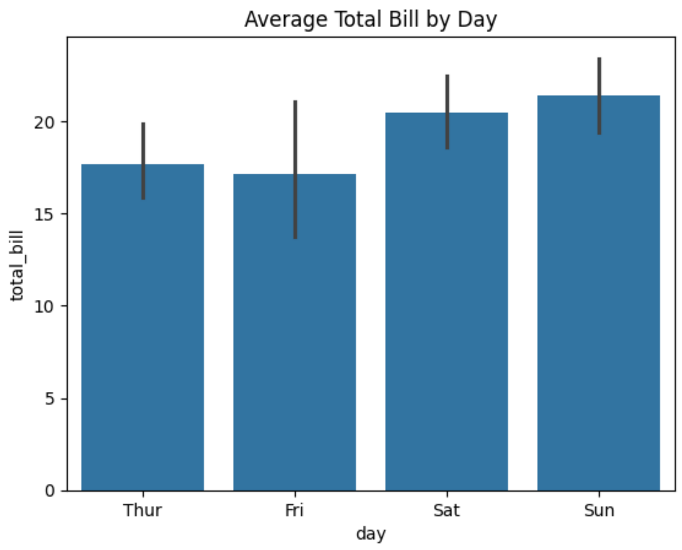
```
df = sns.load_dataset("tips")
```

---

3. Bar Plot

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

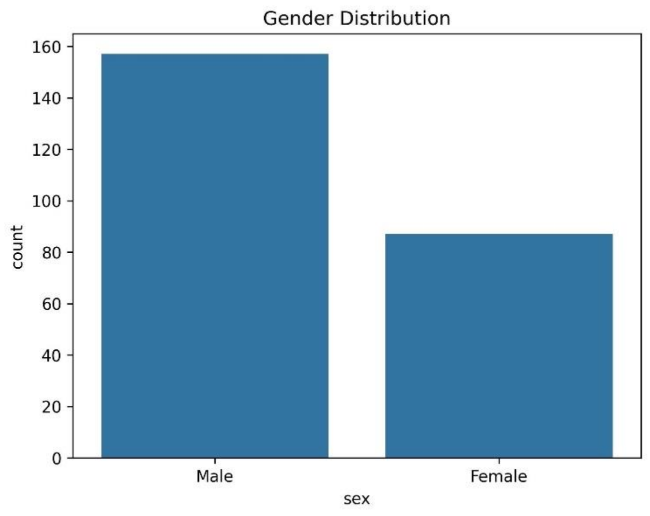
```
sns.barplot(x="day", y="total_bill", data=df)

plt.title("Average Total Bill by Day")

plt.show()
```



4. Count Plot

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

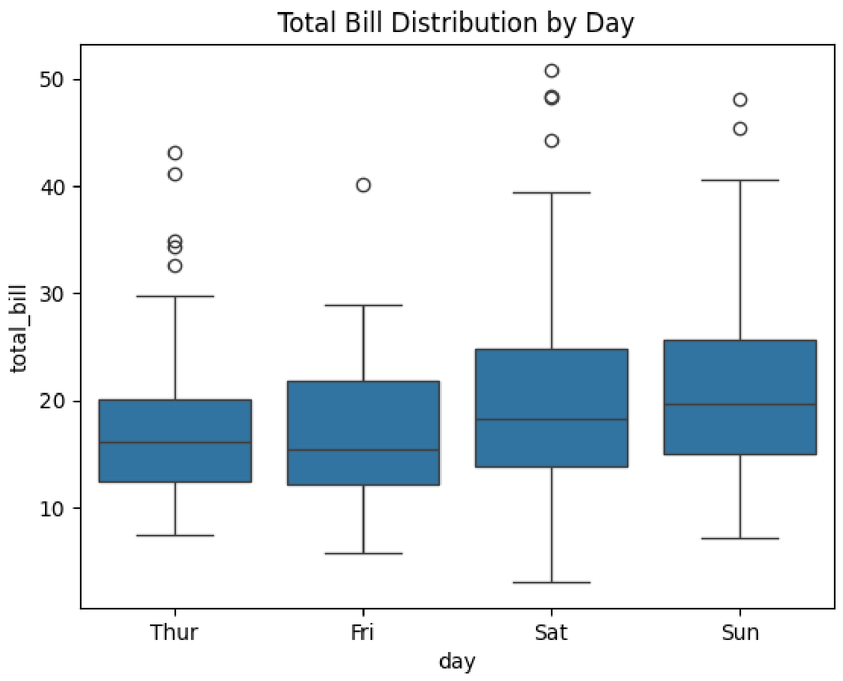
```
sns.countplot(x="sex", data=df)
plt.title("Gender Distribution")
plt.show()
```



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()
```

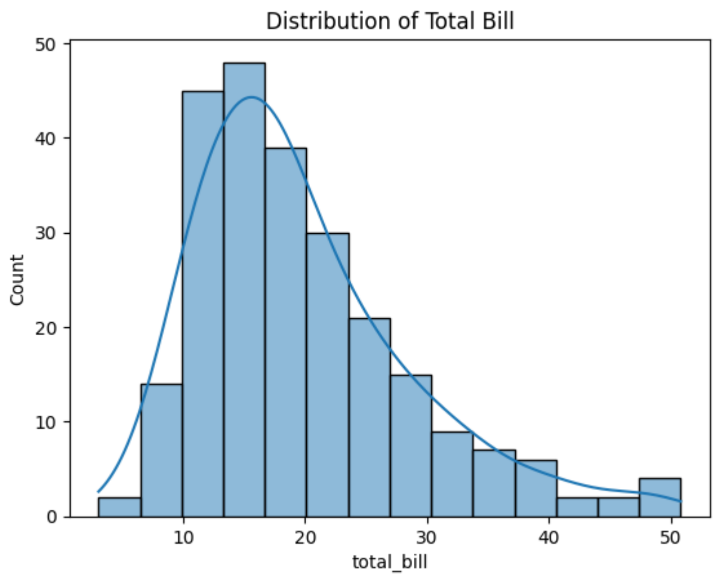


---

6. Histogram / Distribution Plot

- Used to show the distribution of a numeric variable.

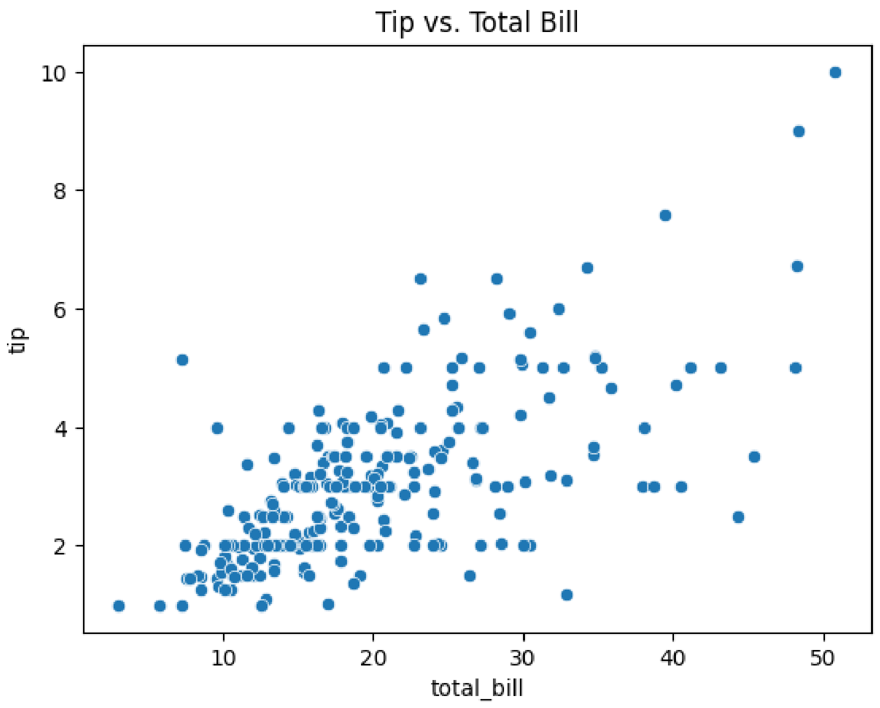
```
sns.histplot(df["total_bill"], kde=True)
plt.title("Distribution of Total Bill")
plt.show()
```



7. Scatter Plot

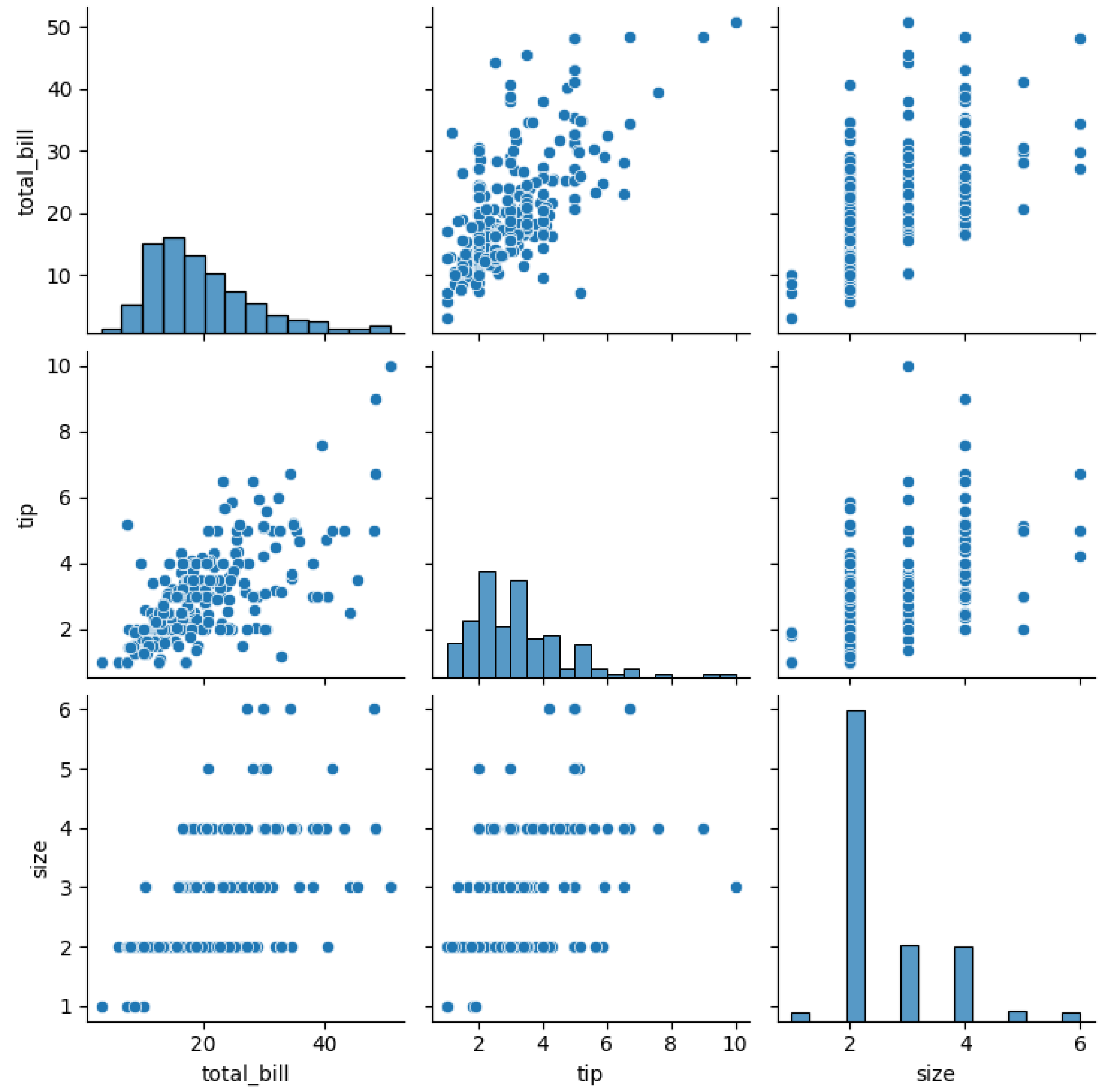
- Shows the relationship between two numeric variables.

```
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.



   - Plot a joint distribution of total_bill and tip.

3. Pairplot

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

4. Boxplot

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

5. Violinplot

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

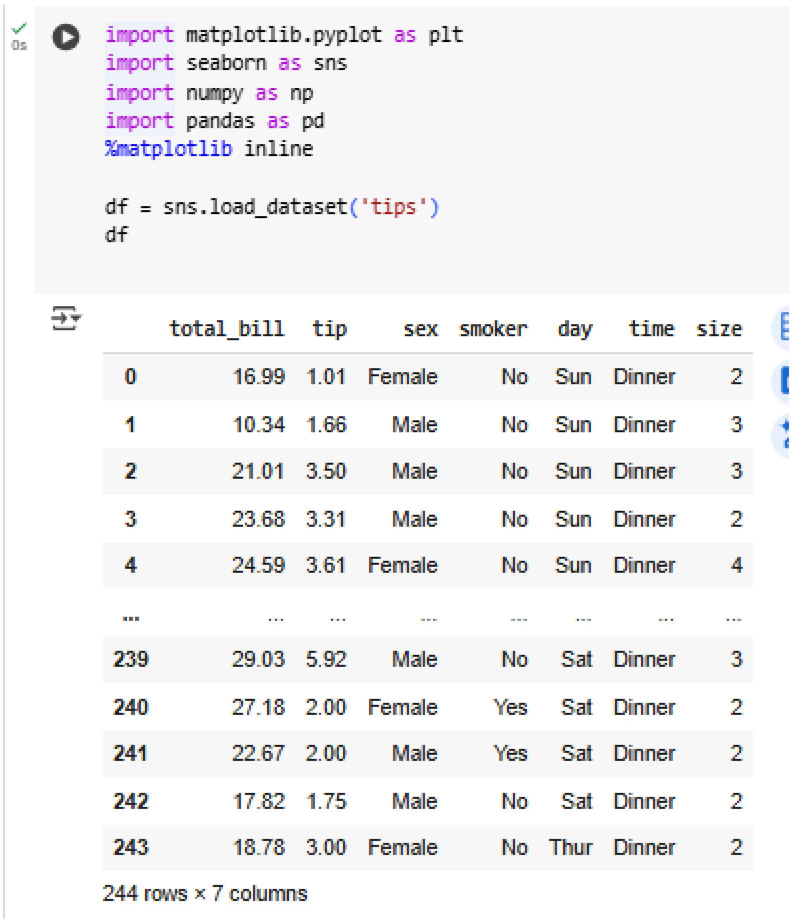
6. Countplot
    - Create a countplot showing the number of observations for each day.
7. Bar Plot
    - Use sns.barplot() to show average tip for each day.

LAB:06 - 07

Exploratory Data Visualization using Matplotlib and Seaborn

Dataset:

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



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

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

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

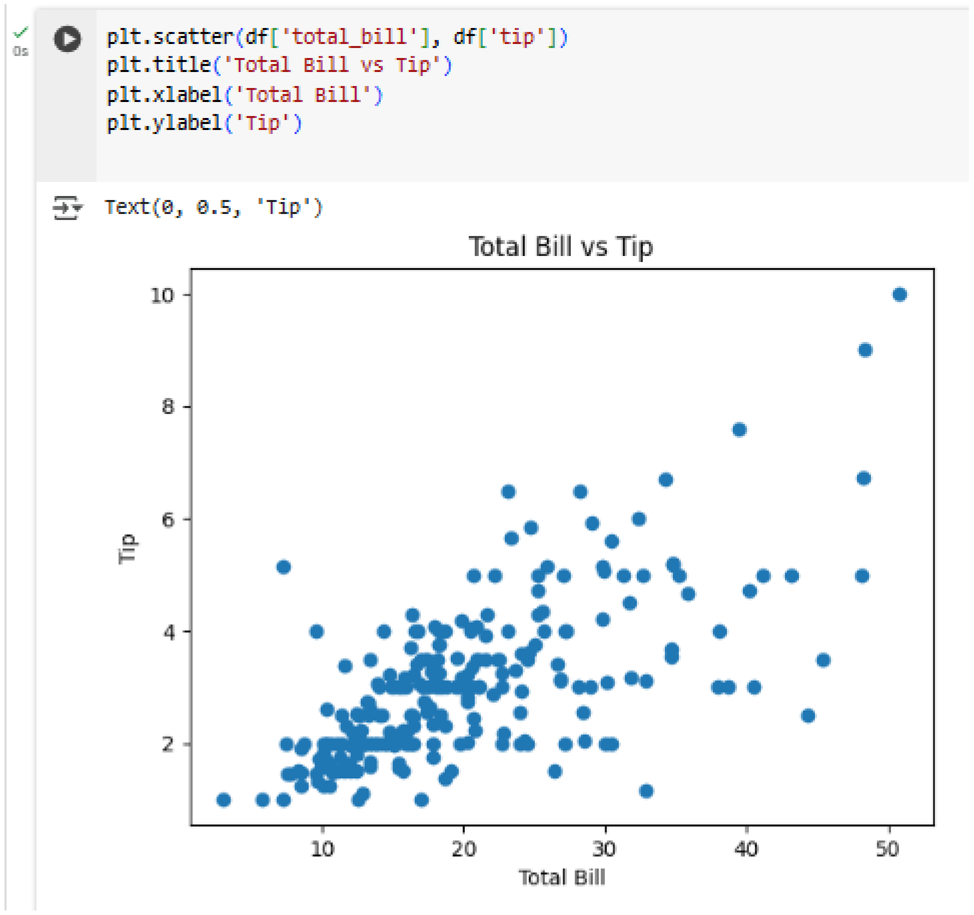
244 rows × 7 columns

Instructions:

Part A: Visualizations using Matplotlib

1. Scatter Plot:

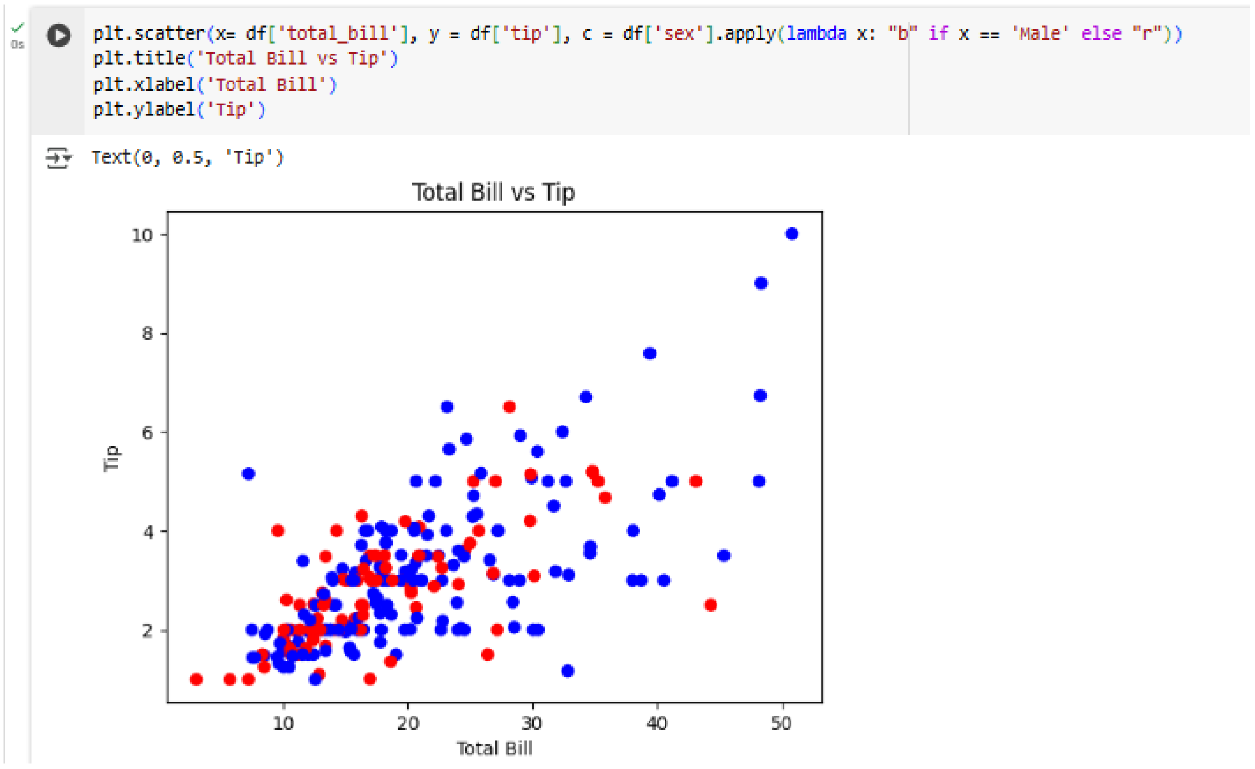
Plot total_bill vs tip with appropriate axis labels and title.



```python
plt.scatter(df['total_bill'], df['tip'])
plt.title('Total Bill vs Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
```

```
Text(0, 0.5, 'Tip')
```

Add color to points based on sex.



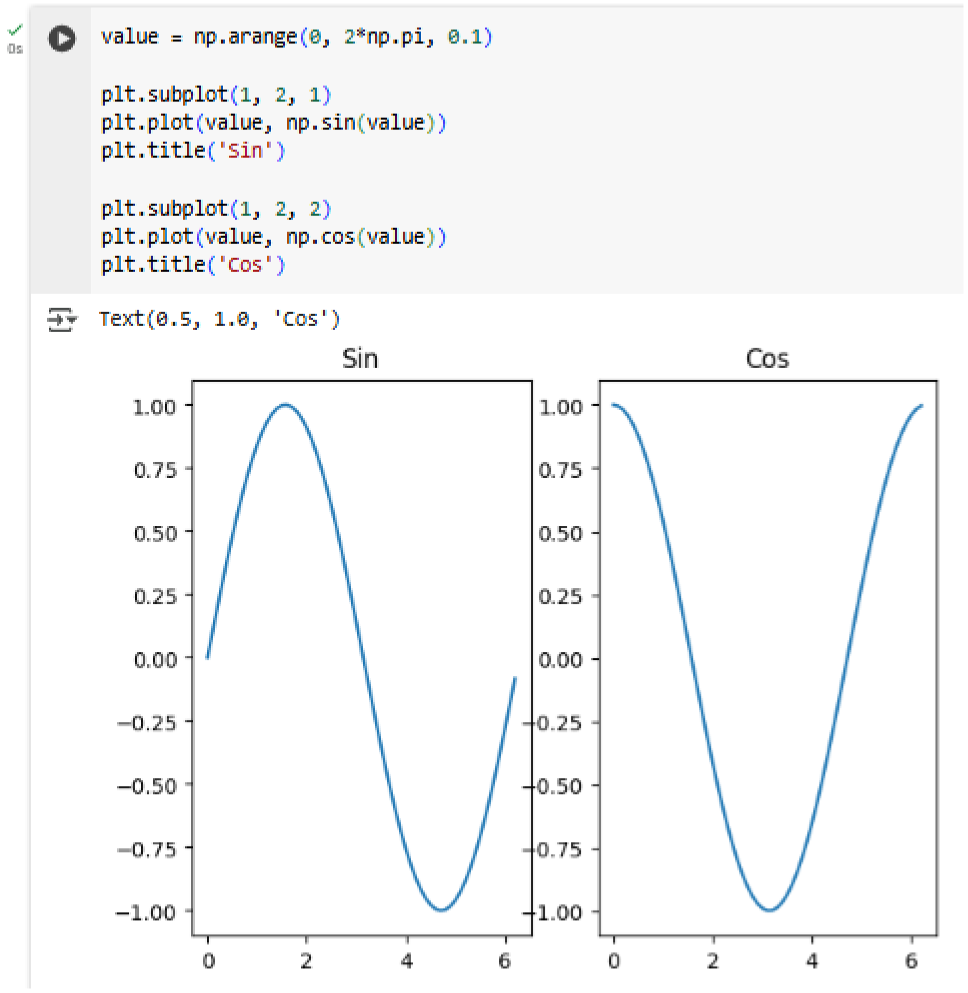
```
plt.scatter(x= df['total_bill'], y = df['tip'], c = df['sex'].apply(lambda x: "b" if x == 'Male' else "r"))
plt.title('Total Bill vs Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
```

```
Text(0, 0.5, 'Tip')
```

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.

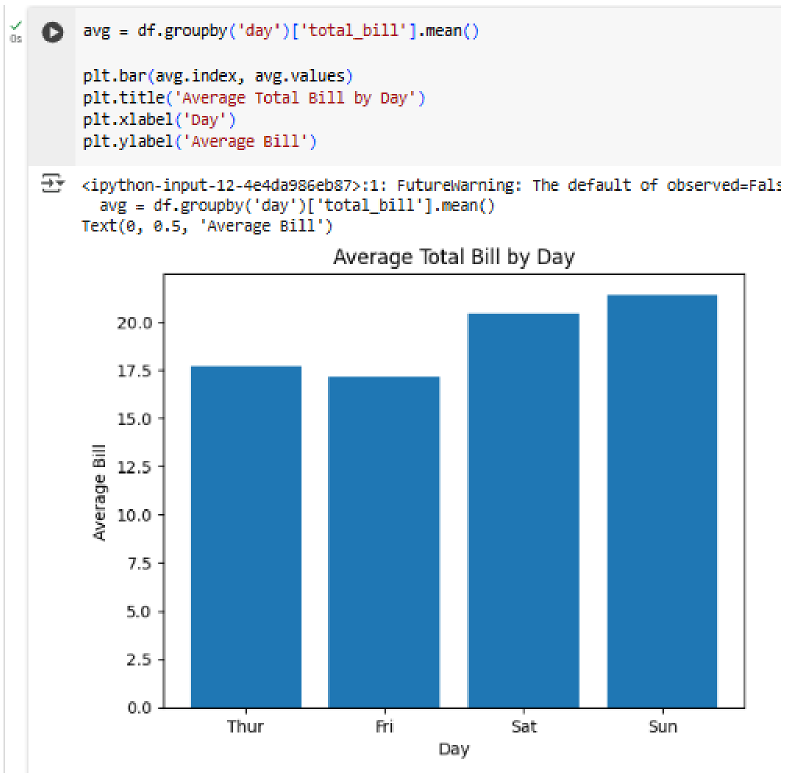


```
value = np.arange(0, 2*np.pi, 0.1)

plt.subplot(1, 2, 1)
plt.plot(value, np.sin(value))
plt.title('Sin')

plt.subplot(1, 2, 2)
plt.plot(value, np.cos(value))
plt.title('Cos')
```

```
Text(0.5, 1.0, 'Cos')
```

3. Bar Plot:

Plot average total_bill for each day using a bar plot.

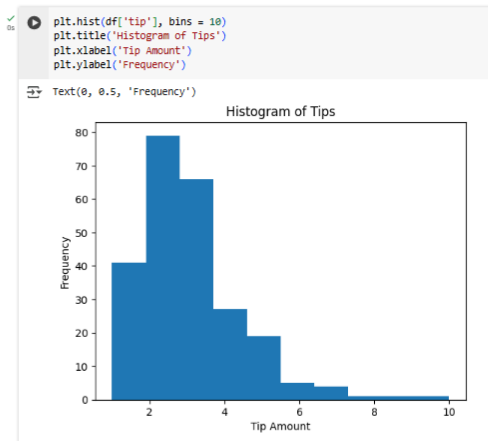


```
avg = df.groupby('day')['total_bill'].mean()

plt.bar(avg.index, avg.values)
plt.title('Average Total Bill by Day')
plt.xlabel('Day')
plt.ylabel('Average Bill')
```

```
<ipython-input-12-4e4da986eb87>:1: FutureWarning: The default of observed=Fals
  avg = df.groupby('day')['total_bill'].mean()
Text(0, 0.5, 'Average Bill')
```

4. Histogram :

Create a histogram of tip values with bins=10 and appropriate labels.

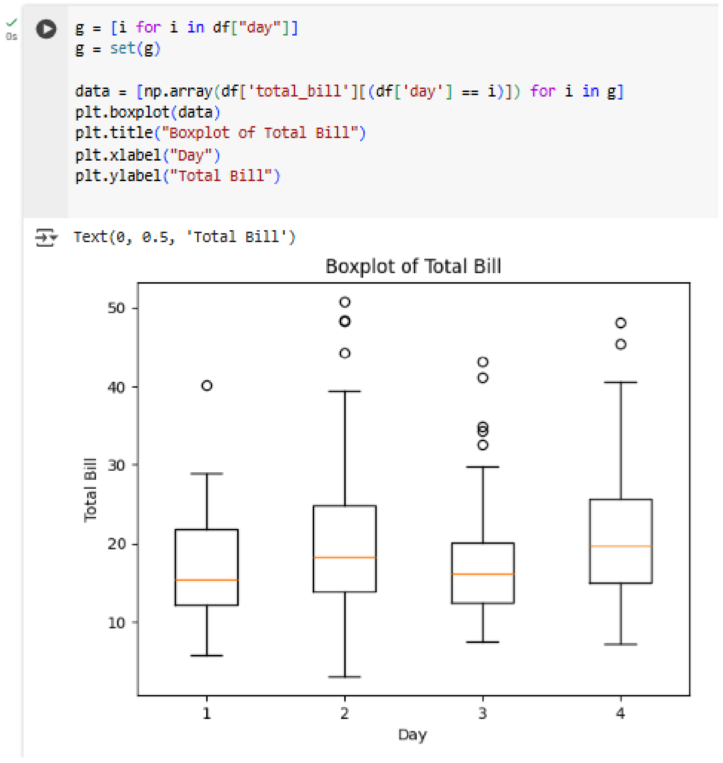


```
plt.hist(df['tip'], bins = 10)
plt.title('Histogram of Tips')
plt.xlabel('Tip Amount')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```

5. Boxplot:

Create a boxplot of total_bill grouped by day.

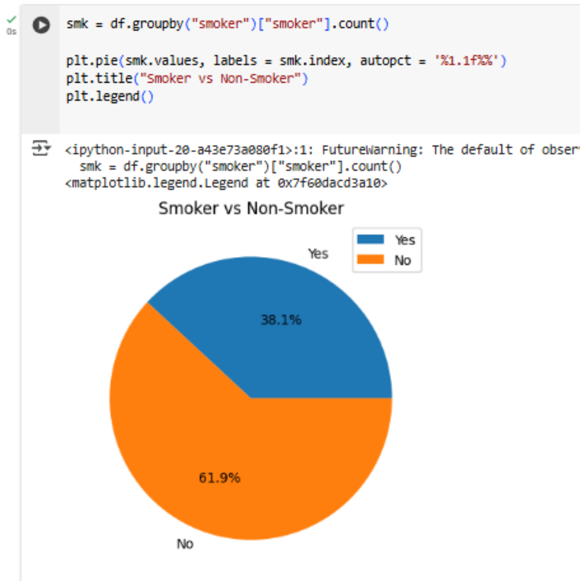


```
g = [i for i in df["day"]]
g = set(g)

data = [np.array(df['total_bill'][(df['day'] == i)]) for i in g]
plt.boxplot(data)
plt.title("Boxplot of Total Bill")
plt.xlabel("Day")
plt.ylabel("Total Bill")
```

```
Text(0, 0.5, 'Total Bill')
```

6. Pie Chart:

Show pie chart of smoker vs non-smoker counts.



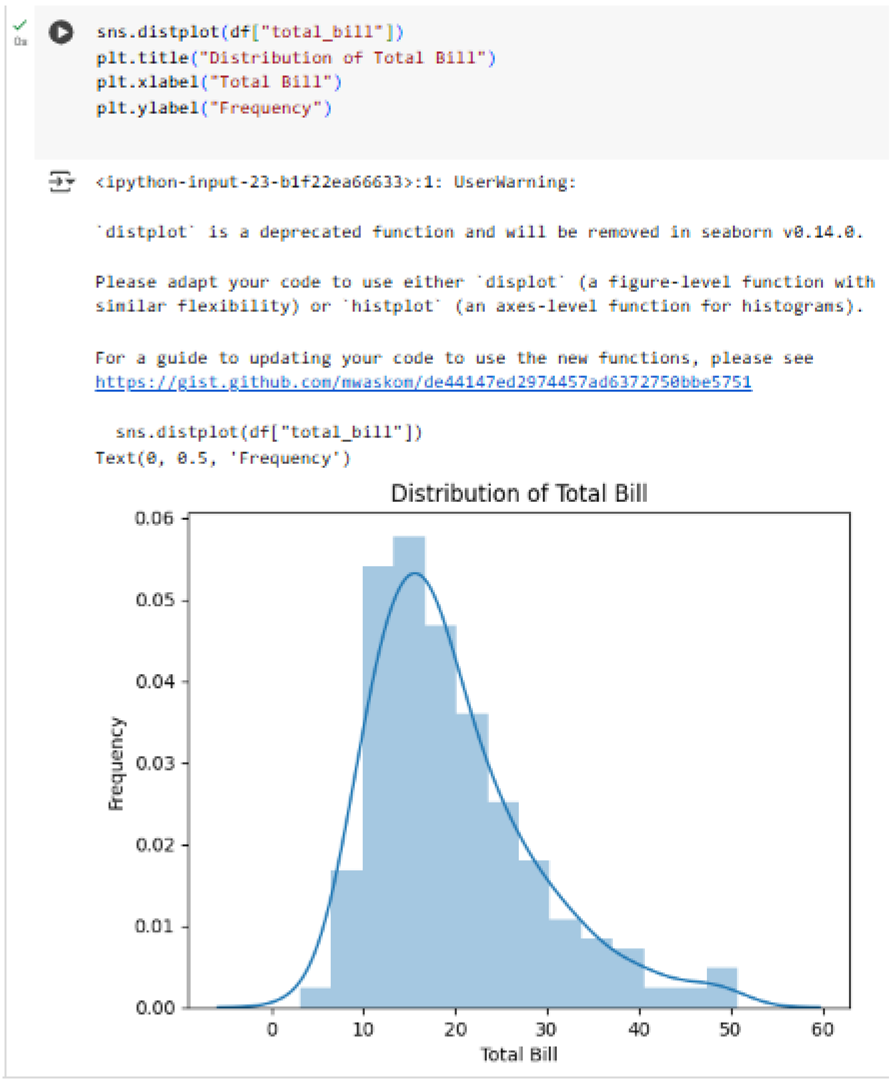
```
smk = df.groupby("smoker")["smoker"].count()

plt.pie(smk.values, labels = smk.index, autopct = '%1.1f%%')
plt.title("Smoker vs Non-Smoker")
plt.legend()
```

```
<ipython-input-20-a43e73a080f1>:1: FutureWarning: The default of obser
  smk = df.groupby("smoker")["smoker"].count()
<matplotlib.legend.Legend at 0x7f60dacd3a10>
```

Part B: Visualizations using Seaborn

1. Distplot:

Create a distribution plot of total_bill.

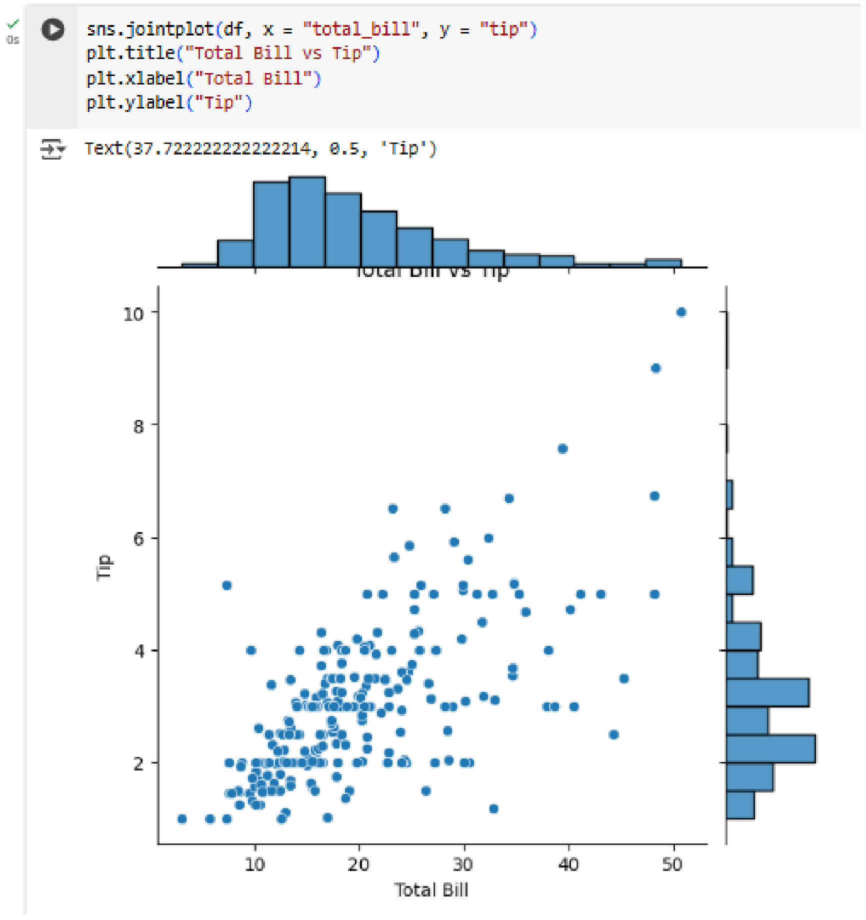


```
sns.distplot(df["total_bill"])
plt.title("Distribution of Total Bill")
plt.xlabel("Total Bill")
plt.ylabel("Frequency")
```

```
<ipython-input-23-b1f22ea66633>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["total_bill"])
Text(0, 0.5, 'Frequency')
```

2. Jointplot:

Plot a joint distribution of total_bill and tip.

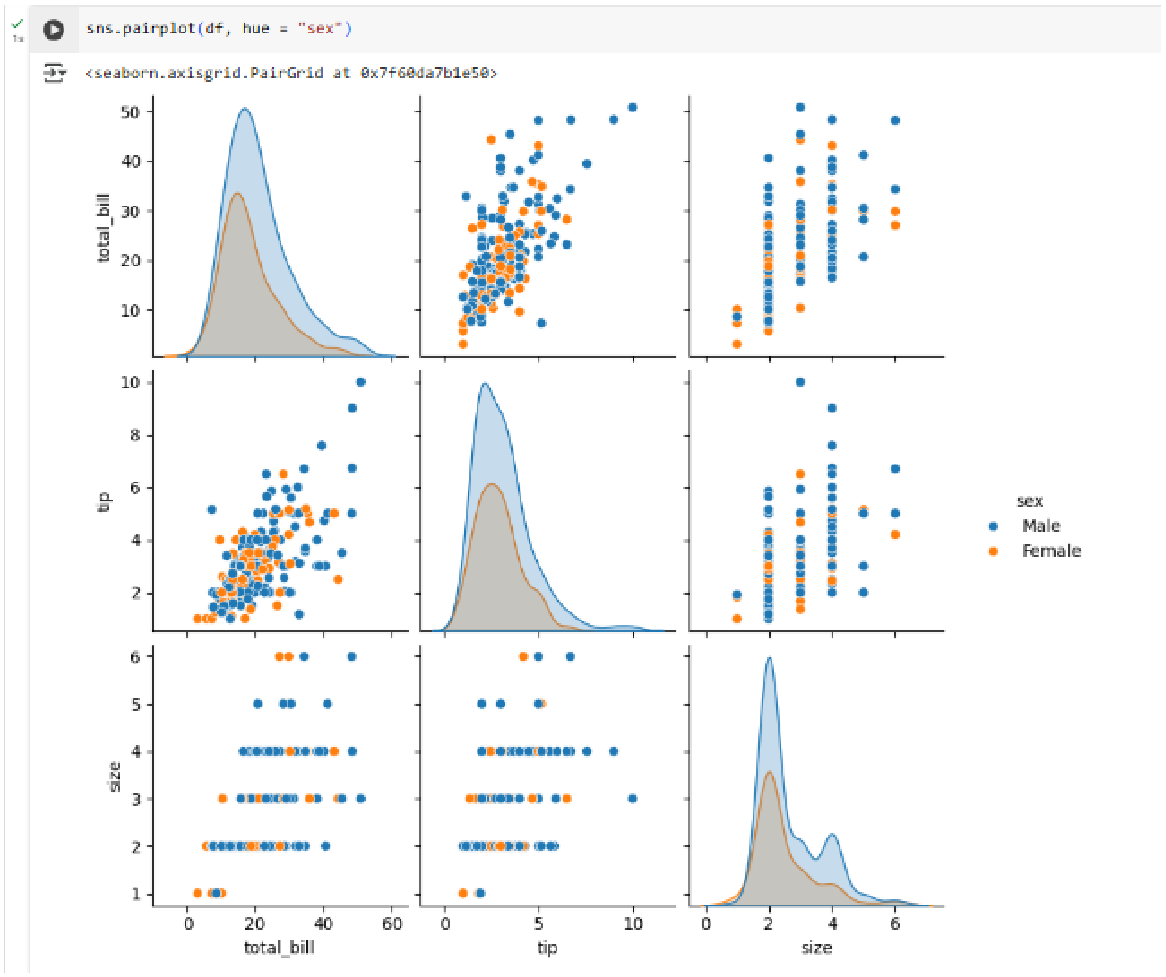


```
sns.jointplot(df, x = "total_bill", y = "tip")
plt.title("Total Bill vs Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
```

```
Text(37.722222222222214, 0.5, 'Tip')
```

3. Pairplot:

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

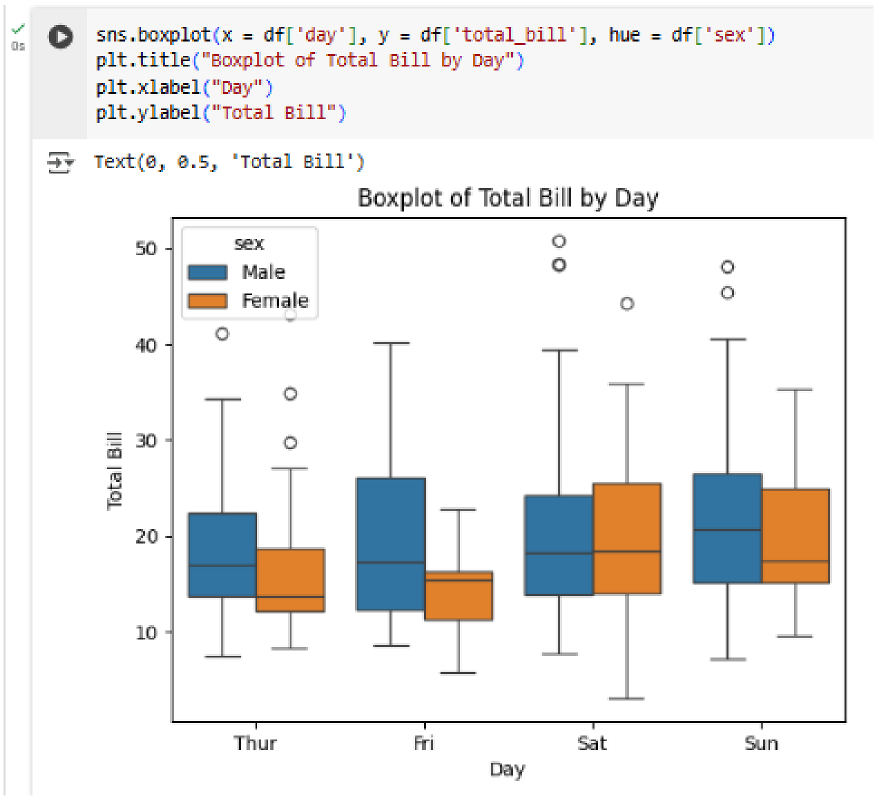


```
sns.pairplot(df, hue = "sex")
```

```
<seaborn.axisgrid.PairGrid at 0x7f60da7b1e50>
```

4. Boxplot:

   Create a boxplot showing total_bill for each day and further grouped by sex.

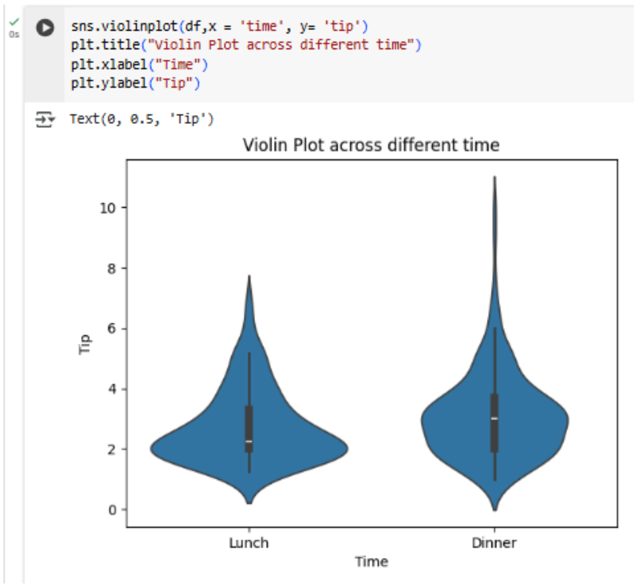


```
sns.boxplot(x = df['day'], y = df['total_bill'], hue = df['sex'])
plt.title("Boxplot of Total Bill by Day")
plt.xlabel("Day")
plt.ylabel("Total Bill")
```

```
Text(0, 0.5, 'Total Bill')
```

5. Violinplot:

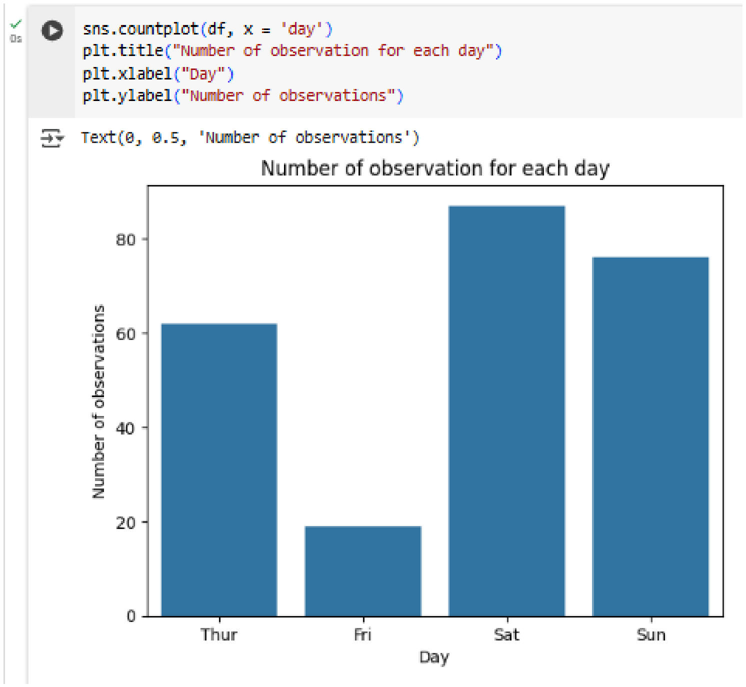
   Create a violin plot comparing tip across different times (Lunch, Dinner).



```
sns.violinplot(df,x = 'time', y= 'tip')
plt.title("Violin Plot across different time")
plt.xlabel("Time")
plt.ylabel("Tip")
```

```
Text(0, 0.5, 'Tip')
```

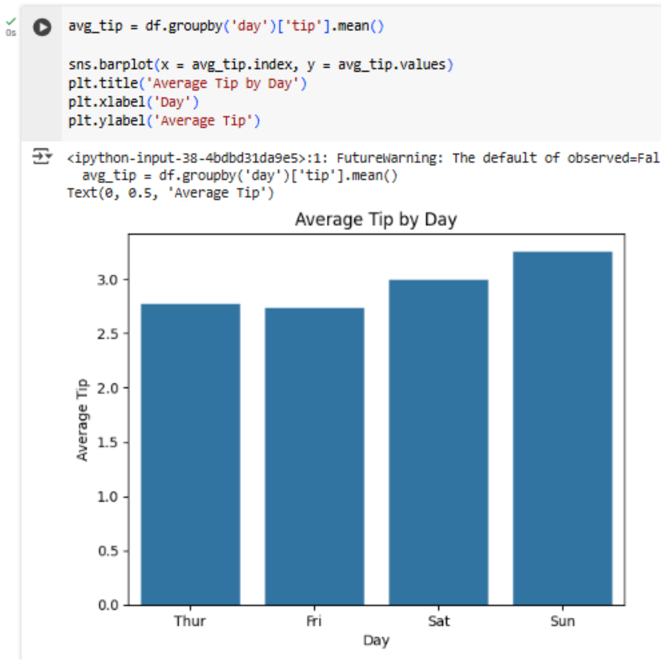
6. Countplot:
   Create a countplot showing the number of observations for each day.



```
sns.countplot(df, x = 'day')
plt.title("Number of observation for each day")
plt.xlabel("Day")
plt.ylabel("Number of observations")
```

```
Text(0, 0.5, 'Number of observations')
```

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



```
avg_tip = df.groupby('day')['tip'].mean()

sns.barplot(x = avg_tip.index, y = avg_tip.values)
plt.title('Average Tip by Day')
plt.xlabel('Day')
plt.ylabel('Average Tip')
```

```
<ipython-input-38-4bdbd31da9e5>:1: FutureWarning: The default of observed=Fal
  avg_tip = df.groupby('day')['tip'].mean()
Text(0, 0.5, 'Average Tip')
```

○

Lab No: 8

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).
2. Calculate:
   - Mean, median, mode
   - Variance and standard deviation
   - Skewness and kurtosis
3. Display summary using df.describe()

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

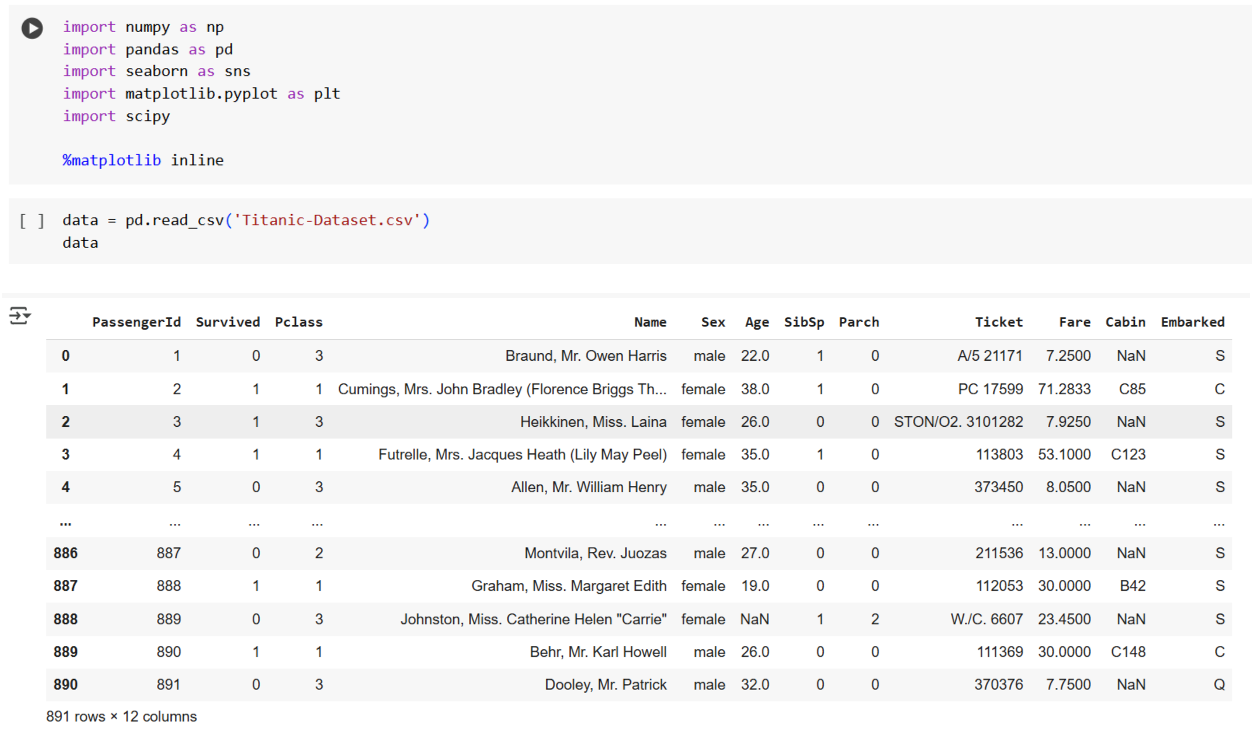
Part B – Inferential Statistics in Python
1. Compute correlation matrix using df.corr().
2. Conduct a t-test using scipy.stats.ttest_ind() to compare two sample means.
3. Interpret p-values and confidence intervals.

Part C – Descriptive and Inferential Stats in R
1. Load the dataset using read.csv() or datasets::iris
2. Compute:
   - mean(), median(), sd(), var()
   - Use summary() and cor()
3. Run a t-test using t.test()
4. Create boxplots and histograms with ggplot2

LAB:08

Part A – Descriptive Statistics in Python

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



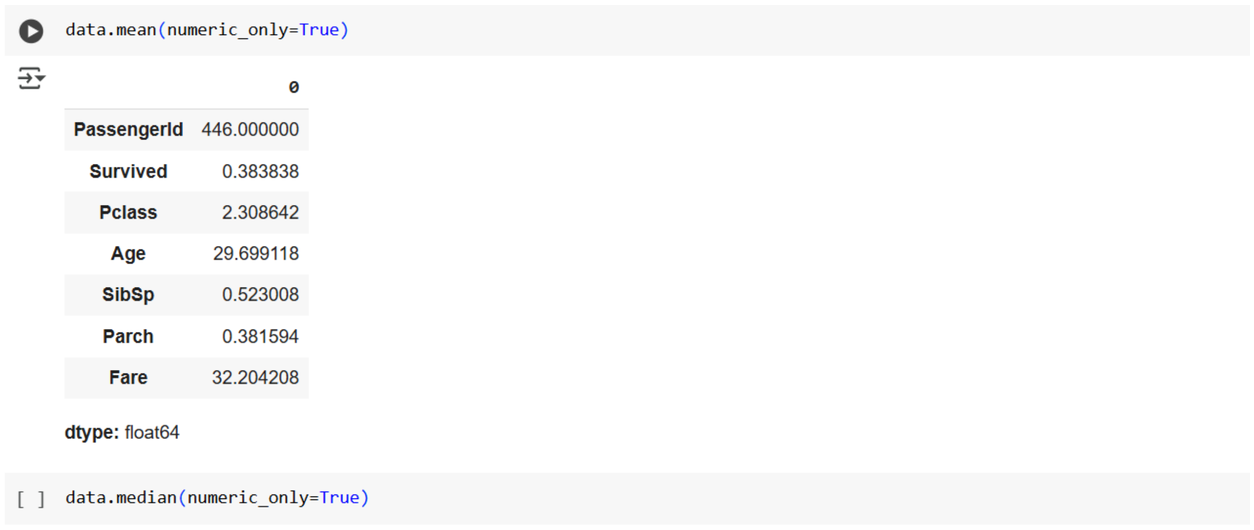
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy

%matplotlib inline
```

```python
data = pd.read_csv('Titanic-Dataset.csv')
data
```

```
     PassengerId  Survived  Pclass                                                Name     Sex   Age  SibSp  Parch            Ticket     Fare  Cabin  Embarked
0              1         0       3                             Braund, Mr. Owen Harris    male  22.0      1      0         A/5 21171   7.2500    NaN         S
1              2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1      0          PC 17599  71.2833    C85         C
2              3         1       3                              Heikkinen, Miss. Laina  female  26.0      0      0  STON/O2. 3101282   7.9250    NaN         S
3              4         1       1        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1      0            113803  53.1000   C123         S
4              5         0       3                            Allen, Mr. William Henry    male  35.0      0      0            373450   8.0500    NaN         S
...          ...       ...     ...                                                 ...     ...   ...    ...    ...               ...      ...    ...       ...
886          887         0       2                               Montvila, Rev. Juozas    male  27.0      0      0            211536  13.0000    NaN         S
887          888         1       1                        Graham, Miss. Margaret Edith  female  19.0      0      0            112053  30.0000    B42         S
888          889         0       3            Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1      2        W./C. 6607  23.4500    NaN         S
889          890         1       1                               Behr, Mr. Karl Howell    male  26.0      0      0            111369  30.0000   C148         C
890          891         0       3                                 Dooley, Mr. Patrick    male  32.0      0      0            370376   7.7500    NaN         Q
891 rows × 12 columns
```

2. Calculate:
- Mean, median, mode
- Variance and standard deviation
- Skewness and kurtosis

Mean:

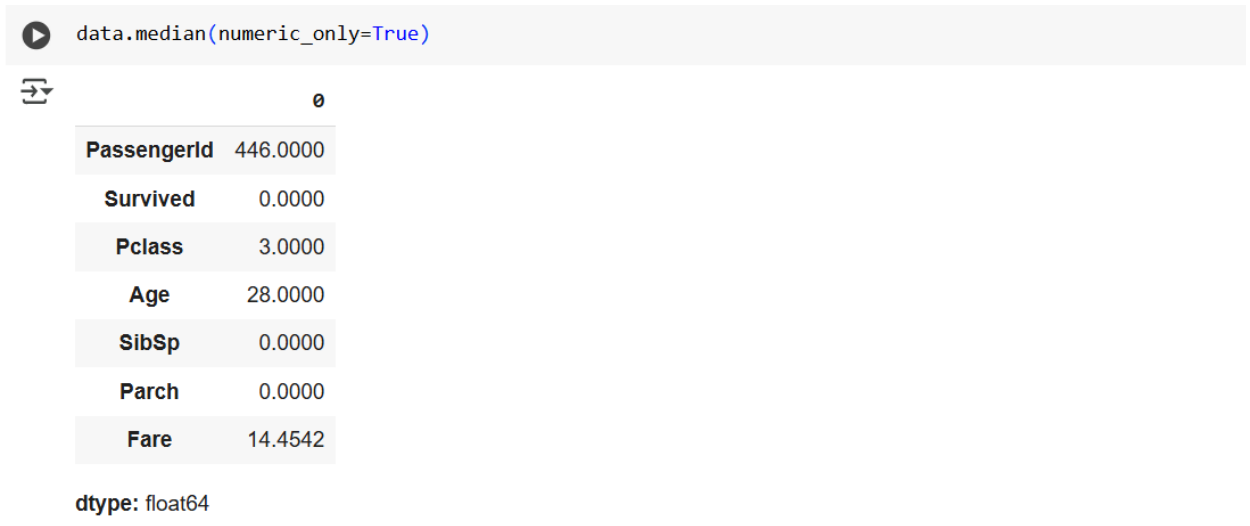


```
data.mean(numeric_only=True)
```

```
                     0
PassengerId  446.000000
   Survived    0.383838
     Pclass    2.308642
        Age   29.699118
      SibSp    0.523008
      Parch    0.381594
       Fare   32.204208

dtype: float64
```

```
[ ]  data.median(numeric_only=True)
```

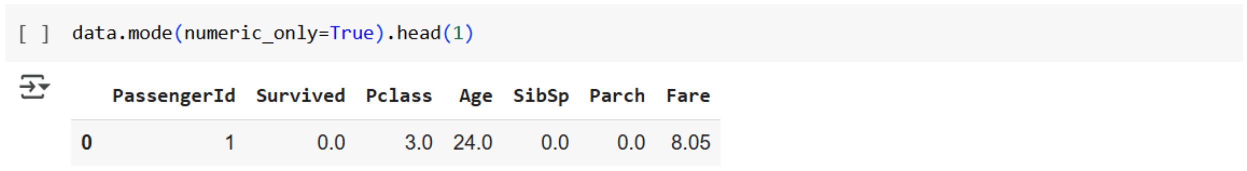
Median:



```
data.median(numeric_only=True)
```

```
                   0
PassengerId  446.0000
   Survived    0.0000
     Pclass    3.0000
        Age   28.0000
      SibSp    0.0000
      Parch    0.0000
       Fare   14.4542

dtype: float64
```

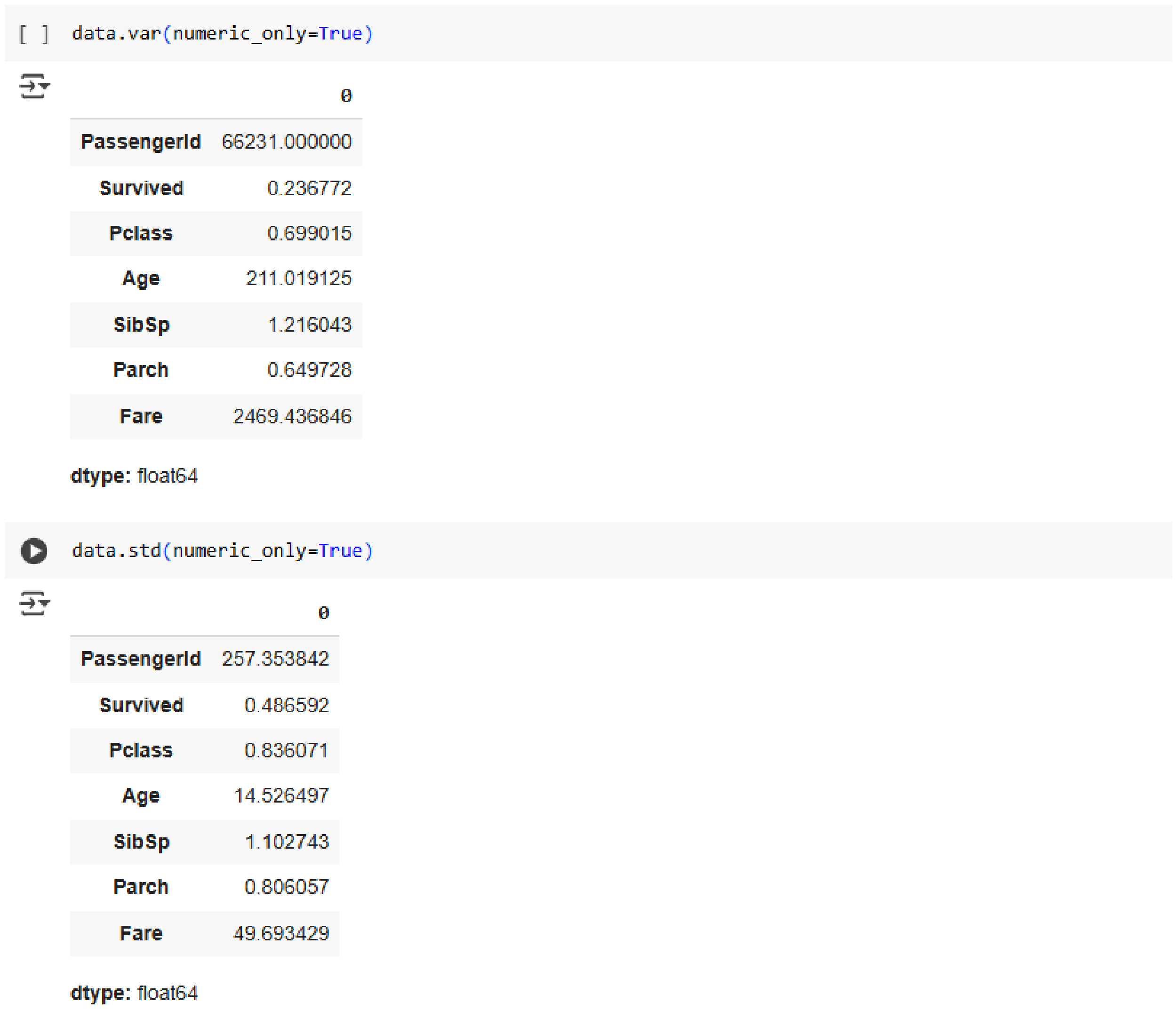
Mode:



```
[ ]  data.mode(numeric_only=True).head(1)
```

```
   PassengerId  Survived  Pclass   Age  SibSp  Parch  Fare
0            1       0.0     3.0  24.0    0.0    0.0  8.05
```

Variance and Standard Deviation:



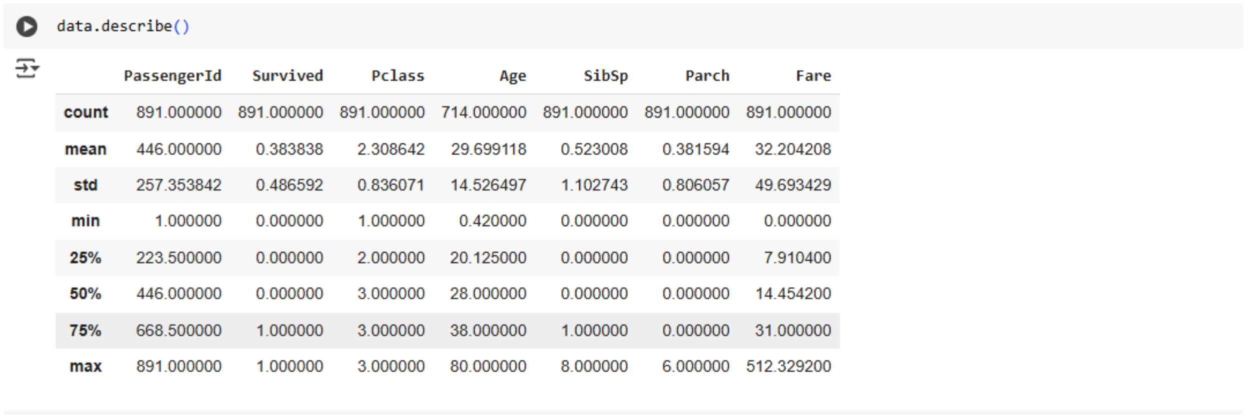
```
[ ]  data.var(numeric_only=True)
```

```
                        0
PassengerId  66231.000000
   Survived      0.236772
     Pclass      0.699015
        Age    211.019125
      SibSp      1.216043
      Parch      0.649728
       Fare   2469.436846

dtype: float64
```

```
data.std(numeric_only=True)
```

```
                      0
PassengerId  257.353842
   Survived    0.486592
     Pclass    0.836071
        Age   14.526497
      SibSp    1.102743
      Parch    0.806057
       Fare   49.693429

dtype: float64
```

3. Display summary using df.describe()



```
data.describe()
```

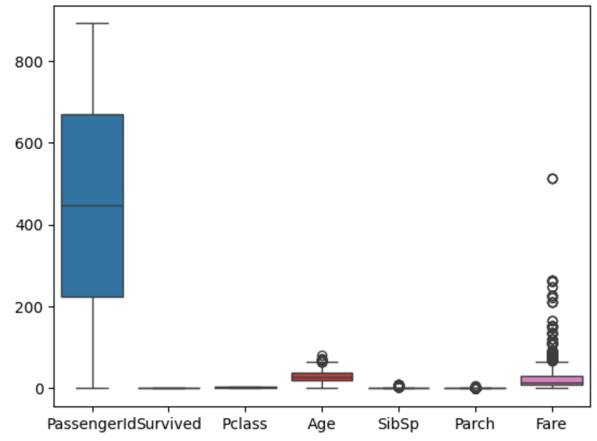
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

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

Plot distributions using seaborn.boxplot():

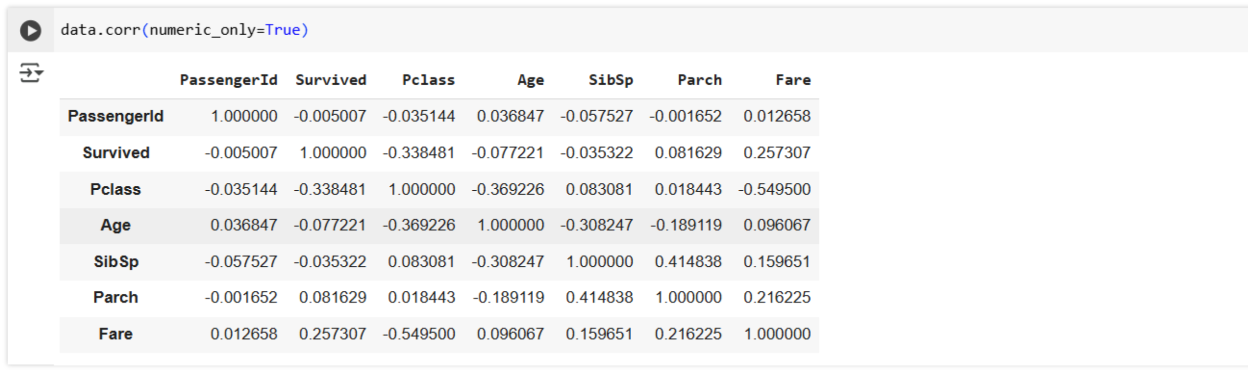
```
sns.boxplot(data)
```

```
<Axes: >
```



Part B – Inferential Statistics in Python

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

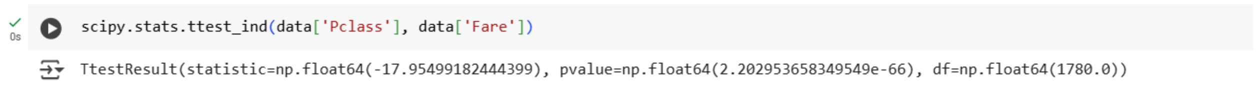


```
data.corr(numeric_only=True)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

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

T-Test of columns data['Pclass'] and data['Fare'] :

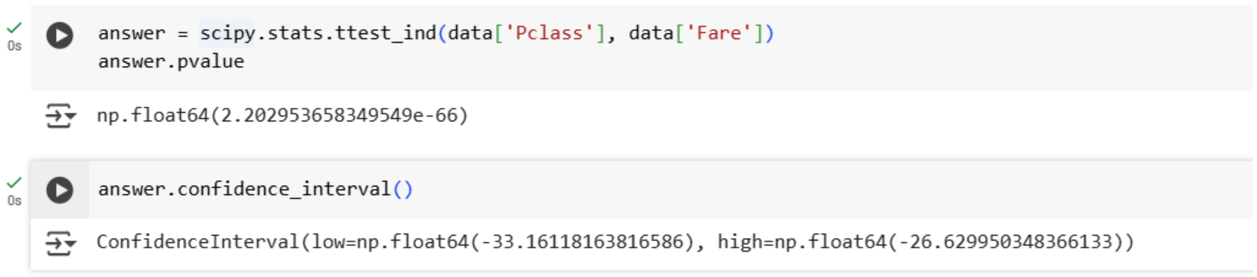


```
scipy.stats.ttest_ind(data['Pclass'], data['Fare'])
```

```
TtestResult(statistic=np.float64(-17.95499182444399), pvalue=np.float64(2.202953658349549e-66), df=np.float64(1780.0))
```

7. Interpret p-values and confidence intervals.

P-value and Confidence Interval of columns data['Pclass'] and data['Fare'] :



```
answer = scipy.stats.ttest_ind(data['Pclass'], data['Fare'])
answer.pvalue
```

```
np.float64(2.202953658349549e-66)
```

```
answer.confidence_interval()
```

```
ConfidenceInterval(low=np.float64(-33.16118163816586), high=np.float64(-26.629950348366133))
```

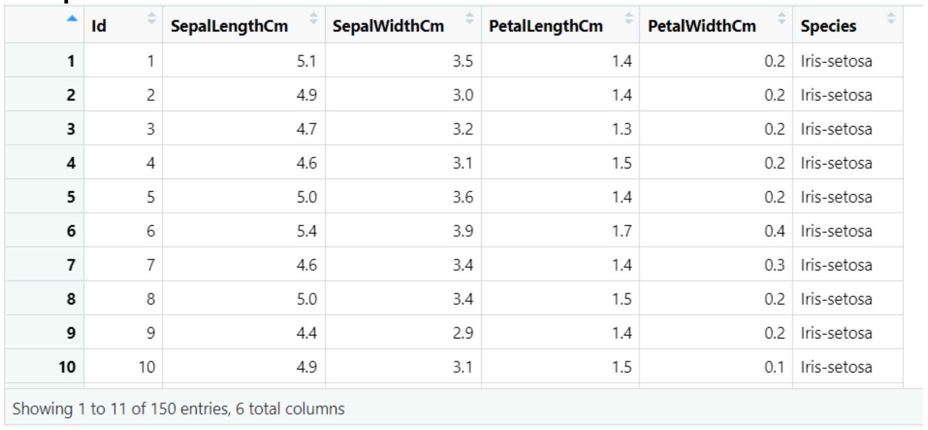
Part C – Descriptive and Inferential Stats in R

1. Load the dataset using read.csv() or datasets::iris

Source Code:

```
library(tidyverse)
library(readr)
library(ggplot2)

data_iris <- read_csv("Iris.csv")
view(data_iris)
```

Output:



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	5.1	3.5	1.4	0.2	Iris-setosa
2	2	4.9	3.0	1.4	0.2	Iris-setosa
3	3	4.7	3.2	1.3	0.2	Iris-setosa
4	4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	5.0	3.6	1.4	0.2	Iris-setosa
6	6	5.4	3.9	1.7	0.4	Iris-setosa
7	7	4.6	3.4	1.4	0.3	Iris-setosa
8	8	5.0	3.4	1.5	0.2	Iris-setosa
9	9	4.4	2.9	1.4	0.2	Iris-setosa
10	10	4.9	3.1	1.5	0.1	Iris-setosa

Showing 1 to 11 of 150 entries, 6 total columns

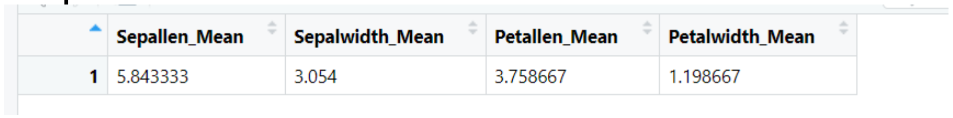
2. Compute:
   - mean(), median(), sd(), var()
   - Use summary() and cor()

Mean:

Source Code:

```
mean_data <- data_iris %>% select(SepalLengthCm, SepalWidthCm, PetalLengthCm,
                     PetalWidthCm) %>% summarise(
                      Sepallen_Mean = mean(SepalLengthCm),
                      Sepalwidth_Mean = mean(SepalWidthCm),
                      Petallen_Mean = mean(PetalLengthCm),
                      Petalwidth_Mean = mean(PetalWidthCm),
                     )
view(mean_data)
```

Output:



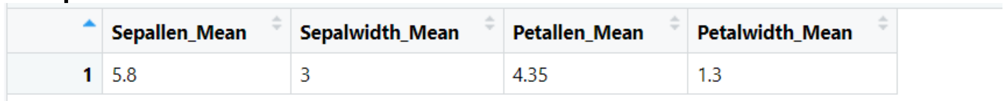
	Sepallen_Mean	Sepalwidth_Mean	Petallen_Mean	Petalwidth_Mean
1	5.843333	3.054	3.758667	1.198667

Median:

Source Code:

```
median_data <- data_iris %>% select(SepalLengthCm, SepalWidthCm, PetalLengthCm,
                    PetalWidthCm) %>% summarise(
                     Sepallen_Mean = median(SepalLengthCm),
                     Sepalwidth_Mean = median(SepalWidthCm),
                     Petallen_Mean = median(PetalLengthCm),
                     Petalwidth_Mean = median(PetalWidthCm),
                    )
view(median_data)
```

Output:



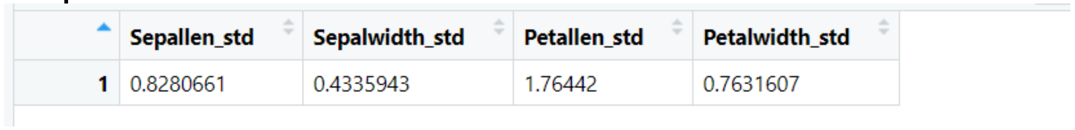
	Sepallen_Mean	Sepalwidth_Mean	Petallen_Mean	Petalwidth_Mean
1	5.8	3	4.35	1.3

Standard Deviation:

Source Code:

```
std_data <- data_iris %>% mutate(
                     Sepallen_std = sd(SepalLengthCm) ,
                     Sepalwidth_std = sd(SepalWidthCm),
                     Petallen_std = sd(PetalLengthCm),
                     Petalwidth_std = sd(PetalWidthCm)
                    ) %>% select(
                     Sepallen_std,
                     Sepalwidth_std,
                     Petallen_std,
                     Petalwidth_std
                     ) %>% head(1)

view(std_data)
```

Output:



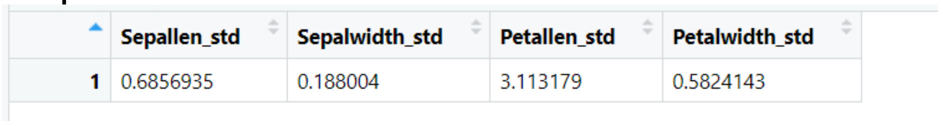
	Sepallen_std	Sepalwidth_std	Petallen_std	Petalwidth_std
1	0.8280661	0.4335943	1.76442	0.7631607

Variance:

Source Code:

```
var_data <- data_iris %>% mutate(
                    Sepallen_std = var(SepalLengthCm) ,
                    Sepalwidth_std = var(SepalWidthCm),
                    Petallen_std = var(PetalLengthCm),
                    Petalwidth_std = var(PetalWidthCm)
                  ) %>% select(
                    Sepallen_std,
                    Sepalwidth_std,
                    Petallen_std,
                    Petalwidth_std
                    ) %>% head(1)

view(var_data)
```

Output:



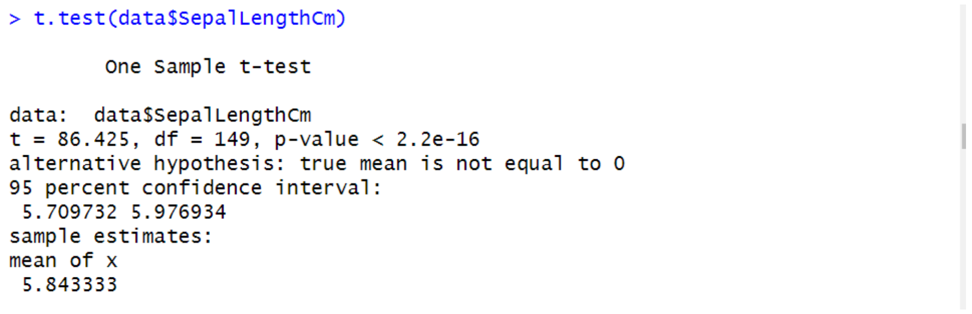
	Sepallen_std	Sepalwidth_std	Petallen_std	Petalwidth_std
1	0.6856935	0.188004	3.113179	0.5824143

3. Run a t-test using t.test()

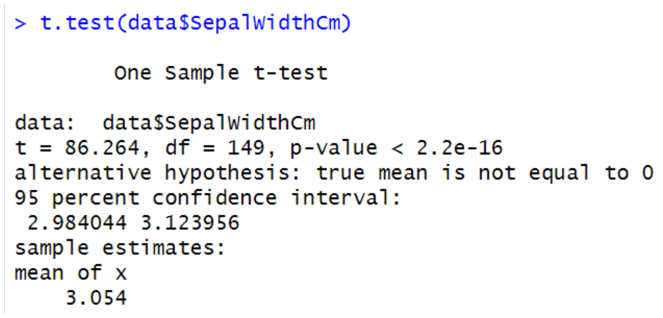
Source Code:

```
t.test(data_iris$SepalLengthCm)
t.test(data_iris$SepalWidthCm)
t.test(data_iris$PetalLengthCm)
t.test(data_iris$PetalWidthCm)
```

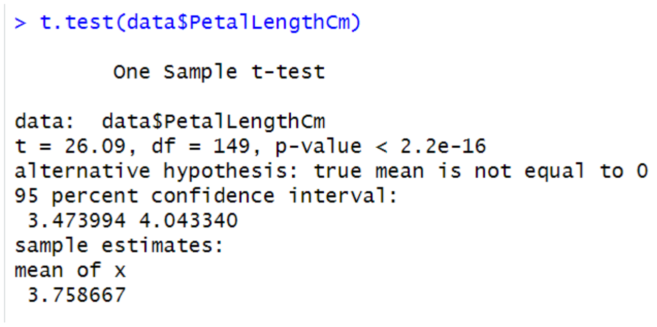
Output:



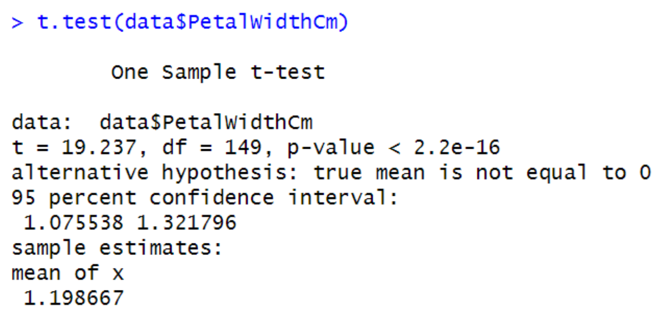
```
> t.test(data$SepalLengthCm)

        One Sample t-test

data:  data$SepalLengthCm
t = 86.425, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 5.709732 5.976934
sample estimates:
mean of x 
 5.843333 
```



```
> t.test(data$SepalWidthCm)

        One Sample t-test

data:  data$SepalWidthCm
t = 86.264, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 2.984044 3.123956
sample estimates:
mean of x
    3.054
```



```
> t.test(data$PetalLengthCm)

        One Sample t-test

data:  data$PetalLengthCm
t = 26.09, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 3.473994 4.043340
sample estimates:
mean of x
 3.758667
```



```
> t.test(data$PetalWidthCm)

        One Sample t-test

data:  data$PetalWidthCm
t = 19.237, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1.075538 1.321796
sample estimates:
mean of x
 1.198667
```

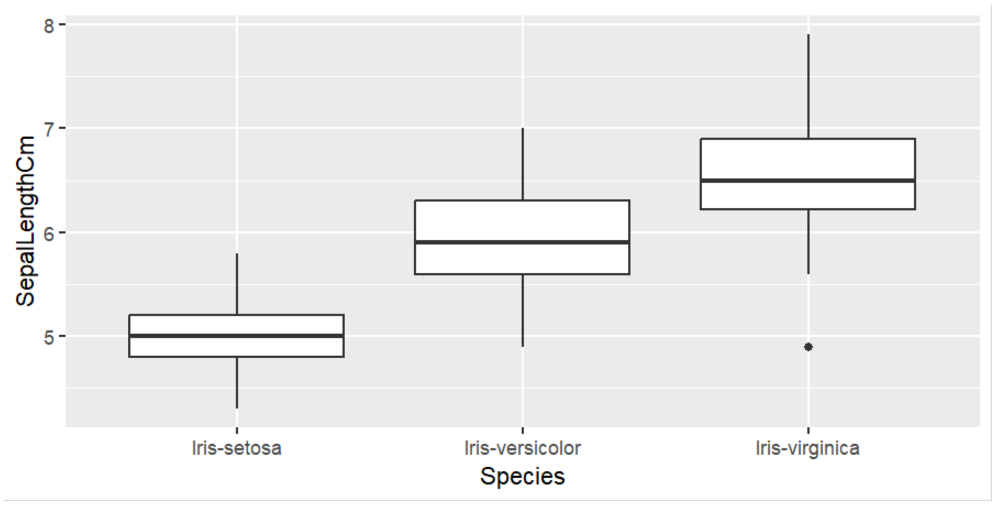
4. Create boxplots and histograms with ggplot2

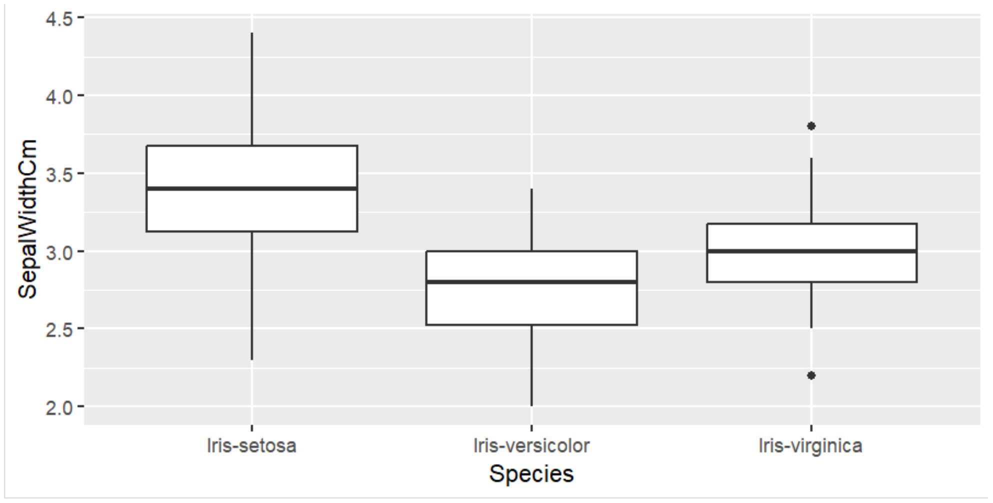
BoxPlot:

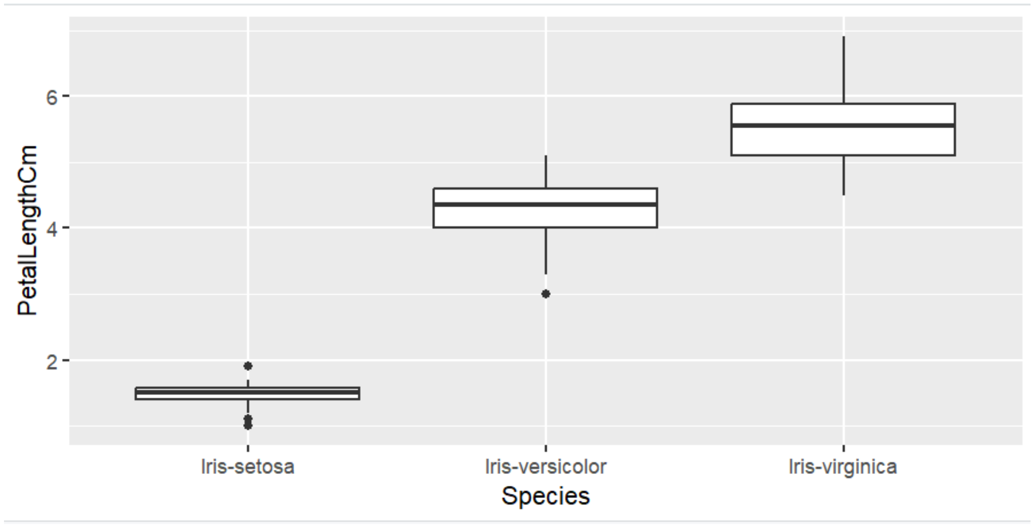
Source Code:

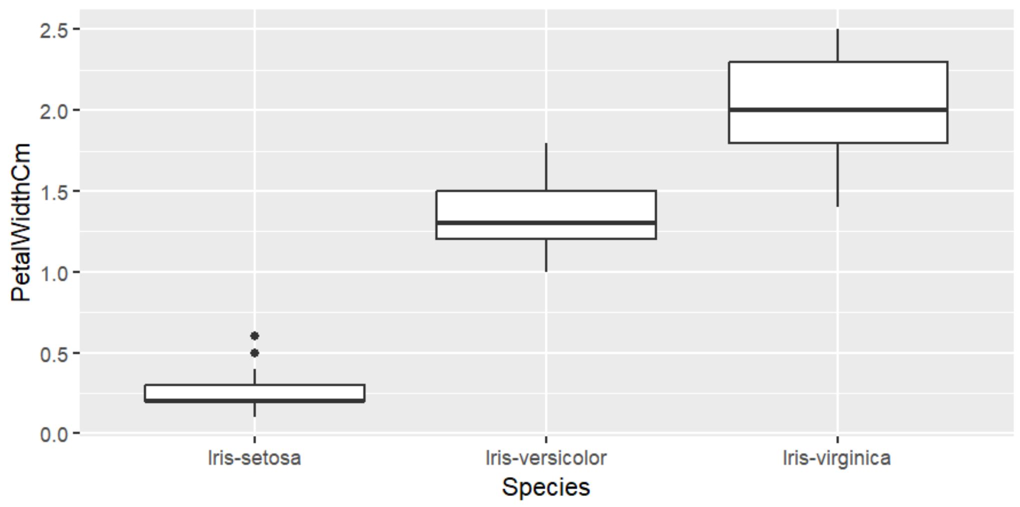
```
box_1 <- ggplot(data_iris, aes(x=Species, y=SepalLengthCm)) + geom_boxplot()
box_1
box_2 <- ggplot(data_iris, aes(x=Species, y=SepalWidthCm)) + geom_boxplot()
box_2
box_3 <- ggplot(data_iris, aes(x=Species, y=PetalLengthCm)) + geom_boxplot()
box_3
box_4 <- ggplot(data_iris, aes(x=Species, y=PetalWidthCm)) + geom_boxplot()
box_4
```

Output:


8
7
6
5
SepalLengthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Species


4.5
4.0
3.5
3.0
2.5
2.0
SepalWidthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Species


6
4
2
PetalLengthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Species

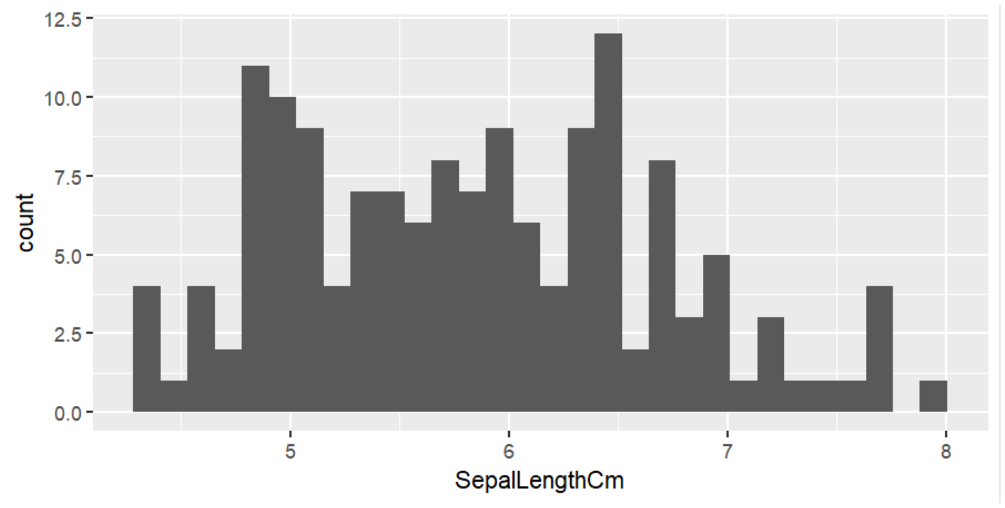


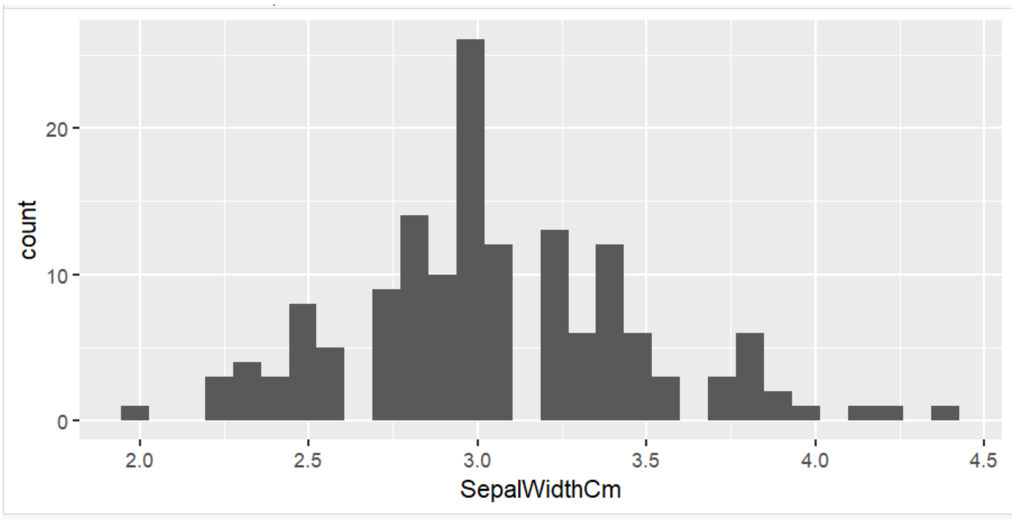
Histogram:

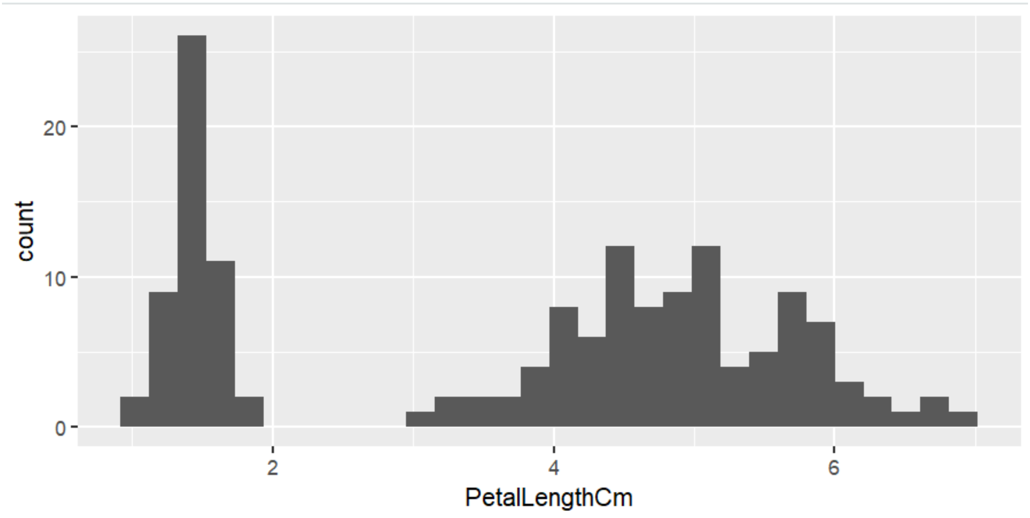
Source Code:

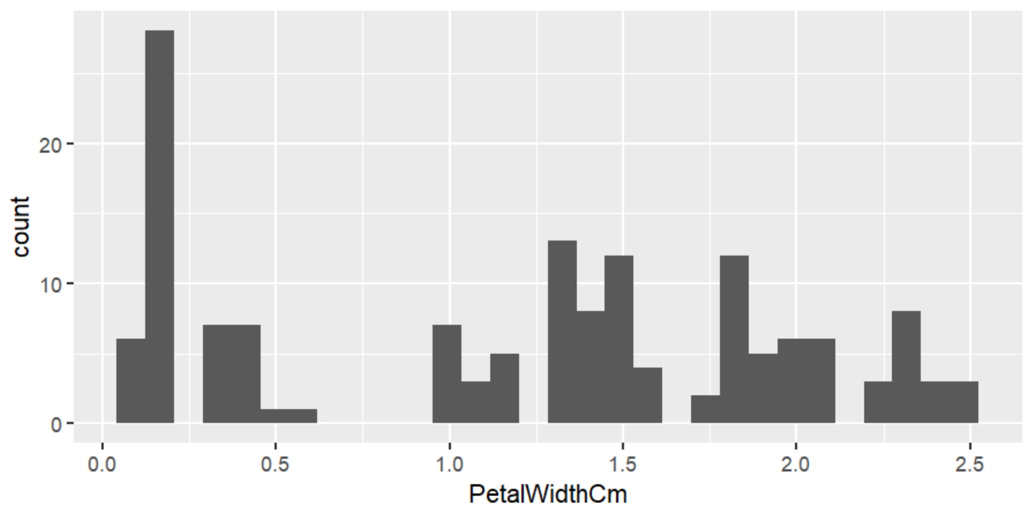
```
hist_1 <- ggplot(data_iris, aes(x=SepalLengthCm)) + geom_histogram()
hist_1
hist_2 <- ggplot(data_iris, aes(x=SepalWidthCm)) + geom_histogram()
hist_2
hist_3 <- ggplot(data_iris, aes(x=PetalLengthCm)) + geom_histogram()
hist_3
hist_4 <- ggplot(data_iris, aes(x=PetalWidthCm)) + geom_histogram()
hist_4
```

Output:


12.5
10.0
7.5
5.0
2.5
0.0
count
5
6
7
8
SepalLengthCm


20
10
0
count
2.0
2.5
3.0
3.5
4.0
4.5
SepalWidthCm


20
10
0
count
2
4
6
PetalLengthCm


count
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
PetalWidthCm

Lab No: 9

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.

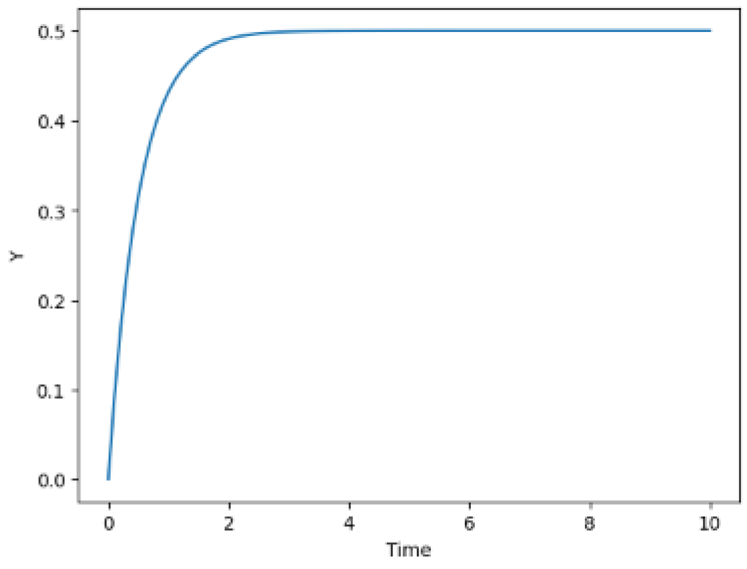
Tasks:

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.

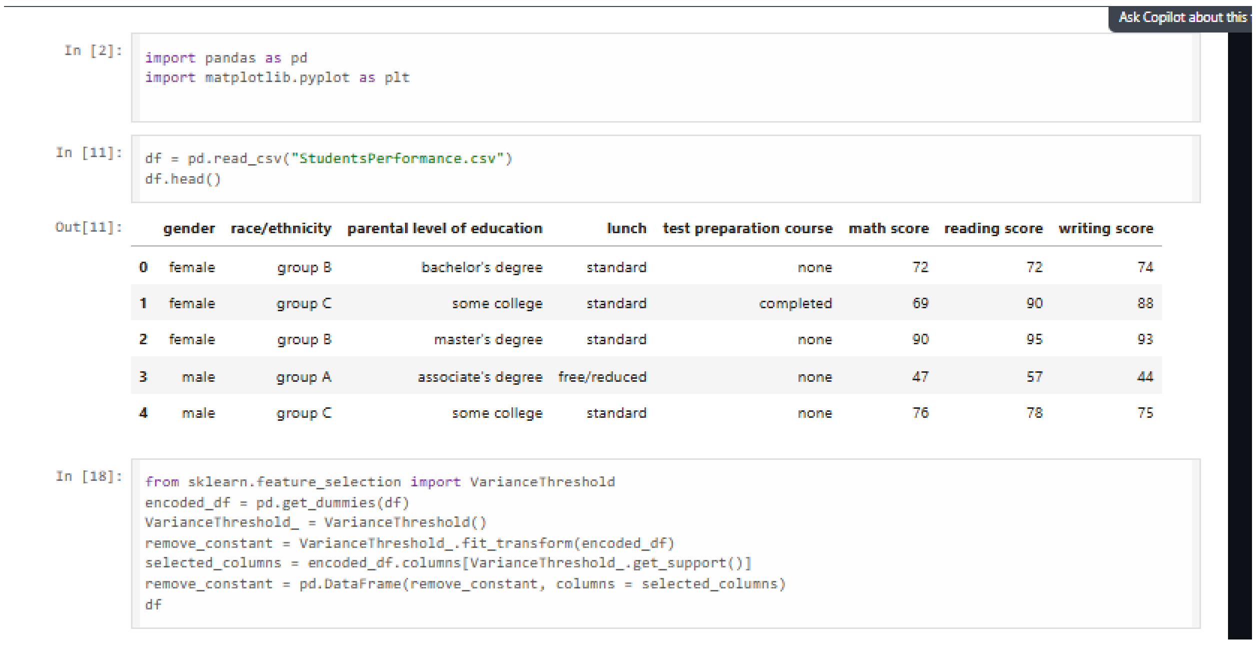
LAB:09

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

```python
def returns_dydt(y,t):
    dydt = (-2 * y) + 1
    return dydt

y0 = 0
t = np.linspace(0,10,100)
y = odeint(returns_dydt,y0,t)

plt.plot(t,y)
plt.xlabel("Time")
plt.ylabel("Y")
plt.show()
```



OEL 2:



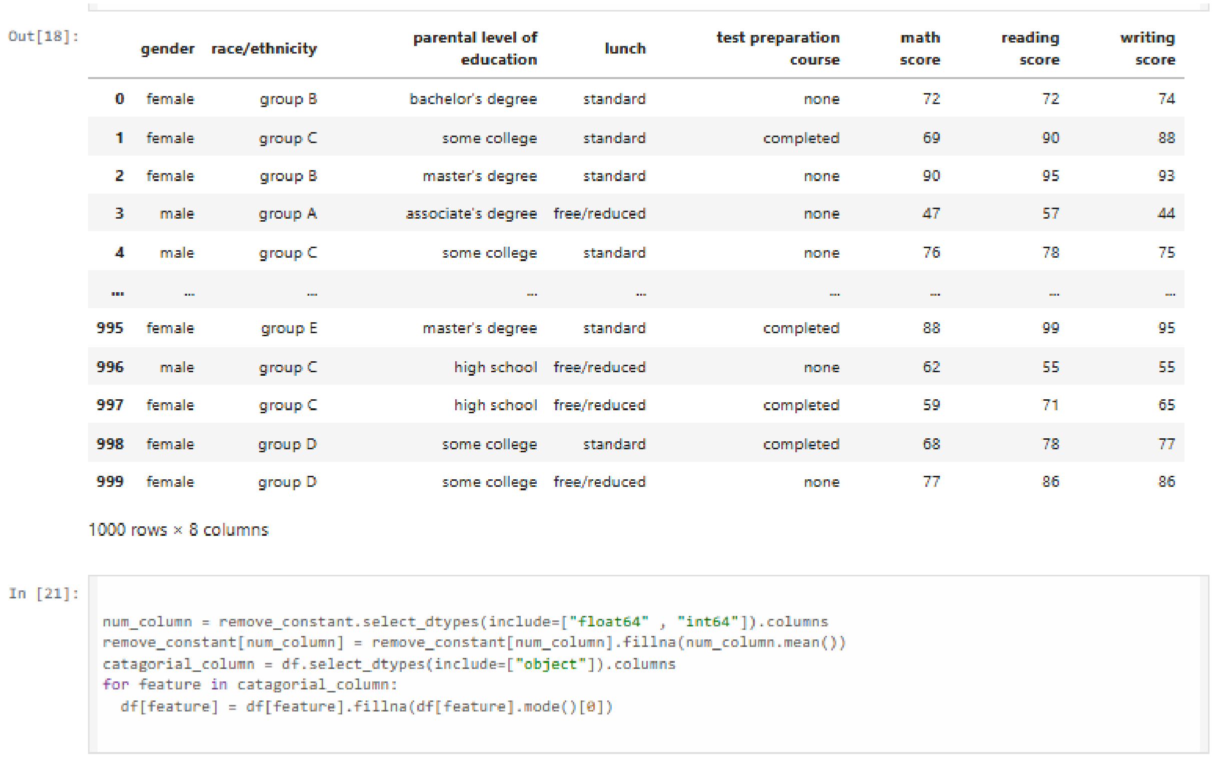
In [2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

In [11]:
```python
df = pd.read_csv("StudentsPerformance.csv")
df.head()
```

Out[11]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

In [18]:
```python
from sklearn.feature_selection import VarianceThreshold
encoded_df = pd.get_dummies(df)
VarianceThreshold_ = VarianceThreshold()
remove_constant = VarianceThreshold_.fit_transform(encoded_df)
selected_columns = encoded_df.columns[VarianceThreshold_.get_support()]
remove_constant = pd.DataFrame(remove_constant, columns = selected_columns)
df
```



Out[18]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

1000 rows × 8 columns

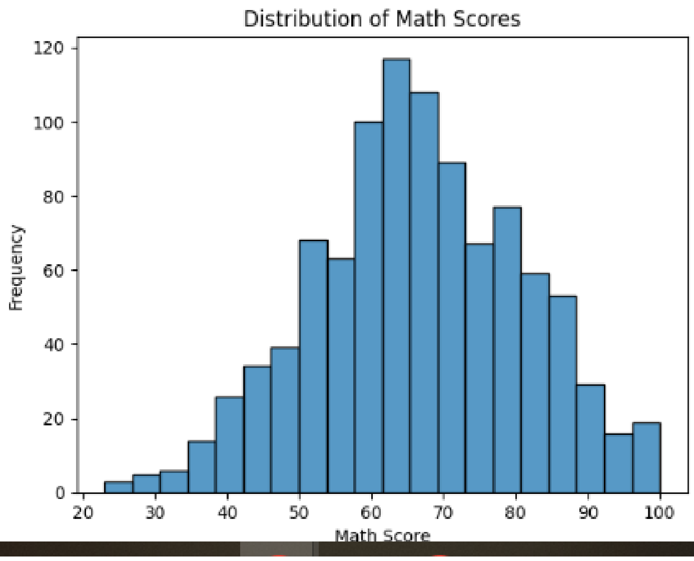
In [21]:
```python
num_column = remove_constant.select_dtypes(include=["float64" , "int64"]).columns
remove_constant[num_column] = remove_constant[num_column].fillna(num_column.mean())
catagorial_column = df.select_dtypes(include=["object"]).columns
for feature in catagorial_column:
  df[feature] = df[feature].fillna(df[feature].mode()[0])
```

In [24]:

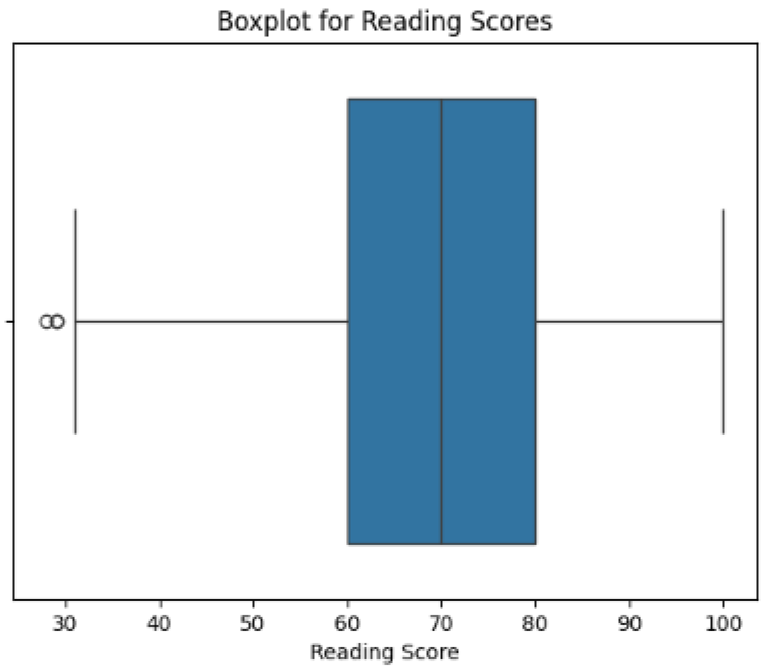
```
from scipy.stats import zscore
for column in ['math score', 'reading score']:
    z_scores = zscore(df[column])
    df = df[(abs(z_scores) < 3)]
```

In [25]:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.histplot(df['math score'], bins=20)
plt.title("Distribution of Math Scores")
plt.xlabel("Math Score")
plt.ylabel("Frequency")
plt.show()
```



In [26]:
```
sns.boxplot(x=df['reading score'])
plt.title("Boxplot for Reading Scores")
plt.xlabel("Reading Score")
plt.show()
```



In [27]:
```
plt.figure(figsize=(8, 5))
sns.countplot(x='gender', df)
plt.title('Countplot of Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```