INDEX

EX.NO: 01
Date:16/07/2025

IMPLEMENT A PYTHON PROGRAM TO PROCESS TEXT AND LIST DATA

AIM:

To write a Python program that processes text (uppercase, lowercase, word count, reverse) and list data (maximum, minimum, sum, sort) based on user choice.

ALGORITHM:

```
1. Start
2. Repeat until user chooses Exit
    1. Display main menu:

            1 → Process Text

            2 → Process List

            3 → Exit

    2. Read user choice
    3. If choice = 1 (Text):

            Show text menu (uppercase, lowercase, word count, reverse)

            Perform selected operation

    4. Else if choice = 2 (List):

            Show list menu (maximum, minimum, sum, sort)

            Perform selected operation

    5. Else if choice = 3:

            Display exit message and stop loop

    6. Else: Print "Invalid choice"

3. End
```

PROGRAM:

```
def process_text(text):
    print("\nText Processing Menu:")
    print("1. Convert to UPPERCASE")
    print("2. Convert to lowercase")
    print("3. Count number of words")
    print("4. Reverse the text")

    choice = input("Choose an option (1-4): ")

    if choice == '1':
        print("Uppercase Text:", text.upper())
    elif choice == '2':
        print("Lowercase Text:", text.lower())
    elif choice == '3':
        words = text.split()
        print("Number of Words:", len(words))
    elif choice == '4':
        print("Reversed Text:", text[::-1])
    else:
        print("Invalid choice!")


def process_list(data_list):
    print("\nList Processing Menu:")
    print("1. Find Maximum")
    print("2. Find Minimum")
    print("3. Calculate Sum")
    print("4. Sort List")

    choice = input("Choose an option (1-4): ")

    try:
        numbers = [float(x) for x in data_list]   # convert all to numbers
    except ValueError:
        print("List contains non-numeric data.")
        return

    if choice == '1':
        print("Maximum Value:", max(numbers))
    elif choice == '2':
        print("Minimum Value:", min(numbers))
    elif choice == '3':
        print("Sum of Elements:", sum(numbers))
    elif choice == '4':
        print("Sorted List:", sorted(numbers))
    else:
        print("Invalid choice!")


# Main Program
while True:
    print("\n=== Text and List Data Processor ===")
    print("1. Process Text")
    print("2. Process List")
    print("3. Exit")

    user_choice = input("Enter your choice (1-3): ")

    if user_choice == '1':
        user_text = input("Enter the text: ")
        process_text(user_text)

    elif user_choice == '2':
        user_input = input("Enter list elements separated by spaces: ")
        user_list = user_input.split()
        process_list(user_list)

    elif user_choice == '3':
        print("Exiting the program. Goodbye!")
        break

    else:
        print("Invalid main choice!")
```

OUTPUT:

```
=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 1
Enter The Text: Shaheen

Text Processing Menu:
1. Convert To Uppercase
2. Convert To Lowercase
3. Count Number Of Words
4. Reverse The Text
Choose An Option (1-4): 1
Uppercase Text: Shaheen

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 1
Enter The Text: Shaheen

Text Processing Menu:
1. Convert To Uppercase
2. Convert To Lowercase
3. Count Number Of Words
4. Reverse The Text
Choose An Option (1-4): 2
Lowercase Text: Shaheen

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 1
Enter The Text: Shaheen

Text Processing Menu:
1. Convert To Uppercase
2. Convert To Lowercase
3. Count Number Of Words
4. Reverse The Text
```

```
Choose An Option (1-4): 2
Lowercase Text: Shaheen

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 1
Enter The Text: Shaheen

Text Processing Menu:
1. Convert To Uppercase
2. Convert To Lowercase
3. Count Number Of Words
4. Reverse The Text
Choose An Option (1-4): 3
Number Of Words: 1

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 1
Enter The Text: Shaheen

Text Processing Menu:
1. Convert To Uppercase
2. Convert To Lowercase
3. Count Number Of Words
4. Reverse The Text
Choose An Option (1-4): 4
Reversed Text: Neehahs

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 2
Enter List Elements Separated By Spaces: 1 2 3 4

List Processing Menu:
1. Find Maximum
2. Find Minimum
3. Calculate Sum
```

```
4. Sort List
Choose An Option (1-4): 1
Maximum Value: 4.0

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 2
Enter List Elements Separated By Spaces: 1 2 3 4

List Processing Menu:
1. Find Maximum
2. Find Minimum
3. Calculate Sum
4. Sort List
Choose An Option (1-4): 2
Minimum Value: 1.0

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 2
Enter List Elements Separated By Spaces: 1 2 3 4

List Processing Menu:
1. Find Maximum
2. Find Minimum
3. Calculate Sum
4. Sort List
Choose An Option (1-4): 3
Sum Of Elements: 10.0

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 2
Enter List Elements Separated By Spaces: 1 2 3 4 5

List Processing Menu:
1. Find Maximum
2. Find Minimum
```

```
3. Calculate Sum
4. Sort List
Choose An Option (1-4): 4
Sorted List: [1.0, 2.0, 3.0, 4.0, 5.0]

=== Text And List Data Processor ===
1. Process Text
2. Process List
3. Exit
Enter Your Choice (1-3): 3
Exiting The Program. Goodbye!
```

RESULT:

The program was successfully executed and produced correct outputs for both text and list processing operations.

EX.NO: 02
Date: 23/07/2025

READ AND ANALYZE CSV & JSON DATA

AIM:

To implement a Python program in Google Colab using Jupyter Notebook that demonstrates reading and processing CSV and JSON data defined within the notebook itself.

ALGORITHM:

1. Start the program.
2. Define sample CSV data as a multi-line string within the notebook.
3. Define sample JSON data as a list of dictionaries within the notebook.
4. Create a function read_csv() to:
   - Treat the CSV string as a file.
   - Use the CSV reader to parse rows.
   - Display the number of rows and first few rows.
5. Create a function read_json() to:
   - Access the predefined JSON data.
   - Check its type.
   - Display the total records and first few records.
6. Create a menu() function to:
   - Display options: Read CSV, Read JSON, Exit.
   - Call the respective function based on user input.
   - Repeat until the user chooses Exit.
7. Run the menu function.
8. Stop the program.

PROGRAM:

Json file:

```
# Sample JSON data
json_data = [
    {"id": 1, "name": "Harish", "age": 20},
    {"id": 2, "name": "Asha", "age": 22},
    {"id": 3, "name": "Manoj", "age": 21}
]
```

Csv file:

```
# Sample CSV data
csv_data = """id,name,age
1,Harish,20
2,Asha,22
3,Manoj,21
"""
```

PYTHON CODE:

```
import csv
import json
import io

# Function to read CSV
def read_csv():
    f = io.StringIO(csv_data)   # treat string as file
    reader = csv.reader(f)
    rows = list(reader)
    print("Total rows:", len(rows))
    print("First 3 rows:", rows[:3])

# Function to read JSON
def read_json():
    data = json_data
    print("Data type:", type(data))
    if isinstance(data, list):
        print("Total records:", len(data))
        print("First 3 records:", data[:3])
    else:
        print("Keys:", list(data.keys()))
```

```
# Menu function
def menu():
    while True:
        print("\n1. Read CSV")
        print("2. Read JSON")
        print("3. Exit")
        ch = input("Enter choice: ")
        if ch == '1':
            read_csv()
        elif ch == '2':
            read_json()
        elif ch == '3':
            print("Exiting...")
            break
        else:
            print("Invalid choice")
menu()
```

OUTPUT:

```
1. Read Csv
2. Read Json
3. Exit
Enter Choice:
Invalid Choice

1. Read Csv
2. Read Json
3. Exit
Enter Choice: 1
Total Rows: 4
First 3 Rows: [['Id', 'Name', 'Age'], ['1', 'Harish', '20'], ['2', 'Asha', '22']]


1. Read Csv
2. Read Json
3. Exit
Enter Choice: 2
Data Type: <Class 'List'>
Total Records: 3
First 3 Records: [{'Id': 1, 'Name': 'Harish', 'Age': 20}, {'Id': 2, 'Name': 'Asha', 'Age': 22}, {'Id':
3, 'Name': 'Manoj', 'Age': 21}]

1. Read Csv
2. Read Json
3. Exit
Enter Choice: 3
Exiting...
```

RESULT:

The program successfully demonstrates reading CSV and JSON data defined within the Jupyter Notebook itself. The user can view the total number of rows or records and the first few entries from both CSV and JSON formats through a simple menu-driven interface.

EX.NO: 03
Date: 30/07/2025

EXTRACT AND DISPLAY THE DATA FROM JSON

AIM:

To build a Python script that extracts and displays structured information such as subject marks, total, and average from JSON data.

ALGORITHM:

1. Import the json module.
2. Store the JSON student data inside the program as a string.
3. Use json.loads() to parse the JSON string into a Python object.
4. For each student in the JSON data:
    - ○ Extract name and id.
    - ○ Extract marks for math, science, and english.
    - ○ Calculate total = sum of marks.
    - ○ Calculate average = total / 3.
    - ○ Print the student's details, marks, total, and average.

PROGRAM:

Json file :

```
import json

# JSON data inside the program
data = '''
[
  {
    "name": "Shaheen",
    "id": "S101",
    "marks": {"math": 92, "science": 88, "english": 79}
  },
  {
    "name": "Sameeha",
    "id": "S102",
    "marks": {"math": 85, "science": 90, "english": 87}
  }
]
'''
```

PYTHON CODE:

```
import json

# Load JSON string
students = json.loads(data)

# Process each student
for s in students:
    name = s["name"]
    sid = s["id"]
    m = s["marks"]

    total = m["math"] + m["science"] + m["english"]
    avg = total / 3

    print(f"Student: {name} (ID: {sid})")
    print(f"  Math: {m['math']}, Science: {m['science']}, English: {m['english']}")
    print(f"  Total: {total}, Average: {avg:.2f}\n")
```

OUTPUT:

```
Student: Shaheen (Id: S101)
  Math: 92, Science: 88, English: 79
  Total: 259, Average: 86.33

Student: Sameeha (Id: S102)
  Math: 85, Science: 90, English: 87
  Total: 262, Average: 87.33
```

RESULT:

The program successfully extracts structured information from JSON and displays each student’s marks, total, and average.

EX.NO: 04
Date: 31/07/2025

ARRAYS IN DIFFERENT WAYS AND DISPLAY THEIR ATTRIBUTES (SHAPE, SIZE, DATATYPE, DIMENSIONS, ETC…)

AIM:

To write a Python program using NumPy to create arrays in different ways and display their attributes (shape, size, datatype, dimensions, etc.).

ALGORITHM:

1. Import the numpy module.
2. Display a menu for array initialization choices.
3. Based on user choice:
    - Create a normal array with user inputs.
    - Create zero array, ones array, linspace array, or reshaped array.
4. Display the array created.
5. Print its attributes: shape, size, data type, dimensions, and itemsize.
6. Repeat until the user exits.

PROGRAM:

```
import numpy as np


def array_attributes(arr):
  print("\nArray: \n", arr)
  print("Shape: ", arr.shape)
  print("Size: ", arr.size)
  print("Data Type: ", arr.dtype)
  print("Dimensions: ", arr.ndim)
  print("Itemsize: ", arr.itemsize)


def initialize_array(choice):
  if choice == 1:
    n = int(input("Enter the number of elements: "))
    elements=[]
    for i in range(n):
      val=int(input(f"Enter the element {i+1}: "))
      elements.append(val)
    return np.array(elements)

  elif choice == 2:
    r = int(input("Enter number of rows: "))
    c = int(input("Enter number of columns: "))
    return np.zeros((r,c), dtype=int)

  elif choice == 3:
    r = int(input("Enter number of rows: "))
    c = int(input("Enter number of columns: "))
    return np.ones((r,c), dtype=int)

  elif choice == 4:
    start = int(input("Enter the starting number: "))
    end = int(input("Enter the ending number: "))
    n = int(input("Enter the number of elements: "))
    return np.linspace(start, end, n)

  elif choice == 5:
    r = int(input("Enter number of rows: "))
    c = int(input("Enter number of columns: "))
    total = r*c
    print(f"Enter {total} elements: ")
    elements=[]
    for i in range(total):
      val=int(input(f"Enter the element {i+1}: "))
      elements.append(val)
    return np.array(elements).reshape(r,c)

  else:
    print("Invalid choice")
    return np.array([1,2,3])

def menu():
  print("\n==== ARRAY INITIALIZATION MENU ===")
  print("1. Normal Array (user input)")
  print("2. Zero Array")
  print("3. Ones Array")
  print("4. Arrange Array")
  print("5. Reshaped Array")
  print("0. Exit")

if __name__ == "__main__":
  while True:
    menu()
    choice = int(input("Enter your choice: "))

    if choice == 0:
      print("Exiting the program...")
      break

    arr = initialize_array(choice)
    array_attributes(arr)
```

OUTPUT:

```
==== Array Initialization Menu ===
1. Normal Array (User Input)
2. Zero Array
3. Ones Array
4. Arrange Array
5. Reshaped Array
0. Exit
Enter Your Choice: 1
Enter The Number Of Elements: 3
Enter The Element 1: 1
Enter The Element 2: 2
Enter The Element 3: 3

Array:
 [1 2 3]
Shape:  (3,)
Size:  3
Data Type:  Int64
Dimensions:  1
Itemsize:  8



==== Array Initialization Menu ===
1. Normal Array (User Input)
2. Zero Array
3. Ones Array
4. Arrange Array
5. Reshaped Array
0. Exit
Enter Your Choice: 0
Exiting The Program...
```

RESULT:

The program successfully creates arrays using NumPy and displays their attributes like shape, size, dimensions, etc.

EX.NO: 05
Date: 13/08/2025

NUMPY INDEXING, SLICING, RESHAPING, AND TRANSPOSE

AIM:

To demonstrate the concepts of array indexing, slicing, reshaping, and transposing using NumPy in Python.

ALGORITHM:

1. **Indexing:**
   - ○ Take input as space-separated integers and convert them into a NumPy array.
   - ○ Ask the user to enter an index (positive/negative).
   - ○ Display the element at that index.
2. **Slicing:**
   - ○ Take another set of space-separated integers and convert them into a NumPy array.
   - ○ Ask the user for start index, end index, and step value.
   - ○ Perform slicing using the given values and display the sliced array.
3. **Reshaping and Transpose:**
   - ○ Take a list of space-separated integers and convert them into a NumPy array.
   - ○ Ask the user to input number of rows and columns.
   - ○ Check if reshaping is possible (rows × cols = array size).
   - ○ If possible, reshape the array and display it.
   - ○ Perform transpose operation on the reshaped array and display it.
   - ○ If not possible, display an error message.

PROGRAM:

```
import numpy as np

#Indexing
print("\n---Indexing---")
arr1=np.array(list(map(int, input("Enter number for indexing (space-separated):").split())))
index=int(input("Enter index to access (can use neagtive too);"))
print(f"Element at index {index}:", arr1[index])

#Slicing
print("\n--Slicing--")
arr2=np.array(list(map(int, input("Enter numbers for slicing (space-seprated):").split())))
start = int(input("Enter start index for slicing: "))
end = int(input("Enter end index for slicing: "))
step = int(input("Enter step value (default = 1): ")or 1)
print(f"Sliced Array arr[{start}:{end}:{step}]: ", arr2[start:end:step])

#Reshaping and Transpose
print("---Reshaping and Transpose---")
arr3 = np.array(list(map(int, input("Enter numbers for reshaping (space-separated):
").split())))
rows = int(input("Enter number of rows for Reshape: "))
cols = int(input("Enter number of columns for Reshape: "))
if rows*cols==arr3.size:
  reshaped = arr3.reshape(rows, cols)
  print(f"\n Reshaped Array({rows}x{cols}):\n", reshaped)
  transposed = reshaped.T
  print(f"\nTranspose of the Array ({cols}x{rows}):\n",transposed)
else:
  print(f"Reshape not possible! (Array size = {arr3.size}, but {rows}x{cols} = {rows*cols})")
```

OUTPUT:

```
---Indexing---
Enter Number For Indexing (Space-Separated):1 2 3 4 5 6
Enter Index To Access (Can Use Neagtive Too);-3
Element At Index -3: 4

--Slicing--
Enter Numbers For Slicing (Space-Seprated):1 2 3 4 5 6
Enter Start Index For Slicing: 3
Enter End Index For Slicing: 6
Enter Step Value (Default = 1): 2
Sliced Array Arr[3:6:2]:  [4 6]

---Reshaping And Transpose---
Enter Numbers For Reshaping (Space-Separated): 1 2 3 4
Enter Number Of Rows For Reshape: 2
Enter Number Of Columns For Reshape: 2

 Reshaped Array(2x2):
 [[1 2]
 [3 4]]

Transpose Of The Array (2x2):
 [[1 3]
 [2 4]]
```

RESULT:

The program successfully demonstrates how to perform indexing, slicing, reshaping, and transpose operations on NumPy arrays based on user inputs.

EX.NO: 06
Date: 14/08/2025

STUDENT MARK SHEET USING NUMPY AND PANDAS

AIM:

To write a Python program using NumPy and Pandas to create a student mark sheet, calculate total and average marks for each student, and display them in tabular format.

ALGORITHM:

1. Import the numpy and pandas modules.
2. Input the number of students and subjects.
3. Read the subject names.
4. For each student:
   - Input the student's name.
   - Input marks for all subjects.
   - Store the marks in a list.
5. Convert the list of marks into a NumPy array.
6. Create a Pandas DataFrame with subject names as columns and student names as row indices.
7. Add Total and Average columns.
8. Display the student mark sheet in tabular form.

PROGRAM:

```
import numpy as np
import pandas as pd

# Number of students and subjects
students = int(input("Enter the number of students: "))
subjects = int(input("Enter the number of subjects: "))

# Get subject names
subject_names = []
for j in range(subjects):
    sub = input(f"Enter the name of Subject {j+1}: ")
    subject_names.append(sub)

# Get marks for each student
marks_list = []
student_names = []

for i in range(students):
    name = input(f"\nEnter the name of Student {i+1}: ")
    student_names.append(name)
    student_marks = []
    for j in range(subjects):
        mark = int(input(f"  Enter marks in {subject_names[j]}: "))
        student_marks.append(mark)
    marks_list.append(student_marks)

# Convert to NumPy array
marks = np.array(marks_list)

# Create DataFrame
df = pd.DataFrame(marks, columns=subject_names, index=student_names)

# Add Total and Average columns
df["Total"] = df.sum(axis=1)
df["Average"] = df.mean(axis=1)

# Display Student Mark Sheet
print("\n=== Student Mark Sheet ===")
print(df)
```

OUTPUT:

```
Enter The Number Of Students: 3
Enter The Number Of Subjects: 5
Enter The Name Of Subject 1: Maths
Enter The Name Of Subject 2: Science
Enter The Name Of Subject 3: English
Enter The Name Of Subject 4: Social
Enter The Name Of Subject 5: Language

Enter The Name Of Student 1: Harish
  Enter Marks In Maths: 96
  Enter Marks In Science: 88
  Enter Marks In English: 96
  Enter Marks In Social: 87
  Enter Marks In Language: 79

Enter The Name Of Student 2: Sowmiya
  Enter Marks In Maths: 88
  Enter Marks In Science: 98
  Enter Marks In English: 78
  Enter Marks In Social: 70
  Enter Marks In Language: 54

Enter The Name Of Student 3: Hari
  Enter Marks In Maths: 87
  Enter Marks In Science: 69
  Enter Marks In English: 95
  Enter Marks In Social: 92
  Enter Marks In Language: 79

=== Student Mark Sheet ===
          Maths  Science  English  Social  Language  Total     Average
Harish      96       88       96      87        79    446  148.666667
Sowmiya     88       98       78      70        54    388  129.333333
Hari        87       69       95      92        79    422  140.666667
```

RESULT:

The program successfully generates a student mark sheet, displays subject-wise marks, and calculates total and average marks for each student in a clear tabular format.

EX.NO: 07
Date: 21/08/2025

EMPLOYEE SALARY ANALYSIS USING SERIES AND DATA FRAMES

AIM:

To write a Python program that stores employee details (ID, Name, Salary) and displays them using Pandas Series and DataFrame.

ALGORITHM:

1. Start the program.
2. Import the pandas library.
3. Read the number of employees n.
4. Initialize empty lists for ID, Name, and Salary.
5. For each employee:
    - Input Employee ID.
    - Input Employee Name.
    - Input Salary.
    - Append details to respective lists.
6. Create a Series from the salary list and display it.
7. Create a DataFrame using Employee ID, Name, and Salary.
8. Display the DataFrame.
9. Stop the program.

PROGRAM:

```
import pandas as pd

# Input: Number of Employees
n = int(input("Enter number of employees: "))

# Initialize lists
emp_id, emp_name, salary = [], [], []

# Input Employee Details
for i in range(n):
    print(f"\nEmployee {i+1}:")
    emp_id.append(int(input("ID: ")))
    emp_name.append(input("Name: "))
    salary.append(float(input("Salary: ")))

# Create Series
s = pd.Series(salary)
print("\nEmployee Salaries (Series):")
print(s)

# Create DataFrame
df = pd.DataFrame({"ID": emp_id, "Name": emp_name, "Salary": salary})
print("\nEmployee DataFrame:")
print(df)
```

OUTPUT:

```
Enter Number Of Employees: 2

Employee 1:
Id: 101
Name: Harish
Salary: 7.5

Employee 2:
Id: 102
Name: Pavi
Salary: 7.5



Employee Salaries (Series):
0    7.5
1    7.5
Dtype: Float64

Employee Dataframe:
    Id    Name  Salary
0  101  Harish     7.5
1  102    Pavi     7.5
```

RESULT:

The program successfully accepts employee details and displays:

- Employee salaries using Pandas Series.
- Complete employee information using a Pandas DataFrame.

EX.NO: 08
Date: 22/08/2025

DICE THROW SIMULATION USING NUMPY

AIM:

To simulate two dice throws using NumPy and display the frequency and distribution of their sums.

ALGORITHM:

1. Set the number of throws and dice sides.
2. Generate random values for both dice using np.random.randint().
3. Calculate the sum of both dice for each throw.
4. Count frequencies of each possible sum.
5. Display results in a frequency table and ASCII histogram.

PROGRAM:

```
import numpy as np

# Parameters
throws, sides = 1000, 6

# Simulate dice throws and calculate sums
sums = np.random.randint(1, sides + 1, throws) + np.random.randint(1, sides + 1, throws)

# Count frequencies
freq = {i: np.count_nonzero(sums == i) for i in range(2, 2 * sides + 1)}

# Display frequency table
print("\nSum | Frequency | Percentage")
print("-----------------------------")
for val, count in freq.items():
    print(f"{val:^3} | {count:^9} | {count/throws*100:>7.2f}%")

# ASCII histogram
print("\nSum Distribution (ASCII Histogram)")
for val, count in freq.items():
    print(f"{val:2}: {'*' * (count * 50 // throws)}")
```

OUTPUT:

```
Sum | Frequency | Percentage
-----------------------------
 2 |    32    |   3.20%
 3 |    60    |   6.00%
 4 |    74    |   7.40%
 5 |    105   |  10.50%
 6 |    146   |  14.60%
 7 |    160   |  16.00%
 8 |    149   |  14.90%
 9 |    104   |  10.40%
10 |    88    |   8.80%
11 |    51    |   5.10%
12 |    31    |   3.10%

Sum Distribution (Ascii Histogram)
 2: *
 3: ***
 4: ***
 5: *****
 6: *******
 7: ********
 8: *******
 9: *****
10: ****
11: **
12: *
```

RESULT:

The program simulates dice throws, shows frequency of each sum, and visualizes results using an ASCII histogram.

EX.NO: 09
Date: 28/08/2025

MATPLOTLIB BASICS – MONTHLY SALES DATA

AIM:

To visualize Monthly Sales Data using Line, Bar, Scatter, Pie, and Histogram plots in Python using Matplotlib.

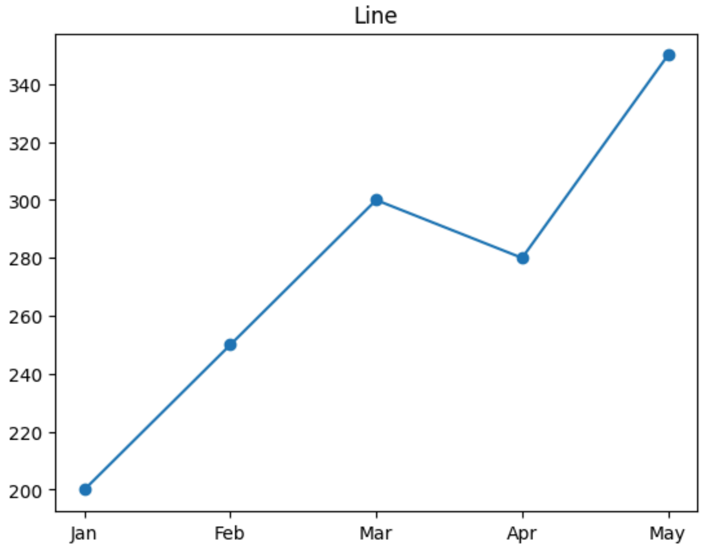
ALGORITHM:

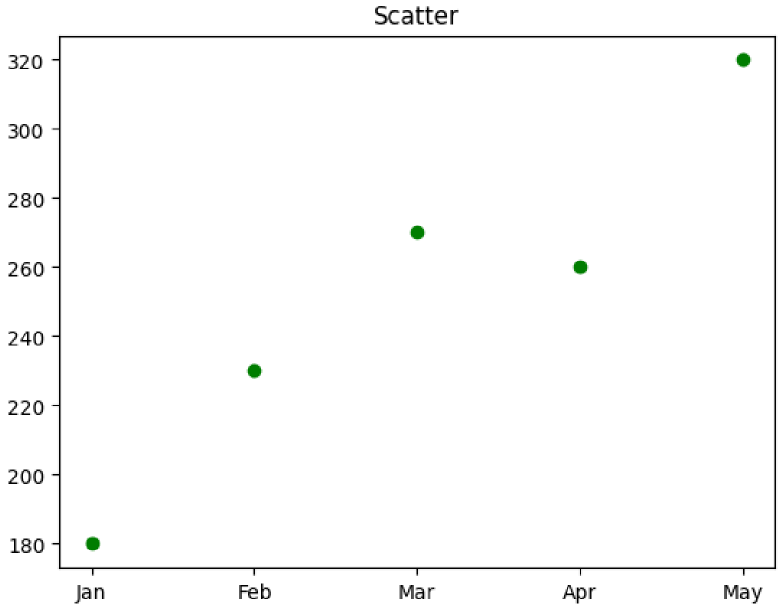
1. Import matplotlib.pyplot.
2. Define lists for months and sales data.
3. Plot Line, Bar, Scatter, Pie, and Histogram using respective functions.
4. Use plt.show() to display each plot.

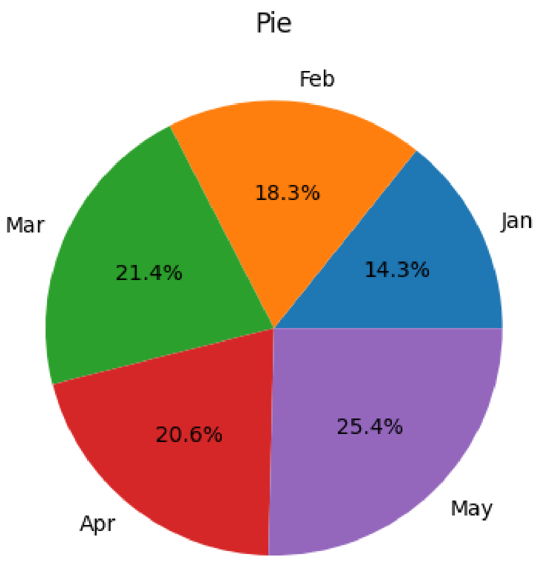
PROGRAM:

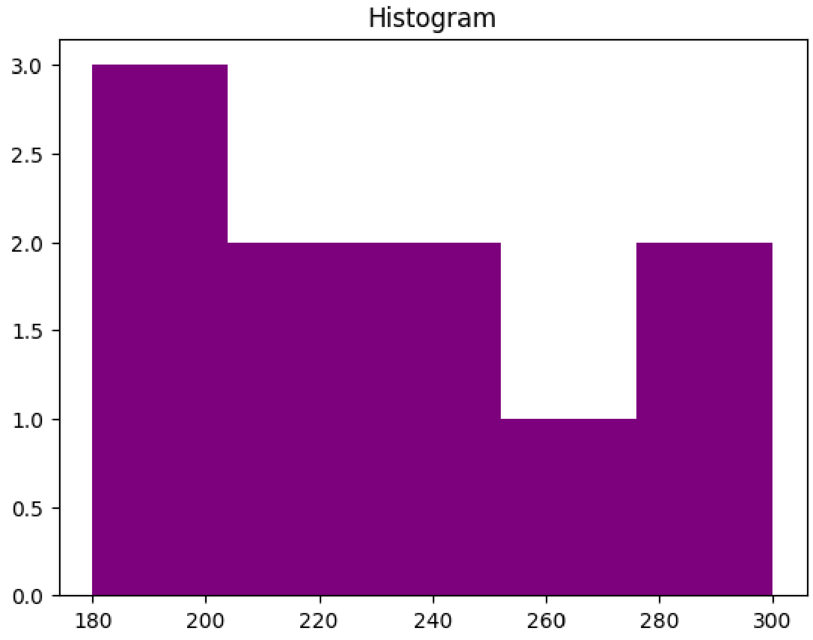
```
import matplotlib.pyplot as plt

m = ["Jan","Feb","Mar","Apr","May"]
s1 = [200,250,300,280,350]
s2 = [180,230,270,260,320]

plt.plot(m,s1,marker='o'); plt.title("Line"); plt.show()
plt.bar(m,s2,color='orange'); plt.title("Bar"); plt.show()
plt.scatter(m,s2,color='green'); plt.title("Scatter"); plt.show()
plt.pie(s2,labels=m,autopct="%1.1f%%"); plt.title("Pie"); plt.show()
plt.hist([180,200,190,250,300,280,220,210,230,260],bins=5,color='purple')
plt.title("Histogram"); plt.show()
```

OUTPUT:


Line
340
320
300
280
260
240
220
200
Jan
Feb
Mar
Apr
May


Scatter
320
300
280
260
240
220
200
180
Jan
Feb
Mar
Apr
May





RESULT:

Different visualizations of monthly sales data are displayed successfully.

EX.NO: 10
Date: 29/08/2025

MATPLOTLIB ADVANCED – WEATHER TRENDS

AIM:

To analyze weather trends using advanced matplotlib plots such as Stacked Area, Pie, Scatter, and Histogram.

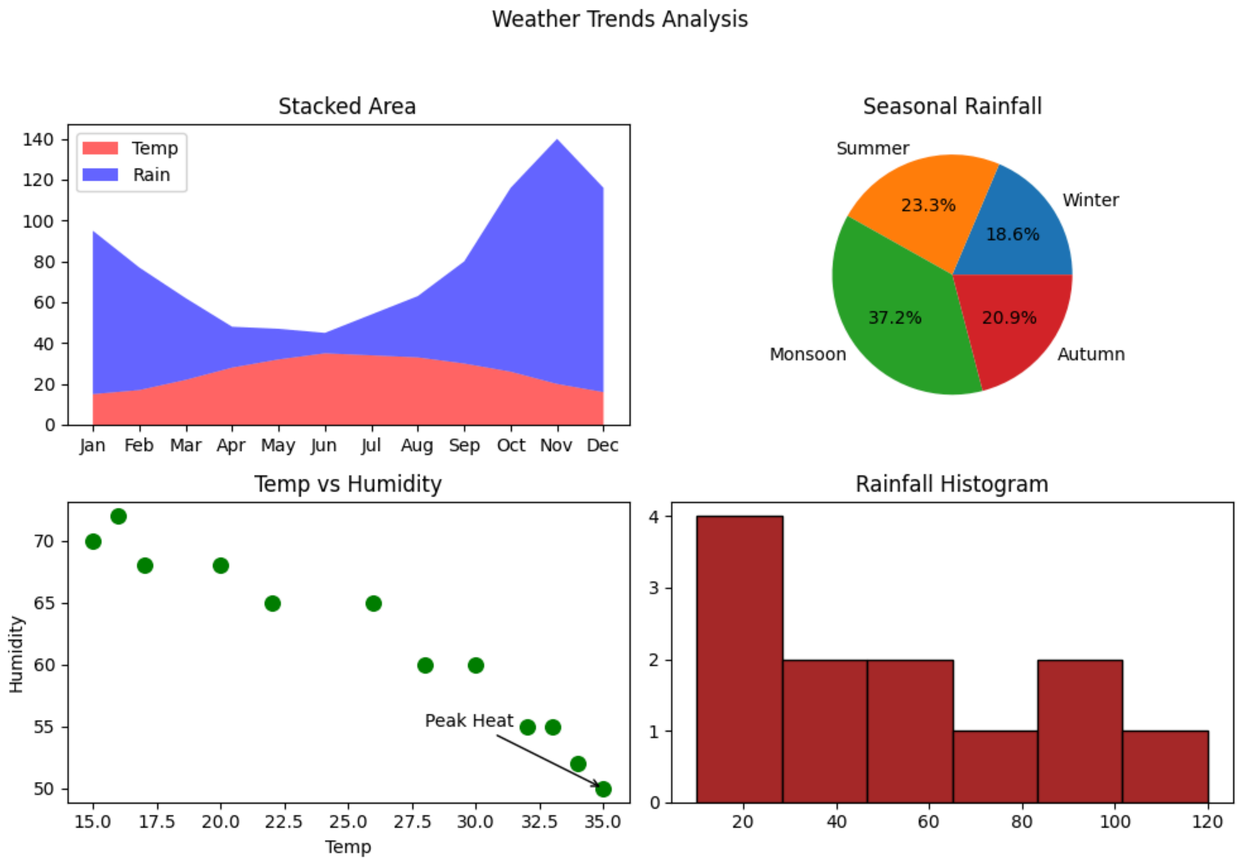
ALGORITHM:

1. Import matplotlib and numpy.
2. Define data for temperature, rainfall, and humidity.
3. Create subplots using plt.subplots().
4. Plot:
   a) Stacked Area for Temperature & Rainfall
   b) Pie chart for seasonal rainfall
   c) Scatter for Temperature vs Humidity
   d) Histogram for rainfall distribution
5. Use plt.tight_layout() and plt.show() to display.

PROGRAM:

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

# --- Short and Easy Version: Weather Trends ---
months =
np.array(["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"])
temp = [15,17,22,28,32,35,34,33,30,26,20,16]
rain = [80,60,40,20,15,10,20,30,50,90,120,100]
hum = [70,68,65,60,55,50,52,55,60,65,68,72]
fig, axs = plt.subplots(2,2,figsize=(10,7))

# Stacked Area Plot
axs[0,0].stackplot(months,temp,rain,labels=["Temp","Rain"],colors=['r','b'],alpha=0.6)
axs[0,0].set_title("Stacked Area"); axs[0,0].legend()

# Pie Chart
axs[0,1].pie([200,250,400,225],labels=["Winter","Summer","Monsoon","Autumn"],autopct=
"%1.1f%%")
axs[0,1].set_title("Seasonal Rainfall")

# Scatter Plot
axs[1,0].scatter(temp,hum,color='g',s=70)
axs[1,0].set_title("Temp vs Humidity")
axs[1,0].set_xlabel("Temp"); axs[1,0].set_ylabel("Humidity")
axs[1,0].annotate("Peak Heat",xy=(35,50),xytext=(28,55),arrowprops=dict(arrowstyle='->'))

# Histogram
axs[1,1].hist(rain,bins=6,color='brown',edgecolor='black')
axs[1,1].set_title("Rainfall Histogram")

plt.suptitle("Weather Trends Analysis")
plt.tight_layout(rect=[0,0,1,0.95])
plt.show()
```

OUTPUT:



RESULT:

Weather trend visualizations are displayed successfully using advanced matplotlib features.

EX.NO: 11
Date: 04/09/2025

BASIC DASHBOARD IN GOOGLE COLAB
WEBSITE ANALYTICS

AIM:

To create an interactive data visualization dashboard using Google Colab widgets and Matplotlib.

ALGORITHM:

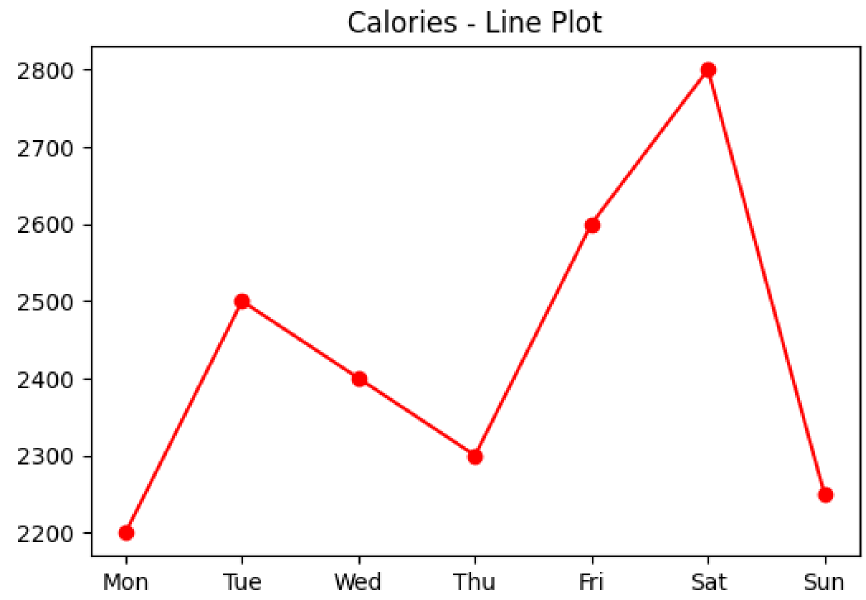
1. Import matplotlib and ipywidgets modules.
2. Define data for steps, calories, and sleep hours.
3. Define a plotting function that takes plot type and metric as input.
4. Based on user selection, draw the respective plot (Line, Bar, Scatter, Pie, Histogram).
5. Use interact() to create an interactive dashboard.

PROGRAM:

```
import matplotlib.pyplot as plt
import numpy as np
from ipywidgets import interact

# --- Short & Easy: Basic Dashboard (Google Colab) ---
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
steps = [5000,7500,8000,6500,9000,10000,7000]
cal = [2200,2500,2400,2300,2600,2800,2250]
sleep = [7,6.5,8,7.5,6,8,7]

def dash(ptype, metric):
    plt.figure(figsize=(6,4))
    data, color = (steps,'b') if metric=='Steps' else (cal,'r') if metric=='Calories' else (sleep,'g')
    if ptype=='Line Plot': plt.plot(days,data,marker='o',color=color)
    elif ptype=='Bar Chart': plt.bar(days,data,color=color)
    elif ptype=='Scatter Plot': plt.scatter(days,data,color=color,s=80)
    elif ptype=='Pie Chart': plt.pie(data,labels=days,autopct='%1.1f%%')
    else: plt.hist(data,bins=6,color=color,edgecolor='black')
    plt.title(f"{metric} - {ptype}"); plt.show()

interact(dash,ptype=['Line Plot','Bar Chart','Scatter Plot','Pie
Chart','Histogram'],metric=['Steps','Calories','Sleep Hours'])
```

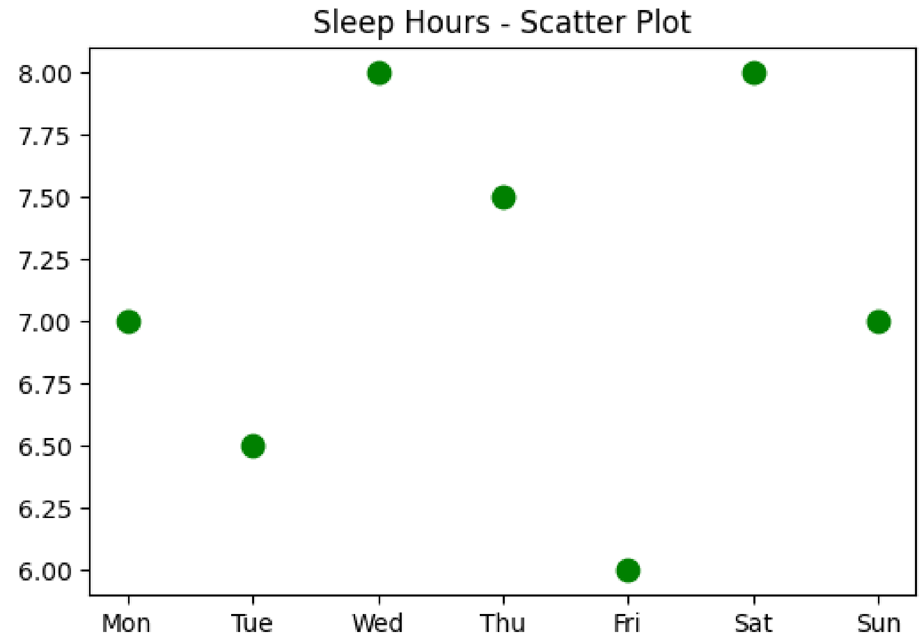
OUTPUT:

Ptype: Line Plot
Metric: Calories



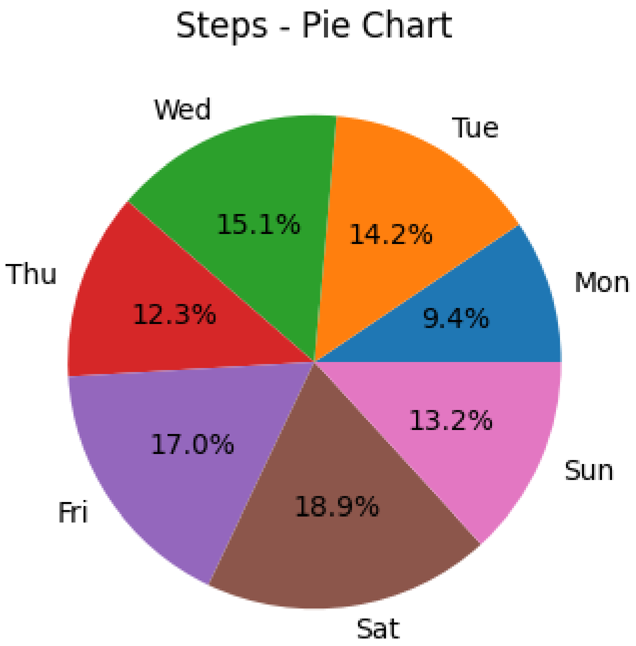
Dash
Def Dash(Ptype, Metric)
<No Docstring>

Ptype: Scatter Plot
Metric: Sleep Hours



Dash
Def Dash(Ptype, Metric)
<No Docstring>

Ptype: Pie Chart
Metric: Steps



Dash
Def Dash(Ptype, Metric)
<No Docstring>

RESULT:

An interactive dashboard is created in Google Colab where users can visualize Steps, Calories, and Sleep Hours using different plot types.

EX.NO: 12
Date: 12/09/2025

VISUALIZE SALES AND STUDENT DATA USING SEABORN LIBRARY

AIM:

To visualize and analyze sales and student data using Seaborn for better understanding of trends and distributions.

ALGORITHM:

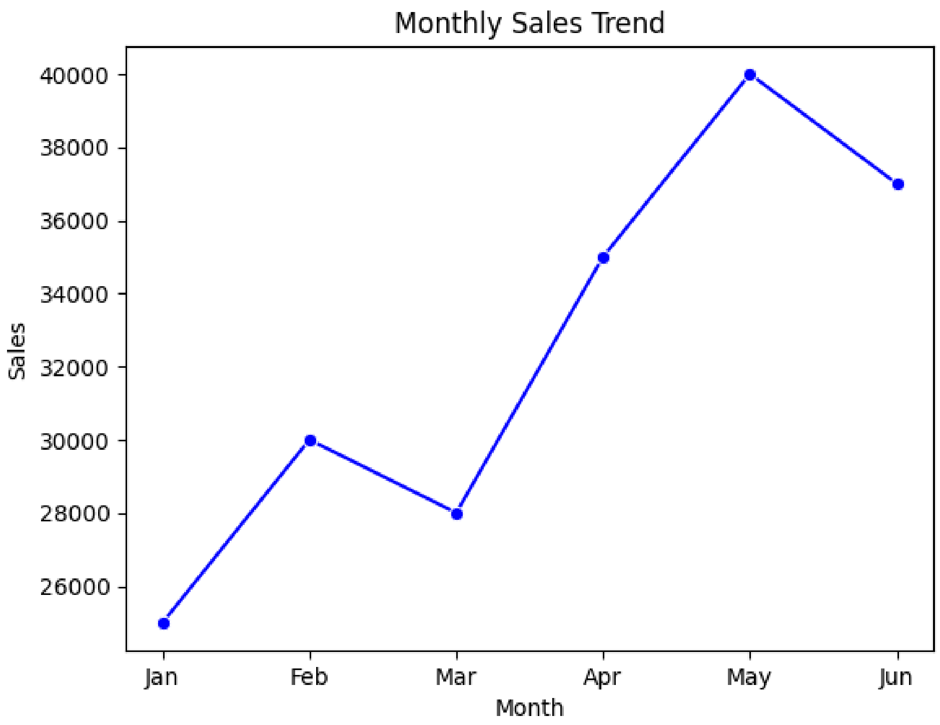
1. Import pandas, seaborn, and matplotlib libraries.
2. Create sample datasets for sales and student data.
3. Plot line and bar charts for sales trends and region comparison.
4. Plot bar, box, and count plots for student marks, departments, and gender.
5. Display insights based on observed data.

PROGRAM:

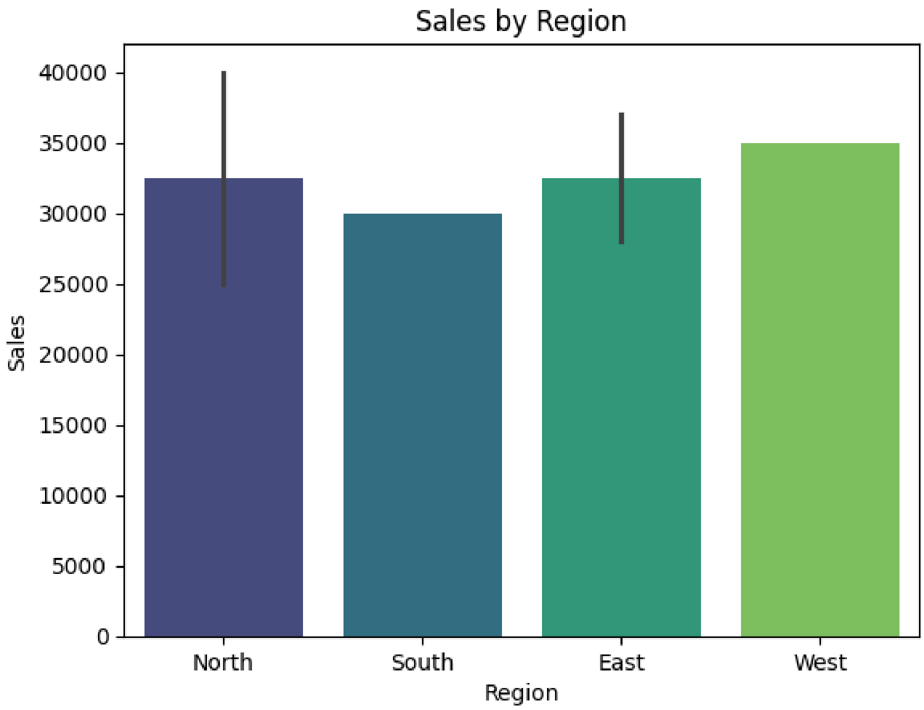
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# --- Short & Easy: Seaborn Data Visualization ---
sales = pd.DataFrame({
    'Month': ['Jan','Feb','Mar','Apr','May','Jun'],
    'Sales': [25000,30000,28000,35000,40000,37000],
    'Region': ['North','South','East','West','North','East']
})

print("Sales Data:\n", sales, "\n")
sns.lineplot(x='Month', y='Sales', data=sales, marker='o', color='b')
plt.title('Monthly Sales Trend'); plt.show()

sns.barplot(x='Region', y='Sales', data=sales, palette='viridis')
plt.title('Sales by Region'); plt.show()

students = pd.DataFrame({
    'Student': ['Rahul','Priya','Amit','Sneha','Kiran','Fatima'],
    'Marks': [85,90,78,92,88,76],
    'Gender': ['Male','Female','Male','Female','Male','Female'],
    'Dept': ['CS','IT','CS','IT','ECE','ECE']
})

print("Student Data:\n", students, "\n")
sns.barplot(x='Student', y='Marks', data=students, palette='Set2')
plt.title('Marks of Students'); plt.show()

sns.boxplot(x='Dept', y='Marks', data=students, palette='Set3')
plt.title('Dept-wise Marks'); plt.show()

sns.countplot(x='Gender', data=students, palette='coolwarm')
plt.title('Gender Distribution'); plt.show()

print("Insights:\n1. Sales increase till May.\n2. North & East lead in sales.\n3. Sneha scored
highest (92).\n4. IT students perform slightly better.\n5. More female students overall.")
```

OUTPUT:

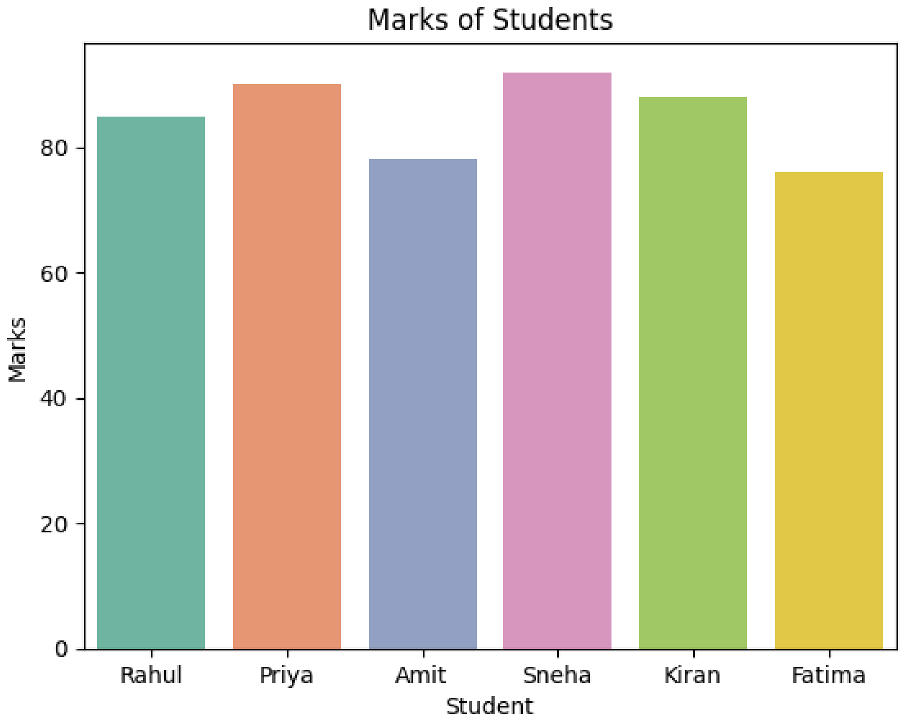
```
Sales Data:
Month  Sales Region
0   Jan  25000  North
1   Feb  30000  South
2   Mar  28000   East
3   Apr  35000   West
4   May  40000  North
5   Jun  37000   East
```



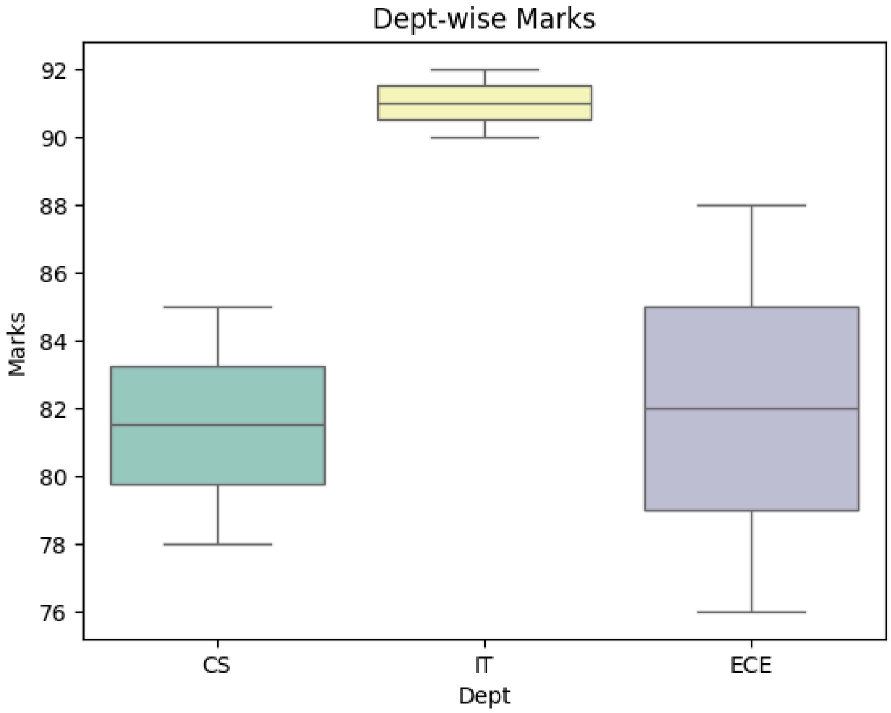
```
/Tmp/Ipython-Input-3606687357.Py:16: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0. Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Barplot(X='Region', Y='Sales', Data=Sales, Palette='Viridis')
```



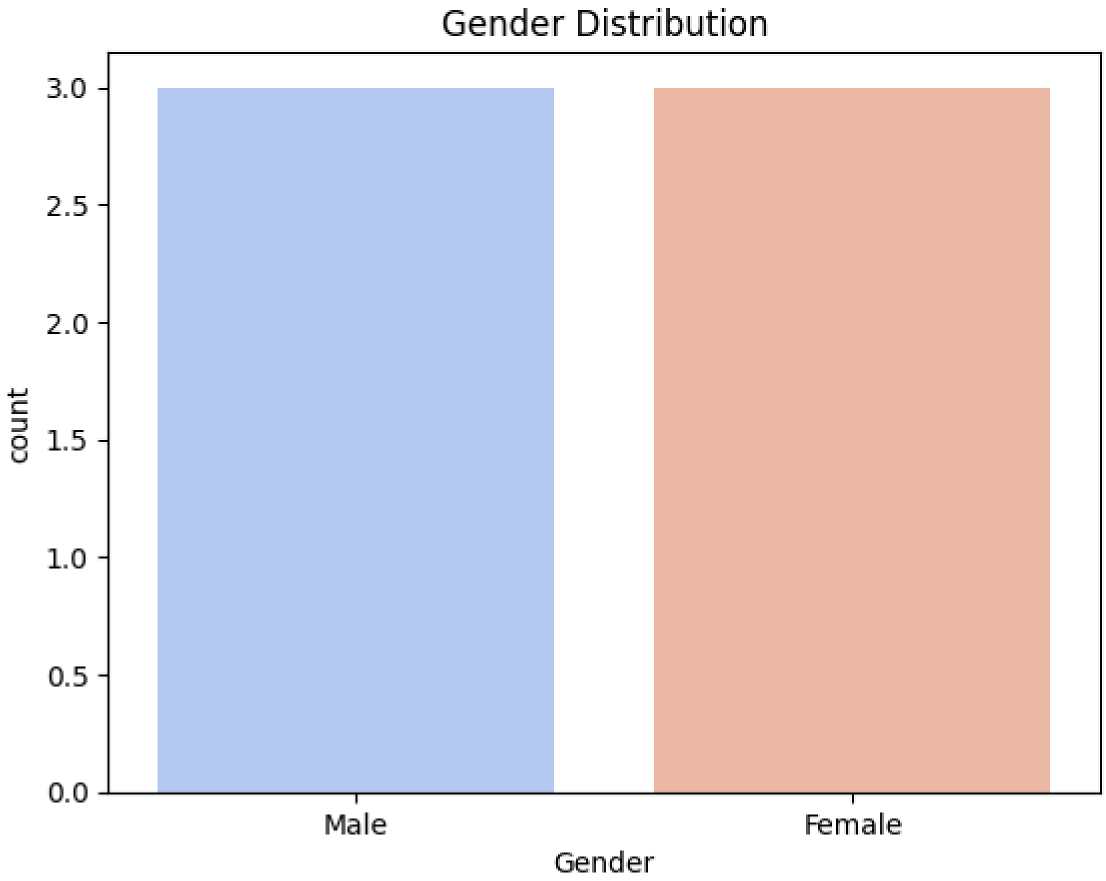
```
Student Data:
  Student  Marks  Gender Dept
0   Rahul     85    Male   Cs
1   Priya     90  Female   It
2    Amit     78    Male   Cs
3   Sneha     92  Female   It
4   Kiran     88    Male  Ece
5  Fatima     76  Female  Ece


/Tmp/Ipython-Input-3606687357.Py:27: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Barplot(X='Student', Y='Marks', Data=Students, Palette='Set2')
```



```
/Tmp/Ipython-Input-3606687357.Py:30: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Boxplot(X='Dept', Y='Marks', Data=Students, Palette='Set3')
```



```
/Tmp/Ipython-Input-3606687357.Py:33: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Countplot(X='Gender', Data=Students, Palette='Coolwarm')
```



Insights:
1. Sales Increase Till May.
2. North & East Lead In Sales.
3. Sneha Scored Highest (92).
4. It Students Perform Slightly Better.
5. More Female Students Overall.

RESULT:

Various Seaborn visualizations were successfully created to represent sales and student data trends.

EX.NO: 13
Date: 15/09/2025

TEAM-BASED DATASET SELECTION AND CLEANING

AIM:

To perform dataset cleaning by handling missing values, duplicates, and inconsistent data.

ALGORITHM:

1. Create a sample dataset with missing and duplicate values.
2. Inspect the dataset using isnull() and duplicated().
3. Fill missing numeric values using median.
4. Remove duplicate rows using drop_duplicates().
5. Standardize text data (e.g., Gender) using str.title().
6. Display the cleaned dataset.

PROGRAM:

```
import pandas as pd

# --- Short & Easy: Team-Based Dataset Cleaning ---
data = {
    'Student': ['Rahul','Priya','Amit','Sneha','Kiran','Fatima','Raj','Ayesha','Vijay','Sara'],
    'Gender': ['Male','Female','Male','Female','Male','Female','Male','Female','Male','Female'],
    'Math': [88,92,79,None,85,76,95,None,82,90],
    'Read': [95,85,80,89,None,77,90,93,84,None],
    'Write': [90,None,78,88,84,75,92,91,None,87]
}

df = pd.DataFrame(data)
print("Original Data:\n", df, "\n")
print("Missing Values:\n", df.isnull().sum(), "\n")
print("Duplicates:", df.duplicated().sum(), "\n")

# Cleaning
df['Math'] = df['Math'].fillna(df['Math'].median())
df['Read'] = df['Read'].fillna(df['Read'].median())
df['Write'] = df['Write'].fillna(df['Write'].median())
df = df.drop_duplicates()
df['Gender'] = df['Gender'].str.title()

print("Cleaned Data:\n", df, "\n")
print("Missing After Cleaning:\n", df.isnull().sum())
```

OUTPUT:

```
===== Original Dataset =====
  Student  Gender  Math Score  Reading Score  Writing Score
0   Rahul        Male            88.0            95.0                  90.0
1   Priya     Female            92.0            85.0                  Nan
2   Amit         Male            79.0             80.0                 78.0
3   Sneha    Female             Nan             89.0                  88.0
4   Kiran        Male            85.0            Nan                   84.0
5  Fatima     Female    76.0            77.0                  75.0
6  Raj           Male            95.0            90.0                  92.0
7  Ayesha   Female    Nan             93.0                  91.0
8   Vijay        Male            82.0            84.0                  Nan
9   Sara         Female          90.0            Nan                   87.0

===== Missing Values =====
Student          0
Gender           0
Math Score       2
Reading Score 2
Writing Score  2
Dtype: Int64

===== Duplicate Rows =====
0

===== Cleaned Dataset =====
  Student  Gender  Math Score  Reading Score  Writing Score
0   Rahul        Male            88.0            95.0            90.0
1   Priya     Female    92.0            85.0            87.5
2   Amit         Male            79.0            80.0            78.0
3   Sneha    Female    86.5            89.0            88.0
4   Kiran        Male            85.0            87.0            84.0
5   Fatima    Female      76.0          77.0            75.0
6   Raj         Male    95.0            90.0            92.0
7   Ayesha    Female          86.5            93.0            91.0
8   Vijay        Male            82.0            84.0            87.5
9   Sara       Female    90.0            87.0            87.0

===== Missing Values After Cleaning =====
Student          0
Gender                   0
Math Score       0
Reading Score 0
```

```
Writing Score  0
Dtype: Int64
```

RESULT:

The dataset was cleaned successfully by replacing missing values, removing duplicates, and standardizing text data.

EX.NO: 14
Date: 26/09/2025

EXPLORATORY DATA ANALYSIS (EDA) ON IRIS DATASET

AIM:

To analyze the Iris dataset using Python for understanding feature distribution, relationships, and insights about different flower species.

ALGORITHM:

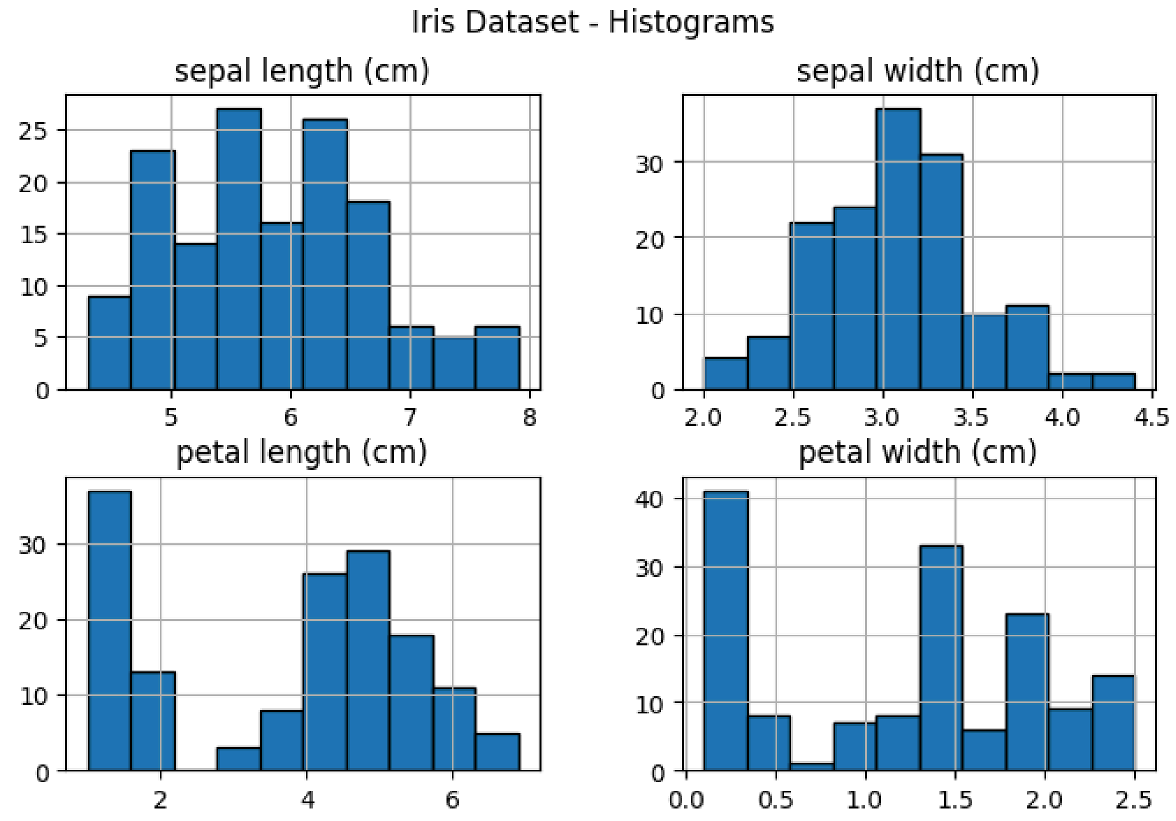
1. Import libraries: pandas, matplotlib, seaborn, and sklearn.datasets.
2. Load the Iris dataset using load_iris().
3. Convert data into a Pandas DataFrame.
4. Display basic info, head, and summary statistics.
5. Check for missing values using isnull().sum().
6. Plot histograms and boxplots for univariate analysis.
7. Plot pairplot for multivariate analysis.
8. Display correlation matrix and heatmap.
9. Print key insights.

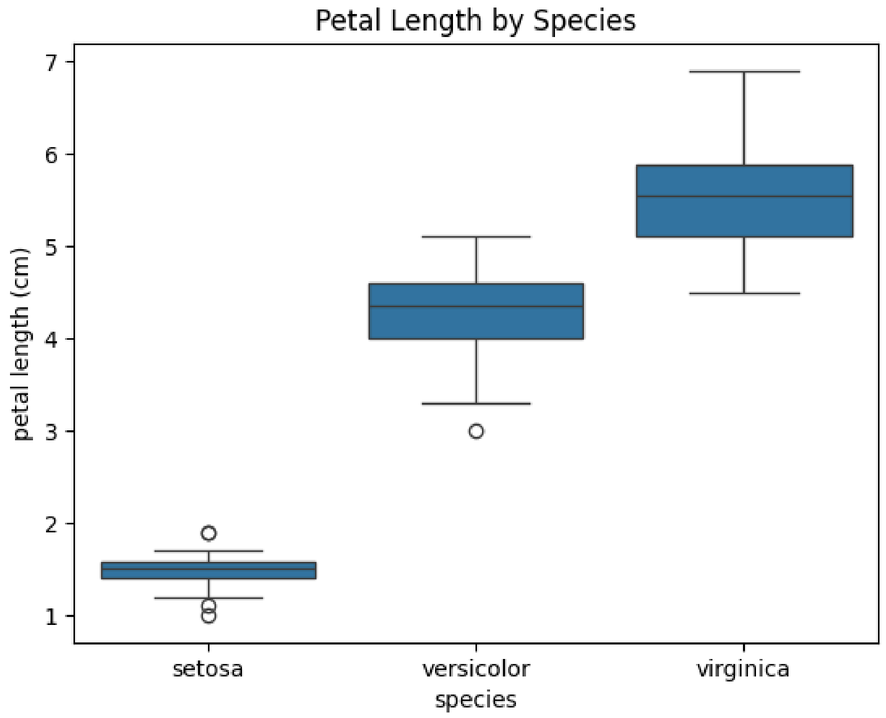
PROGRAM:

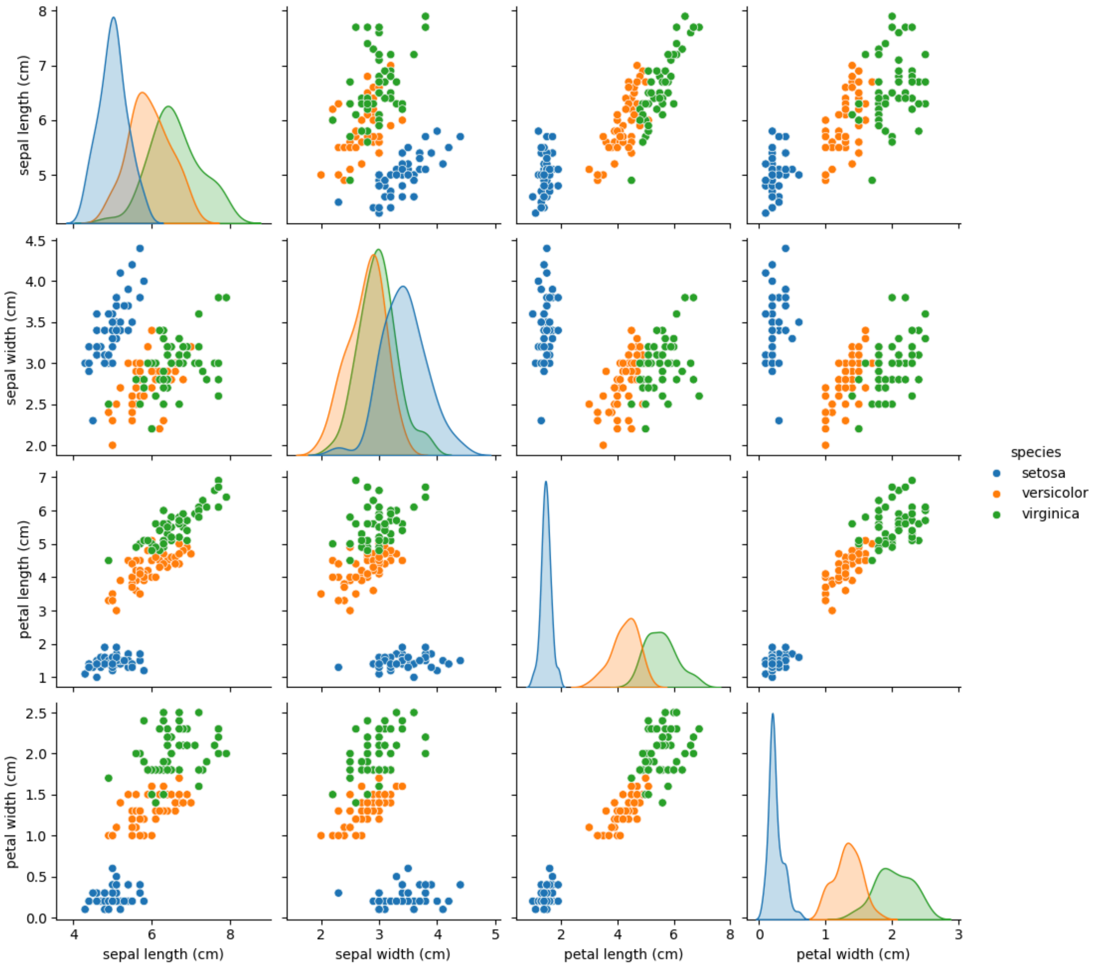
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

# Load dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['species'] = pd.Series(data.target).map({0:'setosa', 1:'versicolor', 2:'virginica'})

# Basic info
print(df.info())
print(df.head())
print(df.isnull().sum())

# Histograms
df.hist(edgecolor='black', figsize=(8,5))
plt.suptitle('Iris Dataset - Histograms')
plt.show()

# Boxplot
sns.boxplot(x='species', y='petal length (cm)', data=df)
plt.title('Petal Length by Species')
plt.show()

# Pairplot
sns.pairplot(df, hue='species')
plt.show()

# Correlation heatmap (numeric only)
corr = df.drop('species', axis=1).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

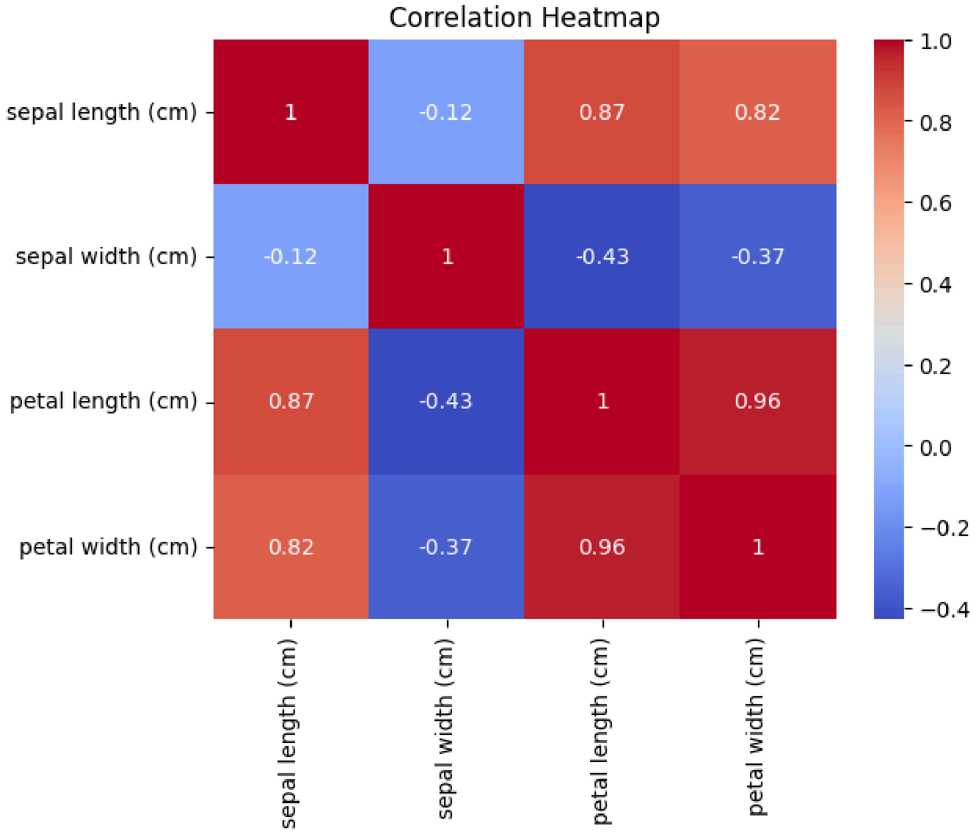
OUTPUT:

```
<Class 'Pandas.Core.Frame.Dataframe'>
Rangeindex: 150 Entries, 0 To 149
Data Columns (Total 5 Columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Sepal Length (Cm)  150 Non-Null    Float64
 1   Sepal Width (Cm)   150 Non-Null    Float64
 2   Petal Length (Cm)  150 Non-Null    Float64
 3   Petal Width (Cm)   150 Non-Null    Float64
 4   Species            150 Non-Null    Object
Dtypes: Float64(4), Object(1)
Memory Usage: 6.0+ Kb
None
   Sepal Length (Cm)  Sepal Width (Cm)  Petal Length (Cm)  Petal Width (Cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  Species
0  Setosa
1  Setosa
2  Setosa
3  Setosa
4  Setosa
Sepal Length (Cm)    0
Sepal Width (Cm)     0
Petal Length (Cm)    0
Petal Width (Cm)     0
Species              0
Dtype: Int64
```


Iris Dataset - Histograms
sepal length (cm)
0
5
10
15
20
25
5
6
7
8
sepal width (cm)
0
10
20
30
2.0
2.5
3.0
3.5
4.0
4.5
petal length (cm)
0
10
20
30
2
4
6
petal width (cm)
0
10
20
30
40
0.0
0.5
1.0
1.5
2.0
2.5


Petal Length by Species
petal length (cm)
1
2
3
4
5
6
7
setosa
versicolor
virginica
species


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
species
setosa
versicolor
virginica
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)


Correlation Heatmap
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
1
-0.12
0.87
0.82
-0.12
1
-0.43
-0.37
0.87
-0.43
1
0.96
0.82
-0.37
0.96
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

RESULT:

EDA on the Iris dataset was performed successfully.

- No missing values found.
- Each species has 50 samples.
- Petal features show strong correlation.
- Setosa is easily separable, while Versicolor and Virginica overlap slightly.
- Dataset is clean and ready for ML modeling.

EX.NO: 15
Date: 29/09/2025

EDA ON TITANIC DATASET

AIM:

To perform Exploratory Data Analysis (EDA) on the Titanic dataset using Python libraries — Pandas, Seaborn, and Matplotlib — and extract meaningful insights through data visualization.

ALGORITHM:

1. Import Libraries:
   Import the necessary Python libraries — pandas, seaborn, and matplotlib.pyplot.
2. Load Dataset:
   Use Seaborn's built-in Titanic dataset with sns.load_dataset('titanic').
3. Display Basic Information:
   - View dataset structure using info().
   - Display the first 5 rows using head().
   - Show summary statistics using describe().
4. Check Missing Values:
   Identify columns with missing data using isnull().sum().
5. Univariate Analysis:
   - Plot survival count (survived column).
   - Plot passenger class distribution (pclass).
   - Plot age distribution using a histogram with KDE.
6. Bivariate Analysis:
   - Compare survival rates across genders.
   - Compare survival by passenger class.
   - Use boxplot to visualize relationship between age and survival.
7. Correlation Analysis:
   - Compute numeric correlations using corr().
   - Display correlation heatmap using Seaborn's heatmap().
8. Interpret Insights:
   Derive key findings from the graphs and correlations.

PROGRAM:

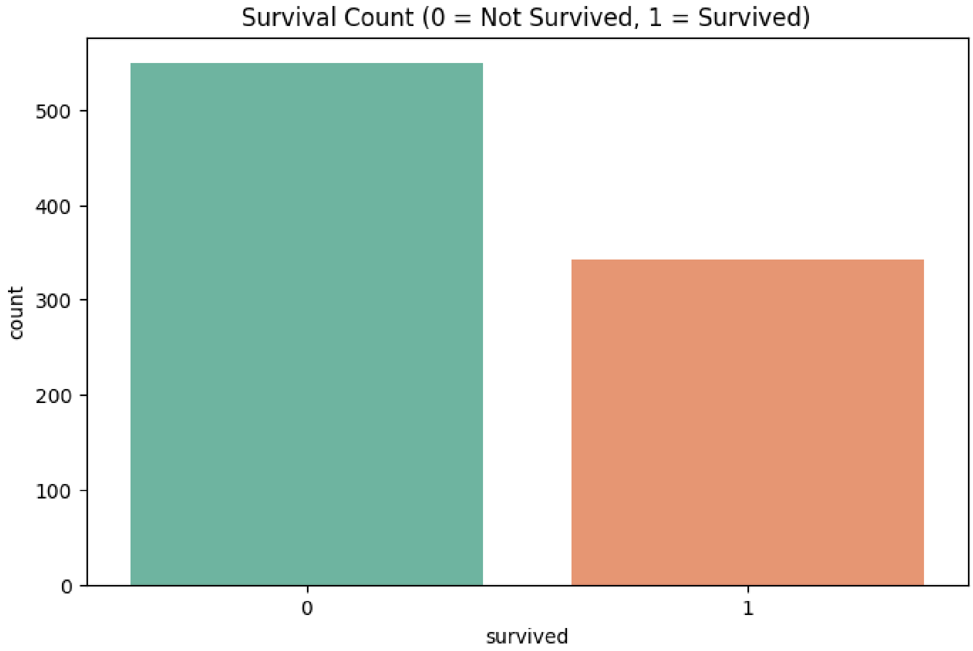
```
# Step 1: Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load Titanic dataset
df = sns.load_dataset('titanic')

# Step 3: Display basic information
print("===== Dataset Information =====")
print(df.info())

print("\n===== First 5 Rows =====")
print(df.head())

print("\n===== Summary Statistics =====")
print(df.describe(include='all'))

# Step 4: Check for missing values
print("\n===== Missing Values =====")
print(df.isnull().sum())

# Step 5: Univariate Analysis
plt.figure(figsize=(8,5))
sns.countplot(x='survived', data=df, palette='Set2')
plt.title("Survival Count (0 = Not Survived, 1 = Survived)")
plt.show()

plt.figure(figsize=(8,5))
sns.countplot(x='pclass', data=df, palette='Set3')
plt.title("Passenger Class Distribution")
plt.show()

plt.figure(figsize=(8,5))
sns.histplot(df['age'], kde=True, color='teal')
plt.title("Age Distribution of Passengers")
plt.show()

# Step 6: Bivariate Analysis
plt.figure(figsize=(8,5))
sns.countplot(x='sex', hue='survived', data=df, palette='coolwarm')
plt.title("Survival Count by Gender")
plt.show()

plt.figure(figsize=(8,5))
sns.countplot(x='pclass', hue='survived', data=df, palette='coolwarm')
plt.title("Survival Count by Passenger Class")
plt.show()

plt.figure(figsize=(8,5))
sns.boxplot(x='survived', y='age', data=df, palette='Set2')
plt.title("Age vs Survival")
plt.show()

# Step 7: Correlation and Heatmap
corr = df.corr(numeric_only=True)
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap="YlGnBu")
plt.title("Correlation Heatmap")
plt.show()

# Step 8: Insights
print("\n===== Key Insights =====")
print("1. Around 38% of passengers survived the Titanic disaster.")
print("2. Females had a significantly higher survival rate than males.")
print("3. Passengers in 1st class had a much higher chance of survival.")
print("4. Younger passengers (especially children) had higher survival chances.")
print("5. There are missing values in 'age', 'deck', and 'embark_town' columns.")
```

OUTPUT:

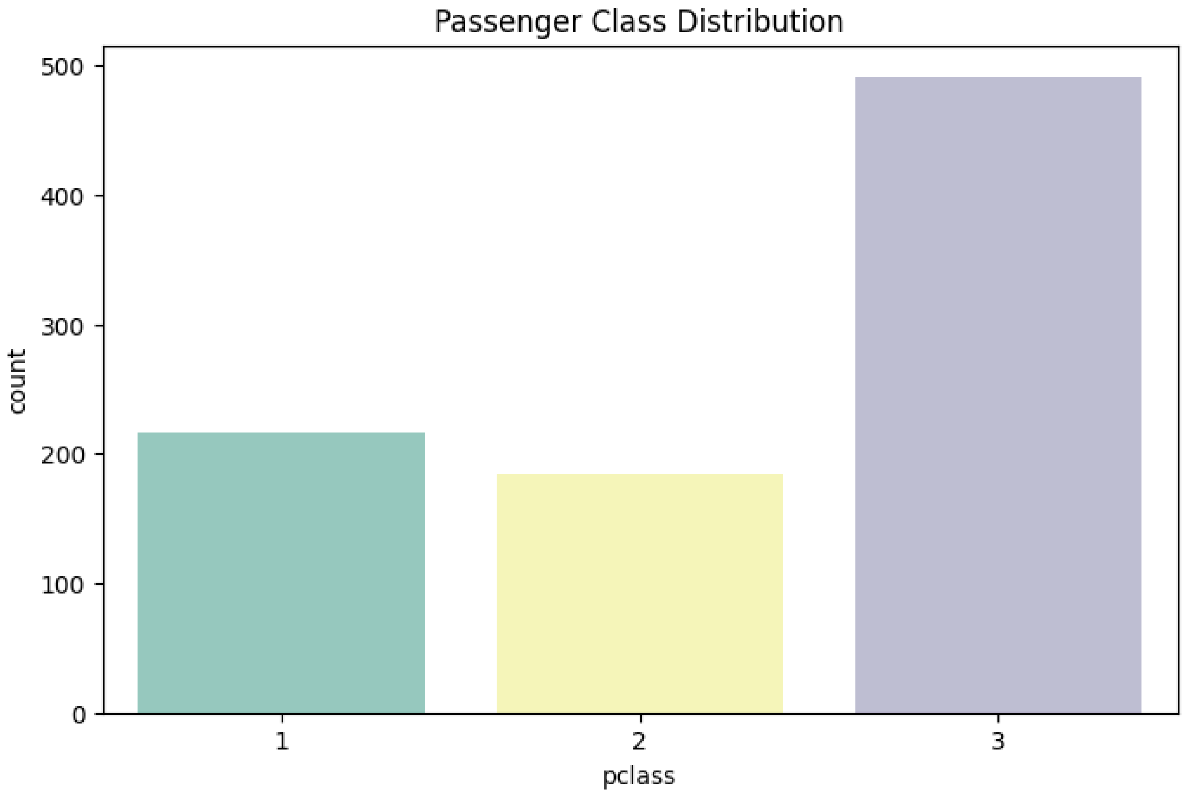
```
===== Dataset Information =====
<Class 'Pandas.Core.Frame.Dataframe'>
Rangeindex: 891 Entries, 0 To 890
Data Columns (Total 15 Columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Survived     891 Non-Null    Int64
 1   Pclass       891 Non-Null    Int64
 2   Sex          891 Non-Null    Object
 3   Age          714 Non-Null    Float64
 4   Sibsp        891 Non-Null    Int64
 5   Parch        891 Non-Null    Int64
 6   Fare         891 Non-Null    Float64
 7   Embarked     889 Non-Null    Object
 8   Class        891 Non-Null    Category
 9   Who          891 Non-Null    Object
 10  Adult_male   891 Non-Null    Bool
 11  Deck         203 Non-Null    Category
 12  Embark_town  889 Non-Null    Object
 13  Alive        891 Non-Null    Object
 14  Alone        891 Non-Null    Bool
Dtypes: Bool(2), Category(2), Float64(2), Int64(4), Object(5)
Memory Usage: 80.7+ Kb
None

===== First 5 Rows =====
   Survived  Pclass     Sex   Age  Sibsp  Parch     Fare Embarked  Class  \
0         0       3    Male  22.0      1      0   7.2500        S  Third
1         1       1  Female  38.0      1      0  71.2833        C  First
2         1       3  Female  26.0      0      0   7.9250        S  Third
3         1       1  Female  35.0      1      0  53.1000        S  First
4         0       3    Male  35.0      0      0   8.0500        S  Third

     Who  Adult_male Deck  Embark_town Alive  Alone
0    Man        True  Nan  Southampton    No  False
1  Woman       False    C    Cherbourg   Yes  False
2  Woman       False  Nan  Southampton   Yes   True
3  Woman       False    C  Southampton   Yes  False
4    Man        True  Nan  Southampton    No   True

===== Summary Statistics =====
          Survived     Pclass   Sex        Age      Sibsp      Parch  \
Count   891.000000  891.000000   891  714.000000  891.000000  891.000000
Unique        Nan        Nan     2        Nan        Nan        Nan
Top           Nan        Nan  Male        Nan        Nan        Nan
Freq          Nan        Nan   577        Nan        Nan        Nan
Mean     0.383838    2.308642   Nan   29.699118    0.523008    0.381594
Std      0.486592    0.836071   Nan   14.526497    1.102743    0.806057
Min      0.000000    1.000000   Nan    0.420000    0.000000    0.000000
25%      0.000000    2.000000   Nan   20.125000    0.000000    0.000000
50%      0.000000    3.000000   Nan   28.000000    0.000000    0.000000
75%      1.000000    3.000000   Nan   38.000000    1.000000    0.000000
Max      1.000000    3.000000   Nan   80.000000    8.000000    6.000000

              Fare Embarked  Class  Who Adult_male Deck  Embark_town Alive  \
Count   891.000000      889    891  891        891  203          889   891
Unique        Nan         3      3    3          2    7            3     2
Top           Nan         S  Third  Man       True    C  Southampton    No
Freq          Nan       644    491  537        537   59          644   549
Mean     32.204208      Nan    Nan  Nan        Nan  Nan          Nan   Nan
Std      49.693429      Nan    Nan  Nan        Nan  Nan          Nan   Nan
Min       0.000000      Nan    Nan  Nan        Nan  Nan          Nan   Nan
25%       7.910400      Nan    Nan  Nan        Nan  Nan          Nan   Nan
50%      14.454200      Nan    Nan  Nan        Nan  Nan          Nan   Nan
75%      31.000000      Nan    Nan  Nan        Nan  Nan          Nan   Nan
Max     512.329200      Nan    Nan  Nan        Nan  Nan          Nan   Nan

       Alone
Count    891
Unique     2
Top     True
Freq     537
Mean     Nan
Std      Nan
Min      Nan
25%      Nan
50%      Nan
75%      Nan
Max      Nan

===== Missing Values =====
Survived        0
Pclass          0
Sex             0
Age           177
Sibsp           0
```

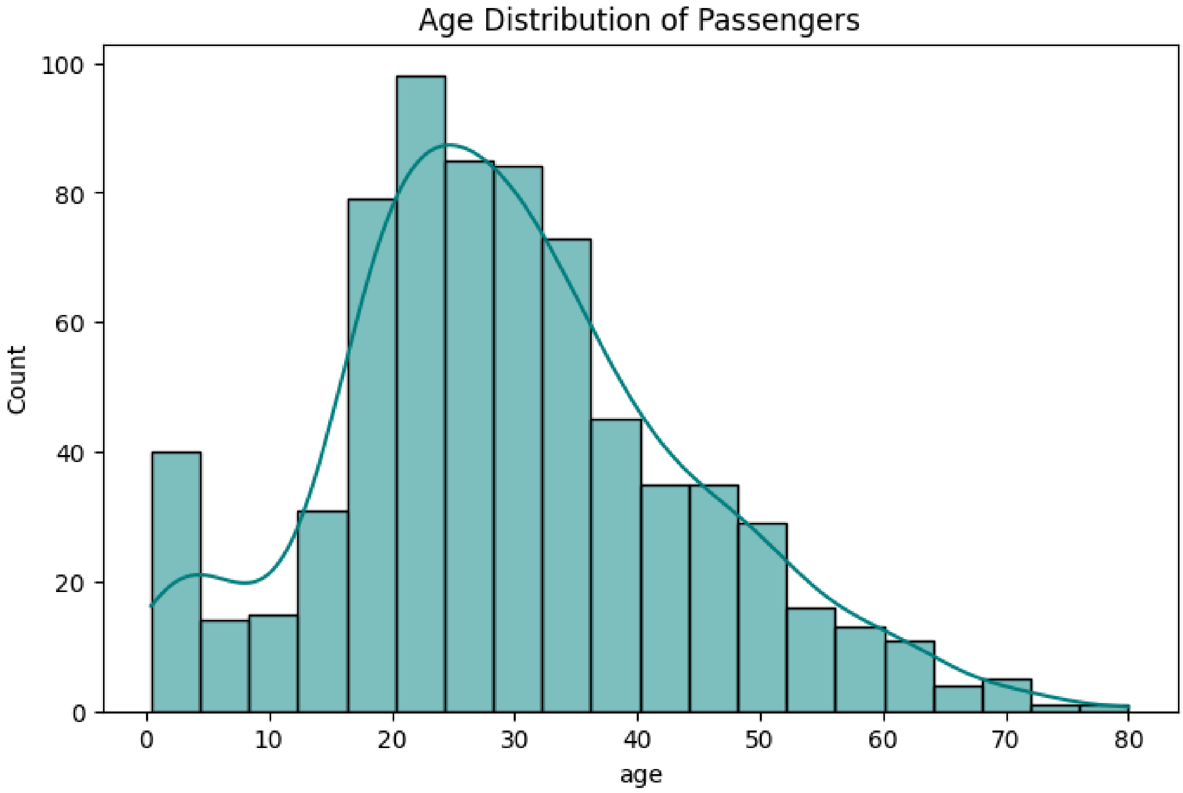
```
Parch           0
Fare            0
Embarked        2
Class           0
Who             0
Adult_male      0
Deck          688
Embark_town     2
Alive           0
Alone           0
Dtype: Int64
```

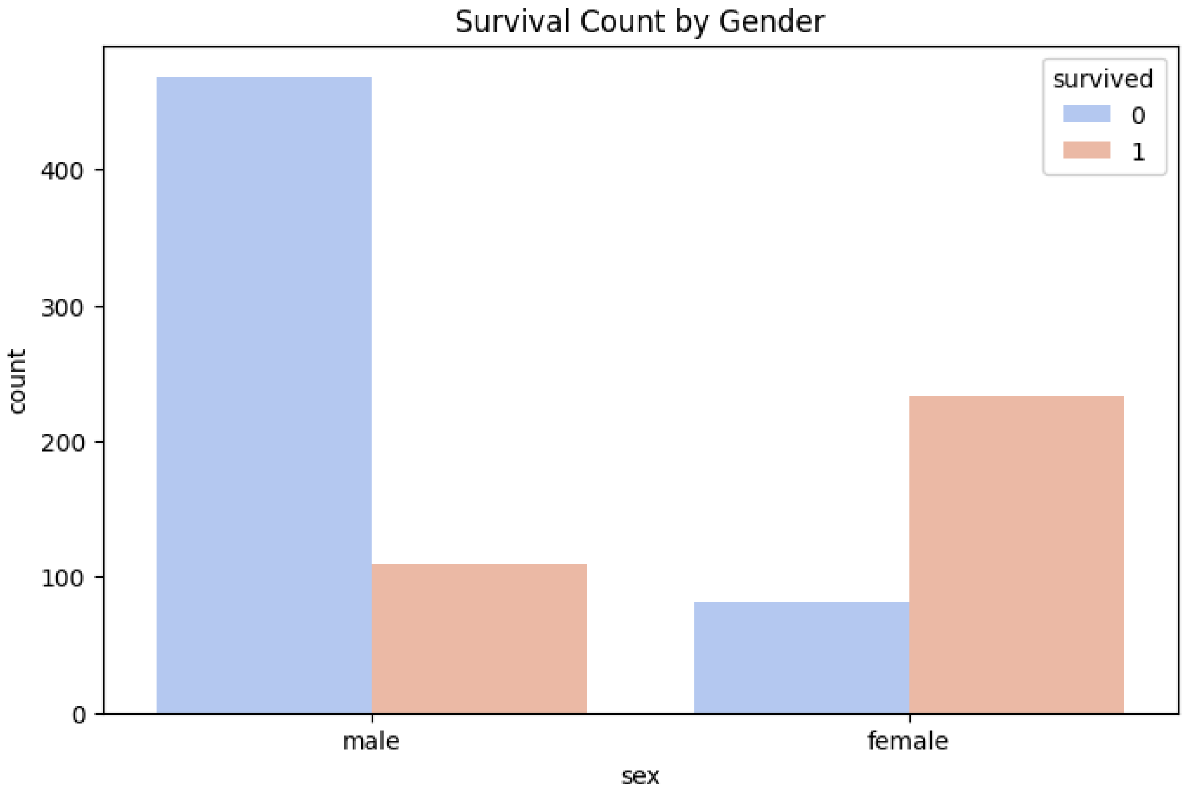
```
/Tmp/Ipython-Input-2046847747.Py:25: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Countplot(X='Survived', Data=Df, Palette='Set2')
```

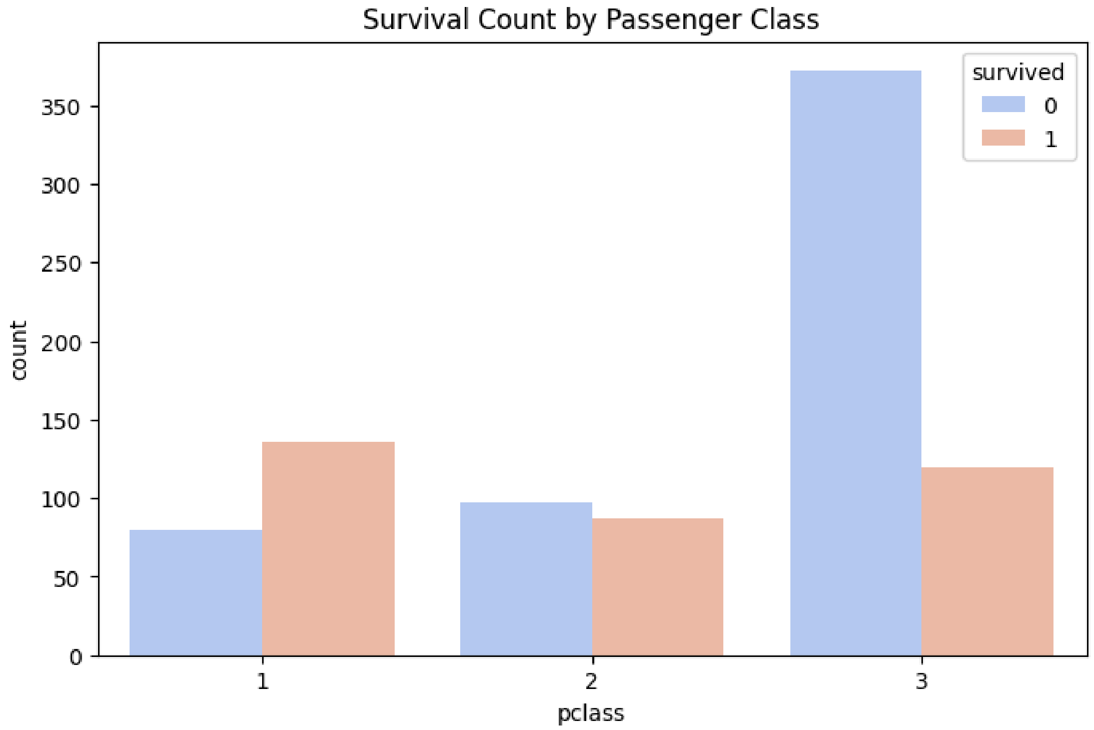


```
/Tmp/Ipython-Input-2046847747.Py:30: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Countplot(X='Pclass', Data=Df, Palette='Set3')
```

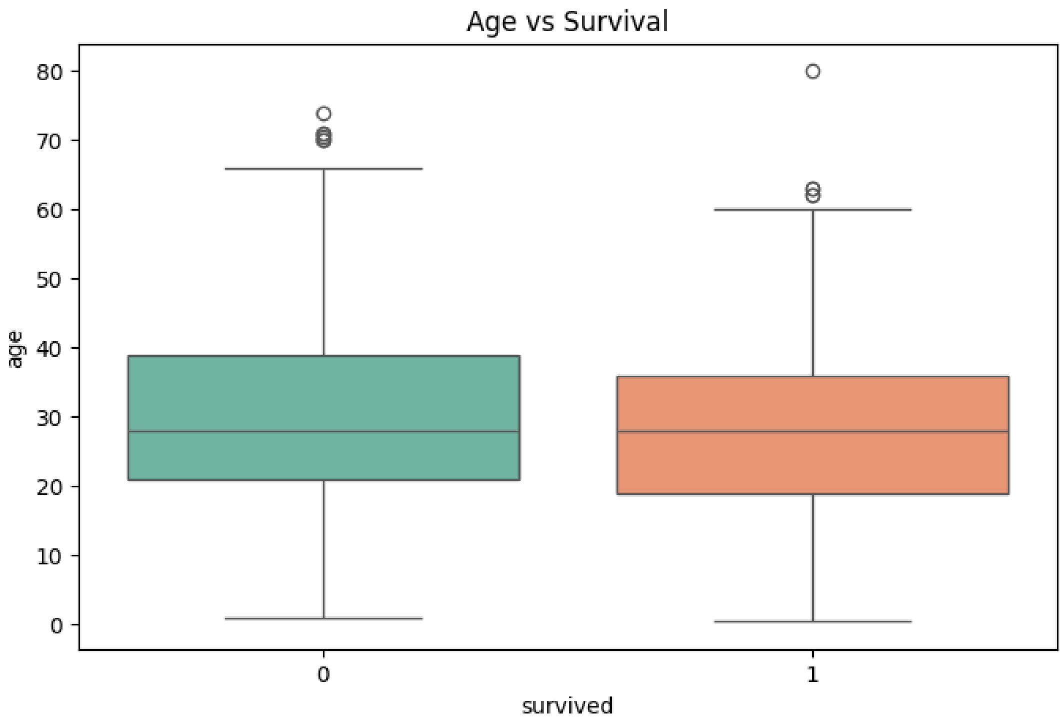

Passenger Class Distribution
500
400
300
200
100
0
count
1
2
3
pclass

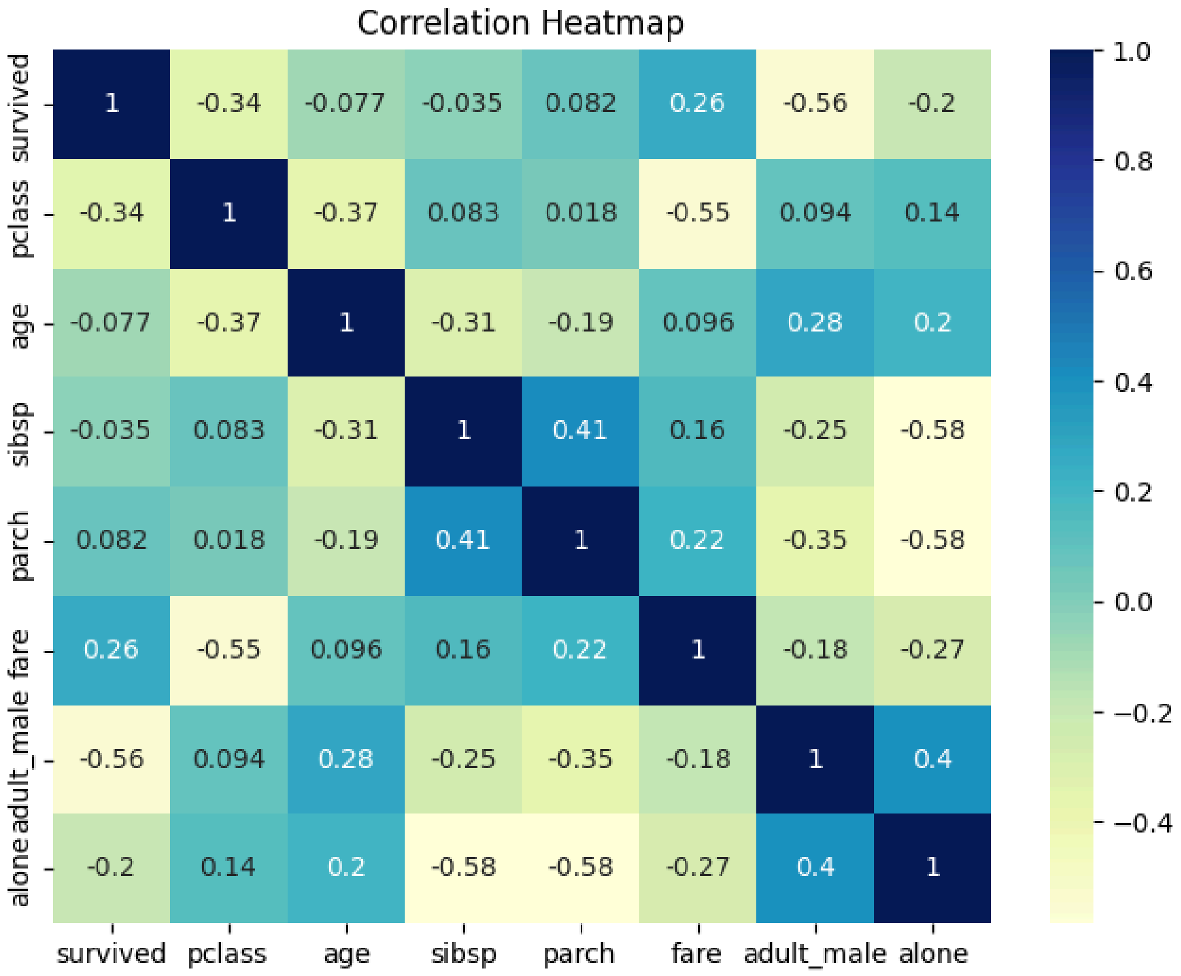

Age Distribution of Passengers
100
80
60
40
20
0
Count
0
10
20
30
40
50
60
70
80
age





```
/Tmp/Ipython-Input-2046847747.Py:51: Futurewarning:

Passing `Palette` Without Assigning `Hue` Is Deprecated And Will Be Removed In V0.14.0.
Assign The `X` Variable To `Hue` And Set `Legend=False` For The Same Effect.

  Sns.Boxplot(X='Survived', Y='Age', Data=Df, Palette='Set2')
```





===== Key Insights =====

1. Around 38% Of Passengers Survived The Titanic Disaster.
2. Females Had A Significantly Higher Survival Rate Than Males.
3. Passengers In 1st Class Had A Much Higher Chance Of Survival.
4. Younger Passengers (Especially Children) Had Higher Survival Chances.
5. There Are Missing Values In 'Age', 'Deck', And 'Embark_town' Columns.

RESULT:

- The Titanic dataset was successfully loaded and analyzed.
- Visualizations revealed the following key insights:
    1. Around 38% of passengers survived.
    2. Females had a much higher survival rate compared to males.
    3. 1st class passengers had the highest chance of survival.
    4. Younger passengers, especially children, were more likely to survive.
    5. Columns like ‘age’, ‘deck’, and ‘embark_town’ contain missing values.