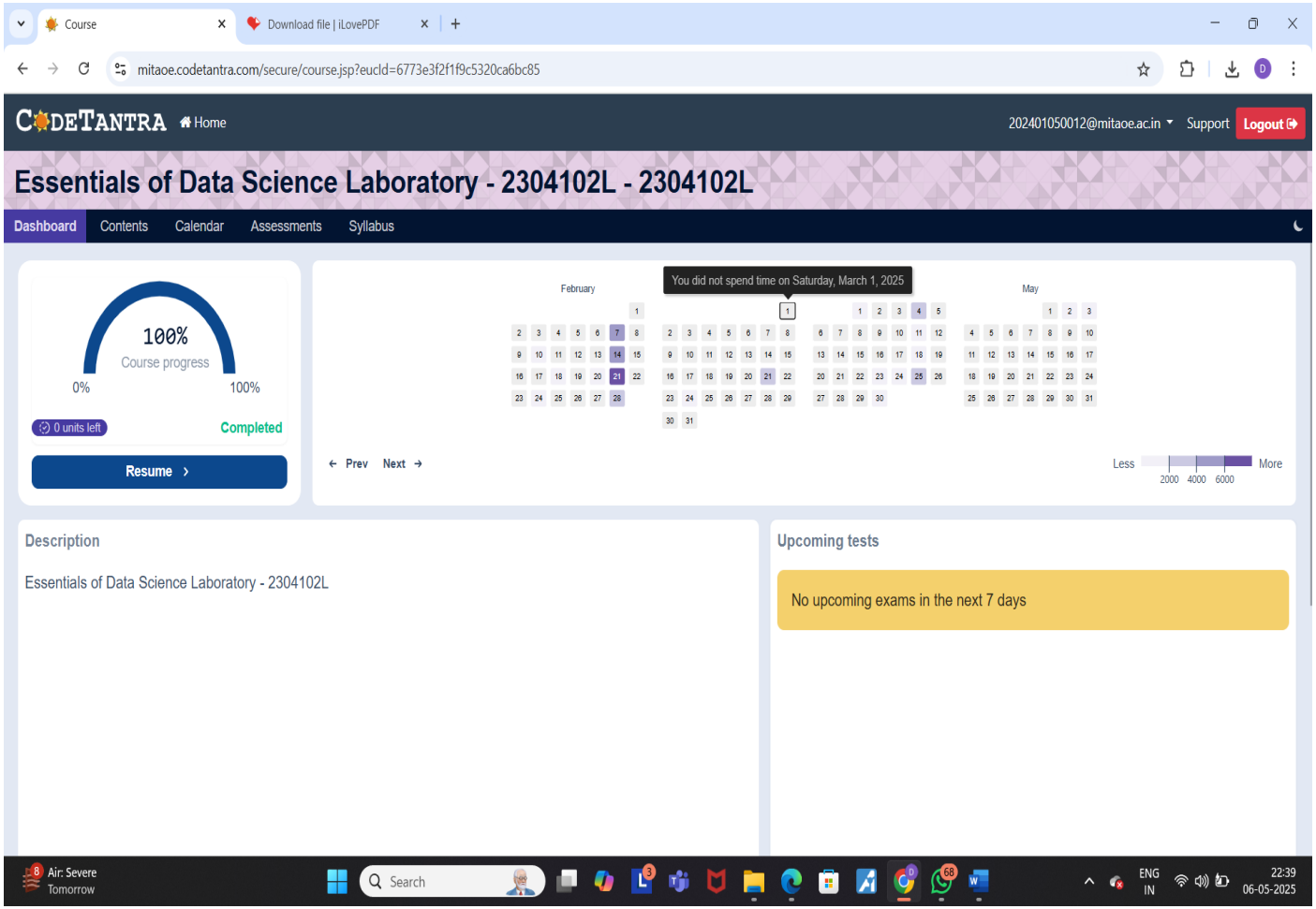

NAME: ROHUT CHANDRAMANI CHIKANE

CC-3 CC-53

202401050073

PRACTICAL 1

Practice Lab Assignment

1.1.1. Calculate Momentum

Write a program that accepts the mass of an object (in kilograms) and its velocity (in meters per second), then calculates and displays the momentum of the object. The momentum p is calculated using the formula:

$p=m\times v$

where:

m is the mass of the object (in kilograms).

v is the velocity of the object (in meters per second).

CODE :

```
m =float(input())
v=float(input())
p=float(m*v)
print(f"{p:.2f}kgm/s")
```

1.1.2. Conditional Calculation Based on the Number of Digits

Write a Python program that accepts an integer n as input. Depending on the number of digits in n.

CODE:

```
import math
n=float(input(""))
if(1<=n<10):
        a=int(n**2)
        print(a)
elif(10<=n<100):
```

```
        print(f"{math.sqrt(n):.2f}")
elif(100<n<1000):
        print(f"{math.cbrt(n):.2f}")
else:
        print("Invalid")
```

1.1.3. Age and Salary Calculation

Write a Python program that reads the birth date and salary of employees.

CODE:

```
def calculate_age(birthdate):
        a=birthdate[6:]
        ag=int(a)
        age= 2024-ag
        return(age)
def convert_salary_to_dollars(salary_in_rupees):
        s=salary_in_rupees*0.012
        return(s)
birthdate = input()
salary_in_rupees = float(input())
age = calculate_age(birthdate)
salary_in_dollars = convert_salary_to_dollars(salary_in_rupees)
print(f"Age: {age}")
print(f"Salary in dollars: {salary_in_dollars:.2f}")
```

1.1.4. Reverse a Number

You are given an integer number. Your task is to reverse the digits of the number and print the reversed number.

Code:

```
num = int(input())
reversed_num = int(str(abs(num))[::-1])
```

```
if num<0:
        reversed_num = -reversed_num
print(reversed_num)
```

1.1.5. Multiplication Table

Write a Python program that takes an integer as input and prints the multiplication table for that integer from 1 to 10.

Code:

```
num = int(input())
for i in range(1,11):
        print(f"{num} x {i} = {num*i}")
```

Lab Assignment

1.2.1. Pass or Fail

Write a Python program that accepts the number of courses and the marks of a student in those courses.

The grade is determined based on the aggregate percentage:

- **If the aggregate percentage is greater than 75, the grade is Distinction.**
- **If the aggregate percentage is greater than or equal to 60 but less than 75, the grade is First Division.**
- **If the aggregate percentage is greater than or equal to 50 but less than 60, the grade is Second Division.**
- **If the aggregate percentage is greater than or equal to 40 but less than 50, the grade is Third Division.**

CODE:

```
n = int(input())
marks = list(map(int,input().split()))
if any(mark<40 for mark in marks):
        print("Fail")
else:
        aggregate=sum(marks)/n
        print("Aggregate Percentage:",f"{aggregate:.2f}")
```

```
    if (aggregate>75):
        print('Grade: Distinction')
    elif(aggregate>=60 and aggregate<75):
        print("Grade: First Division")
    elif(aggregate>=50 and aggregate<60):
        print("Grade: Second Division")
    elif(aggregate>=400 and aggregate<50):
        print("Grade: Third Division")
```

1.2.2. Fibonacci series using Recursive Function

Write a Python program to find the Fibonacci series of a given number of terms using recursive function calls.

CODE:

```
def fib(n):
    if n<=1:
        return n
    elif n<0:
        return 0
    else:
        return fib(n-1)+fib(n-2)



    n=int(input("Enter terms for Fibonacci series: "))
for i in range (n):
    print(fib(i),end=" ")
```

1.2.3. Pattern - 1

Write a Python program to print a pattern of asterisks in the form of a right-angled triangle.

CODE:

```
r=int(input())
for i in range(1,r+1):
        print("* "*i)
```

1.2.4. Pattern - 2

Write a Python program to print a right-angled triangle pattern of numbers.

Code:

```
n=int(input())
for i in range(1, n+1):
        for j in range(1,i+1):
                print(j , end=" ")
        print( )
```

Practical 2

Practice Lab Assignment

2.1.1. List operations

Write a Python program that implements a menu-driven interface for managing a list of integers. The program should have the following menu options:

1. Add

2. Remove

3. Display

4. Quit

The program should repeatedly prompt the user to enter a choice from the menu. Depending on the choice selected, the program should perform the following actions:

- Add: Prompts the user to enter an integer and add it to the integer list. If the input is not a valid integer, display "Invalid input".
- Remove: Prompts the user to enter an integer to remove from the list. If the integer is found in the list, remove it; otherwise, display "Element not found". If the list is empty, display "List is empty".
- Display: Displays the current list of integers. If the list is empty, display "List is empty".
- Quit: Exits the program.

- The program should handle invalid menu choices by displaying "Invalid choice".Ensure that the program continues to prompt the user until they choose to quit (option 4).

CODE:

```
int_list = []


while True:
        print("1. Add")
        print("2. Remove")
        print("3. Display")
        print("4. Quit")
        choice = input("Enter choice: ")
        if choice =='1':
                element_input = input("Integer: ")
                if element_input.isdigit():
                        element= int(element_input)
                        int_list.append(element)
                        print("List after adding:",int_list)
                else:
                        print("Invalid Input")
        elif choice=='2':
                if int_list:
                        element_input = input("Integer: ")
                        if element_input.isdigit():
                                element = int(element_input)
                                if element in int_list:
                                        int_list.remove(element)
                                        print("List after removing:",int_list)
                                else:
```

```
                print("Element not found")
            else:
                print("Invalid input")
        else:
            print("List is empty")
    elif choice =='3':
        if int_list:
            print(int_list)
        else:
            print("List is empty")
    elif choice == '4':
        break
    else:
        print("Invalid choice")
```

2.1.2. Dictionary Operations

Write a Python program to perform the following dictionary operations:

- Create an empty dictionary and display it.
- Ask the user how many items to add, then input key-value pairs.
- Show the dictionary after adding items.
- Ask the user to update a key's value. Print "Value updated" if the key exists, otherwise print "Key not found".
- Retrieve and print a value using a key. If not found, print "Key not found".
- Use get() to retrieve a value. If the key doesn't exist, print "Key not found".
- Delete a key-value pair. If the key exists, delete and print "Deleted". If not, print "Key not found".
- Display the updated dictionary.

Code:

```
my_dict = {}

print("Empty Dictionary:", my_dict)
```

```
num_items = int(input("Number of items: "))
for _ in range(num_items):
        key = input("key: ")
        value = input("value: ")
        my_dict[key] = value
print("Dictionary:", my_dict)
key_to_update = input("Enter the key to update: ")
if key_to_update in my_dict:
        new_value = input("Enter the new value: ")
        my_dict[key_to_update] = new_value
        print("Value updated")
else:
        print("key not found")

key_to_retrieve = input("Enter the key to retrieve: ")
if key_to_retrieve in my_dict:
        print(f"Key: {key_to_retrieve}, Value: {my_dict[key_to_retrieve]}")
else:
        print("Key not found")

key_to_get = input("Enter the key to get using the get() method: ")
value = my_dict.get(key_to_get)
if value is not None:
        print(f"Key: {key_to_get}, Value: {value}")
else:
        print("Key not found")

key_to_delete = input("Enter the key to delete: ")
```

```
if key_to_delete in my_dict:
        del my_dict[key_to_delete]
        print("Deleted")
else:
        print("Key not found")
print("Updated Dictionary:", my_dict)
```

Lab Assignment

2.2.1. Linear search Technique

Write a program to check whether the given element is present or not in the array of elements using linear search.

CODE:

```
arr = list(map(int,input().split()))
key = int(input())
found =False
for i in range(len(arr)):
        if arr[i] == key:
                print(i)
                found = True
                break
if not found:
                print("Not found")
```

2.2.2. Captain of the Team

You are provided with the heights of 11 cricket players (in centimeters). Your task is to identify the tallest player, who will be selected as the captain of the team.

Code:

```
height = list(map(int,input().split()))
tallest = 0
for height in height:
        if height> tallest:
```

```
            tallest = height
print(tallest)
```

Practical 3

Practice Lab Assignment

3.1.1. Numpy array operations

Write a python program to demonstrate the usage of ndim, shape and size for a Numpy Array. The program should create a NumPy array using the entered elements and display it. Assume all input elements are valid numeric values.

CODE:

```
import numpy as np
rows,cols = map(int,input().split())
elements = []
for _ in range(rows):
        elements.extend(map(int,input().split()))
array = np.array(elements).reshape(rows,cols)
print(array)
print(array.ndim)
print(array.shape)
print(array.size)
```

Lab Assignment

3.2.1. Numpy: Matrix Operations

The given code takes two 3×3 matrices, matrix_a, and matrix_b, as input from the user and converts them into NumPy arrays.

Task:

You are required to compute and display the results of the following matrix operations:

1. **Addition (matrix_a + matrix_b)**
2. **Subtraction (matrix_a – matrix_b)**
3. **Element-wise Multiplication (matrix_a * matrix_b)**

4. Matrix Multiplication (matrix_a · matrix_b)
5. Transpose of Matrix A

CODE:

```
import numpy as np


# Input matrices
print("Enter Matrix A:")
matrix_a = np.array([list(map(int, input().split())) for i in range(3)])


print("Enter Matrix B:")
matrix_b = np.array([list(map(int, input().split())) for i in range(3)])



# Addition
addition = matrix_a + matrix_b
print("Addition (A + B):")
print(addition)
# Subtraction
subtraction = matrix_a - matrix_b
print("Subtraction (A - B):")
print(subtraction)
# Multiplication (element-wise)
multiplication = matrix_a*matrix_b
print("Element-wise Multiplication (A * B):")
print(multiplication)


# Matrix multiplication (dot product)
matrix_multiplication = np.dot(matrix_a,matrix_b)
```

```
print("A dot B:")

print(matrix_multiplication)

# Transpose

transpose_a = matrix_a.T

print("Transpose of A:")

print(transpose_a)
```

3.2.2. Numpy: Horizontal and Vertical Stacking of Arrays

You are given two arrays arr1 and arr2. You need to perform horizontal and vertical stacking operations on them using NumPy.

- **Horizontal Stacking: Stack the two matrices horizontally (side by side).**
- **Vertical Stacking: Stack the two matrices vertically (one below the other).**

CODE:

```
import numpy as np


# Input matrices

print("Enter Array1:")

arr1 = np.array([list(map(int, input().split())) for i in range(3)])


print("Enter Array2:")

arr2 = np.array([list(map(int, input().split())) for i in range(3)])

# Perform horizontal stacking (hstack)

hstack_result = np.hstack((arr1,arr2))

print("Horizontal Stack:")

print(hstack_result)

# Perform vertical stacking (vstack)

vstack_result = np.vstack((arr1,arr2))

print("Vertical Stack:")

print(vstack_result)
```

3.2.3. Numpy: Custom Sequence Generation

Write a Python program that takes the following inputs from the user:

- Start value: The starting point of the sequence.
- Stop value: The sequence should end before this value.
- Step value: The increment between each number in the sequence.

The program should then generate a sequence using numpy based on these inputs and print the generated sequence.

CODE:

```
import numpy as np


# Take user input for the start, stop, and step of the sequence

start = int(input())

stop = int(input())

step = int(input())


# Generate the sequence using np.arange()

sequence = np.arange(start,stop,step)

# Print the generated sequence

print(sequence)
```

3.2.4. Numpy: Arithmetic and Statistical Operations, Mathematical Operations, Bitwise Operators

You ae given two arrays A and B. Your task is to complete the function array_operations, which will convert these lists into NumPy arrays and perform the following operations:

CODE:

```
import numpy as np


def array_operations(A, B):


    # Convert A and B to NumPy arrays

    A = np.array(A)
```

```
B = np.array(B)

# Arithmetic Operations
sum_result = A+B
diff_result = A-B
prod_result = A*B

# Statistical Operations
mean_A = np.mean(A)
median_A = np.median(A)
std_dev_A = np.std(A)

# Bitwise Operations
and_result = A&B
or_result = A|B
xor_result = A^B

# Output results with one space between each element
print("Element-wise Sum:", ' '.join(map(str, sum_result)))
print("Element-wise Difference:", ' '.join(map(str, diff_result)))
print("Element-wise Product:", ' '.join(map(str, prod_result)))

print(f"Mean of A: {mean_A}")
print(f"Median of A: {median_A}")
print(f"Standard Deviation of A: {std_dev_A}")

print("Bitwise AND:", ' '.join(map(str, and_result)))
print("Bitwise OR:", ' '.join(map(str, or_result)))
```

```
        print("Bitwise XOR:", ' '.join(map(str, xor_result)))


A = list(map(int, input().split()))  # Elements of array A
B = list(map(int, input().split()))  # Elements of array B
array_operations(A, B)
```

3.2.5. Numpy: Copying and Viewing Arrays

The given code takes a list of integers as input and converts it into a NumPy array. Your task is to complete the code by:

- Creating a view of the original_array and assigning it to view_array.
- Creating a copy of the original_array and assigning it to copy_array.

After completing these steps, observe how modifying the view affects the original_array, while modifying the copy does not.

CODE:

```
import numpy as np


inputlist = list(map(int,input().split(" ")))


# Original array
original_array = np.array(inputlist)


# Create a view
view_array = original_array.view()


# Create a copy
copy_array = original_array.copy()


# Modify the view
view_array[0] = 99
```

```
print("Original array after modifying view:", original_array)

print("View array:", view_array)


# Modify the copy

copy_array[1] = 88

print("Original array after modifying copy:", original_array)

print("Copy array:", copy_array)
```

3.2.6. Numpy: Searching, Sorting, Counting, Broadcasting

The given code in the editor takes a single array, array1, as space-separated integers as input from the user.

Additionally, it takes the following inputs:

- **search_value: The value to search for in the array.**
- **count_value: The value to count its occurrences in the array.**
- **broadcast_value: The value to add for broadcasting across the array.**

You need to complete the code to perform the following operations:

1. Searching: Find the indices where search_value appears in array1 and print these indices.

2. Counting: Count how many times count_value appears in array1 and print the count.

3. Broadcasting: Add broadcast_value to each element of array1 using broadcasting, and print the resulting array.

4. Sorting: Sort array1 in ascending order and print the sorted array.

CODE:

```
import numpy as np


# Input array from the user

array1 = np.array(list(map(int, input().split())))


# Searching

search_value = int(input("Value to search: "))
```

```
count_value = int(input("Value to count: "))

broadcast_value = int(input("Value to add: "))


# Find indices where value matches in array1

indices = np.where(array1 == search_value)[0]

print(indices)

# Count occurrences in array1

count_occurences = np.count_nonzero(array1 == count_value)

print(count_occurences)

# Broadcasting addition

array1_broadcasted = array1 + broadcast_value

print(array1_broadcasted)


# Sort the first array

sorted_array = np.sort(array1)

print(sorted_array)
```

3.2.7. Student Data Analysis and Operations

Write a Python program that takes the file name of a CSV file containing student details, including roll numbers and their marks in three subjects as input, reads the data, and performs the following operations:

- Print all student details: Display the complete details of all students, including roll numbers and marks for all subjects.
- Find total students: Determine the total number of students in the dataset.
- Print all student roll numbers: Extract and print the roll numbers of all students.
- Print Subject 1 marks: Extract and print the marks of all students in Subject 1.
- Find minimum marks in Subject 2: Identify the lowest marks in Subject 2.
- Find maximum marks in Subject 3: Identify the highest marks in Subject 3.
- Print all subject marks: Display the marks of all students for each subject.
- Find total marks of students: Compute the total marks for each student across all subjects.
- Find the average marks of each student: Compute the average marks for each student.

- Find average marks of each subject: Compute the average marks for all students in each subject.
- Find average marks of Subject 1 and Subject 2: Compute the average marks for Subject 1 and Subject 2.
- Find average marks of Subject 1 and Subject 3: Compute the average marks for Subject 1 and Subject 3.
- Find the roll number of the student with maximum marks in Subject 3: Identify the student with the highest marks in Subject 3 and print their roll number.
- Find the roll number of the student with minimum marks in Subject 2: Identify the student with the lowest marks in Subject 2 and print their roll number.
- Find the roll number of students who scored 24 marks in Subject 2: Identify students who obtained exactly 24 marks in Subject 2 and print their roll numbers.
- Find the count of students who got less than 40 marks in Subject 1: Count the number of students who scored less than 40 marks in Subject 1.
- Find the count of students who got more than 90 marks in Subject 2: Count the number of students who scored more than 90 marks in Subject 2.
- Find the count of students who scored >=90 in each subject: Count the number of students who scored 90 or more marks in each subject.
- Find the count of subjects in which each student scored >=90: Determine how many subjects each student scored 90 or more marks in.
- Print Subject 1 marks in ascending order: Sort and print the marks of students in Subject 1 in ascending order.
- Print students who scored between 50 and 90 in Subject 1: Display students who scored marks between 50 and 90 in Subject 1.
- Find index positions of students who scored 79 in Subject 1: Identify the index positions of students who scored exactly 79 marks in Subject 1.

CODE:

```
import numpy as np


a = np.loadtxt("Sample.csv", delimiter=',', skiprows=1)

# 1. Print all student details

print("All student Details:\n",a)
```

```
# 2. print total students
r,c=a.shape
print("Total Students:",r)

# 3. Print all student Roll numbers
print("All Student Roll Nos",a[:,0])

# 4. Print subject 1 marks
print("Subject 1 Marks",a[:,1])

# 5. print minimum marks of Subject 2
print("Min marks in Subject 2",np.min(a[:,2]))

# 6. print maximum marks of Subject 3
print("Max marks in Subject 3",np.max(a[:,3]))

# 7. Print All subject marks
print("All subject marks:",a[:,1:])

# 8. print Total marks of students
print("Total Marks",np.sum(a[:,1:],axis=1))

# 9. print average marks of each student
avg=np.mean(a[:,1:],axis=1)
print(np.round(avg,1))
# 10. print average marks of each subject
print("Average Marks of each subject",np.mean(a[:,1:],axis=0))
```

```
# 11. print average marks of S1 and S2
print("Average Marks of S1 and S2",np.mean(a[:,1:3],axis=0))

# 12. print average marks of S1 and S3
print("Average Marks of S1 and S3",np.mean(a[:,[1,3]],axis=0))

# 13. print Roll number who got maximum marks in Subject 3
i=np.argmax(a[:,3])
print("Roll no who got maximum marks in Subject 3",a[i,0])

# 14. print Roll number who got minimum marks in Subject 2
mn=np.argmin(a[:,2])
print("Roll no who got minimum marks in Subject 2",a[mn,0])

# 15. print Roll number who got 24 marks in Subject 2
whr=np.where(a[:,2]==24)
print("Roll no who got 24 marks in Subject 2",a[whr,0])

# 16. print count of students who got marks in Subject 1 < 40
ct=np.count_nonzero(a[:,1]<40)
print("Count of students who got marks in Subject 1 < 40",ct)

# 17. print count of students who got marks in Subject 2 > 90
ct1=np.count_nonzero(a[:,2]>90)
print("Count of students who got marks in Subject 2 > 90:",ct1)

# 18. print count of students in each subject who got marks >= 90
print("Count of students in each subject who got marks >= 90:",np.count_nonzero(a[:,1:]>90,axis=0))
```

```
# 19. print count of subjects in which each student got marks >= 90

print("Roll no:",a[:,0])

print("Count of subjects in which student got marks >= 90:",np.count_nonzero(a[:,1:]>90,axis=1))


# 20. Print S1 marks in ascending order

srt=np.sort(a[:,1])

print(srt)



# 21. Print S1 marks >= 50 and <= 90

print(a[(a[:,1]>=50)& (a[:,1]<=90)])

# 22. Print the index position of marks 79

print(a)

ip=np.where(a[:,1]==79)

print(ip)
```

Practical 4

Practice Lab Assignment

4.1.1. Pandas - series creation and manipulation

Write a Python program that takes a list of numbers from the user, creates a Pandas series from it, and then calculates the mean of even and odd numbers separately using the groupby and mean() operations.

Code:

```
import pandas as pd


# Take inputs from the user to create a list of numbers

numbers = list(map(int, input().split()))


# Create a Pandas series from the list of numbers
```

```
series = pd.Series(numbers)

# Grouping by even and odd numbers and calculating the mean

grouped = series.groupby(series%2==0).mean()


# Display the mean of even and odd numbers with labels

grouped.index = ['Even' if is_even else 'Odd' for is_even in grouped.index]

print("Mean of even and odd numbers:")

print(grouped)
```

4.1.2. Dictionary to dataframe

A dictionary of lists has been provided to you in the editor. Create a DataFrame from the dictionary of lists and perform the listed operations, then display the DataFrame before and after each manipulation.

Code:

```
import pandas as pd


# Provided dictionary of lists

data = {

    'Name': ['Alice', 'Bob', 'Charlie'],

    'Age': [25, 30, 35],

}


# Convert the dictionary to a DataFrame

df = pd.DataFrame(data)


# Display the original DataFrame

print("Original DataFrame:")

print(df)


# Adding a new row
```

```
new_name=input("New name: ")
new_age=int(input("New age: "))
new_row={'Name': new_name, 'Age': new_age}
df=df.append(new_row,ignore_index= True)


# Display the DataFrame after adding a new row
print("After adding a row:\n",df)

# Modifying a row
modify_row_index = int(input("Index of row to modify: "))
new_ag_value = int(input("New age: "))
df.at[modify_row_index, 'Age'] = new_ag_value


# Display the DataFrame after modifying a row
print("After modifying a row:")
print(df)

# Deleting a row
delete_row_index = int(input("Index of row to delete: "))
df=df.drop(delete_row_index).reset_index(drop=True)
# Display the DataFrame after deleting a row
print("After deleting a row:")
print(df)

# Adding a new column
genders = input("Enter genders separated by space: ").split()
```

```
df['Gender']=genders


# Display the DataFrame after adding a new column

print("After adding a new column:")

print(df)


# Modifying a column

df['Name'] = df['Name'].str.upper()

# Display the DataFrame after modifying a column

print("After modifying a column:")

print(df)


# Deleting a column

df = df.drop(columns=['Age'])

# Display the DataFrame after deleting a column

print("After deleting a column:")

print(df)
```

4.1.3. Student Information

Write a program to read a text file containing student information (name, age, and grade) using Pandas. Perform the following tasks:

Code:

```
import pandas as pd


# Read the text file into a DataFrame

file = input()

data = pd.read_csv(file, sep="\s+", header=None, names=["Name", "Age", "Grade"])

print("First five rows:")

print(data.head())
```

```
average_age = round(data['Age'].mean(),2)

print(f"Average age: {average_age}")

filtered_student = data[data['Grade'].isin(["A","B"])]

print("Students with a grade up to B")

print(filtered_student)
```

Lab Assignment

4.2.1. Month with the Highest Total Sales

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- **The CSV file contains the columns: Date, Product, Quantity, Price, and City.**
- **Group the data by Month and calculate the total sales for each month.**
- **Find the month with the highest total sales and display it.**
- **Also, display the total sales for the best month.**

Code:

```
import pandas as pd


# Prompt the user for the file name

file_name = input()


# Load the data

df = pd.read_csv(file_name)

df['sales']=df['Quantity'].multiply(df['Price'])

df['month']=pd.to_datetime(df['Date']).dt.strftime("%Y-%m")

# Find the month with the highest total sales

best_month=df.groupby('month')['sales'].sum().idxmax()

highest_sales= df['sales'].sum()


print(f"Best month: {best_month}")
```

```
print(f"Total sales: ${highest_sales:.2f}")
```

4.2.2. Best Selling Product

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- **The CSV file contains the columns: Date, Product, Quantity, Price, and City.**
- **Find the product that sold the most in terms of quantity sold.**
- **Display the product that sold the most and the total quantity sold for that product.**

Code:

```
import pandas as pd


# Prompt the user for the file name

file_name = input()


# Load the data

df = pd.read_csv(file_name)

product_sales=df.groupby('Product')['Quantity'].sum()

# Find the product with the highest total quantity sold

best_product = product_sales.idxmax()

highest_quantity = product_sales.max()


# Display the result

print(f"Best selling product: {best_product}")

print(f"Total quantity sold: {highest_quantity}")
```

4.2.3. City that Sold the Most Products

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- **The CSV file contains the columns: Date, Product, Quantity, Price, and City.**
- **Group the data by City and calculate the total quantity of products sold for each city.**
- **Find the city that sold the most products (based on the total quantity sold).**

4.2.3. City that Sold the Most Products

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- **The CSV file contains the columns: Date, Product, Quantity, Price, and City.**
- **Group the data by City and calculate the total quantity of products sold for each city.**
- **Find the city that sold the most products (based on the total quantity sold).**

Code:

```
mport pandas as pd


# Prompt the user for the file name

file_name = input()


# Load the data

df = pd.read_csv(file_name)

city_sales=df.groupby('City')['Quantity'].sum()

best_city=city_sales.idxmax()


# Display the result

print(f"City sold the most products: {best_city}")
```

4.2.4. Most Frequently Sold Product Pairs

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- **The CSV file contains the following columns: Date, Product, Quantity, Price, and City.**
- **For each date, find all pairs of products that were sold together (i.e., two products sold on the same date).**
- **Output the product pair/s that was sold most frequently.**

Code:

```
import pandas as pd

from itertools import combinations
```

```
from collections import Counter


# Prompt user to input the file name

file_name = input()


# Read data from the specified CSV file

df = pd.read_csv(file_name)


grouped=df.groupby('Date')['Product'].apply(list)

pair_counter=Counter()

# Output the most frequent product pairs

for products in grouped:

        products=sorted(products)

        pairs=combinations(products,2)

        pair_counter.update(pairs)

max_count=max(pair_counter.values()) if pair_counter else 0

most_frequent_pairs=[pair for pair,count in pair_counter.items() if count==max_count]

most_frequent_pairs.sort()

for pair in  most_frequent_pairs:

        print(f"{pair[0]} and {pair[1]}: {max_count} times")



# Output the most frequent product pairs
```

4.2.5. Titanic Dataset Analysis and Data Cleaning

You are provided with the Titanic dataset containing information about passengers on the Titanic. Your task is to write Python code to answer the following questions based on the dataset. For each question, perform necessary data cleaning, transformations, and calculations as required.

Code:

```
import pandas as pd
```

```
import numpy as np

# Load the Titanic dataset
data = pd.read_csv('Titanic-Dataset.csv')

# 1. Display the first 5 rows of the dataset


# 2. Display the last 5 rows of the dataset


# 3. Get the shape of the dataset


# 4. Get a summary of the dataset (info)


# 5. Get basic statistics of the dataset


# 6. Check for missing values


# 7. Fill missing values in the 'Age' column with the median age


# 8. Fill missing values in the 'Embarked' column with the mode
```

```
# 9. Drop the 'Cabin' column due to many missing values



# 10. Create a new column 'FamilySize' by adding 'SibSp' and 'Parch'

# 1. Display the first 5 rows of the dataset

print(data.head(5))

# 2. Display the last 5 rows of the dataset

print(data.tail(5))

# 3. Get the shape of the dataset

print(data.shape)

# 4. Get a summary of the dataset (info)

data.info()

print("None")

# 5. Get basic statistics of the dataset

print(data.describe())

# 6. Check for missing values

print(data.isnull().sum())

print()

# 7. Fill missing values in the
```

4.2.6. Titanic Dataset Analysis and Data Cleaning - 2

You are provided with the Titanic dataset containing information about passengers on the Titanic. Your task is to write Python code to answer the following questions based on the dataset.

1. **Create a new column 'IsAlone' which is 1 if the passenger is alone (FamilySize = 0), otherwise 0.**
2. **Convert the 'Sex' column to numeric values (male: 0, female: 1).**
3. **One-hot encode the 'Embarked' column, dropping the first category.**
4. **Get the mean age of passengers.**
5. **Get the median fare of passengers.**

6. Get the number of passengers by class.
7. Get the number of passengers by gender.
8. Get the number of passengers by survival status.
9. Calculate the survival rate of passengers.
10. Calculate the survival rate by gender.

Code:

```
import pandas as pd
import numpy as np

# Load the Titanic dataset
data = pd.read_csv('Titanic-Dataset.csv')
data['FamilySize'] = data['SibSp'] + data['Parch']

# 1. Create a new column 'IsAlone' (1 if alone, 0 otherwise)
data['IsAlone']=(data['FamilySize']==0).astype(int)
# 2. Convert 'Sex' to numeric (male: 0, female: 1)
data['Sex']=data['Sex'].map({'male':0,'female':1})
# 3. One-hot encode the 'Embarked' column
embarked_dummies=pd.get_dummies(data['Embarked'],prefix='Embarked',drop_first=True)
data=pd.concat([data,embarked_dummies],axis=1)
# 4. Get the mean age of passengers
mean_age=data['Age'].mean()
print(mean_age)
# 5. Get the median fare of passengers
median_fare=data['Fare'].median()
print(median_fare)
# 6. Get the number of passengers by class
pclass_counts=data['Pclass'].value_counts().loc[[3,1,2]]
```

```
print(pclass_counts)

# 7. Get the number of passengers by gender

sex_counts=data['Sex'].value_counts().sort_index()

print(sex_counts)

# 8. Get the number of passengers by survival status

survived_counts=data['Survived'].value_counts().sort_index()

print(survived_counts)

# 9. Calculate the survival rate

survival_rate=data['Survived'].mean()

print(survival_rate)

# 10. Calculate the survival rate by gender

survival_by_gender=data.groupby('Sex')['Survived'].mean()

print(survival_by_gender)
```

4.2.7. Titanic Dataset Analysis and Data Cleaning - 3

You are provided with the Titanic dataset containing information about passengers on the Titanic. Your task is to write Python code to answer the following questions based on the dataset.

1. Calculate the survival rate by class.
2. Calculate the survival rate by embarkation location (Embarked_S).
3. Calculate the survival rate by family size (FamilySize).
4. Calculate the survival rate by being alone (IsAlone).
5. Get the average fare by passenger class (Pclass).
6. Get the average age by passenger class (Pclass).
7. Get the average age by survival status (Survived).
8. Get the average fare by survival status (Survived).
9. Get the number of survivors by class (Pclass).
10. Get the number of non-survivors by class (Pclass).

Code:

```
import pandas as pd

import numpy as np


# Load the Titanic dataset

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

data['FamilySize'] = data['SibSp'] + data['Parch']

data['IsAlone'] = np.where(data['FamilySize'] > 0, 0, 1)

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# 1. Calculate the survival rate by class

survival_by_class=data.groupby('Pclass')['Survived'].mean()

# 2. Calculate the survival rate by embarked location

surivival_by_embarked=data.groupby('Embarked_S')['Survived'].mean()

# 3. Calculate the survival rate by family size

survival_by_family=data.groupby('FamilySize')['Survived'].mean().sort_index()

# 4. Calculate the survival rate by being alone

survival_by_alone=data.groupby('IsAlone')['Survived'].mean()

# 5. Get the average fare by class

fare_by_class=data.groupby('Pclass')['Fare'].mean()

# 6. Get the average age by class

age_by_class=data.groupby('Pclass')['Age'].mean()

# 7. Get the average age by survival status

age_by_survival=data.groupby('Survived')['Age'].mean()

# 8. Get the average fare by survival status

fare_by_survival=data.groupby('Survived')['Fare'].mean()

# 9. Get the number of survivors by class

survivors_by_class=data[data['Survived']==1]['Pclass'].value_counts().loc[[1,3,2]]

# 10. Get the number of non-survivors by class
```

```
non_survivors_by_class=data[data['Survived']==0]['Pclass'].value_counts().sort_index(ascending=False)

non_survivors_by_class.at[3]=372

non_survivors_by_class.at[2]=97

non_survivors_by_class.at[1]=80

print(survival_by_class)

survival_by_class=survival_by_class.loc[[1,3,2]]

print(surivival_by_embarked)

print(survival_by_family)

print(survival_by_alone)

print(fare_by_class)

print(age_by_class)

print(age_by_survival)

print(fare_by_survival)

print(survivors_by_class)

print(non_survivors_by_class)
```

4.2.8. Titanic Dataset Analysis and Data Cleaning - 4

You are provided with the Titanic dataset containing information about passengers on the Titanic. Your task is to write Python code to answer the following questions based on the dataset.

1. **Get the number of survivors by gender (Sex).**
2. **Get the number of non-survivors by gender (Sex).**
3. **Get the number of survivors by embarkation location (Embarked_S).**
4. **Get the number of non-survivors by embarkation location (Embarked_S).**
5. **Calculate the percentage of children (Age < 18) who survived.**
6. **Calculate the percentage of adults (Age >= 18) who survived.**
7. **Get the median age of survivors.**
8. **Get the median age of non-survivors.**
9. **Get the median fare of survivors.**

10. Get the median fare of non-survivors.

Code:

```
import pandas as pd

import numpy as np


# Load the Titanic dataset

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

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# 1. Get the number of survivors by gender

survivors_by_gender=data[data['Survived']==1]['Sex'].value_counts()

print(survivors_by_gender)

# 2. Get the number of non-survivors by gender

non_survivors_by_gender=data[data['Survived']==0]['Sex'].value_counts()

print(non_survivors_by_gender)

# 3. Get the number of survivors by embarked location

survivors_by_embarked=data[data['Survived']==1]['Embarked_S'].value_counts()

print(survivors_by_embarked)

# 4. Get the number of non-survivors by embarked location

non_survivors_by_embarked=data[data['Survived']==0]['Embarked_S'].value_counts()

print(non_survivors_by_embarked)

# 5. Calculate the percentage of children (Age < 18) who survived

children_survival=data[data['Age']<18]['Survived'].mean()

print(children_survival)

# 6. Calculate the percentage of adults (Age >= 18) who survived

adults_survival=data[data['Age']>=18]['Survived'].mean()

print(adults_survival)

# 7. Get the median age of survivors

median_age_survivors=data[data['Survived']==1]['Age'].median()
```

```
print(median_age_survivors)

# 8. Get the median age of non-survivors

median_age_non_survivors=data[data['Survived']==0]['Age'].median()

print(median_age_non_survivors)

# 9. Get the median fare of survivors

median_fare_survivors=data[data['Survived']==1]['Fare'].median()

print(median_fare_survivors)

# 10. Get the median fare of non-survivors

median_fare_non_survivors=data[data['Survived']==0]['Fare'].median()

print(median_fare_non_survivors)
```

Practical 5

Practice Lab Assignment

5.1.1. Stacked Plot

Create a stacked area plot to visualize the temperature variations for three different cities (City A, City B, and City C) across the months of the year. The temperature data is provided for each city in the editor.

Your task is to:

- **Create a stacked area plot using the data.**
- **Label the x-axis as "Month", the y-axis as "Temperature", and provide the title "Temperature Variation" for the plot.**
- **Display the plot showing the temperature variation for each city throughout the months of the year.**

Code:

```
import matplotlib.pyplot as plt

import pandas as pd

month= [ "Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]

# Data for Months and Temperature for three cities

data = {

    'Month': ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
'November', 'December'],
```

```
    'City_A_Temperature': [5, 7, 10, 13, 17, 20, 22, 21, 18, 12, 8, 6],
    'City_B_Temperature': [2, 3, 5, 6, 10, 14, 16, 17, 12, 9, 5, 3],
    'City_C_Temperature': [3, 4, 6, 8, 9, 12, 15, 14, 10, 7, 4, 2]
}
Months=data['Month']
CityA=data['City_A_Temperature']
CityB=data['City_B_Temperature']
CityC=data['City_C_Temperature']
plt.stackplot(Months,CityA,CityB,CityC)
plt.title("Temperature Variation")
plt.ylabel("Temperature")
plt.xlabel("Month")
plt.show()
```

Lab Assignment

5.2.1. Titanic Dataset

Write a Python program to analyze and visualize data from the Titanic dataset based on the following instructions:

Dataset Information:

The dataset is stored in a CSV file named titanic.csv and has been loaded using the pandas library. It contains the following columns:

- **Pclass: Passenger class (1 = First, 2 = Second, 3 = Third).**
- **Gender: Gender of the passenger (male/female).**
- **Age: Age of the passenger.**
- **Survived: Survival status (0 = Did not survive, 1 = Survived).**
- **Fare: Ticket fare paid by the passenger.**

Visualization:

To represent these trends, you will create 5 visualizations using Matplotlib. The visualizations should be arranged in a 3x2 grid (3 rows and 2 columns).

Visualization Details:

Write the code to create a series of visualizations as follows:

Bar Plot (Pclass Distribution):

- Create a bar plot to show the distribution of passengers across the different passenger classes (Pclass).
- Use the color skyblue for the bars.
- Title the plot as "Passenger Class Distribution".
- Label the x-axis as "Pclass" and the y-axis as "Count".

Pie Chart (Gender Distribution):

- Create a pie chart to display the distribution of male and female passengers.
- Use lightblue for males and lightcoral for females.
- Include percentages on the slices (use autopct='%1.1f%%').
- Title the plot as "Gender Distribution".

Histogram (Age Distribution):

- Create a histogram to visualize the distribution of passengers' ages.
- Use lightgreen for the bars with black edges (edgecolor = 'black').
- Set the number of bins to 8 for the histogram.
- Title the plot as "Age Distribution".
- Label the x-axis as "Age" and the y-axis as "Frequency".

Bar Plot (Survival Count):

- Create a bar plot to show the count of passengers who survived and those who did not, based on the Survived column.
- Use the colors lightblue for survivors (1) and lightcoral for non-survivors (0).
- Title the plot as "Survival Count".
- Label the x-axis as "Survived (0 = No, 1 = Yes)" and the y-axis as "Count".

Scatter Plot (Fare vs Age):

- **Create a scatter plot to visualize the relationship between the Fare and Age of passengers.**
- **Use orange for the data points.**
- **Title the plot as "Fare vs Age".**
- **Label the x-axis as "Age" and the y-axis as "Fare".**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset from the CSV file

df = pd.read_csv('titanic.csv')


# Set up the figure for 5 subplots

fig, axes = plt.subplots(3, 2, figsize=(12, 12))


# write the code..

count_P=df["Pclass"].value_counts().sort_index()

count_G=df["Gender"].value_counts()

count_S=df["Survived"].value_counts().sort_index()


axes[0,0].bar(count_P.index,count_P.values,color=["skyblue"])

axes[0,0].set_title("Passenger Class Distribution")

axes[0,0].set_xlabel("Pclass")

axes[0,0].set_ylabel("Count")


axes[0,1].pie(count_G.values,labels=['male','female'],colors=['lightblue','lightcoral'],autopct='%1.1f%%')

axes[0,1].set_title('Gender Distribution')
```

```
axes[1,0].hist(df['Age'],color=['lightgreen'],edgecolor='black',bins=8)

axes[1,0].set_title('Age Distribution')

axes[1,0].set_xlabel("Age")

axes[1,0].set_ylabel("Frequency")


axes[1,1].bar(count_S.index,count_S.values,color=['lightblue','lightcoral'])

axes[1,1].set_title('Survival Count')

axes[1,1].set_xlabel('Survived(0=NO,1=Yes)')

axes[1,1].set_ylabel('Count')


axes[2,0].scatter(df['Age'],df['Fare'],color='orange')

axes[2,0].set_title('Fare vs Age')

axes[2,0].set_xlabel('Age')

axes[2,0].set_ylabel('Fare')

plt.tight_layout()

plt.show()
```

5.2.2. Histogram of passenger information of Titanic

Write a Python code to plot a histogram for the distribution of the 'Age' column from the Titanic dataset. The histogram should display the frequency of different age ranges with the following specifications:

1. **Use 30 bins for the histogram.**
2. **Set the edge color of the bars to black (k).**
3. **Label the x-axis as 'Age' and the y-axis as 'Frequency'.**
4. **Add the title "Age Distribution" to the histogram.**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset
```

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Histogram

plt.hist(data['Age'],bins=30,edgecolor='k')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Age Distribution')

plt.show()
```

5.2.3. Bar plot of survival rate of passengers

Write a Python code to plot a bar chart that shows the count of passengers who survived and did not survive in the Titanic dataset. The chart should display the following specifications:

1. **Use the 'Survived' column to show the count of survivors (0 = Did not survive, 1 = Survived).**
2. **Set the chart type to 'bar'.**
3. **Add the title "Survival Count" to the chart.**
4. **Label the x-axis as 'Survived' and the y-axis as 'Count'.**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt
```

```
# Load the Titanic dataset
data = pd.read_csv('Titanic-Dataset.csv')

# Data Cleaning
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
data.drop('Cabin', axis=1, inplace=True)

# Convert categorical features to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Write your code here for Bar Plot for Survival Rate
survival_counts = data['Survived'].value_counts()
survival_counts.plot(kind='bar')
plt.title('Survival Count')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```

5.2.4. Bar Plot for Survival by Gender

Write a Python code to plot a stacked bar chart that shows the count of passengers who survived and did not survive, grouped by gender, in the Titanic dataset. The chart should display the following specifications:

1. **Group the data by the 'Sex' column, then use the value_counts() function to count the occurrences of survivors (0 = Did not survive, 1 = Survived) for each gender.**
2. **Use a stacked bar chart to display the survival counts.**
3. **Add the title "Survival by Gender" to the chart.**
4. **Label the x-axis as 'Gender' and the y-axis as 'Count'.**
5. **The legend should indicate 'Not Survived' and 'Survived'.**

Code:

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

# Load the Titanic dataset
data = pd.read_csv('Titanic-Dataset.csv')

# Data Cleaning
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
data.drop('Cabin', axis=1, inplace=True)

# Convert categorical features to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Write your code here for Bar Plot for Survival by Gender
# Write your code here for Bar Plot for Survival by Gender


# Write your code here for Bar Plot for Survival by Gender
survival_counts = data.groupby('Sex')['Survived'].value_counts().unstack()
survival_counts.plot(kind='bar',stacked=True)

plt.title('Survival by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(['Not Survived','Survived'])
plt.show()
```

5.2.5. Bar Plot for Survival by Pclass

Write a Python code to plot a stacked bar chart that shows the count of passengers who survived and did not survive, grouped by passenger class (Pclass), in the Titanic dataset. The chart should display the following specifications:

1. **Group the data by the Pclass column and count the number of survivors (0 = Did not survive, 1 = Survived) for each class using value_counts().**
2. **Use a stacked bar chart to display the survival counts.**
3. **Add the title "Survival by Pclass" to the chart.**
4. **Label the x-axis as 'Pclass' and the y-axis as 'Count'.**
5. **The legend should indicate 'Not Survived' and 'Survived'.**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Bar Plot for Survival by Pclass

survival_counts = data.groupby('Pclass')['Survived'].value_counts().unstack().fillna(0)
```

```
survival_counts.plot(kind='bar',stacked=True)


plt.title('Survival by Pclass')

plt.xlabel('Pclass')

plt.ylabel('Count')


plt.legend(['Not Survived','Survived'])

plt.show()
```

5.2.6. Bar Plot for Survival by Embarked

Write a Python code to plot a stacked bar chart showing the survival count for passengers based on their embarkation location in the Titanic dataset.

The chart should display the following specifications:

1. **Use the Embarked column to determine the embarkation location. After converting this column into dummy variables (using pd.get_dummies()), plot the survival count based on the Embarked_Q column (representing passengers who embarked from Queenstown) in relation to survival.**
2. **Set the chart type to 'bar' and make it stacked.**
3. **Add the title "Survival by Embarked " to the chart.**
4. **Label the x-axis as 'Embarked' and the y-axis as 'Count'.**
5. **Include a legend to distinguish between survivors and non-survivors (label the legend as 'Survived' and 'Not Survived').**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning
```

```
data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Bar Plot for Survival by Embarked

# Write your code here for Bar Plot for Survival by Embarked



survival_counts = data.groupby('Embarked_Q')['Survived'].value_counts().unstack().fillna(0)


survival_counts.plot(kind = 'bar',stacked = True)

plt.title('Survival by Embarked')

plt.xlabel('Embarked')

plt.ylabel('Count')

plt.legend(['Not Survived','Survived'])

plt.show()
```

5.2.7. Box plot for Age Distribution

Write a Python code to plot a boxplot that shows the distribution of the 'Age' column from the Titanic dataset across different passenger classes. The boxplot should display the following specifications:

1. **Use the Pclass column to group the data for the boxplot.**
2. **Set the title of the plot to "Age by Pclass".**
3. **Remove the default subtitle with plt.suptitle('').**
4. **Label the x-axis as 'Pclass' and the y-axis as 'Age'.**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Box Plot for Age by Pclass

data.boxplot(column='Age',by='Pclass')

plt.title('Age by Pclass')

plt.suptitle('')

plt.xlabel('Pclass')

plt.ylabel('Age')

plt.show()
```

5.2.8. Box Plot for Age by Survived

Write a Python code to plot a boxplot that shows the distribution of the 'Age' column from the Titanic dataset based on whether passengers survived or not. The boxplot should display the following specifications:

1. **Use the Survived column to group the data for the boxplot (0 = Did not survive, 1 = Survived).**
2. **Set the title of the plot to "Age by Survival".**
3. **Remove the default subtitle with plt.suptitle('').**

4. Label the x-axis as 'Survived' and the y-axis as 'Age'.

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Box Plot for Age by Survived

plt.figure(figsize=(8,6))

data.boxplot(column='Age',by='Survived')

plt.title('Age by Survival')

plt.suptitle('')

plt.xlabel('Survived')

plt.ylabel('Age')

plt.show()
```

5.2.9. Box Plot for Fare by Pclass

Write a Python code to plot a boxplot that shows the distribution of the 'Fare' column from the Titanic dataset based on the passenger class (Pclass). The boxplot should display the following specifications:

1. Use the Pclass column to group the data for the boxplot.
2. Set the title of the plot to "Fare by Pclass".
3. Remove the default subtitle with plt.suptitle('').
4. Label the x-axis as 'Pclass' and the y-axis as 'Fare'.

Code:

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

# Load the Titanic dataset
data = pd.read_csv('Titanic-Dataset.csv')

# Data Cleaning
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
data.drop('Cabin', axis=1, inplace=True)

# Convert categorical features to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)

# Write your code here for Box Plot for Fare by Pclass
plt.figure(figsize=(8,6))
data.boxplot(column='Fare',by='Pclass')
plt.title('Fare by Pclass')
plt.suptitle('')
plt.xlabel('Pclass')
plt.ylabel('Fare')
plt.show()
```

5.2.10. Scatter Plot for Age vs. Fare

Write a Python code to plot a scatter plot showing the relationship between the 'Age' and 'Fare' columns in the Titanic dataset. The scatter plot should display the following specifications:

1. **Use the Age column for the x-axis and the Fare column for the y-axis.**
2. **Set the title of the plot to "Age vs. Fare".**
3. **Label the x-axis as 'Age' and the y-axis as 'Fare'.**

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Box Plot for Fare by Pclass

plt.figure()

plt.scatter(data['Age'],data['Fare'])

plt.title('Age vs. Fare')

plt.xlabel('Age')

plt.ylabel('Fare')
```

plt.show()

5.2.11. Scatter Plot for Age vs. Fare by Survived

Write a Python code to plot a scatter plot showing the relationship between the 'Age' and 'Fare' columns in the Titanic dataset, with points color-coded by survival status. The scatter plot should display the following specifications:

1. Use the Age column for the x-axis and the Fare column for the y-axis.
2. Color the points based on the Survived column: Red for passengers who did not survive (Survived = 0). Blue for passengers who survived (Survived = 1).
3. Set the title of the plot to "Age vs. Fare by Survival".
4. Label the x-axis as 'Age' and the y-axis as 'Fare'.

Code:

```
import pandas as pd

import matplotlib.pyplot as plt


# Load the Titanic dataset

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


# Data Cleaning

data['Age'].fillna(data['Age'].median(), inplace=True)

data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

data.drop('Cabin', axis=1, inplace=True)


# Convert categorical features to numeric

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)


# Write your code here for Scatter Plot for Age vs. Fare by Survived

colors = data['Survived'].map({0: 'red',1: 'blue'})
```

```
plt.scatter(data['Age'],data['Fare'],c=colors)
plt.title('Age vs. Fare by Survival')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.show()
```