

EDS screenshots of LAB ASSIGNMENTS

NAME: Niharika Nishikant Shelke

BATCH: ME-2

Roll no. 45

Prn: 202401090024

PRACTICAL 1

The screenshot shows a browser window with multiple tabs open. The active tab is for a practical assignment titled "1.1.1. Calculate Momentum" on the website mitaoe.codetantra.com. The assignment instructions state that the program should accept mass (m) and velocity (v) as input and output momentum (p = m * v). A sample formula $p = m \times v$ is provided. The code editor contains the following Python code:

```
m=float(input())
v=float(input())
p=m*v
print("%0.2f"%p,end=' ')
print("kgm/s")
```

The code is submitted and tested. The results show two test cases passed with average times of 0.010 s and 0.014 s, and maximum times of 10.00 ms and 14.00 ms respectively. The expected and actual outputs for both test cases are 5.0 and 10.0, with the output unit being 50.00kgm/s.

At the bottom of the browser window, the Windows taskbar is visible, showing various application icons like File Explorer, Edge, and Microsoft Word.

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 .

Constraints:
 $1 \leq n \leq 999$

Input Format:
The input consists of a single integer n .

Output Format:
If n is a single-digit number, print its square.
If n is a two-digit number, print its square root (rounded to two decimal places).
If n is a three-digit number, print its cube root (rounded to two decimal places).
Else print "Invalid".

Sample Test Cases

```
condition...
1 n=int(input())
2 v if(n>=0 and n<=9):
3   print(n*n)
4
5 v elif(n>=10 and n<=99):
6   p=n**0.5
7   print("%0.2f"%p)
8
```

Average time: 0.006 s (5.57 ms) Maximum time: 0.008 s (8.00 ms)
4 out of 4 shown test case(s) passed
3 out of 3 hidden test case(s) passed

Test case 1 (5 ms)
Expected output: 9
Actual output: 9
81

Test case 2 (7 ms)

Test case 3 (6 ms)

Terminal Test cases

1.1.3. Age and Salary Calculation

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

Input Format:
The input consists of:
A string representing the birth date of the employee in the format $DD - MM - YYYY$.
A floating-point number representing the salary of the employee in rupees.

Output Format:
The output should include:
The age of the employee.
The salary of the employee in dollars.

Note:
1INR=0.012USD

Sample Test Cases

```
birthDate...
1 from datetime import datetime
2
3 v def calculate_age(birthdate):
4   date_object = datetime.strptime(birthdate, "%d-%m-%Y")
5   today = datetime.today()
6 v   if((today.month, today.day) < (date_object.month, date_object.day)):
7     age = today.year - date_object.year
8
```

Average time: 0.029 s (29.25 ms) Maximum time: 0.066 s (66.00 ms)
2 out of 2 shown test case(s) passed
1 out of 2 hidden test case(s) passed

Test case 1 (66 ms)
Expected output: 15-06-1991
Actual output: 15-06-1991
50000
Age: 33
Salary-in-dollars: 600.00
Salary-in-dollars: 600.00

Test case 2 (7 ms)

Terminal Test cases

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.

Input Format:
The input is an integer.

Output Format:
Print a single integer which is the reversed number.

Sample Test Cases

Explorer: reverseN...
1 num=int(input())
2 n1=str(num)
3 print(n1[: :-1])

Average time: 0.007 s (7.20 ms) | Maximum time: 0.010 s (10.00 ms)
2 out of 2 shown test case(s) passed
3 out of 3 hidden test case(s) passed

Test case 1 (10 ms)
Expected output: 5367
Actual output: 5367
7635

Test case 2 (6 ms)

Terminal | Test cases | < Prev | Reset | Submit | Next >

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.

Input Format:
The first line of input contains an integer that represents the number for which the multiplication table is to be printed.

Output Format:
Print the multiplication table for the given number .

Sample Test Cases

Explorer: multiplica...
1 i=int(input())
2 n=1
3 while n<=10:
4 print(i,"x",n,"=",i*n)
5 n=n+1

Average time: 0.004 s (3.50 ms) | Maximum time: 0.004 s (4.00 ms)
2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (4 ms)
Expected output: 8
Actual output: 8
8 · x · 1 = 8
8 · x · 2 = 16
8 · x · 3 = 24
8 · x · 4 = 32
8 · x · 5 = 40

Terminal | Test cases | < Prev | Reset | Submit | Next >

Course

2304102 ALL PR: Public L | Upload files - niharika-0 | +

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e447f1f9c5320ca6bce4/6773e49af1f9c5320ca6bd79/677...

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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.

Input Format:
The first input will be an integer n , the number of courses.
The second input will be n integers representing the marks of the student in each of the n courses, separated by a space.

Output Format:
If the student passes all courses:

Sample Test Cases

passorFail.py

```
1 n = int(input())
2 marks = list(map(int,input().split(" ")))
3 v if all (mark>=40 for mark in marks):
4     per = sum(marks)/n
5     print(f"Aggregate Percentage: {per:.2f}")
6 v if per>=75:
7     print("Grade: Distinction")
8 v elif 60<=per<75:
```

Average time: 0.007 s Maximum time: 0.009 s
7.50 ms 9.00 ms

2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (6 ms)
Expected output: 5
Actual output: 5
56 78 97 86 93 Aggregate Percentage: 82.00 Grade: Distinction

Test case 2 (8 ms)
Terminal Test cases

< Prev Reset Submit Next >

21:41 06-05-2025

photos - Google Drive | Apple Music - Web Playe | Course | 2304102 ALL PR: Public L | Upload files - niharika-0 | +

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e447f1f9c5320ca6bce4/6773e49af1f9c5320ca6bd79/65f...

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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.

Expected Output-1:
Enter terms for Fibonacci series: 5
0 1 1 2 3

Expected Output-2:
Enter terms for Fibonacci series: 9
0 1 1 2 3 5 8 13 21

Instructions:

- Your input and output must follow the input and output layout mentioned in the visible sample test case.
- Hidden test cases will only pass when users' input and output match the expected input and output.

Sample Test Cases

fib.py

```
1 v def fib(i):
2 v     if(i==0):
3         return 0
4     v elif(i==1):
5         return 1
6     v else:
7         return fib(i-1)+fib(i-2)
8 v = = =
```

Average time: 0.011 s Maximum time: 0.016 s
11.50 ms 16.00 ms

2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (16 ms)
Expected output: Enter terms for Fibonacci series: 5
Actual output: Enter terms for Fibonacci series: 5
0 1 1 2 3

Test case 2 (8 ms)

< Prev Reset Submit Next >

1.2.3. Pattern - 1

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

Input Format:
The input is an integer, representing the number of rows in the pattern.

Output Format:
The output should display the pattern of asterisks (*), with each row containing an increasing number of asterisks.

Note:
Refer to the displayed test cases for the sample pattern.

Sample Test Cases

rightangl...

```
1 a=int(input())
2 for i in range (1,a+1):
3     print('* '*i)
```

Average time: 0.004 s (4.00 ms) | Maximum time: 0.006 s (6.00 ms)

2 out of 2 shown test case(s) passed | 4 out of 4 hidden test case(s) passed

Test case 1 (4 ms)
Expected output
5
* *
* * *
* * * *
* * * * *
Actual output
5
* *
* * *
* * * *
* * * * *

Terminal Test cases < Prev Reset Submit Next >

1.2.4. Pattern - 2

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

Input Format:
The input is an integer, representing the number of rows in the pattern.

Output Format:
The output should display the pattern of numbers, with each row containing increasing numbers starting from 1 up to the row number.

Note:
Refer to the displayed test cases for the sample pattern.

Sample Test Cases

numberP...

```
1 n=int(input())
2 i=1
3 while (i<=n):
4     j=1
5     while (j<=i):
6         print(j,end=" ")
7         j+=1
8     print()
```

Average time: 0.009 s (8.75 ms) | Maximum time: 0.013 s (13.00 ms)

2 out of 2 shown test case(s) passed | 2 out of 2 hidden test case(s) passed

Test case 1 (13 ms)
Expected output
5
1 2
1 2 3
1 2 3 4
1 2 3 4 5
Actual output
5
1 2
1 2 3
1 2 3 4
1 2 3 4 5

Terminal Test cases < Prev Reset Submit Next >

PRACTICAL 2:

The screenshot shows a browser window with multiple tabs open, including a course page from CodeTantra and a file upload page. The main content area displays a Python code editor for performing dictionary operations. The code uses a for loop to add items to a dictionary and then prints it. Below the code, test cases are shown with expected and actual outputs matching.

```
# 1. Create an empty dictionary and display it
my_dict = {}
print("Empty Dictionary:", my_dict)
n = int(input("Number of items: "))
size = n
for _ in range(n):
    key = input("key: ")
    value = input("value: ")
    my_dict[key] = value
print(my_dict)
```

Average time	Maximum time
0.057 s	0.074 s
56.50 ms	74.00 ms

Test case 1 (40 ms)

Expected output	Actual output
Empty Dictionary: {}	Empty Dictionary: {}
Number of items: 1	Number of items: 1
key: Name	key: Name
value: Alice	value: Alice
Dictionary: {'Name': 'Alice'}	Dictionary: {'Name': 'Alice'}
Enter the key to update: Name	Enter the key to update: Name

Sample Test Cases +

Terminal Test cases

< Prev Reset Submit Next >

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.

Input format:

- The first line of input contains the array of integers which are separated by space
- The last line of input contains the key element to be searched

Output format:

- If the element is found, print the index.
- If the element is not found, print **Not found**.

Sample Test Case:

Input:
1 2 3 4 3 5 6
3

Output:
2

Sample Test Cases +

CTP1709...

```
1 arr = list(map(int,input().split(" ")))
2
3 key = int(input())
4
5 for i in range(len(arr)):
6     if arr[i] == key:
7         print(i)
8         break
```

Average time: 0.009 s Maximum time: 0.011 s
8.50 ms 11.00 ms 2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (11 ms)
Expected output: 1 2 3 4 3 5 6
Actual output: 1 2 3 4 3 5 6
3
2

Test case 2 (8 ms)

Terminal Test cases < Prev Reset Submit Next >

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.

Input Format:

The first line of input will contain 11 integers, each representing the height of a player (in centimeters), each separated by a space.

Output Format:

The output should be the height (in centimeters) of the tallest player.

Sample Test Cases +

captainof...

```
1 heights = list(map(int,input().split(" ")))
2
3 captain = max(heights)
4
5 print(captain)
```

Average time: 0.005 s Maximum time: 0.007 s
5.33 ms 7.00 ms 1 out of 1 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (5 ms)
Expected output: 171 169 185 156 174 191 186 190 187
172 160
191
Actual output: 171 169 185 156 174 191 186 190 187
172 160
191

Terminal Test cases < Prev Reset Submit Next >

PRACTICAL 3:

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.

Input Format:

- User inputs the number of rows and columns with space separated values.
- User inputs elements of the array row-wise followed line by line, separated by spaces.

Output Format:

- The created NumPy array based on the input dimensions and elements.
- Dimensions (ndim): Number of dimensions of the array.
- Shape: Tuple representing the shape of the array (number of rows, number of columns).
- Size: Total number of elements in the array.

Note: Use reshape() function to reshape the input array with the specified number of rows and columns.

Sample Test Cases

numpyarr...

```
1 import numpy as np
2 import numpy as np
3 rows,cols= list(map(int,input().split()))
4 matrix= []
5 for i in range(rows):
6     row = list(map(int,input().split()))
7     matrix.append(row)
8 matrix= np.array(matrix).reshape(rows,cols)
```

Average time: 0.010 s Maximum time: 0.017 s
10.00 ms 17.00 ms

3 out of 3 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (14 ms)

Expected output: 3 4
1 2 3 4
5 6 7 8
9 10 11 12

Actual output: 3 4
1 2 3 4
5 6 7 8
9 10 11 12

Terminal Test cases

Submit



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

Input Format:

- The user will input 3 rows for matrix_a, each containing 3 integers separated by spaces.
- Similarly, the user will input 3 rows for matrix_b, each containing 3 integers separated by spaces.

Sample Test Cases

matrixOp...

```
1 import numpy as np
2
3 # Input matrices
4 print("Enter Matrix A:")
5 matrix_a = np.array([list(map(int, input().split())) for
6 i in range(3)])
7
7 print("Enter Matrix B:")
8
```

Average time: 0.022 s Maximum time: 0.033 s
21.75 ms 33.00 ms

2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 (33 ms)

Expected output: Enter Matrix A:
1 2 3
4 5 6
7 8 9

Actual output: Enter Matrix A:
1 2 3
4 5 6
7 8 9

Enter Matrix B:
1 1 1

Enter Matrix B:
1 1 1

Terminal Test cases

Submit

21:42 06-05-2025



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).

Input Format:

- The program should first prompt the user to input two 3×3 arrays.
- Each array consists of 3 rows, and each row contains 3 space-separated integers.
- The user will input the two arrays row by row.

Output Format:

- The program should display the result of the Horizontal Stack (side-by-side stacking) of the two arrays.
- The program should then display the result of the Vertical Stack (one below the other) of the two arrays.

Sample Test Cases

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.

Input Format:

- The user will input three integer values: start, stop, and step, each on a new line.

Output Format:

- The program should print the generated sequence based on the input values.

Sample Test Cases

Course

2304102 ALL PR: Public L | Upload files - niharika-0 | +

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e451f1f9c5320ca6bd17/6773e4d1f1f9c5320ca6bdb8/677...

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

3.2.4. Numpy: Arithmetic and Statistical Operations, Mathematics

You are 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:

- 1. Arithmetic Operations:**
 - Compute the element-wise sum, difference, and product of the two arrays.
- 2. Statistical Operations:**
 - Calculate the mean, median, and standard deviation of array A.
- 3. Bitwise Operations:**
 - Perform bitwise AND, bitwise OR, and bitwise XOR on the arrays (ex: $A_1 \text{ OR } B_1$).

Input Format:

- The first line contains space-separated integers representing the elements of array A.
- The second line contains space-separated integers representing the elements of array B.

Sample Test Cases

different...

```
1 import numpy as np
2
3 def array_operations(A, B):
4     # Convert A and B to NumPy arrays
5     A = np.array(A)
6     B = np.array(B)
7
8     # Arithmetic Operations
```

Average time: 0.012 s Maximum time: 0.025 s
12.33 ms 25.00 ms

1 out of 1 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 25 ms

Expected output	Actual output
1 2 3 4	1 2 3 4
5 6 7 8	5 6 7 8

Element-wise Sum: 6 8 10 12 Element-wise Sum: 6 8 10 12

Element-wise Difference: -4 -4 -4 -4 Element-wise Difference: -4 -4 -4 -4

Element-wise Product: 5 12 21 32 Element-wise Product: 5 12 21 32

Mean of A: 2.5 Mean of A: 2.5

Terminal Test cases

< Prev Reset Submit Next >

21:43 06-05-2025

photos - Google Drive | Apple Music - Web Playe | Course | 2304102 ALL PR: Public L | Upload files - niharika-0 | +

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e451f1f9c5320ca6bd17/6773e4d1f1f9c5320ca6bdb8/677...

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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.

Input Format:

- A single line of space-separated integers.

Output Format:

- After modifying the view:

```
Original array after modifying view: <original_array>
View array: <view_array>
```

- After modifying the copy:

Sample Test Cases

copyAnd...

```
1 import numpy as np
2
3 inputlist = list(map(int,input().split(" ")))
4
5 # Original array
6 original_array = np.array(inputlist)
7
8 # Create a view
```

Average time: 0.006 s Maximum time: 0.009 s
6.25 ms 9.00 ms

2 out of 2 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test case 1 9 ms

Expected output	Actual output
10 20 30 40 50 60 70 80	10 20 30 40 50 60 70 80
Original array after modifying view: [9 9 20 30 40 50 60 70 80]	Original array after modifying view: [99 20 30 40 50 60 70 80]
View array: [99 20 30 40 50 60 70 80]	View array: [99 20 30 40 50 60 70 80]
Original array after modifying copy: [9 9 20 30 40 50 60 70 80]	Original array after modifying copy: [99 20 30 40 50 60 70 80]

Terminal Test cases

< Prev Reset Submit Next >

3.2.6. Numpy: Searching, 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.

Input Format:
1 A single line containing space-separated integers representing `array1`

Sample Test Cases

arrayOpe...

```

1 import numpy as np
2
3 # Input array from the user
4 array1 = np.array(list(map(int, input().split())))
5
6 # Searching
7 search_value = int(input("Value to search: "))
8 count_value = int(input("Value to count: "))

```

Average time: 0.016 s (15.75 ms) | Maximum time: 0.019 s (19.00 ms)

2 out of 2 shown test case(s) passed | 2 out of 2 hidden test case(s) passed

Test case 1 (19 ms)

Expected output	Actual output
1 1 1 2 2 2	1 1 1 2 2 2
Value to search: 1	Value to search: 1
Value to count: 2	Value to count: 2
Value to add: 2	Value to add: 2
[0-1-2]	[0-1-2]
3	3

Terminal Test cases < Prev Reset Submit Next >

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

Sample Test Cases

Operatio...

```

1 import numpy as np
2
3 a = np.loadtxt("Sample.csv", delimiter=',', skiprows=1)
4 # 1. Print all student details
5 print("All student Details:\n",a)
6
7 # 2. print total students
8 r,c=a.shape

```

Average time: 0.028 s (28.00 ms) | Maximum time: 0.028 s (28.00 ms)

1 out of 1 shown test case(s) passed

Test case 1 (28 ms)

Expected output	Actual output
All student Details:	All student Details:
[[301, 67, 77, 88],	[301, 67, 77, 88]
[302, 78, 88, 77],	[302, 78, 88, 77]
[303, 45, 56, 89],	[303, 45, 56, 89]
[304, 88, 98, 45],	[304, 88, 98, 45]
[305, 78, 88, 99],	[305, 78, 88, 99]

Terminal Test cases < Prev Reset Submit Next >

PRACTICAL 4:

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.

Input Format:

- The user should enter a list of numbers separated by space when prompted.

Output Format:

- The program should display the mean of even and odd numbers separately.
- Each mean value should be displayed with a label indicating whether it corresponds to even or odd numbers.

Sample Test Cases

Code Editor (seriesMa...):

```
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
even_mean = series.groupby(series % 2 == 0).mean()
odd_mean = series.groupby(series % 2 == 1).mean()
```

Test Case 1: 1 2 3 4 5 6 7 8 9 10

Expected output	Actual output
Mean-of-even-and-odd-numbers: 5.0	Mean-of-even-and-odd-numbers: 5.0
Odd.....5.0	Odd.....5.0
Even....6.0	Even....6.0
dtype:float64	dtype:float64

Test case 1 29 ms

Test cases

Average time: 0.012 s | Maximum time: 0.029 s | 11.83 ms

3 out of 3 shown test case(s) passed | 3 out of 3 hidden test case(s) passed

21:44 06-05-2025

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.

Create the DataFrame:

- Convert the dictionary to a Pandas DataFrame.

Add a new row:

- Take inputs from the user for the new row data (name, age).
- Add the new row to the DataFrame.
- Display the DataFrame after adding the new row.

Modify a row:

- Modify a specific row by changing the age. Take the row index and new age value from the user.
- Display the DataFrame after modifying the row.

Delete a row:

Sample Test Cases

Code Editor (datafram...):

```
import pandas as pd
# Provided dictionary of lists
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35]
}
```

Test Case 1 112 ms

Expected output	Actual output
Original-DataFrame:	Original-DataFrame:
.....Name.....AgeName.....Age
0.....Alice....25	0.....Alice....25
1.....Bob....30	1.....Bob....30
2.....Charlie....35	2.....Charlie....35
New-name: Susan	New-name: Susan

Test cases

Average time: 0.149 s | Maximum time: 0.187 s | 149.50 ms

1 out of 1 shown test case(s) passed | 1 out of 1 hidden test case(s) passed

21:44 06-05-2025

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:

- Display the first five rows of the data frame.
- Calculate the average age of the students(limit the average age up to 2 decimal places).
- Filter out the students who have a grade above a certain threshold(consider the threshold grade is 'B').

Note:
Refer to the displayed test cases for better understanding.

Sample Test Cases

Code Editor:

```
1 import pandas as pd
2
3 # Read the text file into a DataFrame
4 file = input()
5 data = pd.read_csv(file, sep="\s+", header=None, names=["Name", "Age", "Grade"])
6 print("First five rows:")
7 print(data.head(5))
```

Test Results:

Average time	Maximum time
0.039 s 39.00 ms	0.056 s 56.00 ms

1 out of 1 shown test case(s) passed
1 out of 1 hidden test case(s) passed

Test Case Details:

Test case 1	56 ms
Expected output	Actual output
studentdata.txt	studentdata.txt
First five rows:	First five rows:
... Name ... Age ... Grade Name ... Age ... Grade ...
0 - John - 25 - A	0 - John - 25 - A
1 - Alin - 22 - B	1 - Alin - 22 - B
2 - Emma - 24 - A	2 - Emma - 24 - A

Terminal **Test cases** **< Prev** **Reset** **Submit** **Next >**

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.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
```

Sample Test Cases

Code Editor:

```
1 import pandas as pd
2
3 # Prompt the user for the file name
4 file_name = input()
5
6 # Load the data
7 df = pd.read_csv(file_name)
```

Test Results:

Average time	Maximum time
0.027 s 26.67 ms	0.051 s 51.00 ms

1 out of 1 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test Case Details:

Test case 1	51 ms
Expected output	Actual output
sales_data.csv	sales_data.csv
Best month: 2025-01	Best month: 2025-01
Total sales: \$1210.00	Total sales: \$1210.00

Terminal **Test cases** **< Prev** **Reset** **Submit** **Next >**

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.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
```

Sample Test Cases

Test Results:

Average time: 0.020 s | Maximum time: 0.038 s | 20.00 ms | 38.00 ms

1 out of 1 shown test case(s) passed | 2 out of 2 hidden test case(s) passed

Test case 1 (38 ms)
Expected output: sales_data.csv
Actual output: sales_data.csv
Best-selling product: Product A
Total quantity sold: 23

Terminal **Test cases** **Submit** **Reset** **Debug** **Next >** **< Prev**

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).

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
```

Sample Test Cases

Test Results:

Average time: 0.022 s | Maximum time: 0.040 s | 22.00 ms | 40.00 ms

1 out of 1 shown test case(s) passed | 2 out of 2 hidden test case(s) passed

Test case 1 (40 ms)
Expected output: sales_data.csv
Actual output: sales_data.csv
City sold the most products: Los Angeles

Terminal **Test cases** **Submit** **Reset** **Debug** **Next >** **< Prev**

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.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
```

Test Cases

Code Editor:

```
frequent... sales_dat...
1 import pandas as pd
2 from itertools import combinations
3 from collections import Counter
4
5 # Prompt user to input the file name
6 file_name = input()
7
8 # Read data from the specified CSV file
```

Test Results:

Average time 0.025 s	Maximum time 0.050 s
25.33 ms	50.00 ms

1 out of 1 shown test case(s) passed
2 out of 2 hidden test case(s) passed

Test Case 1: Expected output: sales_data.csv, Actual output: sales_data.csv

Product A and Product B: 2-times
Product A and Product C: 2-times

Terminal

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.

- Display the first 5 rows of the dataset.
- Display the last 5 rows of the dataset.
- Get the shape of the dataset (number of rows and columns).
- Get a summary of the dataset (using .info()).
- Get basic statistics (mean, standard deviation, etc.) of the dataset using .describe().
- Check for missing values and display the count of missing values for each column.
- Fill missing values in the 'Age' column with the median age.
- Fill missing values in the 'Embarked' column with the most frequent value (mode).
- Drop the 'Cabin' column due to many missing values.
- Create a new column, 'FamilySize' by adding the 'SibSp' and 'Parch' columns.

Sample Test Cases

Code Editor:

```
titanicDat...
1 import pandas as pd
2 import numpy as np
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6
7 # 1. Display the first 5 rows of the dataset
8 # 1. Display the first 5 rows of the dataset
```

Test Cases

Test Results:

Average time 0.198 s	Maximum time 0.198 s
198.00 ms	198.00 ms

1 out of 1 shown test case(s) passed

Test Case 1: Expected output: titanicDat..., Actual output: titanicDat...

PassengerId Survived Pclass ... Fare Cabin Embarked ...
1 0 3 ... 7.2500 S ...
2 1 1 ... 71.2833 C ...
3 3 1 ... 7.9250 S ...
4 0 3 ... 7.2500 S ...
5 1 2 ... 71.2833 C ...
6 2 3 ... 7.9250 S ...
7 0 3 ... 7.2500 S ...
8 1 1 ... 7.2500 S ...
9 1 2 ... 71.2833 C ...
10 3 1 ... 7.9250 S ...
11 0 3 ... 7.2500 S ...
12 1 2 ... 71.2833 C ...
13 2 3 ... 7.9250 S ...
14 0 3 ... 7.2500 S ...
15 1 1 ... 7.2500 S ...
16 1 2 ... 71.2833 C ...
17 3 1 ... 7.9250 S ...
18 0 3 ... 7.2500 S ...
19 1 2 ... 71.2833 C ...
20 2 3 ... 7.9250 S ...
21 0 3 ... 7.2500 S ...
22 1 1 ... 7.2500 S ...
23 1 2 ... 71.2833 C ...
24 3 1 ... 7.9250 S ...
25 0 3 ... 7.2500 S ...
26 1 2 ... 71.2833 C ...
27 2 3 ... 7.9250 S ...
28 0 3 ... 7.2500 S ...
29 1 1 ... 7.2500 S ...
30 1 2 ... 71.2833 C ...
31 3 1 ... 7.9250 S ...
32 0 3 ... 7.2500 S ...
33 1 2 ... 71.2833 C ...
34 2 3 ... 7.9250 S ...
35 0 3 ... 7.2500 S ...
36 1 1 ... 7.2500 S ...
37 1 2 ... 71.2833 C ...
38 3 1 ... 7.9250 S ...
39 0 3 ... 7.2500 S ...
40 1 2 ... 71.2833 C ...
41 2 3 ... 7.9250 S ...
42 0 3 ... 7.2500 S ...
43 1 1 ... 7.2500 S ...
44 1 2 ... 71.2833 C ...
45 3 1 ... 7.9250 S ...
46 0 3 ... 7.2500 S ...
47 1 2 ... 71.2833 C ...
48 2 3 ... 7.9250 S ...
49 0 3 ... 7.2500 S ...
50 1 1 ... 7.2500 S ...
51 1 2 ... 71.2833 C ...
52 3 1 ... 7.9250 S ...
53 0 3 ... 7.2500 S ...
54 1 2 ... 71.2833 C ...
55 2 3 ... 7.9250 S ...
56 0 3 ... 7.2500 S ...
57 1 1 ... 7.2500 S ...
58 1 2 ... 71.2833 C ...
59 3 1 ... 7.9250 S ...
60 0 3 ... 7.2500 S ...
61 1 2 ... 71.2833 C ...
62 2 3 ... 7.9250 S ...
63 0 3 ... 7.2500 S ...
64 1 1 ... 7.2500 S ...
65 1 2 ... 71.2833 C ...
66 3 1 ... 7.9250 S ...
67 0 3 ... 7.2500 S ...
68 1 2 ... 71.2833 C ...
69 2 3 ... 7.9250 S ...
70 0 3 ... 7.2500 S ...
71 1 1 ... 7.2500 S ...
72 1 2 ... 71.2833 C ...
73 3 1 ... 7.9250 S ...
74 0 3 ... 7.2500 S ...
75 1 2 ... 71.2833 C ...
76 2 3 ... 7.9250 S ...
77 0 3 ... 7.2500 S ...
78 1 1 ... 7.2500 S ...
79 1 2 ... 71.2833 C ...
80 3 1 ... 7.9250 S ...
81 0 3 ... 7.2500 S ...
82 1 2 ... 71.2833 C ...
83 2 3 ... 7.9250 S ...
84 0 3 ... 7.2500 S ...
85 1 1 ... 7.2500 S ...
86 1 2 ... 71.2833 C ...
87 3 1 ... 7.9250 S ...
88 0 3 ... 7.2500 S ...
89 1 2 ... 71.2833 C ...
90 2 3 ... 7.9250 S ...
91 0 3 ... 7.2500 S ...
92 1 1 ... 7.2500 S ...
93 1 2 ... 71.2833 C ...
94 3 1 ... 7.9250 S ...
95 0 3 ... 7.2500 S ...
96 1 2 ... 71.2833 C ...
97 2 3 ... 7.9250 S ...
98 0 3 ... 7.2500 S ...
99 1 1 ... 7.2500 S ...
100 1 2 ... 71.2833 C ...
101 3 1 ... 7.9250 S ...
102 0 3 ... 7.2500 S ...
103 1 2 ... 71.2833 C ...
104 2 3 ... 7.9250 S ...
105 0 3 ... 7.2500 S ...
106 1 1 ... 7.2500 S ...
107 1 2 ... 71.2833 C ...
108 3 1 ... 7.9250 S ...
109 0 3 ... 7.2500 S ...
110 1 2 ... 71.2833 C ...
111 2 3 ... 7.9250 S ...
112 0 3 ... 7.2500 S ...
113 1 1 ... 7.2500 S ...
114 1 2 ... 71.2833 C ...
115 3 1 ... 7.9250 S ...
116 0 3 ... 7.2500 S ...
117 1 2 ... 71.2833 C ...
118 2 3 ... 7.9250 S ...
119 0 3 ... 7.2500 S ...
120 1 1 ... 7.2500 S ...
121 1 2 ... 71.2833 C ...
122 3 1 ... 7.9250 S ...
123 0 3 ... 7.2500 S ...
124 1 2 ... 71.2833 C ...
125 2 3 ... 7.9250 S ...
126 0 3 ... 7.2500 S ...
127 1 1 ... 7.2500 S ...
128 1 2 ... 71.2833 C ...
129 3 1 ... 7.9250 S ...
130 0 3 ... 7.2500 S ...
131 1 2 ... 71.2833 C ...
132 2 3 ... 7.9250 S ...
133 0 3 ... 7.2500 S ...
134 1 1 ... 7.2500 S ...
135 1 2 ... 71.2833 C ...
136 3 1 ... 7.9250 S ...
137 0 3 ... 7.2500 S ...
138 1 2 ... 71.2833 C ...
139 2 3 ... 7.9250 S ...
140 0 3 ... 7.2500 S ...
141 1 1 ... 7.2500 S ...
142 1 2 ... 71.2833 C ...
143 3 1 ... 7.9250 S ...
144 0 3 ... 7.2500 S ...
145 1 2 ... 71.2833 C ...
146 2 3 ... 7.9250 S ...
147 0 3 ... 7.2500 S ...
148 1 1 ... 7.2500 S ...
149 1 2 ... 71.2833 C ...
150 3 1 ... 7.9250 S ...
151 0 3 ... 7.2500 S ...
152 1 2 ... 71.2833 C ...
153 2 3 ... 7.9250 S ...
154 0 3 ... 7.2500 S ...
155 1 1 ... 7.2500 S ...
156 1 2 ... 71.2833 C ...
157 3 1 ... 7.9250 S ...
158 0 3 ... 7.2500 S ...
159 1 2 ... 71.2833 C ...
160 2 3 ... 7.9250 S ...
161 0 3 ... 7.2500 S ...
162 1 1 ... 7.2500 S ...
163 1 2 ... 71.2833 C ...
164 3 1 ... 7.9250 S ...
165 0 3 ... 7.2500 S ...
166 1 2 ... 71.2833 C ...
167 2 3 ... 7.9250 S ...
168 0 3 ... 7.2500 S ...
169 1 1 ... 7.2500 S ...
170 1 2 ... 71.2833 C ...
171 3 1 ... 7.9250 S ...
172 0 3 ... 7.2500 S ...
173 1 2 ... 71.2833 C ...
174 2 3 ... 7.9250 S ...
175 0 3 ... 7.2500 S ...
176 1 1 ... 7.2500 S ...
177 1 2 ... 71.2833 C ...
178 3 1 ... 7.9250 S ...
179 0 3 ... 7.2500 S ...
180 1 2 ... 71.2833 C ...
181 2 3 ... 7.9250 S ...
182 0 3 ... 7.2500 S ...
183 1 1 ... 7.2500 S ...
184 1 2 ... 71.2833 C ...
185 3 1 ... 7.9250 S ...
186 0 3 ... 7.2500 S ...
187 1 2 ... 71.2833 C ...
188 2 3 ... 7.9250 S ...
189 0 3 ... 7.2500 S ...
190 1 1 ... 7.2500 S ...
191 1 2 ... 71.2833 C ...
192 3 1 ... 7.9250 S ...
193 0 3 ... 7.2500 S ...
194 1 2 ... 71.2833 C ...
195 2 3 ... 7.9250 S ...
196 0 3 ... 7.2500 S ...
197 1 1 ... 7.2500 S ...
198 1 2 ... 71.2833 C ...
199 3 1 ... 7.9250 S ...
200 0 3 ... 7.2500 S ...
201 1 2 ... 71.2833 C ...
202 2 3 ... 7.9250 S ...
203 0 3 ... 7.2500 S ...
204 1 1 ... 7.2500 S ...
205 1 2 ... 71.2833 C ...
206 3 1 ... 7.9250 S ...
207 0 3 ... 7.2500 S ...
208 1 2 ... 71.2833 C ...
209 2 3 ... 7.9250 S ...
210 0 3 ... 7.2500 S ...
211 1 1 ... 7.2500 S ...
212 1 2 ... 71.2833 C ...
213 3 1 ... 7.9250 S ...
214 0 3 ... 7.2500 S ...
215 1 2 ... 71.2833 C ...
216 2 3 ... 7.9250 S ...
217 0 3 ... 7.2500 S ...
218 1 1 ... 7.2500 S ...
219 1 2 ... 71.2833 C ...
220 3 1 ... 7.9250 S ...
221 0 3 ... 7.2500 S ...
222 1 2 ... 71.2833 C ...
223 2 3 ... 7.9250 S ...
224 0 3 ... 7.2500 S ...
225 1 1 ... 7.2500 S ...
226 1 2 ... 71.2833 C ...
227 3 1 ... 7.9250 S ...
228 0 3 ... 7.2500 S ...
229 1 2 ... 71.2833 C ...
230 2 3 ... 7.9250 S ...
231 0 3 ... 7.2500 S ...
232 1 1 ... 7.2500 S ...
233 1 2 ... 71.2833 C ...
234 3 1 ... 7.9250 S ...
235 0 3 ... 7.2500 S ...
236 1 2 ... 71.2833 C ...
237 2 3 ... 7.9250 S ...
238 0 3 ... 7.2500 S ...
239 1 1 ... 7.2500 S ...
240 1 2 ... 71.2833 C ...
241 3 1 ... 7.9250 S ...
242 0 3 ... 7.2500 S ...
243 1 2 ... 71.2833 C ...
244 2 3 ... 7.9250 S ...
245 0 3 ... 7.2500 S ...
246 1 1 ... 7.2500 S ...
247 1 2 ... 71.2833 C ...
248 3 1 ... 7.9250 S ...
249 0 3 ... 7.2500 S ...
250 1 2 ... 71.2833 C ...
251 2 3 ... 7.9250 S ...
252 0 3 ... 7.2500 S ...
253 1 1 ... 7.2500 S ...
254 1 2 ... 71.2833 C ...
255 3 1 ... 7.9250 S ...
256 0 3 ... 7.2500 S ...
257 1 2 ... 71.2833 C ...
258 2 3 ... 7.9250 S ...
259 0 3 ... 7.2500 S ...
260 1 1 ... 7.2500 S ...
261 1 2 ... 71.2833 C ...
262 3 1 ... 7.9250 S ...
263 0 3 ... 7.2500 S ...
264 1 2 ... 71.2833 C ...
265 2 3 ... 7.9250 S ...
266 0 3 ... 7.2500 S ...
267 1 1 ... 7.2500 S ...
268 1 2 ... 71.2833 C ...
269 3 1 ... 7.9250 S ...
270 0 3 ... 7.2500 S ...
271 1 2 ... 71.2833 C ...
272 2 3 ... 7.9250 S ...
273 0 3 ... 7.2500 S ...
274 1 1 ... 7.2500 S ...
275 1 2 ... 71.2833 C ...
276 3 1 ... 7.9250 S ...
277 0 3 ... 7.2500 S ...
278 1 2 ... 71.2833 C ...
279 2 3 ... 7.9250 S ...
280 0 3 ... 7.2500 S ...
281 1 1 ... 7.2500 S ...
282 1 2 ... 71.2833 C ...
283 3 1 ... 7.9250 S ...
284 0 3 ... 7.2500 S ...
285 1 2 ... 71.2833 C ...
286 2 3 ... 7.9250 S ...
287 0 3 ... 7.2500 S ...
288 1 1 ... 7.2500 S ...
289 1 2 ... 71.2833 C ...
290 3 1 ... 7.9250 S ...
291 0 3 ... 7.2500 S ...
292 1 2 ... 71.2833 C ...
293 2 3 ... 7.9250 S ...
294 0 3 ... 7.2500 S ...
295 1 1 ... 7.2500 S ...
296 1 2 ... 71.2833 C ...
297 3 1 ... 7.9250 S ...
298 0 3 ... 7.2500 S ...
299 1 2 ... 71.2833 C ...
300 2 3 ... 7.9250 S ...
301 0 3 ... 7.2500 S ...
302 1 1 ... 7.2500 S ...
303 1 2 ... 71.2833 C ...
304 3 1 ... 7.9250 S ...
305 0 3 ... 7.2500 S ...
306 1 2 ... 71.2833 C ...
307 2 3 ... 7.9250 S ...
308 0 3 ... 7.2500 S ...
309 1 1 ... 7.2500 S ...
310 1 2 ... 71.2833 C ...
311 3 1 ... 7.9250 S ...
312 0 3 ... 7.2500 S ...
313 1 2 ... 71.2833 C ...
314 2 3 ... 7.9250 S ...
315 0 3 ... 7.2500 S ...
316 1 1 ... 7.2500 S ...
317 1 2 ... 71.2833 C ...
318 3 1 ... 7.9250 S ...
319 0 3 ... 7.2500 S ...
320 1 2 ... 71.2833 C ...
321 2 3 ... 7.9250 S ...
322 0 3 ... 7.2500 S ...
323 1 1 ... 7.2500 S ...
324 1 2 ... 71.2833 C ...
325 3 1 ... 7.9250 S ...
326 0 3 ... 7.2500 S ...
327 1 2 ... 71.2833 C ...
328 2 3 ... 7.9250 S ...
329 0 3 ... 7.2500 S ...
330 1 1 ... 7.2500 S ...
331 1 2 ... 71.2833 C ...
332 3 1 ... 7.9250 S ...
333 0 3 ... 7.2500 S ...
334 1 2 ... 71.2833 C ...
335 2 3 ... 7.9250 S ...
336 0 3 ... 7.2500 S ...
337 1 1 ... 7.2500 S ...
338 1 2 ... 71.2833 C ...
339 3 1 ... 7.9250 S ...
340 0 3 ... 7.2500 S ...
341 1 2 ... 71.2833 C ...
342 2 3 ... 7.9250 S ...
343 0 3 ... 7.2500 S ...
344 1 1 ... 7.2500 S ...
345 1 2 ... 71.2833 C ...
346 3 1 ... 7.9250 S ...
347 0 3 ... 7.2500 S ...
348 1 2 ... 71.2833 C ...
349 2 3 ... 7.9250 S ...
350 0 3 ... 7.2500 S ...
351 1 1 ... 7.2500 S ...
352 1 2 ... 71.2833 C ...
353 3 1 ... 7.9250 S ...
354 0 3 ... 7.2500 S ...
355 1 2 ... 71.2833 C ...
356 2 3 ... 7.9250 S ...
357 0 3 ... 7.2500 S ...
358 1 1 ... 7.2500 S ...
359 1 2 ... 71.2833 C ...
360 3 1 ... 7.9250 S ...
361 0 3 ... 7.2500 S ...
362 1 2 ... 71.2833 C ...
363 2 3 ... 7.9250 S ...
364 0 3 ... 7.2500 S ...
365 1 1 ... 7.2500 S ...
366 1 2 ... 71.2833 C ...
367 3 1 ... 7.9250 S ...
368 0 3 ... 7.2500 S ...
369 1 2 ... 71.2833 C ...
370 2 3 ... 7.9250 S ...
371 0 3 ... 7.2500 S ...
372 1 1 ... 7.2500 S ...
373 1 2 ... 71.2833 C ...
374 3 1 ... 7.9250 S ...
375 0 3 ... 7.2500 S ...
376 1 2 ... 71.2833 C ...
377 2 3 ... 7.9250 S ...
378 0 3 ... 7.2500 S ...
379 1 1 ... 7.2500 S ...
380 1 2 ... 71.2833 C ...
381 3 1 ... 7.9250 S ...
382 0 3 ... 7.2500 S ...
383 1 2 ... 71.2833 C ...
384 2 3 ... 7.9250 S ...
385 0 3 ... 7.2500 S ...
386 1 1 ... 7.2500 S ...
387 1 2 ... 71.2833 C ...
388 3 1 ... 7.9250 S ...
389 0 3 ... 7.2500 S ...
390 1 2 ... 71.2833 C ...
391 2 3 ... 7.9250 S ...
392 0 3 ... 7.2500 S ...
393 1 1 ... 7.2500 S ...
394 1 2 ... 71.2833 C ...
395 3 1 ... 7.9250 S ...
396 0 3 ... 7.2500 S ...
397 1 2 ... 71.2833 C ...
398 2 3 ... 7.9250 S ...
399 0 3 ... 7.2500 S ...
400 1 1 ... 7.2500 S ...
401 1 2 ... 71.2833 C ...
402 3 1 ... 7.9250 S ...
403 0 3 ... 7.2500 S ...
404 1 2 ... 71.2833 C ...
405 2 3 ... 7.9250 S ...
406 0 3 ... 7.2500 S ...
407 1 1 ... 7.2500 S ...
408 1 2 ... 71.2833 C ...
409 3 1 ... 7.9250 S ...
410 0 3 ... 7.2500 S ...
411 1 2 ... 71.2833 C ...
412 2 3 ... 7.9250 S ...
413 0 3 ... 7.2500 S ...
414 1 1 ... 7.2500 S ...
415 1 2 ... 71.2833 C ...
416 3 1 ... 7.9250 S ...
417 0 3 ... 7.2500 S ...
418 1 2 ... 71.2833 C ...
419 2 3 ... 7.9250 S ...
420 0 3 ... 7.2500 S ...
421 1 1 ... 7.2500 S ...
422 1 2 ... 71.2833 C ...
423 3 1 ... 7.9250 S ...
424 0 3 ... 7.2500 S ...
425 1 2 ... 71.2833 C ...
426 2 3 ... 7.9250 S ...
427 0 3 ... 7.2500 S ...
428 1 1 ... 7.2500 S ...
429 1 2 ... 71.2833 C ...
430 3 1 ... 7.9250 S ...
431 0 3 ... 7.2500 S ...
432 1 2 ... 71.2833 C ...
433 2 3 ... 7.9250 S ...
434 0 3 ... 7.2500 S ...
435 1 1 ... 7.2500 S ...
436 1 2 ... 71.2833 C ...
437 3 1 ... 7.9250 S ...
438 0 3 ... 7.2500 S ...
439 1 2 ... 71.2833 C ...
440 2 3 ... 7.9250 S ...
441 0 3 ... 7.2500 S ...
442 1 1 ... 7.2500 S ...
443 1 2 ... 71.2833 C ...
444 3 1 ... 7.9250 S ...
445 0 3 ... 7.2500 S ...
446 1 2 ... 71.2833 C ...
447 2 3 ... 7.9250 S ...
448 0 3 ... 7.2500 S ...
449 1 1 ... 7.2500 S ...
450 1 2 ... 71.2833 C ...
451 3 1 ... 7.9250 S ...
452 0 3 ... 7.2500 S ...
453 1 2 ... 71.2833 C ...
454 2 3 ... 7.9250 S ...
455 0 3 ... 7.2500 S ...
456 1 1 ... 7.2500 S ...
457 1 2 ... 71.2833 C ...
458 3 1 ... 7.9250 S ...
459 0 3 ... 7.2500 S ...
460 1 2 ... 71.2833 C ...
461 2 3 ... 7.9250 S ...
462 0 3 ... 7.2500 S ...
463 1 1 ... 7.2500 S ...
464 1 2 ... 71.2833 C ...
465 3 1 ... 7.9250 S ...
466 0 3 ... 7.2500 S ...
467 1 2 ... 71.2833 C ...
468 2 3 ... 7.9250 S ...
469 0 3 ... 7.2500 S ...
470 1 1 ... 7.2500 S ...
471 1 2 ... 71.2833 C ...
472 3 1 ... 7.9250 S ...
473 0 3 ... 7.2500 S ...
474 1 2 ... 71.2833 C ...
475 2 3 ... 7.9250 S ...
476 0 3 ... 7.2500 S ...
477 1 1 ... 7.2500 S ...
478 1 2 ... 71.2833 C ...
479 3 1 ... 7.9250 S ...
480 0 3 ... 7.2500 S ...
481 1 2 ... 71.2833 C ...
482 2 3 ... 7.9250 S ...
483 0 3 ... 7.2500 S ...
484 1 1 ... 7.2500 S ...
485 1 2 ... 71.2833 C ...
486 3 1 ... 7.9250 S ...
487 0 3 ... 7.2500 S ...
488 1 2 ... 71.2833 C ...
489 2 3 ... 7.9250 S ...
490 0 3 ... 7.2500 S ...
491 1 1 ... 7.2500 S ...
492 1 2 ... 71.2833 C ...
493 3 1 ... 7.9250 S ...
494 0 3 ... 7.2500 S ...
495 1 2 ... 71.2833 C ...
496 2 3 ... 7.9250 S ...
497 0 3 ... 7.2500 S ...
498 1 1 ... 7.2500 S ...
499 1

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.

- Create a new column 'IsAlone' which is 1 if the passenger is alone (FamilySize = 0), otherwise 0.
- Convert the 'Sex' column to numeric values (male: 0, female: 1).
- One-hot encode the 'Embarked' column, dropping the first category.
- Get the mean age of passengers.
- Get the median fare of passengers.
- Get the number of passengers by class.
- Get the number of passengers by gender.
- Get the number of passengers by survival status.
- Calculate the survival rate of passengers.
- Calculate the survival rate by gender.

The Titanic dataset contains columns as shown below,

P	S	C	Q	E
---	---	---	---	---

Sample Test Cases

titanicDat...

```
1 import pandas as pd
2 import numpy as np
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6 data['FamilySize'] = data['SibSp'] + data['Parch']
7
8 # 1. Create a new column 'IsAlone' (1 if alone, 0 if not)
```

Average time: 0.073 s / Maximum time: 0.073 s / 73.00 ms

1 out of 1 shown test case(s) passed

Test case 1 73 ms

Expected output	Actual output
29.69911764705882	29.69911764705882
14.4542	14.4542
3...491	3...491
1...216	1...216
2...184	2...184
Name: Pclass, dtype: int64	Name: Pclass, dtype: int64

Debug Test cases Terminal

< Prev Reset Submit Next >

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.

- Calculate the survival rate by class.
- Calculate the survival rate by embarkation location (Embarked_S).
- Calculate the survival rate by family size (FamilySize).
- Calculate the survival rate by being alone (IsAlone).
- Get the average fare by passenger class (Pclass).
- Get the average age by passenger class (Pclass).
- Get the average age by survival status (Survived).
- Get the average fare by survival status (Survived).
- Get the number of survivors by class (Pclass).
- Get the number of non-survivors by class (Pclass).

The Titanic dataset contains columns as shown below,

P	S	C	Q	E
---	---	---	---	---

Sample Test Cases

titanicDat...

```
1 import pandas as pd
2 import numpy as np
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6 data['FamilySize'] = data['SibSp'] + data['Parch']
7 data['IsAlone'] = np.where(data['FamilySize'] > 0, 0, 1)
8 data = pd.get_dummies(data, columns=['Embarked'], dropna=True)
```

Average time: 0.088 s / Maximum time: 0.088 s / 88.00 ms

1 out of 1 shown test case(s) passed

Test case 1 88 ms

Expected output	Actual output
Pclass	Pclass
1...0.629630	1...0.629630
2...0.472826	2...0.472826
3...0.242363	3...0.242363
Name: Survived, dtype: float64	Name: Survived, dtype: float64
Embarked_S	Embarked_S

Debug Test cases Terminal

< Prev Reset Submit Next >

photos - Google Drive | Apple Music - Web Playe Course | 2304102 ALL PR: Public L | Upload files - nishaarika-0 | + | - | X | mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e455f1f9c5320ca6bd19/6773e4e4f1f9c5320ca6bdce/67e... | N | : | Logout

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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.

The Titanic dataset contains columns as shown below,

P	S							E
---	---	--	--	--	--	--	--	---

Sample Test Cases +

Explorer titanicData...

```
1 import pandas as pd
2 import numpy as np
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6 data = pd.get_dummies(data, columns=['Embarked'], drop_first=True)
7 # 1. Get the number of survivors by gender
```

Average time Maximum time
0.085 s 0.085 s
85.00 ms 85.00 ms

1 out of 1 shown test case(s) passed

Test case 1 85 ms

Expected output	Actual output
female.....233	female.....233
male.....109	male.....109
Name: Sex, dtype: int64	Name: Sex, dtype: int64
male.....468	male.....468
female.....81	female.....81
Name: Sex, dtype: int64	Name: Sex, dtype: int64

Terminal Test cases < Prev Reset Submit Next >

ENG IN 21:45 06-05-2025

PRACTICAL 5:

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e45af1f9c5320ca6bd1d/6773e4f9f1f9c5320ca6bdd2/677... [Logout](#)

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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.

Sample Test Cases

Stacked Plot...

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

# Data for Months and Temperature for three cities
data = {
    'Month': ['January', 'February', 'March', 'April',
              'May', 'June', 'July', 'August', 'September', 'October',
              'November', 'December'],
    'City A': [30, 32, 35, 38, 40, 42, 45, 48, 50, 52, 55, 58],
    'City B': [25, 28, 30, 32, 35, 38, 40, 42, 45, 48, 50, 52],
    'City C': [20, 22, 25, 28, 30, 32, 35, 38, 40, 42, 45, 48]
}
```

Average time: 0.613 s Maximum time: 0.613 s 613.00 ms 1 out of 1 shown test case(s) passed

Test case 1 613 ms

Expected output Actual output

Terminal Test cases

< Prev Reset Submit Next >

21:46 06-05-2025

mitaoe.codetantra.com/secure/course.jsp?euclid=6773e3f2f1f9c5320ca6bc85#/contents/6773e45af1f9c5320ca6bd1d/6773e4f9f1f9c5320ca6bdd2/677... [Logout](#)

CODETANTRA Home

202401090024@mitaoe.ac.in Support Logout

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).

Sample Test Cases

titanicData...

```
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))
```

Average time: 1.859 s Maximum time: 1.859 s 1859.00 ms 1 out of 1 shown test case(s) passed

Test case 1 1859 ms

Expected output Actual output

Terminal Test cases

< Prev Reset Submit Next >

21:46 06-05-2025

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.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

Histogram...

```
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)
```

Average time: 0.554 s | Maximum time: 0.554 s | 554.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 | 554 ms | Expected output | Actual output

Age Distribution

250
200
150
100
50
0

235

250
200
150
100
50
0

235

Terminal | Test cases | < Prev | Reset | Submit | Next >

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**'.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

BarPlotOf...

```
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)
```

Average time: 0.529 s | Maximum time: 0.529 s | 529.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 | 529 ms | Expected output | Actual output

Survival Count

500
400
300
200
100
0

520

500
400
300
200
100
0

520

Terminal | Test cases | < Prev | Reset | Submit | Next >

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'.

The Titanic dataset contains columns as shown below,

PasengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Class	Embarked
1	0	1	Allen, Mr. William Henry	male	30	1	0	A/5 21171	1	STO
2	1	1	Allen, Mrs. Ernestine	female	31	0	0	349308	1	STO
3	0	3	Anderson, Mr. Harold	male	39	0	0	347349	3	STO
4	1	3	Anderson, Mrs. Harold	female	40	0	0	347350	3	STO
5	0	1	Anglicus, Mr. James	male	22	1	2	347360	1	STO
6	1	1	Anglicus, Mrs. James	female	23	1	2	347361	1	STO
7	0	3	Archibald, Mr. Fredrick	male	35	0	0	347362	3	STO
8	1	3	Archibald, Mrs. Fredrick	female	35	0	0	347363	3	STO
9	0	1	Archibald, Miss. Mary	female	23	0	0	347364	1	STO
10	1	1	Archibald, Mr. William	male	35	0	0	347365	1	STO

Sample Test Cases

BarPlotOr...
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)

Average time: 0.767 s Maximum time: 0.767 s 767.00 ms 1 out of 1 shown test case(s) passed

Test case 1 767 ms
Expected output
Actual output

Terminal Test cases

< Prev Reset Submit Next >

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'.

The Titanic dataset contains columns as shown below,

PasengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Class	Embarked
1	0	1	Allen, Mr. William Henry	male	30	1	0	A/5 21171	1	STO
2	1	1	Allen, Mrs. Ernestine	female	31	0	0	349308	1	STO
3	0	3	Anderson, Mr. Harold	male	39	0	0	347349	3	STO
4	1	3	Anderson, Mrs. Harold	female	40	0	0	347350	3	STO
5	0	1	Anglicus, Mr. James	male	22	1	2	347360	1	STO
6	1	1	Anglicus, Mrs. James	female	23	1	2	347361	1	STO
7	0	3	Archibald, Mr. Fredrick	male	35	0	0	347362	3	STO
8	1	3	Archibald, Mrs. Fredrick	female	35	0	0	347363	3	STO
9	0	1	Archibald, Miss. Mary	female	23	0	0	347364	1	STO
10	1	1	Archibald, Mr. William	male	35	0	0	347365	1	STO

Sample Test Cases

BarPlotOr...
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)

Average time: 0.773 s Maximum time: 0.773 s 773.00 ms 1 out of 1 shown test case(s) passed

Test case 1 773 ms
Expected output
Actual output

Terminal Test cases

< Prev Reset Submit Next >

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').

The Titanic dataset contains columns as shown below,

P as se r e d	S ur vi ve d	P cl as s	N a m e	S ex	A ge	Si b S p	P ar ch	Ti ck et	F ar e	C ab in	E m ba rk ed

Sample Test Cases

Code Editor (Python):

```

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)

```

Execution Results:

Average time: 0.647 s | Maximum time: 0.647 s | 647.00 ms | 1 out of 1 shown test case(s) passed

Test case 1: 647 ms

Expected output | Actual output

Terminal | Test cases | < Prev | Reset | Submit | Next >

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'.

The Titanic dataset contains columns as shown below,

P as se r e d Id	S ur vi ve d	P cl as s	N a m e	S ex	A ge	Si b S p	P ar ch	Ti ck et	F ar e	C ab in	E m ba rk ed

Sample Test Cases

Code Editor (Python):

```

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)

```

Execution Results:

Average time: 0.625 s | Maximum time: 0.625 s | 625.00 ms | 1 out of 1 shown test case(s) passed

Test case 1: 625 ms

Expected output | Actual output

Terminal | Test cases | < Prev | Reset | Submit | Next >

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**'.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

Code Editor (BoxPlotF...)

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6
7 # Data Cleaning
8 data['Age'].fillna(data['Age'].median(), inplace=True)

```

Average time: 0.545 s / 545.00 ms | Maximum time: 0.545 s / 545.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 (545 ms)

Expected output

Actual output

Terminal | Test cases | < Prev | Reset | Submit | Next >

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**'.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

Code Editor (BoxPlotF...)

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6
7 # Data Cleaning
8 data['Age'].fillna(data['Age'].median(), inplace=True)

```

Average time: 0.634 s / 634.00 ms | Maximum time: 0.634 s / 634.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 (634 ms)

Expected output

Actual output

Terminal | Test cases | < Prev | Reset | Submit | Next >

Course

2304102 ALL PR: Public L | Upload files - niharika-0 | +

Logout

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'.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

Submit

Explorer

```
AgeFareS...
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6
7 # Data Cleaning
8 data['Age'].fillna(data['Age'].median(), inplace=True)
```

Average time: 0.475 s / 475.00 ms | Maximum time: 0.475 s / 475.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 (475 ms)

Expected output

Actual output

Terminal

Test cases

< Prev Reset Submit Next >

21:48 06-05-2025

photos - Google Drive | Apple Music - Web Playe | Course | 2304102 ALL PR: Public L | Upload files - niharika-0 | +

Logout

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'.

The Titanic dataset contains columns as shown below,

Pasenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Sample Test Cases

Submit

Explorer

```
AgeFareS...
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Load the Titanic dataset
5 data = pd.read_csv('Titanic-Dataset.csv')
6
7 # Data Cleaning
8 data['Age'].fillna(data['Age'].median(), inplace=True)
```

Average time: 0.517 s / 517.00 ms | Maximum time: 0.517 s / 517.00 ms | 1 out of 1 shown test case(s) passed

Test case 1 (517 ms)

Expected output

Actual output

Terminal

Test cases

< Prev Reset Submit Next >