Rajalakshmi Engineering College

Name: bhagawath narayanan n
Email: 241501034@rajalakshmi.edu.in
Roll no: 241501034
Phone: 6374835866
Branch: REC
Department: I AIML AD
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 7_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 46.5

Section 1 : Coding

1. Problem Statement

Arjun manages a busy customer service center and wants to analyze the distribution of customer wait times to improve service efficiency. He decides to group the wait times into intervals of 5 minutes each and count how many customers fall into each interval bucket.

Help him implement this bucketing and counting task using NumPy.

Bucketing Logic:

Divide the wait times into intervals (buckets) of size 5 minutes, e.g.:

[0–5), [5–10), [10–15), …

Use NumPy's digitize function to determine which bucket each wait time falls into.

Count the number of wait times in each bucket and generate bucket labels.

Input Format

The first line contains an integer n, the number of customer wait times recorded.

The second line contains n space-separated floating-point numbers representing the wait times (in minutes).

Output Format

The first line of output is the text:

Wait Time Buckets and Counts:

Each subsequent line prints the bucket range and the number of wait times in that bucket, formatted as:

<bucket_range>: <count>

where <bucket_range> is the lower and upper bound of the bucket (inclusive lower bound, exclusive upper bound), for example:

0-5: 3

5-10: 2

10-15: 1

The output uses the default string formatting of Python's print() function (no extra spaces, no special formatting beyond the specified lines).

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 10
2.0 3.0 7.0 8.0 12.0 14.0 18.0 19.0 21.0 25.0

Output: Wait Time Buckets and Counts:
0-5: 2
5-10: 2
10-15: 2
15-20: 2
20-25: 1

Answer

```
import math

def generate_wait_time_buckets(wait_times):
    if not wait_times:
        return {}

    bucket_size = 5.0
    max_wait_time = max(wait_times)

    num_buckets_to_generate = math.ceil(max_wait_time / bucket_size)

    bucket_counts = [0] * int(num_buckets_to_generate)

    for wait_time in wait_times:
        bucket_index = math.floor(wait_time / bucket_size)

        if bucket_index < num_buckets_to_generate:
            bucket_counts[int(bucket_index)] += 1

    formatted_buckets = {}
    for i in range(int(num_buckets_to_generate)):
        lower_bound = i * bucket_size
        upper_bound = (i + 1) * bucket_size
        bucket_label = f"{int(lower_bound)}-{int(upper_bound)}"
        formatted_buckets[bucket_label] = bucket_counts[i]

    return formatted_buckets

def solve():
    try:
        n = int(input())
        if not (1 <= n <= 100):
            print("Constraint violation: n must be between 1 and 100.")
            return
```

```
        wait_times_str = input().split()
        wait_times = [float(time) for time in wait_times_str]

        if len(wait_times) != n:
            print("Error: Number of wait times does not match n.")
            return

        for wt in wait_times:
            if wt < 0:
                print("Constraint violation: Wait times must be non-negative.")
                return

        bucket_results = generate_wait_time_buckets(wait_times)

        print("Wait Time Buckets and Counts:")

        if wait_times:
            bucket_size = 5.0
            max_wait_time = max(wait_times)

            num_buckets_to_print = math.ceil(max_wait_time / bucket_size)

            for i in range(int(num_buckets_to_print)):
                lower_bound = i * bucket_size
                upper_bound = (i + 1) * bucket_size
                bucket_label = f"{int(lower_bound)}-{int(upper_bound)}"
                print(f"{bucket_label}: {bucket_results.get(bucket_label, 0)}")
        else:
            pass

    except ValueError:
        print("Invalid input. Please ensure n is an integer and wait times are valid
floating-point numbers.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    solve()
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Arjun is a data scientist working on an image processing task. He needs to normalize the pixel values of a grayscale image matrix to scale between 0 and 1. The input image data is provided as a matrix of integers.

Help him to implement the task using the numpy package.

Formula:

To normalize each pixel value in the image matrix:

normalized_pixel = (pixel - min_pixel) / (max_pixel - min_pixel)

where min_pixel and max_pixel are the minimum and maximum pixel values in the image matrix, respectively. If all pixel values are the same, the normalized image matrix should be filled with zeros.

Input Format

The first line of input consists of an integer value, rows, representing the number of rows in the image matrix.

The second line of input consists of an integer value, cols, representing the number of columns in the image matrix.

The next rows lines each consist of cols integer values separated by a space, representing the pixel values of the image matrix.

Output Format

The output prints: normalized_image

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 2
3
1 2 3
4 5 6

Output: [[0. 0.2 0.4]
 [0.6 0.8 1.]]

Answer

```
import numpy as np

def normalize_image(image_matrix):
    min_pixel = np.min(image_matrix)
    max_pixel = np.max(image_matrix)

    if max_pixel == min_pixel:
        normalized_image = np.zeros_like(image_matrix, dtype=float)
    else:
        normalized_image = (image_matrix - min_pixel) / (max_pixel - min_pixel)

    return normalized_image

def solve():
    rows = int(input())
    cols = int(input())

    image_data = []
    for _ in range(rows):
        row_values = list(map(int, input().split()))
        image_data.append(row_values)

    image_matrix = np.array(image_data, dtype=float)

    normalized_result = normalize_image(image_matrix)

    print("[", end="")
    for i, row in enumerate(normalized_result):
        if i > 0:
            print("\n ", end="")

        print("[", end="")
        for j, pixel_val in enumerate(row):
            # Custom formatting to match the expected output:
            # If the pixel value is a whole number (0.0 or 1.0), print with only one
decimal place.
            # Otherwise, print with more precision (e.g., 0.2, 0.4, 0.6, 0.8).
            # For 0.0, print "0."
```

```
            # For 1.0, print "1."
            if pixel_val == 0.0:
                print("0.", end="")
            elif pixel_val == 1.0:
                print("1.", end="")
            else:
                # For intermediate values, use f-string with .1f which correctly handles
0.2, 0.4 etc.
                # However, for values like 0.0 it would print 0.0, so the above checks are
needed.
                print(f"{pixel_val:.1f}", end="")
                # If there are values like 0.123, and you need 0.1, then .1f is fine.
                # If you need to match exactly 0.2, 0.4 etc., the .1f is generally
appropriate.

            if j < len(row) - 1:
                print(" ", end="")
        print("]", end="")
    print("]")


if __name__ == "__main__":
    solve()
```

Status : Partially correct **Marks : 7.5/10**

3. Problem Statement

You're analyzing the daily returns of a set of financial assets over a period of time. Each day is represented as a row in a 2D array, where each column represents the return of a specific asset on that day.

Your task is to identify which days had all positive returns across every asset using numpy, and output a boolean array indicating these days.

Input Format

The first line of input consists of two integer values, rows and cols, separated by a space.

Each of the next rows lines consists of cols float values representing the returns

of the assets for that day.

Output Format

The first line of output prints: "Days where all asset returns were positive:"

The second line of output prints: the boolean array positive_days, indicating True for days where all asset returns were positive and False otherwise.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3 4
0.01 0.02 0.03 0.04
0.05 0.06 0.07 0.08
-0.01 0.02 0.03 0.04
Output: Days where all asset returns were positive:
[True True False]

Answer

```
# You are using Python
import numpy as np

def analyze_positive_returns():
    # Read rows and columns
    rows, cols = map(int, input().split())

    # Initialize a list to store daily returns
    returns_data = []

    # Read daily returns
    for _ in range(rows):
        daily_returns = list(map(float, input().split()))
        returns_data.append(daily_returns)

    # Convert to a NumPy array
    returns_array = np.array(returns_data)

    # Check if all returns for each day are positive
    # (returns_array > 0) creates a boolean array where True indicates a positive
return
```

```
    # .all(axis=1) checks if all values in each row are True
    positive_days = (returns_array > 0).all(axis=1)

    # Print the output
    print("Days where all asset returns were positive:")
    print(positive_days)

# Call the function to execute the analysis
analyze_positive_returns()
```

Status : Correct **Marks : 10/10**

4. Problem Statement

A company conducted a customer satisfaction survey where each respondent provides their RespondentID and an optional textual Feedback. Sometimes, respondents submit their ID without any feedback or with empty feedback.

Your task is to process the survey responses using pandas to replace any missing or empty feedback with the phrase "No Response". Finally, print the cleaned survey responses exactly as shown in the sample output.

Input Format

The first line contains an integer n, the number of survey responses.

Each of the next n lines contains:

A RespondentID (a single alphanumeric string without spaces),

Followed optionally by a Feedback string, which may be empty or missing.

If no feedback is provided after the RespondentID, treat it as missing.

Output Format

Print the line:

Survey Responses with Missing Feedback Filled:

Then print the cleaned survey data as a table with two columns: RespondentID

and Feedback.

The table should have the headers exactly as:

RespondentID Feedback

Print each respondent's data on a new line, aligned to match the output produced by pandas.DataFrame.to_string(index=False).

For any missing or empty feedback, print "No Response" in the Feedback column.

Maintain the spacing and alignment exactly as shown in the sample outputs.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
101 Great service
102
103 Loved it
104

Output: Survey Responses with Missing Feedback Filled:

```
RespondentID     Feedback
         101 Great service
         102   No Response
         103      Loved it
         104   No Response
```

Answer

```
# You are using Python
import pandas as pd
import numpy as np
```

```
def process_survey_responses():
    n = int(input())

    respondent_ids = []
    feedbacks = []

    for _ in range(n):
        line = input()
        parts = line.split(maxsplit=1)

        respondent_id = parts[0]
        feedback = ""
        if len(parts) > 1:
            feedback = parts[1]

        respondent_ids.append(respondent_id)
        feedbacks.append(feedback)

    df = pd.DataFrame({
        'RespondentID': respondent_ids,
        'Feedback': feedbacks
    })

    # Replace empty strings with "No Response"
    df['Feedback'] = df['Feedback'].replace('', 'No Response')

    print("Survey Responses with Missing Feedback Filled:")
    print(df.to_string(index=False))

# Call the function to process the survey responses
process_survey_responses()
```

Status : Correct **Marks : 10/10**

5. Problem Statement

A software development company wants to classify its employees based on their years of service at the company. They want to categorize employees into three experience levels: Junior (less than 3 years), Mid (3 to 6 years, inclusive), and Senior (more than 6 years).

Experience Level Classification:

Junior: Years at Company < 3

Mid: 3 ≤ Years at Company < 6

Senior: Years at Company > 5

You need to create a Python program using the pandas library that reads employee data, processes it into a DataFrame, and adds a new column "Experience Level" to display the appropriate classification for each employee.

Input Format

First line: an integer n representing the number of employees.

Next n lines: each line has a string Name and a floating-point number Years at Company (space-separated).

Output Format

First line: "Employee Data with Experience Level:"

The employee data table printed with no index column, and with columns: Name, Years at Company, Experience Level.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
Alice 2
Bob 4
Charlie 7
Diana 3
Evan 6

Output: Employee Data with Experience Level:

Name	Years at Company	Experience Level
Alice	2.0	Junior
Bob	4.0	Mid

```
Charlie          7.0        Senior
  Diana          3.0           Mid
   Evan          6.0        Senior
```

Answer

```
# You are using Python
import pandas as pd
import io

def classify_employees():
    n = int(input())

    employee_data = []
    for _ in range(n):
        name, years_at_company = input().split()
        employee_data.append({'Name': name, 'Years at Company':
float(years_at_company)})

    df = pd.DataFrame(employee_data)

    # Define the classification logic
    def get_experience_level(years):
        if years < 3:
            return 'Junior'
        elif 3 <= years < 6:
            return 'Mid'
        else: # years >= 6
            return 'Senior'

    # Apply the classification to create the new 'Experience Level' column
    df['Experience Level'] = df['Years at Company'].apply(get_experience_level)

    print("Employee Data with Experience Level:")
    # Print the DataFrame without the index
    print(df.to_string(index=False))

classify_employees()
```

Status : Partially correct ***Marks : 9/10***