Rajalakshmi Engineering College

Name: Sudharsan R
Email: 240701543@rajalakshmi.edu.in
Roll no: 240701543
Phone: 9176060959
Branch: REC
Department: I CSE FF
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 7_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. 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 numpy as np

def classify_employees():
```

```
    # Read the number of employees
    n = int(input())

    # Initialize lists to store employee data
    names = []
    years_at_company = []

    # Read employee data
    for _ in range(n):
        line = input().split()
        name = line[0]
        years = float(line[1])
        names.append(name)
        years_at_company.append(years)

    # Create a pandas DataFrame
    employee_df = pd.DataFrame({
        'Name': names,
        'Years at Company': years_at_company
    })

    # Define the conditions for classification
    # Based on the problem statement and sample output:
    # Junior: Years at Company < 3
    # Mid: 3 <= Years at Company < 6
    # Senior: Years at Company >= 6
    conditions = [
        employee_df['Years at Company'] < 3,
        (employee_df['Years at Company'] >= 3) & (employee_df['Years at Company']
< 6),
        employee_df['Years at Company'] >= 6
    ]

    # Define the corresponding choices for each condition
    choices = ['Junior', 'Mid', 'Senior']

    # Use numpy.select to apply the conditions and create the 'Experience Level'
column
    employee_df['Experience Level'] = np.select(conditions, choices)

    # Print the specified header
    print("Employee Data with Experience Level:")
```

```
    # Print the DataFrame without the index
    print(employee_df.to_string(index=False))

# Call the function to execute the program
classify_employees()
```

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

2. 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

def process_survey_responses():
    # Read the number of survey responses
    n = int(input())
```

```
    # Initialize a list to store the parsed data
    survey_data = []

    # Process each survey response line
    for _ in range(n):
        line = input()

        # Split the line into RespondentID and Feedback.
        # Use split(' ', 1) to split only on the first space,
        # ensuring that feedback with spaces is handled correctly.
        parts = line.split(' ', 1)

        respondent_id = parts[0]
        feedback = ""

        # Check if there is a feedback part and if it's empty after stripping whitespace
        if len(parts) == 1:
            # If only one part, it means no feedback was provided after the RespondentID
            feedback = "No Response"
        else:
            # If there's a second part, it's the potential feedback string
            feedback_content = parts[1].strip() # Remove leading/trailing whitespace
            if feedback_content == "":
                # If the feedback content is empty after stripping, replace with "No Response"
                feedback = "No Response"
            else:
                # Otherwise, use the provided feedback
                feedback = feedback_content

        # Add the processed data to our list
        survey_data.append({'RespondentID': respondent_id, 'Feedback': feedback})

    # Create a pandas DataFrame from the processed data
    df = pd.DataFrame(survey_data)

    # Print the specified header
    print("Survey Responses with Missing Feedback Filled:")
```

```
    # Print the DataFrame without the index, maintaining the specified alignment
    # using to_string()
    print(df.to_string(index=False))

# Call the function to execute the program
process_survey_responses()
```

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

3. 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 numpy as np
import math

def analyze_wait_times():
    # Read the number of customer wait times
    n = int(input())
```

```
    # Handle the case where there are no wait times
    if n == 0:
        print("Wait Time Buckets and Counts:")
        return

    # Read the space-separated wait times and convert them to a NumPy array of
floats
    wait_times_str = input().split()

    # If the input line for wait times is empty, even if n > 0 (e.g., if n=1 but input is
just a newline)
    if not wait_times_str:
        print("Wait Time Buckets and Counts:")
        return

    wait_times = np.array([float(x) for x in wait_times_str])

    # Calculate the maximum wait time to determine the range of buckets needed
    max_val = np.max(wait_times)

    # Determine the upper exclusive limit for the bucket labels based on the
largest wait time.
    # This ensures that buckets are generated up to the nearest 5-minute interval
that covers max_val.
    # For example, if max_val is 21.0, the limit is 25. If max_val is 25.0, the limit is
25.
    # We also ensure that if max_val is 0, at least the 0-5 bucket is considered.
    if max_val == 0:
        upper_exclusive_limit_for_labels = 5
    else:
        # Calculate the smallest multiple of 5 that is greater than or equal to
max_val
        upper_exclusive_limit_for_labels = int(math.ceil(max_val / 5)) * 5
        # Ensure that if max_val itself is 0, we still target at least 5 for the upper limit
        upper_exclusive_limit_for_labels = max(5, upper_exclusive_limit_for_labels)

    # Calculate the raw bucket index for each wait time.
    # For a wait time 't', its bucket index is floor(t / 5).
    # e.g., 2.0 -> 0, 7.0 -> 1, 21.0 -> 4, 25.0 -> 5
    raw_bucket_indices = np.floor(wait_times / 5).astype(int)

    # Determine the highest bucket index that should be *displayed* in the output.
```

```
    # This is (upper_exclusive_limit_for_labels / 5) - 1.
    # For example, if upper_exclusive_limit_for_labels is 25 (meaning buckets up
to [20-25)),
    # the highest display index is 4 (for the 20-25 bucket).
    max_display_idx = int(upper_exclusive_limit_for_labels / 5) - 1

    # Filter the raw bucket indices to only include those that correspond to
    # the buckets that should be displayed.
    # This handles the case where a wait time is exactly at an upper boundary
    # (e.g., 25.0 in sample 1), preventing it from being counted in the
    # preceding bucket if the problem implies strict exclusion.
    # It also ensures indices are non-negative as per problem constraints.
    relevant_indices = raw_bucket_indices[(raw_bucket_indices >= 0) &
(raw_bucket_indices <= max_display_idx)]

    # Count the occurrences of each relevant bucket index.
    # np.bincount creates an array where each index corresponds to a bucket
index
    # and its value is the count of occurrences.
    # minlength ensures the array is long enough to include counts for all
    # buckets up to max_display_idx, even if some have 0 customers.
    if relevant_indices.size > 0:
        counts = np.bincount(relevant_indices, minlength=max_display_idx + 1)
    else:
        # If no wait times fall into the relevant display range, create an array of
zeros.
        counts = np.zeros(max_display_idx + 1, dtype=int)

    # Print the specified output header
    print("Wait Time Buckets and Counts:")

    # Iterate from bucket index 0 up to max_display_idx to print the results
    for i in range(max_display_idx + 1):
        lower_bound = i * 5
        upper_bound = (i + 1) * 5
        count = counts[i]
        # Print each bucket range and its count
        print(f"{lower_bound}-{upper_bound}: {count}")

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

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

4. 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

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

# Read number of rows and columns
rows = int(input())
cols = int(input())

# Read image matrix
data = [list(map(int, input().split())) for _ in range(rows)]
image = np.array(data, dtype=float)

# Find min and max pixel values
min_pixel = np.min(image)
max_pixel = np.max(image)

# Normalize using the given formula
if min_pixel == max_pixel:
    normalized_image = np.zeros((rows, cols))
else:
    normalized_image = (image - min_pixel) / (max_pixel - min_pixel)

print(normalized_image)
```

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

5. 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

# Read number of rows and columns
rows, cols = map(int, input().split())

# Read the returns matrix
data = [list(map(float, input().split())) for _ in range(rows)]
returns = np.array(data)

# Check for days where all returns are positive
positive_days = np.all(returns > 0, axis=1)

# Print the output as per the specified format
```

```
print("Days where all asset returns were positive:")
print(positive_days)
```

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