# Rajalakshmi Engineering College

Name: Rakshitha RD
Email: 241501163@rajalakshmi.edu.in
Roll no: 241501163
Phone: 9444821024
Branch: REC
Department: I AI & ML FB
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

## NeoColab_REC_CS23221_Python Programming

## REC_Python_Week 7_COD

Attempt : 1
Total Mark : 50
Marks Obtained : 50

## Section 1 : Coding

1. Problem Statement

Sita works as a sales analyst and needs to analyze monthly sales data for different cities. She receives lists of cities, months, and corresponding sales values and wants to create a pandas DataFrame using a MultiIndex of cities and months.

Help her to implement this task and calculate total sales for each city.

***Input Format***

The first line of input consists of an integer value, n, representing the number of records.

The second line of input consists of n space-separated city names.

The third line of input consists of n space-separated month names.

The fourth line of input consists of n space-separated float values representing sales for each city-month combination.

***Output Format***

The first line of output prints: "Monthly Sales Data with MultiIndex:"

The next lines print the DataFrame with MultiIndex (City, Month) and their corresponding sales values.

The following line prints: "\nTotal Sales Per City:"

The final lines print the total sales per city, computed by grouping the sales data on city names.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 4
NYC NYC LA LA
Jan Feb Jan Feb
100 200 300 400

Output: Monthly Sales Data with MultiIndex:

```
            Sales
City Month
NYC  Jan    100.0
     Feb    200.0
LA   Jan    300.0
     Feb    400.0

Total Sales Per City:
      Sales
City
LA    700.0
NYC   300.0
```

***Answer***

```
import pandas as pd
```

```
# Read input values
n = int(input())
cities = input().split()
months = input().split()
sales = list(map(float, input().split()))

# Create MultiIndex DataFrame
index = pd.MultiIndex.from_arrays([cities, months], names=["City", "Month"])
df = pd.DataFrame({"Sales": sales}, index=index)

# Print the required output
print("Monthly Sales Data with MultiIndex:")
print(df)

print("\nTotal Sales Per City:")
print(df.groupby("City").sum())
```

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

2. Problem Statement

Alex is a data scientist analyzing the relationship between two financial indicators over time. He has collected two time series datasets representing daily values of these indicators over several months. Alex wants to understand how these two indicators correlate at different time lags to identify possible leading or lagging behaviors.

Your task is to help Alex compute the cross-correlation of these two time series using numpy, so he can analyze the similarity between the two signals at various time shifts.

***Input Format***

The first line of input consists of space-separated float values representing the first time series, array1.

The second line of input consists of space-separated float values representing the second time series, array2.

***Output Format***

The first line of output prints: "Cross-correlation of the two time series:"

The second line of output prints: the 1D numpy array cross_corr representing the cross-correlation of array1 and array2 across different lags.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 1.0 2.0 3.0
4.0 5.0 6.0

Output: Cross-correlation of the two time series:
[ 6. 17. 32. 23. 12.]

***Answer***

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

# Read input values
array1 = np.array(list(map(float, input().split())))
array2 = np.array(list(map(float, input().split())))

# Compute cross-correlation
cross_corr = np.correlate(array1, array2, mode='full')

# Print the output
print("Cross-correlation of the two time series:")
print(cross_corr)
```

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

3. Problem Statement

Rekha works in hospital data management and receives patient records with missing or incomplete data. She needs to clean the records by performing the following tasks:

Calculate the mean of the available Age values.Replace any missing (NaN) values in the Age column with this mean age.Remove any rows where the

Diagnosis value is missing (NaN).Reset the DataFrame index after removing these rows.

Implement this data cleaning task using the pandas package.

***Input Format***

The first line of input contains an integer n representing the number of patient records.

The second line contains the CSV header — comma-separated column names (e.g., "Name,Age,Diagnosis,Gender").

The next n lines each contain one patient record in comma-separated format.

***Output Format***

The first line of output is the text:

Cleaned Hospital Records:

The next lines print the cleaned pandas DataFrame (as produced by print(cleaned_df)).

This will include the updated values of the Age column (with missing ages filled by the mean age), and any rows with missing Diagnosis removed.

The DataFrame will be displayed using the default pandas print() representation.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
PatientID,Name,Age,Diagnosis
1,John Doe,45,Flu
2,Jane Smith,,Cold
3,Bob Lee,50,
4,Alice Green,38,Fever
5,Tom Brown,,Infection
Output: Cleaned Hospital Records:

```
   PatientID        Name       Age  Diagnosis
0          1    John Doe  45.000000       Flu
1          2  Jane Smith  44.333333      Cold
2          4  Alice Green  38.000000     Fever
3          5   Tom Brown  44.333333  Infection
```

***Answer***

```
import pandas as pd

# Read input values
n = int(input().strip())
header = input().strip().split(',')
data = [input().strip().split(',') for _ in range(n)]

# Create DataFrame
df = pd.DataFrame(data, columns=header)

# Convert Age column to numeric, handling missing values
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')

# Compute mean age from available values
mean_age = df['Age'].mean()

# Fill missing Age values with mean age
df['Age'].fillna(mean_age, inplace=True)

# Drop rows where Diagnosis is empty (instead of NaN handling)
df = df[df['Diagnosis'].str.strip() != '']

# Reset the index after removal
df.reset_index(drop=True, inplace=True)

# Print results
print("Cleaned Hospital Records:")
print(df)
```

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

4. Problem Statement

A company tracks the monthly sales data of various products. You are

given a table where each row represents a product and each column represents its monthly sales in sequential months.

Your task is to compute the cumulative monthly sales for each product using numpy, where the cumulative sales for a month is the total sales from month 1 up to that month.

***Input Format***

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

Each of the next products lines consists of months integer values representing the monthly sales data of a product.

***Output Format***

The first line of output prints: "Cumulative Monthly Sales:"

The second line of output prints: the 2D numpy array cumulative_array that contains the cumulative sales data for each product.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 2 4
10 20 30 40
5 15 25 35

Output: Cumulative Monthly Sales:
[[ 10  30  60 100]
 [  5  20  45  80]]

***Answer***

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

# Read input values
products, months = map(int, input().split())

# Read sales data for each product
```

```
sales_data = [list(map(int, input().split())) for _ in range(products)]

# Convert to NumPy array
sales_array = np.array(sales_data)

# Compute cumulative sum along columns (monthly progression)
cumulative_array = np.cumsum(sales_array, axis=1)

# Print the result
print("Cumulative Monthly Sales:")
print(cumulative_array)
```

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

5. Problem Statement

Sita is analyzing her company's daily sales data to find all sales values that are multiples of 5 and exceed 100. She wants to filter these specific sales values from the list.

Help her to implement the task using the numpy package.

Formula:

To filter sales values:

Select all values s from sales such that (s % 5 == 0) and (s > 100)

***Input Format***

The first line of input consists of an integer value, n, representing the number of sales entries.

The second line of input consists of n floating-point values, sales, separated by spaces, representing daily sales figures.

***Output Format***

The output prints: filtered_sales

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
50.0 100.0 105.0 150.0 99.0
Output: [105. 150.]

***Answer***

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

# Read input values
n = int(input().strip())
sales = np.array(list(map(float, input().split())))

# Apply filtering condition
filtered_sales = sales[(sales % 5 == 0) & (sales > 100)]

# Print the result
print(filtered_sales)
```

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

# Rajalakshmi Engineering College

Name: Rakshitha RD
Email: 241501163@rajalakshmi.edu.in
Roll no: 241501163
Phone: 9444821024
Branch: REC
Department: I AI & ML FB
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 : 45

## 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

# Read input values
n = int(input().strip())
```

```
data = [input().strip().split() for _ in range(n)]

# Convert data into a DataFrame
df = pd.DataFrame(data, columns=["Name", "Years at Company"])
df["Years at Company"] = df["Years at Company"].astype(float)  # Convert to float

# Define experience level classification function
def classify_experience(years):
    if years < 3:
        return "Junior"
    elif 3 <= years < 6:
        return "Mid"
    else:
        return "Senior"

# Apply classification to create a new column
df["Experience Level"] = df["Years at Company"].apply(classify_experience)

# Print the result
print("Employee Data with Experience Level:")
print(df.to_string(index=False))
```

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

2. 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

# Read input values
n = int(input().strip())
wait_times = np.array(list(map(float, input().split())))

# Determine bucket edges dynamically
max_wait = np.ceil(max(wait_times))  # Round up to the nearest bucket limit
bins = np.arange(0, max_wait + 5, 5)  # Create bucket edges (0, 5, 10, ...)

# Assign wait times to buckets using np.digitize
bucket_indices = np.digitize(wait_times, bins, right=False)

# Count occurrences in each bucket
bucket_counts = {f"{int(bins[i])}-{int(bins[i+1])}": (bucket_indices == i+1).sum() for
i in range(len(bins) - 1)}

# Print results
print("Wait Time Buckets and Counts:")
for bucket, count in bucket_counts.items():
    if count > 0:  # Print only non-empty buckets
        print(f"{bucket}: {count}")
```

***Status :*** Partially correct ***Marks : 6/10***

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

# Read input values
rows, cols = map(int, input().split())

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

# Identify days where all asset returns are positive
positive_days = np.all(returns > 0, axis=1)
```

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

***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 input values
rows = int(input().strip())
cols = int(input().strip())

# Read pixel values into a NumPy array
image_matrix = np.array([list(map(int, input().split())) for _ in range(rows)])

# Get min and max pixel values
min_pixel = np.min(image_matrix)
max_pixel = np.max(image_matrix)

# Normalize the image matrix
if min_pixel == max_pixel:
    normalized_image = np.zeros_like(image_matrix, dtype=float)  # Fill with zeros
if all pixels are the same
else:
    normalized_image = (image_matrix - min_pixel) / (max_pixel - min_pixel)

# Print the normalized matrix
print(normalized_image)
```

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

5. 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

# Read input values
n = int(input().strip())
data = [input().strip().split(maxsplit=1) for _ in range(n)]

# Convert to DataFrame
df = pd.DataFrame(data, columns=["RespondentID", "Feedback"])

# Fill missing or empty feedback
df["Feedback"] = df["Feedback"].fillna("No Response").replace("", "No Response")

# Print the formatted output
print("Survey Responses with Missing Feedback Filled:")
print(df.to_string(index=False))
```

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