Rajalakshmi Engineering College

Name: krithika narasimhan
Email: 240701277@rajalakshmi.edu.in
Roll no: 240701277
Phone: 9677451731
Branch: REC
Department: I CSE FC
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 7_COD

Attempt : 1
Total Mark : 50
Marks Obtained : 48.5

Section 1 : Coding

1. 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
import io
```

```
import sys

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

    # Drop rows with missing 'Diagnosis'
    df.dropna(subset=['Diagnosis'], inplace=True)

    # Reset index after dropping rows
    df.reset_index(drop=True, inplace=True)

    return df

def main():
    # Read number of records
    n = int(input())

    # Read CSV header (comma-separated column names)
    header = input()

    # Read n lines of CSV data
    data_lines = [input() for _ in range(n)]

    # Combine header and data into CSV format string
    csv_data = '\n'.join([header] + data_lines)

    # Use StringIO to read the CSV data into pandas DataFrame
    df = pd.read_csv(io.StringIO(csv_data))

    cleaned_df = clean_hospital_records(df)

    print("Cleaned Hospital Records:")
    print(cleaned_df)

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

Status : Partially correct **Marks : 8.5/10**

2. 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

def create_multiindex_sales(cities, months, sales_values):
    # Create a MultiIndex from cities and months
    index = pd.MultiIndex.from_tuples(list(zip(cities, months)), names=["City",
"Month"])

    # Create DataFrame with sales data
    df = pd.DataFrame({"Sales": sales_values}, index=index)

    # Compute total sales per city by grouping on level=0 (City)
    total_sales = df.groupby(level=0).sum()

    return df, total_sales

def main():
    # Read number of records
    n = int(input())

    # Read cities (space separated)
    cities = input().split()

    # Read months (space separated)
    months = input().split()

    # Read sales values (space separated floats)
    sales_values = list(map(float, input().split()))
```

```
    sales_df, total_sales_df = create_multiindex_sales(cities, months,
sales_values)

    print("Monthly Sales Data with MultiIndex:")
    print(sales_df)

    print("\nTotal Sales Per City:")
    print(total_sales_df)

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

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

3. 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

```
import numpy as np

def cumulative_sales(sales_data):
    # Calculate cumulative monthly sales for each product
    cumulative = np.cumsum(sales_data, axis=1)
    return cumulative

def main():
    # Input dimensions: products and months
    products, months = map(int, input().split())

    # Input sales data rows
    data = [list(map(int, input().split())) for _ in range(products)]
    sales_array = np.array(data)

    # Calculate cumulative sales
    cumulative_array = cumulative_sales(sales_array)

    print("Cumulative Monthly Sales:")
    print(cumulative_array)

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

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

4. 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

```
import numpy as np

def compute_cross_correlation(array1, array2):
    # Compute the cross-correlation
    cross_corr = np.correlate(array1, array2, mode='full')
    return cross_corr
```

```
def main():
    # Get input arrays from the user
    array1 = np.array(list(map(float, input().split())))
    array2 = np.array(list(map(float, input().split())))

    # Compute cross-correlation
    cross_corr = compute_cross_correlation(array1, array2)

    print("Cross-correlation of the two time series:")
    print(cross_corr)

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

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

```
import numpy as np

# Read the number of sales values
n = int(input())

# Read the sales values (space-separated integers or floats)
sales = np.array(list(map(float, input().split())))

# Create mask: values that are multiples of 5 and greater than 100
mask = (sales % 5 == 0) & (sales > 100)

# Filtered sales values
filtered_sales = sales[mask]

print(filtered_sales)
```

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