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_CY

Attempt : 1
Total Mark : 50
Marks Obtained : 45.5

Section 1 : Coding

1. Problem Statement

You are working as a data analyst for a small retail store that wants to track the stock levels of its products. Each product has a unique Name (such as "Toothpaste", "Shampoo", "Soap") and an associated Quantity in stock. Management wants to identify which products have zero stock so they can be restocked.

Write a Python program using the pandas library to help with this task. The program should:

Read the number of products, n.Read n lines, each containing the Name of the product and its Quantity, separated by a space.Convert this data into a pandas DataFrame.Identify and display the Name and Quantity of products with zero stock.If no products have zero stock, display: No products with zero stock.

Input Format

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

The next n lines each contain:

<Product_ID> <Quantity>

where <Product_ID> is a single word (e.g., "Shampoo") and <Quantity> is a non-negative integer (e.g., 5).

Output Format

The first line of output prints:

Products with Zero Stock:

If there are any products with zero stock, the following lines print the pandas DataFrame showing those products with two columns: Product_ID and Quantity.

The column headers Product_ID and Quantity are printed in the second line.

Each subsequent line shows the product's name and quantity, aligned under the respective headers, with no index column.

The output formatting (spacing and alignment) follows the default pandas to_string(index=False) style.

If no products have zero stock, print:

No products with zero stock.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
P101 10
P102 0
P103 5

Output: Products with Zero Stock:
Product_ID Quantity
    P102   0

Answer

```
import pandas as pd

def main():
    # Number of products
    n = int(input())

    # Read product data
    data = [input().split() for _ in range(n)]
    df = pd.DataFrame(data, columns=['Product_ID', 'Quantity'])
    df['Quantity'] = df['Quantity'].astype(int)

    # Find products with zero stock
    zero_stock_products = df[df['Quantity'] == 0]

    print("Products with Zero Stock:")
    if not zero_stock_products.empty:
        print(zero_stock_products[['Product_ID', 'Quantity']].to_string(index=False))
    else:
        print("No products with zero stock.")

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

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

2. Problem Statement

Rekha is a meteorologist analyzing rainfall data collected over 5 years, with monthly rainfall recorded for each year. She wants to find the total rainfall each year and also identify the month with the maximum rainfall for every year.

Help her to implement the task using the numpy package.

Formula:

Yearly total rainfall = sum of all 12 months' rainfall for each year

Month with max rainfall = index of the maximum rainfall value within the 12 months for each year (0-based index)

Input Format

The input consists of 5 lines.

Each line contains 12 floating-point values separated by spaces, representing the rainfall data (in mm) for each month of that year.

Output Format

The first line of output prints: yearly_totals

The second line of output prints: max_rainfall_months

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 7.0 8.0 9.0 10.0 11.0 12.0
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0
3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0
4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0
5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0

Output: [78. 90. 102. 114. 126.]
[11 11 11 11 11]

Answer

```
import numpy as np

# Read 5x12 rainfall data (5 years, 12 months each)
data = [list(map(float, input().split())) for _ in range(5)]
rainfall = np.array(data)

# Yearly total rainfall (sum across months axis)
yearly_totals = rainfall.sum(axis=1)

# Index of month with max rainfall each year
max_rainfall_months = rainfall.argmax(axis=1)

print(yearly_totals)
print(max_rainfall_months)
```

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

3. Problem Statement

Arjun is monitoring hourly temperature data recorded continuously for multiple days. He needs to calculate the average temperature for each day based on 24 hourly readings.

Help him to implement the task using the numpy package.

Formula:

Reshape the temperature readings into rows where each row has 24 readings (one day).

Average temperature per day = mean of 24 hourly readings in each row.

Input Format

The first line of input consists of an integer value, n, representing the total number of temperature readings.

The second line of input consists of n floating-point values separated by spaces, representing hourly temperature readings.

Output Format

The output prints: avg_per_day

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 30
30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0
30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0
Output: [30.]

Answer

```
import numpy as np

# Read number of temperature readings
n = int(input())

# Read temperature readings (space-separated floats)
temps = np.array(list(map(float, input().split())))

# Reshape into rows with 24 columns (hours)
days = temps.reshape(-1, 24)

# Compute average temperature per day (mean across columns)
avg_per_day = days.mean(axis=1)

print(avg_per_day)
```

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

4. Problem Statement

Rekha works as an e-commerce data analyst. She receives transaction data containing purchase dates and needs to extract the month and day from these dates using the pandas package.

Help her implement this task by performing the following steps:

Convert the Purchase Date column to datetime format, treating invalid date entries as NaT (missing).

Create two new columns:

Purchase Month, containing the month (as an integer) extracted from the Purchase Date.

Purchase Day, containing the day (as an integer) extracted from the Purchase Date. Keep the rest of the data as is.

Input Format

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

The second line contains the CSV header — comma-separated column names.

The next n lines each contain a transaction record in comma-separated format.

Output Format

The first line of output is the text:

Transformed E-commerce Transaction Data:

The next lines print the pandas DataFrame with:

The original columns (including Purchase Date, which is now in datetime format or NaT if invalid).

Two additional columns: Purchase Month and Purchase Day.

The output uses the default pandas DataFrame string representation as produced by print(transformed_df).

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
Customer,Purchase Date
Alice,2023-05-15

Bob,2023-06-20
Charlie,2023-07-01
Output: Transformed E-commerce Transaction Data:

	Customer	Purchase Date	Purchase Month	Purchase Day
0	Alice	2023-05-15	5	15
1	Bob	2023-06-20	6	20
2	Charlie	2023-07-01	7	1

Answer

```
import pandas as pd
import io

def transform_purchase_dates(df):
    # Convert 'Purchase Date' column to datetime format
    df['Purchase Date'] = pd.to_datetime(df['Purchase Date'], errors='coerce')

    # Extract month and day into new columns
    df['Purchase Month'] = df['Purchase Date'].dt.month
    df['Purchase Day'] = df['Purchase Date'].dt.day

    return df

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

    # Read header
    header = input()

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

    # Combine to form CSV data
    csv_data = '\n'.join([header] + data_lines)

    # Read DataFrame
    df = pd.read_csv(io.StringIO(csv_data))

    transformed_df = transform_purchase_dates(df)

    print("Transformed E-commerce Transaction Data:")
    print(transformed_df)
```

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

Status : Partially correct ***Marks : 7/10***

5. Problem Statement

Arjun is developing a system to monitor environmental sensors installed in different rooms of a smart building. Each sensor records multiple temperature readings throughout the day. To compare sensor data fairly despite differing scales, Arjun needs to normalize each sensor's readings so that they have a mean of zero and standard deviation of one.

Help him implement this normalization using numpy.

Normalization Formula:

Input Format

The first line of input consists of two integers: sensors (number of sensors) and samples (number of readings per sensor).

The next sensors lines each contain samples space-separated floats representing the sensor readings.

Output Format

The first line of output prints: "Normalized Sensor Data:"

The next lines print the normalized readings as a numpy array, where each row corresponds to a sensor's normalized values.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3 3

1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

Output: Normalized Sensor Data:
[[-1.22474487 0. 1.22474487]
[-1.22474487 0. 1.22474487]
[-1.22474487 0. 1.22474487]]

Answer

```
import numpy as np

def normalize_sensor_data(sensor_data):
    # Compute row-wise mean and std
    row_mean = np.mean(sensor_data, axis=1, keepdims=True)
    row_std = np.std(sensor_data, axis=1, keepdims=True)

    # Normalize: (value - row_mean) / row_std
    normalized_data = (sensor_data - row_mean) / row_std

    return normalized_data

def main():
    # Read number of sensors and samples
    sensors, samples = map(int, input().split())

    # Read sensor data
    data_lines = [list(map(float, input().split())) for _ in range(sensors)]
    sensor_data = np.array(data_lines)

    # Normalize
    normalized_data = normalize_sensor_data(sensor_data)

    print("Normalized Sensor Data:")
    print(normalized_data)

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

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