MapUp-Data-Assessment-F / README.md

saransh-mapup MapUp-Data-Assessment-F 5 hours ago

145 lines (93 loc) · 7.55 KB

Preview Code Blame Raw

MapUp - Python Assessment

Overview

This assessment is designed to evaluate your proficiency in Python programming, data manipulation, and analysis, as well as your ability to work with Excel. Below, you'll find details on each component of the assessment and the tasks you should complete. Best of luck!

Important Points to Note:

- The assessment will be tested using our internal set of test cases. Scripts must be developed in accordance with the template shared. Please use the following template to create your scripts:
  - 📁 templates
    - 📄 python_task_1.py
    - 📄 python_task_2.py
- We've clearly outlined the interfaces of our functions, specifying the input and output data types with distinct signatures.
- Any deviation especially in naming conventions and providing arguments will impact the correct assessment of your work

Submission structure

There should be a folder named `submission` in the root of your repository. This folder should contain the following:

- 📂 submissions
  - 📄 python_task_1.py
  - 📄 python_task_2.py
  - 📄 excel_assessment.xlsm

Result Submission:

- Data that you need to work with is in the folder `datasets` . Store your process outputs in the structure mentioned below
- Clone the provided GitHub repository.
- Add the following members as collaborators to your repo
  - `venkateshn@mapup.ai`
  - `namanjeetsingh@mapup.ai`
  - `saranshj@mapup.ai`
  - `varuna@mapup.ai`
- Submit the link to your repository via the provided Google Form for evaluation.

MapUp - Excel Assessment

You have to submit an excel assessment along with your python task. This evaluation tests your proficiency in Conditional Formatting, Excel Formulae, and Data Manipulation

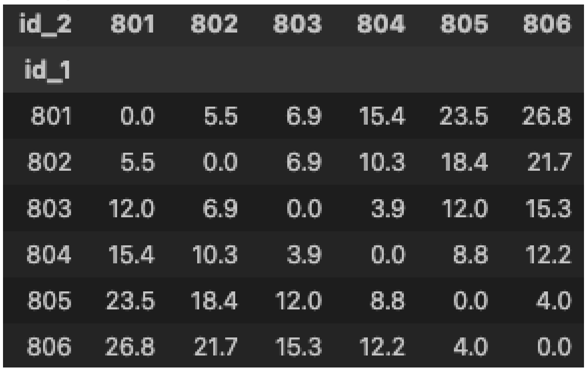
Python Task 1

Question 1: Car Matrix Generation

Under the function named `generate_car_matrix` write a logic that takes the `dataset-1.csv` as a DataFrame. Return a new DataFrame that follows the following rules:

- values from `id_2` as columns
- values from `id_1` as index
- dataframe should have values from `car` column
- diagonal values should be 0.

Sample result dataframe:



id_2	801	802	803	804	805	806
id_1						
801	0.0	5.5	6.9	15.4	23.5	26.8
802	5.5	0.0	6.9	10.3	18.4	21.7
803	12.0	6.9	0.0	3.9	12.0	15.3
804	15.4	10.3	3.9	0.0	8.8	12.2
805	23.5	18.4	12.0	8.8	0.0	4.0
806	26.8	21.7	15.3	12.2	4.0	0.0

Question 2: Car Type Count Calculation

Create a Python function named `get_type_count` that takes the `dataset-1.csv` as a DataFrame. Add a new categorical column `car_type` based on values of the column `car` :

- `low` for values less than or equal to 15,
- `medium` for values greater than 15 and less than or equal to 25,
- `high` for values greater than 25.

Calculate the count of occurrences for each `car_type` category and return the result as a dictionary. Sort the dictionary alphabetically based on keys.

Question 3: Bus Count Index Retrieval

Create a Python function named `get_bus_indexes` that takes the `dataset-1.csv` as a DataFrame. The function should identify and return the indices as a list (sorted in ascending order) where the `bus` values are greater than twice the mean value of the `bus` column in the DataFrame.

Question 4: Route Filtering

Create a python function `filter_routes` that takes the `dataset-1.csv` as a DataFrame. The function should return the sorted list of values of column `route` for which the average of values of `truck` column is greater than 7.

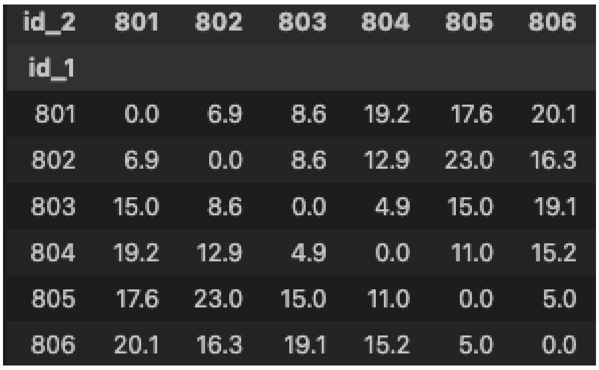
Question 5: Matrix Value Modification

Create a Python function named `multiply_matrix` that takes the resulting DataFrame from Question 1, as input and modifies each value according to the following logic:

- If a value in the DataFrame is greater than 20, multiply those values by 0.75,
- If a value is 20 or less, multiply those values by 1.25.

The function should return the modified DataFrame which has values rounded to 1 decimal place.

Sample result dataframe:



id_2	801	802	803	804	805	806
id_1						
801	0.0	6.9	8.6	19.2	17.6	20.1
802	6.9	0.0	8.6	12.9	23.0	16.3
803	15.0	8.6	0.0	4.9	15.0	19.1
804	19.2	12.9	4.9	0.0	11.0	15.2
805	17.6	23.0	15.0	11.0	0.0	5.0
806	20.1	16.3	19.1	15.2	5.0	0.0

Question 6: Time Check

You are given a dataset, `dataset-2.csv`, containing columns `id`, `id_2`, and timestamp (`startDay`, `startTime`, `endDay`, `endTime`). The goal is to verify the completeness of the time data by checking whether the timestamps for each unique (`id`, `id_2`) pair cover a full 24-hour period (from 12:00:00 AM to 11:59:59 PM) and span all 7 days of the week (from Monday to Sunday).

Create a function that accepts `dataset-2.csv` as a DataFrame and returns a boolean series that indicates if each (`id`, `id_2`) pair has incorrect timestamps. The boolean series must have multi-index (`id`, `id_2`).

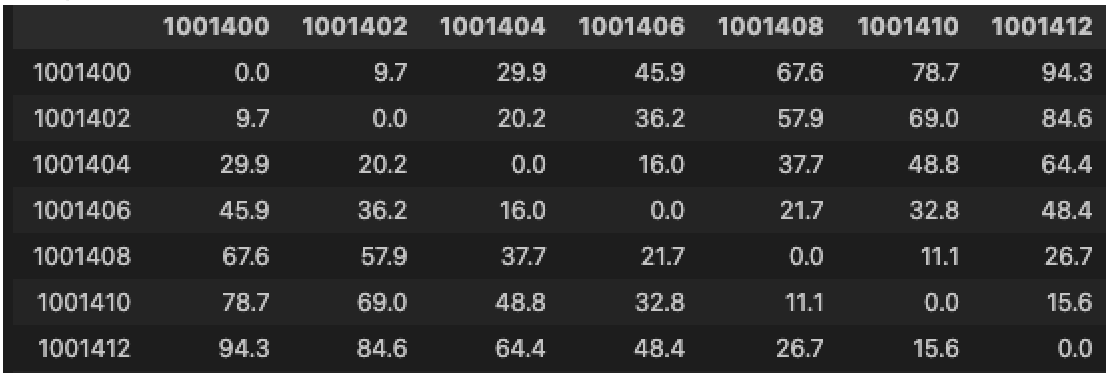
Python Task 2

Question 1: Distance Matrix Calculation

Create a function named `calculate_distance_matrix` that takes the `dataset-3.csv` as input and generates a DataFrame representing distances between IDs.

The resulting DataFrame should have cumulative distances along known routes, with diagonal values set to 0. If distances between toll locations A to B and B to C are known, then the distance from A to C should be the sum of these distances. Ensure the matrix is symmetric, accounting for bidirectional distances between toll locations (i.e. A to B is equal to B to A).

Sample result dataframe:



	1001400	1001402	1001404	1001406	1001408	1001410	1001412
1001400	0.0	9.7	29.9	45.9	67.6	78.7	94.3
1001402	9.7	0.0	20.2	36.2	57.9	69.0	84.6
1001404	29.9	20.2	0.0	16.0	37.7	48.8	64.4
1001406	45.9	36.2	16.0	0.0	21.7	32.8	48.4
1001408	67.6	57.9	37.7	21.7	0.0	11.1	26.7
1001410	78.7	69.0	48.8	32.8	11.1	0.0	15.6
1001412	94.3	84.6	64.4	48.4	26.7	15.6	0.0

Question 2: Unroll Distance Matrix

Create a function `unroll_distance_matrix` that takes the DataFrame created in Question 1. The resulting DataFrame should have three columns: columns `id_start` , `id_end` , and `distance` .

All the combinations except for same `id_start` to `id_end` must be present in the rows with their distance values from the input DataFrame.

Question 3: Finding IDs within Percentage Threshold

Create a function `find_ids_within_ten_percentage_threshold` that takes the DataFrame created in Question 2 and a reference value from the `id_start` column as an integer.

Calculate average distance for the reference value given as an input and return a sorted list of values from `id_start` column which lie within 10% (including ceiling and floor) of the reference value's average.

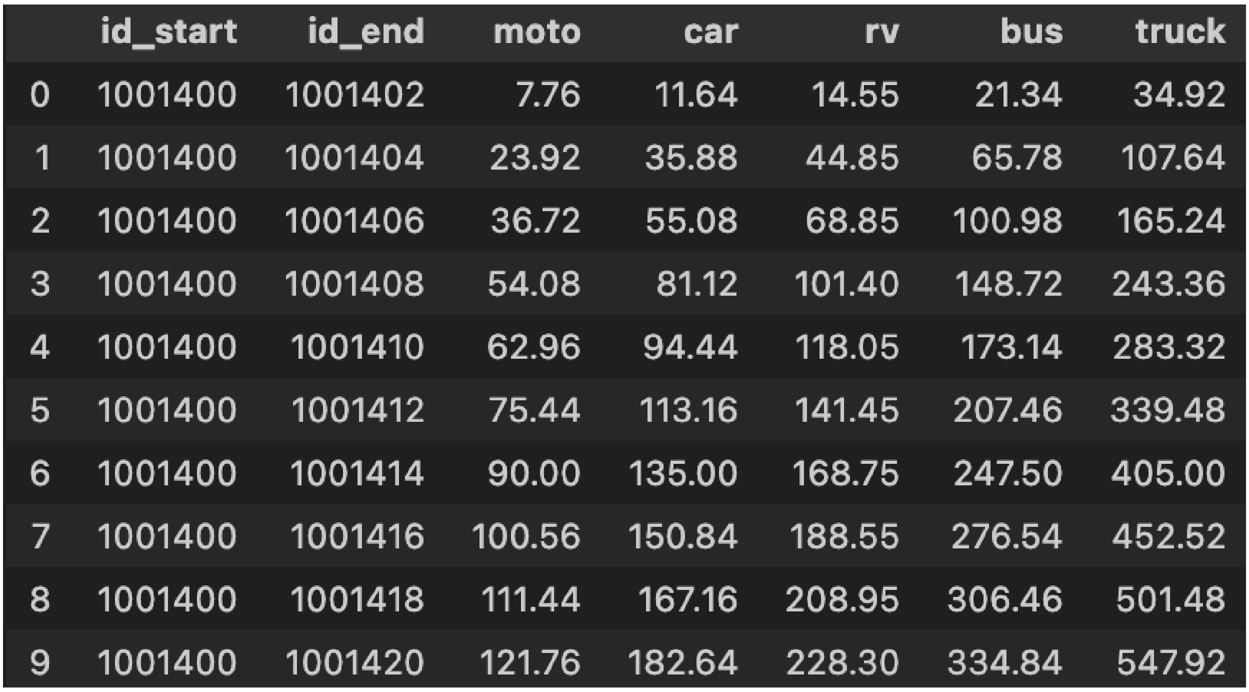
Question 4: Calculate Toll Rate

Create a function `calculate_toll_rate` that takes the DataFrame created in Question 2 as input and calculates toll rates based on vehicle types.

The resulting DataFrame should add 5 columns to the input DataFrame: `moto` , `car` , `rv` , `bus` , and `truck` with their respective rate coefficients. The toll rates should be calculated by multiplying the distance with the given rate coefficients for each vehicle type:

- 0.8 for `moto`
- 1.2 for `car`
- 1.5 for `rv`
- 2.2 for `bus`
- 3.6 for `truck`

Sample result dataframe:



	id_start	id_end	moto	car	rv	bus	truck
0	1001400	1001402	7.76	11.64	14.55	21.34	34.92
1	1001400	1001404	23.92	35.88	44.85	65.78	107.64
2	1001400	1001406	36.72	55.08	68.85	100.98	165.24
3	1001400	1001408	54.08	81.12	101.40	148.72	243.36
4	1001400	1001410	62.96	94.44	118.05	173.14	283.32
5	1001400	1001412	75.44	113.16	141.45	207.46	339.48
6	1001400	1001414	90.00	135.00	168.75	247.50	405.00
7	1001400	1001416	100.56	150.84	188.55	276.54	452.52
8	1001400	1001418	111.44	167.16	208.95	306.46	501.48
9	1001400	1001420	121.76	182.64	228.30	334.84	547.92

Question 5: Calculate Time-Based Toll Rates

Create a function named `calculate_time_based_toll_rates` that takes the DataFrame created in Question 3 as input and calculates toll rates for different time intervals within a day.

The resulting DataFrame should have these five columns added to the input: start_day, start_time, end_day, and end_time.

- `start_day` , `end_day` must be strings with day values (from Monday to Sunday in proper case)
- `start_time` and `end_time` must be of type datetime.time() with the values from time range given below.

Modify the values of vehicle columns according to the following time ranges:

Weekdays (Monday - Friday):

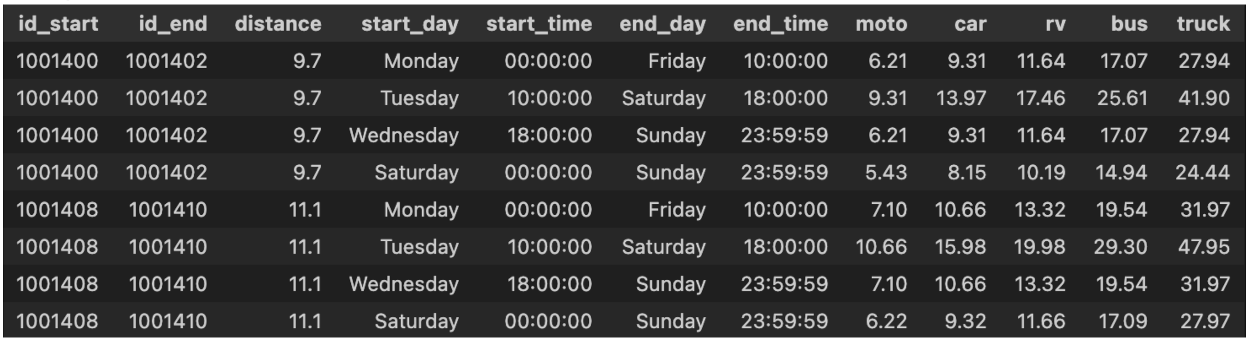
- From 00:00:00 to 10:00:00: Apply a discount factor of 0.8
- From 10:00:00 to 18:00:00: Apply a discount factor of 1.2
- From 18:00:00 to 23:59:59: Apply a discount factor of 0.8

Weekends (Saturday and Sunday):

- Apply a constant discount factor of 0.7 for all times.

For each unique (`id_start` , `id_end`) pair, cover a full 24-hour period (from 12:00:00 AM to 11:59:59 PM) and span all 7 days of the week (from Monday to Sunday).

Sample result dataframe:



id_start	id_end	distance	start_day	start_time	end_day	end_time	moto	car	rv	bus	truck
1001400	1001402	9.7	Monday	00:00:00	Friday	10:00:00	6.21	9.31	11.64	17.07	27.94
1001400	1001402	9.7	Tuesday	10:00:00	Saturday	18:00:00	9.31	13.97	17.46	25.61	41.90
1001400	1001402	9.7	Wednesday	18:00:00	Sunday	23:59:59	6.21	9.31	11.64	17.07	27.94
1001400	1001402	9.7	Saturday	00:00:00	Sunday	23:59:59	5.43	8.15	10.19	14.94	24.44
1001408	1001410	11.1	Monday	00:00:00	Friday	10:00:00	7.10	10.66	13.32	19.54	31.97
1001408	1001410	11.1	Tuesday	10:00:00	Saturday	18:00:00	10.66	15.98	19.98	29.30	47.95
1001408	1001410	11.1	Wednesday	18:00:00	Sunday	23:59:59	7.10	10.66	13.32	19.54	31.97
1001408	1001410	11.1	Saturday	00:00:00	Sunday	23:59:59	6.22	9.32	11.66	17.09	27.97