In [122]:
```
##task 1 question 1
```

In [95]:
```
import pandas as pd
import numpy as np
```

In [96]:
```
def generate_car_matrix(df):
    matrix = df.pivot(index='id_1', columns='id_2', values='car').fillna(0)
    matrix_array=matrix.values
    np.fill_diagonal(matrix_array,0)
    modified_matrix=pd.DataFrame(matrix_array,index=matrix.index,columns=matrix.columns)
    return modified_matrix
```

In [97]:
```
df=pd.read_csv('dataset-1.csv')
result_matrix=generate_car_matrix(df)
print(result_matrix)
```

```
id_2    801    802    803    804    805    806    807    808    809    821  \
id_1
801    0.00   2.80   6.00   7.70  11.70  13.40  16.90  19.60  21.00  23.52
802    2.80   0.00   3.40   5.20   9.20  10.90  14.30  17.10  18.50  20.92
803    6.00   3.40   0.00   2.00   6.00   7.70  11.10  13.90  15.30  17.72
804    7.70   5.20   2.00   0.00   4.40   6.10   9.50  12.30  13.70  16.12
805   11.70   9.20   6.00   4.40   0.00   2.00   5.40   8.20   9.60  12.02
806   13.40  10.90   7.70   6.10   2.00   0.00   3.80   6.60   8.00  10.42
807   16.90  14.30  11.10   9.50   5.40   3.80   0.00   2.90   4.30   6.82
808   19.60  17.10  13.90  12.30   8.20   6.60   2.90   0.00   1.70   4.12
809   21.00  18.50  15.30  13.70   9.60   8.00   4.30   1.70   0.00   2.92
821   23.52  20.92  17.72  16.12  12.02  10.42   6.82   4.12   2.92   0.00
822   24.67  22.07  18.87  17.27  13.17  11.57   7.97   5.27   4.07   1.80
823   26.53  23.93  20.73  19.13  15.03  13.43   9.83   7.13   5.93   3.67
824   27.92  25.32  22.12  20.52  16.42   7.80  11.22   8.52   7.32   5.06
825   29.08  26.48  23.28  21.68  17.58  15.98  12.38   9.68   8.48   6.22
826   30.87  28.27  25.07  23.47  19.37  17.77  14.17  11.47  10.27   8.01
827   32.53  29.93  26.73  25.13  21.03  19.43  15.83  13.13  11.93   9.43
829   36.32  33.72  30.52  28.92  24.82  23.22  19.62  16.92  15.72  13.26
```

In [98]:
```
##task 1 question 2
```

In [99]:
```python
import pandas as pd

def get_type_count(df):
    df['car_type'] = pd.cut(df['car'], bins=[-float('inf'), 15, 25, float('inf')], labels=['low', 'medium', 'high'])

    type_counts = df['car_type'].value_counts().to_dict()

    sorted_type_counts = dict(sorted(type_counts.items()))

    return sorted_type_counts

df = pd.read_csv('dataset-1.csv')

result_counts = get_type_count(df)
print(result_counts)
```

```
{'high': 56, 'low': 196, 'medium': 89}
```

In [100]:
```python
## Task 1 question 3
```

In [101]:
```python
import pandas as pd

def get_bus_indexes(df):
    bus_mean = df['bus'].mean()

    bus_indexes = df[df['bus'] > 2 * bus_mean].index.tolist()

    bus_indexes.sort()

    return bus_indexes


df = pd.read_csv('dataset-1.csv')

result_indexes = get_bus_indexes(df)
print(result_indexes)
```

```
[2, 7, 12, 17, 25, 30, 54, 64, 70, 97, 144, 145, 149, 154, 160, 201, 206, 210, 215, 234, 235, 245, 250, 309, 314, 319, 322, 323, 334, 340]
```

In [102]:
```python
## Task 1 question 4
```

In [103]:
```
import pandas as pd

def filter_routes(df):
    route_avg_truck = df.groupby('route')['truck'].mean()

    filtered_routes = route_avg_truck[route_avg_truck > 7].index.tolist()

    filtered_routes.sort()

    return filtered_routes

df = pd.read_csv('dataset-1.csv')

result_routes = filter_routes(df)
print(result_routes)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [121]:
```
## Task 1 question 5
```

In [105]:
```
import pandas as pd

def multiply_matrix(matrix):
    modified_matrix = matrix.copy()

    modified_matrix[matrix > 20] *= 0.75
    modified_matrix[(matrix <= 20) & (matrix > 0)] *= 1.25
    modified_matrix = modified_matrix.round(1)

    return modified_matrix


modified_result_matrix = multiply_matrix(result_matrix)

print(modified_result_matrix)
```

```
id_2   801   802   803   804   805   806   807   808   809   821   822   823  \
id_1
801    0.0   3.5   7.5   9.6  14.6  16.8  21.1  24.5  15.8  17.6  18.5  19.9
802    3.5   0.0   4.2   6.5  11.5  13.6  17.9  21.4  23.1  15.7  16.6  17.9
803    7.5   4.2   0.0   2.5   7.5   9.6  13.9  17.4  19.1  22.2  23.6  15.5
804    9.6   6.5   2.5   0.0   5.5   7.6  11.9  15.4  17.1  20.2  21.6  23.9
805   14.6  11.5   7.5   5.5   0.0   2.5   6.8  10.2  12.0  15.0  16.5  18.8
806   16.8  13.6   9.6   7.6   2.5   0.0   4.8   8.2  10.0  13.0  14.5  16.8
807   21.1  17.9  13.9  11.9   6.8   4.8   0.0   3.6   5.4   8.5  10.0  12.3
808   24.5  21.4  17.4  15.4  10.2   8.2   3.6   0.0   2.1   5.2   6.6   8.9
809   15.8  23.1  19.1  17.1  12.0  10.0   5.4   2.1   0.0   3.6   5.1   7.4
821   17.6  15.7  22.2  20.2  15.0  13.0   8.5   5.2   3.6   0.0   2.2   4.6
822   18.5  16.6  23.6  21.6  16.5  14.5  10.0   6.6   5.1   2.2   0.0   2.8
823   19.9  17.9  15.5  23.9  18.8  16.8  12.3   8.9   7.4   4.6   2.8   0.0
824   20.9  19.0  16.6  15.4  20.5   9.8  14.0  10.6   9.2   6.3   4.5   2.2
825   21.8  19.9  17.5  16.3  22.0  20.0  15.5  12.1  10.6   7.8   5.9   3.7
826   23.2  21.2  18.8  17.6  24.2  22.2  17.7  14.3  12.8  10.0   8.2   5.9
827   24.4  22.4  20.0  18.8  15.8  24.3  19.8  16.4  14.9  11.8  10.0   7.7
829   27.2  25.3  22.9  21.7  18.6  17.4  24.5  21.2  19.7  16.6  14.8  12.5
830   28.7  26.8  24.4  23.2  20.1  18.9  16.2  23.6  22.1  19.0  17.2  14.9
831   29.4  27.5  25.1  23.9  20.8  19.6  16.9  24.8  23.3  20.2  18.4  16.1

id_2   824   825   826   827   829   830   831
id_1
801   20.9  21.8  23.2  24.4  27.2  28.7  29.4
802   19.0  19.9  21.2  22.4  25.3  26.8  27.5
803   16.6  17.5  18.8  20.0  22.9  24.4  25.1
804   15.4  16.3  17.6  18.8  21.7  23.2  23.9
805   20.5  22.0  24.2  15.8  18.6  20.1  20.8
806   18.5  20.0  22.2  24.3  17.4  18.9  19.6
807   14.0  15.5  17.7  19.8  24.5  16.2  16.9
808   10.6  12.1  14.3  16.4  21.2  23.6  24.8
809    9.2  10.6  12.8  14.9  19.7  22.1  23.3
821    6.3   7.8  10.0  11.8  16.6  19.0  20.2
822    4.5   5.9   8.2  10.0  14.8  17.2  18.4
823    2.2   3.7   5.9   7.7  12.5  14.9  16.1
824    0.0   2.1   4.4   6.2  11.0  13.3  14.6
825    2.1   0.0   2.8   4.6   9.3  11.7  12.9
826    4.4   2.8   0.0   2.6   7.3   9.6  10.9
827    6.2   4.6   2.6   0.0   5.2   7.6   8.8
829   16.0   9.3   7.3   5.2   0.0   3.0   4.2
830   13.3   0.0   9.6   7.6   3.0   0.0   1.7
831   14.6  12.9  10.9   8.8   4.2   1.7   0.0
```

In [129]:
```
## Task 1 question 6
```

In [137]:
```
import pandas as pd
from datetime import datetime

def check_time_completeness(df):
    df = pd.read_csv('dataset-2.csv')
    df['start_datetime'] = df.apply(lambda row: f"{row['startDay']} {row['startTime']}", axis=1)
    df['end_datetime'] = df.apply(lambda row: f"{row['endDay']} {row['endTime']}", axis=1)
    df['start_datetime'] = pd.to_datetime(df['start_datetime'], format='%A %H:%M:%S')
    df['end_datetime'] = pd.to_datetime(df['end_datetime'], format='%A %H:%M:%S')
    df['duration'] = df['end_datetime'] - df['start_datetime']
    completeness_check = df.groupby(['id', 'id_2']).apply(
        lambda x: (
            (x['duration'].sum() >= pd.Timedelta(days=1)) and
            (x['start_datetime'].min().time() == pd.Timestamp('00:00:00').time()) and
            (x['end_datetime'].max().time() == pd.Timestamp('23:59:59').time()) and
            (len(x['start_datetime'].dt.dayofweek.unique()) == 7)
        )
    )

    return completeness_check
file_path = 'dataset-2.csv'
result = check_time_completeness(file_path)

print(result)
```

```
id       id_2
1014000  -1          False
1014002  -1          False
1014003  -1          False
1030000  -1          False
          1030002    False
                     ...
1330016   1330006    False
          1330008    False
          1330010    False
          1330012    False
          1330014    False
Length: 9254, dtype: bool
```

In [118]:
```
## Task 2 question 1
```

In [147]:
```python
import pandas as pd

def calculate_distance_matrix(pf):
    df = pd.read_csv('dataset-3.csv')
    distances = {}
    for _, row in df.iterrows():
        id_start = row['id_start']
        id_end = row['id_end']
        distance = row['distance']
        if id_start not in distances:
            distances[id_start] = {}
        if id_end not in distances:
            distances[id_end] = {}

        distances[id_start][id_end] = distance
        distances[id_end][id_start] = distance

    unique_ids = sorted(list(set(df['id_start'].unique()) | set(df['id_end'].unique())))
    num_ids = len(unique_ids)
    distance_matrix = [[0.0] * num_ids for _ in range(num_ids)]
    for i in range(num_ids):
        for j in range(num_ids):
            id_i = unique_ids[i]
            id_j = unique_ids[j]

            if id_j in distances[id_i]:
                distance_matrix[i][j] = distances[id_i][id_j]
    for k in range(num_ids):
        for i in range(num_ids):
            for j in range(num_ids):
                if distance_matrix[i][k] != 0 and distance_matrix[k][j] != 0:
                    if distance_matrix[i][j] == 0 or distance_matrix[i][j] > distance_matrix[i][k] + distance_matrix[k][j]:
                        distance_matrix[i][j] = distance_matrix[i][k] + distance_matrix[k][j]
    for i in range(num_ids):
        distance_matrix[i][i] = 0

    return pd.DataFrame(distance_matrix, index=unique_ids, columns=unique_ids)

file_path = 'dataset-3.csv'

result_matrix = calculate_distance_matrix('dataset-3.csv')

print(result_matrix)
```

```
1001488    152.0    138.8    125.2  ...    107.3    134.0    142.5    153.2
1004354    158.0    144.8    131.2  ...    101.3    128.0    136.5    147.2
1004355    160.0    146.8    133.2  ...     99.3    126.0    134.5    145.2
1004356    156.0    142.8    129.2  ...    103.3    130.0    138.5    149.2

         1001470  1001472  1001488  1004354  1004355  1004356
1001400    428.3    444.3    264.5    270.5    272.5    268.5
1001402    418.6    434.6    254.8    260.8    262.8    258.8
1001404    398.4    414.4    234.6    240.6    242.6    238.6
1001406    382.4    398.4    218.6    224.6    226.6    222.6
1001408    360.7    376.7    196.9    202.9    204.9    200.9
1001410    349.6    365.6    185.8    191.8    193.8    189.8
1001412    334.0    350.0    170.2    176.2    178.2    174.2
1001414    315.8    331.8    152.0    158.0    160.0    156.0
1001416    302.6    318.6    138.8    144.8    146.8    142.8
1001418    289.0    305.0    125.2    131.2    133.2    129.2
1001420    276.1    292.1    112.3    118.3    120.3    116.3
1001422    266.5    282.5    102.7    108.7    110.7    106.7
1001424    255.1    271.1     91.3     97.3     99.3     95.3
1001426    236.5    252.5     72.7     78.7     80.7     76.7
```

In [117]:
```
## Task 2 question 2
```

In [173]:
```python
import pandas as pd
from itertools import product
df = pd.read_csv('dataset-3.csv')

def unroll_distance_matrix(df):
    unique_ids = pd.concat([df['id_start'], df['id_end']]).unique()
    combinations = list(product(unique_ids, repeat=2))
    combinations = [pair for pair in combinations if pair[0] != pair[1]]

    unrolled_data = []
    for pair in combinations:
        distance = df[(df['id_start'] == pair[0]) & (df['id_end'] == pair[1])]['distance'].values
        if len(distance) > 0:
            unrolled_data.append({
                'id_start': pair[0],
                'id_end': pair[1],
                'distance': distance[0]
            })
        else:
            unrolled_data.append({
                'id_start': pair[0],
                'id_end': pair[1],
                'distance': None
            })

    unrolled_df = pd.DataFrame(unrolled_data)
    return unrolled_df
result_df = unroll_distance_matrix(df)
print(result_df)
```

```
      id_start   id_end  distance
0      1001400  1001402       9.7
1      1001400  1001404       NaN
2      1001400  1001406       NaN
3      1001400  1001408       NaN
4      1001400  1001410       NaN
...        ...      ...       ...
1801   1001472  1001464       NaN
1802   1001472  1001466       NaN
1803   1001472  1001468       NaN
1804   1001472  1001470       NaN
1805   1001472  1001437       NaN

[1806 rows x 3 columns]
```

In [165]:
```python
## task 2 question 3
```

In [172]:
```
import pandas as pd
df = pd.read_csv('dataset-3.csv')

def find_ids_within_ten_percentage_threshold(df, reference_value):
    reference_df = df[df['id_start'] == reference_value]
    avg_distance = reference_df['distance'].mean()
    threshold_min = avg_distance * 0.9
    threshold_max = avg_distance * 1.1
    filtered_ids = df.groupby('id_start')['distance'].mean()
    filtered_ids = filtered_ids[(filtered_ids >= threshold_min) & (filtered_ids <= threshold_max)]
    sorted_ids = filtered_ids.sort_values().index.tolist()
    return sorted_ids

reference_value = 1001400
result = find_ids_within_ten_percentage_threshold(df, reference_value)
print(result)
```

```
[1001456, 1001430, 1001420, 1001446, 1001400, 1001450, 1001468]
```

In [163]:
```
## task 2 question 4
```

In [155]:
```python
import pandas as pd

def calculate_toll_rate(df):
    rate_coefficients = {
        'moto': 0.8,
        'car': 1.2,
        'rv': 1.5,
        'bus': 2.2,
        'truck': 3.6
    }

    for vehicle_type, rate in rate_coefficients.items():
        df[vehicle_type] = df['distance'] * rate

    return df



sample_data = {
    'id_start': ['A', 'B', 'C', 'D', 'E'],
    'id_end': ['X', 'Y', 'Z', 'W', 'V'],
    'distance': [15, 20, 25, 30, 35]
}

sample_df = pd.DataFrame(sample_data)

result_with_rates = calculate_toll_rate(sample_df)

print(result_with_rates)
```

```
  id_start id_end  distance  moto   car    rv   bus  truck
0        A      X        15  12.0  18.0  22.5  33.0   54.0
1        B      Y        20  16.0  24.0  30.0  44.0   72.0
2        C      Z        25  20.0  30.0  37.5  55.0   90.0
3        D      W        30  24.0  36.0  45.0  66.0  108.0
4        E      V        35  28.0  42.0  52.5  77.0  126.0
```

In [156]:
```python
##task 2 question 5
```

In [157]:
```
import pandas as pd
from datetime import time

def calculate_time_based_toll_rates(df):
    time_intervals = [
        (time(0, 0, 0), time(10, 0, 0), 0.8),
        (time(10, 0, 0), time(18, 0, 0), 1.2),
        (time(18, 0, 0), time(23, 59, 59), 0.8)
    ]
    weekend_discount = 0.7

    df['start_day'] = pd.to_datetime(df['start_day']).dt.day_name().str.capitalize()
    df['end_day'] = pd.to_datetime(df['end_day']).dt.day_name().str.capitalize()

    df['start_time'] = pd.to_datetime(df['start_time']).dt.time
    df['end_time'] = pd.to_datetime(df['end_time']).dt.time

    for start, end, discount in time_intervals:
        weekday_mask = (
            (df['start_time'] >= start) & (df['start_time'] < end) &
            (df['end_time'] >= start) & (df['end_time'] < end) &
            (df['start_day'].isin(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'])) &
            (df['end_day'].isin(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']))
        )
        df.loc[weekday_mask, ['moto', 'car', 'rv', 'bus', 'truck']] *= discount

    weekend_mask = (
        (df['start_day'].isin(['Saturday', 'Sunday'])) &
        (df['end_day'].isin(['Saturday', 'Sunday']))
    )
    df.loc[weekend_mask, ['moto', 'car', 'rv', 'bus', 'truck']] *= weekend_discount

    return df
sample_data = {
    'id_start': ['A', 'B', 'C', 'D', 'E'],
    'id_end': ['X', 'Y', 'Z', 'W', 'V'],
    'moto': [20, 30, 25, 35, 40],
    'car': [25, 35, 30, 40, 45],
    'rv': [30, 40, 35, 45, 50],
    'bus': [35, 45, 40, 50, 55],
    'truck': [40, 50, 45, 55, 60],
    'start_day': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05'],
    'start_time': ['08:00:00', '12:00:00', '15:00:00', '20:00:00', '22:00:00'],
    'end_day': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05'],
    'end_time': ['10:00:00', '14:00:00', '17:00:00', '22:00:00', '23:00:00']
}

sample_df = pd.DataFrame(sample_data)

result_time_based_rates = calculate_time_based_toll_rates(sample_df)

print(result_time_based_rates)
```

```
  id_start id_end  moto   car    rv   bus  truck  start_day start_time  \
0        A      X  14.0  17.5  21.0  24.5   28.0     Sunday   08:00:00
1        B      Y  36.0  42.0  48.0  54.0   60.0     Monday   12:00:00
2        C      Z  30.0  36.0  42.0  48.0   54.0    Tuesday   15:00:00
3        D      W  28.0  32.0  36.0  40.0   44.0  Wednesday   20:00:00
4        E      V  32.0  36.0  40.0  44.0   48.0   Thursday   22:00:00

     end_day  end_time
0     Sunday  10:00:00
1     Monday  14:00:00
2    Tuesday  17:00:00
3  Wednesday  22:00:00
4   Thursday  23:00:00
```

In [ ]: