DataEng S24: Data Transformation In-Class Assignment

Submit: Make a copy of this document and use it to record your results. Store a PDF copy of the document in your git repository along with any needed code. Submit the in-class activity submission form by Friday at 10:00 pm.

A. [MUST] Initial Discussion Questions

Discuss the following questions among your working group members at the beginning of the week and place your own response into this space. If desired, also include responses from your group members.

1. In the lecture we mentioned the benefits of Data Transformation, but can you think of any problems that might arise with Data Transformation?

Data loss:
There is a chance of data loss when there are undergoing transformation operations and changes in the data types.

Introduction of Errors:
Errors or inconsistencies in the data may be introduced because of transformation logic defects.

2. Should data transformation occur before data validation in your data pipeline or after?

Before data validation:
Data transformation can be carried out to convert unprocessed data into a format that is easier to use for validation. The data may become easier to verify and more consistent with this method. without using plagiarized words.

After data validation:
Before transforming data, it is generally more typical to do data validation to make sure the data satisfies specific requirements and quality standards. We can prevent faulty data from spreading throughout your transformation process in this way. We may start the transformation process once the data has been verified and determined to be correct and clean.

B. [MUST] Small Sample of TriMet data

Here is sample data for one trip of one TriMet bus on one day (February 15, 2023): bc_trip259172515_230215.csv It's in .csv format not json format, but otherwise, the data is a typical subset of the data that you are using for your class project.

We recommend that you use google Colab or a Jupyter notebook for this assignment, though any python environment should suffice.

Use the pandas.read_csv() method to read the data into a DataFrame.

```
data = pd.read_csv("/content/drive/MyDrive/bc_trip259172515_230215.csv")
data
```

	EVENT_NO_TRIP	EVENT_NO_STOP	OPD_DATE	VEHICLE_ID	METERS	ACT_TIME	GPS_LONGITUDE	GPS_LATITUDE	GPS_SATELLITES	GPS_HDOP
0	259172515	259172517	15FEB2023:00:00:00	4223	40	20469	-122.648137	45.493082	12	0.7
1	259172515	259172517	15FEB2023:00:00:00	4223	48	20474	-122.648240	45.493070	12	0.8
2	259172515	259172517	15FEB2023:00:00:00	4223	57	20479	-122.648352	45.493123	12	0.8
3	259172515	259172517	15FEB2023:00:00:00	4223	73	20484	-122.648385	45.493262	12	0.7
4	259172515	259172517	15FEB2023:00:00:00	4223	112	20489	-122.648347	45.493582	12	0.8
...	...	...	...	...	...	...	...	...	...	...
156	259172515	259172530	15FEB2023:00:00:00	4223	5834	21389	-122.677057	45.528040	11	0.8
157	259172515	259172531	15FEB2023:00:00:00	4223	5838	21424	-122.677003	45.528037	6	2.0
158	259172515	259172531	15FEB2023:00:00:00	4223	5858	21429	-122.676765	45.528043	7	1.1
159	259172515	259172531	15FEB2023:00:00:00	4223	5889	21434	-122.676370	45.528047	10	1.1
160	259172515	259172531	15FEB2023:00:00:00	4223	5918	21439	-122.675990	45.528065	11	0.8

161 rows × 10 columns

Next steps: View recommended plots

C. [MUST] Filtering

Some of the columns in our TriMet data are not generally useful for our class project. For example, our contact at TriMet told us that the EVENT_NO_STOP column is not used and can be safely eliminated for any type of analysis of the data.

Use pandas.DataFrame.drop() to filter the EVENT_NO_STOP column.

For this in-class assignment we won't need the GPS_SATELLITES or GPS_HDOP columns, so drop them as well.

```
[6] # Use DataFrame.drop() to remove the EVENT_NO_STOP, GPS_SATELLITES, and GPS_HDOP columns
    columns_to_drop = ['EVENT_NO_STOP', 'GPS_SATELLITES', 'GPS_HDOP']
    data_filtered = data.drop(columns=columns_to_drop)
```

```
# Display the first few rows of the filtered DataFrame
print("\nFirst few rows of the DataFrame after dropping EVENT_NO_STOP, GPS_SATELLITES, and GPS_HDOP columns:")
print(data_filtered.head())
```

```
First few rows of the DataFrame after dropping EVENT_NO_STOP, GPS_SATELLITES, and GPS_HDOP columns:
   EVENT_NO_TRIP           OPD_DATE  VEHICLE_ID  METERS  ACT_TIME  \
0      259172515  15FEB2023:00:00:00        4223      40     20469
1      259172515  15FEB2023:00:00:00        4223      48     20474
2      259172515  15FEB2023:00:00:00        4223      57     20479
3      259172515  15FEB2023:00:00:00        4223      73     20484
4      259172515  15FEB2023:00:00:00        4223     112     20489

   GPS_LONGITUDE  GPS_LATITUDE
0    -122.648137     45.493082
1    -122.648240     45.493070
2    -122.648352     45.493123
3    -122.648385     45.493262
4    -122.648347     45.493582
```

Next, start over and this time try filtering these same columns using the usecols parameter of the read_csv() method.

```
[9] data_path = "/content/drive/MyDrive/bc_trip259172515_230215.csv"
```

```
[10] columns_to_include = ['EVENT_NO_TRIP', 'OPD_DATE', 'VEHICLE_ID', 'METERS', 'ACT_TIME', 'GPS_LONGITUDE', 'GPS_LATITUDE']
```

```
# Read the CSV file and include only the specified columns
df = pd.read_csv(data_path, usecols=columns_to_include)
```

```
[12] # Display the first few rows of the filtered DataFrame
     print("First few rows of the filtered DataFrame:")
     print(df.head())
```

```
First few rows of the filtered DataFrame:
   EVENT_NO_TRIP           OPD_DATE  VEHICLE_ID  METERS  ACT_TIME  \
0      259172515  15FEB2023:00:00:00        4223      40     20469
1      259172515  15FEB2023:00:00:00        4223      48     20474
2      259172515  15FEB2023:00:00:00        4223      57     20479
3      259172515  15FEB2023:00:00:00        4223      73     20484
4      259172515  15FEB2023:00:00:00        4223     112     20489

   GPS_LONGITUDE  GPS_LATITUDE
0    -122.648137     45.493082
1    -122.648240     45.493070
2    -122.648352     45.493123
3    -122.648385     45.493262
4    -122.648347     45.493582
```

Why might we want to filter columns this way instead of using drop()?

There are several benefits to utilizing the usecols parameter in pandas.read_csv() instead of DataFrame when filtering columns during file reading.after reading the file, drop()

It helps in Clarity and Maintainability which means that the code is more understandable and explicit when the columns to be included in the data reading step are specified. With just the

columns that are required, the DataFrame is instantly usable and specifies which columns are pertinent to the investigation.

It helps improve the performance during file reading, just the selected columns are read from the file and loaded into memory when you use usecols to specify columns to include. When working with huge datasets, in particular, this can greatly enhance efficiency and save memory use.

The efficiency would be improved because the dataset loads more quickly since just the designated columns are read into memory. This also saves you money by avoiding the overhead of loading extra columns at first and then deleting them later.

It helps in the reduced I/O because the input/output operations is decreased when you designate which columns to include since the file reading procedure only receives the data for those columns. It helps speed up the data loading.

Minimize Data Transformation: By using DataFrame.drop() to delete columns, you are essentially building a new DataFrame without those extraneous columns. If you can load the appropriate columns directly, you might not need to perform this modification.

D. [MUST] Decoding

Notice that the timestamp for each bread crumb record is encoded in an odd way that might make analysis difficult. The breadcrumb timestamps are represented by two columns, OPD_DATE and ACT_TIME. OPD_DATE merely represents the date on which the bus ran, and it should be constant, unchanging for all breadcrumb records for a single day. The ACT_TIME field indicates an offset, specifically the number of seconds elapsed since midnight on that day.

We're not sure why TriMet represents the breadcrumb timestamps this way. We do know that this encoding of the timestamps makes automated analysis difficult. So your job is to decode TriMet's representation and create a new "TIMESTAMP" column containing a pandas.Timestamp value for each breadcrumb.

Suggestions:

- Use DataFrame.apply() to apply a function to all rows of your DataFrame
- The applied function should input the two to-be-decoded columns, then it should:
    - create a datetime value from the OPD_DATE input using datetime.strptime()
    - create a timedelta value from the ACT_TIME
    - add the timedelta value to the datetime value to produce the resulting timestamp

```python
from datetime import datetime, timedelta
def create_timestamp(row):
    date = datetime.strptime(row['OPD_DATE'], '%d%b%Y:%H:%M:%S')
    # Convert ACT_TIME to a timedelta object (from seconds since midnight)
    time_offset = timedelta(seconds=row['ACT_TIME'])
    timestamp = date + time_offset
    return timestamp
```

```python
[15] # Apply the function to each row and create the TIMESTAMP column
     df['TIMESTAMP'] = df.apply(create_timestamp, axis=1)
```

```python
[16] # Display the first few rows of the DataFrame with the new TIMESTAMP column
     print("First few rows of the DataFrame with the new TIMESTAMP column:")
     print(df.head())
```

```
First few rows of the DataFrame with the new TIMESTAMP column:
   EVENT_NO_TRIP           OPD_DATE  VEHICLE_ID  METERS  ACT_TIME  \
0      259172515  15FEB2023:00:00:00        4223      40     20469
1      259172515  15FEB2023:00:00:00        4223      48     20474
2      259172515  15FEB2023:00:00:00        4223      57     20479
3      259172515  15FEB2023:00:00:00        4223      73     20484
4      259172515  15FEB2023:00:00:00        4223     112     20489

   GPS_LONGITUDE  GPS_LATITUDE           TIMESTAMP
0    -122.648137     45.493082 2023-02-15 05:41:09
1    -122.648240     45.493070 2023-02-15 05:41:14
2    -122.648352     45.493123 2023-02-15 05:41:19
3    -122.648385     45.493262 2023-02-15 05:41:24
4    -122.648347     45.493582 2023-02-15 05:41:29
```

E. [MUST] More Filtering

Now that you have decoded the timestamp you no longer need the OPD_DATE and ACT_TIME columns. Delete them from the DataFrame.

```
# Remove the OPD_DATE and ACT_TIME columns
df = df.drop(columns=['OPD_DATE', 'ACT_TIME'])
```

```
[19] # Display the first few rows of the DataFrame with the new TIMESTAMP column
print("First few rows of the DataFrame with the new TIMESTAMP column and without OPD_DATE and ACT_TIME columns:")
print(df.head())
```

```
First few rows of the DataFrame with the new TIMESTAMP column and without OPD_DATE and ACT_TIME columns:
   EVENT_NO_TRIP  VEHICLE_ID  METERS  GPS_LONGITUDE  GPS_LATITUDE  \
0      259172515        4223      40    -122.648137     45.493082
1      259172515        4223      48    -122.648240     45.493070
2      259172515        4223      57    -122.648352     45.493123
3      259172515        4223      73    -122.648385     45.493262
4      259172515        4223     112    -122.648347     45.493582

            TIMESTAMP
0 2023-02-15 05:41:09
1 2023-02-15 05:41:14
2 2023-02-15 05:41:19
3 2023-02-15 05:41:24
4 2023-02-15 05:41:29
```

F. [MUST] Enhance

Create a new column, called SPEED, that is a calculation of meters traveled per second. Calculate SPEED for each breadcrumb using the breadcrumb's METERS and TIMESTAMP values along with the METERS and TIMESTAMP values for the immediately preceding breadcrumb record.

Utilize the pandas.DataFrame.diff() method for this calculation. diff() allows you to calculate the difference between a cell value and the preceding row's value for that same column. Use diff() to create a new dMETERS column and then again to create a new dTIMESTAMP column. Then use apply() (with a lambda function) to calculate SPEED = dMETERS / dTIMESTAMP. Finally, drop the unneeded dMETERS And dTIMESTAMP columns.

```
[96] # Calculate the difference between rows for METERS and TIMESTAMP
df['dMETERS'] = df['METERS'].diff()
df['dTIMESTAMP'] = df['TIMESTAMP'].diff().dt.total_seconds()  # Convert timedelta to seconds
```

```
[97] # Calculate SPEED as dMETERS / dTIMESTAMP
df['SPEED'] = df['dMETERS'] / df['dTIMESTAMP']
```

```
df.drop(columns=['dMETERS', 'dTIMESTAMP'], inplace=True)
```

```
[99] # Display the first few rows of the DataFrame with the SPEED column
print("First few rows of the DataFrame with the SPEED column:")
print(df.head())
```

```
First few rows of the DataFrame with the SPEED column:
   EVENT_NO_TRIP  VEHICLE_ID  METERS  GPS_LONGITUDE  GPS_LATITUDE  \
0      259172515        4223      40    -122.648137     45.493082
1      259172515        4223      48    -122.648240     45.493070
2      259172515        4223      57    -122.648352     45.493123
3      259172515        4223      73    -122.648385     45.493262
4      259172515        4223     112    -122.648347     45.493582

            TIMESTAMP  SPEED
0 2023-02-15 05:41:09    NaN
1 2023-02-15 05:41:14    1.6
2 2023-02-15 05:41:19    1.8
3 2023-02-15 05:41:24    3.2
4 2023-02-15 05:41:29    7.8
```

```
[100] df['SPEED'] = df['SPEED'].fillna(method='bfill')
```

Question: What is the minimum, maximum and average speed for this bus on this trip? (Suggestion: use the Dataframe.describe() method to find these statistics)

```
[102] # Calculate and display the summary statistics for the SPEED column
      summary_statistics = df['SPEED'].describe()
      # Displaying the statistics
      print("Summary statistics for SPEED:")
      print(summary_statistics)
      # Extracting the minimum, maximum, and average speed
      min_speed = summary_statistics['min']
      max_speed = summary_statistics['max']
      average_speed = summary_statistics['mean']
      # Display the minimum, maximum, and average speed
      print("\nMinimum speed: {:.2f} m/s".format(min_speed))
      print("Maximum speed: {:.2f} m/s".format(max_speed))
      print("Average speed: {:.2f} m/s".format(average_speed))
```

```
Summary statistics for SPEED:
count    161.000000
mean       7.192254
std        4.429027
min        0.000000
25%        3.800000
50%        6.400000
75%       10.800000
max       17.400000
Name: SPEED, dtype: float64

Minimum speed: 0.00 m/s
Maximum speed: 17.40 m/s
Average speed: 7.19 m/s
```

G. [SHOULD] Larger Data Set

Here is breadcrumb data for the same bus TriMet for the entire day (February 15, 2023): bc_veh4223_230215.csv

Do the same transformations (parts C through F) for this larger data set. Be careful, you might need to treat each trip separately. For example, you might need to find all of the unique values for the EVENT_NO_TRIP column and then do the transformations separately on each trip.

Questions:
What was the maximum speed for vehicle #4223 on February 15, 2023?

Where and when did this maximum speed occur?

What was the median speed for this vehicle on this day?

```
[69] data.drop(columns=['OPD_DATE', 'ACT_TIME', 'GPS_SATELLITES', 'GPS_HDOP'], inplace=True)

     data['dMETERS'] = data['METERS'].diff()
     data['dTIMESTAMP'] = data['TIMESTAMP'].diff().dt.total_seconds()  # Convert timedelta to seconds

[71] data['SPEED'] = data['dMETERS'] / data['dTIMESTAMP']

[72] data.drop(columns=['dMETERS', 'dTIMESTAMP'], inplace=True)

[73] max_speed = data['SPEED'].max()
     median_speed = data['SPEED'].median()

[74] max_speed_row = data[data['SPEED'] == max_speed].iloc[0]

     print(f"Maximum speed for vehicle #4223 on February 15, 2023: {max_speed:.2f} m/s")
     print(f"Where and when did this maximum speed occur:")
     print(f"  GPS Longitude: {max_speed_row['GPS_LONGITUDE']}")
     print(f"  GPS Latitude: {max_speed_row['GPS_LATITUDE']}")
     print(f"  Timestamp: {max_speed_row['TIMESTAMP']}")
     print(f"\nMedian speed for this vehicle on this day: {median_speed:.2f} m/s")

Maximum speed for vehicle #4223 on February 15, 2023: 17.40 m/s
Where and when did this maximum speed occur:
  GPS Longitude: -122.660822
  GPS Latitude: 45.505452
  Timestamp: 2023-02-15 05:44:49

Median speed for this vehicle on this day: 7.20 m/s
```

H. [ASPIRE] Full Data Set

Here is breadcrumb data for all TriMet vehicles for the entire day (February 15, 2023):
bc_230215.csv

Do the same transformations (parts C through F) for the entire data set. Again, beware that simple transformations developed in parts C through F probably will need to be modified for the full data set which contains interleaved breadcrumbs from many vehicles.

Questions:
What was the maximum speed for any vehicle on February 15, 2023?

Where and when did this maximum speed occur?

Which vehicle had the fastest mean speed for any single trip on this day? Which vehicle and which trip achieved this fastest average speed?