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?
Alex - Assumptions and assertions that were/are wrong and create malformed data.

2. Should data transformation occur before data validation in your data pipeline or after?
Alex - It can occur in both, but I suspect it should happen after validation if it can be helped.

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.

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.

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

Why might we want to filter columns this way instead of using drop()?
Drop gives a modified table, and we *could* use in-place, but this isn't generally good practice. I would probably prefer doing this just to be hyper selective with what data we want to access. It cuts down on memory usage as well, and the

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

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.

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.

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)
The min speed was 0, the average speed was 6.4, the max was 17.4. Considering the distance was in meters, this must be in km/h which I find kind of interesting.

NO MORE CONTENT COMPLETED BEYOND THIS POINT

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?

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?