**I-540 ramp metering– How To Guide**

**Gracen Alleman, Intern 7/24/2024**

For the I-540 ramp metering, the ramp data from Q Free is processed in Python and visualised Power BI. The data consists of the *onramp occupancy detection*, *onramp volume*, *queue occupancy detection*, and mainline *speed*, *volume* and *occupancy*. All three data sets are read into python and when it is cleaned it is loaded into Power BI. If you wish to add more time periods, read in the three files of that period like the rest and add the files in at the merge for the respective data type, mainline, Onramp, and Queue.

**Python Instructions:**

* Load in the past reports which can be found in the **I-540 Ramp Meter Data Analysis** file.

> pd.read\_excel('data-SixForksOnrampMainlineVol-Apr25-May29.xlsx')

**>** pd.read\_csv('data-SixForksOnrampPassageVolume-May14-29.csv')

> pd.read\_csv('data-Six Forks Onramp Passage Volume-Apr25toMay14.csv')

> pd.read\_csv('data-Six Forks Onramp Queue Occ-Apr25toMay14.csv')

> pd.read\_csv('data-SixForksOnrampQueueOcc-May14-29.csv')

Run the cleaning lines for the data, which includes dropping the *speed* column for **onramp** and dropping the *speed* and *volume* data for **onramp** **occupancy**, convert the volume to volume per hour, and joining the previous data with the new data. This step localizes the time to US eastern time, as the assumption here is that data is in UTC time. This step renames the columns to identify what their origin data frame is. Finally doing a merge by time stamp for all three data sets.

* Drop unused columns *Speed* and *Volume*.

> On14=On14.drop('speed',axis='columns')

>On22=On22.drop('speed',axis='columns')

>Occ=Occ.drop(['volume','speed'],axis=1)

The naming system for the data frames are *Main* for the main line, *On* for the on ramp and *Occ* for the occupancy on ramp and the numbers after are to indicate when it is was loaded in to the model.

* Scale volume to the VPH.

>On25\_1['volume'] =On25\_1['volume'] \*12

>On25\_2["volume"] = On25\_2["volume"] \* 2

>On14['volume']=On14['volume']\*12

* Join the multiple time periods.

>On=pd.concat([On14, On25\_1,On25\_2,On22], ignore\_index=True, axis=0,join="inner")

>Main=pd.concat([Main,Main22],ignore\_index=True,axis=0,join="inner")

>Occ=pd.concat([Occ14,Occ25,Occ22],ignore\_index=True,axis=0,join="inner")

* Localize the data to Eastern Standard time.

>Occ['timestamp'] = pd.to\_datetime(Occ['timestamp'])

>Occ['timestamp\_local'] = Occ['timestamp'].dt.tz\_convert('US/Eastern').dt.tz\_localize(None)

>On['timestamp'] = pd.to\_datetime(On['timestamp'])

>On['timestamp\_local'] = On['timestamp'].dt.tz\_convert('US/Eastern').dt.tz\_localize(None)

>Main22['timestamp'] = pd.to\_datetime(Main22['timestamp'])

>Main22['timestamp']=Main22['timestamp'].dt.tz\_convert('US/Eastern').dt.tz\_localize(None)

* Renaming columns names.

>Main22.rename(columns = {'occupancy':'occupancy (pct)','volume':'volume (vph)','speed':'speed (mph)','timestamp':'timestamp\_local'}, inplace = True)

>Occ.rename(columns = {'occupancy':'occupancy Occ' }, inplace = True)

>On.rename(columns={'occupancy':'occupancy On','volume':'volume On'}, inplace = True

The naming system here is the name of the data that is being recorded followed by the name of the data frame that it is coming from. For the main line, it does not say its origin but instead says the shorthand for the units, which is the same units for all like columns.

* Merging all three datasets to one dataframe.

>temp = pd.merge(left=On,right=Occ,how="left",on='timestamp\_local').fillna("")

>df = pd.merge(left=temp,right=Main,how="left",on='timestamp\_local').fillna("")

The next step includes data manipulation. A day of the week column, *Weekday*, is added where 0 is Monday and 6 is Sunday. A column for *a.m. peak* is added, which is defined as 7 am to 9 am, Monday through Friday. The data frame is run through a function to replace the outliers with NA. This function is ran over 24 times to average by time of day.

* Add the column *weekday*.

>df['weekday'] = df['timestamp\_local'].dt.weekday

* Add the column *AMPeak*, which is a for loop of the data frame adding to a dictionary if is *weekday* and if in between the time 7 and 9.

>for index, row in df.iterrows():

>if df['weekday'].iloc[index] < 5:

>if df['timestamp\_local'].iloc[index].hour >= 7 and df['timestamp\_local'].iloc[index].hour < 9:

>AMPeak[index] = True

>df['AMPeak']= pd.Series(AMPeak)

* Replace outliers with nan accounting for time of day.

1. Time is separated within hour and minutes from timestamp local and made into a new *time* column.

>df['time'] = df['timestamp\_local'].dt.strftime('%H:%M')

1. A dictionary is made where the time is the key.

>for time, group in df.groupby('time'):

>group = group.drop(columns=['time'])

>dfs[time] = group

1. The function *relace\_outliers\_with\_nan* is called for all six numerical variables. For the three *occupancy* columns, the lower bound is treated as 0 and any 0 is not treated as a outlier, because 0 is expected in this column.

>dfs[time] = replace\_outliers\_with\_nan(dfs[time], 'occupancy Occ',no0=False,bounds=False,lowerbound=0)

>dfs[time] = replace\_outliers\_with\_nan(dfs[time], 'volume (vph)')

1. *Replace\_outliers\_with\_nan* uses the interquartile range and replaces outliers using the upper and lower bounds. Before running the test, any zeros are replaced with nans.

>df.loc[df[column] == 0, column] = np.nan

>q1 = np.nanquantile(df[column], 0.25)

>q3 = np.nanquantile(df[column], 0.75)

>iqr = q3 - q1

>upper\_bound = q3 + (1.5 \* iqr)

>lower\_bound = q1 - (1.5 \* iqr)

>df.loc[df[column] < lower\_bound, column] = np.nan

>df.loc[df[column] > upper\_bound, column] = np.nan

1. The data is returned to its original data frame structure with edits using the values of the dictionary to reattach.

>df2 = pd.concat(dfs.values())

Five more columns are added to create categorical variables. The groups for speed go from 20 to 90 in increments of 10 with 2 more groups added for 0 and greater than 90. The two occupancy categorical columns go from 0 to 100 in groups of 10 as a percent. For volume, the main line, which has higher numbers than the onramp, ranges from 1000 to 4000 in groups of 500 where the volume from onramp goes from 500 to 4000 in groups of 250 for the lower numbers going up to 2000.

* The numeric columns are run through for loops to bin the data and categorize variables.

>if df2['speed (mph)'].loc[c] == 0:

>LOS[c] = 0

>elif df2['speed (mph)'].loc[c] <= 20:

>LOS[c] = "20 or less" …

* The last python step is to write the csv for the data frame.

>df3.to\_csv('SixForks Mainline.csv', sep=',')

The data is then brought into the I-540 Power BI dashboard. The dashboard can be updated by refreshing the file **Six Forks Mainline** and having the python output go to the same file location that Power BI is read from. From here, there is one other file in Power BI that does not need any changes, as it is telling Power BI the sort order for the volume visualizations.

**Next steps for the I-540 ramp metering** are to get a more permanent installation of a data stream and a refresh rate of Power BI. Saving the data to a database as cleaned data would be better and more future proofed.