DataHack 2020 - Buissness Track

@author: Xinrui Zhan @author: Shawn Koo @Organizer: DS3 at UCSD

Report

Part 1

In the data cleaning and understanding part, we found out that there are a lot of null values.

In [157]:
```
barts_hotspots.isnull().sum()
```

Out[157]:
```
Date                                                  0
Origin Movement ID                                    0
Origin Display Name                                   0
Destination Movement ID                               0
Destination Display Name                              0
Daily Mean Travel Time (Seconds)                     58
Daily Range - Lower Bound Travel Time (Seconds)      58
Daily Range - Upper Bound Travel Time (Seconds)      58
AM Mean Travel Time (Seconds)                      1466
AM Range - Lower Bound Travel Time (Seconds)       1466
AM Range - Upper Bound Travel Time (Seconds)       1466
PM Mean Travel Time (Seconds)                       928
PM Range - Lower Bound Travel Time (Seconds)        928
PM Range - Upper Bound Travel Time (Seconds)        928
Midday Mean Travel Time (Seconds)                   727
Midday Range - Lower Bound Travel Time (Seconds)    727
Midday Range - Upper Bound Travel Time (Seconds)    727
Evening Mean Travel Time (Seconds)                 1026
Evening Range - Lower Bound Travel Time (Seconds)  1026
Evening Range - Upper Bound Travel Time (Seconds)  1026
Early Morning Mean Travel Time (Seconds)           2411
Early Morning Range - Lower Bound Travel Time (Seconds)    2411
Early Morning Range - Upper Bound Travel Time (Seconds)    2411
Day of the Week                                       0
dtype: int64
```

There are a lot of null values!!! Look closely, we know that for each time period, we have the same number of null values in lower bond, upper bond, and mean travel time. We made a resonable assumation that those rows are the same.

Also, don't worry, for part 1, we do not need to assess every null value. For now, we will just deal with null values in "Daily Mean Travel Time (Seconds)", "AM Mean Travel Time (Seconds)", and "PM Mean Travel Time (Seconds)", which we will use in our first part in our project. Also, we will focusing on rows that the destination is one of the three hotspots

In [158]:
```
display(dest_tb.head())
display(dest_tb.isnull().sum())
```

	Date	Origin	Dest	Daily	AM	PM	Day
0	04/28/2019	3603	3394	639.0	NaN	664.0	Sunday
1	05/18/2019	3603	3394	730.0	NaN	732.0	Saturday
2	04/23/2019	3603	3394	682.0	NaN	759.0	Tuesday
3	05/13/2019	3603	3394	634.0	NaN	650.0	Monday
4	04/14/2019	3603	3396	996.0	NaN	NaN	Sunday

```
Date         0
Origin       0
Dest         0
Daily       44
AM         842
PM         448
Day          0
dtype: int64
```

In [159]:
```
# Take a closer look
null_tb = dest_tb.isnull().assign(**{'Day': dest_tb["Day"]})
null_tb.groupby('Day').mean()
```

Out[159]:

Day	Date	Origin	Dest	Daily	AM	PM
Friday	0.0	0.0	0.0	0.017094	0.452991	0.200855
Monday	0.0	0.0	0.0	0.035556	0.555556	0.213333
Saturday	0.0	0.0	0.0	0.034188	0.508547	0.405983
Sunday	0.0	0.0	0.0	0.038462	0.589744	0.444444
Thursday	0.0	0.0	0.0	0.008547	0.482906	0.217949
Tuesday	0.0	0.0	0.0	0.038462	0.534188	0.192308
Wednesday	0.0	0.0	0.0	0.017094	0.495726	0.247863

What we know?

1. Sunday and Saturday clearly have more null values in
2. AM clear has more null values than PM
3. We have a really small number of null values in Daily

So how do we gonna to do with null values?

A: use the average number groupby day of the week, source, destination to replace the null values in PM and AM

In [160]:
```
clean.isnull().sum()
```

Out[160]:
```
Day         0
Origin      0
Dest        0
Daily       0
AM        309
PM        130
dtype: int64
```

Much less null values!!! Notice that there are still some null values because for some specific combination of origin, destination, and the day of the week, there is not even a one value. However, this will not be a problem for us since if there is not even a one value, it means that nobody takes that ride in that spefic day from that spefici origin to that spefic destination. As a result, it means it is a bad choice (crowd wisdom!!). We will explain more in the report part.

Also, we will use the clean table in following whole part 1

In [161]:
```
display(fisher.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(fisher.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through
```

Origin	AM	PM
3603	624.187451	727.192563
3692	810.809063	912.457459
3760	829.887428	1006.778158

		Daily
Day	**Origin**	
Friday	**3603**	674.615385
	3692	833.923077
	3760	928.346154
Monday	**3603**	655.200000
	3692	755.760000
	3760	861.160000
Saturday	**3603**	681.961538
	3692	789.846154
	3760	914.038462
Sunday	**3603**	656.346154
	3692	727.730769
	3760	860.307692
Thursday	**3603**	674.538462
	3692	829.692308
	3760	900.000000
Tuesday	**3603**	651.423077
	3692	778.615385
	3760	888.346154
Wednesday	**3603**	661.076923
	3692	802.730769
	3760	902.576923

1. With the destination of 3394, if we consider the time of the day (am or pm), we should choose BART 3603 for both am and pm

2. Withe the destination of 3394, if we consider the day of the week, we should choose BART 3603 for all days
3. The reason behind is that 3603 is close enough that the time and the day factor could not even affect our decision

In [162]:
```
display(park.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(park.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through th
```

Origin	AM	PM
3603	547.150015	771.707182
3692	455.616483	597.204420
3760	795.395880	907.986552

Day	Origin	Daily
Friday	**3603**	613.730769
	3692	506.653846
	3760	780.615385
Monday	**3603**	579.800000
	3692	482.920000
	3760	748.840000
Saturday	**3603**	509.769231
	3692	385.076923
	3760	807.423077
Sunday	**3603**	518.346154
	3692	416.076923
	3760	757.961538
Thursday	**3603**	630.307692
	3692	546.692308
	3760	774.730769
Tuesday	**3603**	596.538462
	3692	506.923077
	3760	762.961538
Wednesday	**3603**	605.653846
	3692	529.576923
	3760	796.807692

1. With the destination of 3792, if we consider the time of the day (am or pm), we should choose

BART 3692 for both am and pm

2. Withe the destination of 3792, if we consider the day of the week, we should choose BART 3692 for all days
3. The reason behind is that 3692 is close enough that the time and the day factor could not even affect our decision

In [164]:
```
# solve p1q1p1
palace = clean.loc[clean['Dest'] == 3396] # get the table with the destination o
display(palace.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(palace.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through
```

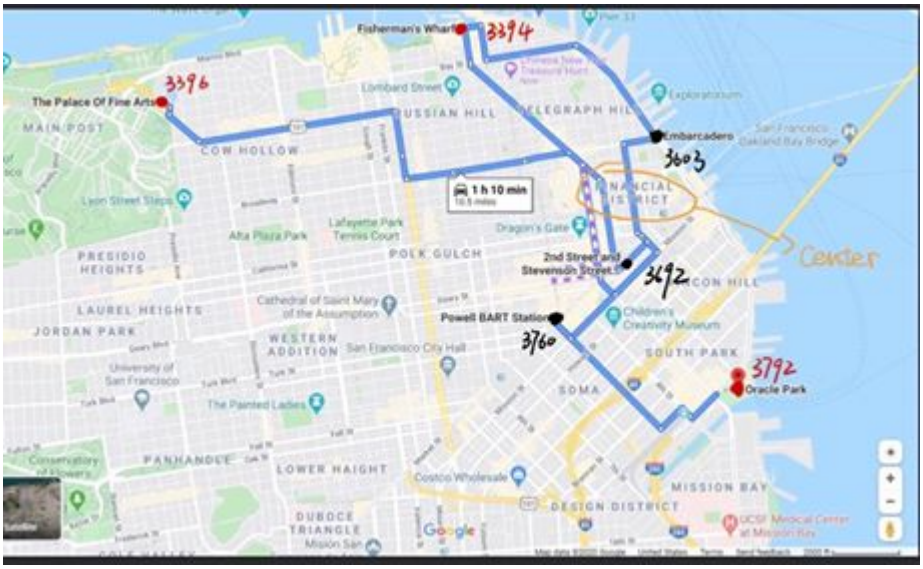
Origin	AM	PM
3603	1108.083333	1345.408668
3692	1241.040000	1430.269016
3760	1476.000000	1558.883495

Day	Origin	Daily
Friday	**3603**	1202.576923
	3692	1337.961538
	3760	1305.681818
Monday	**3603**	1174.560000
	3692	1284.240000
	3760	1351.235294
Saturday	**3603**	1181.000000
	3692	1259.269231
	3760	1293.526316
Sunday	**3603**	1164.681818
	3692	1209.360000
	3760	1220.681818
Thursday	**3603**	1178.615385
	3692	1314.230769
	3760	1318.708333
Tuesday	**3603**	1201.961538
	3692	1293.884615
	3760	1281.352941
Wednesday	**3603**	1210.153846
	3692	1296.269231
	3760	1309.363636

1. With the destination of 3396, if we consider the time of the day (am or pm), we should choose BART 3603 for both am and pm
2. Withe the destination of 3396, if we consider the day of the week, we should choose BART 3603 for all days

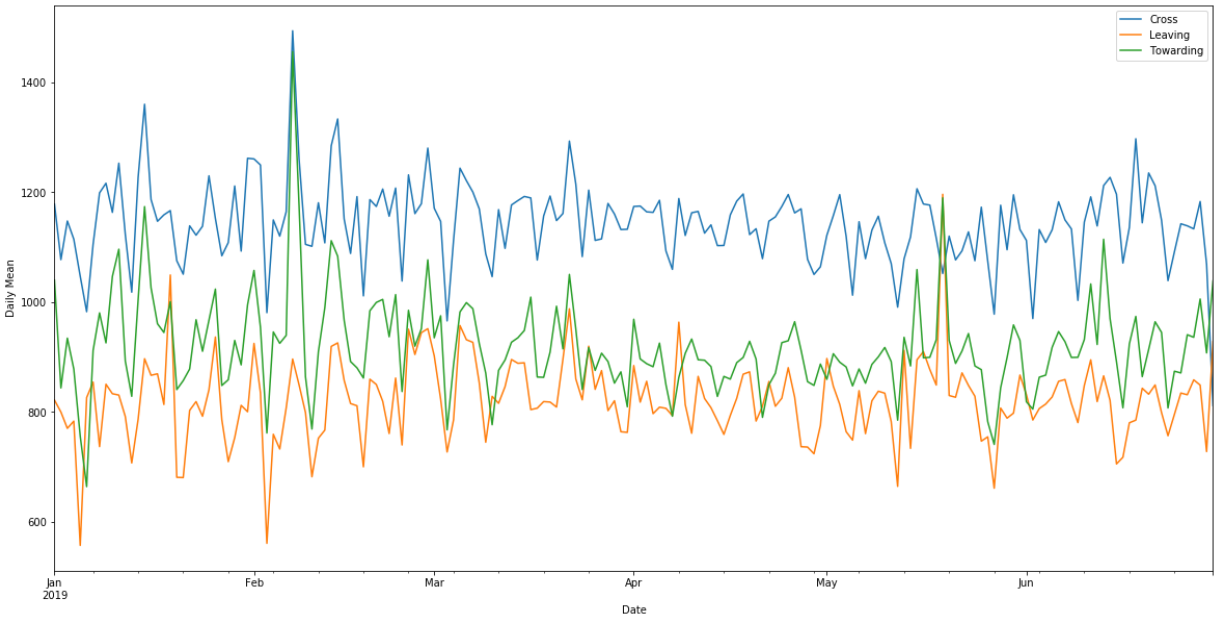
3. The reason behind is that 3603 is close enough that the time and the day factor could not even affect our decision

Part 1 Problem 2: We decide to choose the financial district as our "center". Any trip that is leaving from financial district is counted as "Leaving" and any trip that is going toward "financial district" is countes as "Heading". The trip which accrose the center will be labeled as "Cross".



Now, we reorganize our table and then we visulized each comparson based on "the day of the week", "time of the day", "dirction".

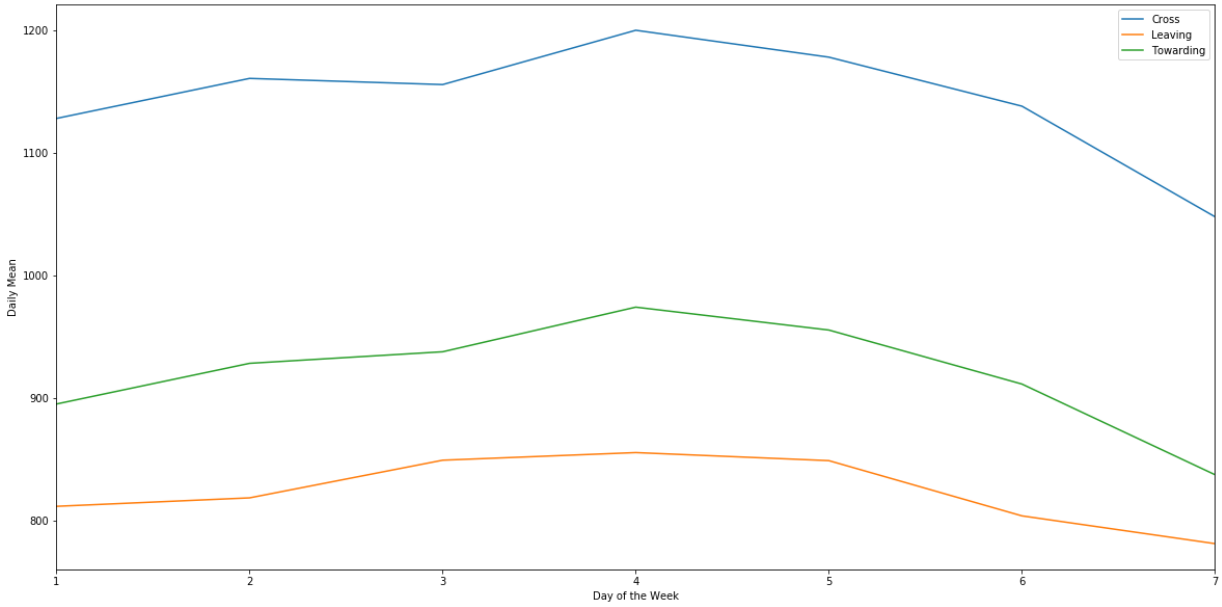
The visualization of each direction across each day!



This is the visualization by date groupby by three different direction What we know:

1. All three different direction all share similar pattern
2. In general, it have a peak on Feb and seems have a weekly pattern inside.

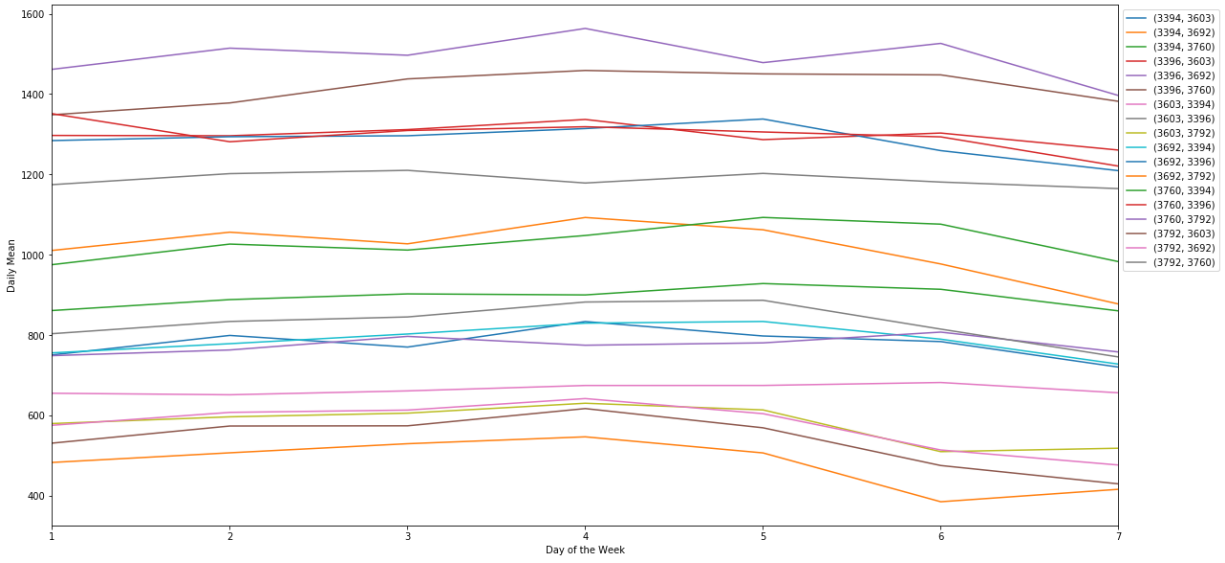
The visualization of each direction of each the day of the week



We can see that:

1. Cross typical have the longest average travel time in general and leaving have the shortest. This is reasonable since cross trips ususally have a large distance and also leaving from downtown ususally takes less time than heading toward downtown.
2. Also, all lines will have a minimum on Sunday and a slightly peak on Thursday
3. The general trend is going up from monday to thursday and then decrease from friday to sunday

The visualization of each speficic combination of origin and destination of each day of the week

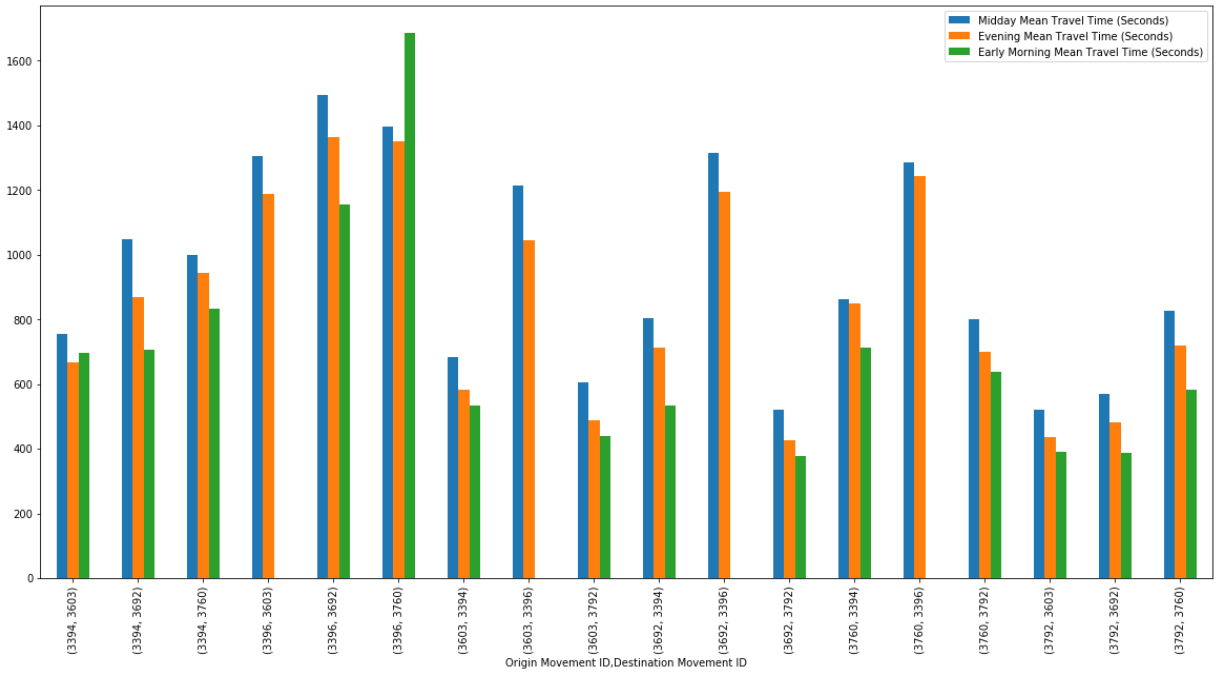


We can see that:

1. In general, the difference between each combination of origin and destination (distance of the trip) will have a much bigger effect than the the day of the week

2. In general, there will be a minimum for each line on Sunday

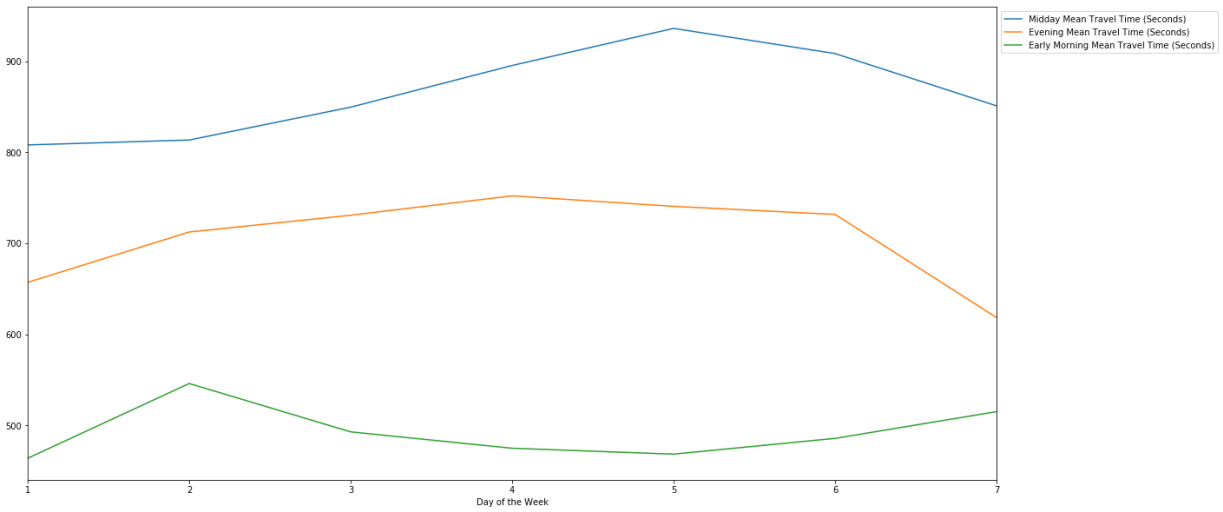
The visualization of the factor of time



What we can see:

1. In general, the travel time from long to short is : Midday > Evening > Early Morning. We do not know the spcific time period for each stage but we can reasonbal guess that in early morning, there are less cars on roads and thus have a shorter average travel time
2. There is a special case, (3396, 3760), which has the longes travel time in early morning, we will evaluate this special case later

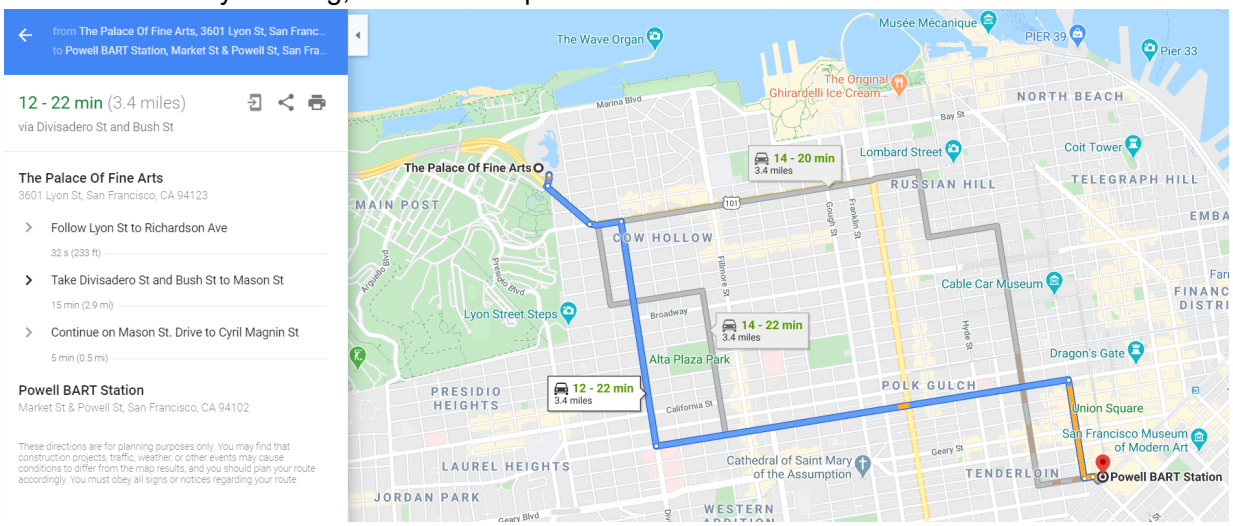
The visualization of factors of day of the week and also the time of the day



What we see:

1. We have a new found! Usually travel time will be shorter on Sunday. However, the early morning mean travel time will slightly increase on Sunday.

Now, we did a small research on this special case go take a look at the google map! In order to show it is the early morning, we set the depature time to 6:00 am



set it to 12:00 pm


noon

It is pretty odd. Why our data is not consistant with google map? Maybe it is time to take a closer look to the data set

So our guess is that there are some apartments around the Palace of Fine Arts and people who usually will get up in the morning and go to the downtown for work. As a result, there will be more cars on the early morning and thus cause a lower mean travel time

Recomendation:

1. Just like what we found on the part 1, choose the best station for each hotspot
2. The distance is always the major factor, so do not consider too much (day of the week, time) both for visitors and uber drivers
3. Sunday morning seems like a really good choice for visitors if you wanna save your time on road
4. From the special case, the palace of fine art will have a higher traffic volumn in the early morning, be careful!

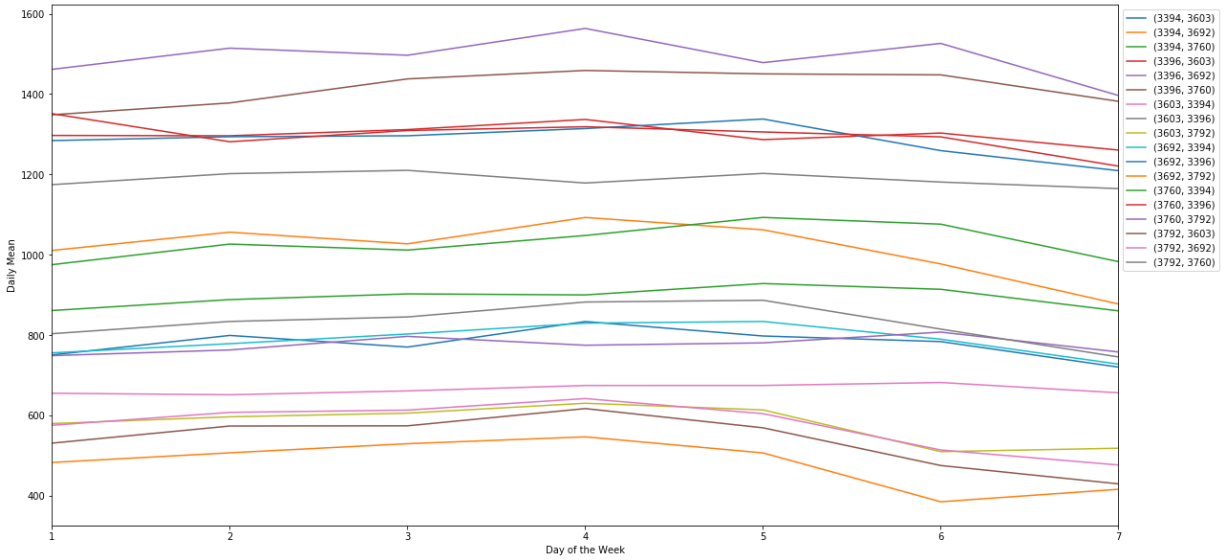
Part 2

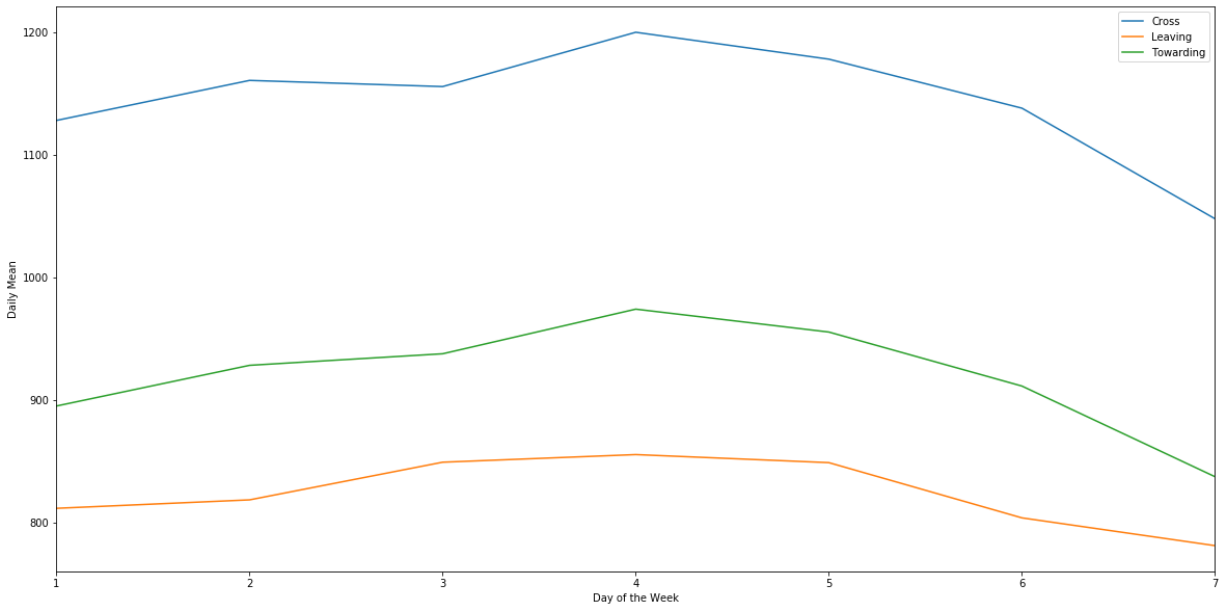
What we know from part 1? What we are gonna do in part 2?

1. The major effect is the distance between two places. However, we do not have the distance data in this project. Thus we we will not include distance as a major factor to our time series model.

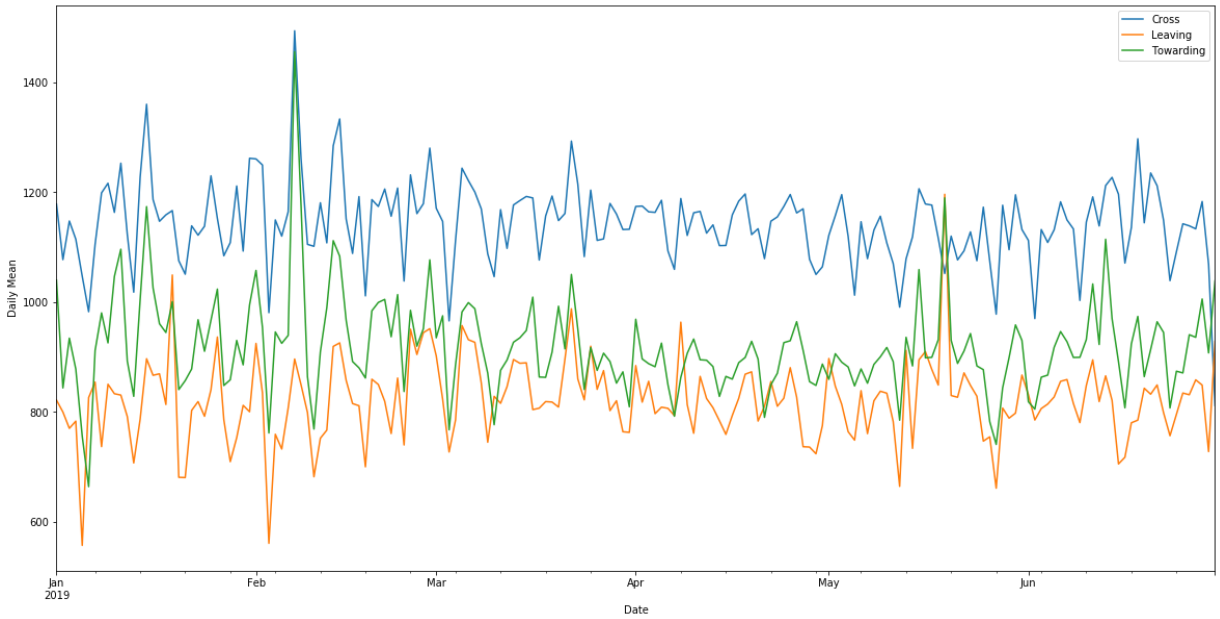
2. We will try to build a model to predict the next two quarters's travel time for each day. However, we did not have enough time, so instead of that, we caculate the next two quarter's travel time based on patterns we found
3. First we used quadratic regression to get the weekly pattern
4. Then we divide the weekly pattern out of our dates data and then we get the visualization of weeks number and travel time. We get all the scalers for each week.
5. Use a simple linear equation to caculate the avarage mean in next two quartar
6. Use the weeks' scalers, we can calculate every week's average travel time in the next two quarter
7. Then use the quadratic equation we regressed before, we caculate every day's data in that week

The pattern of the day of the week

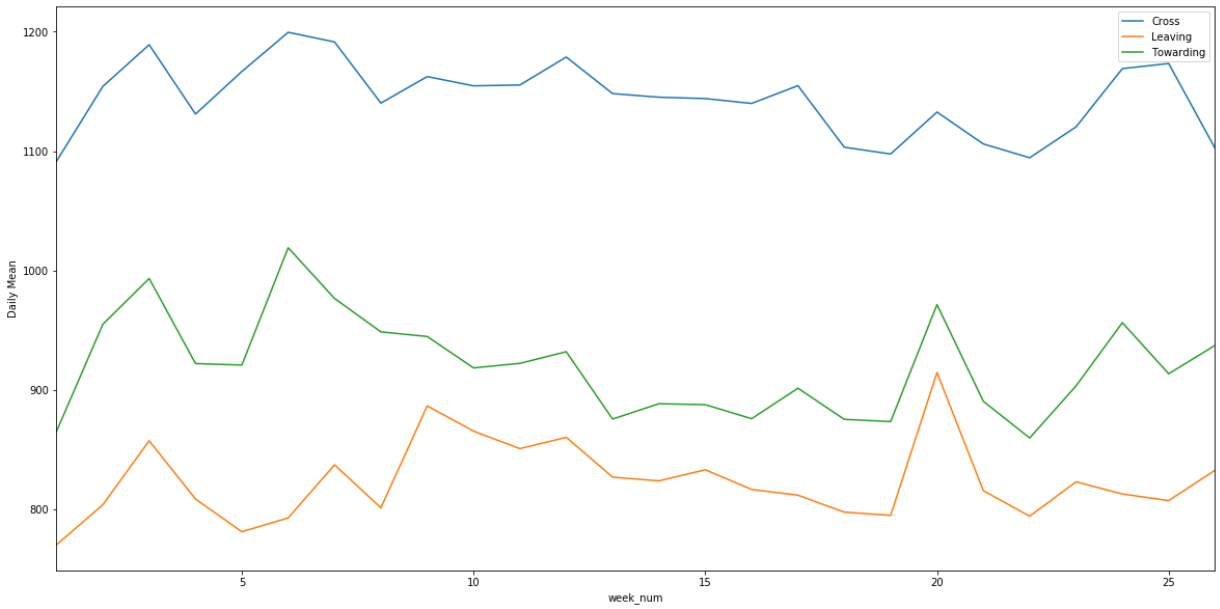




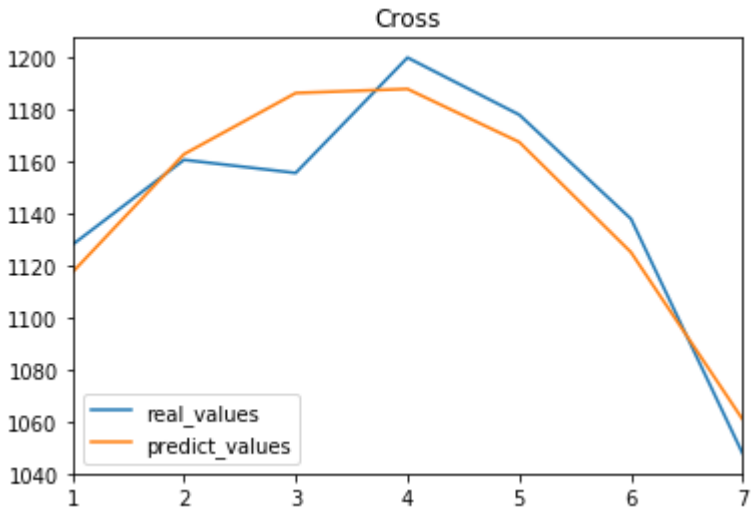
The visualization of all dates

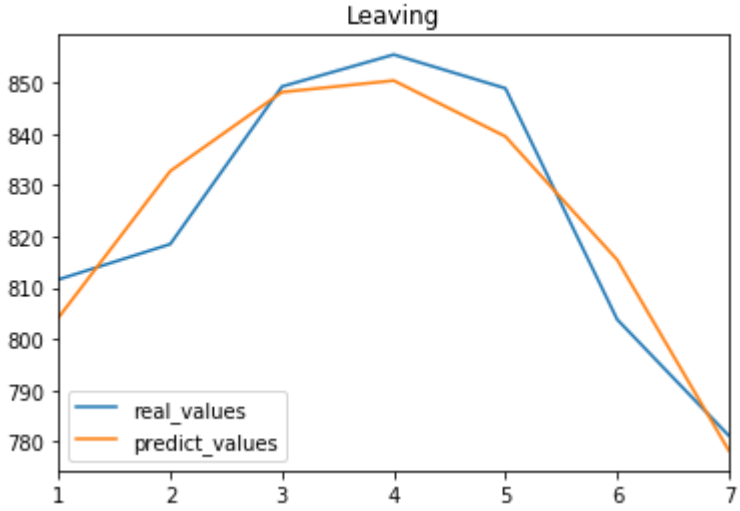


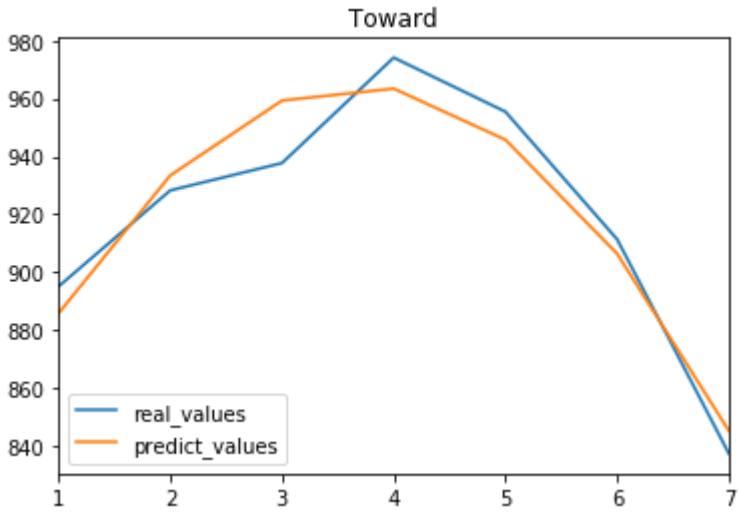
The visualization of weeks number



We used sklearn pipeline to do the quaratic regression, and here is outcome!







And here are the coefs!

In [167]:
```
display(cross_coef, leaving_coef, toward_coef)
display("y = c + bx + ax^2")
```

```
array([1050.06271816,   78.38955191,  -10.97602663])

array([762.44516335,  48.28519046,  -6.57643269])

array([816.40940464,  80.25170493, -10.87639811])

'y = c + bx + ax^2'
```

Now we will do:

4. Then we divide the weekly pattern out of our dates data and then we get the visualization of weeks number and travel time. We get all the scalers for each week.
5. Use a simple linear equation to caculate the avarage mean in next two quartar

In [170]:
```
# the scaler table
scaled_weekly
```

Out[170]:

coordinate	Cross	Leaving	Towarding
week_num			
1	0.953952	0.934522	0.940443
2	1.008809	0.975265	1.037992
3	1.039307	1.040312	1.079745
4	0.988550	0.981057	1.002251
5	1.019703	0.947834	1.000876
6	1.048422	0.961880	1.107766
7	1.041367	1.015838	1.061553
8	0.996572	0.972149	1.031115
9	1.015989	1.075787	1.026899
10	1.009215	1.050097	0.998302
11	1.009839	1.032316	1.002451
12	1.030285	1.043819	1.013011
13	1.003533	1.003373	0.951713
14	1.000880	0.999534	0.965652
15	0.999881	1.010852	0.964836
16	0.996322	0.990792	0.952023
17	1.009339	0.984973	0.979800
18	0.964264	0.967722	0.951452
19	0.959300	0.964481	0.949405
20	0.990017	1.109652	1.055962
21	0.966667	0.989554	0.967894
22	0.956553	0.963631	0.934457
23	0.979154	0.998568	0.981885
24	1.021826	0.986203	1.039506
25	1.025572	0.979528	0.992956
26	0.963505	1.010362	1.018828

In [171]:
```
# then step 5
display(avg_q3, avg_q4, avg_both)
```

```
746.7172529999999

736.239504

741.4783785
```

Finish Step 6

6. Use the weeks' scalers, we can calculate every week's average travel time in the next two quarter

In [169]:
```
nxt_26
```

Out[169]:

coordinate	Cross	Leaving	Towarding
week_num			
1	707.334658	692.928173	697.318221
2	748.009931	723.137639	769.648688
3	770.623592	771.368790	800.607690
4	732.988163	727.432242	743.147650
5	756.087892	702.798658	742.127747
6	777.382230	713.213435	821.384808
7	772.151229	753.221918	787.118645
8	738.936691	720.827402	764.549171
9	753.333515	797.672894	761.423661
10	748.310829	778.624542	740.219541
11	748.773749	765.439988	743.295701
12	763.934392	773.969246	751.125927
13	744.098253	743.979311	705.674427
14	742.130841	741.133037	716.009859
15	741.390169	749.524955	715.405319
16	738.751522	734.650878	705.904728
17	748.403413	730.335958	726.500195
18	714.980560	717.544924	705.480796
19	711.300343	715.141496	703.963621
20	734.076026	822.782846	782.973232
21	716.762803	733.732725	717.672342
22	709.263493	714.511393	692.879435
23	726.021211	740.416878	728.046501
24	757.661821	731.247973	770.771061
25	760.439343	726.299129	736.255078
26	714.417934	749.161210	755.439268

The final step

7. Then use the quadratic equation we regressed before, we caculate every day's data in that week

Is the most exciting part and interesting part. However, sadly we do not have that time!!!!!!

Code

In [96]:
```
# import libraries
import numpy
import pandas as pd
import numpy as np
from IPython.display import Image
import matplotlib.pyplot as plt
import datetime
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
%matplotlib inline
```

In [3]:
```
# import Data
barts_hotspots = pd.read_csv('Dataset/barts_hotspots.csv')
barts_to_all = pd.read_csv('Dataset/barts_to_all.csv')
hotspots_to_all = pd.read_csv('Dataset/hotspots_to_all.csv')
hours_q1 = pd.read_csv('Dataset/hours_q1.csv')
hours_q2 = pd.read_csv('Dataset/hours_q2.csv')
```

Data cleanning and understanding

In [4]:
```
# ger the list of barts and hotspots
barts_lst = barts_to_all["Origin Movement ID"].unique()
hotspots_lst = hotspots_to_all["Origin Movement ID"].unique()
display(barts_lst, hotspots_lst)
```

```
array([3603, 3692, 3760], dtype=int64)
```

```
array([3394, 3792, 3396], dtype=int64)
```

In [5]:
```
# the names of barts and hotspots
tb_hot_bart = pd.Series(barts_hotspots.groupby('Origin Movement ID')["Origin Dis
display(tb_hot_bart)
```

Origin Movement ID	Origin Display Name
3394	Fisherman's Wharf, San Francisco, CA
3396	The Palace Of Fine Arts, 3601 Lyon St, San Fra...
3603	Embarcadero, San Francisco, CA
3692	2nd Street and Stevenson Street (Montgomery BA...
3760	Powell BART Station, Market St and Powell St, ...
3792	Oracle Park, 24 Willie Mays Plaza, San Francis...

In [6]:
```
# add a day of the week column to bars_hotspots table
barts_hotspots["Day of the Week"] = pd.to_datetime(barts_hotspots["Date"]).dt.da
```

Null values

In [7]:
```
barts_hotspots.isnull().sum()
```

Out[7]:
```
Date                                                  0
Origin Movement ID                                    0
Origin Display Name                                   0
Destination Movement ID                               0
Destination Display Name                              0
Daily Mean Travel Time (Seconds)                     58
Daily Range - Lower Bound Travel Time (Seconds)      58
Daily Range - Upper Bound Travel Time (Seconds)      58
AM Mean Travel Time (Seconds)                      1466
AM Range - Lower Bound Travel Time (Seconds)       1466
AM Range - Upper Bound Travel Time (Seconds)       1466
PM Mean Travel Time (Seconds)                       928
PM Range - Lower Bound Travel Time (Seconds)        928
PM Range - Upper Bound Travel Time (Seconds)        928
Midday Mean Travel Time (Seconds)                   727
Midday Range - Lower Bound Travel Time (Seconds)    727
Midday Range - Upper Bound Travel Time (Seconds)    727
Evening Mean Travel Time (Seconds)                 1026
Evening Range - Lower Bound Travel Time (Seconds)  1026
Evening Range - Upper Bound Travel Time (Seconds)  1026
Early Morning Mean Travel Time (Seconds)           2411
Early Morning Range - Lower Bound Travel Time (Seconds)    2411
Early Morning Range - Upper Bound Travel Time (Seconds)    2411
Day of the Week                                       0
dtype: int64
```

There are a lot of null values!!! Look closely, we know that for each time period, we have the same

number of null values in lower bond, upper bond, and mean travel time. We made a resonable assumation that those rows are the same.

Also, don't worry, for part 1, we do not need to assess every null value. For now, we will just deal with null values in "Daily Mean Travel Time (Seconds)", "AM Mean Travel Time (Seconds)", and "PM Mean Travel Time (Seconds)", which we will use in our first part in our project. Also, we will focusing on rows that the destination is one of the three hotspots

In [8]:
```
# get the table that the destination is one of three hotspots, and rename it to b
dest_tb = barts_hotspots.loc[barts_hotspots['Destination Movement ID'].apply(laml
dest_tb = dest_tb.loc[:,["Date", "Origin Movement ID", "Destination Movement ID"
dest_tb = (
        dest_tb.rename(columns = {
            "Origin Movement ID": "Origin",
            "Destination Movement ID": "Dest",
            "Daily Mean Travel Time (Seconds)": "Daily",
            "AM Mean Travel Time (Seconds)": "AM",
            "PM Mean Travel Time (Seconds)": "PM",
            "Day of the Week": "Day"
        }).reset_index(drop=True)
)
display(dest_tb.head())
display(dest_tb.isnull().sum())
```

	Date	Origin	Dest	Daily	AM	PM	Day
0	04/28/2019	3603	3394	639.0	NaN	664.0	Sunday
1	05/18/2019	3603	3394	730.0	NaN	732.0	Saturday
2	04/23/2019	3603	3394	682.0	NaN	759.0	Tuesday
3	05/13/2019	3603	3394	634.0	NaN	650.0	Monday
4	04/14/2019	3603	3396	996.0	NaN	NaN	Sunday

```
Date        0
Origin      0
Dest        0
Daily      44
AM        842
PM        448
Day         0
dtype: int64
```

Much less!!! Good Job

In [9]:
```
# Take a closer look
null_tb = dest_tb.isnull().assign(**{'Day': dest_tb["Day"]})
null_tb.groupby('Day').mean()
```

Out[9]:

Day	Date	Origin	Dest	Daily	AM	PM
Friday	0.0	0.0	0.0	0.017094	0.452991	0.200855
Monday	0.0	0.0	0.0	0.035556	0.555556	0.213333
Saturday	0.0	0.0	0.0	0.034188	0.508547	0.405983
Sunday	0.0	0.0	0.0	0.038462	0.589744	0.444444
Thursday	0.0	0.0	0.0	0.008547	0.482906	0.217949
Tuesday	0.0	0.0	0.0	0.038462	0.534188	0.192308
Wednesday	0.0	0.0	0.0	0.017094	0.495726	0.247863

What we know?

1. Sunday and Saturday clearly have more null values in
2. AM clear has more null values than PM
3. We have a really small number of null values in Daily

So how do we gonna to do with null values?

A: use the average number groupby day of the week, source, destination to replace the null values in PM and AM

In [10]:
```
# get the dictionary for values replacing null values
dict_null = dest_tb.groupby(['Day', 'Origin', "Dest"]).mean().to_dict()
```

In [11]:
```
dest_tb.head()
```

Out[11]:

	Date	Origin	Dest	Daily	AM	PM	Day
0	04/28/2019	3603	3394	639.0	NaN	664.0	Sunday
1	05/18/2019	3603	3394	730.0	NaN	732.0	Saturday
2	04/23/2019	3603	3394	682.0	NaN	759.0	Tuesday
3	05/13/2019	3603	3394	634.0	NaN	650.0	Monday
4	04/14/2019	3603	3396	996.0	NaN	NaN	Sunday

In [12]:
```
def cleaner(row):
    lst = ["Daily", "AM", "PM"]
    day = row['Day']
    origin = row['Origin']
    dest = row['Dest']
    answer = [day, origin, dest]
    for col in lst:
        if np.isnan(row[col]):
            if dict_null[col][day, origin, dest] != np.nan:
                answer.append(dict_null[col][day, origin, dest])
        else:
            answer.append(row[col])
    ind = ['Day', 'Origin', "Dest", "Daily", "AM", "PM"]
    return pd.Series(answer, index=ind)
```

In [13]:
```
clean = dest_tb.apply(cleaner, axis=1)
```

In [14]:
```
clean.isnull().sum()
```

Out[14]:
```
Day         0
Origin      0
Dest        0
Daily       0
AM        309
PM        130
dtype: int64
```

Much less null values!!! Notice that there are still some null values because for some specific combination of origin, destination, and the day of the week, there is not even a one value. However, this will not be a problem for us since if there is not even a one value, it means that nobody takes that ride in that spefic day from that spefici origin to that spefic destination. As a result, it means it is a bad choice (crowd wisdom!!). We will explain more in the report part.

Also, we will use the clean table in following whole part 1

Part 1

In [15]:
```
# solve p1q1p1
fisher = clean.loc[clean['Dest'] == 3394] # get the table with the destination o
display(fisher.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(fisher.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through
```

Origin	AM	PM
3603	624.187451	727.192563
3692	810.809063	912.457459
3760	829.887428	1006.778158

Day	Origin	Daily
Friday	**3603**	674.615385
	3692	833.923077
	3760	928.346154
Monday	**3603**	655.200000
	3692	755.760000
	3760	861.160000
Saturday	**3603**	681.961538
	3692	789.846154
	3760	914.038462
Sunday	**3603**	656.346154
	3692	727.730769
	3760	860.307692
Thursday	**3603**	674.538462
	3692	829.692308
	3760	900.000000
Tuesday	**3603**	651.423077
	3692	778.615385
	3760	888.346154
Wednesday	**3603**	661.076923
	3692	802.730769
	3760	902.576923

1. With the destination of 3394, if we consider the time of the day (am or pm), we should choose BART 3603 for both am and pm
2. Withe the destination of 3394, if we consider the day of the week, we should choose BART 3603 for all days

3. The reason behind is that 3603 is close enough that the time and the day factor could not even affect our decision

In [16]:
```
# solve p1q1p1
palace = clean.loc[clean['Dest'] == 3396] # get the table with the destination o
display(palace.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(palace.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through
```

	AM	PM
Origin		
3603	1108.083333	1345.408668
3692	1241.040000	1430.269016
3760	1476.000000	1558.883495

		Daily
Day	**Origin**	
Friday	**3603**	1202.576923
	3692	1337.961538
	3760	1305.681818
Monday	**3603**	1174.560000
	3692	1284.240000
	3760	1351.235294
Saturday	**3603**	1181.000000
	3692	1259.269231
	3760	1293.526316
Sunday	**3603**	1164.681818
	3692	1209.360000
	3760	1220.681818
Thursday	**3603**	1178.615385
	3692	1314.230769
	3760	1318.708333
Tuesday	**3603**	1201.961538
	3692	1293.884615
	3760	1281.352941
Wednesday	**3603**	1210.153846
	3692	1296.269231
	3760	1309.363636

1. With the destination of 3396, if we consider the time of the day (am or pm), we should choose

BART 3603 for both am and pm

2. Withe the destination of 3396, if we consider the day of the week, we should choose BART 3603 for all days
3. The reason behind is that 3603 is close enough that the time and the day factor could not even affect our decision

In [17]:
```
# solve p1q1p1
park = clean.loc[clean['Dest'] == 3792] # get the table with the destination of 
display(park.groupby('Origin')['AM', 'PM'].mean()) # through the day
display(park.groupby(['Day', 'Origin'])['Daily'].mean().to_frame()) # through th
```

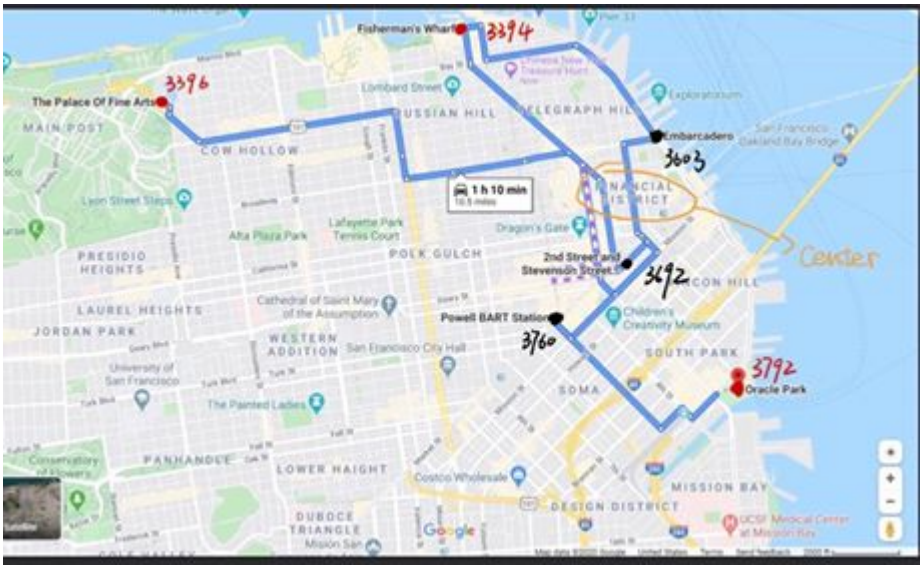
Origin	AM	PM
3603	547.150015	771.707182
3692	455.616483	597.204420
3760	795.395880	907.986552

Day	Origin	Daily
Friday	**3603**	613.730769
	3692	506.653846
	3760	780.615385
Monday	**3603**	579.800000
	3692	482.920000
	3760	748.840000
Saturday	**3603**	509.769231
	3692	385.076923
	3760	807.423077
Sunday	**3603**	518.346154
	3692	416.076923
	3760	757.961538
Thursday	**3603**	630.307692
	3692	546.692308
	3760	774.730769
Tuesday	**3603**	596.538462
	3692	506.923077
	3760	762.961538
Wednesday	**3603**	605.653846
	3692	529.576923
	3760	796.807692

1. With the destination of 3792, if we consider the time of the day (am or pm), we should choose BART 3692 for both am and pm
2. Withe the destination of 3792, if we consider the day of the week, we should choose BART 3692 for all days

3. The reason behind is that 3692 is close enough that the time and the day factor could not even affect our decision

Part 1 Problem 2: We decide to choose the financial district as our "center". Any trip that is leaving from financial district is counted as "Leaving" and any trip that is going toward "financial district" is countes as "Heading". The trip which accrose the center will be labeled as "Cross".



Now, lets label each trips in table barts_hotspots. According to the map, we will label each BART station and each hotspot a coordinate (x, y) in order to represent the geological relations. Given financial disrict as (0,0), we dicide to label:

1. 3603: (1, 1)
2. 3692: (-1, -1)
3. 3760: (-2, -1)
4. 3792: (2, -2)
5. 3394: (-3, 3)
6. 3396: (-4, 2)

Since we only care the sign of each coordinates' difference, the value of each x and y are simply represented the rank of that direction. For example, 3396 is third right to the center and second above to the center so we give it (-3,2)

In [18]:
```
# since now the purpose is to compare each trip, in order to save time, we do not
dct_xy = {3603: (1, 1), 3692: (1, -1), 3760: (-1, -1), 3792: (1, -2), 3394: (-2,
# the function to generate the direction for each trip
def direction_judge(start, end):
    start = dct_xy[start]
    end = dct_xy[end]
    j1 = start[0] * end[0]
    j2 = start[1] * end[1]
    d1 = start[0]**2 + start[1]**2
    d2 = end[0]**2 + end[1]**2
    if j1 < 0 and j2 < 0:
        return "Cross"
    if d1 < d2:
        return "Leaving"
    if d1 >= d2:
        return "Towarding"
compare = barts_hotspots.assign(**{'coordinate': barts_hotspots.apply(lambda x: (
```

In [19]:
```
compare.head()
```

Out[19]:

	Date	Origin Movement ID	Origin Display Name	Destination Movement ID	Destination Display Name	Daily Mean Travel Time (Seconds)	Daily Range - Lower Bound Travel Time (Seconds)	Daily Range Upper Bound Travel Time (Seconds
0	06/09/2019	3396	The Palace Of Fine Arts, 3601 Lyon St, San Fra...	3603	Embarcadero, San Francisco, CA	1588.0	1054.0	2392.(
1	04/28/2019	3603	Embarcadero, San Francisco, CA	3394	Fisherman's Wharf, San Francisco, CA	639.0	512.0	796.(
2	05/18/2019	3603	Embarcadero, San Francisco, CA	3394	Fisherman's Wharf, San Francisco, CA	730.0	520.0	1024.(
3	04/18/2019	3792	Oracle Park, 24 Willie Mays Plaza, San Francis...	3603	Embarcadero, San Francisco, CA	512.0	371.0	705.(
4	04/08/2019	3792	Oracle Park, 24 Willie Mays Plaza, San Francis...	3603	Embarcadero, San Francisco, CA	505.0	364.0	701.(

5 rows × 25 columns

In [54]:
```
# compare of direction based on every day
compare['Date'] = compare['Date'].apply(pd.to_datetime)
compare_date = compare.groupby(["coordinate", "Date"])['Daily Mean Travel Time (S
compare_date
```

Out[54]:

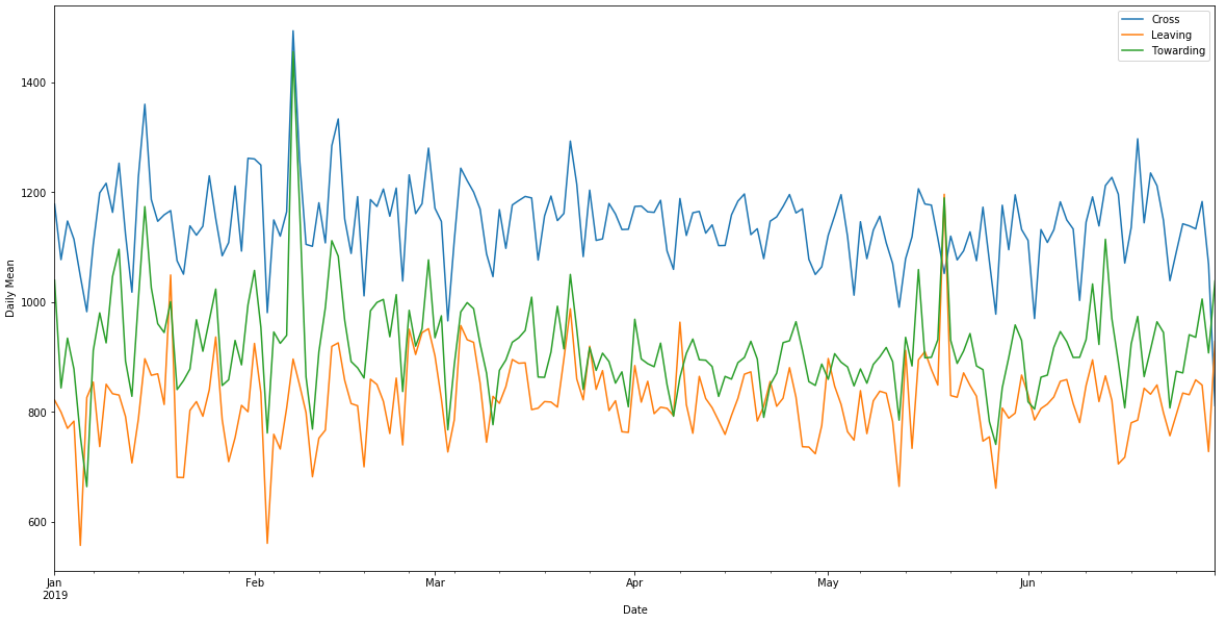
		Daily Mean Travel Time (Seconds)
coordinate	**Date**	
Cross	**2019-01-01**	1178.500000
	2019-01-02	1077.500000
	2019-01-03	1147.750000
	2019-01-04	1114.750000
	2019-01-05	1047.250000
	2019-01-06	982.750000
	2019-01-07	1105.250000
	2019-01-08	1199.000000
	2019-01-09	1216.500000
	2019-01-10	1163.500000
	2019-01-11	1253.000000
	2019-01-12	1123.750000
	2019-01-13	1018.250000
	2019-01-14	1227.750000
	2019-01-15	1360.250000
	2019-01-16	1187.000000
	2019-01-17	1147.250000
	2019-01-18	1159.000000
	2019-01-19	1166.750000
	2019-01-20	1075.500000
	2019-01-21	1051.000000
	2019-01-22	1139.000000
	2019-01-23	1122.000000
	2019-01-24	1138.250000
	2019-01-25	1230.000000
	2019-01-26	1152.250000
	2019-01-27	1084.500000
	2019-01-28	1108.500000
	2019-01-29	1211.500000
	2019-01-30	1093.000000
...	**...**	...

Daily Mean Travel Time (Seconds)

coordinate	Date	
Towarding	**2019-06-01**	818.857143
	2019-06-02	805.428571
	2019-06-03	863.571429
	2019-06-04	867.285714
	2019-06-05	917.857143
	2019-06-06	946.571429
	2019-06-07	928.142857
	2019-06-08	899.428571
	2019-06-09	899.714286
	2019-06-10	932.000000
	2019-06-11	1033.285714
	2019-06-12	923.000000
	2019-06-13	1114.571429
	2019-06-14	970.000000
	2019-06-15	891.714286
	2019-06-16	807.833333
	2019-06-17	924.142857
	2019-06-18	974.142857
	2019-06-19	864.571429
	2019-06-20	914.000000
	2019-06-21	964.428571
	2019-06-22	945.000000
	2019-06-23	807.571429
	2019-06-24	874.428571
	2019-06-25	871.285714
	2019-06-26	940.857143
	2019-06-27	936.142857
	2019-06-28	1005.857143
	2019-06-29	907.714286
	2019-06-30	1038.666667

543 rows × 1 columns

In [55]:
```
# plot the data
tb_date_plot = compare_date.pivot_table(columns = "coordinate", index = "Date", 
ax = tb_date_plot.plot(figsize=(20,10))
ax.set_ylabel('Daily Mean')
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



This is the visualization by date groupby by three different direction What we know:

1. All three different direction all share similar pattern
2. In general, it have a peak on Feb and seems have a weekly pattern inside.

In [20]:
```
# compare and contrast based on day of the week
# replace the words of day of the week to numbers 1-7
dct_day = {"Monday": 1, "Tuesday": 2, "Wednesday": 3, "Thursday": 4, "Friday": 5
compare_day = compare.assign(**{"Day of the Week": compare["Day of the Week"].ap
compare_day = compare_day.groupby(["Origin Movement ID", "Destination Movement I
compare_day
```

Out[20]:

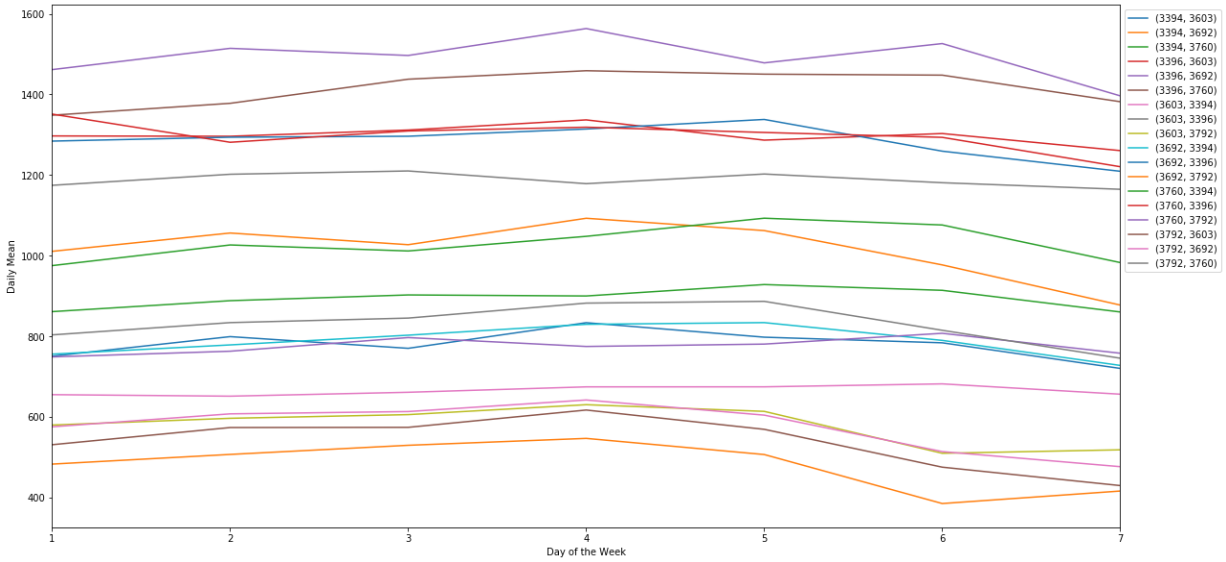
			Daily Mean Travel Time (Seconds)
Origin Movement ID	**Destination Movement ID**	**Day of the Week**	
3394	**3603**	**1**	750.720000
		2	799.153846
		3	770.153846
		4	833.730769
		5	797.769231
		6	783.846154
		7	720.307692
	3692	**1**	1010.760000
		2	1056.192308
		3	1027.230769
		4	1092.807692
		5	1062.230769
		6	977.115385
		7	877.423077
	3760	**1**	975.360000
		2	1026.576923
		3	1011.653846
		4	1047.961538
		5	1092.961538
		6	1076.153846
		7	982.884615
3396	**3603**	**1**	1297.041667
		2	1296.230769
		3	1311.692308
		4	1336.846154
		5	1286.615385
		6	1303.000000
		7	1260.650000
	3692	**1**	1461.640000

Origin Movement ID	Destination Movement ID	Day of the Week	Daily Mean Travel Time (Seconds)
		2	1514.384615
...	...	...	...
3760	3396	6	1293.526316
		7	1220.681818
	3792	1	748.840000
		2	762.961538
		3	796.807692
		4	774.730769
		5	780.615385
		6	807.423077
		7	757.961538
3792	3603	1	531.000000
		2	573.615385
		3	574.115385
		4	617.038462
		5	569.192308
		6	475.307692
		7	429.615385
	3692	1	575.480000
		2	607.576923
		3	613.269231
		4	641.961538
		5	604.384615
		6	513.961538
		7	476.538462
	3760	1	803.480000
		2	833.923077
		3	845.153846
		4	882.307692
		5	886.692308
		6	814.884615
		7	745.560000

126 rows × 1 columns

In [21]:
```
it~

ta
 compare_day.pivot_table(columns = ("Origin Movement ID", "Destination Movement ID
lot.plot(figsize=(20,10))
('Daily Mean')
x_to_anchor=(1,1))
```



We can see that:

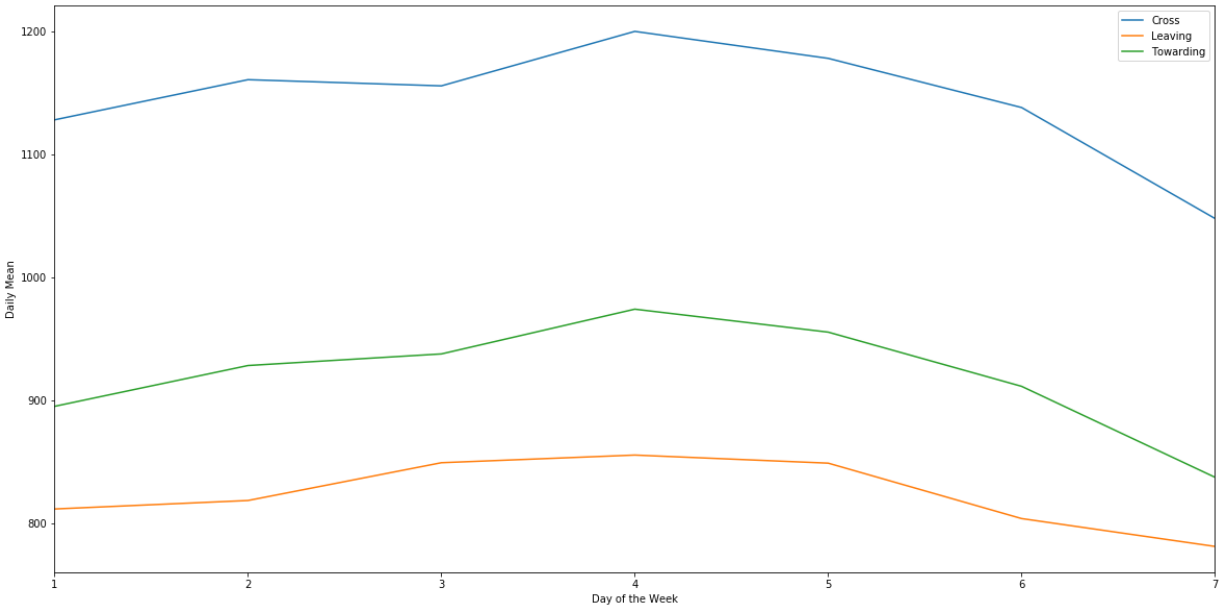
1. In general, the difference between each combination of origin and destination (distance of the trip) will have a much bigger effect than the the day of the week
2. In general, there will be a minimum for each line on Sunday

In [80]:
```
 compare and contrast based on the direction and the day of the week
ompare_day_dir = compare.assign(**{"Day of the Week": compare["Day of the Week"].
ompare_day_dir = compare_day_dir.groupby(["coordinate", "Day of the Week"])['Dail
ompare_day_dir
```

Out[80]:

		Daily Mean Travel Time (Seconds)
coordinate	**Day of the Week**	
Cross	**1**	1128.100000
	2	1160.769231
	3	1155.730769
	4	1200.048077
	5	1178.105769
	6	1138.105769
	7	1047.843137
Leaving	**1**	811.574850
	2	818.468208
	3	849.202247
	4	855.422222
	5	848.848315
	6	803.775862
	7	781.109195
Towarding	**1**	895.068966
	2	928.254144
	3	937.692308
	4	974.098901
	5	955.417582
	6	911.333333
	7	837.352601

In [23]:
```
# plot the data
tb_day_plot = compare_day_dir.pivot_table(columns = "coordinate", index = "Day of
ax = tb_day_plot.plot(figsize=(20,10))
ax.set_ylabel('Daily Mean')
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



We can see that:

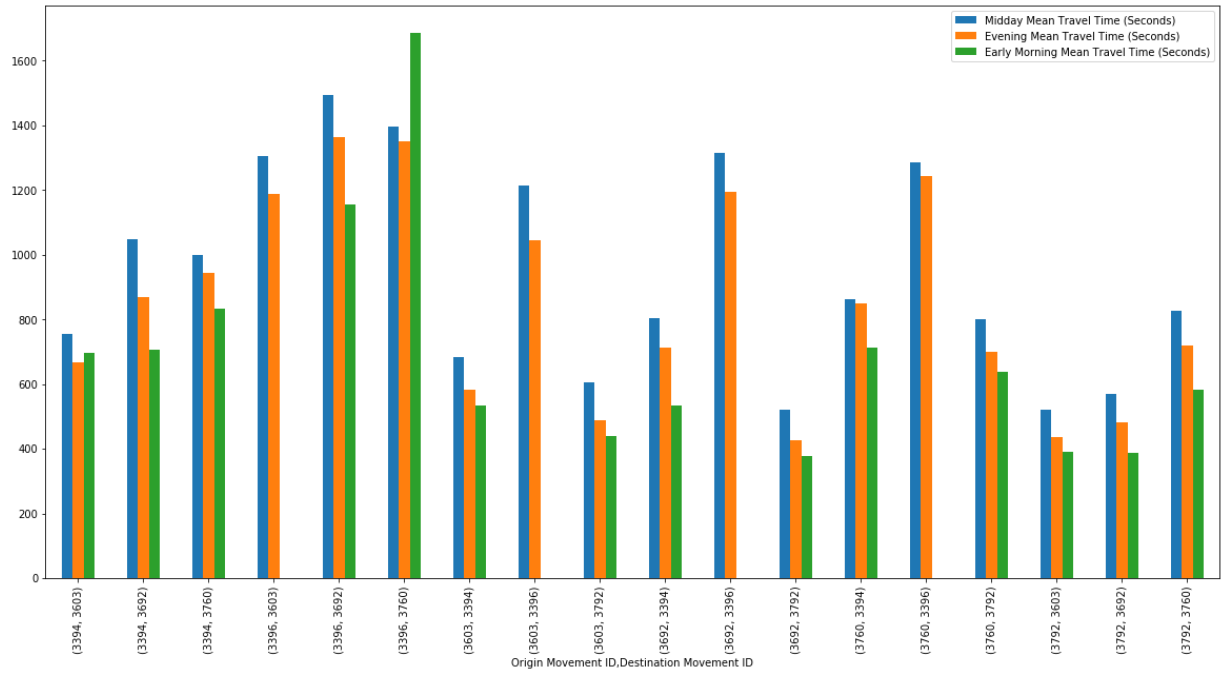
1. Cross typical have the longest average travel time in general and leaving have the shortest. This is reasonable since cross trips ususally have a large distance and also leaving from downtown ususally takes less time than heading toward downtown.
2. Also, all lines will have a minimum on Sunday and a slightly peak on Thursday
3. The general trend is going up from monday to thursday and then decrease from friday to sunday

In [24]:
```
# compare and contrast based on the time of the day
compare_time = compare.groupby(["Origin Movement ID", "Destination Movement ID"]
compare_time
```

Out[24]:

Origin Movement ID	Destination Movement ID	Midday Mean Travel Time (Seconds)	Evening Mean Travel Time (Seconds)	Early Morning Mean Travel Time (Seconds)
3394	**3603**	754.877095	667.865497	697.307692
	3692	1049.000000	870.788889	705.333333
	3760	999.672222	945.329609	831.833333
3396	**3603**	1304.377358	1188.700000	NaN
	3692	1495.503937	1363.133333	1156.727273
	3760	1396.802632	1351.777778	1685.000000
3603	**3394**	682.435754	581.982759	534.600000
	3396	1214.508197	1045.547368	NaN
	3792	606.734807	488.677778	438.892655
3692	**3394**	804.513966	711.474860	532.333333
	3396	1313.676190	1195.585859	NaN
	3792	522.238889	425.688889	376.831461
3760	**3394**	863.150000	849.376404	713.000000
	3396	1286.710526	1245.000000	NaN
	3792	799.848214	699.551402	636.900000
3792	**3603**	520.222222	435.596591	391.422535
	3692	570.005556	481.744444	386.072727
	3760	828.360000	719.909091	584.000000

In [25]:
```
# plot the data
tb_day_plot = compare_time
ax = tb_day_plot.plot(kind = 'bar',figsize=(20,10))
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



What we can see:

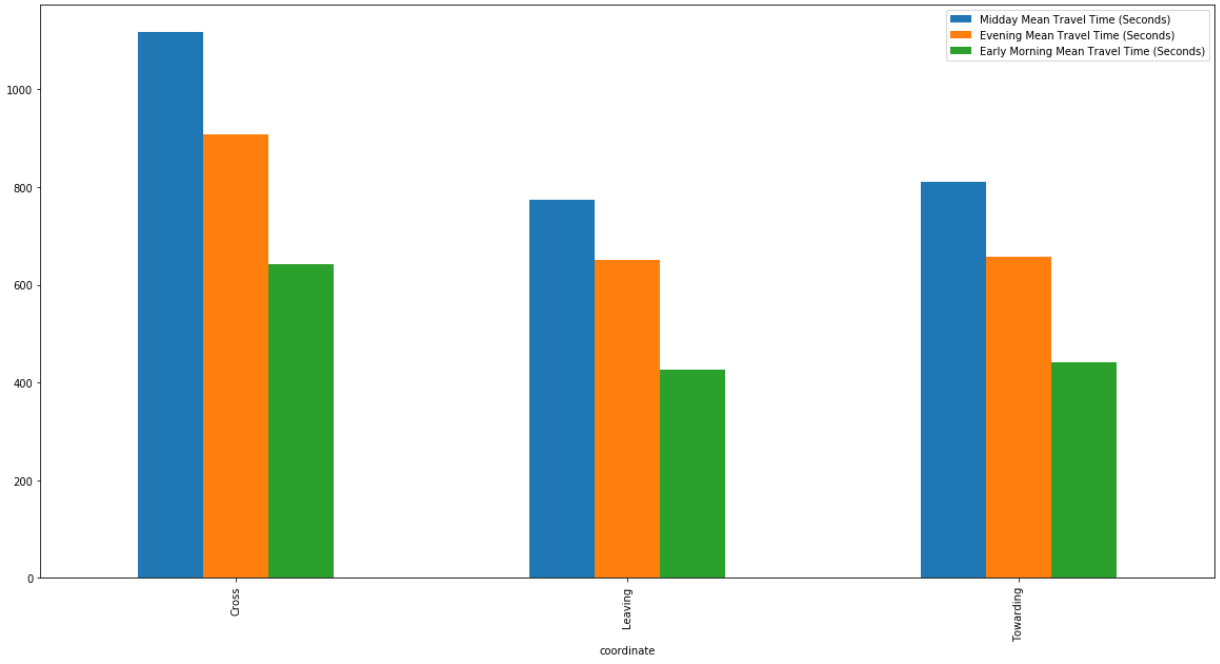
1. In general, the travel time from long to short is : Midday > Evening > Early Morning. We do not know the spcific time period for each stage but we can reasonbal guess that in early morning, there are less cars on roads and thus have a shorter average travel time
2. There is a special case, (3396, 3760), which has the longes travel time in early morning, we will evaluate this special case later

In [26]:
```
# compare and contrast based on the time of the day
compare_time_dir = compare.groupby("coordinate")["Midday Mean Travel Time (Secon
compare_time_dir
```

Out[26]:

coordinate	Midday Mean Travel Time (Seconds)	Evening Mean Travel Time (Seconds)	Early Morning Mean Travel Time (Seconds)
Cross	1117.923858	908.510246	642.406375
Leaving	774.186492	650.748913	425.694301
Towarding	811.635021	657.915049	441.633333

In [27]:
```
# plot the data
tb_day_plot = compare_time_dir
ax = tb_day_plot.plot(kind = 'bar',figsize=(20,10))
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



What we can see:

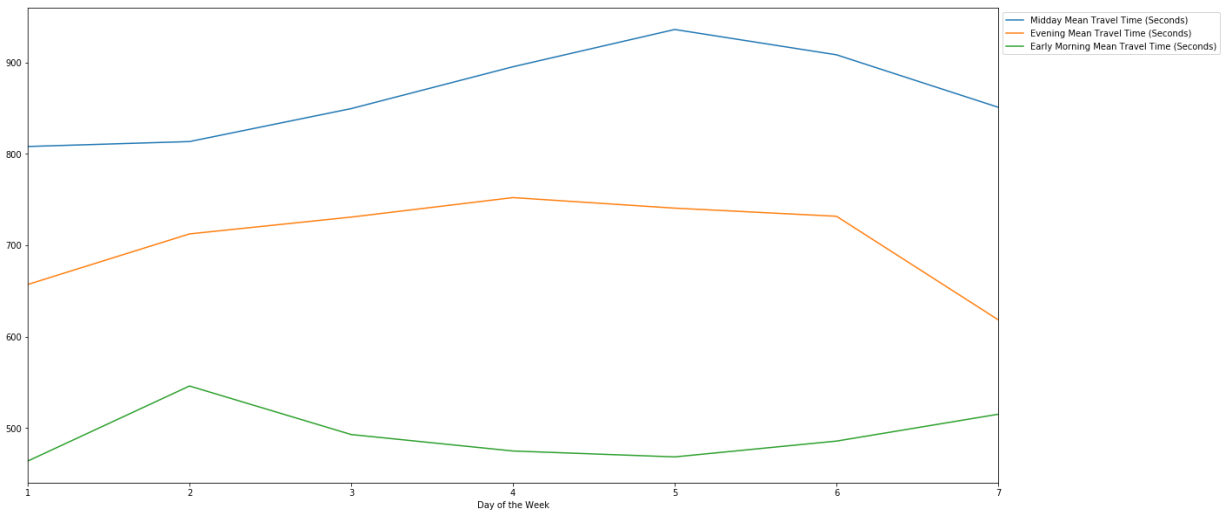
Basically the same with the one before

In [30]:
```
the week and the time of the day
ign(**{"Day of the Week": compare["Day of the Week"].apply(lambda x: dct_day[x])})
e_day.groupby("Day of the Week")["Midday Mean Travel Time (Seconds)", "Evening Mea
```

Out[30]:

Day of the Week	Midday Mean Travel Time (Seconds)	Evening Mean Travel Time (Seconds)	Early Morning Mean Travel Time (Seconds)
1	808.046012	657.013378	463.725275
2	813.440476	712.368902	545.885496
3	849.538682	730.781609	492.753968
4	895.337950	752.106936	474.737705
5	936.125000	740.563798	468.282443
6	908.417722	731.676375	485.589744
7	850.984211	618.343396	514.968992

In [31]:
```
# plot the data
tb_day_plot = compare_time_day
ax = tb_day_plot.plot(figsize=(20,10))
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



What we see:

1. We have a new found! Usually travel time will be shorter on Sunday. However, the early morning mean travel time will slightly increase on Sunday.

In [231]:
```
# special case (3396, 3760)
# first, what are those two locations
temp_dct = tb_hot_bart.to_dict()["Origin Display Name"]
display(temp_dct[3396], temp_dct[3760])
```

```
'The Palace Of Fine Arts, 3601 Lyon St, San Francisco, CA'

'Powell BART Station, Market St and Powell St, San Francisco, CA'
```

```
go take a look at the google map! In order to show it is the early morning, we
set the depature time to 6:00 am
![specialcase morning](Dataset/specialcase1.PNG)
set it to 12:00 pm
![noon](Dateset/specialcase2.PNG)
```

It is pretty odd. Why our data is not consistant with google map? Maybe it is time to take a closer look to the data set

In [239]:
```
special_case = barts_hotspots.loc[(barts_hotspots['Origin Movement ID'] == 3396)
special_case.head()
```

			Lyon St, San Fra...		and Powell St, ...				
1109	06/02/2019	3396	The Palace Of Fine Arts, 3601 Lyon St, San Fra...	3760	Powell BART Station, Market St and Powell St, ...	1292.0	1055.0	1583.(	

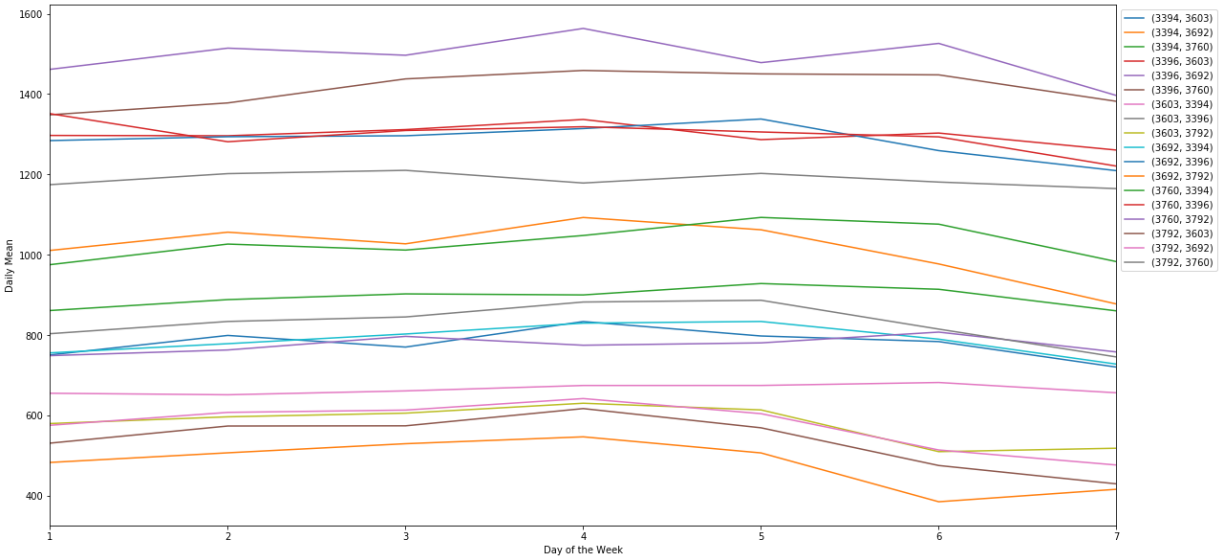
Part 2

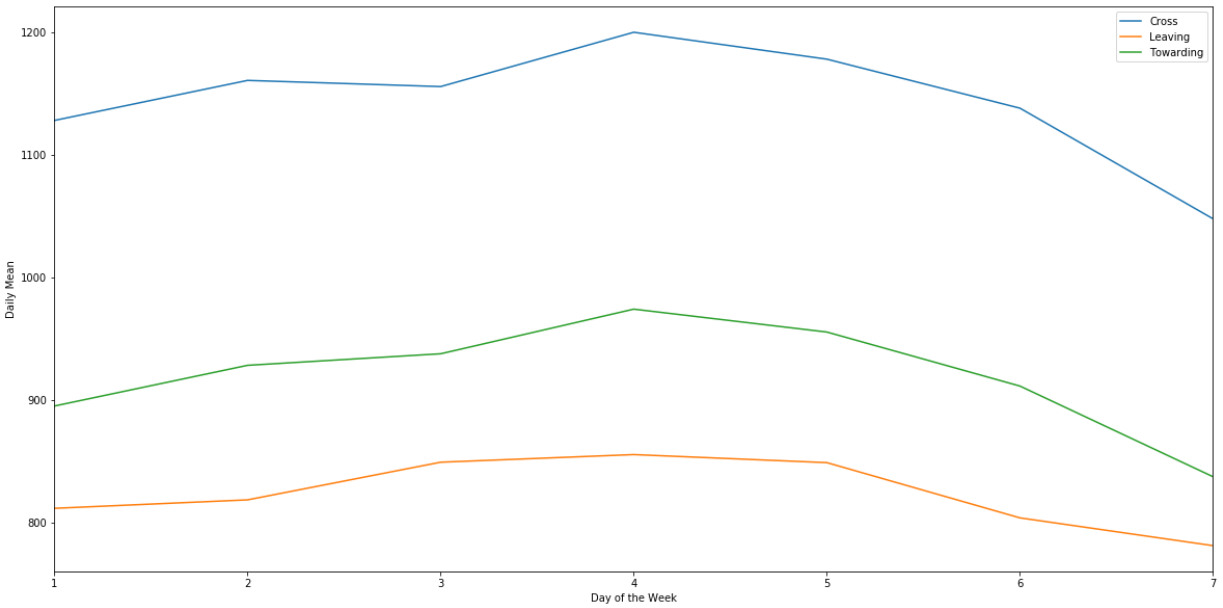
What we know from part 1? What we are gonna do in part 2?

1. The major effect is the distance between two places. However, we do not have the distance data in this project. Thus we we will not include distance as a major factor to our time series model.

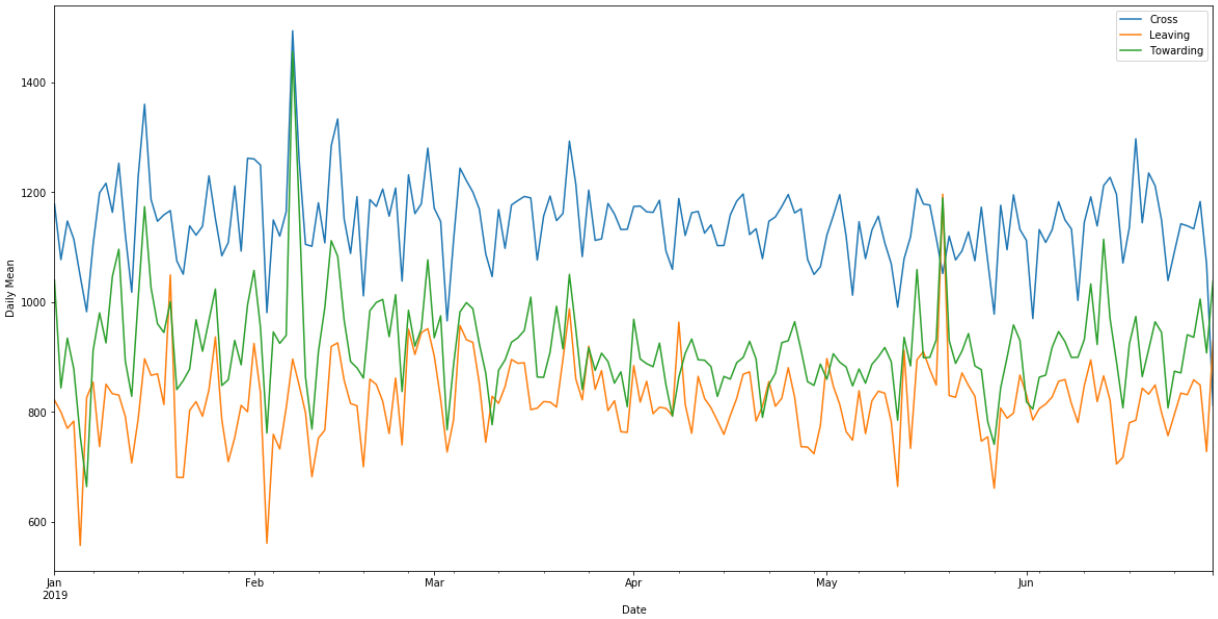
2. Travel time between different combination of origin and destination do share a common pattern of variation through the week. We will extract and include this pattern in our model for forecasting
3. Observing the result of part 1, the time of the day also have a pattern. However, three division "Early Morning", "Evening", "Midday" is not enough for us. At first, we wanna use the data in hours_q1 and hours_q2 to put the hourly factor into our model. However, unfortunately we do not have enough time to do that
4. We will first build a model to forecast the average travel time of whole trips (all combination of origin and destination). Then we will also build 6 specific models for 6 trips between spefic BARTs and hotspots.

The pattern of the day of the week





The visualization of all dates


Cross
Leaving
Towarding
1400
1200
1000
800
600
Daily Mean
Jan
2019
Feb
Mar
Apr
May
Jun
Date

In [68]:
```
# now lets plot the data that for every week (by doing this, we divide the weekly
week_num = compare['Date'].apply(lambda x: x.isocalendar()[1]) # get the week nu
compare = compare.assign(**{"week_num": week_num})
compare_wk = compare.groupby(["coordinate", "week_num"])['Daily Mean Travel Time
compare_wk
```

Out[68]:

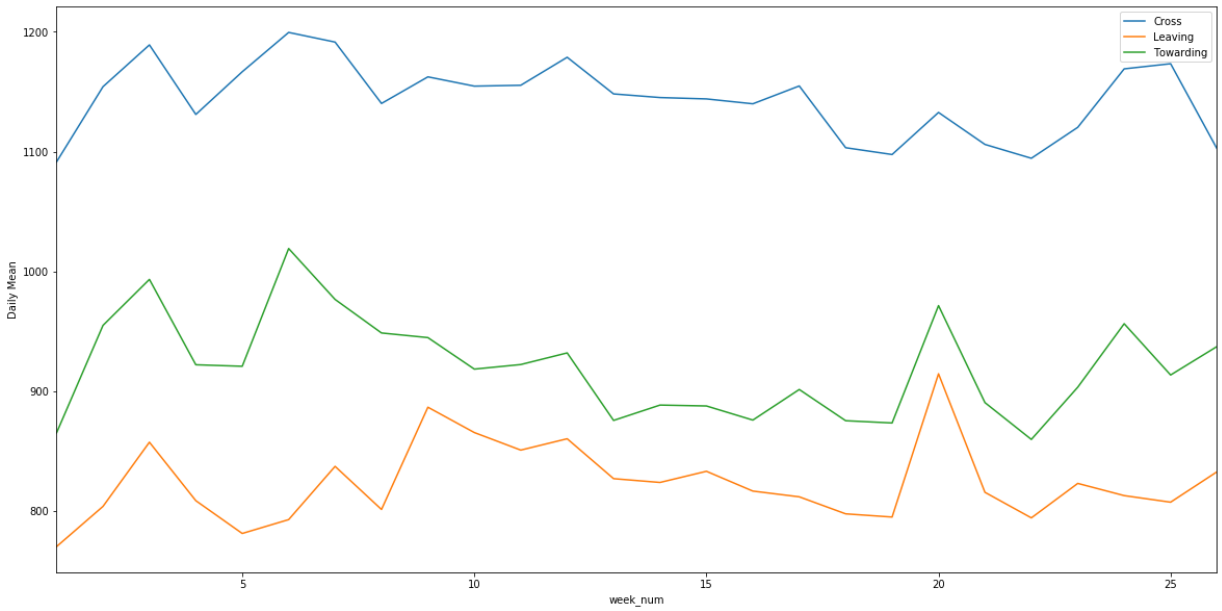
		Daily Mean Travel Time (Seconds)
coordinate	**week_num**	
Cross	**1**	1091.416667
	2	1154.178571
	3	1189.071429
	4	1131.000000
	5	1166.642857
	6	1199.500000
	7	1191.428571
	8	1140.178571
	9	1162.392857
	10	1154.642857
	11	1155.357143
	12	1178.750000
	13	1148.142857
	14	1145.107143
	15	1143.964286
	16	1139.892857
	17	1154.785714
	18	1103.214286
	19	1097.535714
	20	1132.678571
	21	1105.964286
	22	1094.392857
	23	1120.250000
	24	1169.071429
	25	1173.357143
	26	1102.346154
Leaving	**1**	770.100000
	2	803.673913
	3	857.276596
	4	808.446809

Daily Mean Travel Time (Seconds)

coordinate	week_num	
	...	...
	23	822.877551
	24	812.687500
	25	807.187500
	26	832.595745
Towarding	**1**	865.102564
	2	954.836735
	3	993.244898
	4	921.959184
	5	920.693878
	6	1019.021277
	7	976.510204
	8	948.510204
	9	944.632653
	10	918.326531
	11	922.142857
	12	931.857143
	13	875.469388
	14	888.291667
	15	887.541667
	16	875.755102
	17	901.306122
	18	875.229167
	19	873.346939
	20	971.367347
	21	890.354167
	22	859.595745
	23	903.224490
	24	956.229167
	25	913.408163
	26	937.208333

78 rows × 1 columns

In [69]:
```
# plot the data
tb_day_plot = compare_wk.pivot_table(columns = "coordinate", index = "week_num",
ax = tb_day_plot.plot(figsize=(20,10))
ax.set_ylabel('Daily Mean')
ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



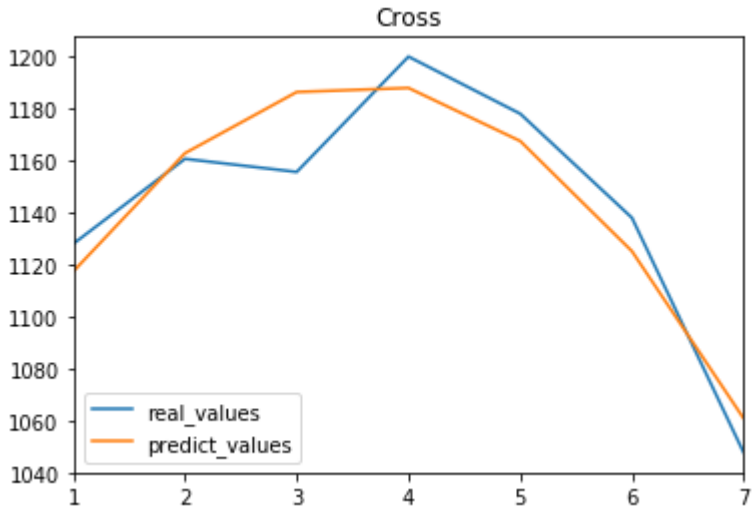
In [112]:
```
# first, get the quadratic regression for the day of the week patter.
# get the three set of data
temp_dct = compare_day_dir.to_dict()['Daily Mean Travel Time (Seconds)']
cross = []
leaving = []
towarding = []
for i in range(1, 8):
    cross.append((i, (temp_dct['Cross', i])))
    leaving.append((i, (temp_dct['Leaving', i])))
    towarding.append((i, (temp_dct['Towarding', i])))
cross = np.array(cross)
leaving = np.array(leaving)
toward = np.array(towarding)
```

In [120]:
```
compare_day_dir
```

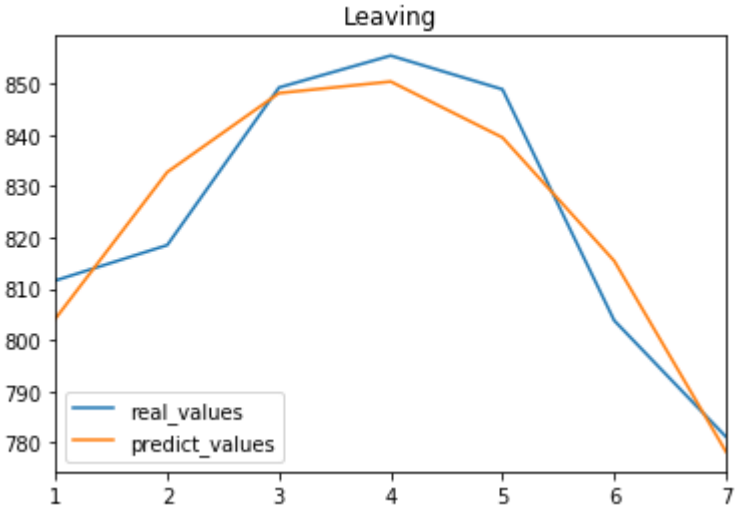
Out[120]:

		Daily Mean Travel Time (Seconds)
coordinate	**Day of the Week**	
Cross	**1**	1128.100000
	2	1160.769231
	3	1155.730769
	4	1200.048077
	5	1178.105769
	6	1138.105769
	7	1047.843137
Leaving	**1**	811.574850
	2	818.468208
	3	849.202247
	4	855.422222
	5	848.848315
	6	803.775862
	7	781.109195
Towarding	**1**	895.068966
	2	928.254144
	3	937.692308
	4	974.098901
	5	955.417582
	6	911.333333
	7	837.352601

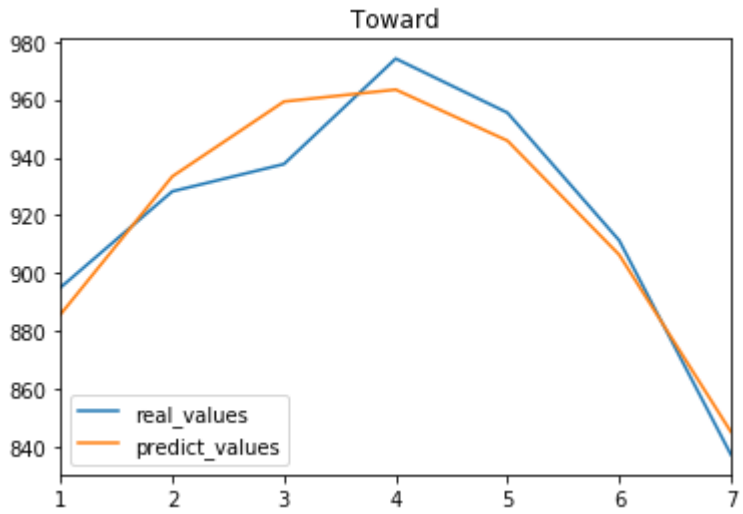
In [135]:
```
# run quadratic regression on each one
model = Pipeline([('poly', PolynomialFeatures(degree=2)),
                  ('linear', LinearRegression(fit_intercept=False))])
cross_x = np.array([x[0] for x in cross])
cross_y = np.array([y[1] for y in cross])
model.fit(cross_x[:, np.newaxis], cross_y)
predict_c = model.predict(cross_x[:, np.newaxis])
p_c = pd.DataFrame({'real_values' : cross_y, "predict_values" : predict_c}, inde
ax = p_c.plot()
ax.set_title("Cross")
plt.show()
cross_coef = model.named_steps['linear'].coef_
```



In [137]:
```
leaving_x = np.array([x[0] for x in leaving])
leaving_y = np.array([y[1] for y in leaving])
model.fit(leaving_x[:, np.newaxis], leaving_y)
predict_l = model.predict(leaving_x[:, np.newaxis])
p_l = pd.DataFrame({'real_values' : leaving_y, "predict_values" : predict_l}, in
ax = p_l.plot()
ax.set_title("Leaving")
plt.show()
leaving_coef = model.named_steps['linear'].coef_
```



In [139]:
```
toward_x = np.array([x[0] for x in toward])
toward_y = np.array([y[1] for y in toward])
model.fit(toward_x[:, np.newaxis], toward_y)
predict_t = model.predict(toward_x[:, np.newaxis])
p_t = pd.DataFrame({'real_values' : toward_y, "predict_values" : predict_t}, inde
ax = p_t.plot()
ax.set_title("Toward")
plt.show()
toward_coef = model.named_steps['linear'].coef_
```



In [116]:
```
display(cross_coef, leaving_coef, toward_coef)
display("y = c + bx + ax^2")
```

```
array([1050.06271816,   78.38955191,  -10.97602663])

array([762.44516335,  48.28519046,  -6.57643269])

array([816.40940464,  80.25170493, -10.87639811])

'y = c + bx + ax^2'
```

Now we want to find out the pattern between weeks. We first caculate the mean (baseline) and then we get the scaler of each week.

In [140]:
```
cross_mean = 0
leaving_mean = 0
towarding_mean = 0

#sum up weekly
for i in range(21):
    temp = compare_day_dir['Daily Mean Travel Time (Seconds)'][i]
    if i < 7:
        cross_mean += temp
    elif i < 14:
        leaving_mean += temp
    else:
        towarding_mean += temp

#taking the average
cross_mean = cross_mean/7
leaving_mean = leaving_mean/7
towarding_mean = towarding_mean/7

print('mean for cross: ',cross_mean, '\n', 'mean for leaving: ',leaving_mean, '\
```

```
mean for cross:  1144.1003932342169
 mean for leaving:  824.0572714118751
 mean for towarding 919.8882621231205
```

In [141]:
```
#scaling by the average
scaled_weekly = compare_wk.pivot_table(columns = "coordinate", index = "week_num"
scaled_weekly
```

Out[141]:

coordinate	Cross	Leaving	Towarding
week_num			
1	1091.416667	770.100000	865.102564
2	1154.178571	803.673913	954.836735
3	1189.071429	857.276596	993.244898
4	1131.000000	808.446809	921.959184
5	1166.642857	781.069767	920.693878
6	1199.500000	792.644444	1019.021277
7	1191.428571	837.108696	976.510204
8	1140.178571	801.106383	948.510204
9	1162.392857	886.510204	944.632653
10	1154.642857	865.340426	918.326531
11	1155.357143	850.687500	922.142857
12	1178.750000	860.166667	931.857143
13	1148.142857	826.836735	875.469388
14	1145.107143	823.673469	888.291667
15	1143.964286	833.000000	887.541667
16	1139.892857	816.469388	875.755102
17	1154.785714	811.673913	901.306122
18	1103.214286	797.458333	875.229167
19	1097.535714	794.787234	873.346939
20	1132.678571	914.416667	971.367347
21	1105.964286	815.448980	890.354167
22	1094.392857	794.086957	859.595745
23	1120.250000	822.877551	903.224490
24	1169.071429	812.687500	956.229167
25	1173.357143	807.187500	913.408163
26	1102.346154	832.595745	937.208333

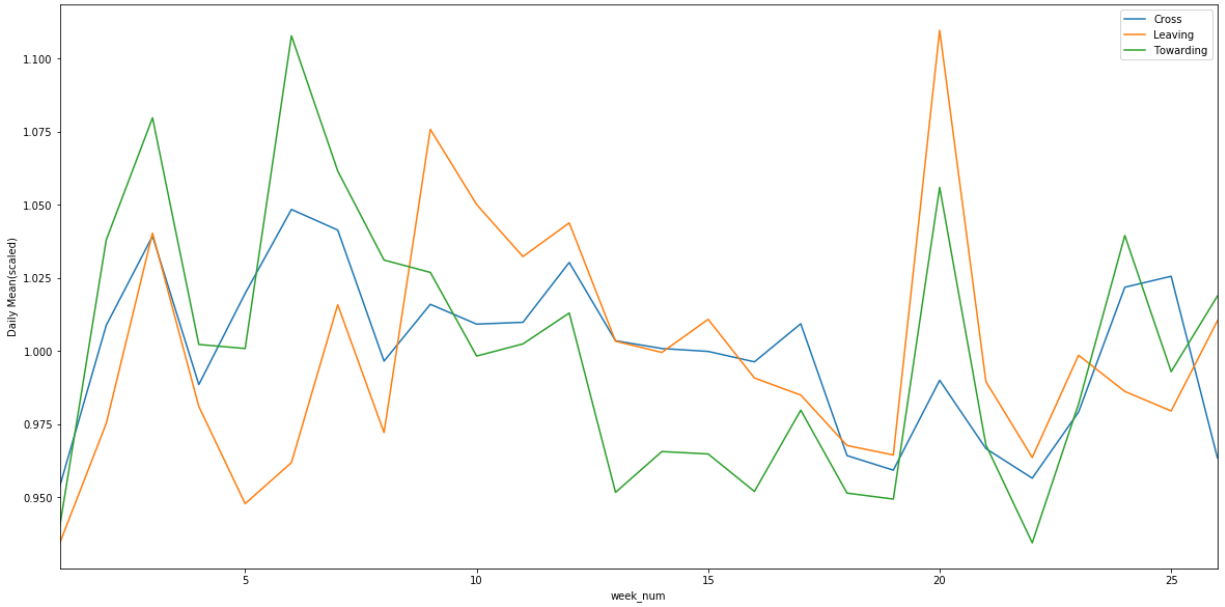
In [142]:
```
for i in range(1,len(scaled_weekly)+1):
    scaled_weekly['Cross'][i] = scaled_weekly['Cross'][i]/cross_mean
    scaled_weekly['Leaving'][i] = scaled_weekly['Leaving'][i]/leaving_mean
    scaled_weekly['Towarding'][i] = scaled_weekly['Towarding'][i]/towarding_mean
```

In [143]:
```
# now we have the scalers
scaled_weekly
```

Out[143]:

coordinate	Cross	Leaving	Towarding
week_num			
1	0.953952	0.934522	0.940443
2	1.008809	0.975265	1.037992
3	1.039307	1.040312	1.079745
4	0.988550	0.981057	1.002251
5	1.019703	0.947834	1.000876
6	1.048422	0.961880	1.107766
7	1.041367	1.015838	1.061553
8	0.996572	0.972149	1.031115
9	1.015989	1.075787	1.026899
10	1.009215	1.050097	0.998302
11	1.009839	1.032316	1.002451
12	1.030285	1.043819	1.013011
13	1.003533	1.003373	0.951713
14	1.000880	0.999534	0.965652
15	0.999881	1.010852	0.964836
16	0.996322	0.990792	0.952023
17	1.009339	0.984973	0.979800
18	0.964264	0.967722	0.951452
19	0.959300	0.964481	0.949405
20	0.990017	1.109652	1.055962
21	0.966667	0.989554	0.967894
22	0.956553	0.963631	0.934457
23	0.979154	0.998568	0.981885
24	1.021826	0.986203	1.039506
25	1.025572	0.979528	0.992956
26	0.963505	1.010362	1.018828

In [145]:
```
# the scaler coeficient
scaled_ax = scaled_weekly.plot(figsize=(20,10))
scaled_ax.set_ylabel('Daily Mean(scaled)')
scaled_ax.legend(bbox_to_anchor=(1,1))
plt.show()
```



In [150]:
```
# first get two average for two quarters
display((1,hours_q1['mean_travel_time'].mean())) # the first quarter
display((2,hours_q2['mean_travel_time'].mean())) # the second quarter
```

```
(1, 767.6730598320364)

(2, 757.1955729311454)
```

In [154]:
```
# only two points, this is fixed line
slope = -10.477749
intercept = 778.1505
avg_q3 = slope*3 + intercept
avg_q4 = slope*4 + intercept
avg_both = (avg_q3 + avg_q4)/2
display(avg_q3, avg_q4, avg_both)
```

```
746.7172529999999

736.239504

741.4783785
```

In [156]:
```
# now we have the average of next two quarter, we want to know next 26 weeks, eve
nxt_26 = scaled_weekly * avg_both
nxt_26
```

Out[156]:

coordinate	Cross	Leaving	Towarding
week_num			
1	707.334658	692.928173	697.318221
2	748.009931	723.137639	769.648688
3	770.623592	771.368790	800.607690
4	732.988163	727.432242	743.147650
5	756.087892	702.798658	742.127747
6	777.382230	713.213435	821.384808
7	772.151229	753.221918	787.118645
8	738.936691	720.827402	764.549171
9	753.333515	797.672894	761.423661
10	748.310829	778.624542	740.219541
11	748.773749	765.439988	743.295701
12	763.934392	773.969246	751.125927
13	744.098253	743.979311	705.674427
14	742.130841	741.133037	716.009859
15	741.390169	749.524955	715.405319
16	738.751522	734.650878	705.904728
17	748.403413	730.335958	726.500195
18	714.980560	717.544924	705.480796
19	711.300343	715.141496	703.963621
20	734.076026	822.782846	782.973232
21	716.762803	733.732725	717.672342
22	709.263493	714.511393	692.879435
23	726.021211	740.416878	728.046501
24	757.661821	731.247973	770.771061
25	760.439343	726.299129	736.255078
26	714.417934	749.161210	755.439268

In [ ]:
```

```