Taxi demand prediction in New York City

Business/Real World Problem

For a given location in New York City, our goal is to **predict the number of pickups in that given location**. Some location require more taxis at a particular time than other locations owing to the presence schools, hospitals, offices etc. The prediction result can be transferred to the taxi drivers via Smartphone app, and they can subsequently move to the locations where predicted pickups are high.

Objectives & Constraints

Objectives: Our objective is to predict the number of pickups as accurately as possible for each region in a 10min interval. We will break up the whole New York City into regions. Now, the 10min interval is chosen because in NYC one can commute 1 mile in approximately 10 minutes given the traffic is normal at that particular time.

Constraints:

- **Latency:** Given a location and current time of a taxi driver, as a taxi driver, he/she excepts to get the predicted pickups in his/her region and the adjoining regions in few seconds. Hence, there is a medium latency requirement.

- **Interpretability:** As long as taxi driver gets good prediction result, he/she is not be much interested in the interpretability of the result. He/she is not much interested in why he/she is getting this result. Hence, there is a no interpretability required.

- **Relative Errors:** Mean Absolute Percentage Error will be the relative error we will consider. Let say the predicted pickups for a particular location are 100, but actual

pickups are 102, the percentage error will be 2% and Absolute error is 2. The taxi driver will be more interested in the percentage error than the absolute error. Let say in some region the predicted pickups are 250, and if taxi driver knows that the relative error is 10% then he/she will consider the predicted result to be in the range of 225 to 275, which is considerable.

Our goal is to reduce the percentage error is low as possible.

1. Data Information

Source of Data: Data can be downloaded from here: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.
Here, we have used Jan- 2015 and Jan- 2016, Feb- 2016, Mar 2016 data.

Data Collection

We have collected all yellow taxi trips data of Jan-2015 and Jan-2016(Will be using only Jan 2015 data)

Information on Taxis

Information on taxis:

- **Yellow Taxi:** Yellow Medallion Taxicabs
  These are the famous NYC yellow taxis that provide transportation exclusively through street-hails. The number of taxicabs is limited by a finite number of medallions issued by the TLC. You access this mode of transportation by standing in the street and hailing an available taxi with your hand. The pickups are not pre-arranged.

- **For Hire Vehicles (FHVs)**
  FHV transportation is accessed by a pre-arrangement with a dispatcher or limo company. These FHVs are not permitted to pick up passengers via street hails, as those rides are not considered pre-arranged.

- **Green Taxi: Street Hail Livery (SHL)**
  The SHL program will allow livery vehicle owners to license and outfit their vehicles with green borough taxi branding, meters, credit card machines, and ultimately the right to accept street hails in addition to pre-arranged rides.

  In this project we are considering only the yellow taxis for the time period of Jan 2015. We have used Jan- 2015 data to make prediction for Jan- 2016 data.

Case study objective:

- Task 1: Incorporate Fourier features as features into Regression models and measure MAPE.
- Task 2: Perform hyper-parameter tuning for Regression models.

```
2a. Linear Regression: Grid Search
2b. Random Forest: Random Search
2c. Xgboost: Random Search
```

- Task 3: Explore more time-series features using Google search/Quora/Stackoverflow to reduce the MAPE to < 12%

Procedure for accomplished above objectives is as :

Task 1:

As the datsets is in time series ,convert into frequecy domain to get frequecy and amplitude values using fourier transform.And these Frequecy and amplitude of region_cum feature can be used as features.These features are important becuase of repeating structure of datasets.

- 1.Find the frequecy according to maximum amplitude
- 2.sort the frequecy to get important frequecy and corresponding amplitude.This is our new feature.
- 3.Concatenate newfeature with datasets .(this datsets is used for training and testing the model to get Train MAPE & Test MAPE)

Task2:

Perform hyper-parameter tuning for Regression models.

- 2a. Linear Regression: Grid Search
- 2b. Random Forest: Random Search
- 2c. Xgboost: Random Search

Task 3:

- Use time series feature in task1
- Split the Train & test dataset into 75-25%
- Use cluster size=30
- Here, I tried to use LSTM model and got MAPE <12%.

Refrence applied ai

Haversine distance

It calculates the great circle distance between two points on a sphere given their longitude and latitude[https://en.wikipedia.org/wiki/Haversine_formula]

To use this distance gpxpy.geo should be install in system.In google colab ,it can be directly fetch from gitclone as shown in below code.

In [165]:

```
Requirement already satisfied: gpxpy in /usr/local/lib/
python3.6/dist-packages (1.3.5)
```

In [0]:

In [0]:

In [168]:

```
Drive already mounted at /content/drive; to attempt to
forcibly remount, call drive.mount("/content/drive", fo
rce_remount=True).
```

In [169]:

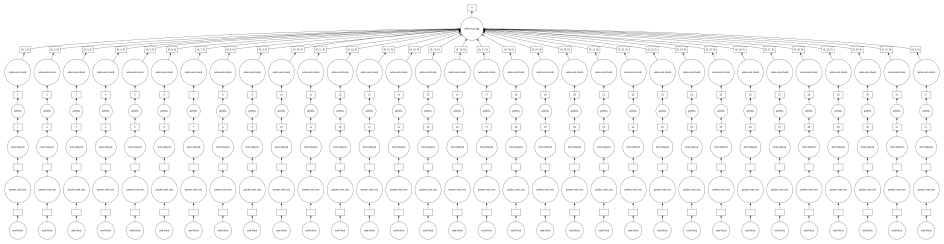
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropof
f_datetime',
       'passenger_count', 'trip_distance', 'pickup_long
itude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_
flag',
       'dropoff_longitude', 'dropoff_latitude', 'paymen
t_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amoun
t',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [170]:

Out[170]:

In [171]:

Out[171]:



ML Problem Formulation

Time-series forecasting and Regression

To find number of pickups, given location cordinates(latitude and longitude) and time, in the query reigion and surrounding regions.

To solve the above we would be using data collected in Jan - Mar 2015 to predict the pickups in Jan - Mar 2016.

Performance metrics

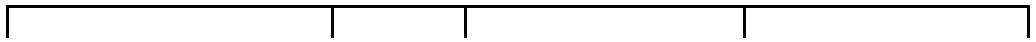
1. Mean Absolute percentage error.
2. Mean Squared error.

Data Cleaning

In this section we will be doing univariate analysis and removing outlier/illegitimate values which may be caused due to some error

In [172]:

Out[172]:



File-Name	File-Size	Number of Records	Number of Features
yellow_tripdata_2015-01	1.84GB	12748986	19
yellow_tripdata_2016-01	1.59GB	10906858	19

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distan
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00

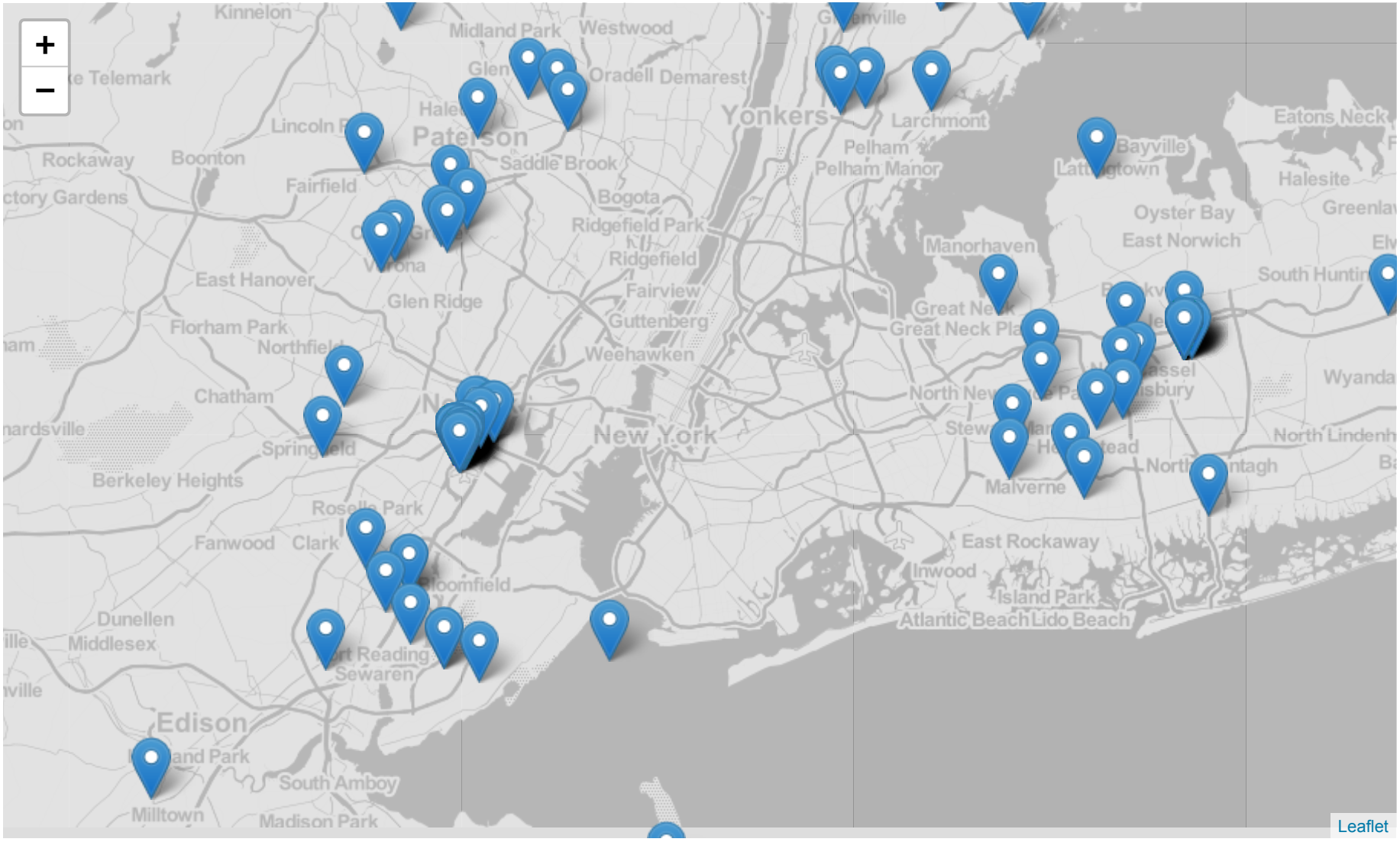
1. Pickup Latitude and Pickup Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bounded by the location cordinates(lat,long) - (40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are not considered by us as we are only concerned with pickups which originate within New York.

In [173]:
```
# Plotting pickup coordinates which are outside the bounding box of NYC
# All the points outside the NYC
outlier_locations=month[((month.pickup_longitude <= -74.15)|(month.pickup_latitude <= 40.5774)|
                         (month.pickup_longitude >= -73.7004)| (month.pickup_latitude >= 40.9176))]
# Map with base location
map_osm = folium.Map(location=[40.734695,-73.990372],tiles='Stamen Toner')
# To plot 100 outliers on map
sample_locations=outlier_locations.head(10000)
#print(sample_locations.tail(3))
for i,j in sample_locations.iterrows():
    if int(j['pickup_latitude']) !=0:
        folium.Marker(list((j['pickup_latitude'],j['pickup_longitude']))).add_to(map_osm)
```

```
map_osm
```

Out[173]:



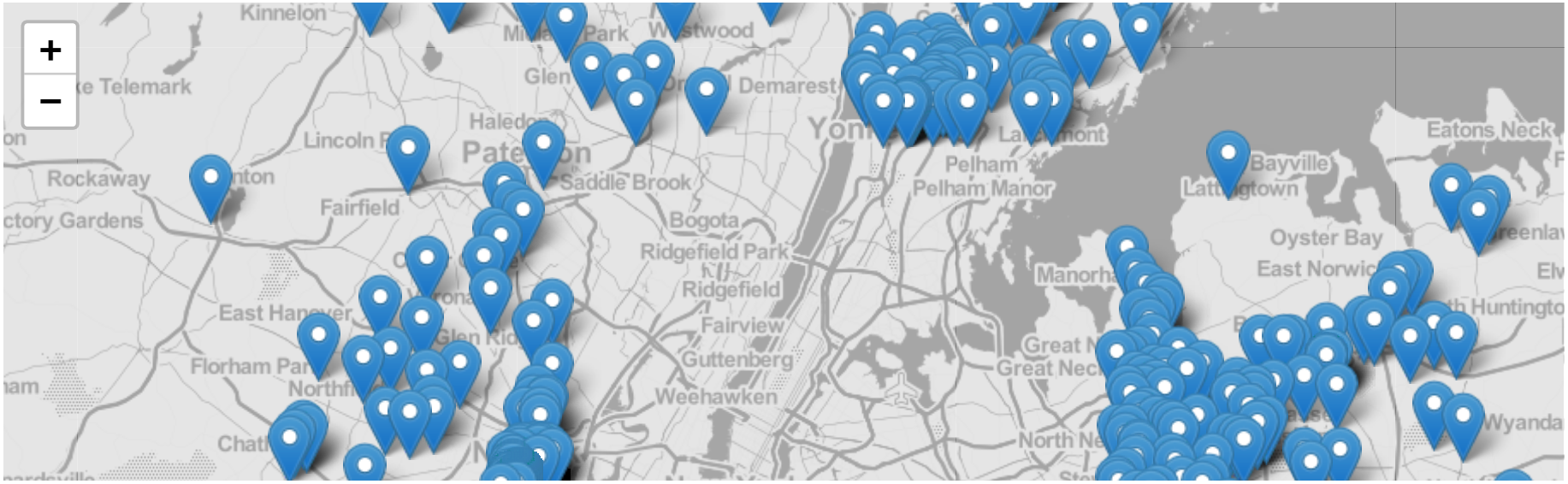
Observation:- As you can see above that there are some points just outside the boundary but there are a few that are in either South america, Mexico or Canada

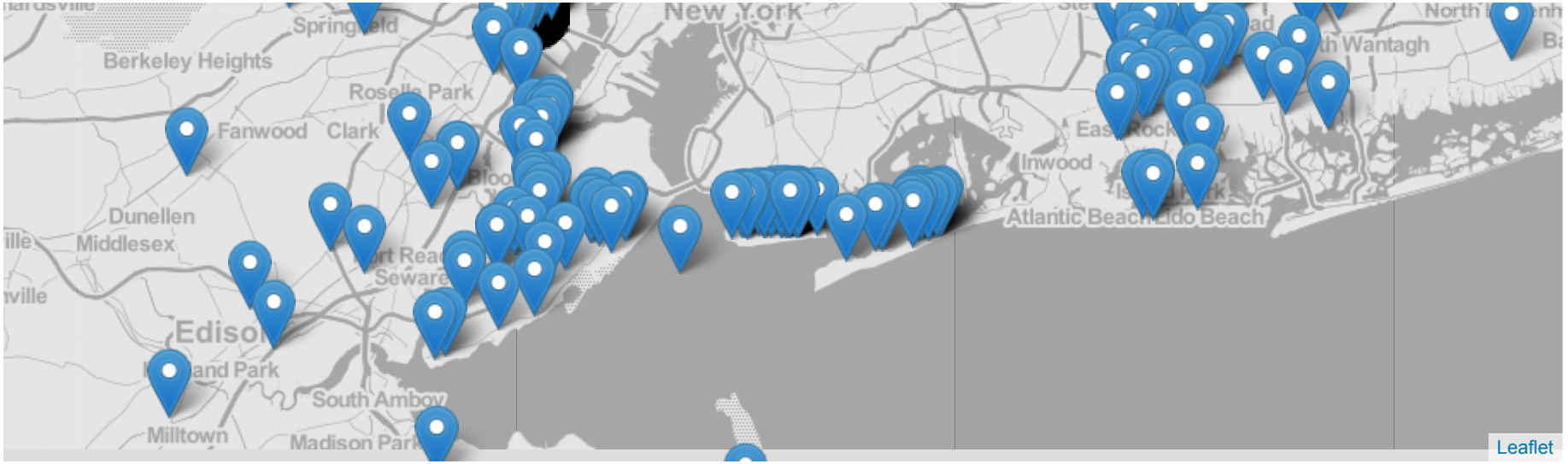
2. Dropoff Latitude & Dropoff Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bounded by the location coordinates(lat,long) - (40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are not considered by us as we are only concerned with dropoffs which are within New York.

In [174]:
```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locations
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitude <= 40.5774)| \
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.9176))]

# creating a map with the a base location
# read more about the folium here: http://folium.readthedocs.io/en/latest/quickstart.html

# note: you dont need to remember any of these, you dont need indeepth knowledge on these maps and plots

map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')

# we will spot only first 100 outliers on the map, plotting all the outliers will take more time
sample_locations = outlier_locations.head(10000)
for i,j in sample_locations.iterrows():
    if int(j['pickup_latitude']) != 0:
        folium.Marker(list((j['dropoff_latitude'],j['dropoff_longitude']))).add_to(map_osm)
map_osm
```

Out[174]:





Observation:- The observations here are similar to those obtained while analysing pickup latitude and longitude

3. Trip Durations:

According to NYC Taxi & Limousine Commision Regulations the maximum allowed trip duration in a 24 hour interval is 12 hours.

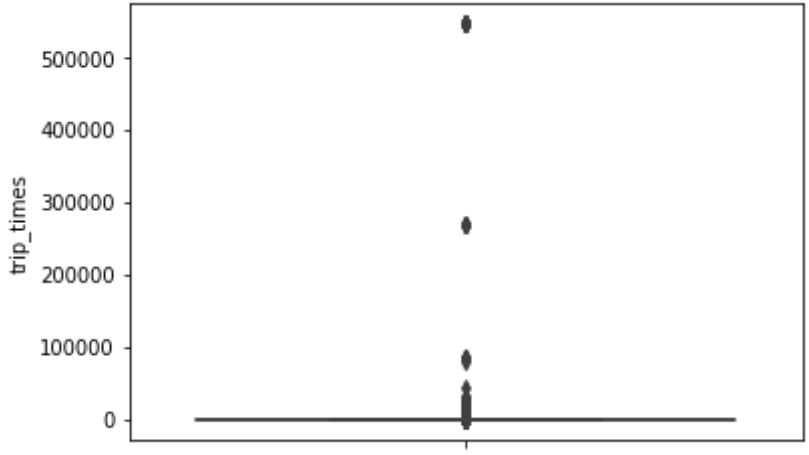
In [0]:
```
# Timestamp are converted to unix so as to get trip_time & speed
# https://stackoverflow.com/a/27914405
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s,
                                                  "%Y-%m-%d %H:%M:%S").timetuple())
def return_with_trip_times(month):
    duration = month[['tpep_pickup_datetime','tpep_dropoff_datetime']].compute()
    #pickups and dropoffs to unix time
    duration_pickup = [convert_to_unix(x) for x in duration['tpep_pickup_datetime'].values]
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'].values]
    #calculate duration of trips
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)

    #append durations of trips and speed in miles/hr to a new dataframe
    new_frame = month[['passenger_count','trip_distance',
                       'pickup_longitude','pickup_latitude',
```

```
                      'dropoff_longitude','dropoff_latitude',
                      'total_amount']].compute()

    new_frame['trip_times'] = durations
    new_frame['pickup_times'] = duration_pickup
    new_frame['Speed'] = 60*(new_frame['trip_distance']/new_frame['trip_times'])

    return new_frame
```

In [176]:
```
%%time
frame_with_durations = return_with_trip_times(month)
```

```
CPU times: user 9min 17s, sys: 1min 57s, total: 11min 15s
Wall time: 10min 38s
```

In [177]:
```
# the skewed box plot shows us the presence of outliers
%matplotlib inline

sns.boxplot(y="trip_times", data =frame_with_durations)
plt.show();
```



In [178]:
```
for i in range(0,100,10):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var ,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

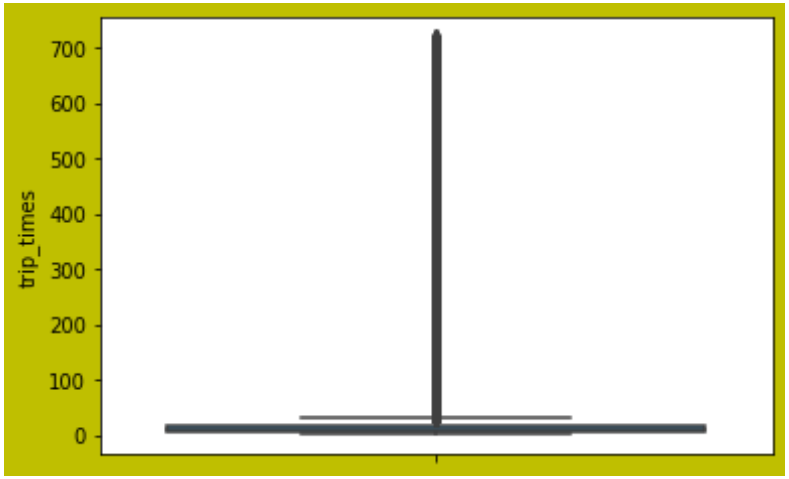
```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333
80 percentile value is 17.633333333333333
90 percentile value is 23.45
100 percentile value is  548555.6333333333
```

In [179]:
```
#looking further from the 99th percecntile
for i in range(90,100):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98 percentile value is 38.71666666666667
99 percentile value is 46.75
100 percentile value is  548555.6333333333
```

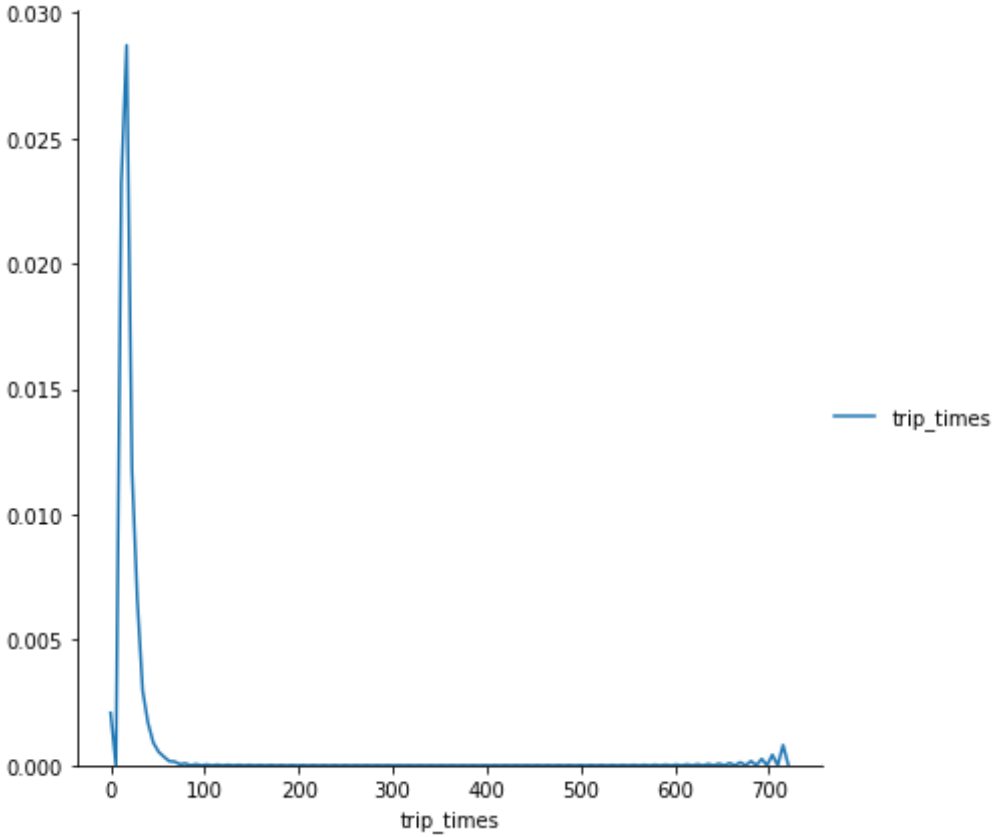
In [0]:
```
#removing data based on our analysis and TLC regulations
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_times>1) & (frame_with_durations.trip
_times<720)]
```

In [181]:
```
#box-plot after removal of outliers
fig3 = plt.figure( facecolor='y', edgecolor='k')
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.show()
```



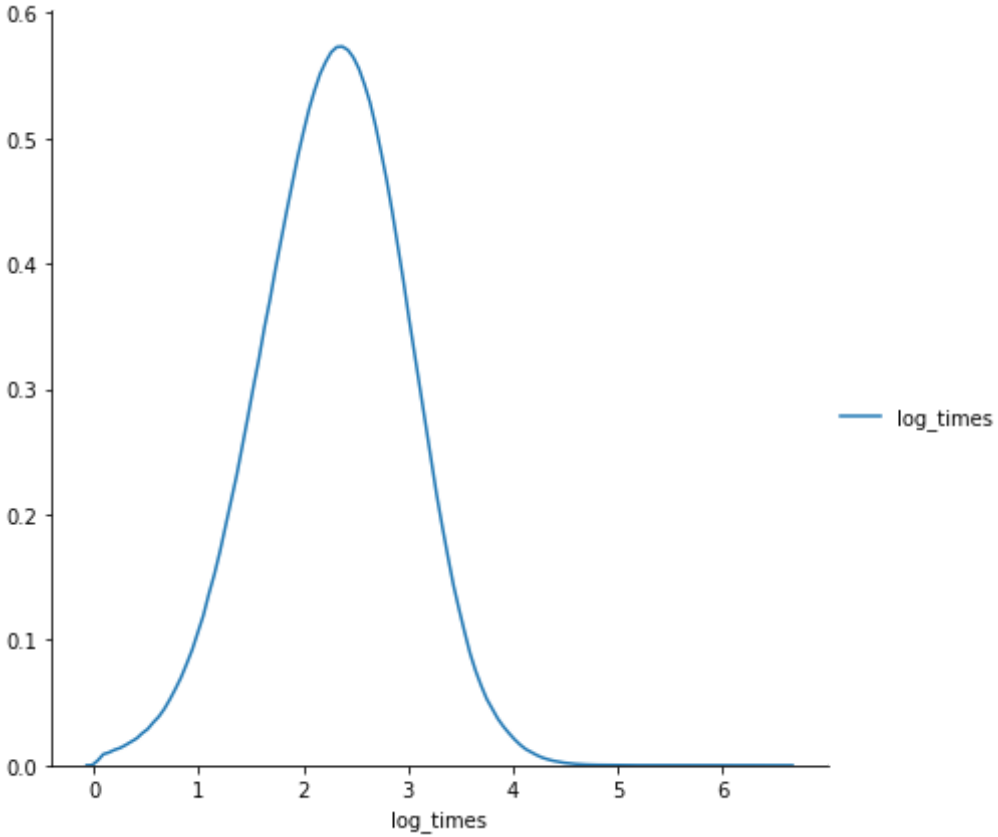
In [182]:
```
#pdf of trip-times after removing the outliers
fig3 = plt.figure( facecolor='y', edgecolor='k')
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend()
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

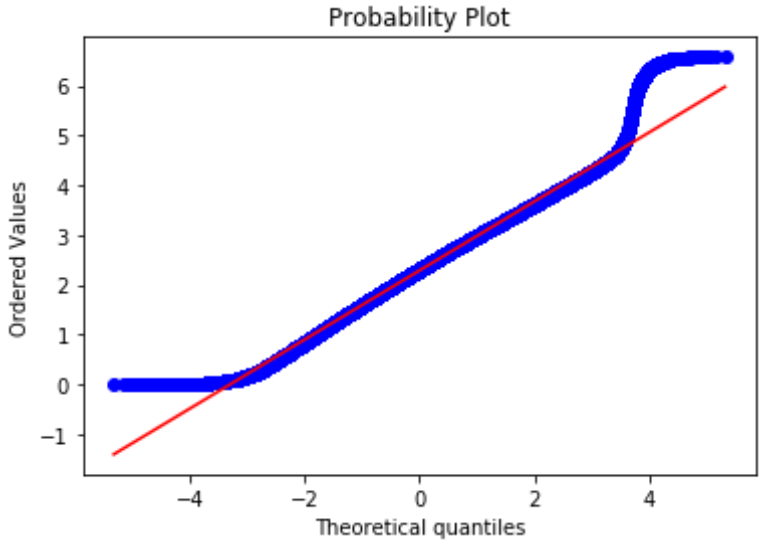


In [0]:
```
#converting the values to log-values to chec for log-normal
import math
frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_durations_modified['trip_times'].values
]
```

In [184]:
```
#pdf of log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```

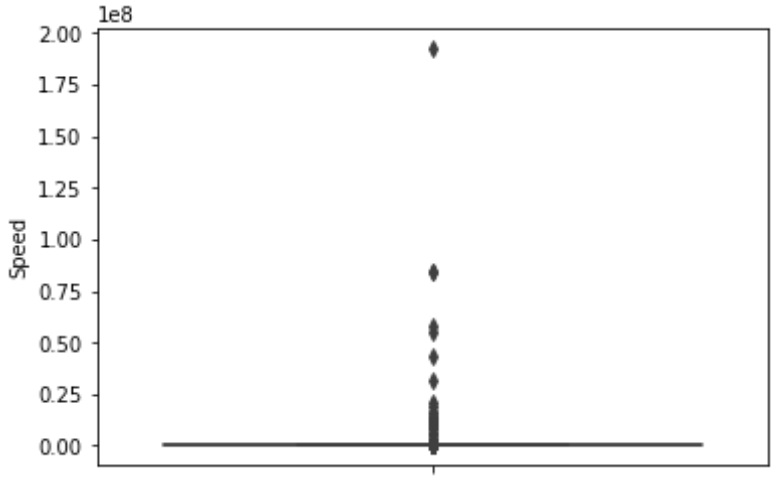


In [185]:
```
import scipy
#Q-Q plot for checking if trip-times is log-normal
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [186]:
```
# check for any outliers in the data after trip duration outliers removed
# box-plot for speeds with outliers
frame_with_durations_modified['Speed'] = 60*(frame_with_durations_modified['trip_distance']/frame_with_durations_mo
dified['trip_times'])
sns.boxplot(y="Speed", data =frame_with_durations_modified)
plt.show()
```



In [187]:
```
#calculating speed values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is 0.0
10 percentile value is 6.409495548961425
20 percentile value is 7.80952380952381
30 percentile value is 8.929133858267717
40 percentile value is 9.98019801980198
50 percentile value is 11.06865671641791
60 percentile value is 12.286689419795222
70 percentile value is 13.796407185628745
80 percentile value is 15.963224893917962
90 percentile value is 20.186915887850468
100 percentile value is  192857142.85714284
```

In [188]:
```
#calculating speed values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
90 percentile value is 20.186915887850468
```

```
91 percentile value is 20.91645569620253
92 percentile value is 21.752988047808763
93 percentile value is 22.721893491124263
94 percentile value is 23.844155844155843
95 percentile value is 25.182552504038775
96 percentile value is 26.80851063829787
97 percentile value is 28.84304932735426
98 percentile value is 31.591128254580514
99 percentile value is 35.7513566847558
100 percentile value is  192857142.85714284
```

In [189]:
```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 35.7513566847558
99.1 percentile value is 36.31084727468969
99.2 percentile value is 36.91470054446461
99.3 percentile value is 37.588235294117645
99.4 percentile value is 38.33035714285714
99.5 percentile value is 39.17580340264651
99.6 percentile value is 40.15384615384615
99.7 percentile value is 41.338301043219076
99.8 percentile value is 42.86631016042781
99.9 percentile value is 45.3107822410148
100 percentile value is  192857142.85714284
```

In [0]:
```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0) & (frame_with_durations.Speed<45.
31)]
```

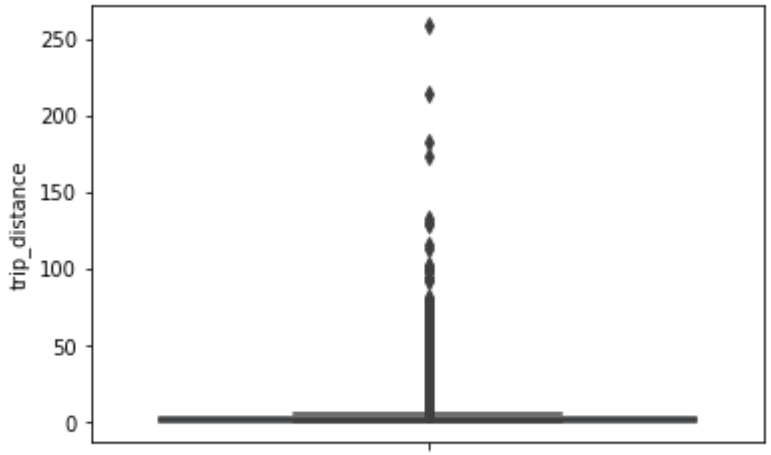
In [191]:
```
#avg.speed of cabs in New-York
sum(frame_with_durations_modified["Speed"]) / float(len(frame_with_durations_modified["Speed"]))
```

Out[191]:
```
12.450173996027528
```

The avg speed in Newyork speed is 12.45miles/hr, so a cab driver can travel2 miles per 10min on avg.

5. Trip Distance

In [192]:
```
# up to now we have removed the outliers based on trip durations and cab speeds
# lets try if there are any outliers in trip distances
# box-plot showing outliers in trip-distance values
sns.boxplot(y="trip_distance", data =frame_with_durations_modified)
plt.show()
```



In [193]:
```
#calculating trip distance values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is 0.01
10 percentile value is 0.66
20 percentile value is 0.9
30 percentile value is 1.1
40 percentile value is 1.39
50 percentile value is 1.69
60 percentile value is 2.07
70 percentile value is 2.6
80 percentile value is 3.6
90 percentile value is 5.97
100 percentile value is  258.9
```

In [194]:
```
#calculating trip distance values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
90 percentile value is 5.97
91 percentile value is 6.45
92 percentile value is 7.07
93 percentile value is 7.85
94 percentile value is 8.72
95 percentile value is 9.6
96 percentile value is 10.6
97 percentile value is 12.1
98 percentile value is 16.03
99 percentile value is 18.17
100 percentile value is  258.9
```

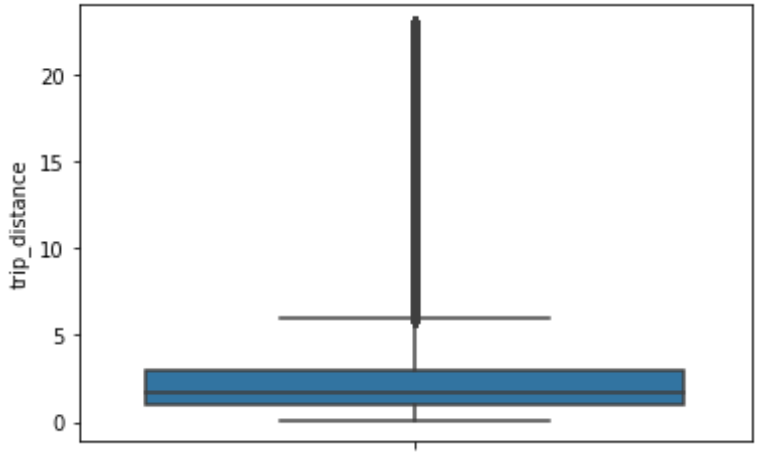
In [195]:
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 18.17
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.83
99.4 percentile value is 19.13
99.5 percentile value is 19.5
99.6 percentile value is 19.96
99.7 percentile value is 20.5
99.8 percentile value is 21.22
99.9 percentile value is 22.57
100 percentile value is  258.9
```

In [0]:
```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_distance>0) & (frame_with_durations.trip_distance<23)]
```

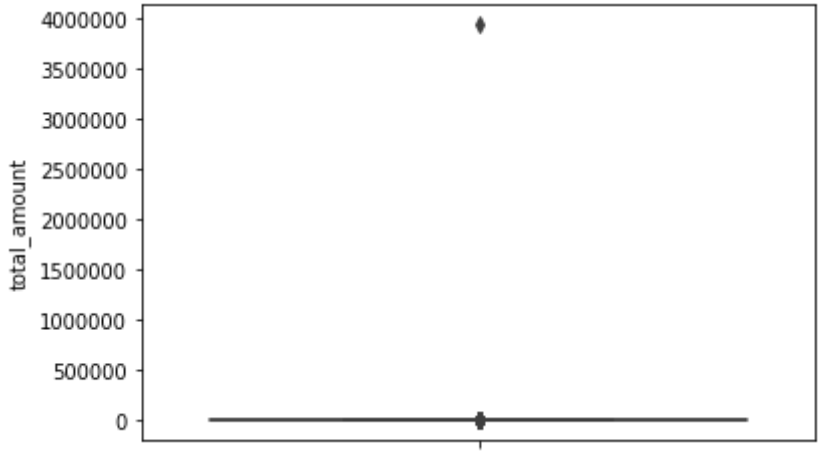
In [197]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
```

```
plt.show()
```



6. Total Fare

In [198]:
```
# lets try if there are any outliers in based on the total_amount
# box-plot showing outliers in fare
sns.boxplot(y="total_amount", data =frame_with_durations_modified)
plt.show()
```



In [199]:
```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is -242.55
10 percentile value is 6.3
20 percentile value is 7.8
30 percentile value is 8.8
40 percentile value is 9.8
50 percentile value is 11.16
60 percentile value is 12.8
70 percentile value is 14.8
80 percentile value is 18.3
90 percentile value is 25.8
100 percentile value is  3950611.6
```

In [200]:
```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

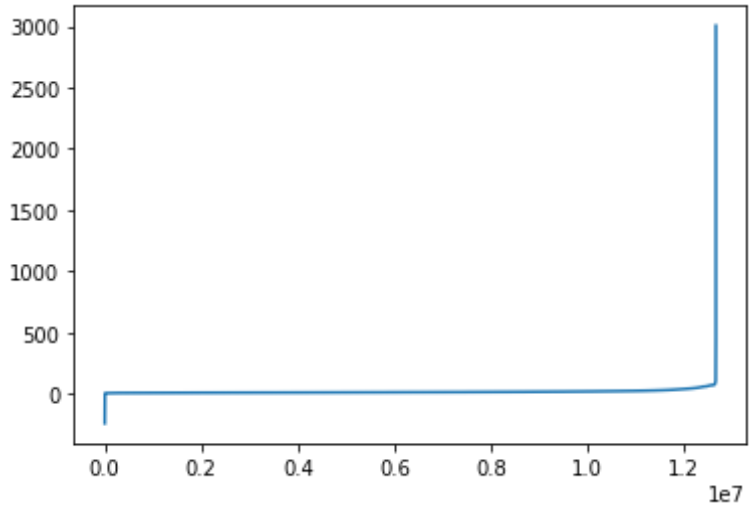
```
90 percentile value is 25.8
91 percentile value is 27.3
92 percentile value is 29.3
93 percentile value is 31.8
94 percentile value is 34.8
95 percentile value is 38.53
96 percentile value is 42.6
97 percentile value is 48.13
98 percentile value is 58.13
99 percentile value is 66.13
100 percentile value is  3950611.6
```

In [201]:
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

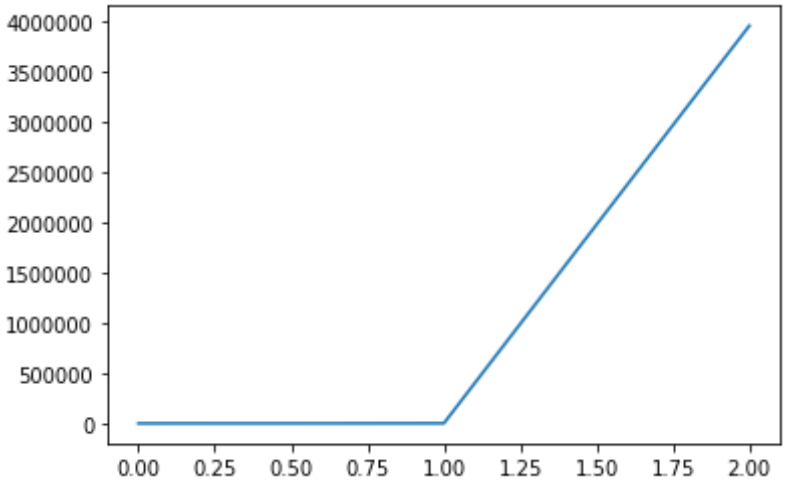
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value is 88.28
100 percentile value is  3950611.6
```

Observation:- Since even the 99.9th percentile does not seem like an outlier, since there is not much variation between the 99.8th percentile and the 99.9th percentile, we are moving on to graphical analysis.

In [202]:
```
#below plot shows us the fare values(sorted) to find a sharp increase to remove those values as outliers
# plot the fare amount excluding last two values in sorted data
plt.plot(var[:-2])
plt.show()
```



In [203]:
```
# a very sharp increase in fare values can be seen
# plotting last three total fare values, and we can observe there is share increase in the values
plt.plot(var[-3:])
plt.show()
```



Remove all outliers/erronous points.

In [0]:
```
import gpxpy.geo
#removing all outliers based on our univariate analysis above
def remove_outliers(new_frame):


    a = new_frame.shape[0]
    print ("Number of pickup records = ",a)
    temp_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_longitude <= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitude <= 40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitude >= 40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitude <= 40.9176))]
    b = temp_frame.shape[0]
    print ("Number of outlier coordinates lying outside NY boundaries:",(a-b))


    temp_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    c = temp_frame.shape[0]
    print ("Number of outliers from trip times analysis:",(a-c))


    temp_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 23)]
    d = temp_frame.shape[0]
    print ("Number of outliers from trip distance analysis:",(a-d))
```

```
    temp_frame = new_frame[(new_frame.Speed <= 65) & (new_frame.Speed >= 0)]
    e = temp_frame.shape[0]
    print ("Number of outliers from speed analysis:",(a-e))

    temp_frame = new_frame[(new_frame.total_amount <1000) & (new_frame.total_amount >0)]
    f = temp_frame.shape[0]
    print ("Number of outliers from fare analysis:",(a-f))


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_longitude <= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitude <= 40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitude >= 40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitude <= 40.9176))]

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 23)]
    new_frame = new_frame[(new_frame.Speed < 45.31) & (new_frame.Speed > 0)]
    new_frame = new_frame[(new_frame.total_amount <1000) & (new_frame.total_amount >0)]

    print ("Total outliers removed",a - new_frame.shape[0])
    print ("---")
    return new_frame
```

In [205]:

```
print ("Removing outliers in the month of Jan-2015")
print ("----")
frame_with_durations_outliers_removed = remove_outliers(frame_with_durations)
print("fraction of data points that remain after removing outliers",
      float(len(frame_with_durations_outliers_removed))/len(frame_with_durations))
```

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12748986
Number of outlier coordinates lying outside NY boundaries: 293919
Number of outliers from trip times analysis: 23889
Number of outliers from trip distance analysis: 92597
Number of outliers from speed analysis: 24473
Number of outliers from fare analysis: 5275
Total outliers removed 377910
---
fraction of data points that remain after removing outliers 0.9703576425607495
```

Data-preperation

Clustering/Segmentation

In [206]:
```
import gpxpy.geo
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude',
                                                'pickup_longitude']].values
neighbours=[]

def find_min_distance(cluster_centers, cluster_len):
    nice_points = 0
    wrong_points = 0
    less2 = []
    more2 = []
    min_dist=1000
    for i in range(0, cluster_len):
        nice_points = 0
        wrong_points = 0
        for j in range(0, cluster_len):
            if j!=i:
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0],
                                                        cluster_centers[i][1],
                                                        cluster_centers[j][0],
                                                        cluster_centers[j][1])
                min_dist = min(min_dist,distance/(1.60934*1000))
                if (distance/(1.60934*1000)) <= 2:
                    nice_points +=1
                else:
                    wrong_points += 1
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters within the vicinity (i.e. inte
rcluster-distance < 2):", np.ceil(sum(less2)/len(less2)), "\nAvg. Number of Clusters outside the vicinity (i.e. i
ntercluster-distance > 2):", np.ceil(sum(more2)/len(more2)),"\nMin inter-cluster distance = ",min_dist,"\n---")

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fit(coords)
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_remove
d[['pickup_latitude', 'pickup_longitude']])
    cluster_centers = kmeans.cluster_centers_
```

```
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

# we need to choose number of clusters so that, there are more number of cluster regions
#that are close to any cluster center
# and make sure that the minimum inter cluster should not be very less
for increment in range(10, 100, 10):
    cluster_centers, cluster_len = find_clusters(increment)
    find_min_distance(cluster_centers, cluster_len)
```

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 8.0
Min inter-cluster distance =  1.0945442325142543
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 16.0
Min inter-cluster distance =  0.7131298007387813
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 22.0
Min inter-cluster distance =  0.5185088176172206
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 32.0
Min inter-cluster distance =  0.5069768450363973
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 38.0
Min inter-cluster distance =  0.365363025983595
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 46.0
Min inter-cluster distance =  0.34704283494187155
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 54.0
Min inter-cluster distance =  0.30502203163244707
---
```

```
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 62.0
Min inter-cluster distance =  0.29220324531738534
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 69.0
Min inter-cluster distance =  0.18257992857034985
---
```

Cluster size=40

Inference:

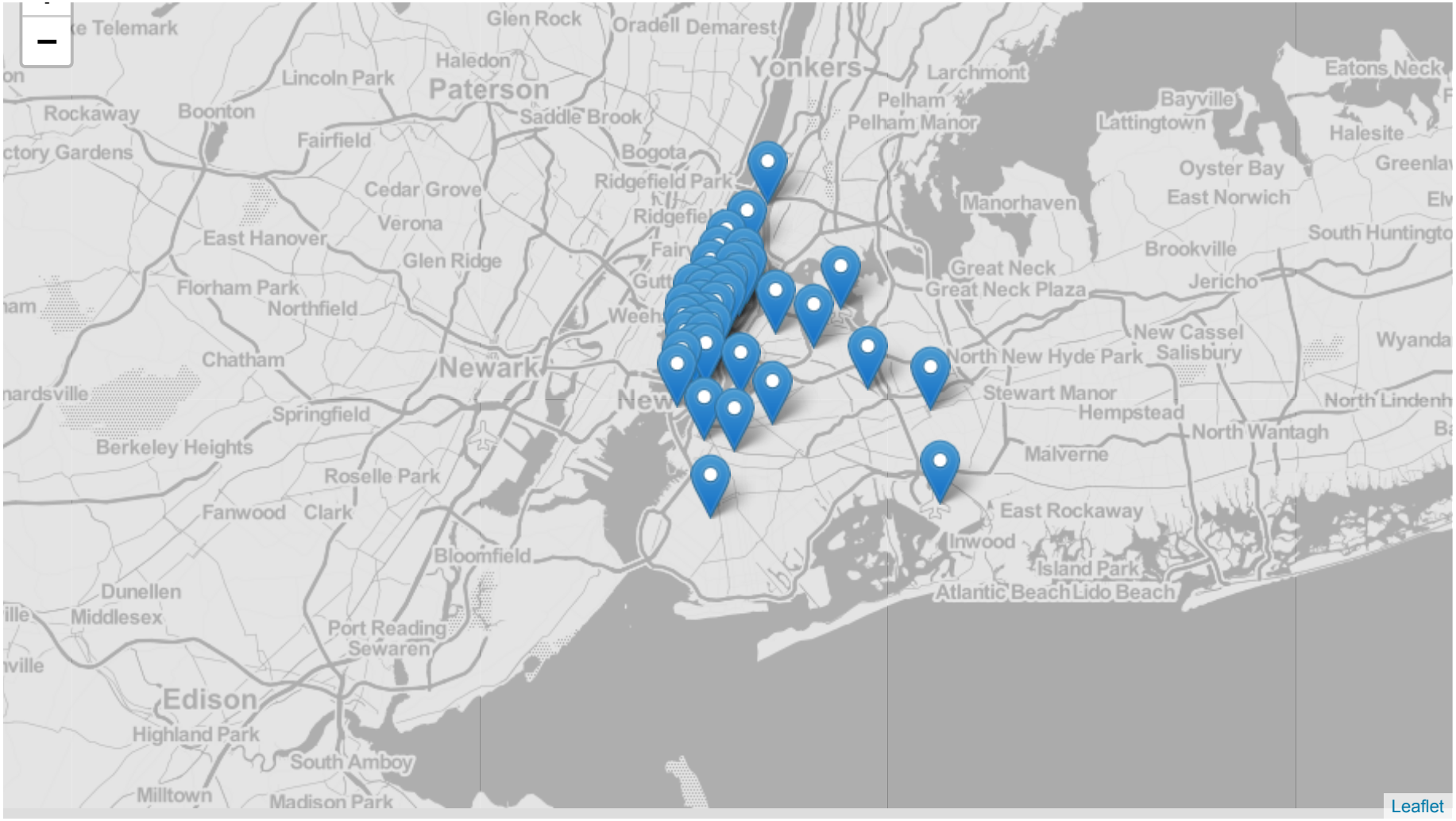
- The main objective was to find a optimal min. distance(Which roughly estimates to the radius of a cluster) between the clusters which we got was 40

In [0]:
```
# if check for the 50 clusters you can observe that there are two clusters with only 0.3 miles apart from each othe
r
# so we choose 30 clusters for solve the further problem

# Getting 30 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(coords)
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed[['pi
ckup_latitude', 'pickup_longitude']])
```

Plotting the cluster centers:

In [208]:
```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])), popup=(str(cluster_centers[i][0])+str(cluste
r_centers[i][1]))).add_to(map_osm)
map_osm
```

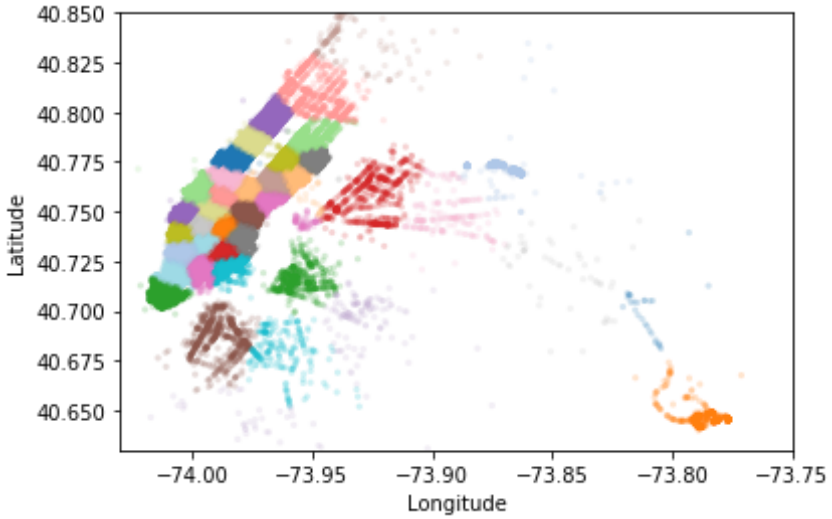
Out[208]:



Plotting the clusters:

In [209]:
```
#Visualising the clusters on a map
def plot_clusters(frame):
    city_long_border = (-74.03, -73.75)
    city_lat_border = (40.63, 40.85)
    fig, ax = plt.subplots(ncols=1, nrows=1)
```

```
    ax.scatter(frame.pickup_longitude.values[:100000], frame.pickup_latitude.values[:100000], s=10, lw=0,
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.2)
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [0]:
```
def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,1433116800],\
                  [1451606400,1454284800,1456790400,1459468800,1462060800,1464739200]]

    start_pickup_unix=unix_times[year-2015][month-1]
    # https://www.timeanddate.com/time/zones/est
    # (int((i-start_pickup_unix)/600)+33) : our unix time is in gmt to we are converting it to est
    tenminutewise_binned_unix_pickup_times=[(int((i-start_pickup_unix)/600)+33) for i in unix_pickup_times]
    frame['pickup_bins'] = np.array(tenminutewise_binned_unix_pickup_times)
    return frame
```

In [0]:
```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2015)
jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cluster','pickup_bins']).count()
```

In [212]:
```
# we add two more columns 'pickup_cluster'(to which cluster it belogns to)
# and 'pickup_bins' (to which 10min intravel the trip belongs to)
jan_2015_frame.head()
```

Out[212]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pickup_tim
0	1	1.59	-73.993896	40.750111	-73.974785	40.750618	17.05	18.050000	1.421349e+(
1	1	3.30	-74.001648	40.724243	-73.994415	40.759109	17.80	19.833333	1.420922e+(
2	1	1.80	-73.963341	40.802788	-73.951820	40.824413	10.80	10.050000	1.420922e+(
3	1	0.50	-74.009087	40.713818	-74.004326	40.719986	4.80	1.866667	1.420922e+(
4	1	3.00	-73.971176	40.762428	-74.004181	40.742653	16.30	19.316667	1.420922e+(

In [213]:
```
# hear the trip_distance represents the number of pickups that are happend in that particular 10min intravel
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels 24*31*60/10 =4464bins)
jan_2015_groupby.head()
```

Out[213]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**33**	104
	34	200
	35	208
	36	141
	37	155

In [214]:
```
# Data Preparation for the months of Jan,Feb and March 2016
def datapreparation(month,kmeans,month_no,year_no):

    print ("Return with trip times..")

    frame_with_durations = return_with_trip_times(month)

    print ("Remove outliers..")
    frame_with_durations_outliers_removed = remove_outliers(frame_with_durations)

    print ("Estimating clusters..")
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed
[['pickup_latitude', 'pickup_longitude']])
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_re
moved_2016[['pickup_latitude', 'pickup_longitude']])

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_no,year_no)
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cl
uster','pickup_bins']).count()

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-01.csv')
month_feb_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-02.csv')
month_mar_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-03.csv')

jan_2016_frame,jan_2016_groupby = datapreparation(month_jan_2016,kmeans,1,2016)
feb_2016_frame,feb_2016_groupby = datapreparation(month_feb_2016,kmeans,2,2016)
mar_2016_frame,mar_2016_groupby = datapreparation(month_mar_2016,kmeans,3,2016)
```

```
Return with trip times..
Remove outliers..
Number of pickup records =  10906858
Number of outlier coordinates lying outside NY boundaries: 214677
Number of outliers from trip times analysis: 27190
Number of outliers from trip distance analysis: 79742
Number of outliers from speed analysis: 21047
Number of outliers from fare analysis: 4991
Total outliers removed 297784
---
Estimating clusters..
Final groupbying..
Return with trip times..
```

```
Remove outliers..
Number of pickup records =  11382049
Number of outlier coordinates lying outside NY boundaries: 223161
Number of outliers from trip times analysis: 27670
Number of outliers from trip distance analysis: 81902
Number of outliers from speed analysis: 22437
Number of outliers from fare analysis: 5476
Total outliers removed 308177
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  12210952
Number of outlier coordinates lying outside NY boundaries: 232444
Number of outliers from trip times analysis: 30868
Number of outliers from trip distance analysis: 87318
Number of outliers from speed analysis: 23889
Number of outliers from fare analysis: 5859
Total outliers removed 324635
---
Estimating clusters..
Final groupbying..
```

Smoothing

In [0]:
```
# Gets the unique bins where pickup values are present for each each reigion

# for each cluster region we will collect all the indices of 10min intravels in which the pickups are happened
# we got an observation that there are some pickpbins that doesnt have any pickups
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,30):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [0]:
```
# for every month we get all indices of 10min intravels in which atleast one pickup got happened

#jan
```

```
jan_2015_unique = return_unq_pickup_bins(jan_2015_frame)
jan_2016_unique = return_unq_pickup_bins(jan_2016_frame)

#feb
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [217]:
```
# for each cluster number of 10min intravels with 0 pickups
for i in range(30):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",4464 - len(set(jan_2015_unique[i
]))))
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1985
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  354
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  153
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  117
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  26
```

```
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  58
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  1190
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1357
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  53
------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  163
------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  35
------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  214
------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
```

there are two ways to fill up these values

Fill the missing value with 0's Fill the missing values with the avg values

Case 1:(values missing at the start) Ex1: _ _ _ x =>ceil(x/4), ceil(x/4), ceil(x/4), ceil(x/4) Ex2: _ _ x => ceil(x/3), ceil(x/3), ceil(x/3) Case 2:(values missing in middle)Ex1: x _ _ y => ceil((x+y)/4), ceil((x+y)/4), ceil((x+y)/4), ceil((x+y)/4) Ex2: x _ _ _ y => ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5) Case 3:(values missing at the end) Ex1: x _ _ _ => ceil(x/4), ceil(x/4), ceil(x/4), ceil(x/4) x _ => ceil(x/2), ceil(x/2)

In [0]:
```
# Fills a value of zero for every bin where no pickup data is present
# the count_values: number pickps that are happened in each region for each 10min intravel
# ther icksups.
```

```
# values: number of unique bins

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 0 to the smoothed data
# we finally return smoothed data
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,30):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [0]:

```
# Fills a value of zero for every bin where no pickup data is present
# the count_values: number pickps that are happened in each region for each 10min intravel
# there wont be any value if there are no picksups.
# values: number of unique bins

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the above markdown ce
ll)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,30):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already visited/resolved
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup bin if it exists
```

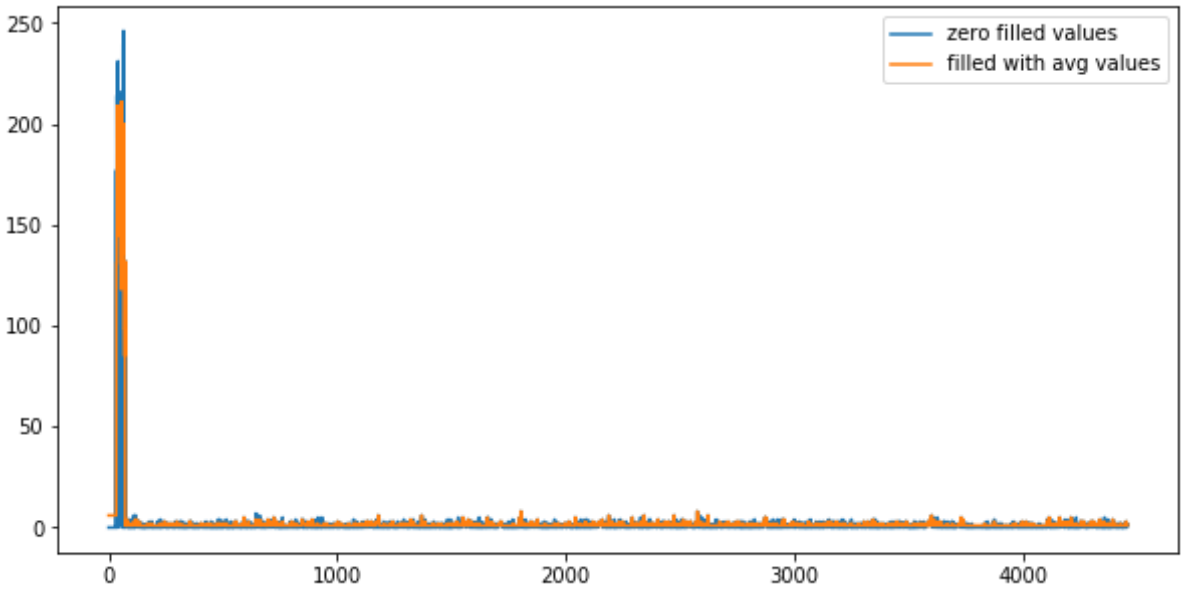
```
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin value which has a pi
ckup value
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing,hence we have no right-l
imit here
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.0/((right_hand_limit-i)+2)*1.0

                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missing,hence we have no left-
limit here
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]:
                            continue
                        else:
                            right_hand_limit=j
                            break
                    smoothed_value=count_values[ind]*1.0/((right_hand_limit-i)+1)*1.0
                    for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                    repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [0]:
```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pickups that are happened
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015_unique)

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_unique)
```

In [221]:
```
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*30*60/10 = 4320
# for each cluster we will have 4464 values, therefore 40*4464 = 178560 (length of the jan_2015_fill)
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

```
number of 10min intravels among all the clusters  133920
```

In [222]:
```
# Smoothing vs Filling
# sample plot that shows two variations of filling missing values
# we have taken the number of pickups for cluster region 2
plt.figure(figsize=(10,5))
plt.plot(jan_2015_fill[4464:8920], label="zero filled values")
plt.plot(jan_2015_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```

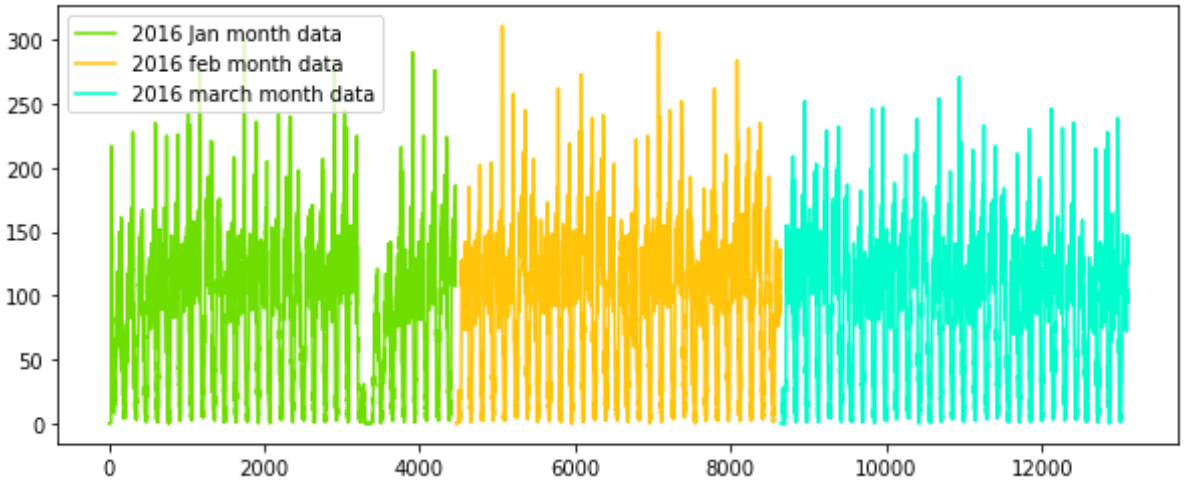


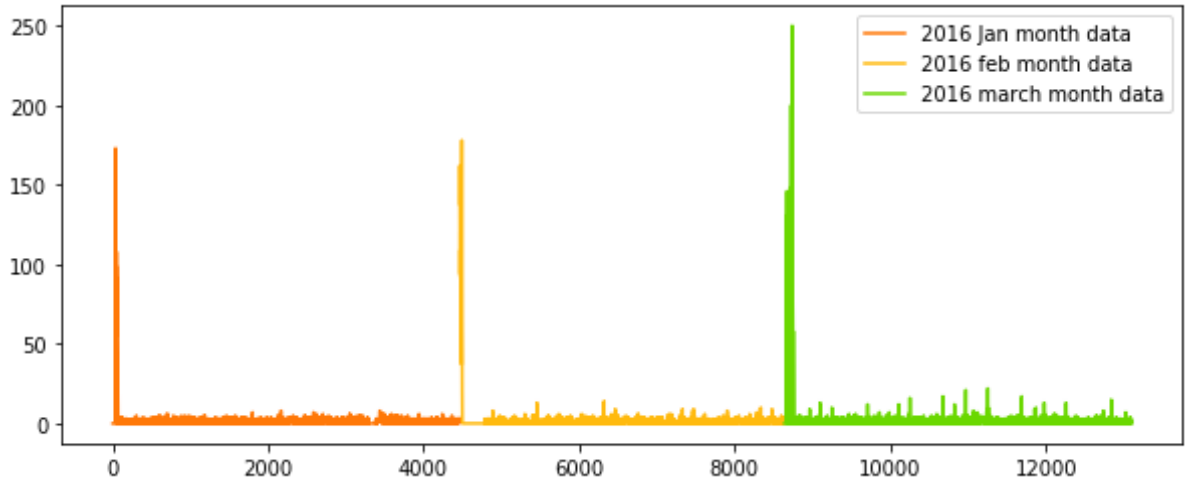
In [0]:
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled with zero
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_unique)
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_unique)
mar_2016_smooth = fill_missing(mar_2016_groupby['trip_distance'].values,mar_2016_unique)

# Making list of all the values of pickup data in every bin for a period of 3 months and storing them region-wise
regions_cum = []

# a =[1,2,3]
# b = [2,3,4]
# a+b = [1, 2, 3, 2, 3, 4]

# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*31*60/10 = 4464
# regions_cum: it will contain 40 lists, each list will contain 4464+4176+4464 values which represents the number o
f pickups
# that are happened for three months in 2016 data

for i in range(0,30):
```

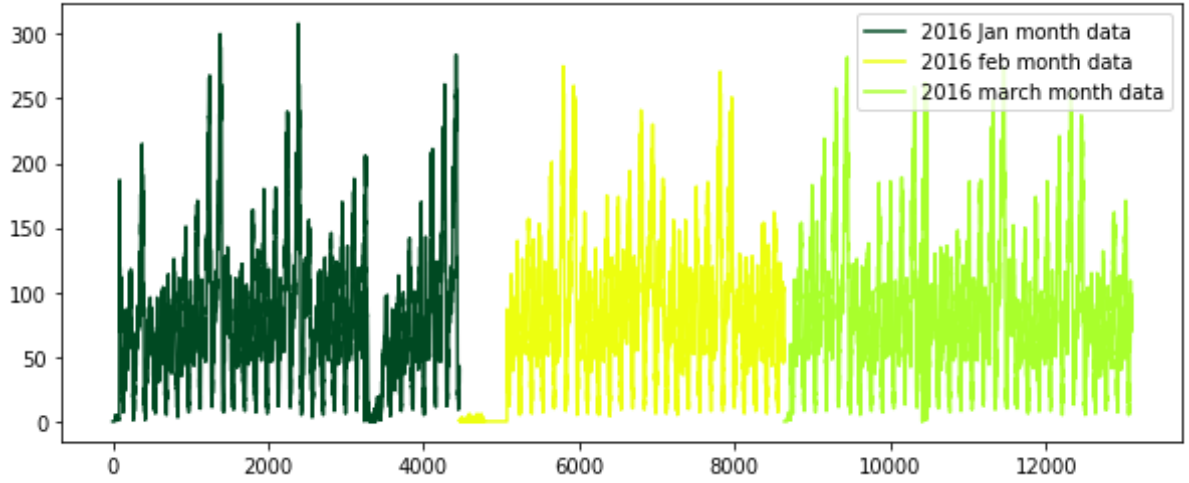
```
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_smooth[4176*i:4176*(i+1)]+mar_2016_smooth[4464*i
:4464*(i+1)])
```

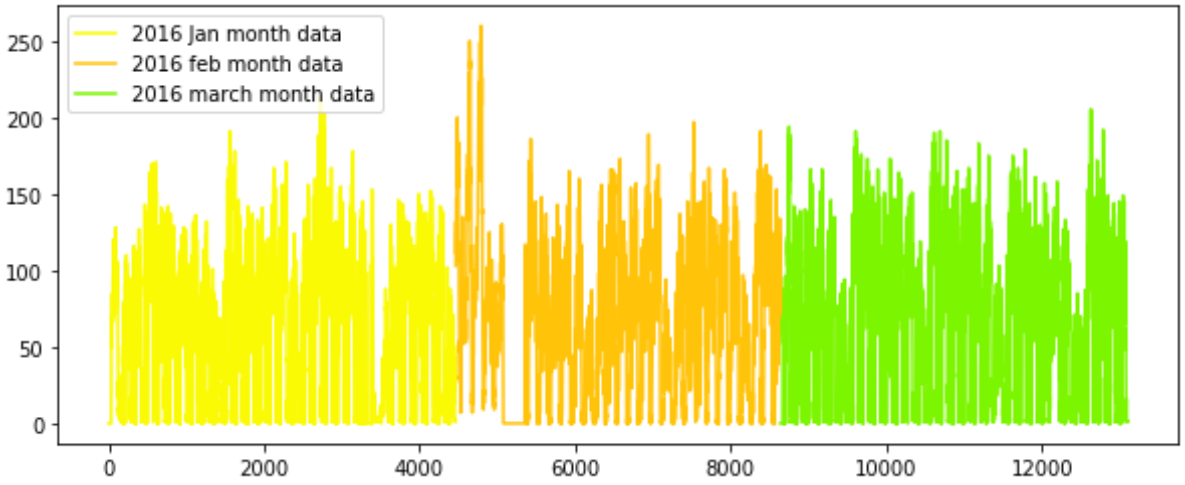
Time series and Fourier Transforms

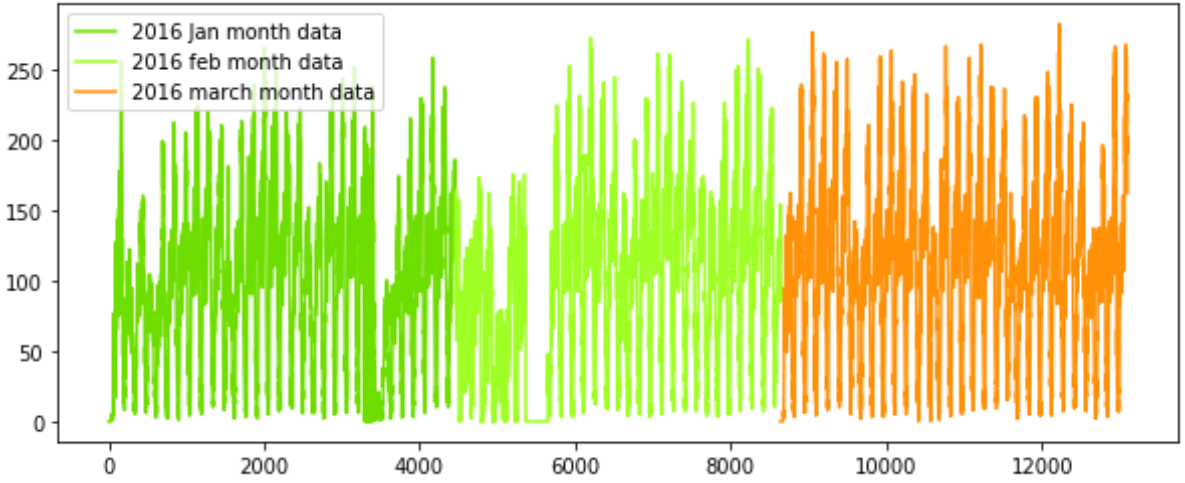
In [224]:
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(30):
    plt.figure(figsize=(10,4))
    plt.plot(first_x,regions_cum[i][:4464], color=uniqueish_color(), label='2016 Jan month data')
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), label='2016 feb month data')
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='2016 march month data')
    plt.legend()
    plt.show()
```

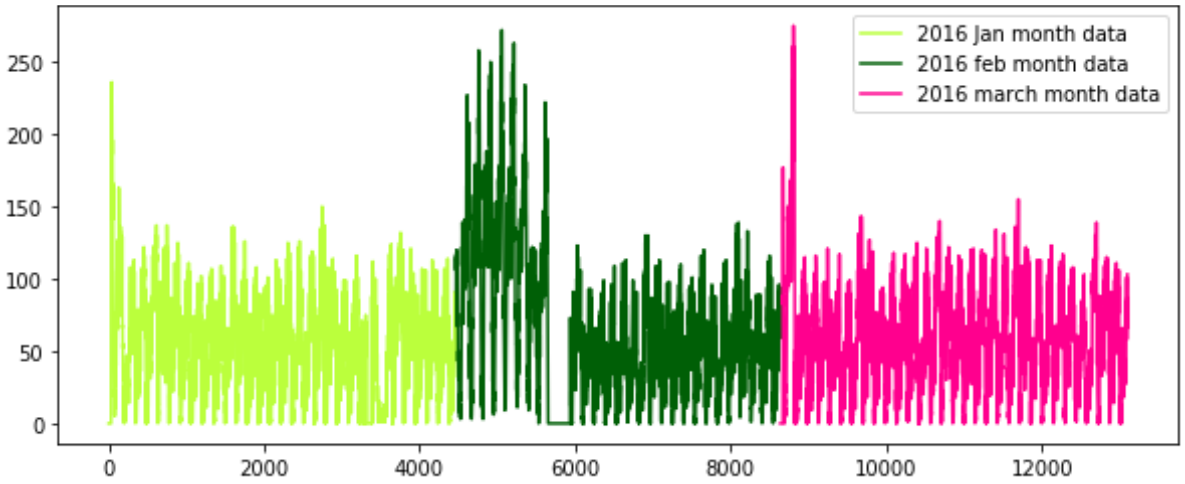


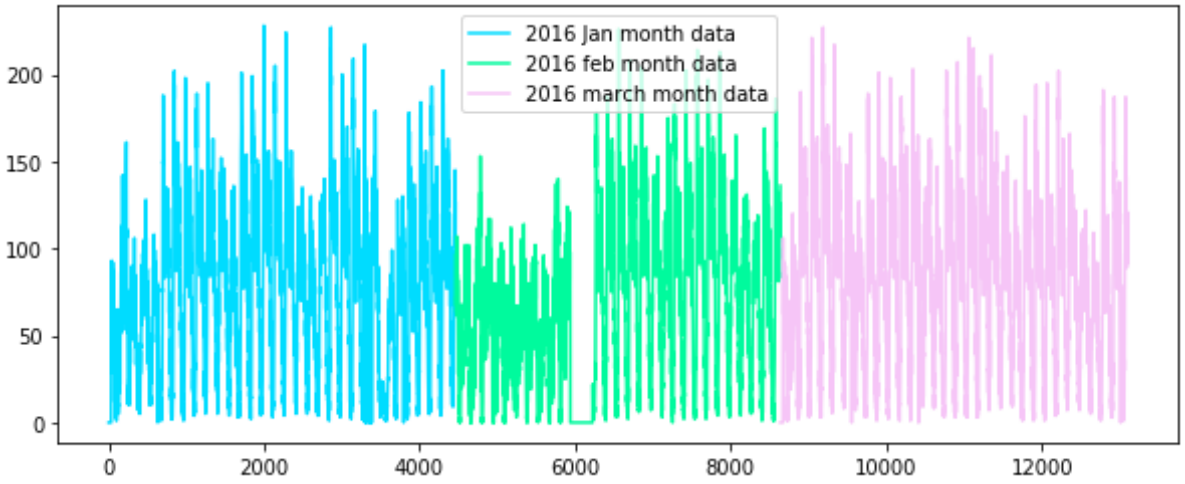


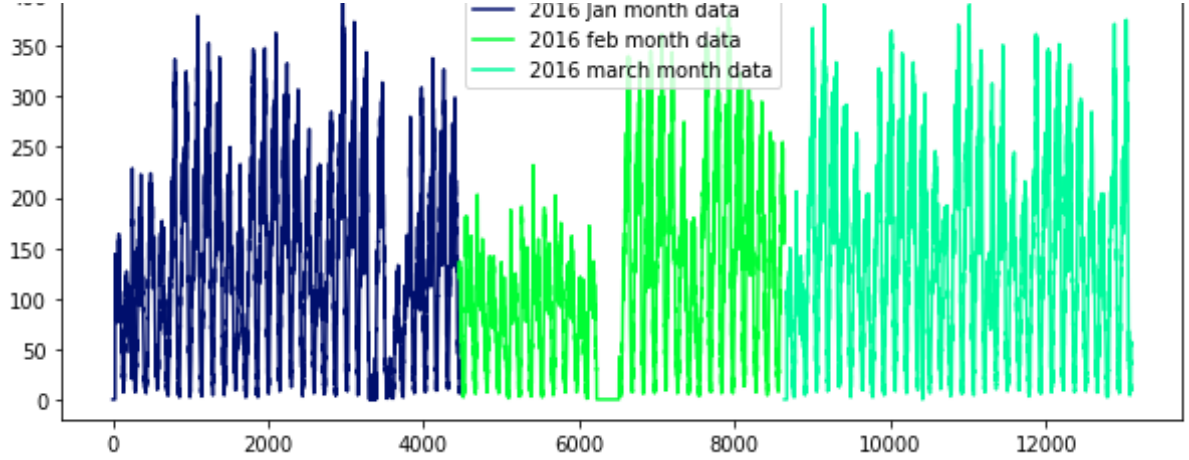


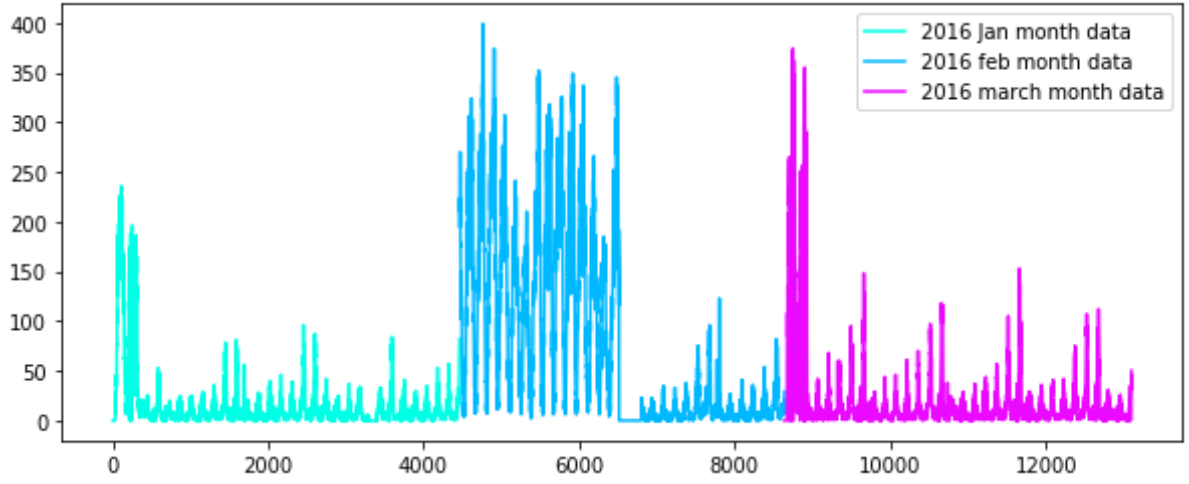


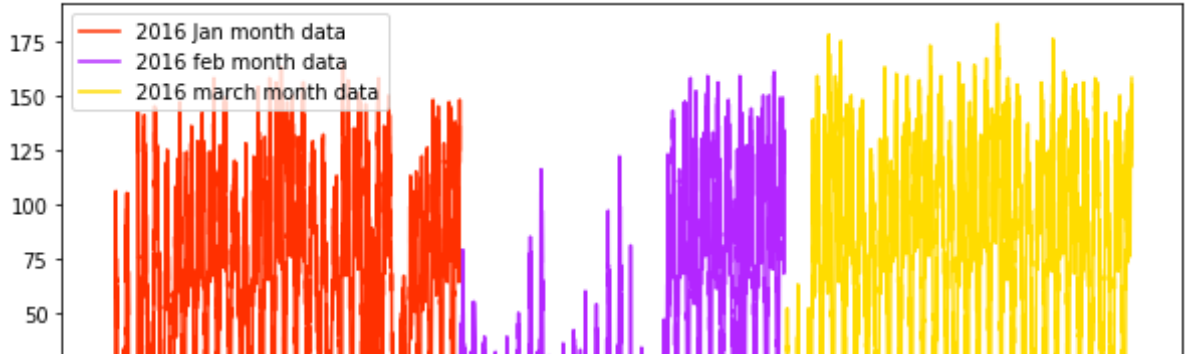


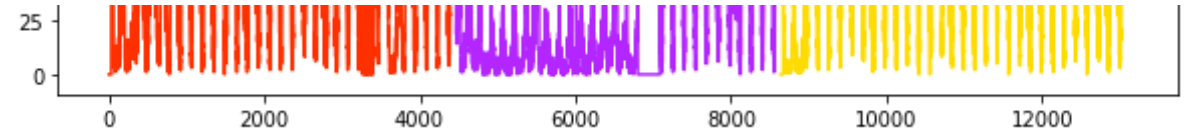


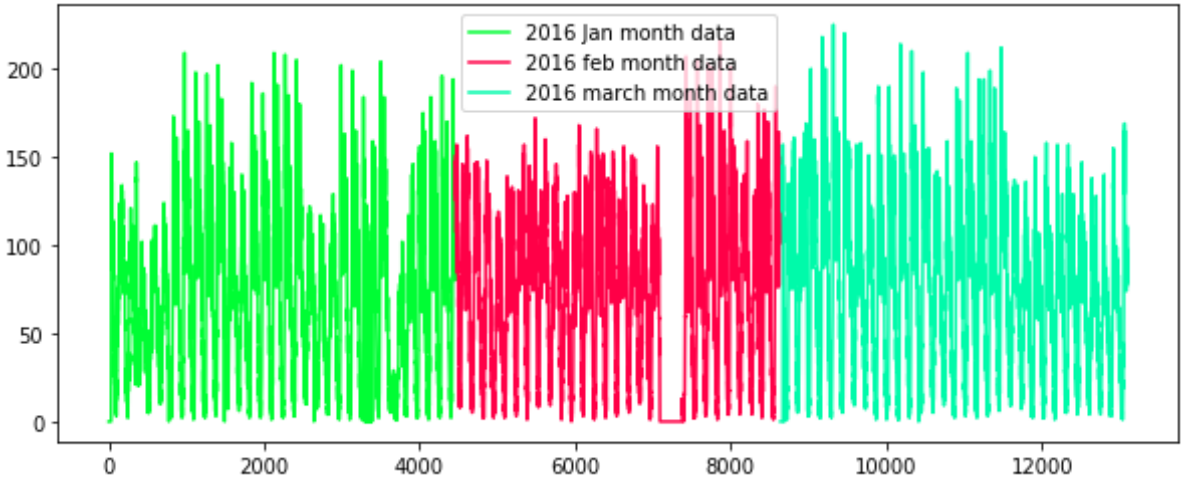


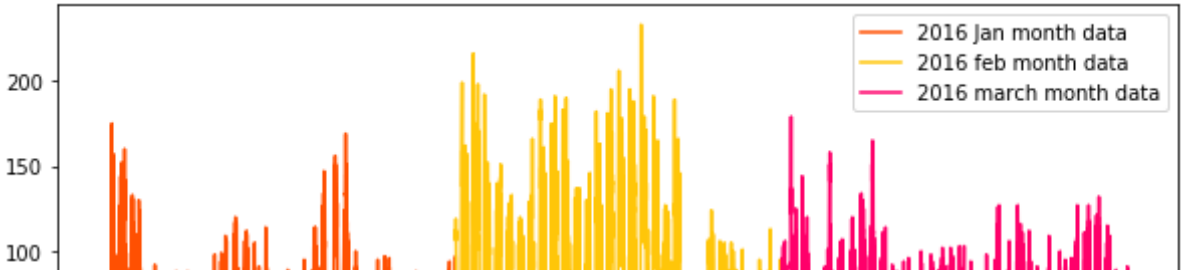


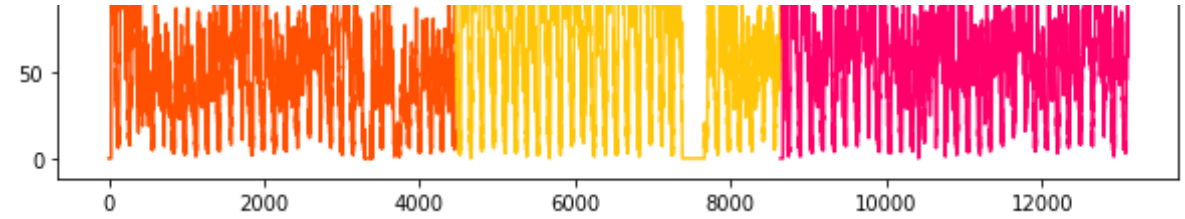


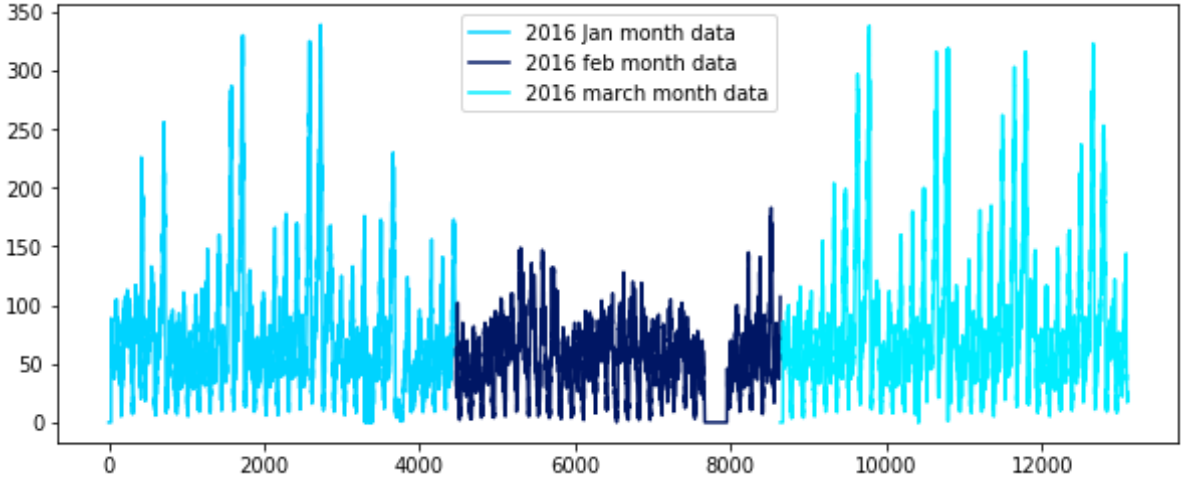


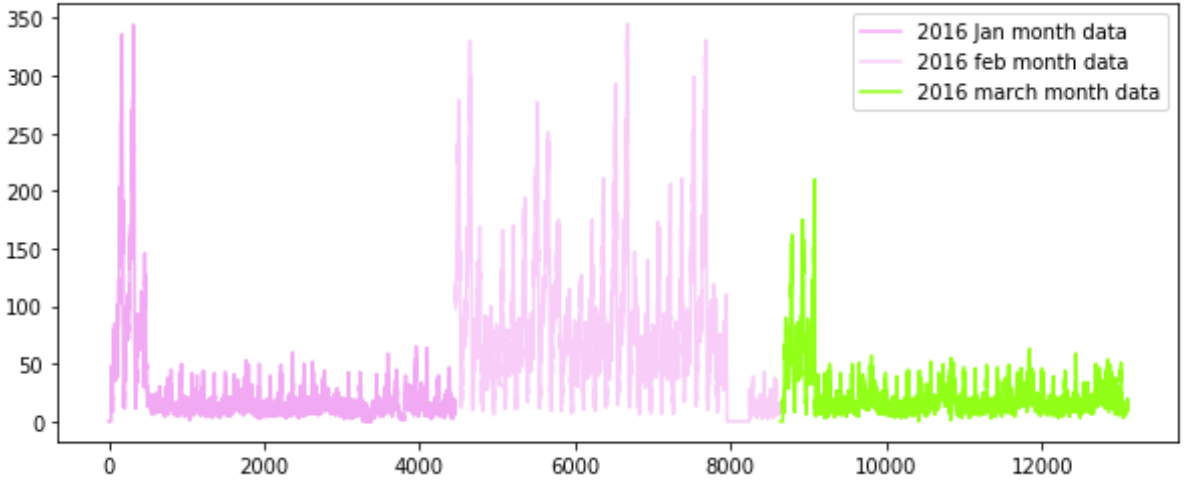


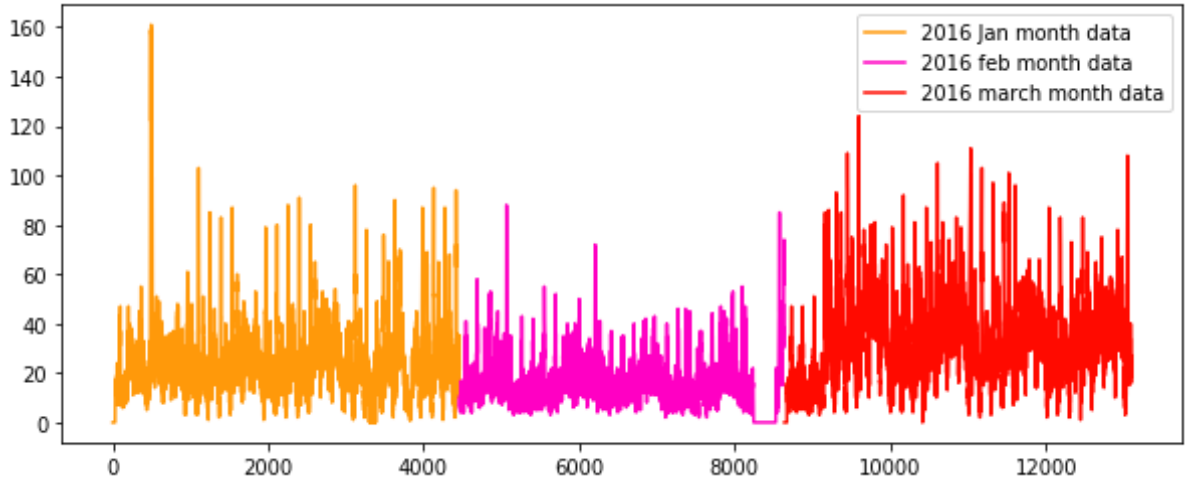


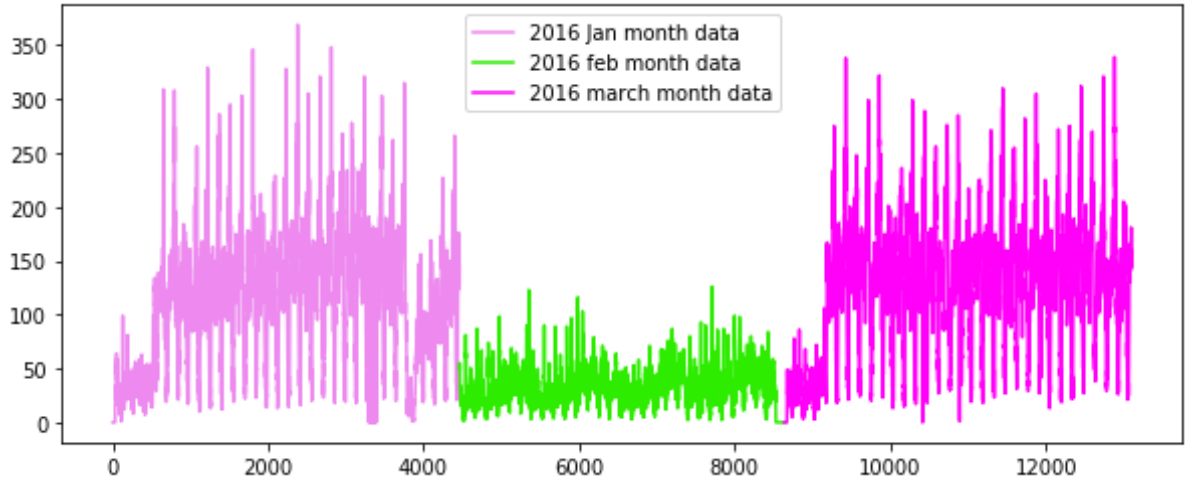


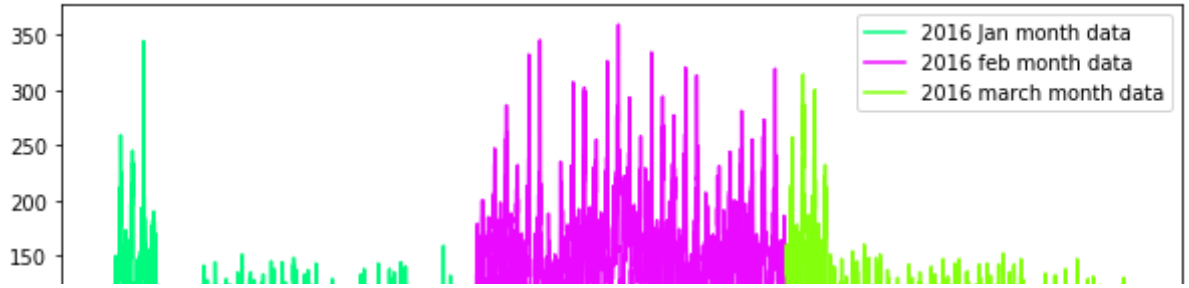


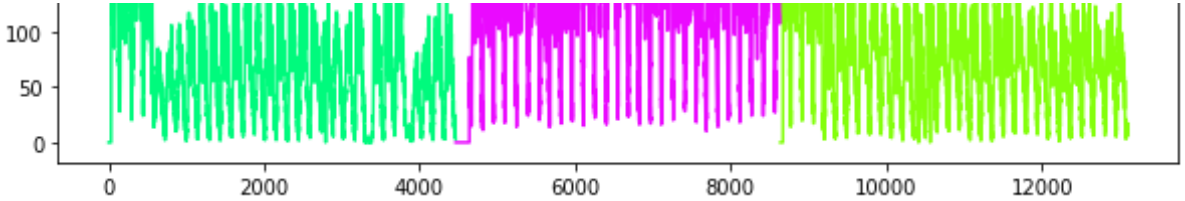


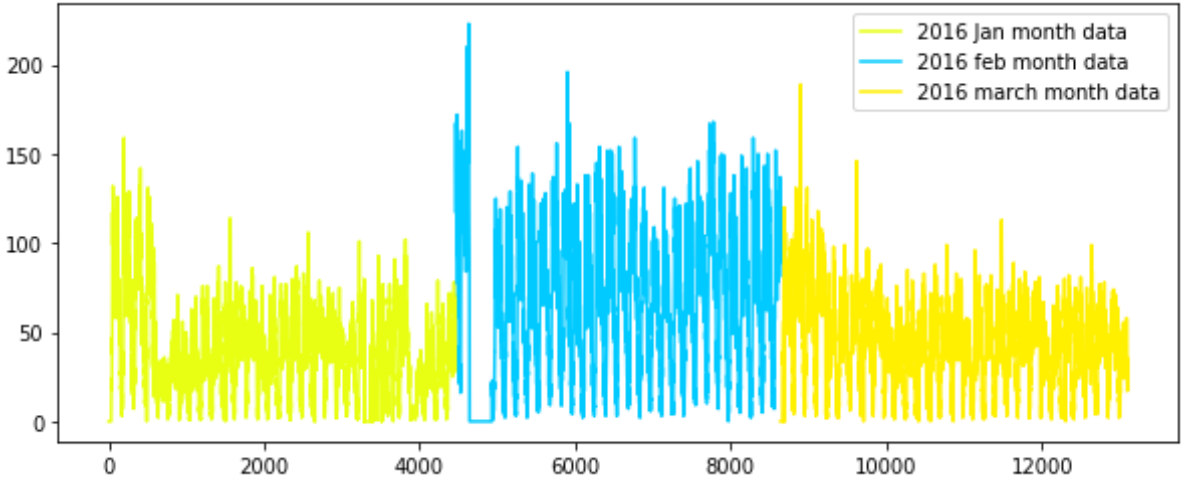


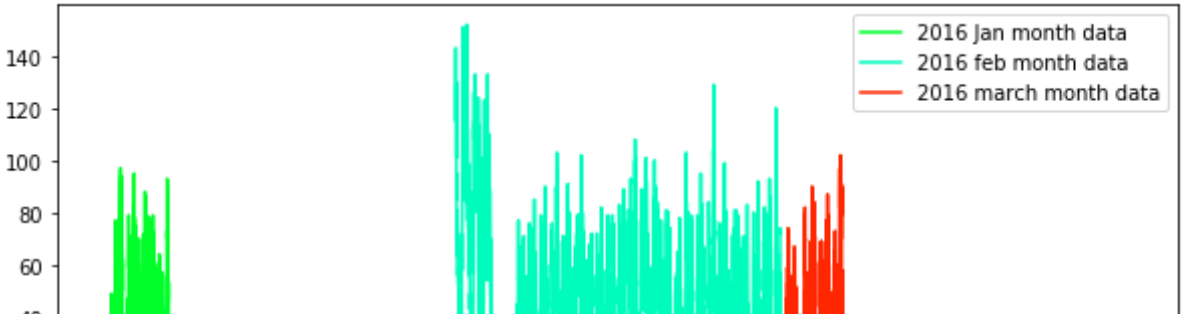


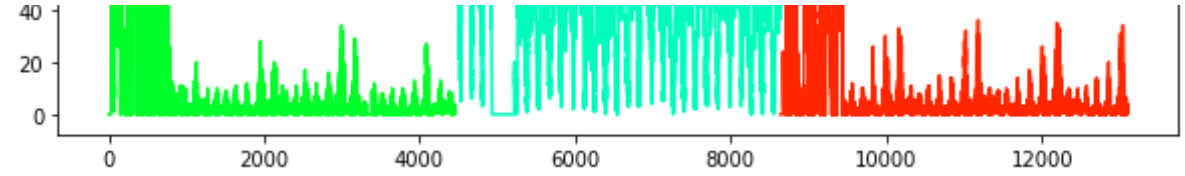


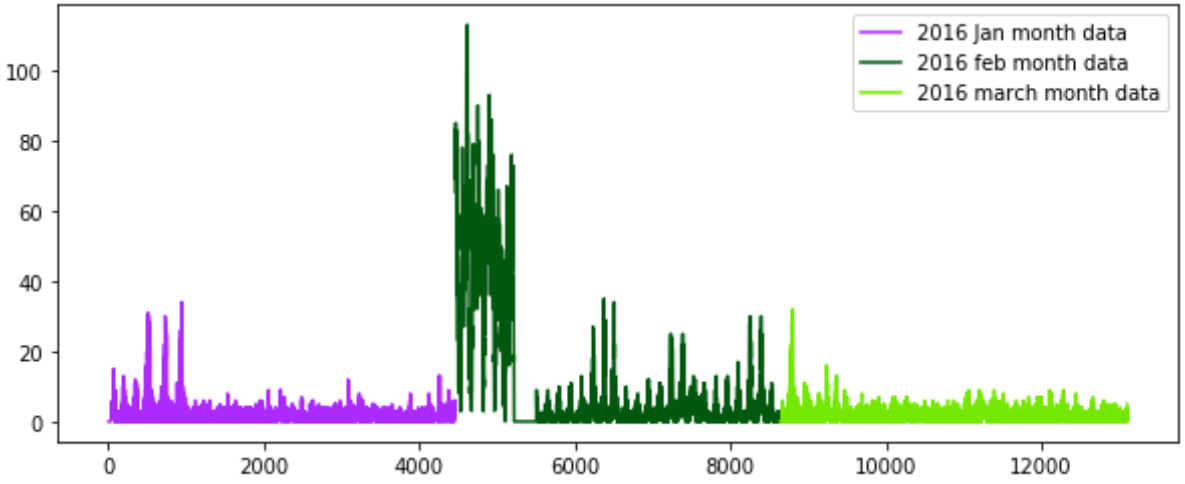


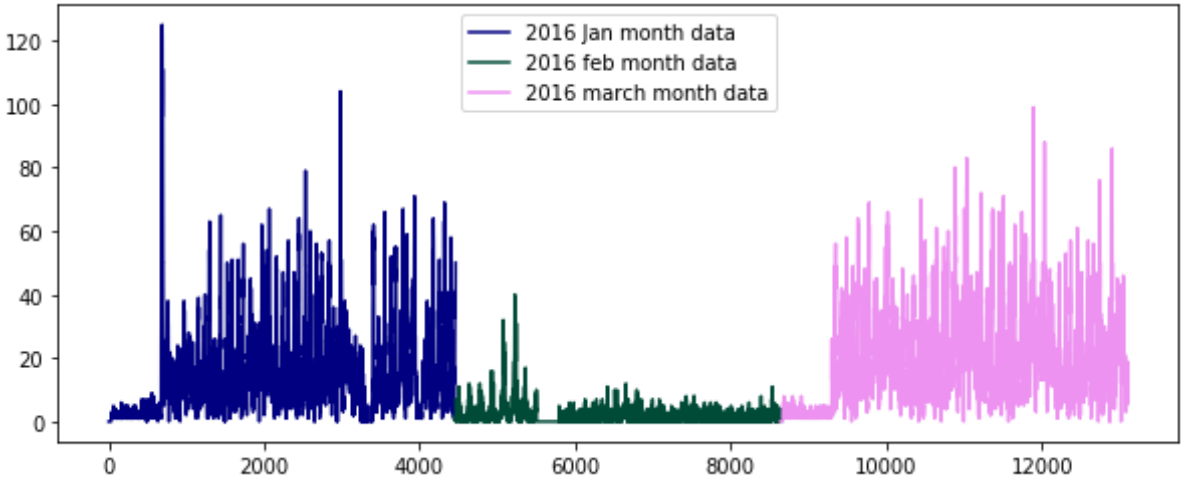


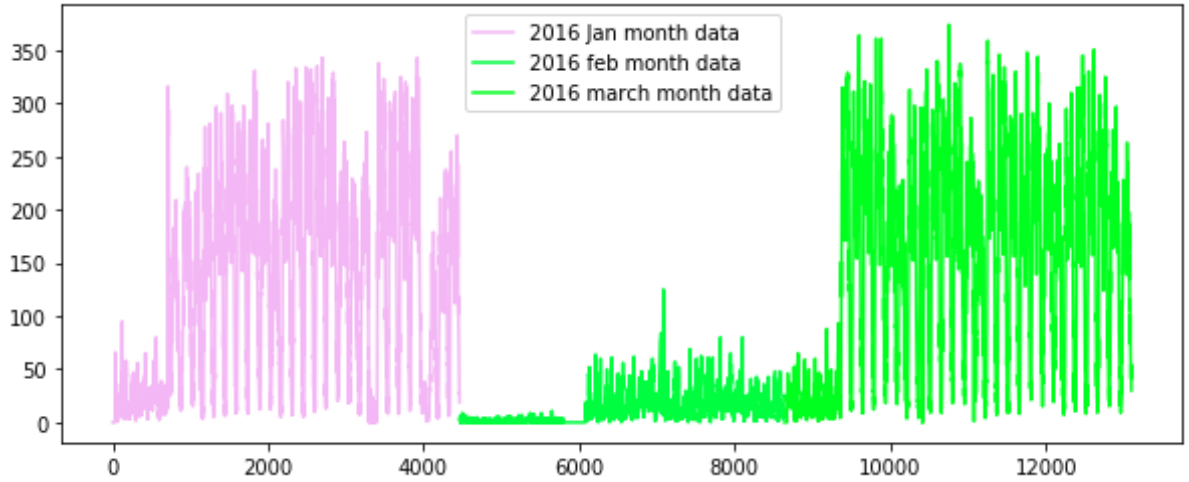


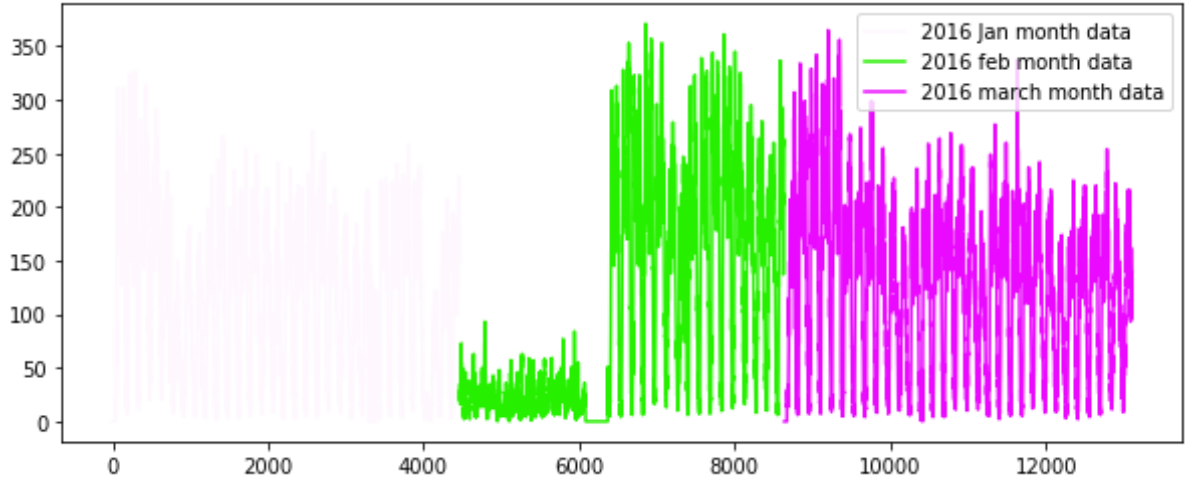


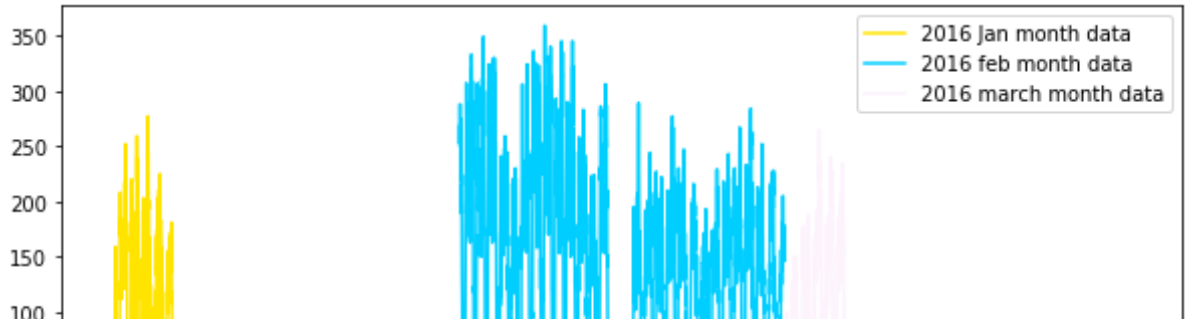


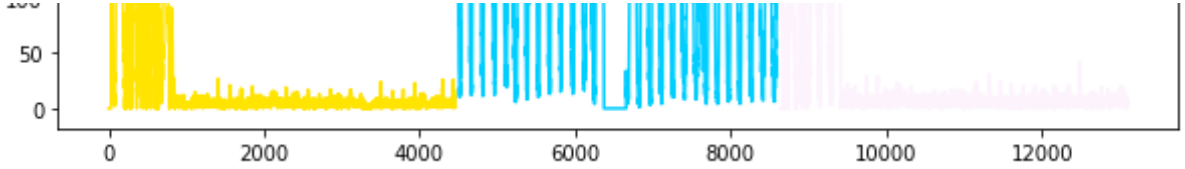


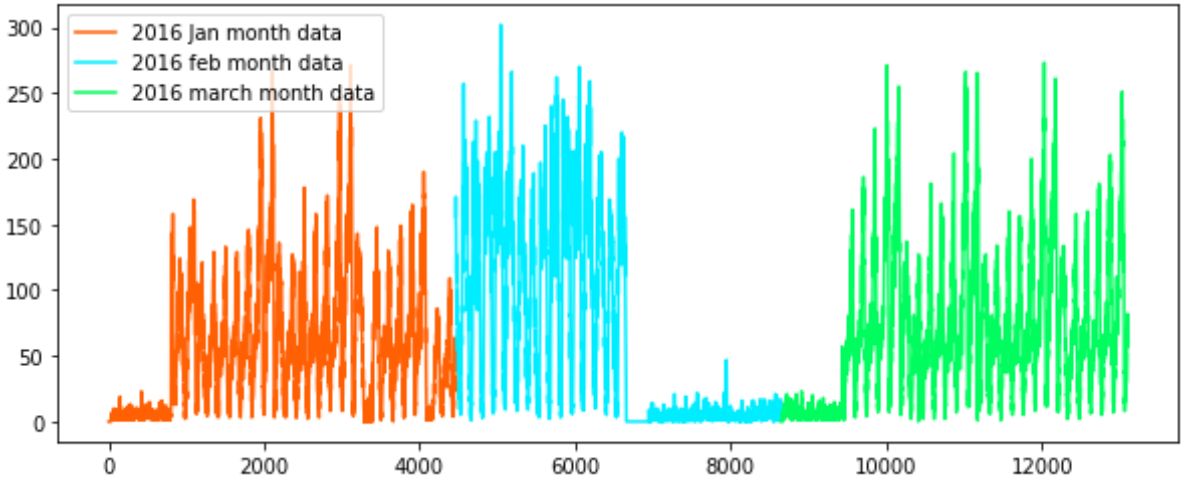


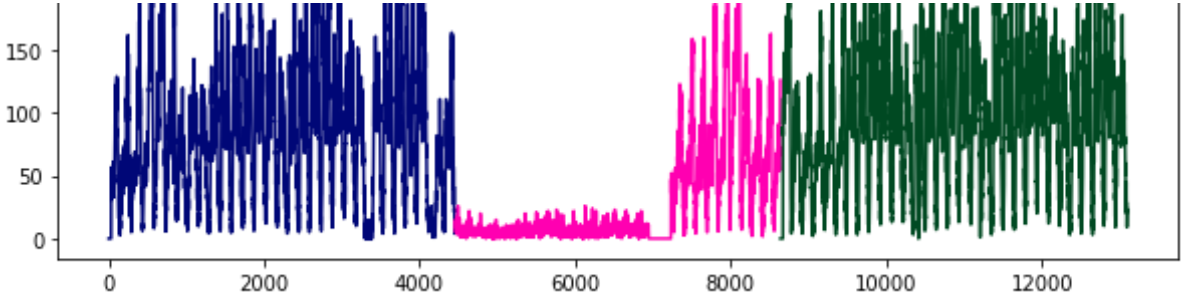


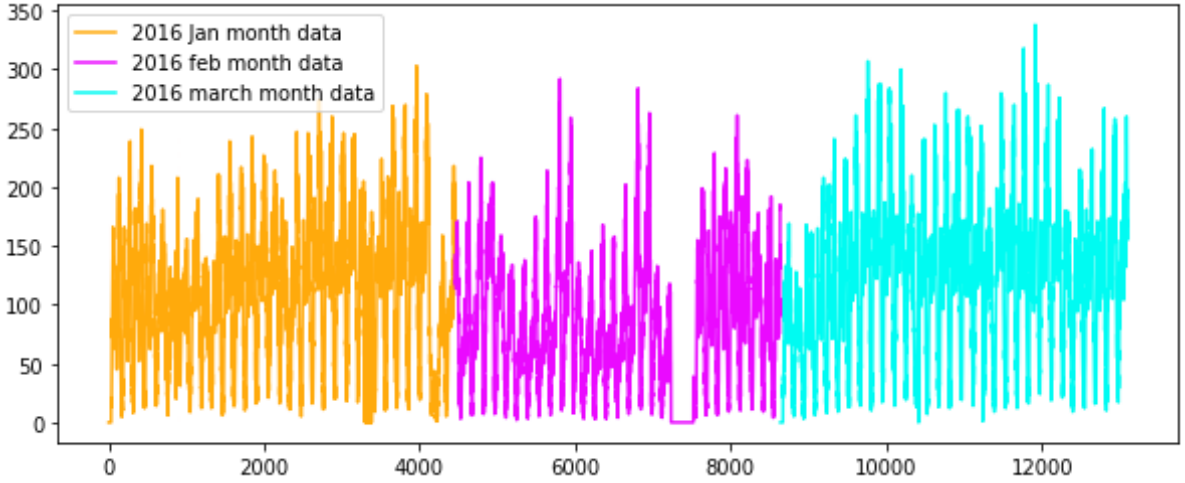


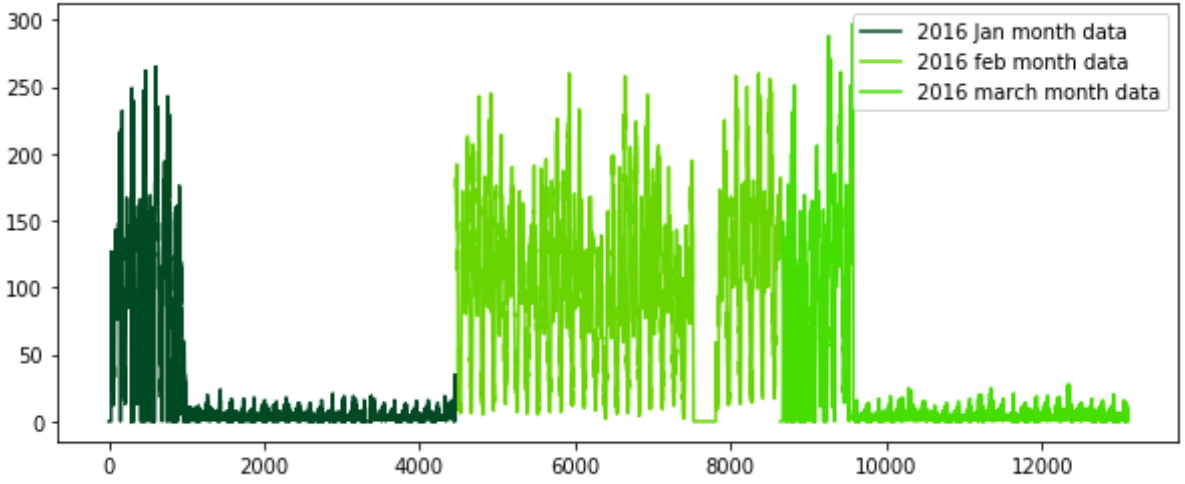


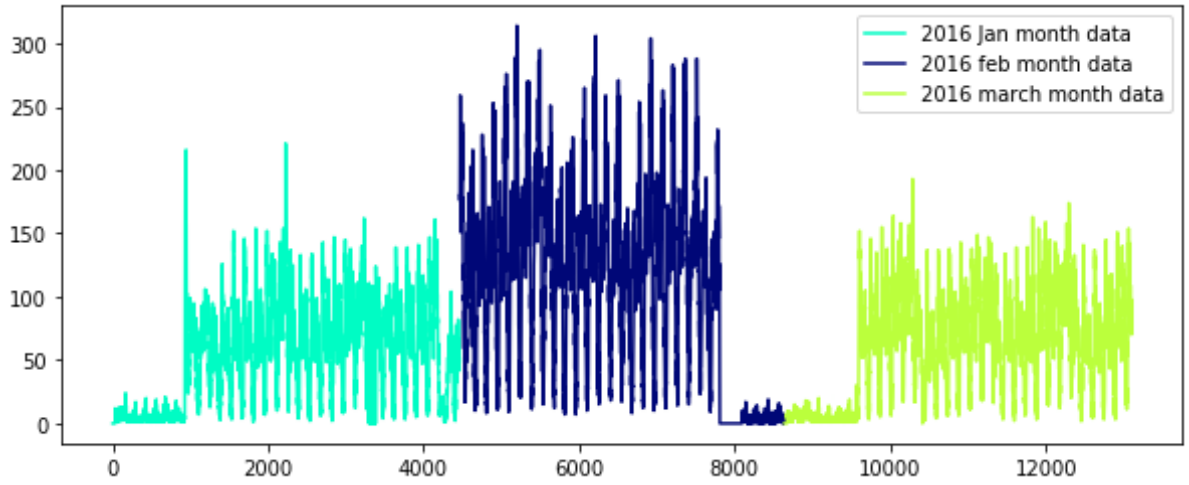


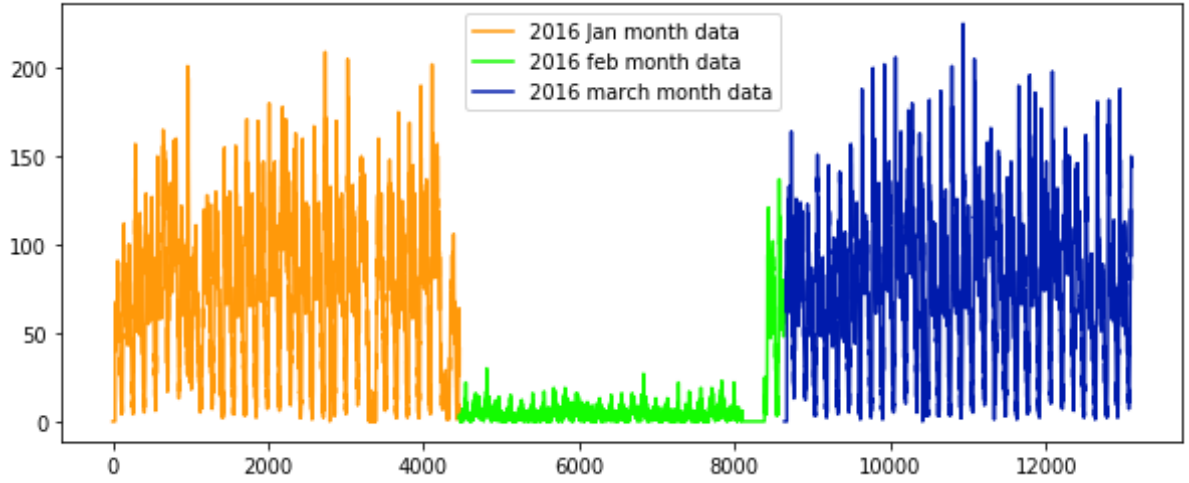




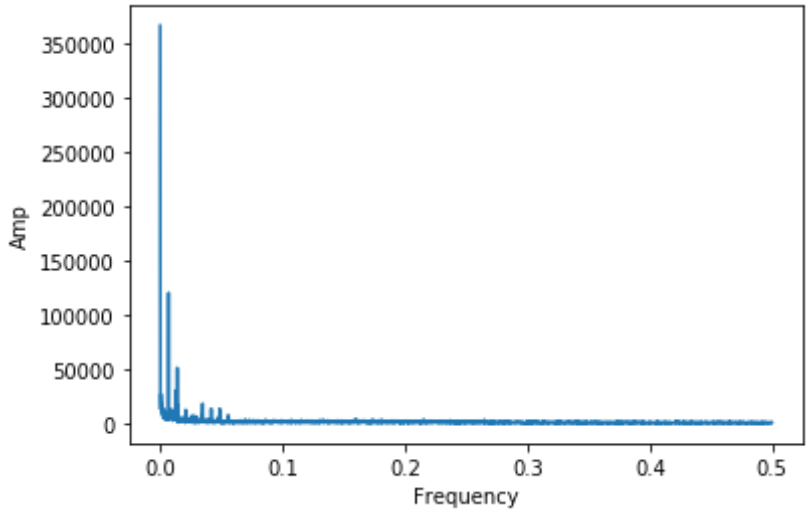








In [225]:
```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html
freq = np.fft.fftfreq(4460, 1)
n = len(freq)
plt.figure()
plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
plt.xlabel("Frequency")
plt.ylabel('Amp')
plt.show()
```



In [0]:
```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) values as jan-2016
ratios_jan = pd.DataFrame()
ratios_jan['Given']=jan_2015_smooth
ratios_jan['Prediction']=jan_2016_smooth
ratios_jan['Ratios']=ratios_jan['Prediction']*1.0/ratios_jan['Given']*1.0
```

In [227]:
```
ratios_jan.tail().style.highlight_max(axis=0)
```

Out[227]:

	Given	Prediction	Ratios
133915	112	50	0.446429
133916	99	45	0.454545
133917	123	26	0.211382
133918	97	46	0.474227
133919	100	41	0.41

Modelling: Baseline Models

Now we get into modelling in order to forecast the pickup densities for the months of Jan, Feb and March of 2016 for which we are using multiple models with two variations

1. Using Ratios of the 2016 data to the 2015 data
2. Using Previous known values of the 2016 data itself to predict the future values

Simple Moving Averages

The First Model used is the Moving Averages Model which uses the previous n values in order to predict the next value

In [0]:
```
def MA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    error=[]
    predicted_values=[]
    window_size=3
    predicted_ratio_values=[]
    for i in range(0,4464*30):
```

```
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Prediction'].values)
[i],1))))
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])/window_size
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


    ratios['MA_R_Predicted'] = predicted_values
    ratios['MA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

In [0]:
```
def MA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=1
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_size:(i+1)])/window_size)
        else:
            predicted_value=int(sum((ratios['Prediction'].values)[0:(i+1)])/(i+1))

    ratios['MA_P_Predicted'] = predicted_values
    ratios['MA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

Weighted Moving Averages

The Moving Avergaes Model used gave equal importance to all the values in the window used, but we know intuitively that the future is more likely to be similar to the latest values and less similar to the older values. Weighted Averages converts this analogy into a mathematical relationship giving the highest weight while computing the averages to the latest previous value and decreasing weights to the subsequent older ones

In [0]:
```
def WA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Prediction'].values)
[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

    ratios['WA_R_Predicted'] = predicted_values
    ratios['WA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

In [0]:
```
def WA_P_Predictions(ratios,month):
```

```
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,4464*30):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Prediction'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Prediction'].values)[j-1]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

    ratios['WA_P_Predicted'] = predicted_values
    ratios['WA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

For the above the Hyperparameter is the window-size (n) which is tuned manually and it is found that the window-size of 2 is optimal for getting the best results using Weighted Moving Averages using previous 2016 values

Exponential Weighted Moving Averages

In [0]:
```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*30):
```

```
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Prediction'].values)
[i],1))))
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].values)[i])

    ratios['EA_R1_Predicted'] = predicted_values
    ratios['EA_R1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

In [0]:
```
def EA_P1_Predictions(ratios,month):
    predicted_value= (ratios['Prediction'].values)[0]
    alpha=0.3
    error=[]
    predicted_values=[]
    for i in range(0,4464*30):
        if i%4464==0:
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Prediction'].values)[i]))

    ratios['EA_P1_Predicted'] = predicted_values
    ratios['EA_P1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

In [0]:
```
mean_err=[0]*6

median_err=[0]*6
ratios_jan,mean_err[0],median_err[0]=MA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[1],median_err[1]=MA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[2],median_err[2]=WA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[3],median_err[3]=WA_P_Predictions(ratios_jan,'jan')
```

```
ratios_jan,mean_err[4],median_err[4]=EA_R1_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[5],median_err[5]=EA_P1_Predictions(ratios_jan,'jan')
```

In [235]:
```
print(len(mean_err))
print(len(median_err))
```

```
6
6
```

Comparison between baseline models

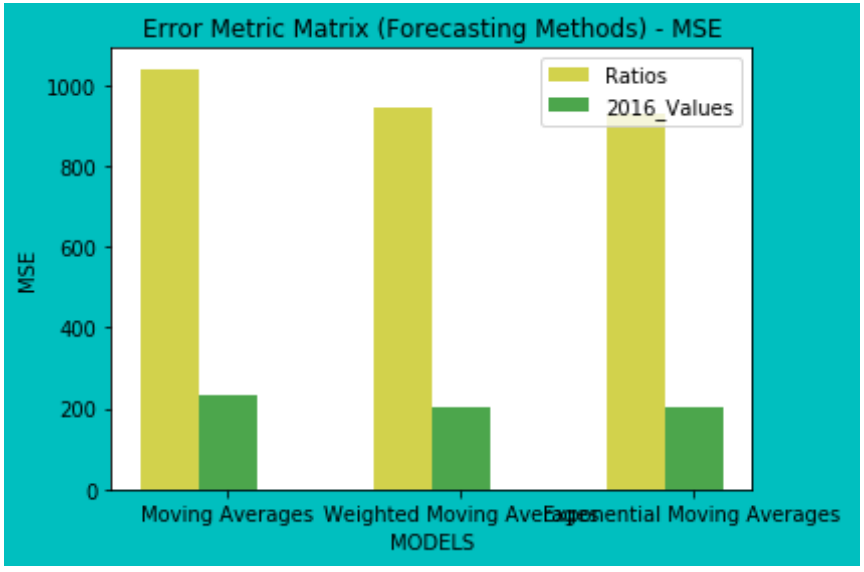
We have chosen our error metric for comparison between models as MAPE (Mean Absolute Percentage Error) so that we can know that on an average how good is our model with predictions and MSE (Mean Squared Error) is also used so that we have a clearer understanding as to how well our forecasting model performs with outliers so that we make sure that there is not much of a error margin between our prediction and the actual value

In [236]:
```
print ("Error Metric Matrix (Forecasting Methods) - MAPE & MSE")
print ("--------------------------------------------------------------------------------------------------------------")
print ("Moving Averages (Ratios) -                          MAPE: ",mean_err[0],"      MSE: ",median_err[0])
print ("Moving Averages (2016 Values) -                     MAPE: ",mean_err[1],"       MSE: ",median_err[1])
print ("--------------------------------------------------------------------------------------------------------------")
print ("Weighted Moving Averages (Ratios) -                 MAPE: ",mean_err[2],"      MSE: ",median_err[2])
print ("Weighted Moving Averages (2016 Values) -            MAPE: ",mean_err[3],"      MSE: ",median_err[3])
print ("--------------------------------------------------------------------------------------------------------------")
print ("Exponential Moving Averages (Ratios) -              MAPE: ",mean_err[4],"      MSE: ",median_err[4])
print ("Exponential Moving Averages (2016 Values) -         MAPE: ",mean_err[5],"      MSE: ",median_err[5])
print ("--------------------------------------------------------------------------------------------------------------")
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------------------------------------
Moving Averages (Ratios) -                          MAPE:  0.21831142210509133       MSE:  1040.1551523297492
Moving Averages (2016 Values) -                     MAPE:  0.151601053669964        MSE:  231.68163829151732
--------------------------------------------------------------------------------------------------------------
Weighted Moving Averages (Ratios) -                 MAPE:  0.21731873135290639       MSE:  946.013903823178
Weighted Moving Averages (2016 Values) -            MAPE:  0.14388121249540275       MSE:  204.2053614097969
--------------------------------------------------------------------------------------------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.21716844300381055       MSE:  930.1732900238949
Exponential Moving Averages (2016 Values) -         MAPE:  0.14360553564203168       MSE:  202.78418458781363
--------------------------------------------------------------------------------------------------------------
```

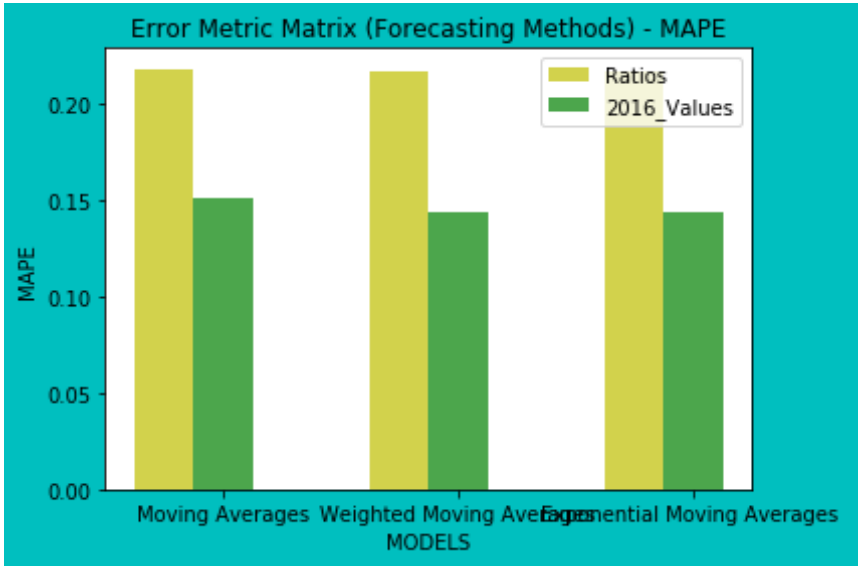
In [237]:
```
n_gro=3
Ratios=(median_err[0],median_err[2],median_err[4])
```

```
Values_2016=(median_err[1],median_err[3],median_err[5])
fig4 = plt.figure( facecolor='c', edgecolor='k')
index = np.arange(n_gro)
bar_width = 0.25
opacity = 0.7

result1 = plt.bar(index, Ratios, bar_width,
alpha=opacity,
color='y',
label='Ratios')

result2 = plt.bar(index + bar_width, Values_2016, bar_width,
alpha=opacity,
color='g',
label='2016_Values')

plt.xlabel('MODELS')
plt.ylabel('MSE')
plt.title('Error Metric Matrix (Forecasting Methods) - MSE')
plt.xticks(index + bar_width, ('Moving Averages', 'Weighted Moving Averages',
                               'Exponential Moving Averages '))
plt.legend()

plt.tight_layout()
plt.show()
```



In [238]:

```
n_gro=3
Ratios=(mean_err[0],mean_err[2],mean_err[4])
Values_2016=(mean_err[1],mean_err[3],mean_err[5])
fig4 = plt.figure( facecolor='c', edgecolor='k')
index = np.arange(n_gro)
bar_width = 0.25
opacity = 0.7

result1 = plt.bar(index, Ratios, bar_width,
alpha=opacity,
color='y',
label='Ratios')

result2 = plt.bar(index + bar_width, Values_2016, bar_width,
alpha=opacity,
color='g',
label='2016_Values')

plt.xlabel('MODELS')
plt.ylabel('MAPE')
plt.title('Error Metric Matrix (Forecasting Methods) - MAPE')
plt.xticks(index + bar_width, ('Moving Averages', 'Weighted Moving Averages',
                               'Exponential Moving Averages '))
plt.legend()

plt.tight_layout()
plt.show()
```



The histogram plot for Error Metric Matrix (Forecasting Methods) - MAPE & MSE is shown as above. From above histogram,It can be conclud that MAPE is best error metric as comapred to MSE metric.

Regression Models

Train-Test Split

Before we start predictions using the tree based regression models we take 3 months of 2016 pickup data and split it such that for every region we have 75% data in train and 25% in test, ordered date-wise for every region

In [0]:
```
# Preparing data to be split into train and test, The below prepares data in cumulative form which will be later sp
lit into test and train
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*31*60/10 = 4464
# regions_cum: it will contain 40 lists, each list will contain 4464+4176+4464 values
#which represents the number of pickups
# that are happened for three months in 2016 data
```

```
# print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 12960

# we take number of pickups that are happened in last 5 10min intravels
number_of_time_stamps = 5

# output varaible
# it is list of lists
# it will contain number of pickups 13099 for each cluster
output = []


# tsne_lat will contain 13104-5=13099 times lattitude of cluster center for every cluster
# Ex: [[cent_lat 13099times],[cent_lat 13099times], [cent_lat 13099times].... 40 lists]
# it is list of lists
tsne_lat = []


# tsne_lon will contain 13104-5=13099 times logitude of cluster center for every cluster
# Ex: [[cent_long 13099times],[cent_long 13099times], [cent_long 13099times].... 40 lists]
# it is list of lists
tsne_lon = []

# we will code each day
# sunday = 0, monday=1, tue = 2, wed=3, thur=4, fri=5,sat=6
# for every cluster we will be adding 13099 values, each value represent to which day of the week that pickup bin b
elongs to
# it is list of lists
tsne_weekday = []

# its an numbpy array, of shape (523960, 5)
# each row corresponds to an entry in out data
# for the first row we will have [f0,f1,f2,f3,f4] fi=number of pickups happened in i+1th 10min intravel(bin)
# the second row will have [f1,f2,f3,f4,f5]
# the third row will have [f2,f3,f4,f5,f6]
# and so on...
tsne_feature = []


tsne_feature = [0]*number_of_time_stamps
fram_final = pd.DataFrame(columns= ['f_1','a_1',
                                    'f_2','a_2',
                                    'f_3','a_3',
```

```
                                        'f_4','a_4',
                                        'f_5','a_5'])
for i in range(0,30):
  # amplitutude €& frequecy are calcualted &  saved in dataframe
    ampJan = np.fft.fft(np.array(regions_cum[i][0:4464]))
    freqJan = np.fft.fftfreq((4464), 1)
    ampFeb = list(np.fft.fft(np.array(regions_cum[i])[4464:(4176+4464)]))
    freqFeb = list(np.fft.fftfreq((4176), 1))
  #print('okay')
    ampMar = list(np.fft.fft(np.array(regions_cum[i])[(4176+4464):(4176+4464+4464)]))
    freqMar = list(np.fft.fftfreq((4464), 1))
    fram_jan = pd.DataFrame(data=freqJan,columns=['Freq'])
    fram_jan = pd.DataFrame(data=ampJan,columns=['Amp'])
    fram_feb =  pd.DataFrame(data=freqFeb,columns=['Freq'])
    fram_feb = pd.DataFrame(data=ampFeb,columns=['Amp'])
    fram_mar =  pd.DataFrame(data=freqMar,columns=['Freq'])
    fram_mar = pd.DataFrame(data=ampMar,columns=['Amp'])




    fram_list_jan = []
    fram_list_feb = []
    fram_list_mar = []

    fram_jan_sort = fram_jan.sort_values(by=['Amp'], ascending=False)[:5].reset_index(drop=True).T
  #print(fram_jan_sorted)
    fram_feb_sort = fram_feb.sort_values(by=['Amp'], ascending=False)[:5].reset_index(drop=True).T
    fram_mar_sort = fram_mar.sort_values(by=['Amp'], ascending=False)[:5].reset_index(drop=True).T
 # print(fram_mar_sort)
  #print(type(fram_jan_sort['Freq'][0]))
    for j in range(0,5):
        fram_list_jan.append(float(fram_jan_sort[j]))
        fram_list_jan.append(float(fram_jan_sort[j]))

        fram_list_feb.append(float(fram_feb_sort[j]))
        fram_list_feb.append(float(fram_feb_sort[j]))

        fram_list_mar.append(float(fram_mar_sort[j]))
        fram_list_mar.append(float(fram_mar_sort[j]))

    data1=[fram_list_jan]*4464
    data2=[fram_list_feb]*4176
    data3=[fram_list_mar]*4464
    col_name=['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','a_5']
```

```
    fram_new_jan = pd.DataFrame(data=data1,columns=col_name)
    fram_new_feb = pd.DataFrame(data=data2,columns=col_name)
    fram_new_mar = pd.DataFrame(data=data3,columns=col_name)


    fram_final = fram_final.append(fram_new_jan, ignore_index=True)
    fram_final = fram_final.append(fram_new_feb, ignore_index=True)
    fram_final = fram_final.append(fram_new_mar, ignore_index=True)
  #print([kmeans.cluster_centers_[i][0]]*5)
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13099)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13099)
  # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
  # our prediction start from 5th 10min intravel since we need to have number of pickups that are happened in last
 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+4464)])
  # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104],
 [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for r in range(0,len(regions
_cum[i])-number_of_time_stamps)]))
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
fram_final.drop(['f_1'],axis=1,inplace=True)

fram_final = fram_final
fram_final = fram_final.fillna(0)
```

Amplitude and frequency of regions_cum are calculated and saved in dataframe .These features are used for further time series analysis.

In [240]:
```
print(tsne_feature.shape)
print(fram_final.shape)
```

```
(392970, 5)
(393120, 9)
```

In [241]:
```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_weekday[0]) == 30*13099 == le
n(output)*len(output[0])
```

Out[241]: True

In [0]:
```
alpha=0.3
predicted_values=[]
predict_list = []
```

```
tsne_flat_exp_avg = []

for r in range(0,30):
    for i in range(0,13104):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

In [243]:
```
print(fram_final.head(4))
```

```
        a_1           f_2  ...           f_5           a_5
0  367173.0  94490.188858  ...  14349.849101  14349.849101
1  367173.0  94490.188858  ...  14349.849101  14349.849101
2  367173.0  94490.188858  ...  14349.849101  14349.849101
3  367173.0  94490.188858  ...  14349.849101  14349.849101

[4 rows x 9 columns]
```

In [244]:
```
# train, test split : 75% 25% split
# Before we star|||||t predictions using the tree based regression models we take 3 months of 2016 pickup data
# and split it such that for every region we have 70% data in train and 30% in test,
# ordered date-wise for every region
print("size of train data :", int(13099*0.75))
print("size of test data :", int(13099*0.25))
```

```
size of train data : 9824
size of test data : 3274
```

Splitting data into 75% to tarn and 25% to test

In [0]:
```
# extracting first 9824 timestamp values i.e 75% of 13099 (total timestamps) for our training data
train_features =  [tsne_feature[i*13099:(13099*i+9824)] for i in range(0,30)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(13099*(i))+9824:13099*(i+1)] for i in range(0,30)]
```

In [0]:
```
train_features =  [tsne_feature[i*13099:(13099*i+9824)] for i in range(0,30)]
```

```
test_features = [tsne_feature[(13099*(i))+9824:13099*(i+1)] for i in range(0,30)]
fram_final_train = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3',
                                         'f_4','a_4','f_5','a_5'])
fram_final_test = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3',
                                        'f_4','a_4','f_5','a_5'])
for i in range(0,30):
  #print(fram_final[i*13099:(13099*i+9824)])
  fram_final_train = fram_final_train.append(fram_final[i*13099:(13099*i+9824)] )
fram_final_train.reset_index(inplace=True)
for i in range(0,30):
  #print(fram_final[(13099*(i))+9824:13099*(i+1)])
  fram_final_test = fram_final_test.append(fram_final[(13099*(i))+9824:13099*(i+1)])
fram_final_test.reset_index(inplace=True)
```

In [0]:
```
fram_final_test.drop(['index'],axis=1,inplace=True)
fram_final_train.drop(['index'],axis=1,inplace=True)
```

In [248]:
```
print("Number of data clusters",len(train_features), "Number of data points in trian data", len(train_features[0]),
 "Each data point contains", len(train_features[0][0]),"features")
print("Number of data clusters",len(train_features), "Number of data points in test data", len(test_features[0]),
"Each data point contains", len(test_features[0][0]),"features")
```

```
Number of data clusters 30 Number of data points in trian data 9824 Each data point contains 5 features
Number of data clusters 30 Number of data points in test data 3275 Each data point contains 5 features
```

In [0]:
```
# extracting first 9824 timestamp values i.e 75% of 13099 (total timestamps) for our training data
tsne_train_flat_lat = [i[:9824] for i in tsne_lat]
tsne_train_flat_lon = [i[:9824] for i in tsne_lon]
tsne_train_flat_weekday = [i[:9824] for i in tsne_weekday]
tsne_train_flat_output = [i[:9824] for i in output]
tsne_train_flat_exp_avg = [i[:9824] for i in predict_list]
```

In [0]:
```
# extracting the rest of the timestamp values i.e 25% of 12956 (total timestamps) for our test data
tsne_test_flat_lat = [i[9824:] for i in tsne_lat]
tsne_test_flat_lon = [i[9824:] for i in tsne_lon]
tsne_test_flat_weekday = [i[9824:] for i in tsne_weekday]
tsne_test_flat_output = [i[9824:] for i in output]
tsne_test_flat_exp_avg = [i[9824:] for i in predict_list]
```

In [0]:
```
# the above contains values in the form of list of lists
#(i.e. list of values of each region), here we make all of them in one list
train_new_features = []
```

```
for i in range(0,30):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,30):
    test_new_features.extend(test_features[i])
```

In [0]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

tsne_train_lat = sum(tsne_train_flat_lat, [])
tsne_train_lon = sum(tsne_train_flat_lon, [])
tsne_train_weekday = sum(tsne_train_flat_weekday, [])
tsne_train_output = sum(tsne_train_flat_output, [])
tsne_train_exp_avg = sum(tsne_train_flat_exp_avg,[])
```

In [0]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

tsne_test_lat = sum(tsne_test_flat_lat, [])
tsne_test_lon = sum(tsne_test_flat_lon, [])
tsne_test_weekday = sum(tsne_test_flat_weekday, [])
tsne_test_output = sum(tsne_test_flat_output, [])
tsne_test_exp_avg = sum(tsne_test_flat_exp_avg,[])
```

In [254]:
```
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1']
df_train = pd.DataFrame(data=train_new_features, columns=columns)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon
df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg

print(df_train.shape)
```

```
(294720, 9)
```

In [255]:
```
df_test = pd.DataFrame(data=test_new_features, columns=columns)
df_test['lat'] = tsne_test_lat
df_test['lon'] = tsne_test_lon
```

```
df_test['weekday'] = tsne_test_weekday
df_test['exp_avg'] = tsne_test_exp_avg
print(df_test.shape)
```

```
(98250, 9)
```

In [256]:
```
df_test.head()
```

Out[256]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg
0	36	40	36	25	22	40.776228	-73.982119	2	23
1	40	36	25	22	24	40.776228	-73.982119	2	23
2	36	25	22	24	21	40.776228	-73.982119	2	21
3	25	22	24	21	15	40.776228	-73.982119	2	16
4	22	24	21	15	14	40.776228	-73.982119	2	14

In [257]:
```
df_test_lm = pd.concat([df_test, fram_final_test], axis=1)
df_train_lm = pd.concat([df_train, fram_final_train], axis=1)
#df_train_lm=df_train_lm.isnull().fillna(0)
df_test_lm.head()
print(df_test_lm.columns)
print(df_train_lm.columns)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg', 'a_1', 'f_2', 'a_2', 'f_3', 'a_3', 'f_4', 'a_4', 'f_5',
       'a_5'],
      dtype='object')
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg', 'a_1', 'f_2', 'a_2', 'f_3', 'a_3', 'f_4', 'a_4', 'f_5',
       'a_5'],
      dtype='object')
```

In [258]:
```
nan_rows = df_train_lm[df_train_lm.isnull().T.any().T]
print(nan_rows)
```

```
Empty DataFrame
Columns: [ft_5, ft_4, ft_3, ft_2, ft_1, lat, lon, weekday, exp_avg, a_1, f_2, a_2, f_3, a_3, f_4, a_4, f_5, a_5]
Index: []
```

Using Linear Regression

Here,df_train_lm & df_test_lm dataframe is used

In [0]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection  import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import RandomizedSearchCV
```

In [260]:
```
std_train=StandardScaler().fit_transform(df_train_lm)
std_test=StandardScaler().fit_transform(df_test_lm)
# parameter tuning
classifier_Lr= SGDRegressor(loss='squared_loss',penalty='l2')
lambda_val=[10**-14, 10**-12, 10**-10, 10**-8, 10**-6,
          10**-4, 10**-2, 10**0, 10**2, 10**4, 10**6]
HP={'alpha':lambda_val}
# 3fold cross-validation
grid_param=GridSearchCV(classifier_Lr,HP,
                        scoring='neg_mean_absolute_error',cv=3)
grid_param.fit(std_train,tsne_train_output)
best_alpha=grid_param.best_params_['alpha']
print('best_alpha==',best_alpha)
```

```
best_alpha== 1e-10
```

In [261]:
```
print(grid_param.best_estimator_)
print(grid_param.best_params_)
```

```
SGDRegressor(alpha=1e-10, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
{'alpha': 1e-10}
```

In [262]:
```
#applying linear regress ion with best hyper-parameter
clf = SGDRegressor(loss = "squared_loss", penalty = "l2", alpha = best_alpha)
clf.fit(std_train, tsne_train_output)
```

Out[262]:
```
SGDRegressor(alpha=1e-10, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
```

```
learning_rate='invscaling', loss='squared_loss', max_iter=1000,
n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
warm_start=False)
```

In [0]:
```
y_pred = clf.predict(std_test)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = clf.predict(std_train)
lr_train_predictions = [round(value) for value in y_pred]
```

In [0]:
```
df_train=df_train_lm
df_test=df_test_lm
```

Using Random Forest Regressor

In [265]:
```
#Hyper parameter tuning
C=[10,20,40]
random_clf=RandomForestRegressor(n_jobs=-1)
HP1={'n_estimators':C}
n_iter1=10
# 3fold cross-validation
grid_param1=RandomizedSearchCV(random_clf,HP1,
                        scoring='neg_mean_absolute_error',cv=3,
                               n_iter=n_iter1)
grid_param1.fit(df_train,tsne_train_output)
best_alpha1=grid_param1.best_params_['n_estimators']
print('best_alpha==',best_alpha1)
```

```
best_alpha== 40
```

In [266]:
```
clf1 = RandomForestRegressor(max_features='sqrt',min_samples_leaf=4,min_samples_split=3,n_estimators = best_alpha1,
 n_jobs = -1)
clf1.fit(df_train, tsne_train_output)
```

Out[266]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='sqrt', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=4, min_samples_split=3,
                      min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=-1,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False)
```

In [0]:
```
y_pred = clf1.predict(df_test)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = clf1.predict(df_train)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [268]:
```
#feature importances based on analysis using random forest
print (df_train.columns)
print (clf1.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg', 'a_1', 'f_2', 'a_2', 'f_3', 'a_3', 'f_4', 'a_4', 'f_5',
       'a_5'],
      dtype='object')
[0.07715164 0.1079789  0.12044316 0.16696689 0.19771569 0.00083081
 0.0009164  0.00139817 0.28773428 0.01592562 0.00098804 0.00086283
 0.001054   0.00138759 0.00083397 0.01129011 0.00134828 0.0051736 ]
```

Using XgBoost Regressor

In [269]:
```
start=datetime.datetime.now()
HP2= {"max_depth": [3, 4,5],
              "min_child_weight": [3, 4,5,6],
              "gamma":[0,0.1,0.2],
              "colsample_bytree":[0.7,0.8,0.9],
              "nthread":[3,4,5]
              }
clf111 = xgb.XGBRegressor()
n_iter2 = 10
random_clf = RandomizedSearchCV(clf111, param_distributions=HP2,
                                 n_iter=n_iter2)

random_clf.fit(df_train, tsne_train_output)
```

```
[16:00:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:00:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
```

```
arederror.
[16:01:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:01:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:20] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:02:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
```

```
[16:03:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:03:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:04:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
[16:04:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
```

Out[269]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None,
                                          objective='reg:linear',
                                          rand...
                                          reg_lambda=1, scale_pos_weight=1,
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'colsample_bytree': [0.7, 0.8, 0.9],
                                        'gamma': [0, 0.1, 0.2],
                                        'max_depth': [3, 4, 5],
                                        'min_child_weight': [3, 4, 5, 6],
                                        'nthread': [3, 4, 5]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [270]:
```
best_gamma=random_clf.best_params_['gamma']
print('best_gamma==',best_gamma)
```

```
best_gamma== 0.1
```

In [271]:
```
best_max_depth=random_clf.best_params_['max_depth']
print('best_max_depth==',best_max_depth)

best_min_child_weight  = random_clf.best_params_['min_child_weight']
print('best_min_child_weight==',best_min_child_weight)
```

```
best_colsample_bytree=random_clf.best_params_['colsample_bytree']
print('best_colsample_bytree==',best_colsample_bytree)
best_nthread=random_clf.best_params_['nthread']
print('best_nthread==',best_nthread)
```

```
best_max_depth== 5
best_min_child_weight== 5
best_colsample_bytree== 0.7
best_nthread== 5
```

In [272]:
```
clf2=xgb.XGBRegressor( learning_rate =0.1,
                      n_estimators=1000,
max_depth=best_max_depth,
                      min_child_weight=best_min_child_weight,
                      gamma=best_gamma,
                      subsample=0.8,
 reg_alpha=200, reg_lambda=200,
                      colsample_bytree=best_colsample_bytree,
                      nthread=best_nthread









)


clf2.fit(df_train, tsne_train_output)
```

```
[16:04:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squ
arederror.
```

Out[272]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, gamma=0.1,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=5, missing=None, n_estimators=1000,
             n_jobs=1, nthread=5, objective='reg:linear', random_state=0,
             reg_alpha=200, reg_lambda=200, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.8, verbosity=1)
```

In [0]:
```
y_pred = clf2.predict(df_test)
xgb_test_predictions = [round(value) for value in y_pred]
y_pred = clf2.predict(df_train)
xgb_train_predictions = [round(value) for value in y_pred]
```

Calculation of error metrics for different models

In [0]:
```
train_mape=[]
test_mape=[]

train_mape.append((mean_absolute_error(tsne_train_output,
                                       df_train['ft_5'].values))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].values))/(sum(tsne_train_output)/len(t
sne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_predictions))/(sum(tsne_train_output)/len(tsne_
train_output)))
train_mape.append((mean_absolute_error(tsne_train_output, xgb_train_predictions))/(sum(tsne_train_output)/len(tsne_
train_output)))
train_mape.append((mean_absolute_error(tsne_train_output, lr_train_predictions))/(sum(tsne_train_output)/len(tsne_t
rain_output)))

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_5'].values))/(sum(tsne_test_output)/len(tsne_te
st_output)))
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(sum(tsne_test_output)/len(tsne
_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_test_output)/len(tsne_tes
t_output)))
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_test_output)/len(tsne_test
_output)))
test_mape.append((mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_test_output)/len(tsne_test_
output)))
```

LSTM

In [0]:
```
# reference link
# https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
# https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/
# https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/
import numpy
```

```
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [0]:
```
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [277]:
```
# Training datasets
se=pd.Series(tsne_train_output)
df_train['Y']=se.values
se1=pd.Series(tsne_test_output)
df_test['Y']=se1.values
print(df_train.tail())
```

```
        ft_5  ft_4  ft_3  ft_2  ...           a_4           f_5           a_5    Y
294715    82    76    90    75  ...  23029.545055  23029.545055  23029.545055   91
294716    76    90    75    90  ...  23029.545055  23029.545055  23029.545055   80
294717    90    75    90    91  ...  23029.545055  23029.545055  23029.545055  108
294718    75    90    91    80  ...  23029.545055  23029.545055  23029.545055  119
294719    90    91    80   108  ...  23029.545055  23029.545055  23029.545055  134

[5 rows x 19 columns]
```

In [278]:
```
df_test.tail()
```

Out[278]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	a_1	f_2	a_2	f_3	a_3
98245	93	94	101	102	119	40.776147	-73.950194	3	113	328647.0	63619.427209	63619.427209	63619.427209	63619.427209
98246	94	101	102	119	132	40.776147	-73.950194	3	126	328647.0	63619.427209	63619.427209	63619.427209	63619.427209
98247	101	102	119	132	150	40.776147	-73.950194	3	142	328647.0	63619.427209	63619.427209	63619.427209	63619.427209
98248	102	119	132	150	146	40.776147	-73.950194	3	144	328647.0	63619.427209	63619.427209	63619.427209	63619.427209
98249	119	132	150	146	144	40.776147	-73.950194	3	144	328647.0	63619.427209	63619.427209	63619.427209	63619.427209

In [0]:
```
# Normalize
scaler=MinMaxScaler(feature_range=(0, 1))
df_train = scaler.fit_transform(df_train)
#print(df_train[:1])
#print(tsne_train_output)
df_test=scaler.fit_transform(df_test)
```

In [280]:
```
print(len(df_train))
```

```
294720
```

In [0]:
```
# reference https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
# create_dataset code is taken from above link
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [282]:
```
train_size=len(df_train)
test_size=len(df_test)
print(train_size)
```

```
294720
```

In [283]:
```
# 80% of df_train=235776 for training purpose
# 20 % of df_train=58944for validation purpose
training_data=df_train[:235776]
print(len(training_data))
val_data=df_train[235776:]
print(len(val_data))
```

```
235776
58944
```

In [0]:
```
# reshape into X=t and Y=t+1
look_back = 18
```

```
trainX, trainY = create_dataset(training_data, look_back)
valX,valY= create_dataset(val_data,look_back)
testX, testY = create_dataset(df_test, look_back)
```

In [285]:
```
print(trainX.shape[1])
```

```
18
```

In [0]:
```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
valX=numpy.reshape(valX,(valX.shape[0],1,valX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

In [287]:
```
import keras
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
keras.backend.set_epsilon(1)
model.compile(loss='mse', optimizer='adam', metrics=['mae','mape'])
historyy=model.fit(trainX, trainY, epochs=50, batch_size=100, verbose=2,
         validation_data=(valX,valY))
```

```
Train on 235757 samples, validate on 58925 samples
Epoch 1/50
 - 18s - loss: 0.0187 - mean_absolute_error: 0.0987 - mean_absolute_percentage_error: 9.8674 - val_loss: 0.0126 -
val_mean_absolute_error: 0.0933 - val_mean_absolute_percentage_error: 9.3275
Epoch 2/50
 - 17s - loss: 0.0118 - mean_absolute_error: 0.0863 - mean_absolute_percentage_error: 8.6276 - val_loss: 0.0097 -
val_mean_absolute_error: 0.0814 - val_mean_absolute_percentage_error: 8.1405
Epoch 3/50
 - 17s - loss: 0.0093 - mean_absolute_error: 0.0759 - mean_absolute_percentage_error: 7.5887 - val_loss: 0.0079 -
val_mean_absolute_error: 0.0722 - val_mean_absolute_percentage_error: 7.2200
Epoch 4/50
 - 17s - loss: 0.0077 - mean_absolute_error: 0.0680 - mean_absolute_percentage_error: 6.7979 - val_loss: 0.0067 -
val_mean_absolute_error: 0.0651 - val_mean_absolute_percentage_error: 6.5090
Epoch 5/50
 - 17s - loss: 0.0066 - mean_absolute_error: 0.0619 - mean_absolute_percentage_error: 6.1900 - val_loss: 0.0058 -
val_mean_absolute_error: 0.0597 - val_mean_absolute_percentage_error: 5.9740
Epoch 6/50
 - 17s - loss: 0.0058 - mean_absolute_error: 0.0572 - mean_absolute_percentage_error: 5.7236 - val_loss: 0.0053 -
val_mean_absolute_error: 0.0555 - val_mean_absolute_percentage_error: 5.5513
Epoch 7/50
```

```
 - 17s - loss: 0.0053 - mean_absolute_error: 0.0536 - mean_absolute_percentage_error: 5.3602 - val_loss: 0.0048 -
val_mean_absolute_error: 0.0522 - val_mean_absolute_percentage_error: 5.2178
Epoch 8/50
 - 17s - loss: 0.0049 - mean_absolute_error: 0.0507 - mean_absolute_percentage_error: 5.0716 - val_loss: 0.0046 -
val_mean_absolute_error: 0.0497 - val_mean_absolute_percentage_error: 4.9689
Epoch 9/50
 - 17s - loss: 0.0046 - mean_absolute_error: 0.0485 - mean_absolute_percentage_error: 4.8483 - val_loss: 0.0043 -
val_mean_absolute_error: 0.0477 - val_mean_absolute_percentage_error: 4.7697
Epoch 10/50
 - 17s - loss: 0.0044 - mean_absolute_error: 0.0467 - mean_absolute_percentage_error: 4.6693 - val_loss: 0.0042 -
val_mean_absolute_error: 0.0461 - val_mean_absolute_percentage_error: 4.6084
Epoch 11/50
 - 17s - loss: 0.0043 - mean_absolute_error: 0.0452 - mean_absolute_percentage_error: 4.5241 - val_loss: 0.0041 -
val_mean_absolute_error: 0.0449 - val_mean_absolute_percentage_error: 4.4860
Epoch 12/50
 - 17s - loss: 0.0042 - mean_absolute_error: 0.0441 - mean_absolute_percentage_error: 4.4097 - val_loss: 0.0040 -
val_mean_absolute_error: 0.0437 - val_mean_absolute_percentage_error: 4.3685
Epoch 13/50
 - 17s - loss: 0.0041 - mean_absolute_error: 0.0431 - mean_absolute_percentage_error: 4.3131 - val_loss: 0.0039 -
val_mean_absolute_error: 0.0428 - val_mean_absolute_percentage_error: 4.2797
Epoch 14/50
 - 17s - loss: 0.0040 - mean_absolute_error: 0.0423 - mean_absolute_percentage_error: 4.2333 - val_loss: 0.0038 -
val_mean_absolute_error: 0.0420 - val_mean_absolute_percentage_error: 4.2015
Epoch 15/50
 - 17s - loss: 0.0039 - mean_absolute_error: 0.0416 - mean_absolute_percentage_error: 4.1637 - val_loss: 0.0038 -
val_mean_absolute_error: 0.0414 - val_mean_absolute_percentage_error: 4.1408
Epoch 16/50
 - 17s - loss: 0.0038 - mean_absolute_error: 0.0411 - mean_absolute_percentage_error: 4.1059 - val_loss: 0.0037 -
val_mean_absolute_error: 0.0409 - val_mean_absolute_percentage_error: 4.0880
Epoch 17/50
 - 17s - loss: 0.0038 - mean_absolute_error: 0.0405 - mean_absolute_percentage_error: 4.0549 - val_loss: 0.0036 -
val_mean_absolute_error: 0.0403 - val_mean_absolute_percentage_error: 4.0349
Epoch 18/50
 - 17s - loss: 0.0037 - mean_absolute_error: 0.0401 - mean_absolute_percentage_error: 4.0104 - val_loss: 0.0036 -
val_mean_absolute_error: 0.0398 - val_mean_absolute_percentage_error: 3.9815
Epoch 19/50
 - 17s - loss: 0.0037 - mean_absolute_error: 0.0397 - mean_absolute_percentage_error: 3.9656 - val_loss: 0.0035 -
val_mean_absolute_error: 0.0395 - val_mean_absolute_percentage_error: 3.9541
Epoch 20/50
 - 17s - loss: 0.0036 - mean_absolute_error: 0.0393 - mean_absolute_percentage_error: 3.9306 - val_loss: 0.0035 -
val_mean_absolute_error: 0.0391 - val_mean_absolute_percentage_error: 3.9106
Epoch 21/50
 - 17s - loss: 0.0036 - mean_absolute_error: 0.0389 - mean_absolute_percentage_error: 3.8941 - val_loss: 0.0034 -
val_mean_absolute_error: 0.0388 - val_mean_absolute_percentage_error: 3.8794
Epoch 22/50
```

```
 - 17s - loss: 0.0035 - mean_absolute_error: 0.0386 - mean_absolute_percentage_error: 3.8631 - val_loss: 0.0034 -
val_mean_absolute_error: 0.0384 - val_mean_absolute_percentage_error: 3.8418
Epoch 23/50
 - 17s - loss: 0.0035 - mean_absolute_error: 0.0383 - mean_absolute_percentage_error: 3.8301 - val_loss: 0.0033 -
val_mean_absolute_error: 0.0382 - val_mean_absolute_percentage_error: 3.8163
Epoch 24/50
 - 17s - loss: 0.0034 - mean_absolute_error: 0.0380 - mean_absolute_percentage_error: 3.8023 - val_loss: 0.0033 -
val_mean_absolute_error: 0.0378 - val_mean_absolute_percentage_error: 3.7840
Epoch 25/50
 - 17s - loss: 0.0034 - mean_absolute_error: 0.0377 - mean_absolute_percentage_error: 3.7743 - val_loss: 0.0032 -
val_mean_absolute_error: 0.0375 - val_mean_absolute_percentage_error: 3.7531
Epoch 26/50
 - 18s - loss: 0.0033 - mean_absolute_error: 0.0374 - mean_absolute_percentage_error: 3.7450 - val_loss: 0.0032 -
val_mean_absolute_error: 0.0373 - val_mean_absolute_percentage_error: 3.7344
Epoch 27/50
 - 17s - loss: 0.0033 - mean_absolute_error: 0.0372 - mean_absolute_percentage_error: 3.7223 - val_loss: 0.0032 -
val_mean_absolute_error: 0.0370 - val_mean_absolute_percentage_error: 3.6971
Epoch 28/50
 - 17s - loss: 0.0032 - mean_absolute_error: 0.0369 - mean_absolute_percentage_error: 3.6940 - val_loss: 0.0031 -
val_mean_absolute_error: 0.0368 - val_mean_absolute_percentage_error: 3.6817
Epoch 29/50
 - 17s - loss: 0.0032 - mean_absolute_error: 0.0367 - mean_absolute_percentage_error: 3.6718 - val_loss: 0.0031 -
val_mean_absolute_error: 0.0365 - val_mean_absolute_percentage_error: 3.6494
Epoch 30/50
 - 17s - loss: 0.0032 - mean_absolute_error: 0.0365 - mean_absolute_percentage_error: 3.6452 - val_loss: 0.0030 -
val_mean_absolute_error: 0.0363 - val_mean_absolute_percentage_error: 3.6326
Epoch 31/50
 - 17s - loss: 0.0031 - mean_absolute_error: 0.0362 - mean_absolute_percentage_error: 3.6238 - val_loss: 0.0030 -
val_mean_absolute_error: 0.0360 - val_mean_absolute_percentage_error: 3.6038
Epoch 32/50
 - 17s - loss: 0.0031 - mean_absolute_error: 0.0360 - mean_absolute_percentage_error: 3.6009 - val_loss: 0.0030 -
val_mean_absolute_error: 0.0357 - val_mean_absolute_percentage_error: 3.5746
Epoch 33/50
 - 17s - loss: 0.0030 - mean_absolute_error: 0.0358 - mean_absolute_percentage_error: 3.5776 - val_loss: 0.0029 -
val_mean_absolute_error: 0.0355 - val_mean_absolute_percentage_error: 3.5541
Epoch 34/50
 - 17s - loss: 0.0030 - mean_absolute_error: 0.0355 - mean_absolute_percentage_error: 3.5541 - val_loss: 0.0029 -
val_mean_absolute_error: 0.0354 - val_mean_absolute_percentage_error: 3.5378
Epoch 35/50
 - 17s - loss: 0.0030 - mean_absolute_error: 0.0353 - mean_absolute_percentage_error: 3.5348 - val_loss: 0.0028 -
val_mean_absolute_error: 0.0351 - val_mean_absolute_percentage_error: 3.5118
Epoch 36/50
 - 17s - loss: 0.0029 - mean_absolute_error: 0.0351 - mean_absolute_percentage_error: 3.5125 - val_loss: 0.0028 -
val_mean_absolute_error: 0.0349 - val_mean_absolute_percentage_error: 3.4905
Epoch 37/50
```

```
 - 17s - loss: 0.0029 - mean_absolute_error: 0.0349 - mean_absolute_percentage_error: 3.4920 - val_loss: 0.0028 -
val_mean_absolute_error: 0.0347 - val_mean_absolute_percentage_error: 3.4697
Epoch 38/50
 - 17s - loss: 0.0029 - mean_absolute_error: 0.0347 - mean_absolute_percentage_error: 3.4717 - val_loss: 0.0027 -
val_mean_absolute_error: 0.0345 - val_mean_absolute_percentage_error: 3.4499
Epoch 39/50
 - 17s - loss: 0.0028 - mean_absolute_error: 0.0345 - mean_absolute_percentage_error: 3.4524 - val_loss: 0.0027 -
val_mean_absolute_error: 0.0342 - val_mean_absolute_percentage_error: 3.4240
Epoch 40/50
 - 18s - loss: 0.0028 - mean_absolute_error: 0.0343 - mean_absolute_percentage_error: 3.4311 - val_loss: 0.0027 -
val_mean_absolute_error: 0.0341 - val_mean_absolute_percentage_error: 3.4093
Epoch 41/50
 - 17s - loss: 0.0028 - mean_absolute_error: 0.0341 - mean_absolute_percentage_error: 3.4143 - val_loss: 0.0026 -
val_mean_absolute_error: 0.0338 - val_mean_absolute_percentage_error: 3.3797
Epoch 42/50
 - 17s - loss: 0.0027 - mean_absolute_error: 0.0339 - mean_absolute_percentage_error: 3.3926 - val_loss: 0.0026 -
val_mean_absolute_error: 0.0337 - val_mean_absolute_percentage_error: 3.3695
Epoch 43/50
 - 17s - loss: 0.0027 - mean_absolute_error: 0.0338 - mean_absolute_percentage_error: 3.3756 - val_loss: 0.0026 -
val_mean_absolute_error: 0.0335 - val_mean_absolute_percentage_error: 3.3499
Epoch 44/50
 - 18s - loss: 0.0027 - mean_absolute_error: 0.0336 - mean_absolute_percentage_error: 3.3594 - val_loss: 0.0026 -
val_mean_absolute_error: 0.0332 - val_mean_absolute_percentage_error: 3.3230
Epoch 45/50
 - 17s - loss: 0.0026 - mean_absolute_error: 0.0334 - mean_absolute_percentage_error: 3.3383 - val_loss: 0.0025 -
val_mean_absolute_error: 0.0332 - val_mean_absolute_percentage_error: 3.3165
Epoch 46/50
 - 17s - loss: 0.0026 - mean_absolute_error: 0.0332 - mean_absolute_percentage_error: 3.3249 - val_loss: 0.0025 -
val_mean_absolute_error: 0.0328 - val_mean_absolute_percentage_error: 3.2849
Epoch 47/50
 - 17s - loss: 0.0026 - mean_absolute_error: 0.0331 - mean_absolute_percentage_error: 3.3052 - val_loss: 0.0025 -
val_mean_absolute_error: 0.0327 - val_mean_absolute_percentage_error: 3.2716
Epoch 48/50
 - 17s - loss: 0.0026 - mean_absolute_error: 0.0329 - mean_absolute_percentage_error: 3.2881 - val_loss: 0.0024 -
val_mean_absolute_error: 0.0326 - val_mean_absolute_percentage_error: 3.2624
Epoch 49/50
 - 18s - loss: 0.0025 - mean_absolute_error: 0.0327 - mean_absolute_percentage_error: 3.2742 - val_loss: 0.0024 -
val_mean_absolute_error: 0.0324 - val_mean_absolute_percentage_error: 3.2411
Epoch 50/50
 - 17s - loss: 0.0025 - mean_absolute_error: 0.0326 - mean_absolute_percentage_error: 3.2569 - val_loss: 0.0024 -
val_mean_absolute_error: 0.0323 - val_mean_absolute_percentage_error: 3.2306
```

In [0]:
```
# make predictions
trainPredict = model.predict(trainX)
```

```
testPredict = model.predict(testX)
```

In [289]:
```
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 4)                 368
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 5
=================================================================
Total params: 373
Trainable params: 373
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty):
    fig = plt.figure( facecolor='c', edgecolor='k')
    plt.plot(x, vy, 'b', label="Validation Loss")
    plt.plot(x, ty, 'r', label="Train Loss")
    plt.xlabel('Epochs')
    plt.ylabel('loss')
    plt.legend()
    plt.grid()
    plt.show()
```

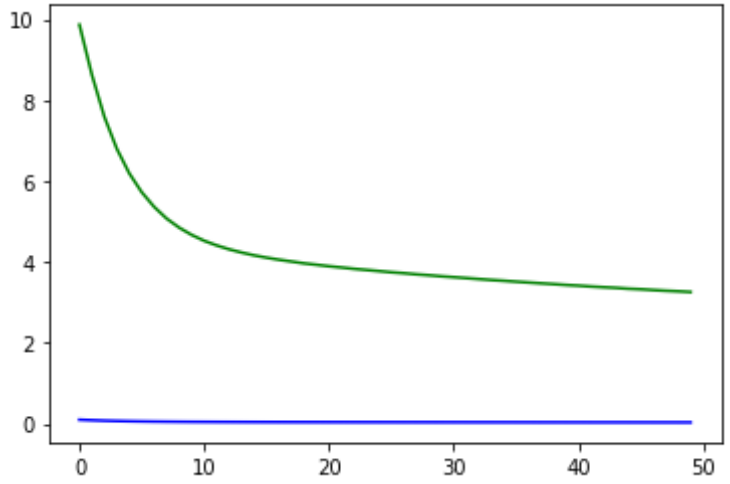
In [291]:
```
score=model.evaluate(testX, testY)
print(score)
```

```
98231/98231 [==============================] - 8s 81us/step
[0.002606318959066143, 0.03398444081203853, 3.3984440844216097]
```

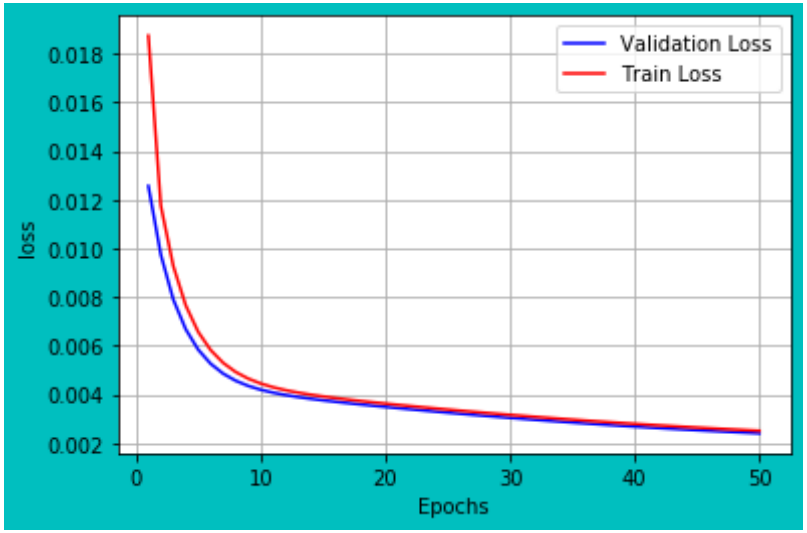
In [292]:
```
test_score=score[0]
test_accuracy_mae=score[1]
```

```
test_accuracy_mape=score[2]
train_accuracy_mape=min(historyy.history['mean_absolute_percentage_error'])
train_accuracy_mae=min(historyy.history['mean_absolute_error'])
print('test score :',test_score)
print('test Accuracy mape :',test_accuracy_mape)
print('test Accuracy mae :',test_accuracy_mae)
print('train accuracy mape:',train_accuracy_mape)
print('train accuracy mae:',train_accuracy_mae)
plt.plot(historyy.history['mean_absolute_error'],'b', label='mean_absolute_error')
plt.plot(historyy.history['mean_absolute_percentage_error'],'g', label='mean_absolute_percentage_error')
plt.show()
```

```
test score : 0.002606318959066143
test Accuracy mape : 3.3984440844216097
test Accuracy mae : 0.03398444081203853
train accuracy mape: 3.256865320677273
train accuracy mae: 0.03256865328409592
```



In [293]:
```
# error plot
x=list(range(1,51))
vy=historyy.history['val_loss'] #validation loss
ty=historyy.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```



In [294]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------------------------------")
print ("Baseline Model -                                Train: ",train_mape[0],"      Test: ",test_mape[0])
print ("Exponential Averages Forecasting -              Train: ",train_mape[1],"      Test: ",test_mape[1])
print ("Linear Regression -                            Train: ",train_mape[3],"      Test: ",test_mape[3])
print ("Random Forest Regression -                      Train: ",train_mape[2],"     Test: ",test_mape[2])
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------
Baseline Model -                                Train:  0.2384932845253011       Test:  0.22768601861332272
Exponential Averages Forecasting -              Train:  0.1383789404255362       Test:  0.13146751837645368
Linear Regression -                            Train:  0.13157942214095766       Test:  0.12833962437153293
Random Forest Regression -                      Train:  0.09865299423203268      Test:  0.12939758852025612
```

In [295]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------------------------------")
print ("Baseline Model -                                Train: ",train_mape[0],"      Test: ",test_mape[0])
print ("Exponential Averages Forecasting -              Train: ",train_mape[1],"      Test: ",test_mape[1])
print ("Linear Regression -                            Train: ",train_mape[4],"      Test: ",test_mape[4])
print ("Random Forest Regression -                      Train: ",train_mape[2],"     Test: ",test_mape[2])
print ("XgBoost Regression -                            Train: ",train_mape[3],"      Test: ",test_mape[3])
print ("LSTM -                                    Train: ",train_accuracy_mae,"    Test: ",test_accuracy_mae)
print ("--------------------------------------------------------------------------------------------------------")
```

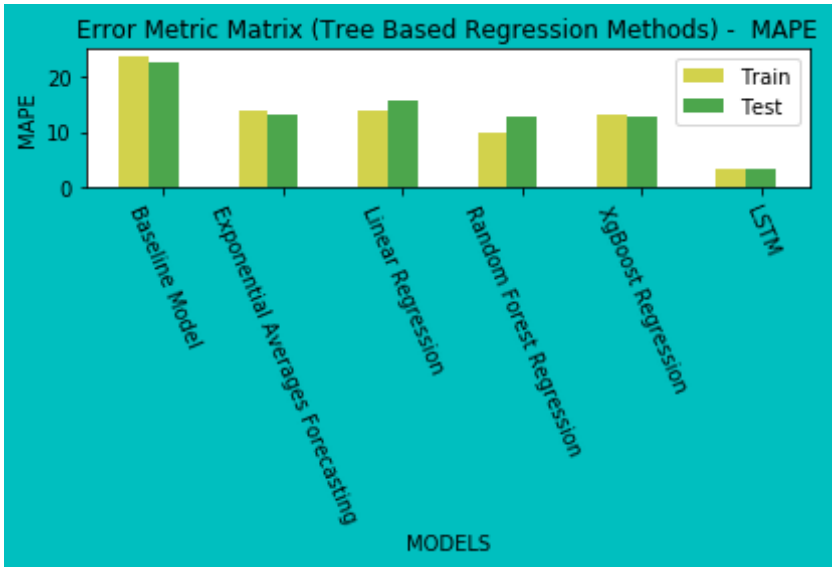
```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------
```

```
Baseline Model -                              Train:  0.2384932845253011       Test:  0.22768601861332272
Exponential Averages Forecasting -            Train:  0.1383789404255362       Test:  0.13146751837645368
Linear Regression -                          Train:  0.13849699364888227       Test:  0.1552765255130889
Random Forest Regression -                    Train:  0.09865299423203268      Test:  0.12939758852025612
XgBoost Regression -                          Train:  0.13157942214095766       Test:  0.12833962437153293
LSTM -                                 Train:  0.03256865328409592     Test:  0.03398444081203853
------------------------------------------------------------------------------------------------------
```

Here, LSTM model for taxi predictions is going to underfitting.

In [296]:
```
n_gro=6

train_mape=(train_mape[0]*100,train_mape[1]*100,train_mape[4]*100,
            train_mape[2]*100,train_mape[3]*100,train_accuracy_mae*100)
test_mape=(test_mape[0]*100,test_mape[1]*100,test_mape[4]*100,
          test_mape[2]*100,test_mape[3]*100,test_accuracy_mae*100)
fig4 = plt.figure( facecolor='c', edgecolor='k')
index = np.arange(n_gro)
bar_width = 0.25
opacity = 0.7

result1 = plt.bar(index, train_mape, bar_width,
alpha=opacity,
color='y',
label='Train')

result2 = plt.bar(index + bar_width, test_mape, bar_width,
alpha=opacity,
color='g',
label='Test')

plt.xlabel('MODELS')
plt.ylabel('MAPE')
plt.title('Error Metric Matrix (Tree Based Regression Methods) -  MAPE')
plt.xticks(index + bar_width, ('Baseline Model', 'Exponential Averages Forecasting',
                               'Linear Regression ',

                               'Random Forest Regression',
                              'XgBoost Regression',
                              'LSTM'),rotation=-67)
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



Observation

From the above result we have following observations:

1. The difference between train error and test error of random forest regressor is high, which clearly shows that random forest regressor is overfitting. Therefore, we are discarding random forest regressor.
2. Also in LSTM regression the values train and test MAPE value is verylow therefore it showing us underfitting nature.
3. The best modelamong all models with lowest train and test MAPE is XGBoost Regressor.having Train MAPE - 0.1333% and Test MAPE - 0.1285%