

Taxi demand prediction in New York City



In [1]:

```
#Importing Libraries
# pip3 install graphviz
#pip3 install dask
#pip3 install toolz
#pip3 install cloudpickle
# https://www.youtube.com/watch?v=ieW3G7ZzRZ0
# https://github.com/dask/dask-tutorial
# please do go through this python notebook: https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
import dask.dataframe as dd#similar to pandas

import pandas as pd#pandas to create small dataframes

# pip3 install folium
# if this doesnt work refere install_folium.JPG in drive
import folium #open street map

# unix time: https://www.unixtimestamp.com/
import datetime #Convert to unix time

import time #Convert to unix time

# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays

# matplotlib: used to plot graphs
import matplotlib
# matplotlib.use('nbagg') : matplotlib uses this protocall which makes plots more user
# interactive like zoom in and zoom out
matplotlib.use('nbagg')
import matplotlib.pyplot as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots

# this lib is used while we calculate the stight line distance between two (Lat,Lon) pa
#irs in miles
import gpxpy.geo #Get the haversine distance

from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os

# download mingw: https://mingw-w64.org/doku.php/download/mingw-builds
# install it in your system and keep the path, mingw_path ='installed path'
#mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mingw64
\\bin'
#os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

# to install xgboost: pip3 install xgboost
# if it didnt happen check install_xgboost.JPG
import xgboost as xgb

# to install sklearn: pip install -U scikit-learn
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
```

```
%matplotlib notebook  
%matplotlib inline
```

Data Information

Get the data from : http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml (2016 data) The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC)

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 (FHV)s

FHV transportation is accessed by a pre-arrangement with a dispatcher or limo company. These FHV's 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.

Credits: Quora

Footnote:

In the given notebook we are considering only the yellow taxis for the time period between Jan - Mar 2015 & Jan - Mar 2016

Data Collection

We Have collected all yellow taxi trips data from jan-2015 to dec-2016(Will be using only 2015 data)

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	1. 62G	11135470	19
yellow_tripdata_2016-07	884Mb	10294080	17
yellow_tripdata_2016-08	854Mb	9942263	17
yellow_tripdata_2016-09	870Mb	10116018	17
yellow_tripdata_2016-10	933Mb	10854626	17
yellow_tripdata_2016-11	868Mb	10102128	17
yellow_tripdata_2016-12	897Mb	10449408	17
yellow_tripdata_2015-01	1.84Gb	12748986	19
yellow_tripdata_2015-02	1.81Gb	12450521	19
yellow_tripdata_2015-03	1.94Gb	13351609	19
yellow_tripdata_2015-04	1.90Gb	13071789	19
yellow_tripdata_2015-05	1.91Gb	13158262	19
yellow_tripdata_2015-06	1.79Gb	12324935	19
yellow_tripdata_2015-07	1.68Gb	11562783	19
yellow_tripdata_2015-08	1.62Gb	11130304	19
yellow_tripdata_2015-09	1.63Gb	11225063	19
yellow_tripdata_2015-10	1.79Gb	12315488	19
yellow_tripdata_2015-11	1.65Gb	11312676	19
yellow_tripdata_2015-12	1.67Gb	11460573	19

In [2]:

```
#Looking at the features
# dask dataframe : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('yellow_tripdata_2015-01.csv')
print(month.columns)

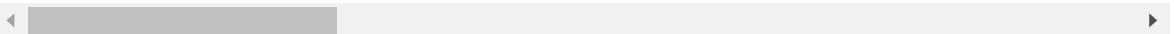
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [3]:

month.head()

Out[3]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
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



In [4]:

```
# However unlike Pandas, operations on dask.dataframes don't trigger immediate computation,
# instead they add key-value pairs to an underlying Dask graph. Recall that in the diagram below,
# circles are operations and rectangles are results.

# to see the visualization you need to install graphviz
# pip3 install graphviz if this doesn't work please check the install_graphviz.jpg in the drive
month.visualize()
```

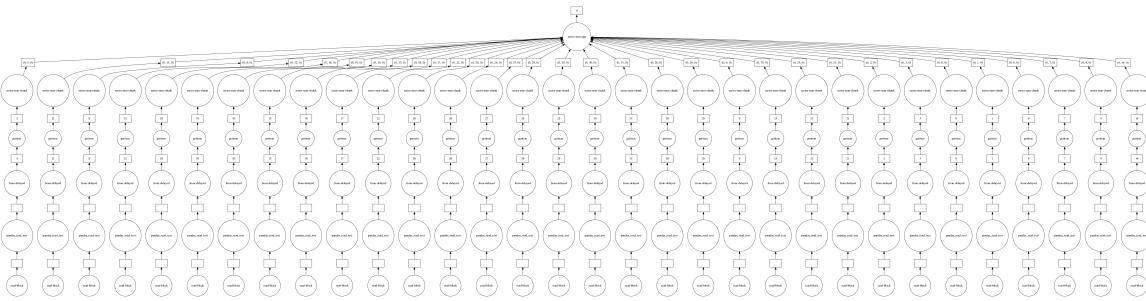
Out[4]:



In [5]:

```
month.fare_amount.sum().visualize()
```

Out[5]:



Features in the dataset:

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. 1. Standard rate 2. JFK 3. Newark 4. Nassau or Westchester 5. Negotiated fare 6. Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip
Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. 1. Credit card 2. Cash 3. No charge 4. Dispute 5. Unknown 6. Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the 0.50 and 1 rush hour and overnight charges.
MTA_tax	0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

ML Problem Formulation

Time-series forecasting and Regression

- To find number of pickups, given location coordinates(latitude and longitude) and time, in the query region 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.

EDA & 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 [6]:

```
#table below shows few datapoints along with all our features
month.head(5)
```

Out[6]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
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> (<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 coordinates not within these coordinates are not considered by us as we are only concerned with pickups which originate within New York.

In [7]:

```
# Plotting pickup coordinates 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.pickup_longitude <= -74.15) | (month.pickup_latitude
<= 40.5774) | \
(month.pickup_longitude >= -73.7004) | (month.pickup_latitude >= 40.
9176))]

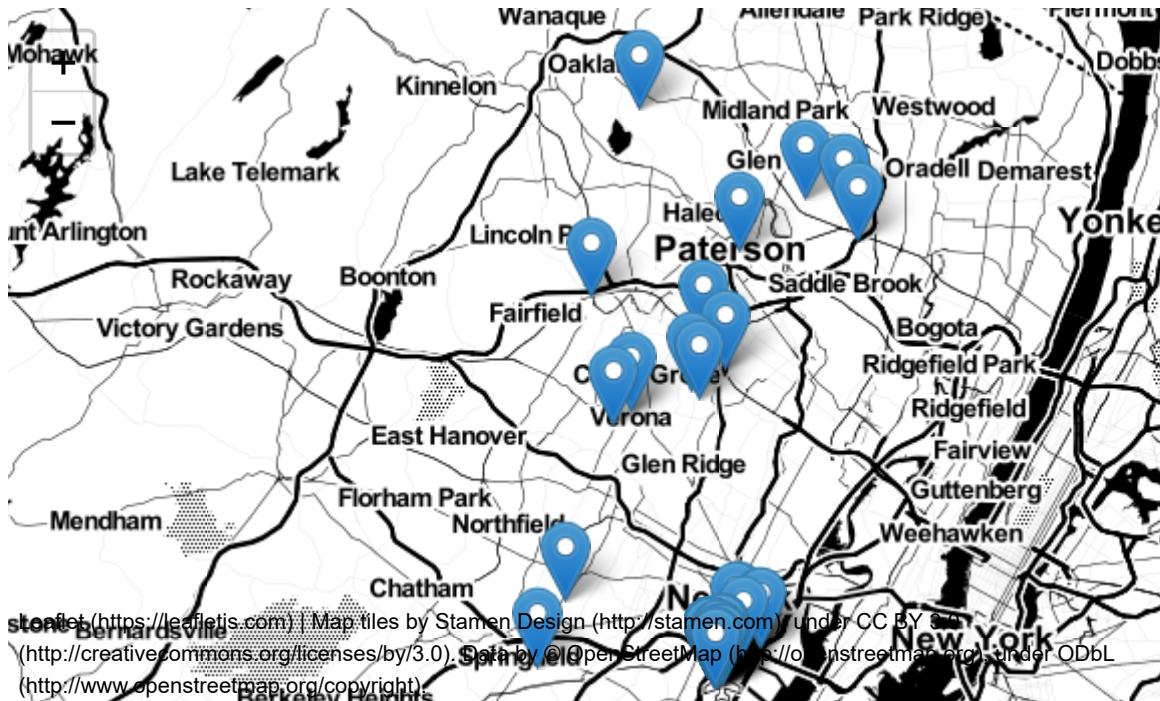
# creating a map with the a base location
# read more about the folium here: http://folium.readthedocs.io/en/latest/quickstart.ht
ml

# note: you dont need to remember any of these, you dont need indepth knowledge on thes
e 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['pickup_latitude'],j['pickup_longitude']))).add_to(map_os
m)
map_osm
```

Out[7]:



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> (<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 coordinates not within these coordinates are not considered by us as we are only concerned with dropoffs which are within New York.

In [8]:

```

# Plotting dropoff coordinates 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

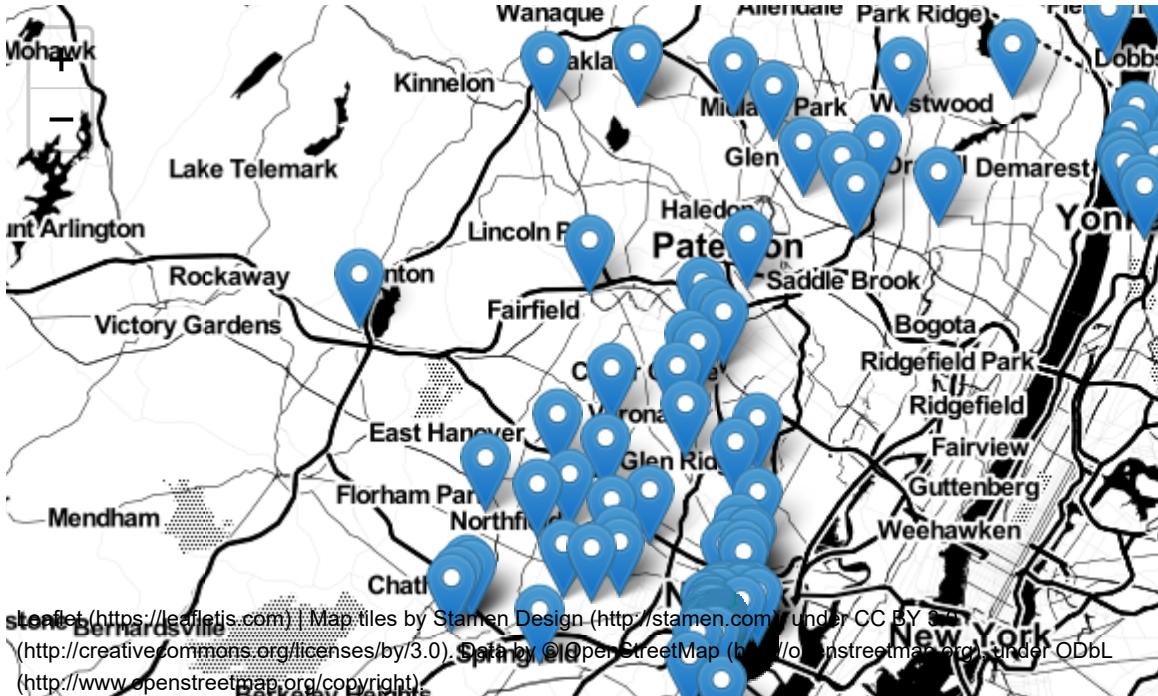
# note: you dont need to remember any of these, you dont need indeepth knowledge on these maps and plots

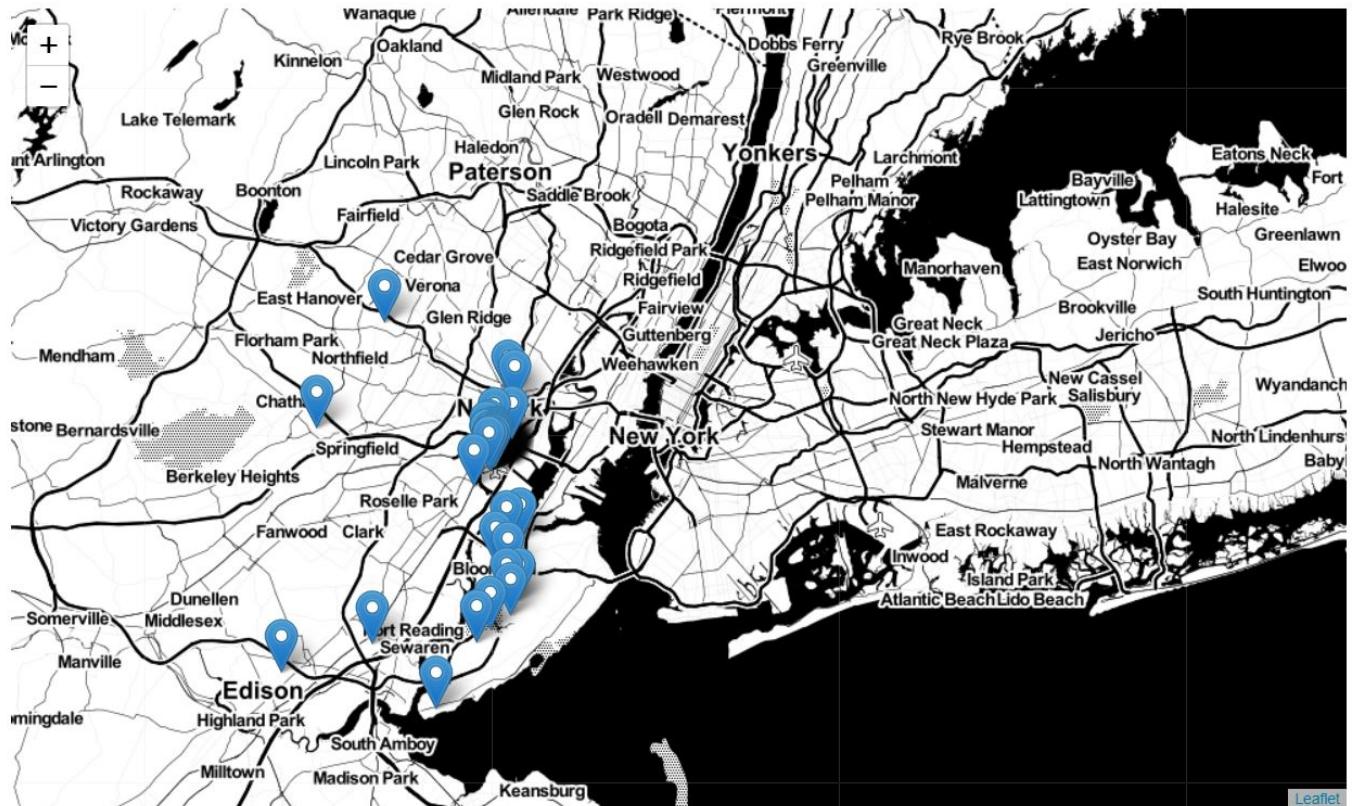
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')

# 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[8]:





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

3. Trip Durations:

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

In [9]:

```
print(month.columns)

Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amou
nt',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [10]:

```
#The timestamps are converted to unix so as to get duration(trip-time) & speed. Also pickup-times in unix are used while binning

# in out data we have time in the format "YYYY-MM-DD HH:MM:SS" we convert this string to
# python time format and then into unix time stamp
# https://stackoverflow.com/a/27914405
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetuple())

# we return a data frame which contains the columns
# 1. 'passenger_count' : self explanatory
# 2. 'trip_distance' : self explanatory
# 3. 'pickup_longitude' : self explanatory
# 4. 'pickup_latitude' : self explanatory
# 5. 'dropoff_longitude' : self explanatory
# 6. 'dropoff_latitude' : self explanatory
# 7. 'total_amount' : total fair that was paid
# 8. 'trip_times' : duration of each trip
# 9. 'pickup_times' : pickup time converted into unix time
# 10. 'Speed' : velocity of each trip
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

# print(frame_with_durations.head())
# passenger_count      trip_distance      pickup_longitude      pickup_latitude dropoff_longitude
# _longitude      dropoff_latitude      total_amount      trip_times      pickup_times
# Speed
# 1           1.59           -73.993896           40.750111           -73.974
785          40.750618           17.05           18.050000           1.421329e+09
5.285319
# 1           3.30           -74.001648           40.724243           -73.994
415          40.759109           17.80           19.833333           1.420902e+09
9.983193
# 1           1.80           -73.963341           40.802788           -73.951
820          40.824413           10.80           10.050000           1.420902e+09
10.746269
# 1           0.50           -74.009087           40.713818           -74.004
326          40.719986           4.80           1.866667           1.420902e+09
16.071429
# 1           3.00           -73.971176           40.762428           -74.004
```

```
181          40.742653      16.30      19.316667  1.420902e+09
9.318378
frame_with_durations = return_with_trip_times(month)
```

In [11]:

```
type(frame_with_durations)
```

Out[11]:

```
pandas.core.frame.DataFrame
```

In [13]:

```
frame_with_durations.head()
```

Out[13]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude
0	1	1.59	-73.993896	40.750111	-73.974785
1	1	3.30	-74.001648	40.724243	-73.994415
2	1	1.80	-73.963341	40.802788	-73.951820
3	1	0.50	-74.009087	40.713818	-74.004326
4	1	3.00	-73.971176	40.762428	-74.004181

◀ ▶

In [14]:

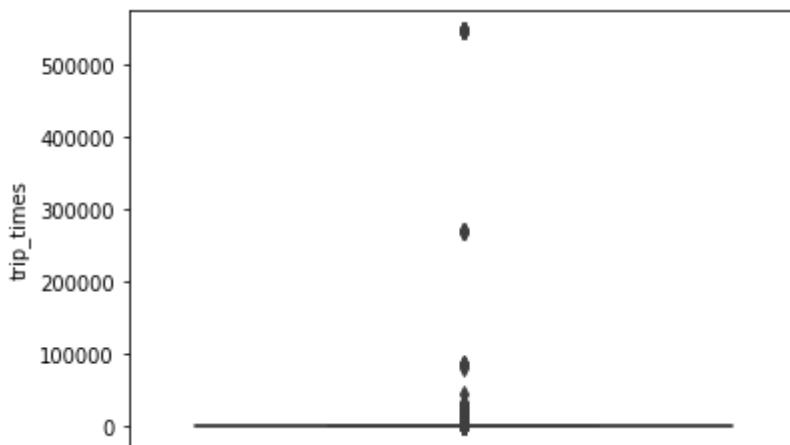
```
frame_with_durations.shape
```

Out[14]:

```
(12748986, 10)
```

In [16]:

```
# the skewed box plot shows us the presence of outliers
%matplotlib notebook
%matplotlib inline
sns.boxplot(y="trip_times", data =frame_with_durations)
plt.show()
```



In [17]:

```
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
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 -1211.0166666666666667
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 14.28333333333333
80 percentile value is 17.63333333333333
90 percentile value is 23.45
100 percentile value is 548555.6333333333
```

In [18]:

```
#Looking further from the 99th percentile
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])
```

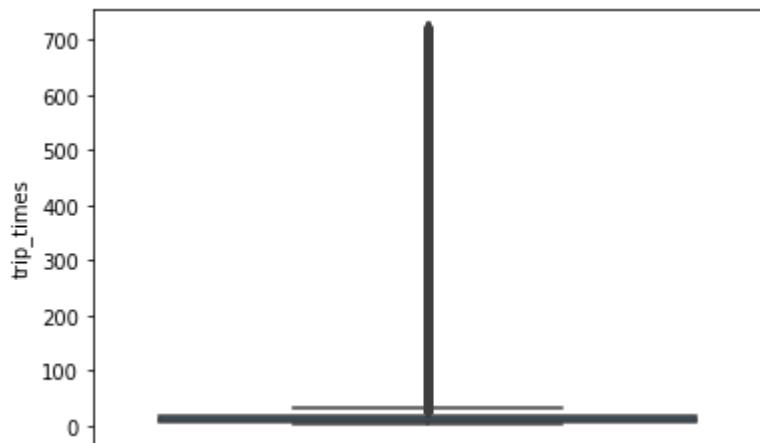
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.38333333333333
93 percentile value is 26.55
94 percentile value is 27.93333333333334
95 percentile value is 29.58333333333332
96 percentile value is 31.68333333333334
97 percentile value is 34.46666666666667
98 percentile value is 38.71666666666667
99 percentile value is 46.75
100 percentile value is 548555.6333333333
```

In [19]:

```
#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)]
```

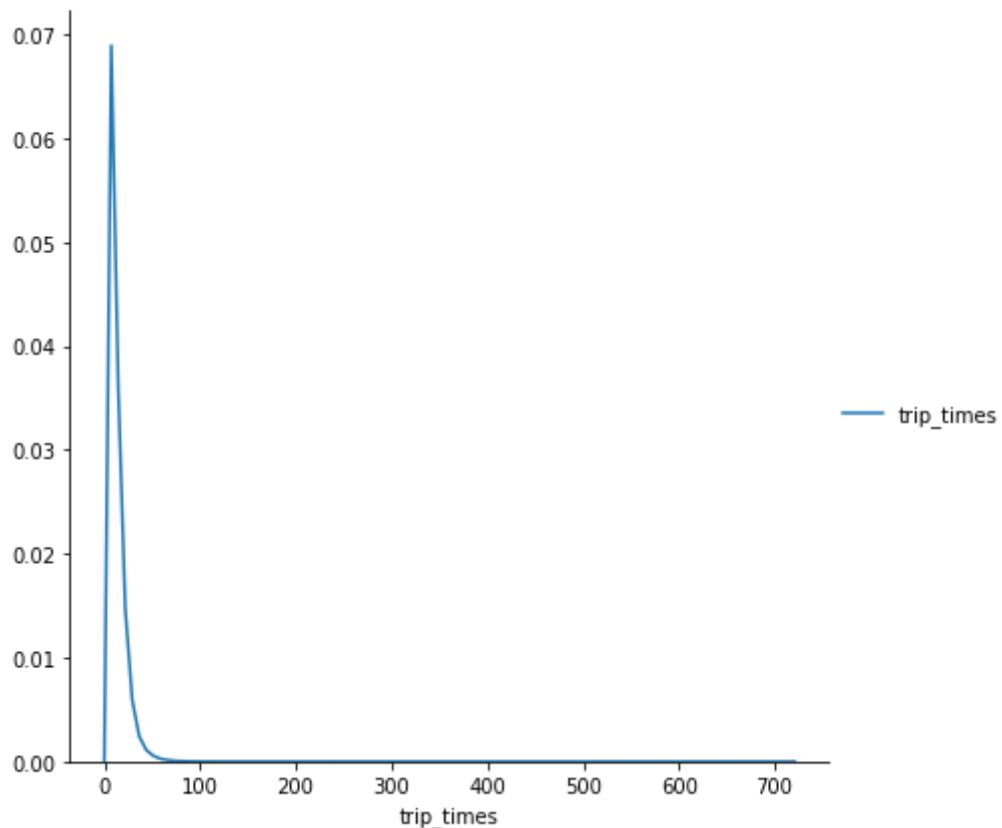
In [20]:

```
#box-plot after removal of outliers
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.show()
```



In [21]:

```
#pdf of trip-times after removing the outliers
sns.FacetGrid(frame_with_durations_modified, size=6) \
    .map(sns.kdeplot, "trip_times") \
    .add_legend();
plt.show();
```

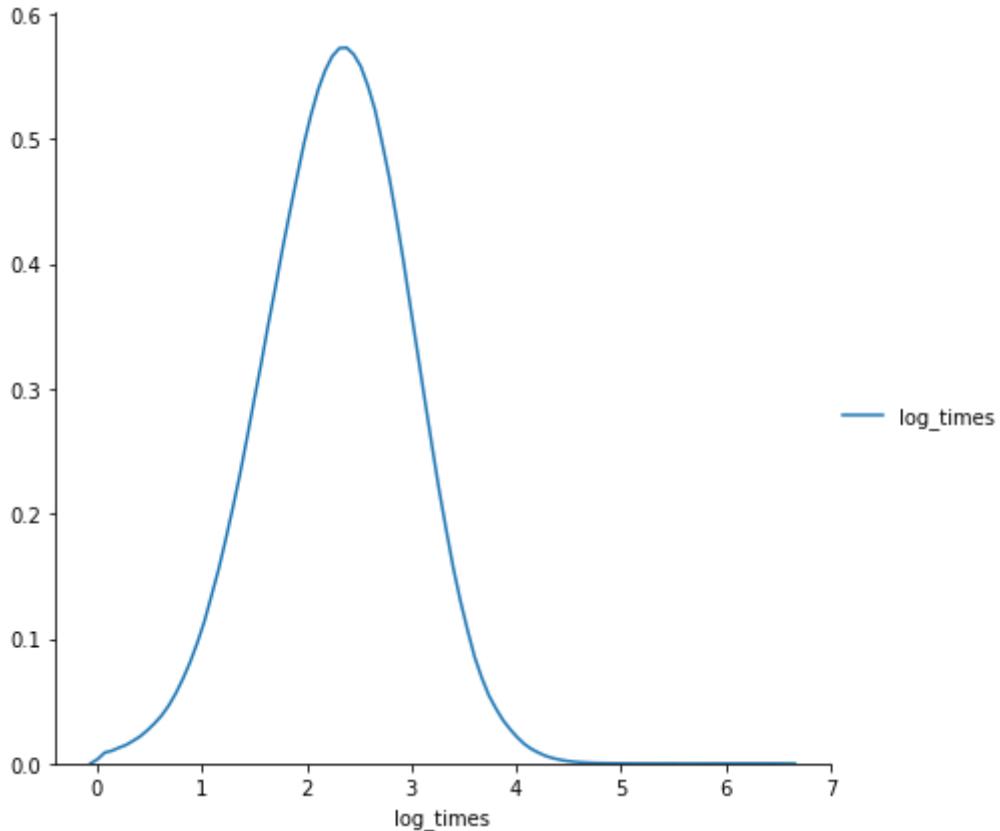


In [22]:

```
#converting the values to log-values to check for Log-normal
import math
frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_durations_m
odified['trip_times'].values]
```

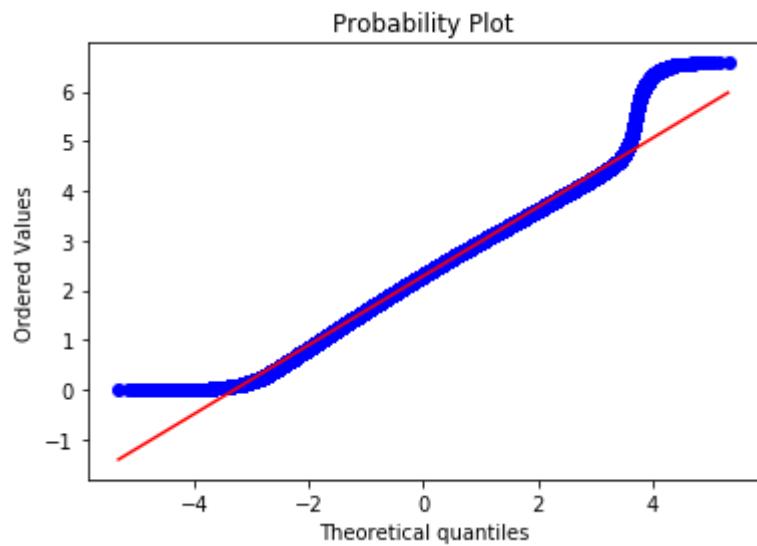
In [23]:

```
#pdf of Log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
    .map(sns.kdeplot,"log_times") \
    .add_legend();
plt.show();
```



In [25]:

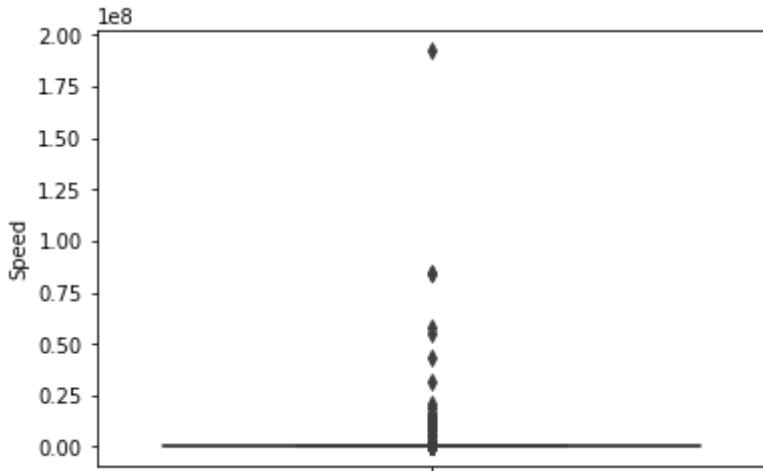
```
#Q-Q plot for checking if trip-times is Log-normal
import scipy
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [26]:

```
# 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_modified['trip_times'])
sns.boxplot(y="Speed", data =frame_with_durations_modified)
plt.show()
```



In [27]:

```
#calculating speed values at each percentile 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 [0]:

```
#calculating speed values at each percentile 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 192857142.857
```

In [28]:

```
#calculating speed values at each percentile 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 [29]:

```
#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 [30]:

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

Out[30]:

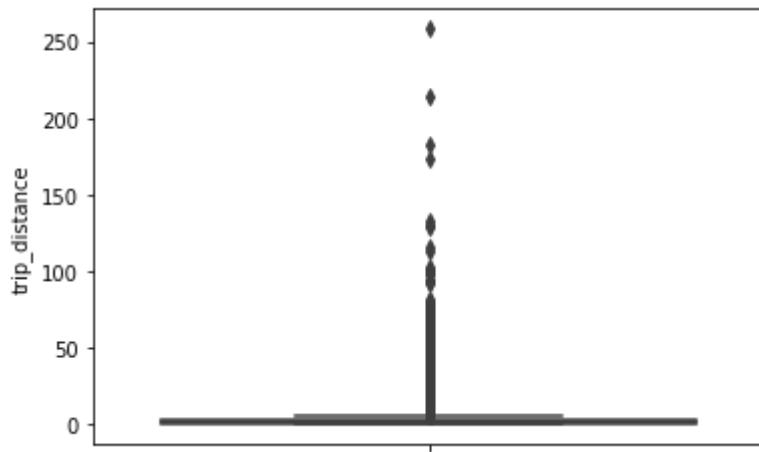
12.450173996027528

The avg speed in New York is 12.45 miles/hr, so a cab driver can travel 2 miles per 10min on avg.

4. Trip Distance

In [31]:

```
# 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 [32]:

```
#calculating trip distance values at each percentile 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 [33]:

```
#calculating trip distance values at each percentile 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 [34]:

```
#calculating trip distance values at each percentile 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])
```

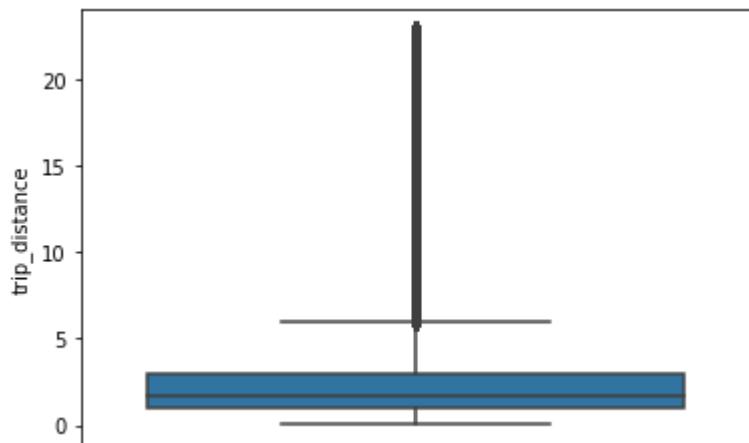
```
99.0 percentile value is 18.17
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.83
99.4 percentile value is 19.13
99.5 percentile value is 19.5
99.6 percentile value is 19.96
99.7 percentile value is 20.5
99.8 percentile value is 21.22
99.9 percentile value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

In [35]:

```
#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 [36]:

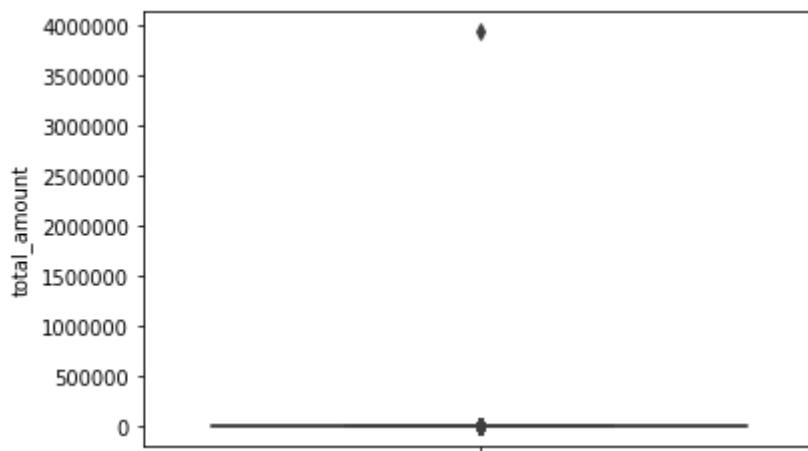
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [37]:

```
# up to now we have removed the outliers based on trip durations, cab speeds, and trip
# distances
# 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 [38]:

```
#calculating total fare amount values at each percentile 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 [39]:

```
#calculating total fare amount values at each percentile 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 [40]:

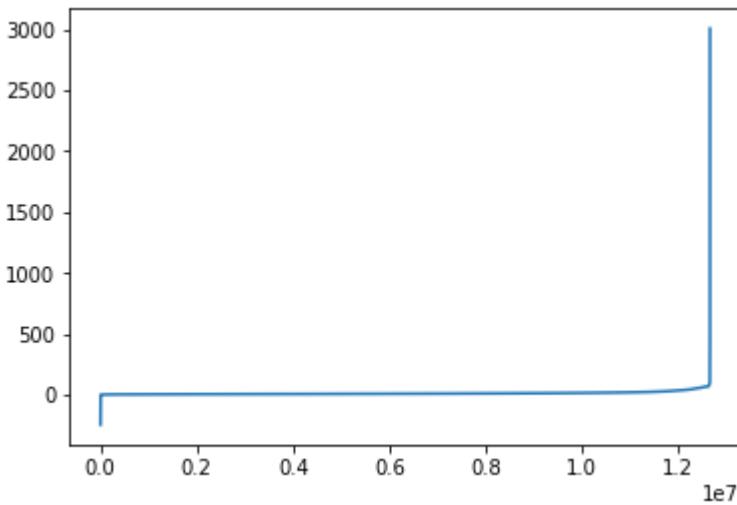
```
#calculating total fare amount values at each percentile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var, axis = None)
    print("{} percentile value is {}".format(99+i, var[int(len(var))*(float(99+i)/100)])))
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:- As even the 99.9th percentile value doesn't look like an outlier, as there is not much difference between the 99.8th percentile and 99.9th percentile, we move on to do graphical analysis

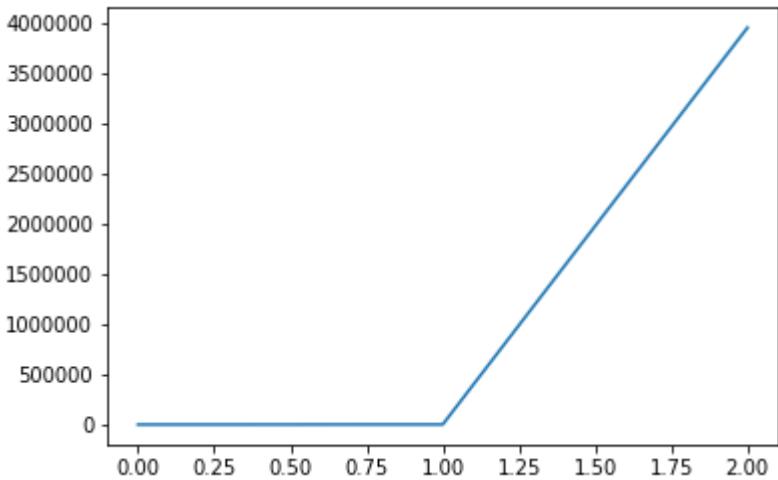
In [41]:

```
#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()
```



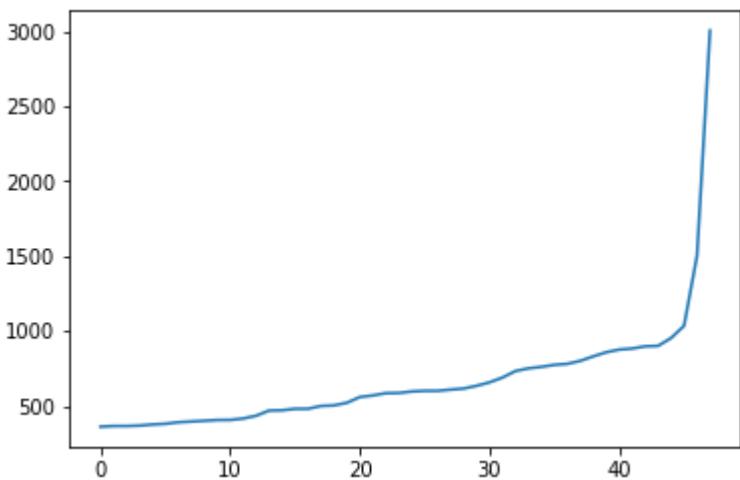
In [42]:

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



In [43]:

```
# now looking at values not including the last two points we again find a drastic increase
# at around 1000 fare value
# we plot last 50 values excluding last two values
plt.plot(var[-50:-2])
plt.show()
```



Remove all outliers/erroneous points.

In [44]:

```
#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 [45]:

```
%time
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 0.970357642560
7495
CPU times: user 4.52 s, sys: 3.39 s, total: 7.91 s
Wall time: 7.88 s
```

Data-preperation

1.0 Clustering/Segmentation

In [46]:

```

#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 = gpixpy.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,"Avg. Number of Clusters with
    in the vicinity (i.e. intercluster-distance < 2):", np.ceil(sum(less2)/len(less2)), "\n
    Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):", np.cei
    l(sum(more2)/len(more2)), "\nMin inter-cluster distance = ",min_dist,"---")

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fi
    t(coords)
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_
    _durations_outliers_removed[['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 region
#s
#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
```

```
---
```

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 [47]:

```
# if check for the 50 clusters you can observe that there are two clusters with only 0.3 miles apart from each other
# so we choose 40 clusters for solve the further problem

# Getting 40 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[['pickup_latitude', 'pickup_longitude']])
```

In [48]:

```
frame_with_durations_outliers_removed.head()
```

Out[48]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude
0	1	1.59	-73.993896	40.750111	-73.974785
1	1	3.30	-74.001648	40.724243	-73.994415
2	1	1.80	-73.963341	40.802788	-73.951820
3	1	0.50	-74.009087	40.713818	-74.004326
4	1	3.00	-73.971176	40.762428	-74.004181

In [49]:

```
frame_with_durations_outliers_removed.shape
```

Out[49]:

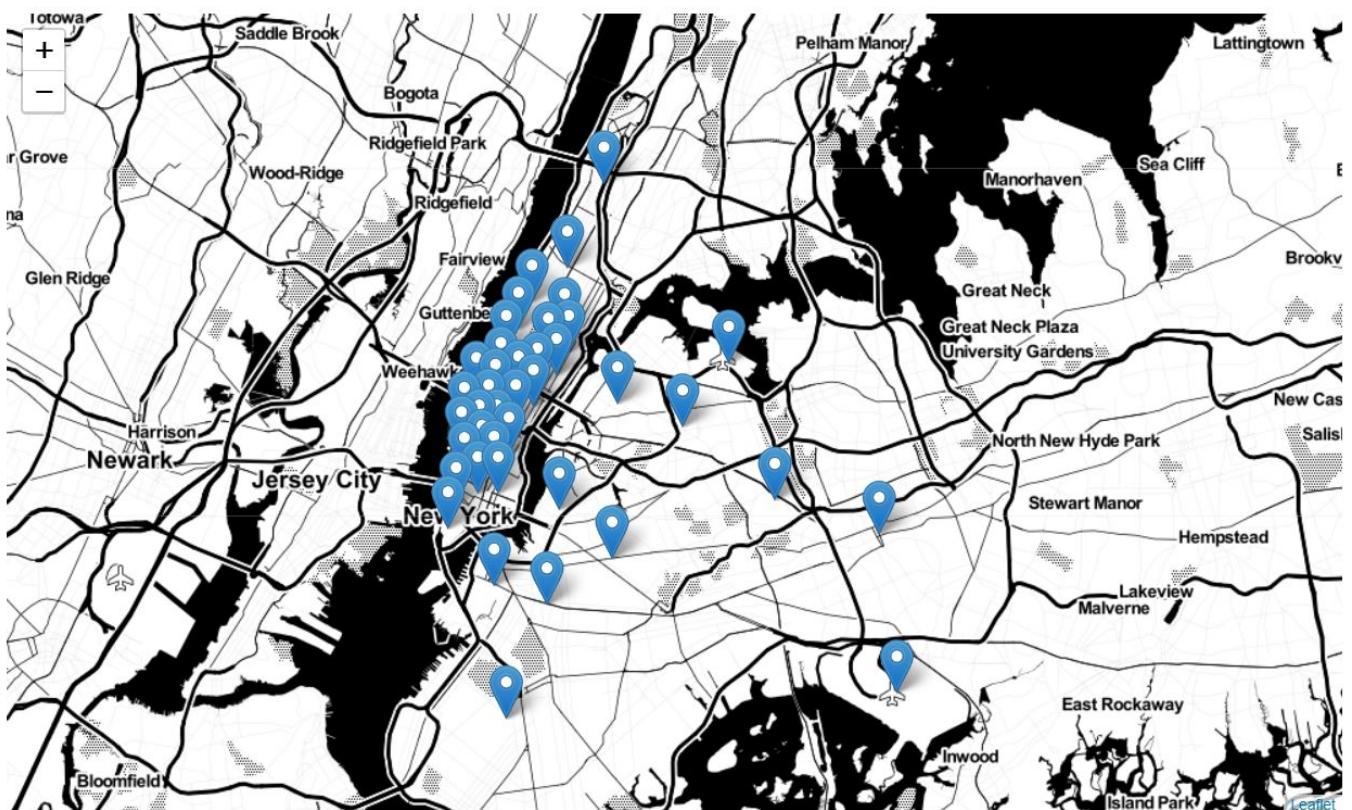
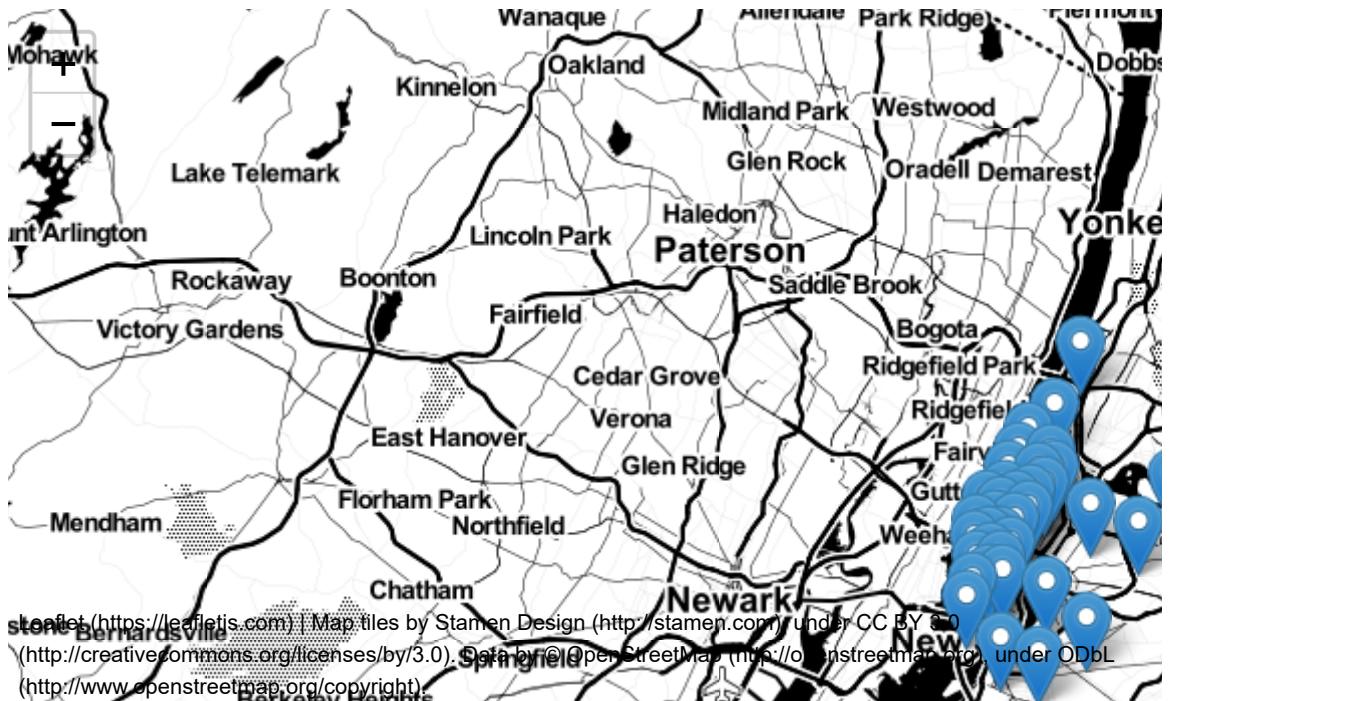
```
(12371076, 11)
```

Plotting the cluster centers:

In [50]:

```
# 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(cluster_centers[i][1]))).add_to(map_osm)
map_osm
```

Out[50]:

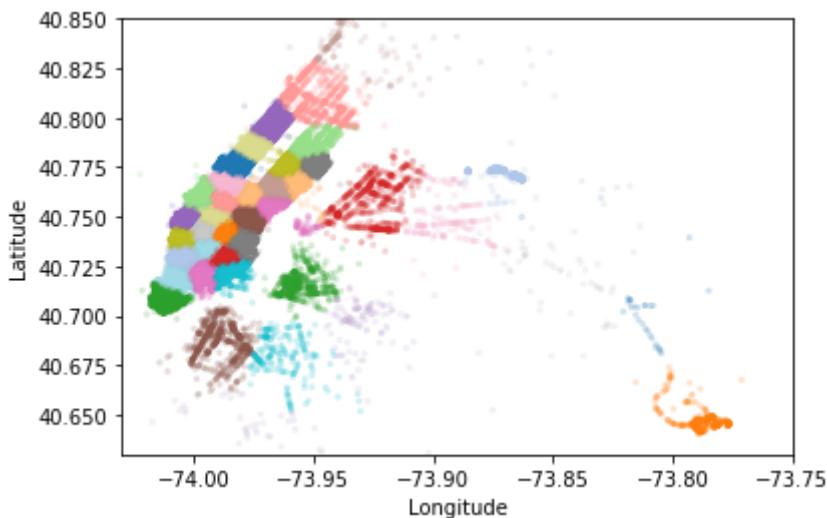


Plotting the clusters:

In [51]:

```
#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)
```



2.0 Time-binning

In [52]:

```
#Refer:https://www.unixtimestamp.com/
# 1420070400 : 2015-01-01 00:00:00
# 1422748800 : 2015-02-01 00:00:00
# 1425168000 : 2015-03-01 00:00:00
# 1427846400 : 2015-04-01 00:00:00
# 1430438400 : 2015-05-01 00:00:00
# 1433116800 : 2015-06-01 00:00:00

# 1451606400 : 2016-01-01 00:00:00
# 1454284800 : 2016-02-01 00:00:00
# 1456790400 : 2016-03-01 00:00:00
# 1459468800 : 2016-04-01 00:00:00
# 1462060800 : 2016-05-01 00:00:00
# 1464739200 : 2016-06-01 00:00:00

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 [53]:

```
# 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 [54]:

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

Out[54]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude
0	1	1.59	-73.993896	40.750111	-73.974785
1	1	3.30	-74.001648	40.724243	-73.994415
2	1	1.80	-73.963341	40.802788	-73.951820
3	1	0.50	-74.009087	40.713818	-74.004326
4	1	3.00	-73.971176	40.762428	-74.004181

In [55]:

```
# 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 devide whole months time into 10min intravels 24*31*60/10 =4464bins)
jan_2015_groupby.head()
```

Out[55]:

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

In [57]:

```
jan_2015_frame.to_csv('jan_2015_frame.csv')
```

In [58]:

```

# upto now we cleaned data and prepared data for the month 2015,
# now do the same operations for months Jan, Feb, March of 2016
# 1. get the dataframe which includes only required columns
# 2. adding trip times, speed, unix time stamp of pickup_time
# 4. remove the outliers based on trip_times, speed, trip_duration, total_amount
# 5. add pickup_cluster to each data point
# 6. add pickup_bin (index of 10min intravel to which that trip belongs to)
# 7. group by data, based on 'pickup_cluster' and 'pickup_bin'

# 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_removed_2016[['pickup_latitude', 'pickup_longitude']])

    print ("Final groupingby..")
    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_cluster','pickup_bins']).count()

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('yellow_tripdata_2016-01.csv')
month_feb_2016 = dd.read_csv('yellow_tripdata_2016-02.csv')
month_mar_2016 = dd.read_csv('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 groupingby..
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 groupingby..
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 groupingby..

```

3.0 Smoothing

In [59]:

```

# Gets the unique bins where pickup values are present for each each region

# for each cluster region we will collect all the indices of 10min intravels in which t
# he pickups have 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(0,40):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values

```

In [60]:

```
# for every month we get all indices of 10min intervals in which atleast one pickup 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 [61]:

```
# for each cluster number of 10min intravels with 0 pickups
for i in range(40):
    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
-----
for the 30 th cluster number of 10min intavels with zero pickups: 1180
```

```
-----  
for the 31 th cluster number of 10min intavels with zero pickups: 42  
-----  
for the 32 th cluster number of 10min intavels with zero pickups: 44  
-----  
for the 33 th cluster number of 10min intavels with zero pickups: 43  
-----  
for the 34 th cluster number of 10min intavels with zero pickups: 39  
-----  
for the 35 th cluster number of 10min intavels with zero pickups: 42  
-----  
for the 36 th cluster number of 10min intavels with zero pickups: 36  
-----  
for the 37 th cluster number of 10min intavels with zero pickups: 321  
-----  
for the 38 th cluster number of 10min intavels with zero pickups: 36  
-----  
for the 39 th cluster number of 10min intavels with zero pickups: 43  
-----
```

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)
Ex2: x _ => ceil(x/2), ceil(x/2)

In [62]:

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

# for every 10min interval(pickup_bin) we will check if it is there in our unique bin,
# if yes then 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(0,40):
        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 [63]:

```

# 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 intra
vel
# there wont be any value if there are no pickups.
# 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 discus
sed in the above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each region
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        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/res
olved
                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 pick
up 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 pic
up-bin value which has a pickup 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 miss
ing,hence we have no right-limit 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/((ri
ght_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 mi
ssing,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 [64]:

```

#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pickups
# that 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 [65]:

```

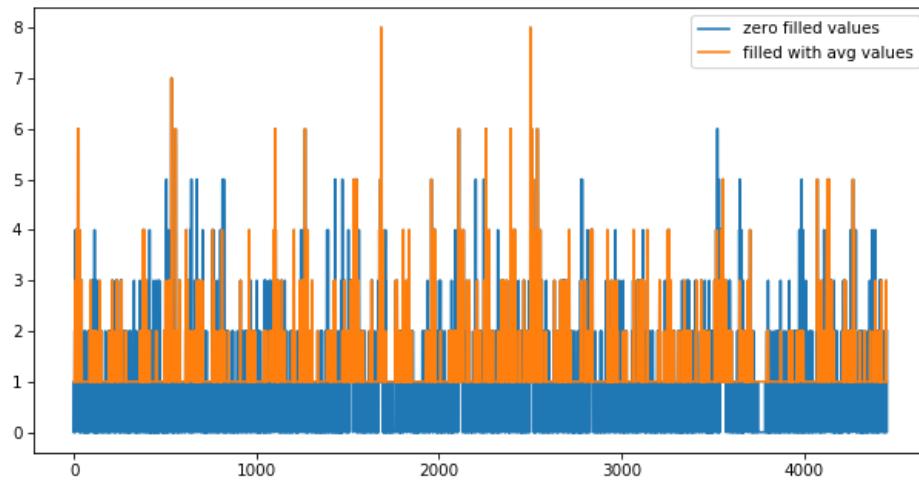
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*30*60/10 = 4320
# for each cluster we will have 4464 values, therefore 40*4464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

```

Number of 10min intervals among all the clusters: 178560

In [0]:

```
# 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]:

```
# why we choose, these methods and which method is used for which data?

# Ans: consider we have data of some month in 2015 jan 1st, 10 _ _ _ 20, i.e there are
# 10 pickups that happened in 1st
# 10min interval, 0 pickups happened in 2nd 10mins interval, 0 pickups happened in 3rd
# 10min interval
# and 20 pickups happened in 4th 10min interval.
# in fill_missing method we replace these values like 10, 0, 0, 20
# where as in smoothing method we replace these values as 6,6,6,6,6, if you can check t
# he number of pickups
# that happened in the first 40min is same in both cases, but if you can observe that w
# e looking at the future values
# when you are using smoothing we are looking at the future number of pickups which mig
# ht cause a data Leakage.

# so we use smoothing for jan 2015th data since it acts as our training data
# and we use simple fill_missing method for 2016 data.
```

In [67]:

```
# Jan-2015 data is smoothed, Jan, Feb & March 2016 data missing values are filled with z
ero
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 an
d 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 w
# hich represents the number of pickups
# that happened for three months in 2016 data

for i in range(0,40):
    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)])

# print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 13104
```

In [70]:

```
import pickle

with open("regions_cum.txt", "wb") as fp:      #Pickling
    pickle.dump(regions_cum, fp)

#with open("test.txt", "rb") as fp:      # Unpickling
#b = pickle.load(fp)
```

In [71]:

```
#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
```

Modeling: Baseline Models

Now we get into modeling 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 i.e $R_t = P_t^{2016}/P_t^{2015}$
2. Using Previous known values of the 2016 data itself to predict the future values

1.0 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

Using Ratio Values - $R_t = (R_{t-1} + R_{t-2} + R_{t-3} \dots R_{t-n})/n$

In [72]:

```

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*40):
        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

```

For the above the Hyperparameter is the window-size (n) which is tuned manually and it is found that the window-size of 3 is optimal for getting the best results using Moving Averages using previous Ratio values therefore we get $R_t = (R_{t-1} + R_{t-2} + R_{t-3})/3$

Next we use the Moving averages of the 2016 values itself to predict the future value using
 $P_t = (P_{t-1} + P_{t-2} + P_{t-3} \dots P_{t-n})/n$

In [73]:

```

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*40):
        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

```

For the above the Hyperparameter is the window-size (n) which is tuned manually and it is found that the window-size of 1 is optimal for getting the best results using Moving Averages using previous 2016 values therefore we get $P_t = P_{t-1}$

2.0 Weighted Moving Averages

The Moving Averages 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

Weighted Moving Averages using Ratio Values -

$$R_t = (N * R_{t-1} + (N - 1) * R_{t-2} + (N - 2) * R_{t-3} \dots 1 * R_{t-n}) / (N * (N + 1) / 2)$$

In [74]:

```

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*40):
        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((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

```

For the above the Hyperparameter is the window-size (n) which is tuned manually and it is found that the window-size of 5 is optimal for getting the best results using Weighted Moving Averages using previous Ratio values therefore we get $R_t = (5 * R_{t-1} + 4 * R_{t-2} + 3 * R_{t-3} + 2 * R_{t-4} + R_{t-5})/15$

Weighted Moving Averages using Previous 2016 Values -

$$P_t = (N * P_{t-1} + (N - 1) * P_{t-2} + (N - 2) * P_{t-3} \dots 1 * P_{t-n})/(N * (N + 1)/2)$$

In [75]:

```

def WA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,4464*40):
        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 therefore we get $P_t = (2 * P_{t-1} + P_{t-2})/3$

3.0 Exponential Weighted Moving Averages

https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average

(https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average) Through weighted averaged we have satisfied the analogy of giving higher weights to the latest value and decreasing weights to the subsequent ones but we still do not know which is the correct weighting scheme as there are infinitely many possibilities in which we can assign weights in a non-increasing order and tune the the hyperparameter window-size. To simplify this process we use Exponential Moving Averages which is a more logical way towards assigning weights and at the same time also using an optimal window-size.

In exponential moving averages we use a single hyperparameter alpha (α) which is a value between 0 & 1 and based on the value of the hyperparameter alpha the weights and the window sizes are configured.

For eg. If $\alpha = 0.9$ then the number of days on which the value of the current iteration is based is~ $1/(1 - \alpha) = 10$ i.e. we consider values 10 days prior before we predict the value for the current iteration. Also the weights are assigned using $2/(N + 1) = 0.18$,where N = number of prior values being considered, hence from this it is implied that the first or latest value is assigned a weight of 0.18 which keeps exponentially decreasing for the subsequent values.

$$R'_t = \alpha * R_{t-1} + (1 - \alpha) * R'_{t-1}$$

In [76]:

```
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*40):
        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
```

$$P'_t = \alpha * P_{t-1} + (1 - \alpha) * P'_{t-1}$$

In [77]:

```
def EA_P1_Predictions(ratios,month):
    predicted_value= (ratios['Prediction'].values)[0]
    alpha=0.3
    error=[]
    predicted_values=[]
    for i in range(0,4464*40):
        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['Pr
    ediction'].values))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

In [78]:

```
mean_err=[0]*10
median_err=[0]*10
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')
```

4.0 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 [79]:

```

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

```

Error Metric Matrix (Forecasting Methods) - MAPE & MSE

```

-----  

-----  

Moving Averages (Ratios) - 3133512 MSE: 1196.2953853046595 MAPE: 0.2278515635  

-----  

Moving Averages (2016 Values) - 2025738 MSE: 254.66309363799283 MAPE: 0.1558345871  

-----  

Weighted Moving Averages (Ratios) - 4871415 MSE: 1053.083529345878 MAPE: 0.2270652914  

Weighted Moving Averages (2016 Values) - 992932 MSE: 224.81054547491038 MAPE: 0.1479482182  

-----  

Exponential Moving Averages (Ratios) - 534 MSE: 1019.3071012544802 MAPE: 0.2275474636148  

Exponential Moving Averages (2016 Values) - 153 MSE: 222.35159610215055 MAPE: 0.1475381297798

```

Please Note:- The above comparisons are made using Jan 2015 and Jan 2016 only

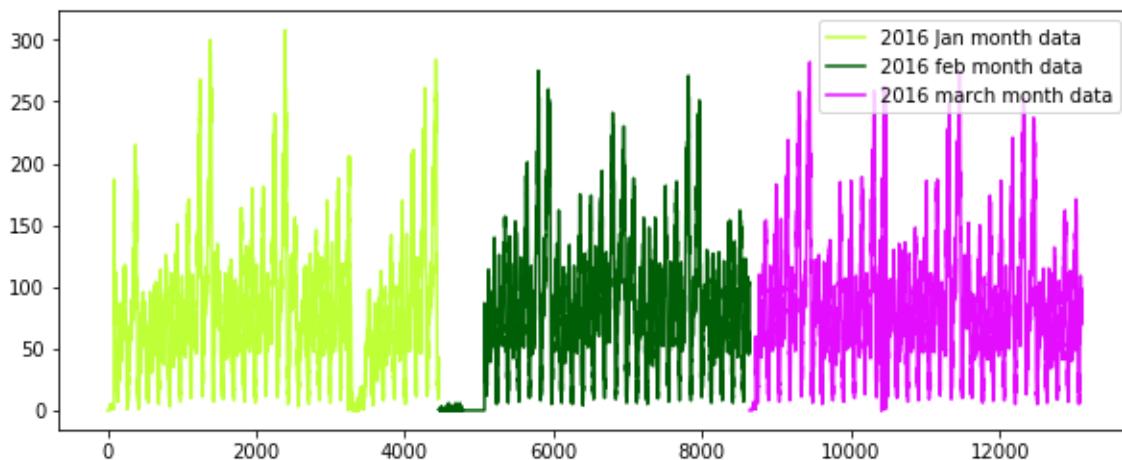
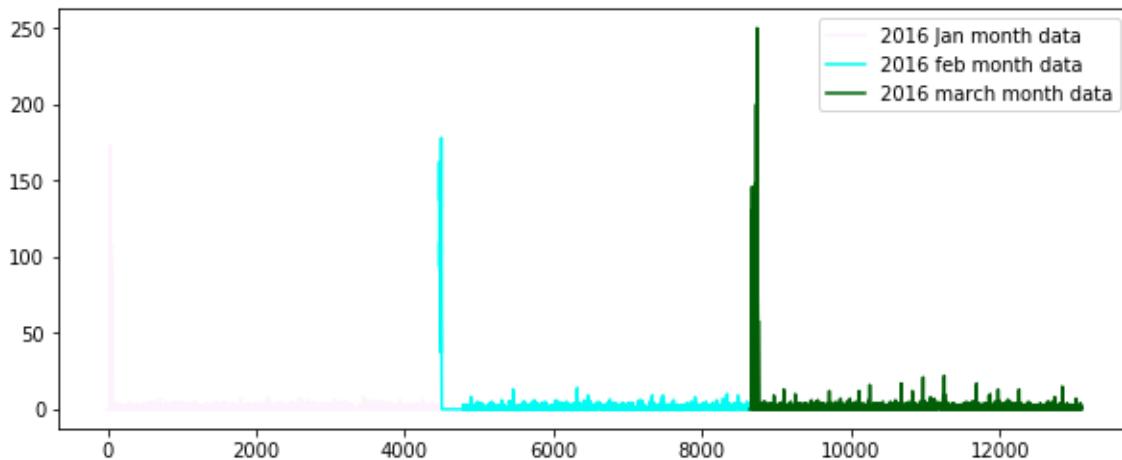
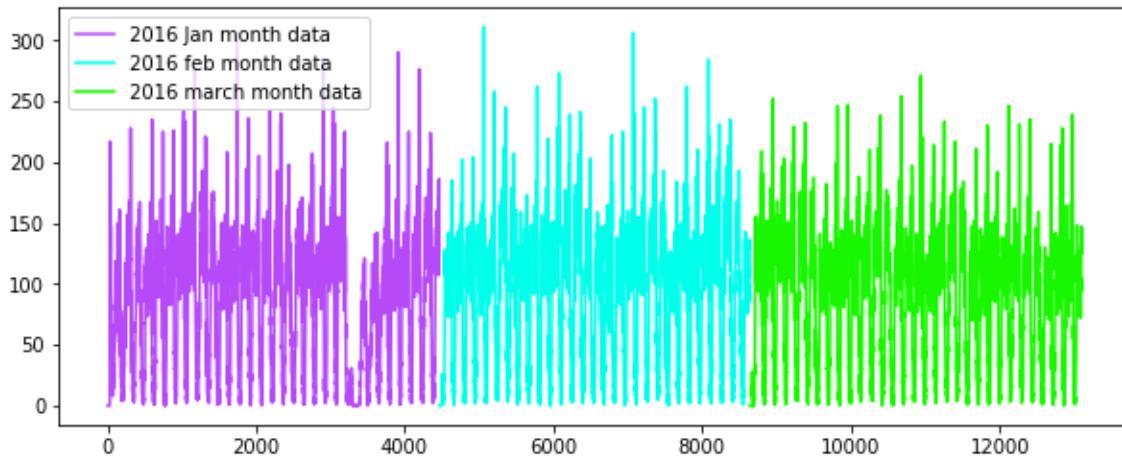
From the above matrix it is inferred that the best forecasting model for our prediction would be:-

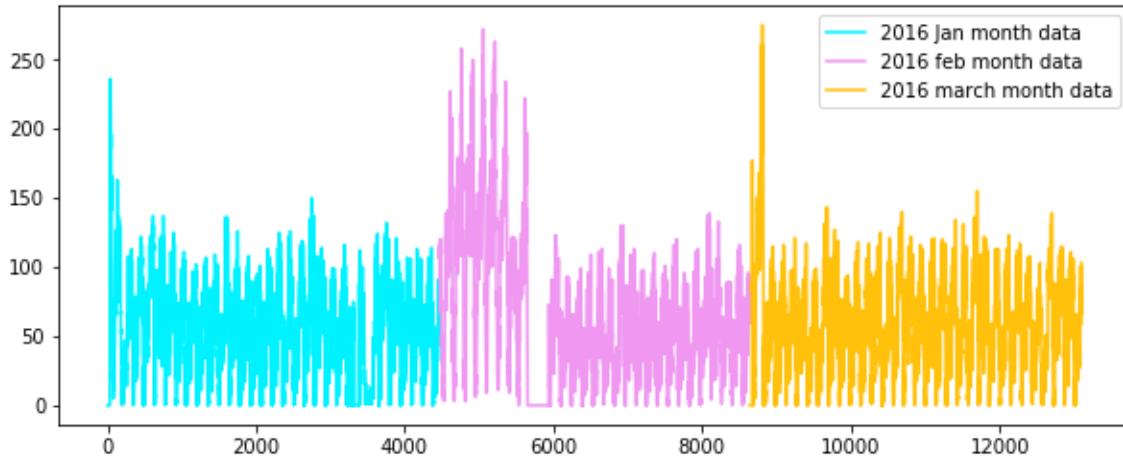
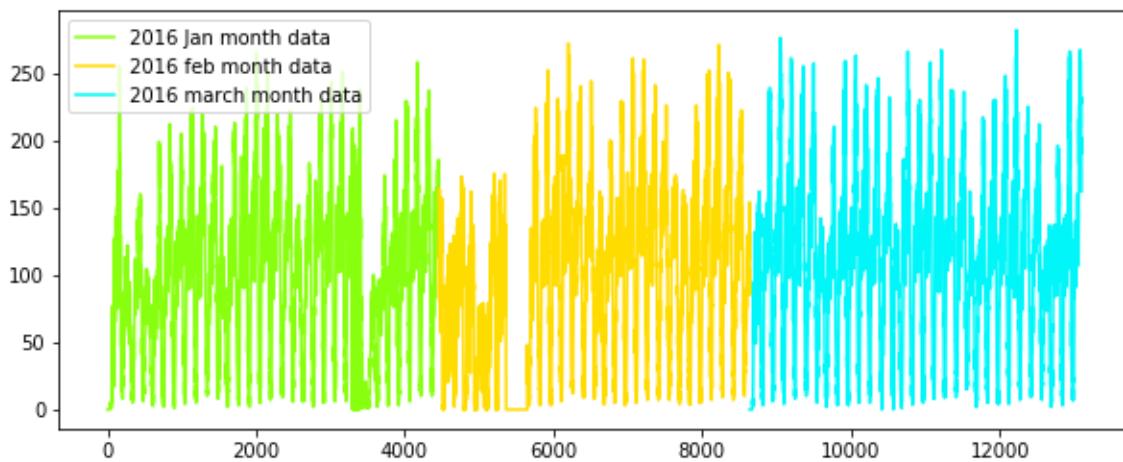
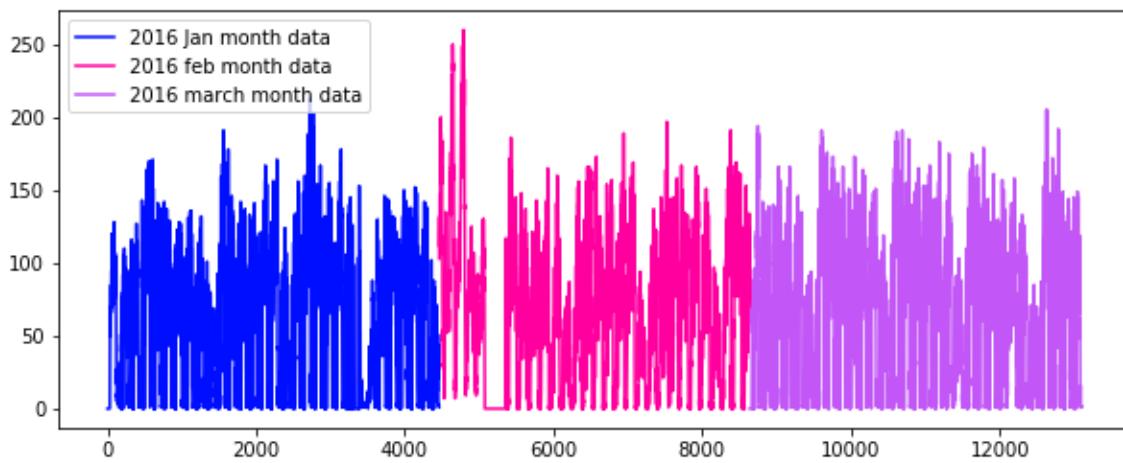
$$P'_t = \alpha * P_{t-1} + (1 - \alpha) * P'_{t-1} \text{ i.e Exponential Moving Averages using 2016 Values}$$

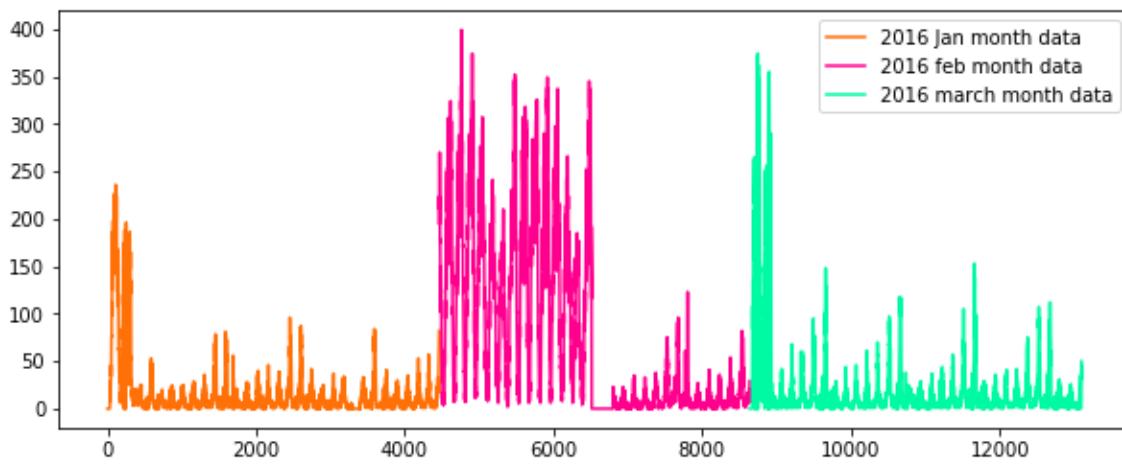
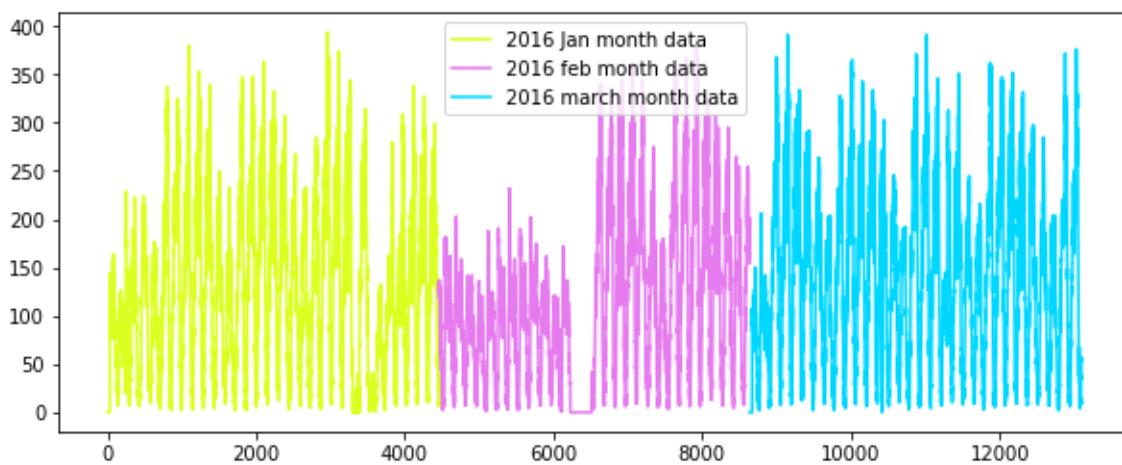
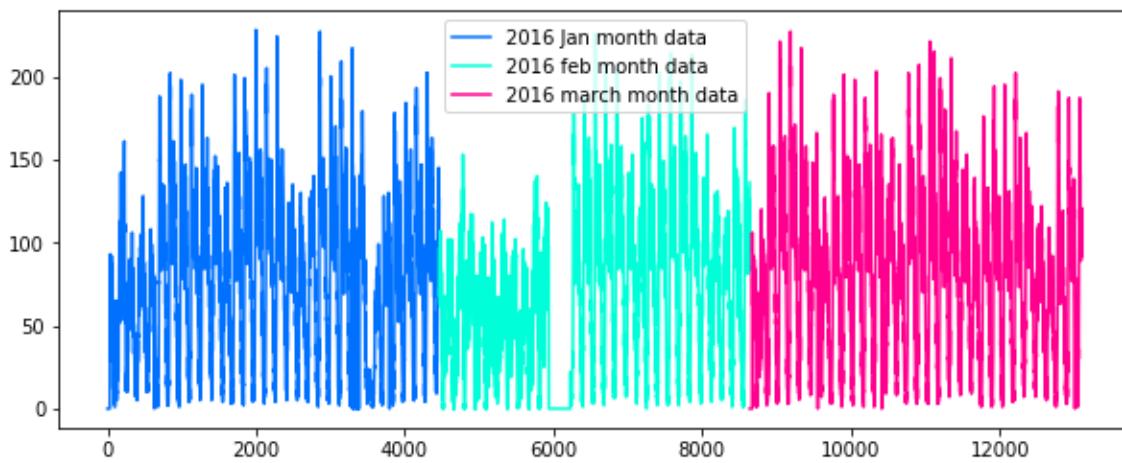
5.0 Time series and Fourier Transforms

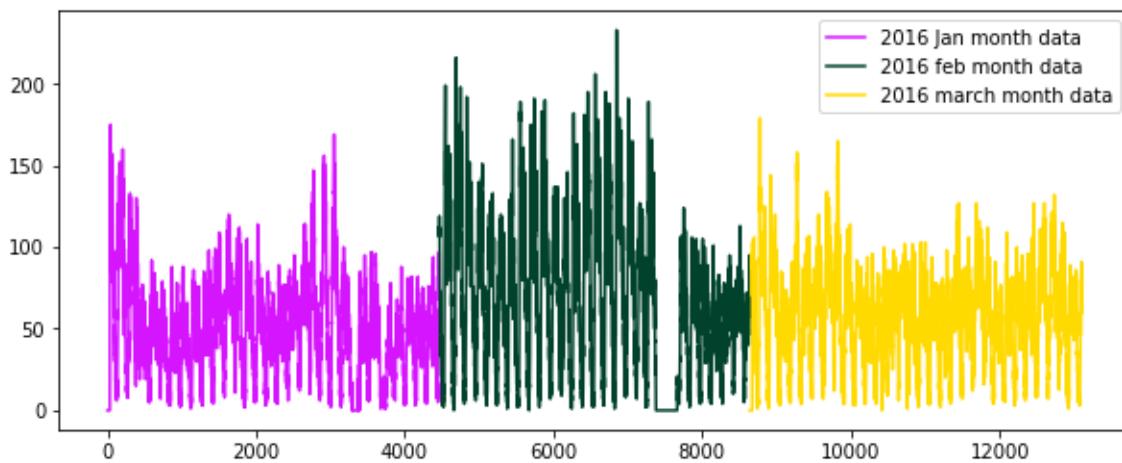
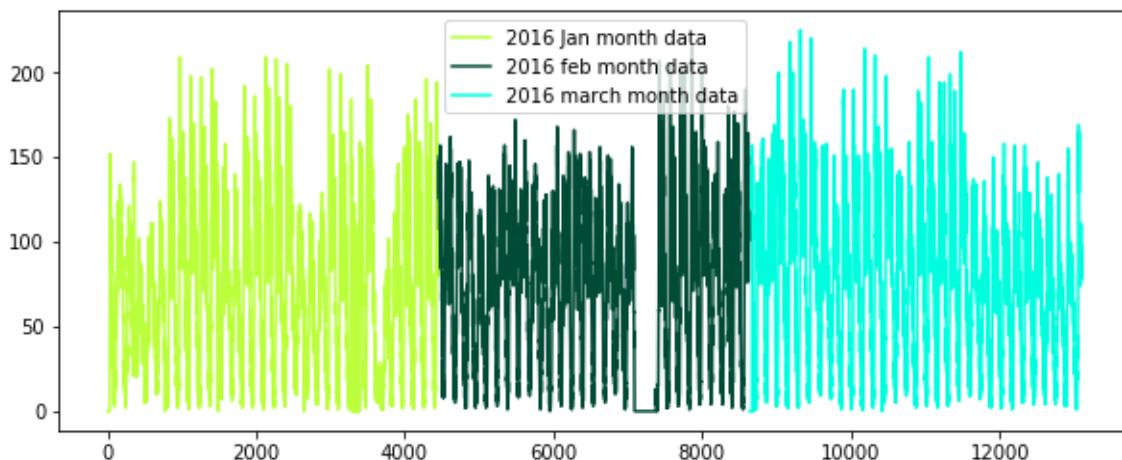
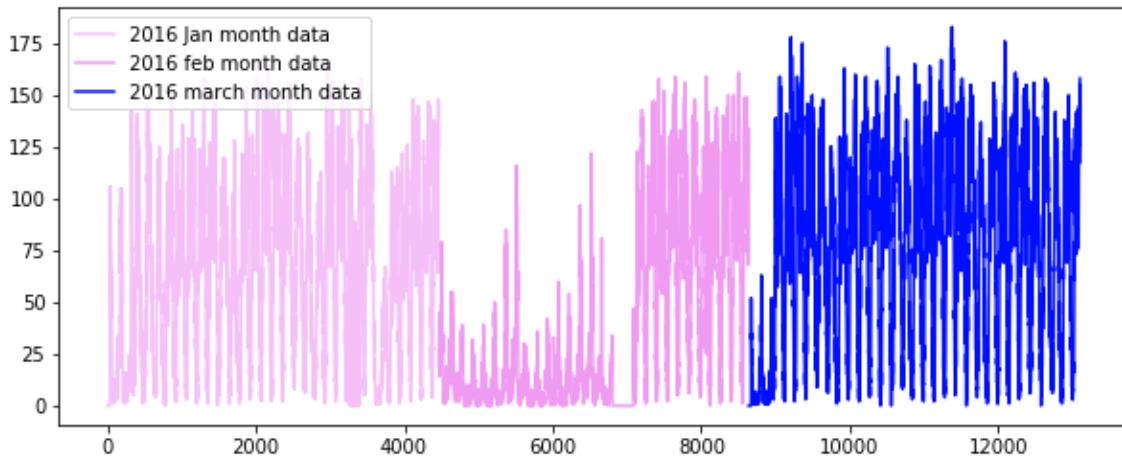
In [80]:

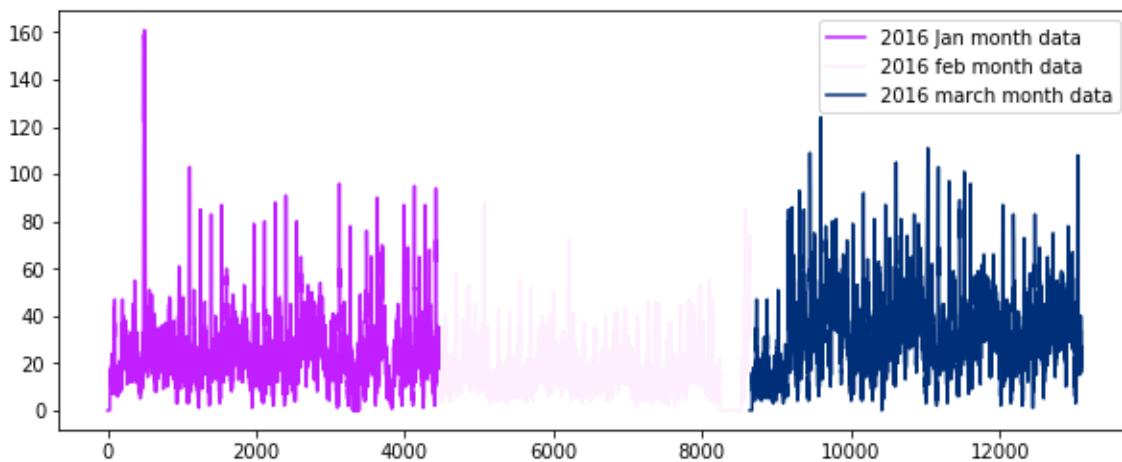
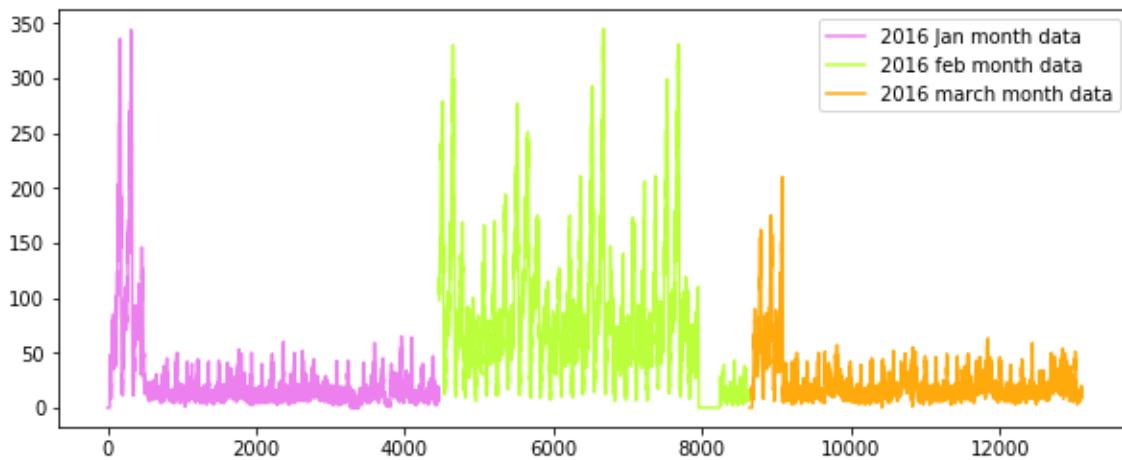
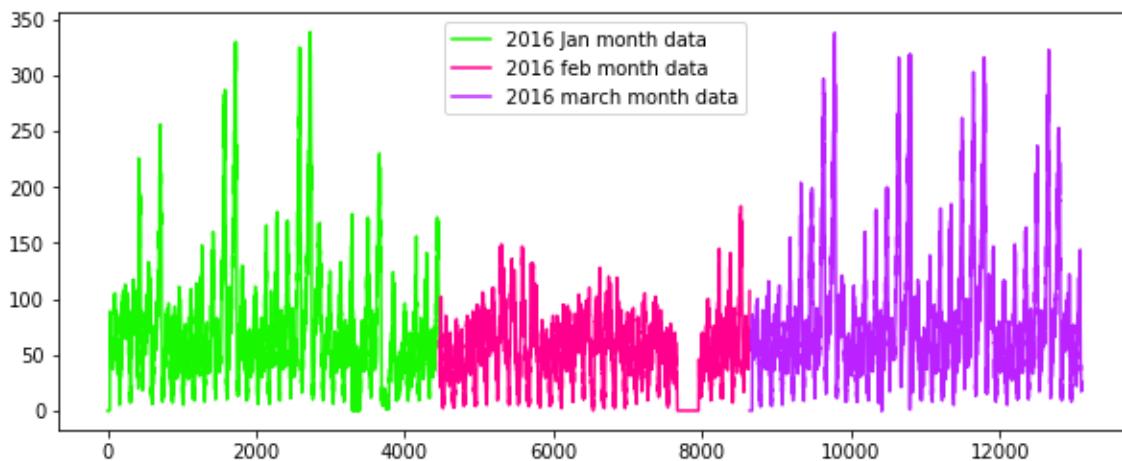
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40):
    plt.figure(figsize=(10,4))
    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()
```

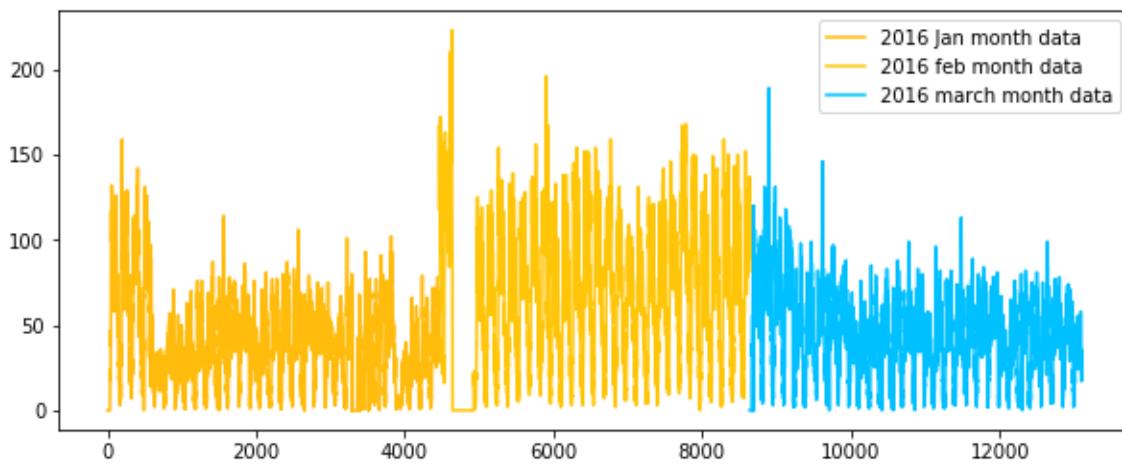
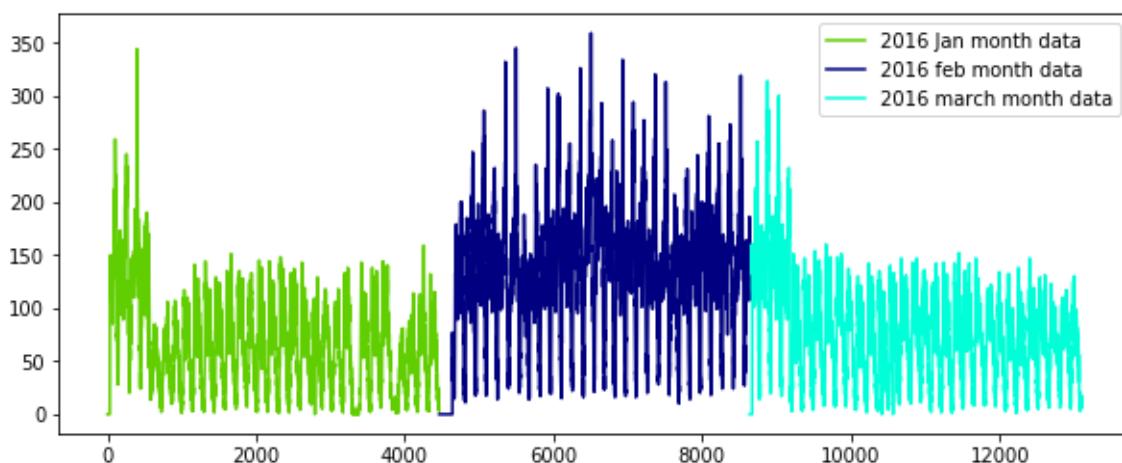
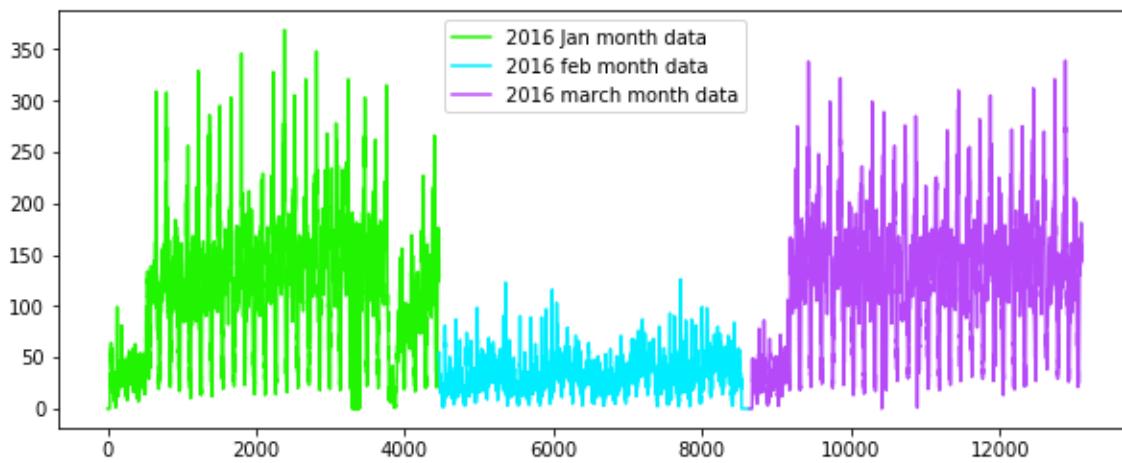


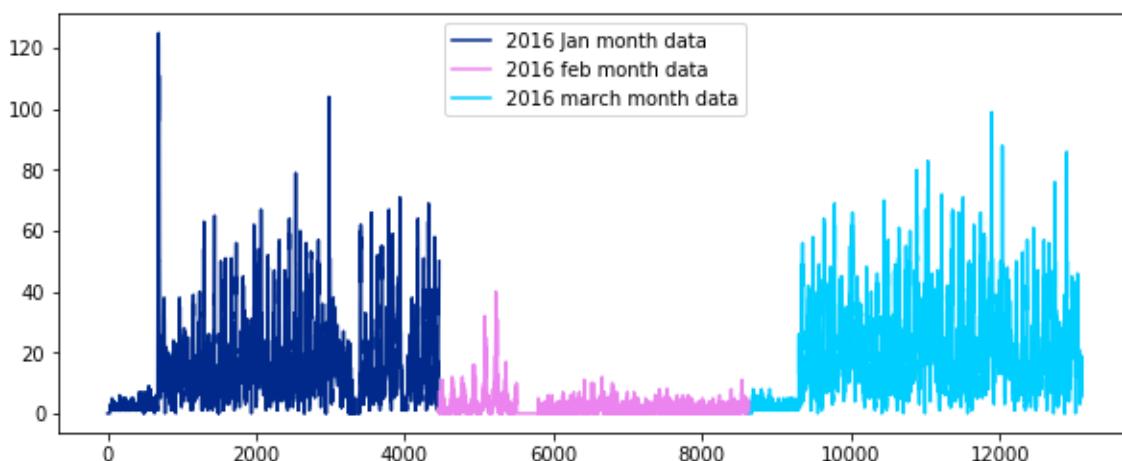
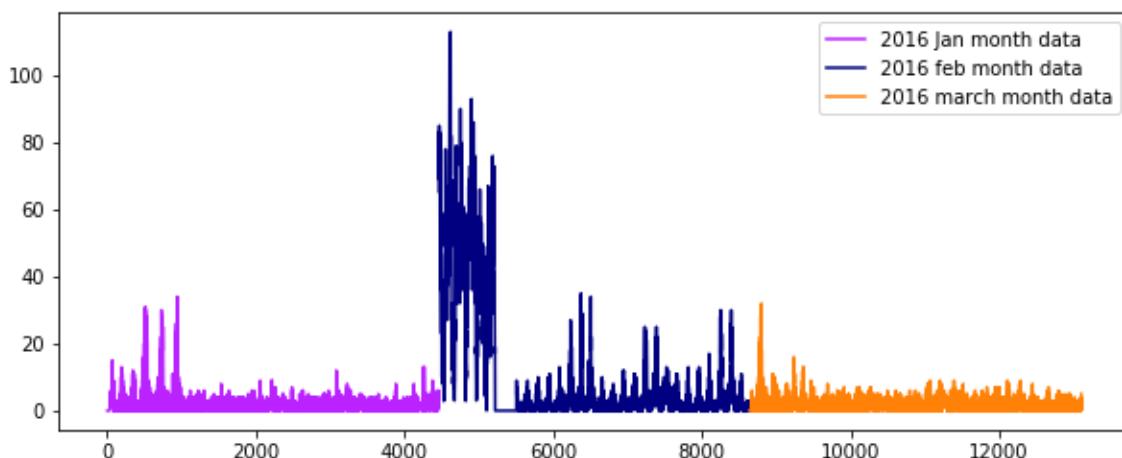
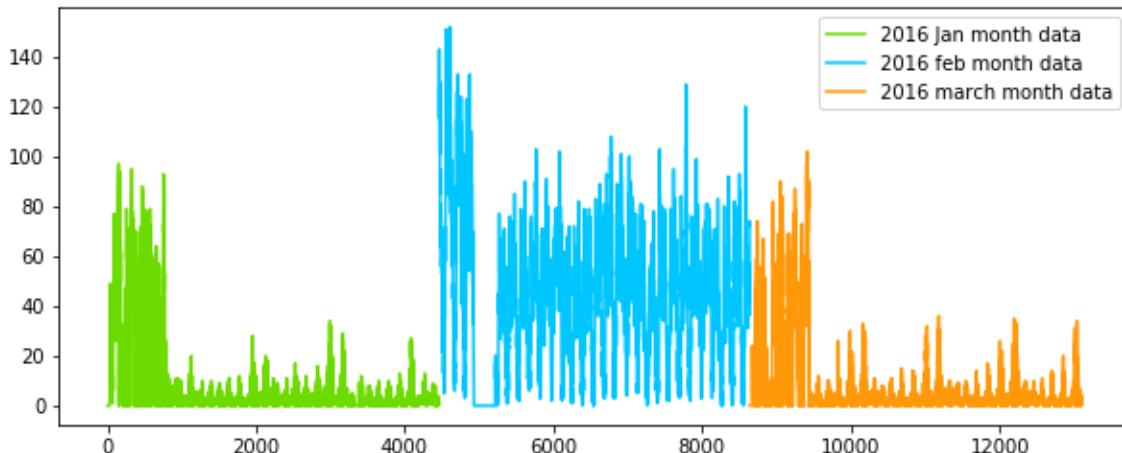


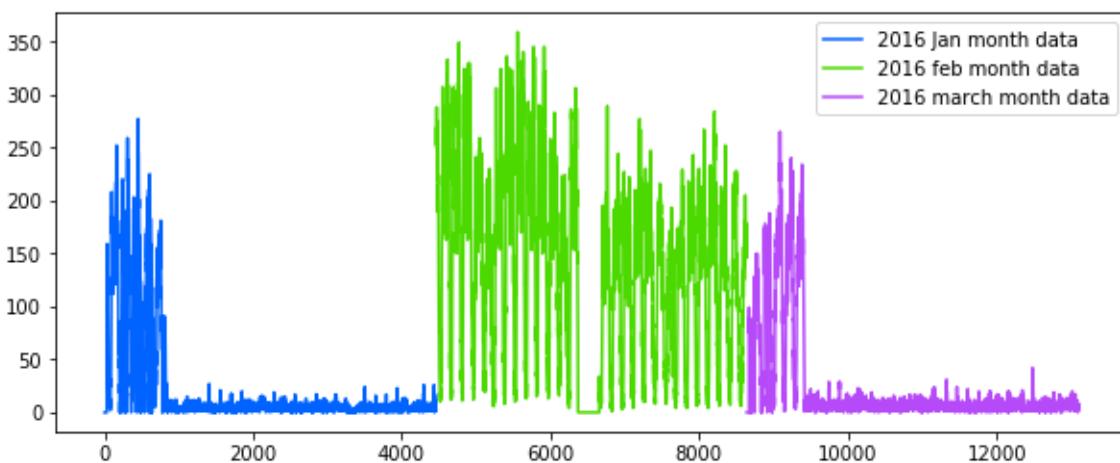
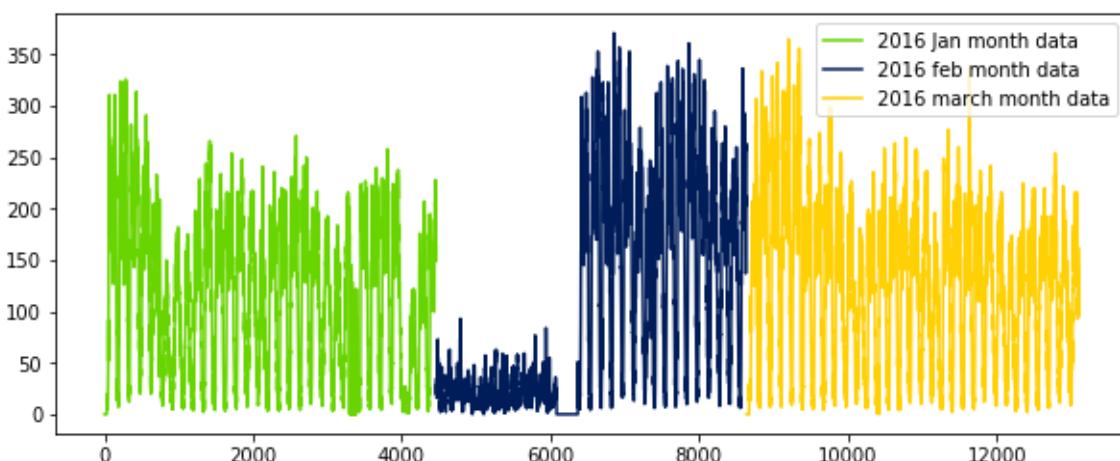
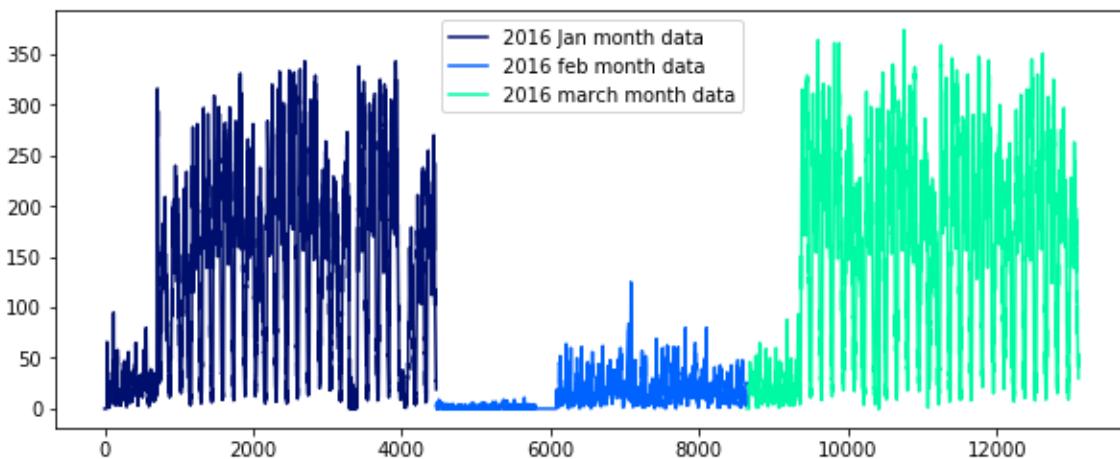


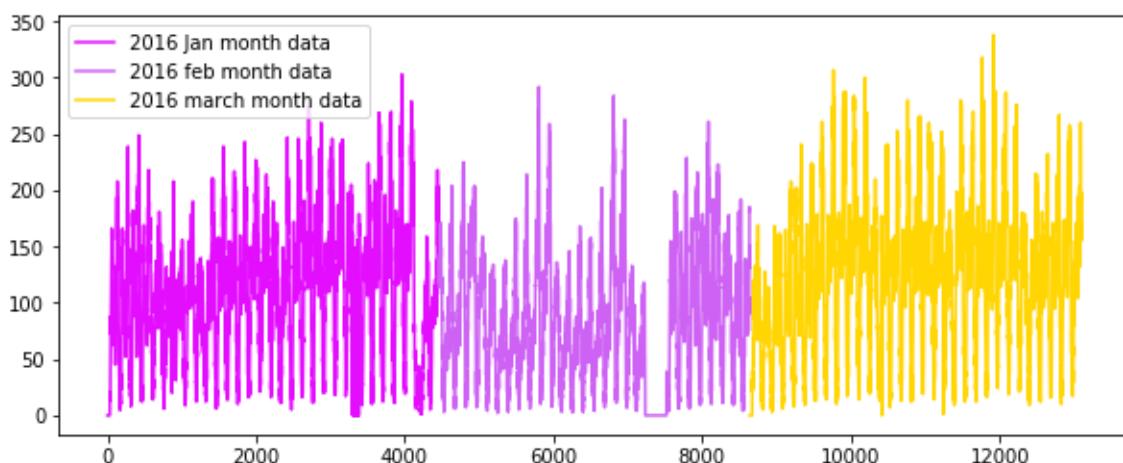
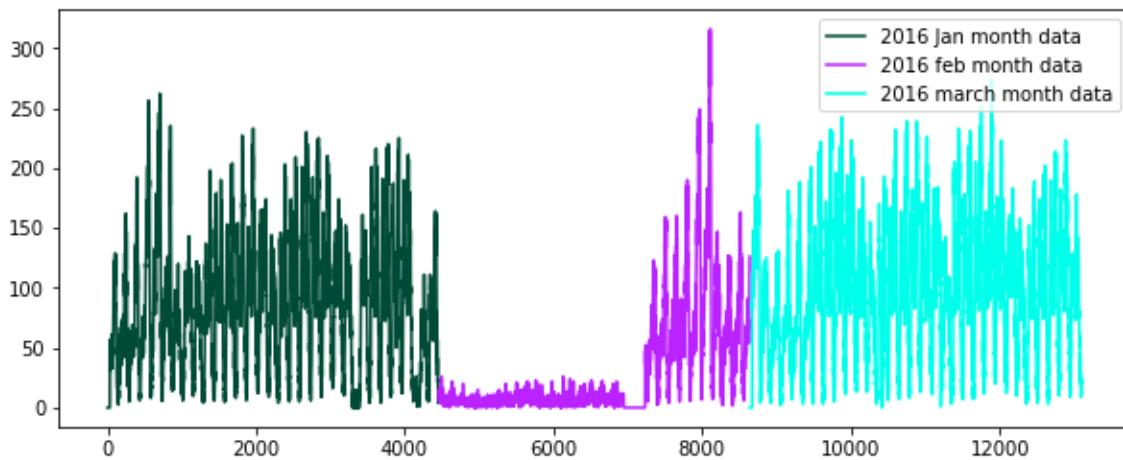
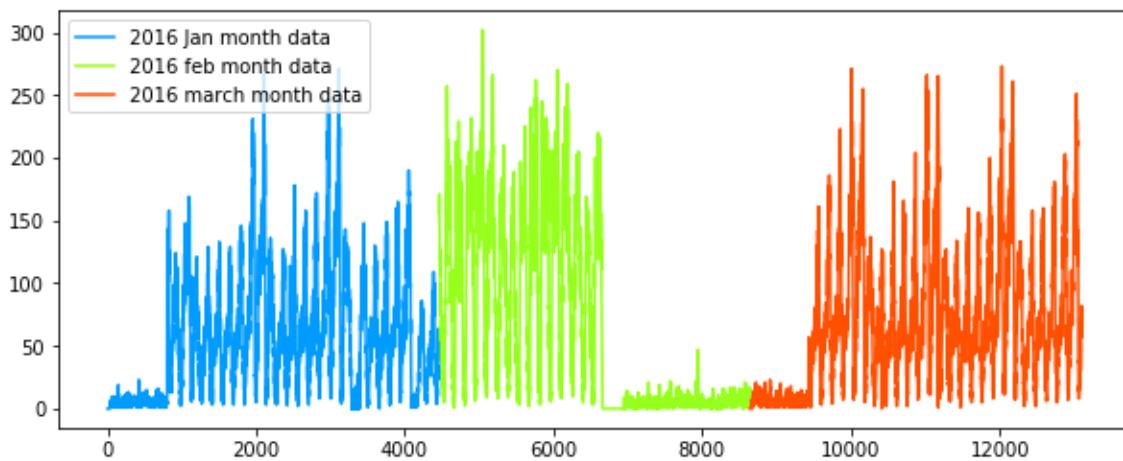


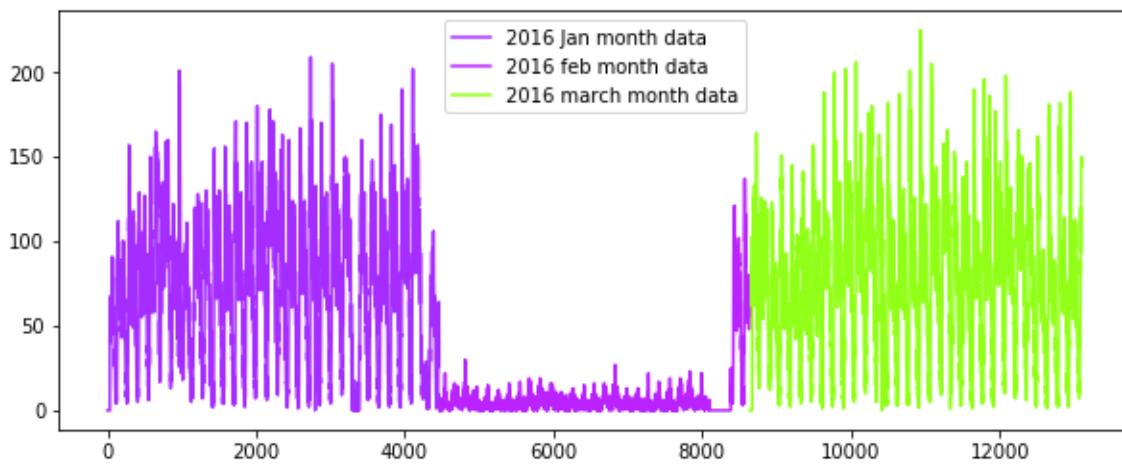
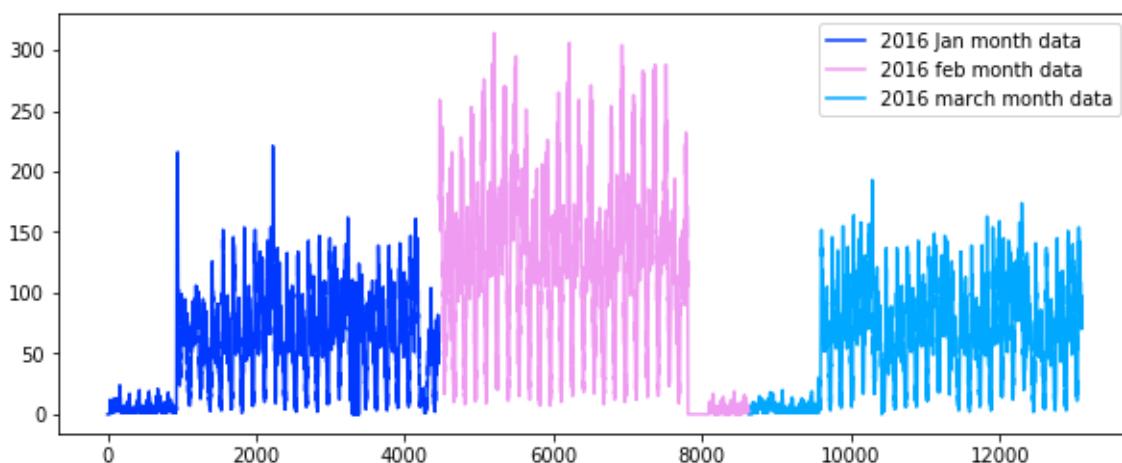
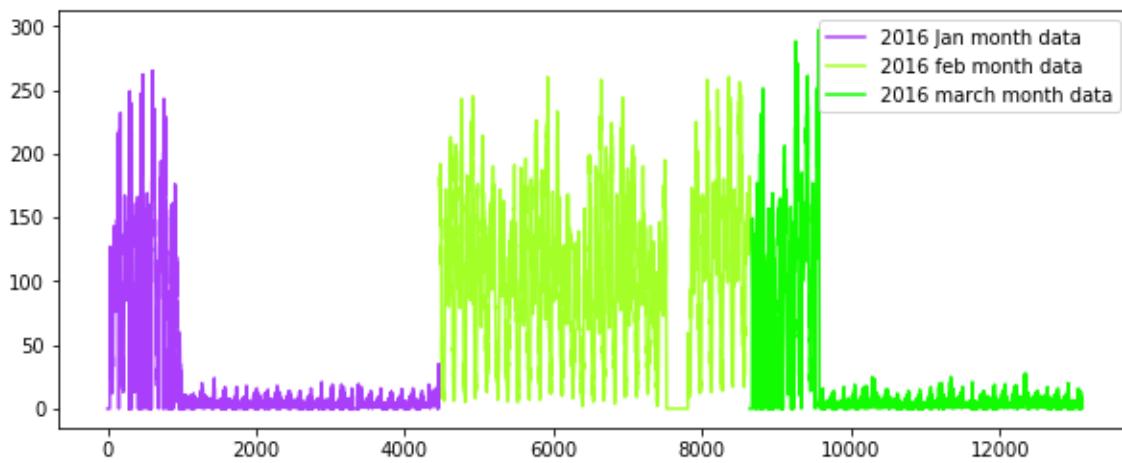


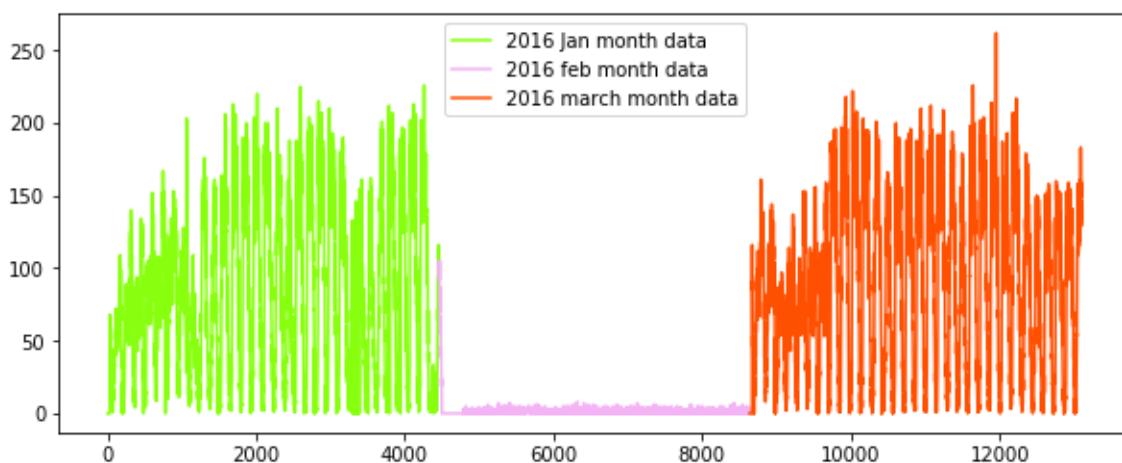
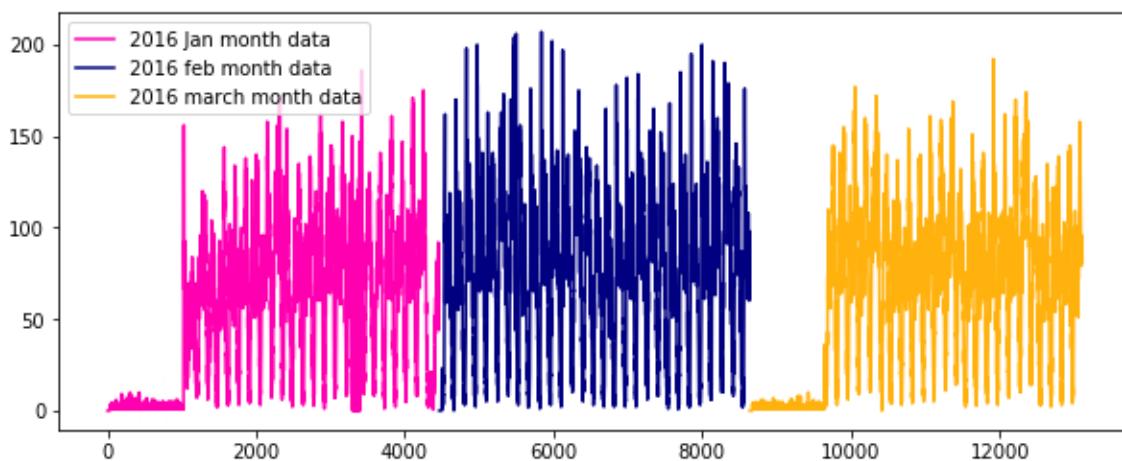
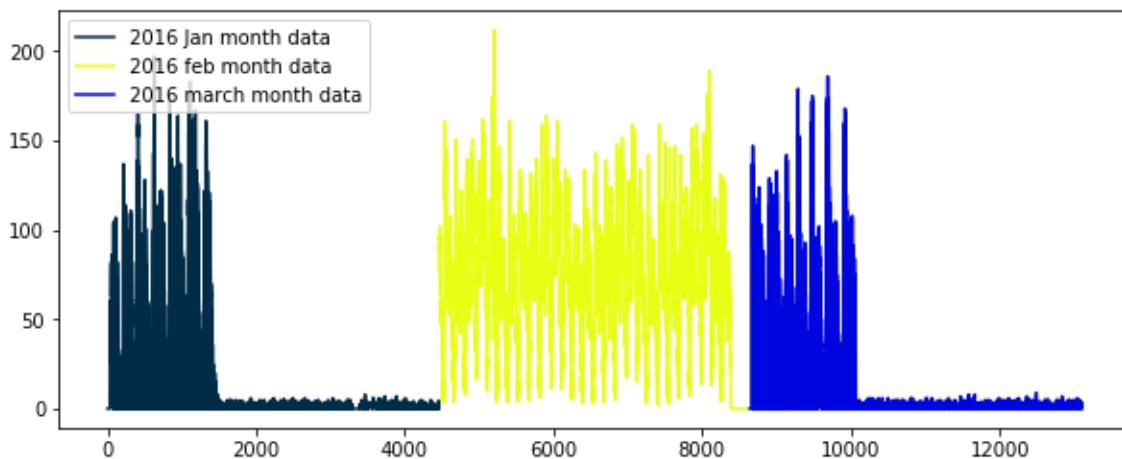


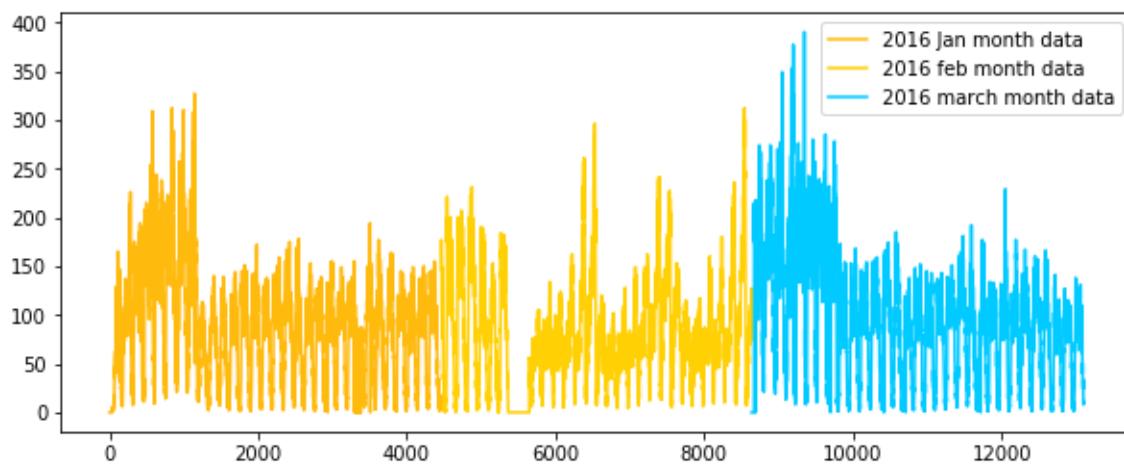
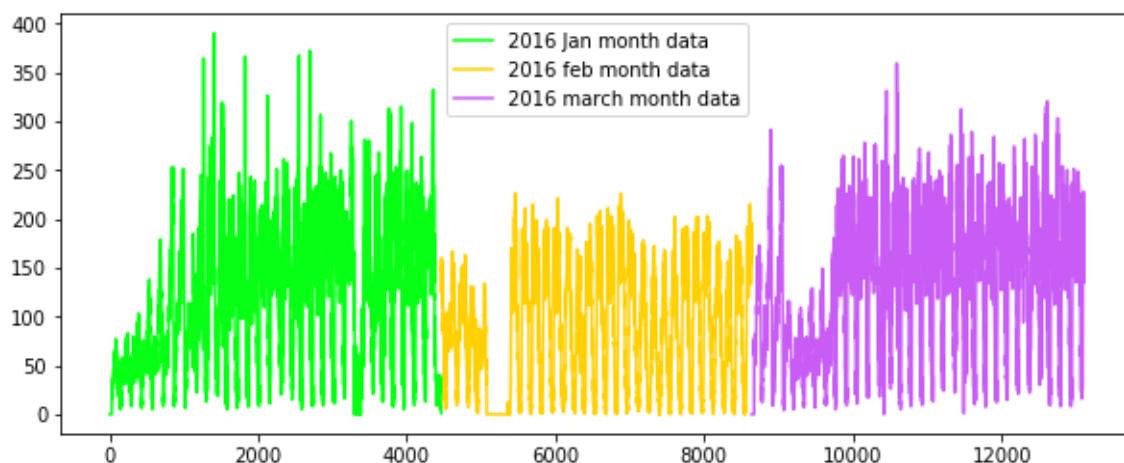
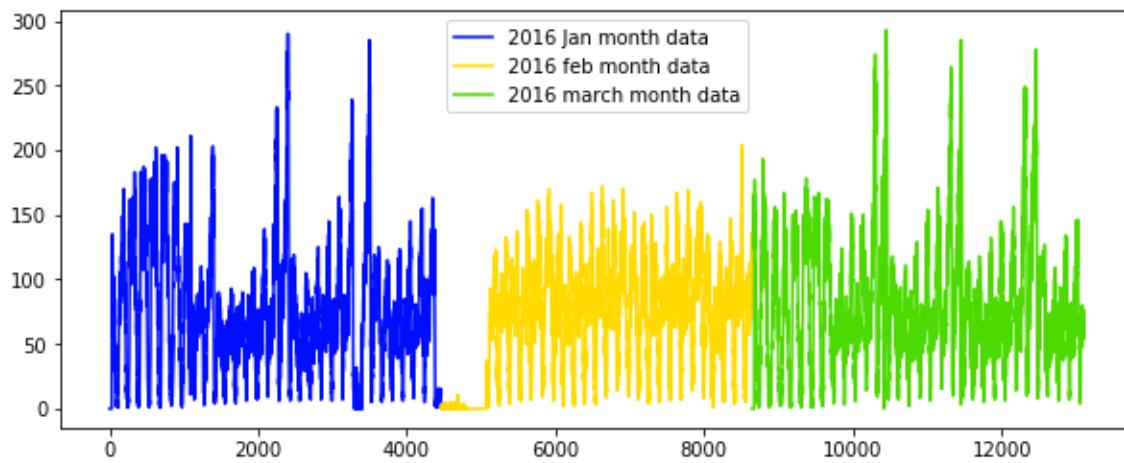


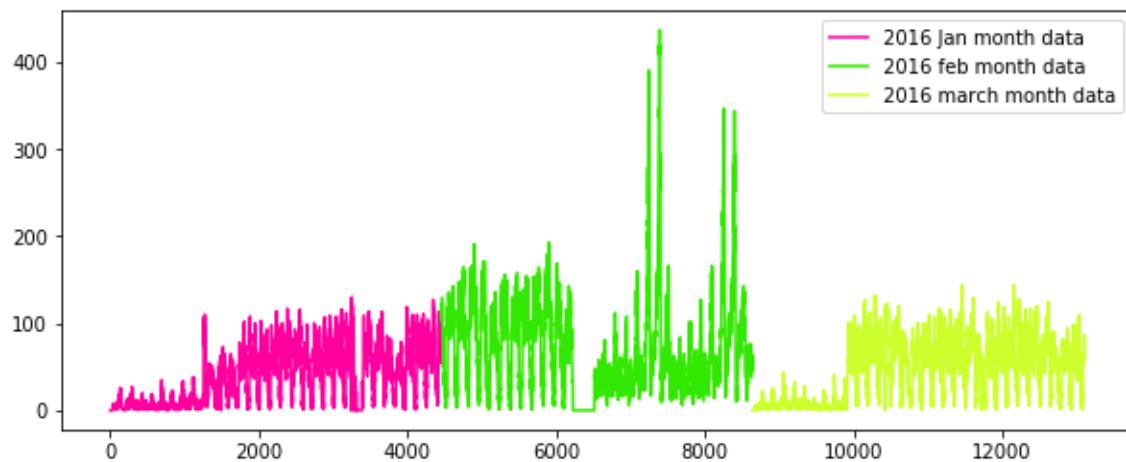
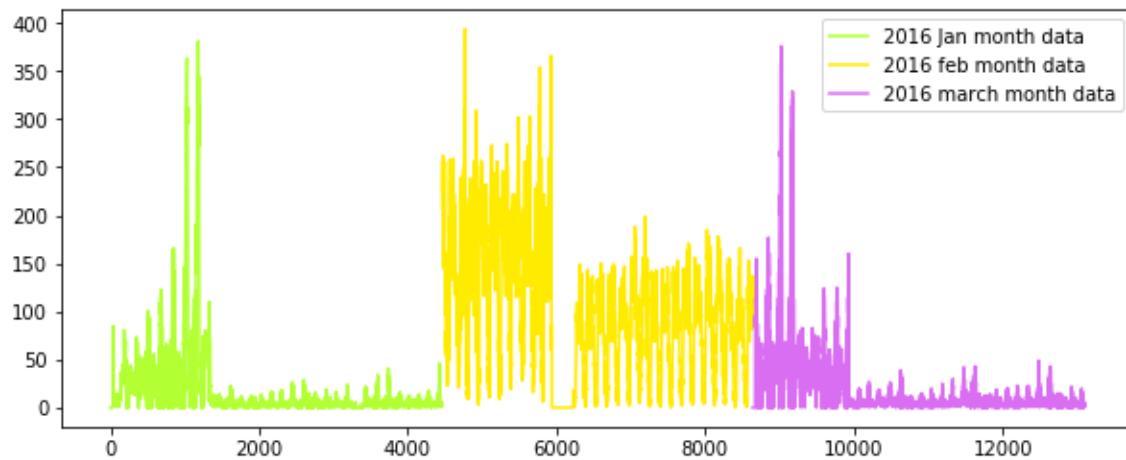
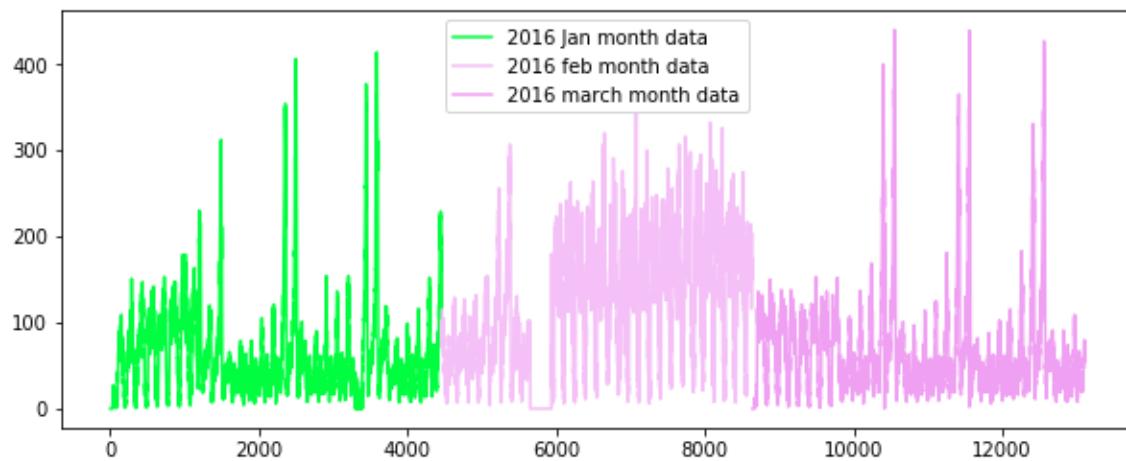


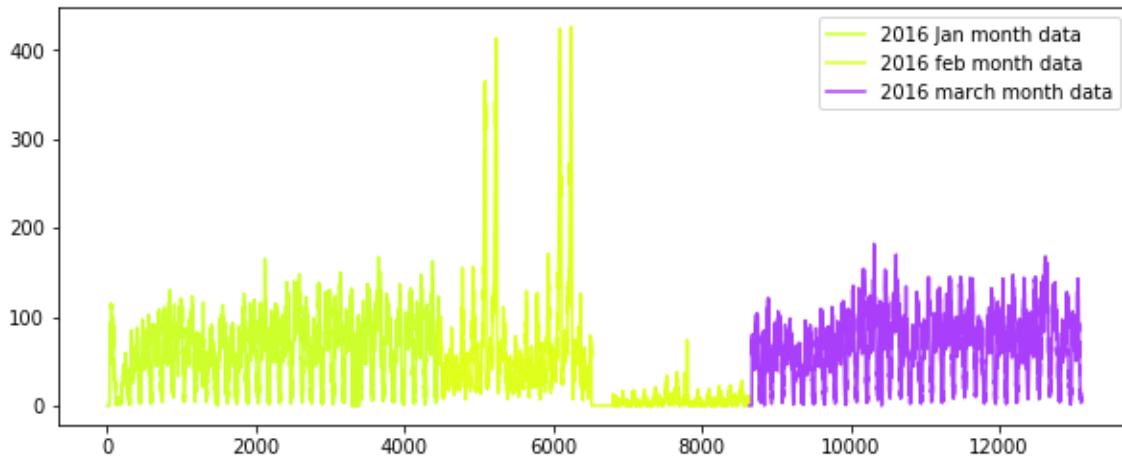






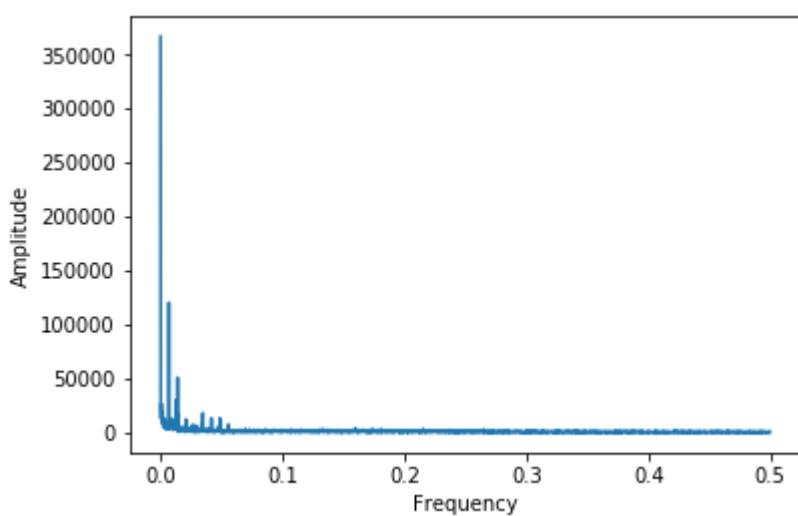






In [81]:

```
# 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("Amplitude")
plt.show()
```



5.1 Fourier features (5 frequencies and its correponding amplitudes)

In [82]:

```
# Fourier features dataframe - Stores fourier features i.e 5 peaks & 5 frequencies for
# all clusters.
fourier_features = pd.DataFrame(['Amp_1', 'Amp_2', 'Amp_3', 'Amp_4', 'Amp_5', 'Freq_1',
'Freq_2', 'Freq_3', 'Freq_4',
'Freq_5'])
region_FF = [] # List storing FF dataframes for every cluster/region
```

In [84]:

```

# iterating through each cluster
for i in range(0,40):

    # for each month calculate fft and get frequency
    # regions_cum hold data for each cluster in format jan,feb,mar. first 4464 values are for jan, next 4176 values are for feb and rest are for march.
    jan_fft_data = regions_cum[i][0:4464]
    feb_fft_data = regions_cum[i][4464:4464+4176]
    mar_fft_data = regions_cum[i][4464+4176: 4464+4176+4464]

    # calculate fft i.e Amplitude & frequency
    jan_fft_amp = np.fft.fft(jan_fft_data)
    jan_fft_freq = np.fft.fftfreq(4464, 1)

    feb_fft_amp = np.fft.fft(feb_fft_data)
    feb_fft_freq = np.fft.fftfreq(4176, 1)

    mar_fft_amp = np.fft.fft(mar_fft_data)
    mar_fft_freq = np.fft.fftfreq(4464, 1)

    # Sort the Amplitude and frequency and consider top 5 values
    jan_fft_amp = sorted(jan_fft_amp, reverse = True)[:5]
    jan_fft_freq = sorted(jan_fft_freq, reverse = True)[:5]

    feb_fft_amp = sorted(feb_fft_amp, reverse = True)[:5]
    feb_fft_freq = sorted(feb_fft_freq, reverse = True)[:5]

    mar_fft_amp = sorted(mar_fft_amp, reverse = True)[:5]
    mar_fft_freq = sorted(mar_fft_freq, reverse = True)[:5]

    # Frequencies & amplitudes remain same for every point in the cluster belonging to a particluar month
    a1 = jan_fft_amp
    a2 = feb_fft_amp
    a3 = mar_fft_amp
    f1 = jan_fft_freq
    f2 = feb_fft_freq
    f3 = mar_fft_freq
    for i in range(5):
        jan_fft_amp[i] = [a1[i]] * 4464
        feb_fft_amp[i] = [a2[i]] * 4176
        mar_fft_amp[i] = [a3[i]] * 4464

        jan_fft_freq[i] = [f1[i]] * 4464
        feb_fft_freq[i] = [f2[i]] * 4176
        mar_fft_freq[i] = [f3[i]] * 4464

    # Converting to numpy array and Transpose to get right dimension.
    jan_fft_amp = np.array(jan_fft_amp).T # converting from shape (5x4464) to (4464x5)
    feb_fft_amp = np.array(feb_fft_amp).T
    mar_fft_amp = np.array(mar_fft_amp).T

    jan_fft_freq = np.array(jan_fft_freq).T
    feb_fft_freq = np.array(feb_fft_freq).T
    mar_fft_freq = np.array(mar_fft_freq).T

    # Joining amplitude and frequency of same month.
    jan_cluster = np.hstack((jan_fft_amp, jan_fft_freq))

```

```

feb_cluster = np.hstack((feb_fft_amp, feb_fft_freq))
mar_cluster = np.hstack((mar_fft_amp, mar_fft_freq))

#combining jan, feb & march clusters together
cluster = np.vstack((jan_cluster, feb_cluster, mar_cluster))

#Cluster Frame stores the features for a single cluster
cluster_features = pd.DataFrame(cluster, columns=['Amp_1', 'Amp_2', 'Amp_3', 'Amp_4', 'Amp_5', 'Freq_1', 'Freq_2',
                                                    'Freq_3', 'Freq_4', 'Freq_5'])
cluster_features = cluster_features.astype(np.float)
region_FF.append(cluster_features)

```

In [86]:

```

fourier_features = region_FF[0]
for i in range(1, len(region_FF)):
    fourier_features = pd.concat([fourier_features, region_FF[i]], ignore_index=True)
fourier_features = fourier_features.fillna(0)

```

In [87]:

```

print(fourier_features.shape)

(1048320, 10)

```

In [88]:

```

fourier_features = fourier_features.astype(np.float)
fourier_features.head()

```

Out[88]:

	Amp_1	Amp_2	Amp_3	Amp_4	Amp_5	Freq_1	Fr
0	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.49
1	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.49
2	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.49
3	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.49
4	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.49

In [89]:

```

fourier_features.to_csv('fourier_features.csv')

```

5.2 Triple Exponential Smoothing a.k.a Holt-Winters Method

Ref: <https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/>
[\(https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/\)](https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/)

The idea behind triple exponential smoothing is to apply exponential smoothing to the seasonal components in addition to level and trend. The smoothing is applied across seasons, e.g. the seasonal component of the 3rd point into the season would be exponentially smoothed with the one from the 3rd point of last season, 3rd point two seasons ago, etc. In math notation we now have four equations (see footnote):

```
lx=a(yx-sx-L)+(1-a)(lx-1+bx-1) #level bx=β(lx-lx-1)+(1-β)bx-1 #trend sx=γ(yx-lx)+(1-γ)sx-L #seasonal
y^x+m=lx+mbx+sx-L+1+(m-1)modL #forecast
```

In [90]:

```
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen
```

In [91]:

```
def initial_seasonal_components(series, slen):
    seasonals = {}
    season_averages = []
    n_seasons = int(len(series)/slen)
    # compute season averages
    for j in range(n_seasons):
        season_averages.append(sum(series[slen*j:slen*j+slen])/float(slen))
    # compute initial values
    for i in range(slen):
        sum_of_vals_over_avg = 0.0
        for j in range(n_seasons):
            sum_of_vals_over_avg += series[slen*j+i]-season_averages[j]
        seasonals[i] = sum_of_vals_over_avg/n_seasons
    return seasonals
```

In [92]:

```
def triple_exponential_smoothing(series, slen, alpha, beta, gamma, n_preds):
    result = []
    seasonals = initial_seasonal_components(series, slen)
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
            result.append(series[0])
            continue
        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append((smooth + m*trend) + seasonals[i%slen])
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-alpha)*(smooth+trend)
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            result.append(smooth+trend+seasonals[i%slen])
    return result
```

In [95]:

```
alpha = 0.2
beta = 0.15
gamma = 0.2
season_len = 24 #corresponds to time period of 1 day
n_preds= 0 # I do not want any future predictions wrt points, hence I have set it to 0

tes_values = []
tes_list = []
tsne_flat_tes_avg = []
for r in range(0,40):
    tes_values = triple_exponential_smoothing(regions_cum[r][0:13104], season_len, alpha, beta, gamma, 0)
    tes_list.append(tes_values[5:])
```

In [98]:

len(tes_values)

Out[98]:

13104

In [102]:

```
import pickle

with open("tes_list.txt", "wb") as fp: #Pickling
    pickle.dump(tes_list, fp)

#with open("test.txt", "rb") as fp: # Unpickling
#    b = pickle.load(fp)
```

Data splitting

1.0 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 70% data in train and 30% in test, ordered date-wise for every region

In [103]:

```

# Preparing data to be split into train and test, The below prepares data in cumulative
# form which will be later split 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 w
# hich 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 varable
# 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 clust
er
# 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 cluste
r
# Ex: [[cent_long 13099times],[cent_long 13099times], [cent_long 13099times].... 40 lis
ts]
# 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 o
f the week that pickup bin belongs to
# it is list of lists
tsne_weekday = []

# its an numpy 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+1t
h 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
for i in range(0,40):
    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 pick
ups 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,x
3..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:]
```

In [104]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_w
eekday[0]) == 40*13099 == len(output)*len(output[0])
```

Out[104]:

True

In [105]:

```

# Getting the predictions of exponential moving averages to be used as a feature in cumulative form

# upto now we computed 8 features for every data point that starts from 50th min of the day
# 1. cluster center Latitude
# 2. cluster center Longitude
# 3. day of the week
# 4. f_t_1: number of pickups that are happened previous t-1th 10min intravel
# 5. f_t_2: number of pickups that are happened previous t-2th 10min intravel
# 6. f_t_3: number of pickups that are happened previous t-3th 10min intravel
# 7. f_t_4: number of pickups that are happened previous t-4th 10min intravel
# 8. f_t_5: number of pickups that are happened previous t-5th 10min intravel

# from the baseline models we said the exponential weighted moving avarage gives us the best error
# we will try to add the same exponential weighted moving avarage at t as a feature to our data
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.3

# it is a temporary array that store exponential weighted moving avarage for each 10min intravel,
# for each cluster it will get reset
# for every cluster it contains 13104 values
predicted_values=[]

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], .. 40 lsits]
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    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 [106]:

```

# train, test split : 70% 30% 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 70% data in train and 30% in test,
# ordered date-wise for every region
print("size of train data :", int(13099*0.7))
print("size of test data :", int(13099*0.3))

size of train data : 9169
size of test data : 3929

```

2.0 Splitting baseline model features & special features into train & test

In [107]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
train_features = [tsne_feature[i*13099:(13099*i+9169)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

3.0 Splitting fourier features into train & test

In [108]:

```
fourier_features_train = pd.DataFrame(columns=['Amp_1', 'Amp_2', 'Amp_3', 'Amp_4', 'Amp_5', 'Freq_1', 'Freq_2', 'Freq_3',
                                              'Freq_4', 'Freq_5'])
fourier_features_test = pd.DataFrame(columns=['Amp_1', 'Amp_2', 'Amp_3', 'Amp_4', 'Amp_5', 'Freq_1', 'Freq_2', 'Freq_3',
                                              'Freq_4', 'Freq_5'])

for i in range(40):
    fourier_features_train = fourier_features_train.append(fourier_features[i*13099 : 13099*i + 9169])

fourier_features_train.reset_index(inplace = True)

for j in range(40):
    fourier_features_test = fourier_features_test.append(fourier_features[j*13099 + 9169 : 13099*(j+1)])

fourier_features_test.reset_index(inplace = True)
```

In [109]:

```
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 40 Number of data points in trian data 9169 Each data point contains 5 features
Number of data clusters 40 Number of data points in test data 3930 Each data point contains 5 features

In [110]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
tsne_train_flat_lat = [i[:9169] for i in tsne_lat]
tsne_train_flat_lon = [i[:9169] for i in tsne_lon]
tsne_train_flat_weekday = [i[:9169] for i in tsne_weekday]
tsne_train_flat_output = [i[:9169] for i in output]
tsne_train_flat_exp_avg = [i[:9169] for i in predict_list]
tsne_train_flat_tes_avg = [i[:9169] for i in tes_list]
```

In [111]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) for our test data
tsne_test_flat_lat = [i[9169:] for i in tsne_lat]
tsne_test_flat_lon = [i[9169:] for i in tsne_lon]
tsne_test_flat_weekday = [i[9169:] for i in tsne_weekday]
tsne_test_flat_output = [i[9169:] for i in output]
tsne_test_flat_exp_avg = [i[9169:] for i in predict_list]
tsne_test_flat_tes_avg = [i[9169:] for i in tes_list]
```

In [112]:

```
# 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(0,40):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,40):
    test_new_features.extend(test_features[i])
```

In [113]:

```
# 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,[])
tsne_train_tes_avg = sum(tsne_train_flat_tes_avg,[])
```

In [114]:

```
# converting lists of lists into single list i.e flatten
# a = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 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,[])
tsne_test_tes_avg = sum(tsne_test_flat_tes_avg,[])
```

In [115]:

```
# 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
df_train['triple_exp'] = tsne_train_tes_avg

print(df_train.shape)
```

(366760, 10)

In [117]:

```
# Preparing the data frame for our train data
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
df_test['triple_exp'] = tsne_test_tes_avg

print(df_test.shape)
```

(157200, 10)

In [118]:

df_test.head()

Out[118]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp
0	143	145	119	113	124	40.776228	-73.982119	4	121	111.270329
1	145	119	113	124	121	40.776228	-73.982119	4	120	109.890526
2	119	113	124	121	131	40.776228	-73.982119	4	127	103.052565
3	113	124	121	131	110	40.776228	-73.982119	4	115	104.410382
4	124	121	131	110	116	40.776228	-73.982119	4	115	118.256624

4.0 Concatenating dataframe of fourier features

In [121]:

```
df_train_final = pd.concat([df_train, fourier_features_train], axis = 1)
df_test_final = pd.concat([df_test, fourier_features_test], axis = 1)
```

In [122]:

```
print(df_train_final.shape)
df_train_final.head()
```

(366760, 21)

Out[122]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp	...
0	0	0	0	0	0	40.776228	-73.982119	4	0	8.245475	...
1	0	0	0	0	0	40.776228	-73.982119	4	0	7.045487	...
2	0	0	0	0	0	40.776228	-73.982119	4	0	5.408308	...
3	0	0	0	0	0	40.776228	-73.982119	4	0	2.349447	...
4	0	0	0	0	0	40.776228	-73.982119	4	0	3.437864	...

5 rows × 21 columns

In [123]:

```
print(df_test_final.shape)
df_test_final.head()
```

(157200, 21)

Out[123]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp	...
0	143	145	119	113	124	40.776228	-73.982119	4	121	111.270329	...
1	145	119	113	124	121	40.776228	-73.982119	4	120	109.890526	...
2	119	113	124	121	131	40.776228	-73.982119	4	127	103.052565	...
3	113	124	121	131	110	40.776228	-73.982119	4	115	104.410382	...
4	124	121	131	110	116	40.776228	-73.982119	4	115	118.256624	...

5 rows × 21 columns

In [124]:

```
# saving the data frames
df_train_final.to_csv('df_train_final.csv')
df_test_final.to_csv('df_test_final.csv')
```

In [135]:

```
with open("tsne_train_output.txt", "wb") as fp:    #Pickling
    pickle.dump(tsne_train_output, fp)
```

In [136]:

```
with open("tsne_test_output.txt", "wb") as fp1:    #Pickling
    pickle.dump(tsne_test_output, fp1)
```

ML models

1.0 Linear Regression with Hyperparameter tuning

In [127]:

```
# find more about LinearRegression function here http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LinearRegression.html
# -----
# default parameters
# sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)

# some of methods of LinearRegression()
# fit(X, y[, sample_weight]) Fit linear model.
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict using the linear model
# score(X, y[, sample_weight]) Returns the coefficient of determination R^2 of the prediction.
# set_params(**params) Set the parameters of this estimator.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geometric-intuition-1-2-copy-8/
# -----
```

#<https://stats.stackexchange.com/questions/153131/gridsearchcv-regression-vs-linear-regression-vs-stats-model-ols>

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV

model= LinearRegression(n_jobs=-1)
parameters = {'fit_intercept':[True,False], 'normalize':[True,False]}
grid = GridSearchCV(model, parameters, scoring='neg_mean_absolute_error', cv=3, n_jobs=-1)
grid.fit(df_train_final, tsne_train_output)
```

Out[127]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
            estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                       n_jobs=-1, normalize=False),
            iid='warn', n_jobs=-1,
            param_grid={'fit_intercept': [True, False],
                        'normalize': [True, False]},
            pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
            scoring='neg_mean_absolute_error', verbose=0)
```

In [128]:

```
grid.best_estimator_
```

Out[128]:

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=-1, normalize=True)
```

1.1 Applying the best hyperparameters

In [130]:

```
lr_reg = LinearRegression(fit_intercept= False, normalize= True, n_jobs = -1)
lr_reg.fit(df_train_final, tsne_train_output)
y_pred = lr_reg.predict(df_test_final)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = lr_reg.predict(df_train_final)
lr_train_predictions = [round(value) for value in y_pred]
```

1.2 MAPE results

In [131]:

```
tr_mape= (mean_absolute_error(tsne_train_output, lr_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output))
tr_mape
```

Out[131]:

0.11910550314977465

In [132]:

```
te_mape= (mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output))
te_mape
```

Out[132]:

0.10861682425280204

In [133]:

```
print('Train MAPE for linear regression model =', tr_mape)
print('Test MAPE for linear regression model =', te_mape)
```

Train MAPE for linear regression model = 0.11910550314977465
Test MAPE for linear regression model = 0.10861682425280204

2.0 Random Forest Regressor

2.1 Hyperparameter tuning

In [146]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

model = RandomForestRegressor(n_jobs=-1)
parameters = {'max_features':['sqrt', 'log2'],'min_samples_leaf':[1, 10, 100],'min_samples_split' : [3,5,7,8],
              'max_depth' : [3, 4, 5, 7], 'n_estimators':[20,40,80,100]}
rs = RandomizedSearchCV(model, parameters, cv=3, scoring='neg_mean_absolute_error',n_jobs=-1)
rs.fit(df_train_final, tsne_train_output)
```

Out[146]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                    estimator=RandomForestRegressor(bootstrap=True,
                                                    criterion='mse',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=
0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_lea
f=0.0,
                                                    n_estimators='warn',
                                                    n_jobs=-1, oob_score=Fa
lse,
                                                    random_state=None, verb
ose=0,
                                                    warm_start=False),
                    iid='warn', n_iter=10, n_jobs=-1,
                    param_distributions={'max_depth': [3, 4, 5, 7],
                                         'max_features': ['sqrt', 'log2'],
                                         'min_samples_leaf': [1, 10, 100],
                                         'min_samples_split': [3, 5, 7, 8],
                                         'n_estimators': [20, 40, 80, 10
0]},
                    pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score=False, scoring='neg_mean_absolute_er
ror',
                    verbose=0)
```

In [147]:

```
rs.best_estimator_
```

Out[147]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=7,
                     max_features='log2', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=100, min_samples_split=3,
                     min_weight_fraction_leaf=0.0, n_estimators=80, n_job
s=-1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

2.2 Applying the best hyperparameters

In [148]:

```
regr1 = RandomForestRegressor(max_features='log2', min_samples_leaf=100, min_samples_split=3, n_estimators=80, max_depth=7, n_jobs=-1)
regr1.fit(df_train_final, tsne_train_output)
```

Out[148]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=7,
                      max_features='log2', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=100, min_samples_split=3,
                      min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=-1,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False)
```

In [149]:

```
# Predicting on test data using our trained random forest model

y_pred = regr1.predict(df_test_final)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr1.predict(df_train_final)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [150]:

```
#feature importances based on analysis using random forest
print (df_train_final.columns)
print (regr1.feature_importances_)

Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg', 'triple_exp', 'index', 'Amp_1', 'Amp_2', 'Amp_3', 'Amp_4',
       'Amp_5', 'Freq_1', 'Freq_2', 'Freq_3', 'Freq_4', 'Freq_5'],
      dtype='object')
[6.64929309e-02 8.73741101e-02 7.24465127e-02 1.16010829e-01
 1.37786932e-01 8.38425622e-05 7.27935204e-04 4.72581827e-06
 1.82732800e-01 2.98808688e-01 3.51378557e-04 1.62507743e-02
 9.69810024e-03 7.57365930e-03 2.46160255e-03 1.16849478e-03
 3.56853611e-06 1.58742227e-05 3.82049240e-06 8.67107293e-07
 2.55362828e-06]
```

2.3 MAPE results

In [151]:

```
tr_mape= (mean_absolute_error(tsne_train_output, rndf_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output))
tr_mape
```

Out[151]:

0.12088704635322765

In [152]:

```
te_mape= (mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output))
te_mape
```

Out[152]:

```
0.11484218010239261
```

In [153]:

```
print('Train MAPE for Random Forest model =', tr_mape)
print('Test MAPE for Random Forest model =', te_mape)
```

```
Train MAPE for Random Forest model = 0.12088704635322765
Test MAPE for Random Forest model = 0.11484218010239261
```

3.0 XGBoost Regressor

3.1 Hyperparameter tuning

In [157]:

```
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV

model = XGBRegressor(n_jobs = -1)
parameters = {'learning_rate': [0.01,0.1,1], 'subsample':[0.7, 0.8, 0.9], 'min_child_weight':[3, 5, 6], 'reg_lambda':[200, 300, 400],
              'max_depth': [3, 4, 5], 'n_estimators': [500, 1000, 1500]}
rs1 = RandomizedSearchCV(model, parameters, scoring = 'neg_mean_absolute_error', cv = 3
)
rs1.fit(df_train_final, tsne_train_output)
```


[21:22:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:li
near is now deprecated in favor of reg:squarederror.

Out[157]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                    estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                    e',
                               colsample_bytree=1,
                               colsample_bynode=1,
                               colsample_bylevel=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',
                               random_state=42,
                               verbosity=1),
                    iid='warn', n_iter=10, n_jobs=None,
                    param_distributions={'learning_rate': [0.01, 0.1, 1],
                                         'max_depth': [3, 4, 5],
                                         'min_child_weight': [3, 5, 6],
                                         'n_estimators': [500, 1000, 1500],
                                         'reg_lambda': [200, 300, 400],
                                         'subsample': [0.7, 0.8, 0.9]},
                    pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score=False, scoring='neg_mean_absolute_error',
                    verbose=0)
```

In [158]:

```
rs1.best_estimator_
```

Out[158]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=5, min_child_weight=6, missing=None, n_estimators=1000,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=200, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.8, verbosity=1)
```

3.2 Applying the best hyperparameters

In [159]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

x_model = xgb.XGBRegressor(
    learning_rate =0.01,
    n_estimators=1000,
    max_depth=5,
    min_child_weight=6,
    gamma=0,
    subsample=0.8,
    reg_alpha=0, reg_lambda=200,
    colsample_bytree=1,nthread=None, n_jobs = -1)
x_model.fit(df_train_final, tsne_train_output)
```

[21:30:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

Out[159]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=5, min_child_weight=6, missing=None, n_estimators=1
000,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state
=0,
             reg_alpha=0, reg_lambda=200, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.8, verbosity=1)
```

In [160]:

```
#predicting with our trained Xg-Boost regressor
# the models x_model is already hyper parameter tuned
# the parameters that we got above are found using grid search

y_pred = x_model.predict(df_test_final)
xgb_test_predictions = [round(value) for value in y_pred]
y_pred = x_model.predict(df_train_final)
xgb_train_predictions = [round(value) for value in y_pred]
```

In [171]:

```
#feature importances
x_model.get_booster().get_score(importance_type='weight')
```

Out[171]:

```
{'Amp_1': 1645,
'Amp_2': 607,
'Amp_4': 1055,
'Freq_1': 62,
'exp_avg': 1894,
'ft_1': 1960,
'ft_2': 1640,
'ft_3': 1911,
'ft_4': 2265,
'ft_5': 2432,
'index': 1399,
'lat': 547,
'lon': 573,
'triple_exp': 6477,
'weekday': 276}
```

3.3 MAPE results

In [162]:

```
tr_mape= (mean_absolute_error(tsne_train_output, xgb_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output))
tr_mape
```

Out[162]:

```
0.10501706666425364
```

In [163]:

```
te_mape= (mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output))
te_mape
```

Out[163]:

```
0.10119894863020236
```

In [164]:

```
print('Train MAPE for XGBoost model =', tr_mape)
print('Test MAPE for XGBoost model =', te_mape)
```

```
Train MAPE for XGBoost model = 0.10501706666425364
Test MAPE for XGBoost model = 0.10119894863020236
```

4.0 Calculating the error metric values for baseline models & special features

In [165]:

```
r1= (mean_absolute_error(tsne_train_output,df_train_final['ft_1'].values))/(sum(tsne_train_output)/len(tsne_train_output))
r2= (mean_absolute_error(tsne_train_output,df_train_final['exp_avg'].values))/(sum(tsne_train_output)/len(tsne_train_output))
r3= (mean_absolute_error(tsne_train_output,df_train_final['triple_exp'].values))/(sum(tsne_train_output)/len(tsne_train_output))

r4= (mean_absolute_error(tsne_test_output, df_test_final['ft_1'].values))/(sum(tsne_test_output)/len(tsne_test_output))
r5= (mean_absolute_error(tsne_test_output, df_test_final['exp_avg'].values))/(sum(tsne_test_output)/len(tsne_test_output))
r6= (mean_absolute_error(tsne_test_output, df_test_final['triple_exp'].values))/(sum(tsne_test_output)/len(tsne_test_output))
```

In [167]:

```
print(r1,r2,r3,r4,r5,r6)
```

```
0.14870666996426116 0.14121603560900353 0.1281986037515331 0.1422552260104
1551 0.13490049942819257 0.12058563943338989
```

5.0 Results & Conclusion

Results summary

In [2]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Regressor", "Train MAPE", "Test MAPE"]
x.add_row(["Base line model", '14.87%', '14.22%'])
x.add_row(["Exponential Moving Averages", '14.12%', '13.49%'])
x.add_row(["Triple exponential smoothing", '12.81%', '12.05%'])
x.add_row(["Linear regression(with hyperparameter tuning)", '11.91%', '10.86%'])
x.add_row(["Random forest(with hyperparameter tuning)", '12.08%', '11.48%'])
x.add_row(["XGBoost(with hyperparameter tuning)", '10.50%', '10.11%'])

print(x)
```

Regressor	Train MAPE	Test MAPE
Base line model	14.87%	14.22%
Exponential Moving Averages	14.12%	13.49%
Triple exponential smoothing	12.81%	12.05%
Linear regression(with hyperparameter tuning)	11.91%	10.86%
Random forest(with hyperparameter tuning)	12.08%	11.48%
XGBoost(with hyperparameter tuning)	10.50%	10.11%

Conclusion

- Addition of the special feature "Triple Exponential Smoothing a.k.a Holt-Winters Method" proved to be very useful as the test MAPE as a result of the feature was the lowest compared to baseline model & the Exponential Moving Averages
- XGBoost regressor provided the best & the least test MAPE of 10.11%. Next best model was linear regression with test MAPE of 10.86%