Taxi demand prediction in New York City

Project Overview

There are roughly 200 million taxi rides in New York City each year. Exploiting an understanding of taxi supply and demand could increase the efficiency of the city's taxi system. In the New York city, people use taxi in a frequency much higher than any other cities of US. Instead of booking a taxi by phone one day ahead of time, New York taxi drivers pick up passengers on street. The ability to predict taxi ridership could present valuable insights to city planners and taxi dispatchers in answering questions such as how to position cabs where they are most needed, how many taxis to dispatch, and how ridership varies over time. Our project focuses on predicting the number of taxi pickups given a time window and a region within New York City.

Problem Statement

This is a supervised learning problem where we need to predict the number of pickups given a 10 min time interval and a region in the New york city. Some location require more taxis at a particular time than other locations owing to the presence schools, hospitals, offices etc. The prediction result can be transferred to the taxi drivers via Smartphone app, and they can subsequently move to the locations where predicted pickups are high. The goal is to predict the no of pickups with minimum Mean Absolute percentage error.

Problem Type

It is a time series forecasting Regression problem.

Time series forecasting: It is a technique for the prediction of events through a sequence of time. The techniques predict future events by analyzing the trends of the past, on the assumption that future trends will hold similar to historical trends.

Coming to our problem every region of NYC has to be broken up into 10 min interval. We already know, about the pickup at time 't', we will predict the pickup at time 't+1' in the same region. Hence, this problem can be thought of as a 'Time Series Prediction' problem. It is a special case of regression problems because the output no of pickups is a continuous value. In short, we will use the data at time 't' to predict for time 't+1'.

In [1]:
```
import dask.dataframe as dd
import pandas as pd
import folium
import datetime
import time
import numpy as np
import matplotlib
matplotlib.use('nbagg')
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams
import gpxpy.geo
from sklearn.cluster import MiniBatchKMeans, KMeans
import math
import pickle
import scipy
import os
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']
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
```

Exploratory Data Analysis

Before building our model we will start by exploring our dataset. You can download the data from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml (http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml). To solve the problem we would be using data collected in Jan - Mar 2015 to predict the pickups in Jan - Mar 2016. Let's start by loading the Jan - 2015 data from CSV file to perform EDA.

Data Collection

<table align = 'left'>
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

</tr>

yellow_tripdata_2015-01

1.84Gb

12748986

19 </tr>

yellow_tripdata_2015-02

1.81Gb

12450521

19 </tr>

yellow_tripdata_2015-03

1.94Gb

13351609

19 </tr> </table>

Features

In [9]:
```
month = dd.read_csv('yellow_tripdata_2015-01.csv')
print(month.columns)
```

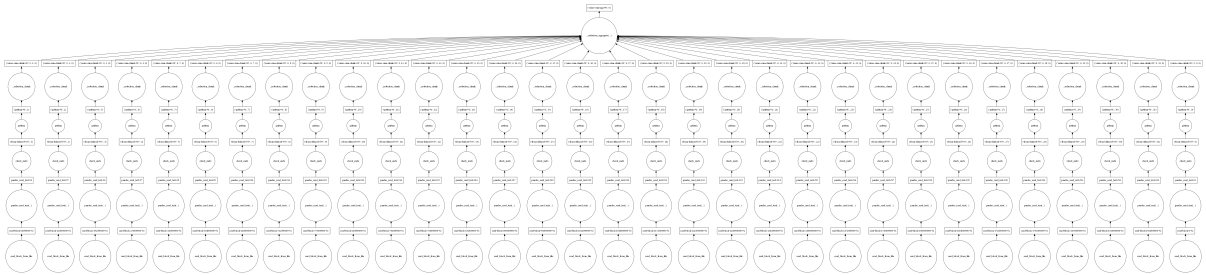
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amoun
t',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [3]:
```
month.visualize()
```

Out[3]:

In [4]:
```
month.fare_amount.sum().visualize()
```

Out[4]:



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 [5]:
```
month.head(5)
```

Out[5]:

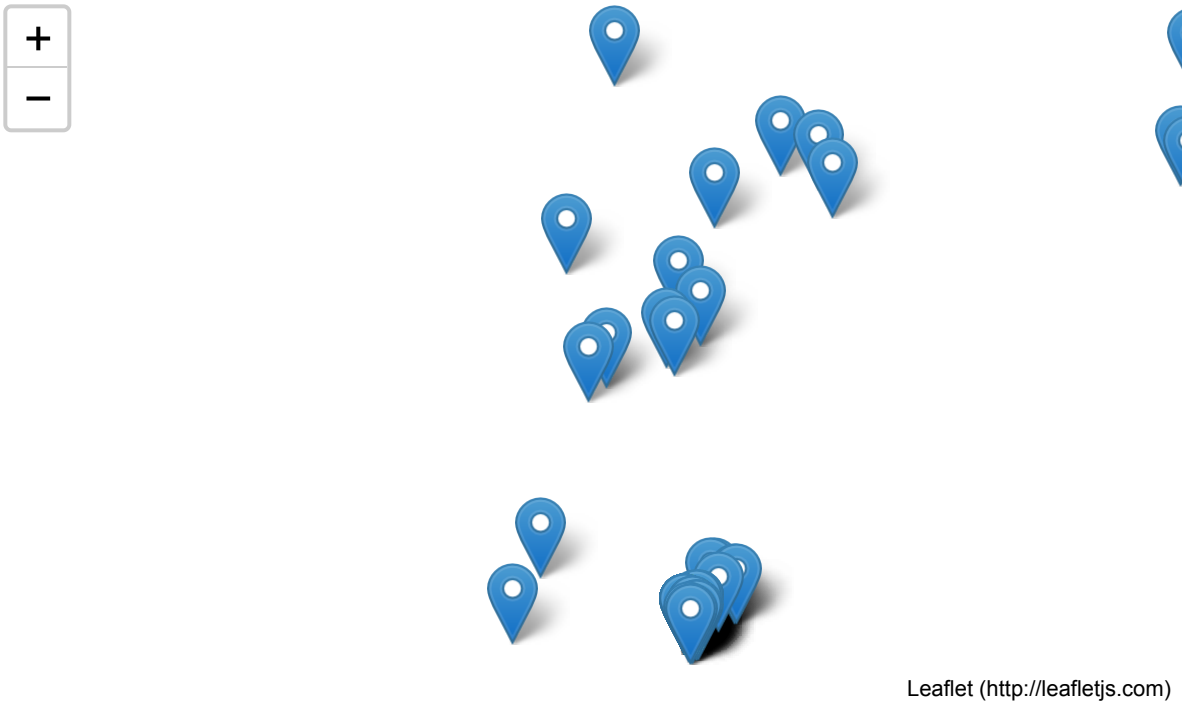
	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distar
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50
4	1	2015-01-10 20:33:39	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 cordinates(lat,long) - (40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are not considered by us as we are only concerned with pickups which originate within New York.

In [6]:
```
# Plotting pickup cordinates which are outside the bounding box of New-York
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_
latitude <= 40.5774)| \
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitu
de >= 40.9176))]
# creating a map with the a base location
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
sample_locations = outlier_locations.head(10000)
#print((sample_locations['pickup_latitude'] == 0).count())

for i,j in sample_locations.iterrows():
    if int(j['pickup_latitude']) != 0:
        folium.Marker(list((j['pickup_latitude'],j['pickup_longitude']))).add_
to(map_osm)
map_osm
```

Out[6]:



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

In [7]:
```
# Plotting dropoff cordinates which are outside the bounding box of New-York
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitude <= 40.5774)| \
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.9176))]

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

# we will spot only first 100 outliers on the map
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[7]:

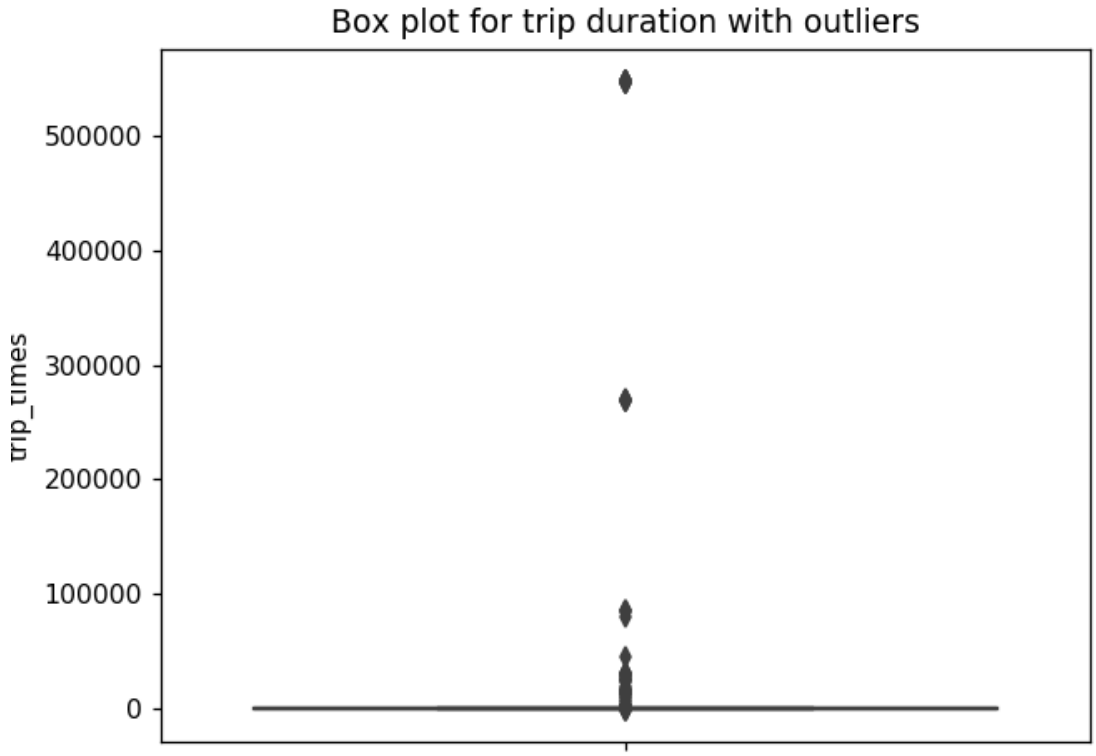
Observation:- The observations here are similar to those obtained while analysing pickup latitude and longitude

3. Trip Durations:

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

In [8]:
```
#The timestamps are converted to unix so as to get duration(trip-time) & speed
 also pickup-times in unix are used while binning

#In our data we have time in the formate "YYYY-MM-DD HH:MM:SS" we convert this
s sting to python time formate and then into unix time stamp

def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").time
tuple())

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_datet
ime'].values]
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_dateti
me'].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','p
ickup_latitude','dropoff_longitude','dropoff_latitude','total_amount']].comput
e()

    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
frame_with_durations = return_with_trip_times(month)
```

In [10]:
```
sns.boxplot(y="trip_times", data =frame_with_durations)
plt.title('Box plot for trip duration with outliers')
plt.show()
```



By the above box we cannot find correct outlier so we will calculate 0-100th percentile to find a the correct percentile value for removal of outliers

In [11]:
```
for i in range(0,100,10):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
#print("100 percentile value is",var[-1])
```

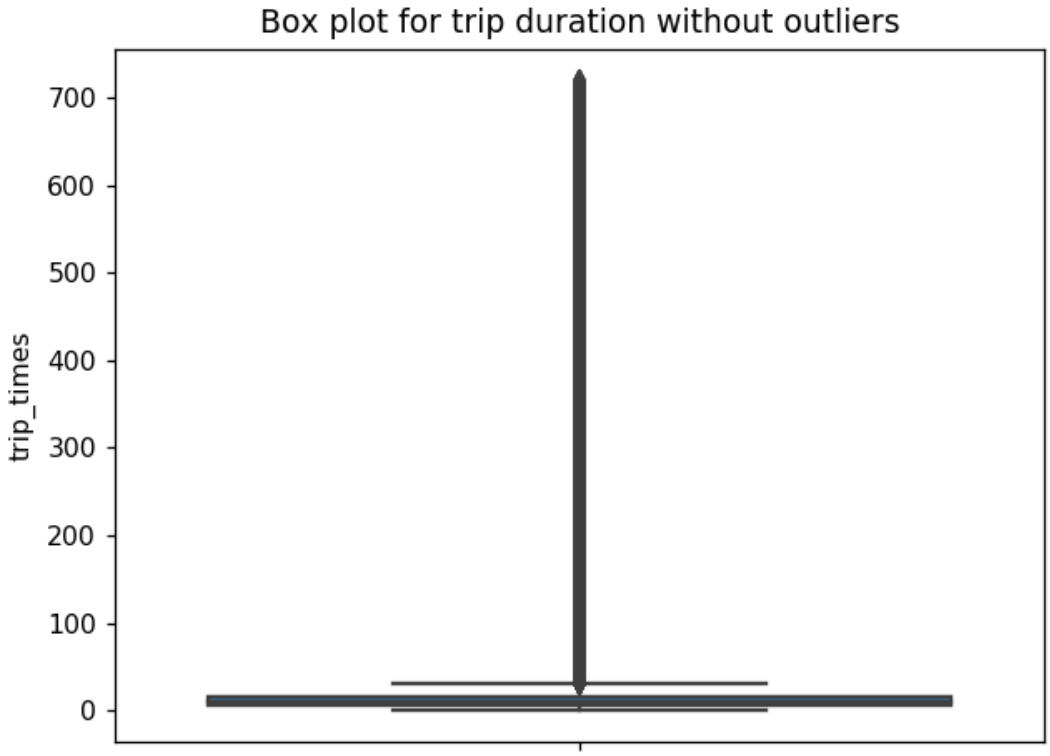
```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333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

In [12]:
```
#Zooming 90-100 percentile
for i in range(90,100):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
print ("100 percentile value is ",var[-1])
```

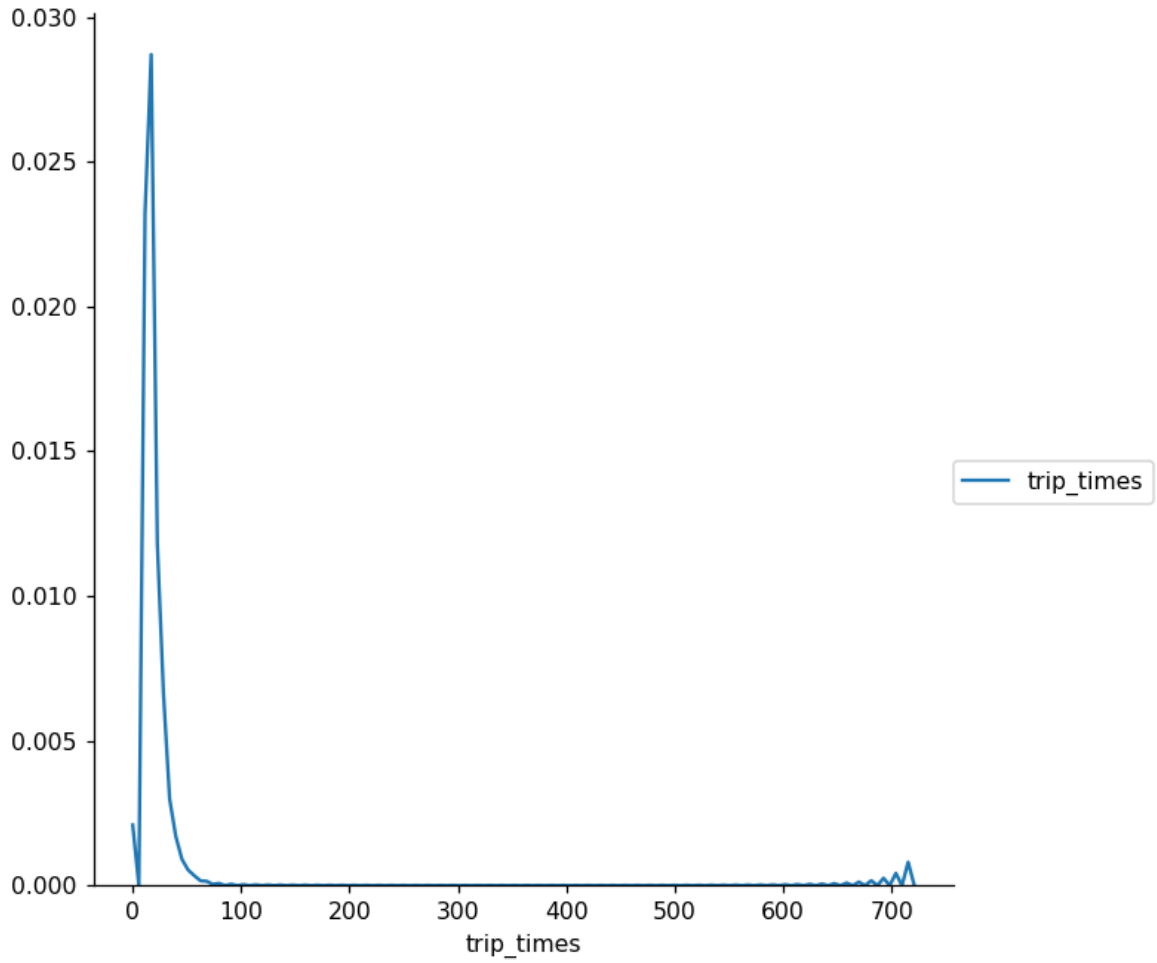
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.68333333333333
97 percentile value is 34.46666666666667
98 percentile value is 38.71666666666667
99 percentile value is 46.75
100 percentile value is  548555.6333333333
```

In [13]:
```
#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 [14]:
```
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.title('Box plot for trip duration without outliers')
plt.show()
```

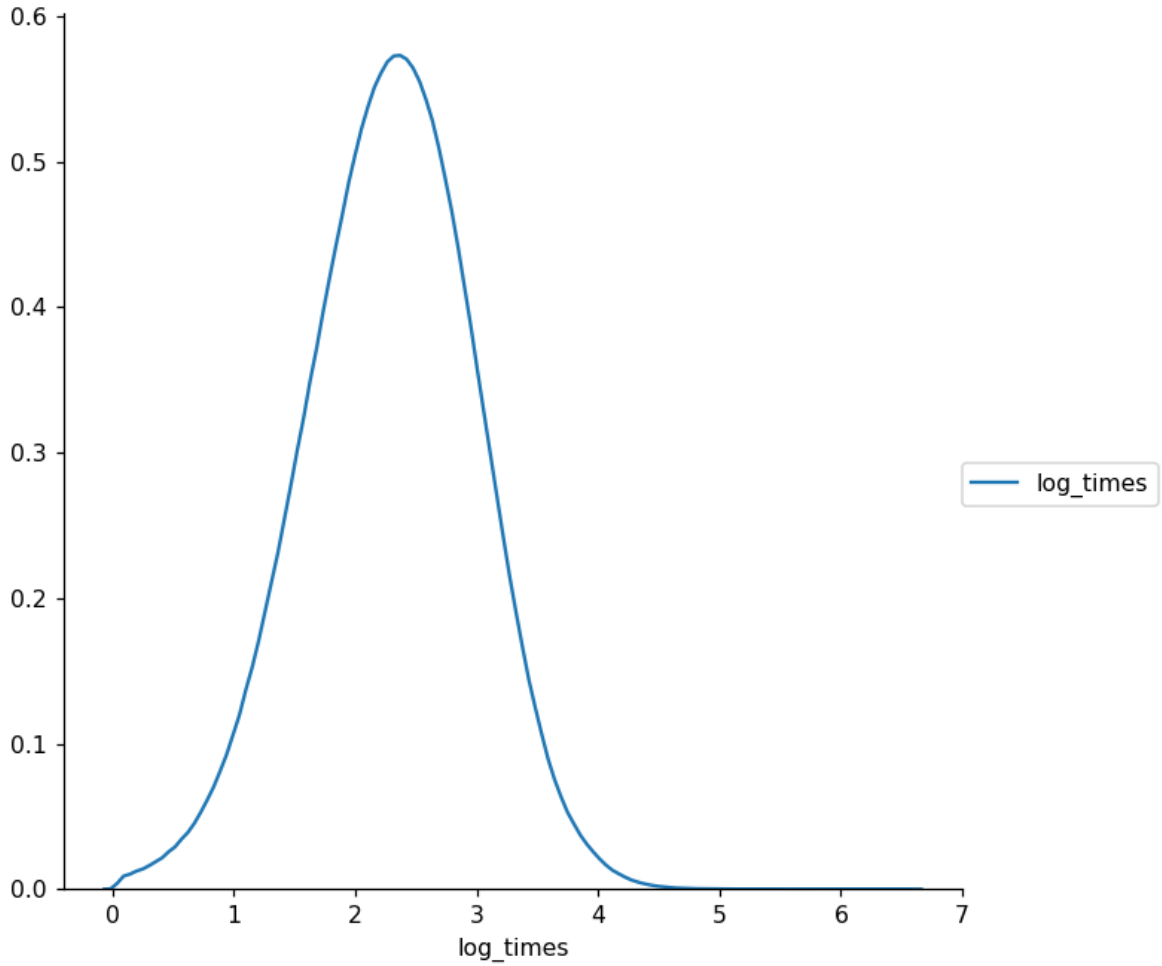


In [15]:
```
#pdf of trip-times after removing the outliers
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```

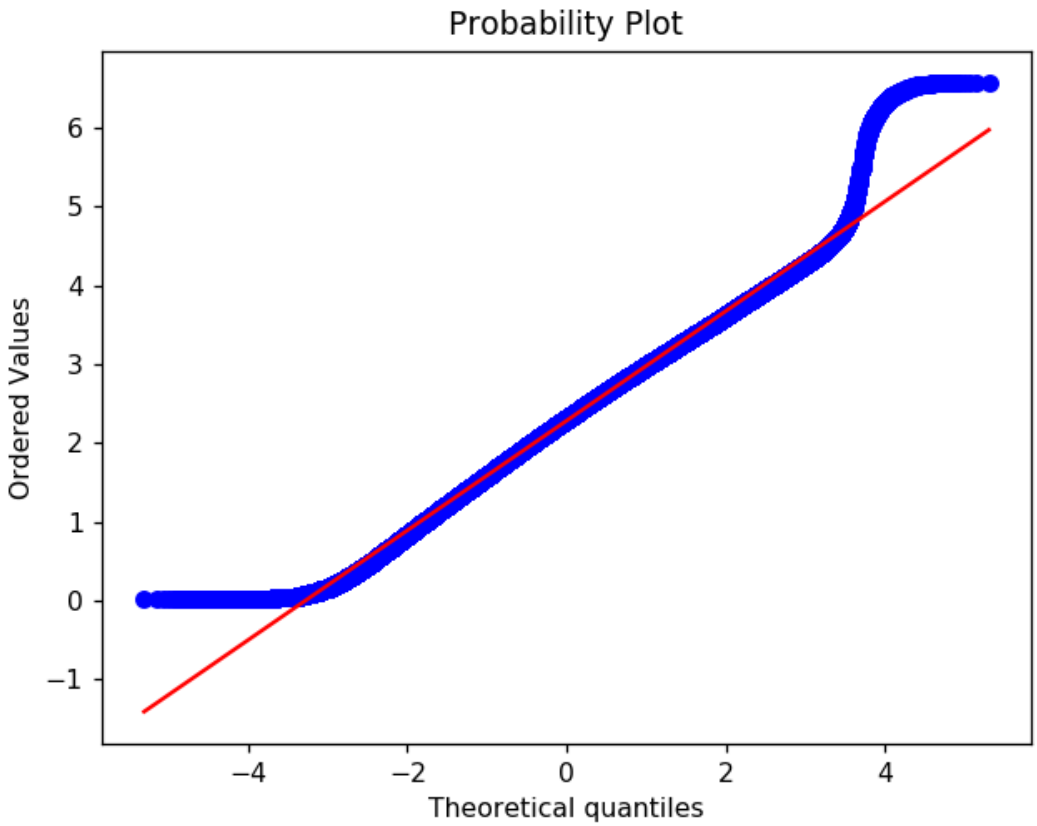


In [16]:
```
#converting the values to log-values to check for log-normal
frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_du
rations_modified['trip_times'].values]
```

In [17]:
```
#pdf of log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```

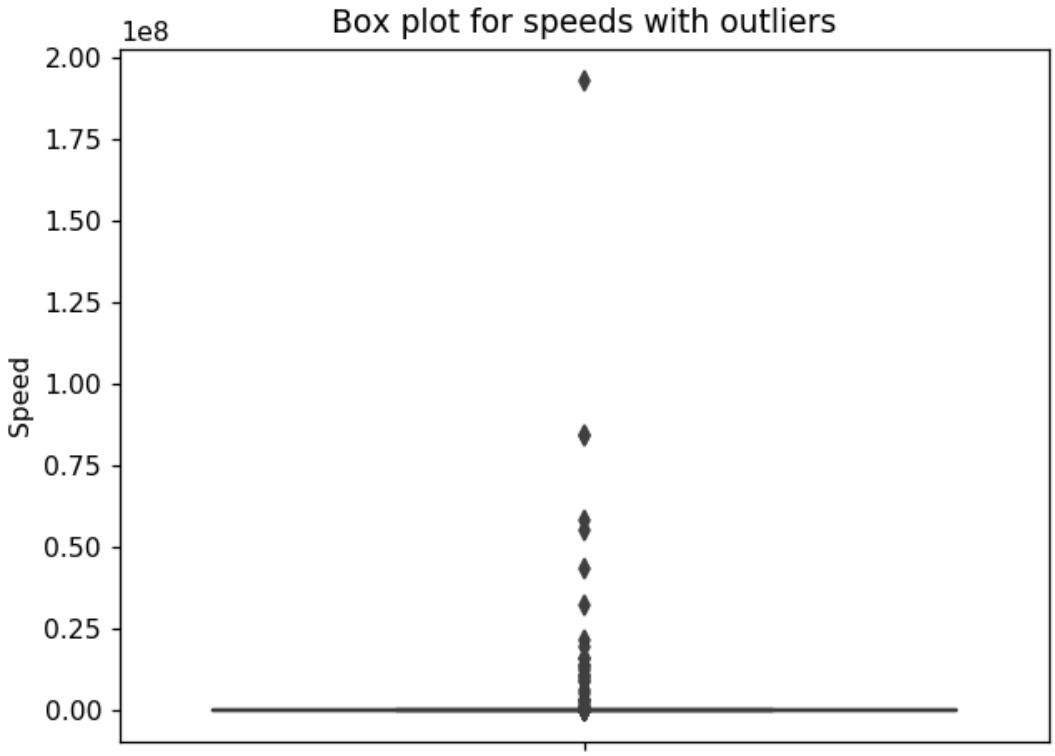


In [18]:
```
#Q-Q plot for checking if trip-times is log-normal
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=p
lt)
plt.show()
```



4. Speed

In [19]:
```
frame_with_durations_modified['Speed'] = 60*(frame_with_durations_modified['tr
ip_distance']/frame_with_durations_modified['trip_times'])
sns.boxplot(y="Speed", data =frame_with_durations_modified)
plt.title('Box plot for speeds with outliers')
plt.show()
```



In [20]:
```
#calculating speed values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
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 [21]:
```
#calculating speed values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
print("100 percentile value is ",var[-1])
```

```
90 percentile value is 20.186915887850468
91 percentile value is 20.916454400875093
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 35.75135055113604
100 percentile value is  192857142.85714284
```

In [22]:
```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.
6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,99+i)
    print("{} percentile value is {}".format(99+i,x))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 35.75135055113604
99.1 percentile value is 36.31081290376664
99.2 percentile value is 36.91470054446461
99.3 percentile value is 37.588235294117645
99.4 percentile value is 38.330334294788756
99.5 percentile value is 39.17580011612381
99.6 percentile value is 40.15384615384615
99.7 percentile value is 41.338029086798095
99.8 percentile value is 42.866243893093184
99.9 percentile value is 45.310675074725154
100 percentile value is  192857142.85714284
```

In [23]:
```
#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)]
```

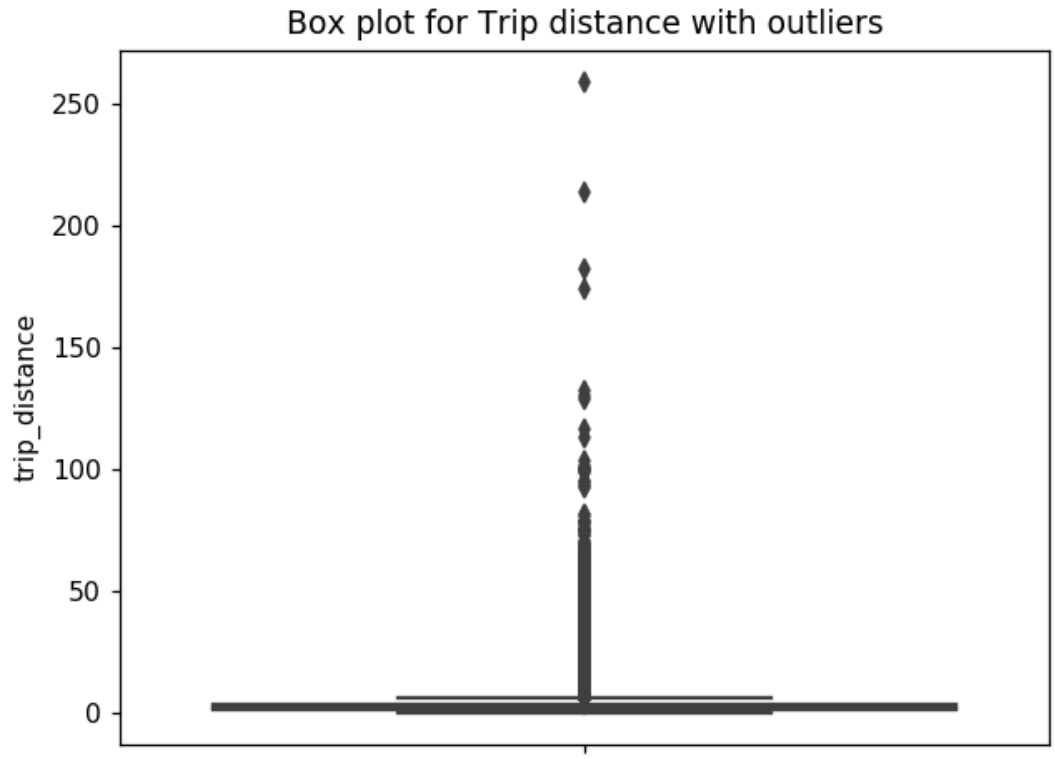
In [24]:
```
#avg.speed of cabs in New-York
sum(frame_with_durations_modified["Speed"]) / float(len(frame_with_durations_m
odified["Speed"]))
```

Out[24]: 12.450173996027528

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

5. Trip Distance

In [25]:
```
# lets try if there are any outliers in trip distances
sns.boxplot(y="trip_distance", data =frame_with_durations_modified)
plt.title('Box plot for Trip distance with outliers')
plt.show()
```



In [27]:
```
#calculating trip distance values at each percntile 0,10,20,30,40,50,60,70,80,
90,100
for i in range(0,100,10):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
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 [28]:
```
#calculating trip distance values at each percntile 90,91,92,93,94,95,96,97,9
8,99,100
for i in range(90,100):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
print("100 percentile value is ",var[-1])
```

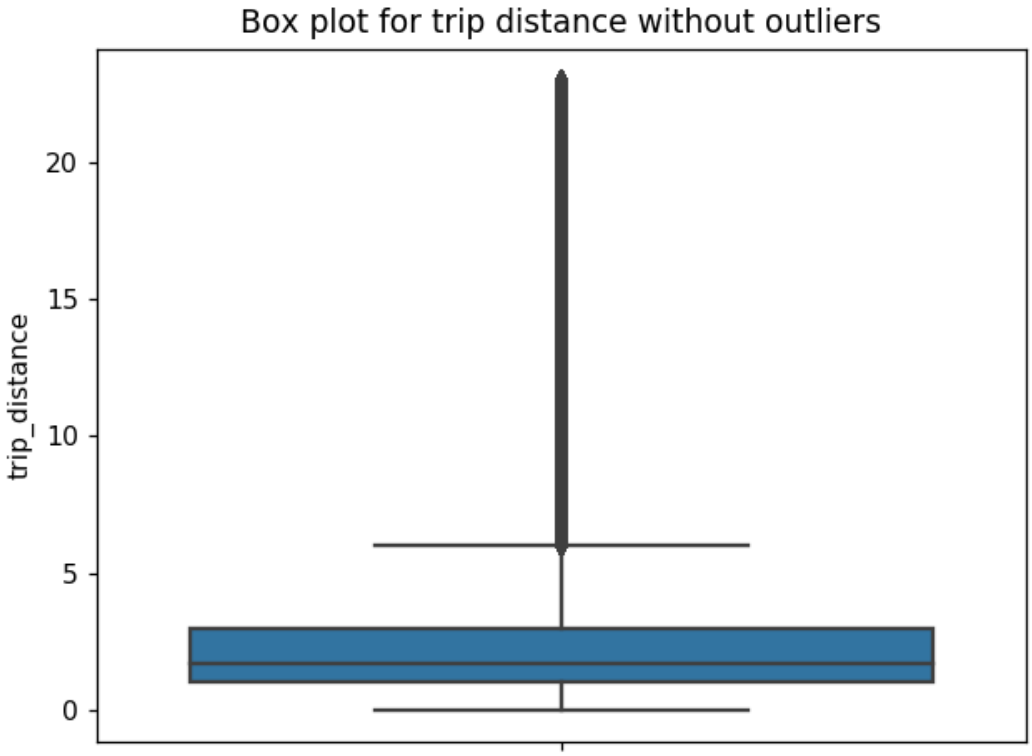
```
90 percentile value is 5.97
91 percentile value is 6.45
92 percentile value is 7.07
93 percentile value is 7.85
94 percentile value is 8.72
95 percentile value is 9.6
96 percentile value is 10.6
97 percentile value is 12.1
98 percentile value is 16.03
99 percentile value is 18.17
100 percentile value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

In [29]:
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

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

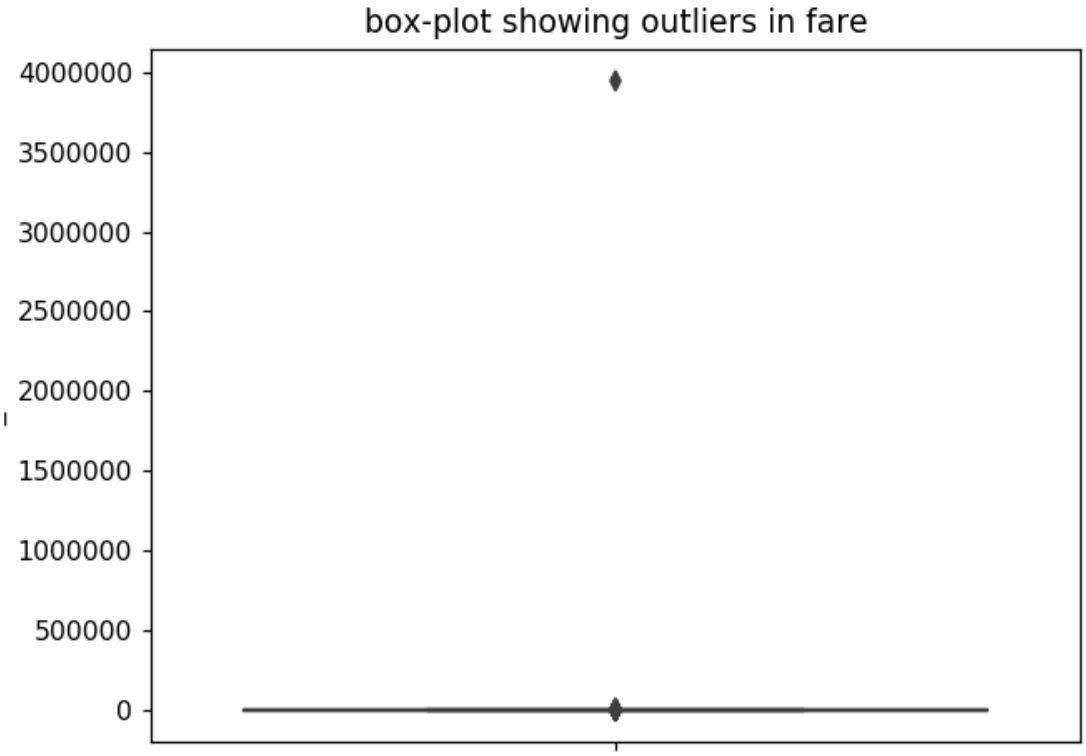
In [30]:
```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_distance>0) & (frame_with_durations.trip_distance<23)]
```

In [31]:
```
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.title('Box plot for trip distance without outliers')
plt.show()
```



5. Total Fare

In [32]:
```
# lets try if there are any outliers in based on the total_amount
sns.boxplot(y="total_amount", data =frame_with_durations_modified)
plt.title('box-plot showing outliers in fare')
plt.show()
```



In [33]:
```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is -242.55
10 percentile value is 6.3
20 percentile value is 7.799999999999999
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 [34]:
```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,i)
    print("{} percentile value is {}".format(i,x))
print("100 percentile value is ",var[-1])
```

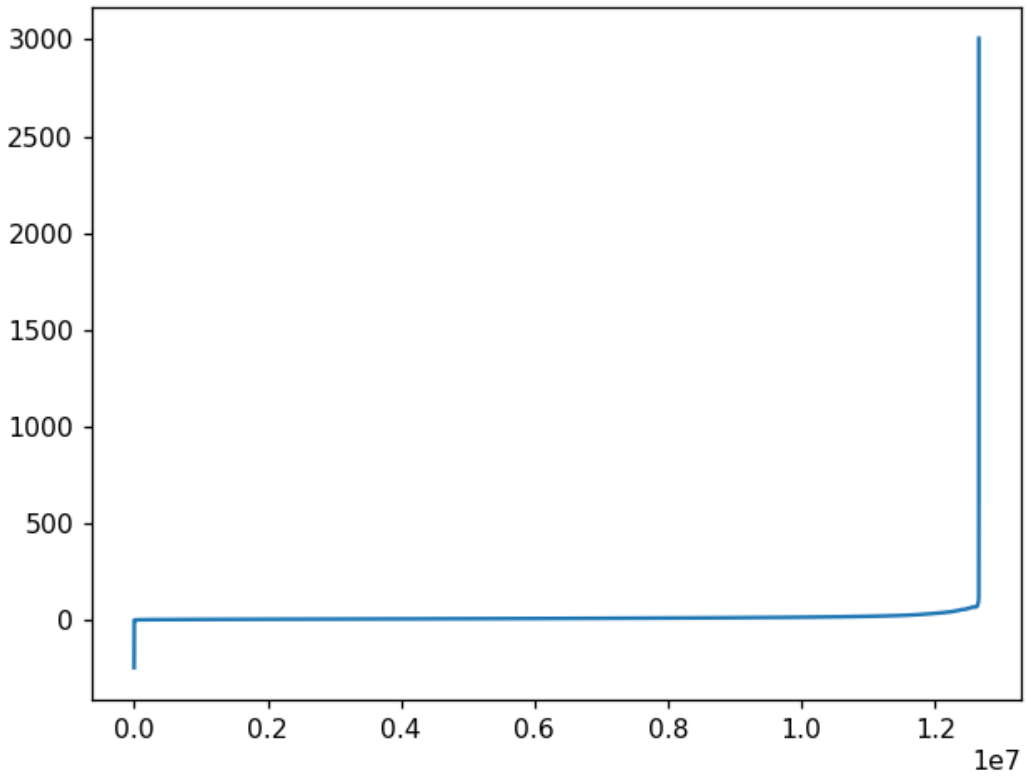
```
90 percentile value is 25.8
91 percentile value is 27.3
92 percentile value is 29.3
93 percentile value is 31.800000000000004
94 percentile value is 34.8
95 percentile value is 38.53
96 percentile value is 42.6
97 percentile value is 48.13
98 percentile value is 58.13
99 percentile value is 66.13
100 percentile value is  3950611.6
```

In [35]:
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    x = np.percentile(var,99+i)
    print("{} percentile value is {}".format(99+i,x))
print("100 percentile value is ",var[-1])
```

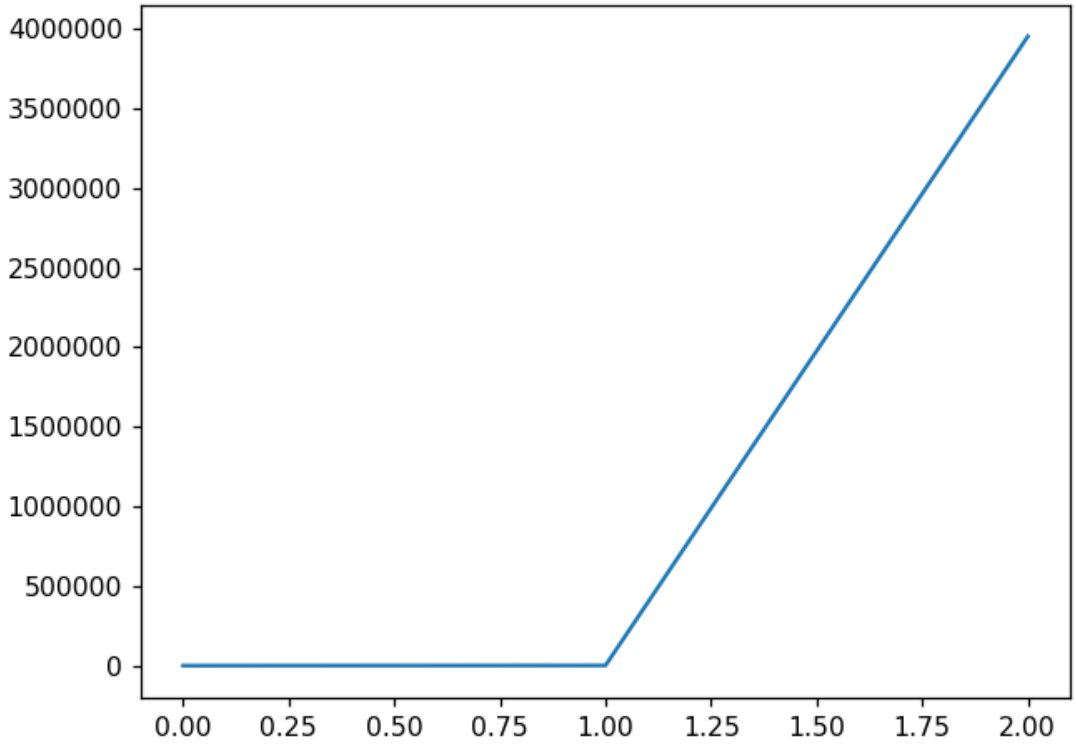
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value is 88.2722400000319
100 percentile value is  3950611.6
```

Observation:- As even the 99.9th percentile value doesnt 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 analyis

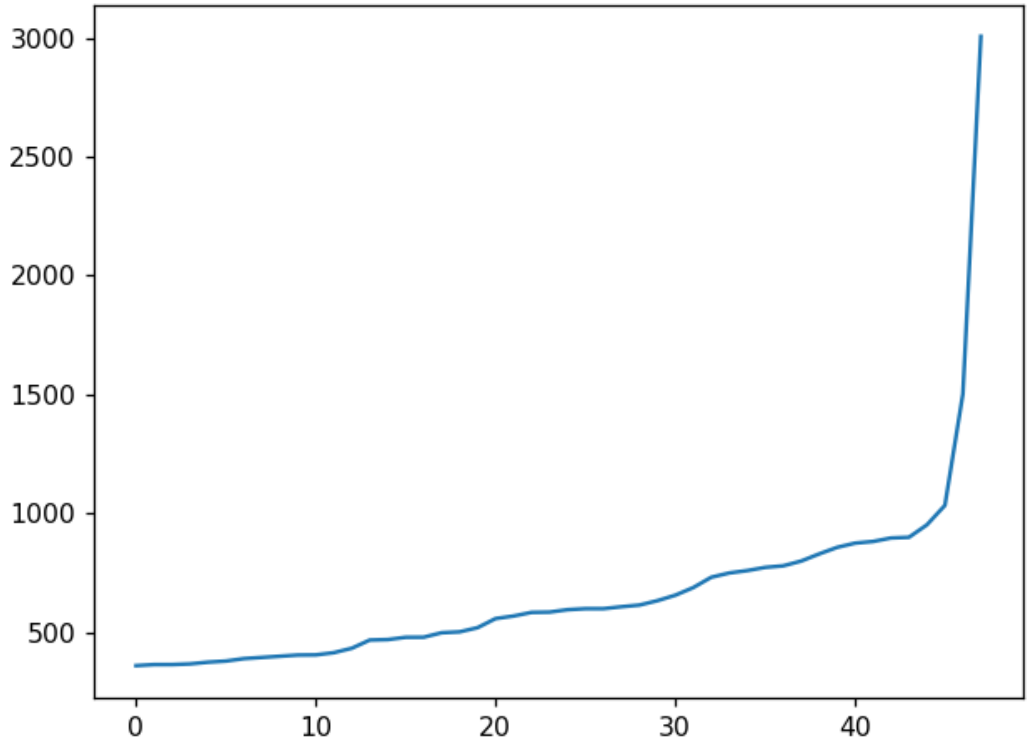
In [36]:
```
#below plot shows us the fare values(sorted) to find a sharp increase to remov
e those values as outliers
# plot the fare amount excluding last two values in sorted data
plt.plot(var[:-2])
plt.show()
```



In [37]:
```
# a very sharp increase in fare values can be seen
# plotting last three total fare values, and we can observe there is increase
 in the values
plt.plot(var[-3:])
plt.show()
```



In [38]:
```
#now looking at values not including the last two points we again find a drast
ic 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/erronous points.

In [39]:
```
#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_fra
me.dropoff_longitude <= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dr
opoff_latitude <= 40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pi
ckup_latitude >= 40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.p
ickup_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_dis
tance < 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_a
mount >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_fram
e.dropoff_longitude <= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dr
opoff_latitude <= 40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pi
ckup_latitude >= 40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.p
ickup_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_dist
ance < 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_am
ount >0)]
```

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

In [40]:
```
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 0.970357642560749
5
```

Data-preperation

Clustering/Segmentation

In [41]:
```
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_lon
gitude']].values
neighbours=[]

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

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_sta
te=42).fit(coords)
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(f
rame_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 clust
er regions
#that are close to any cluster center
# and make sure that the minimum inter cluster should not be very less
for increment in range(10, 100, 10):
    cluster_centers, cluster_len = find_clusters(increment)
    find_min_distance(cluster_centers, cluster_len)
```

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 8.0
Min inter-cluster distance =  1.0945442325142543
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 16.0
Min inter-cluster distance =  0.7131298007387813
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 22.0
Min inter-cluster distance =  0.5185088176172206
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 32.0
Min inter-cluster distance =  0.5069768450363973
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 38.0
Min inter-cluster distance =  0.365363025983595
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 46.0
Min inter-cluster distance =  0.34704283494187155
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 54.0
Min inter-cluster distance =  0.30502203163244707
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 62.0
Min inter-cluster distance =  0.29220324531738534
---
On choosing a cluster size of  90
```

```
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 69.0
Min inter-cluster distance =  0.18257992857034985
---
```

Observation:

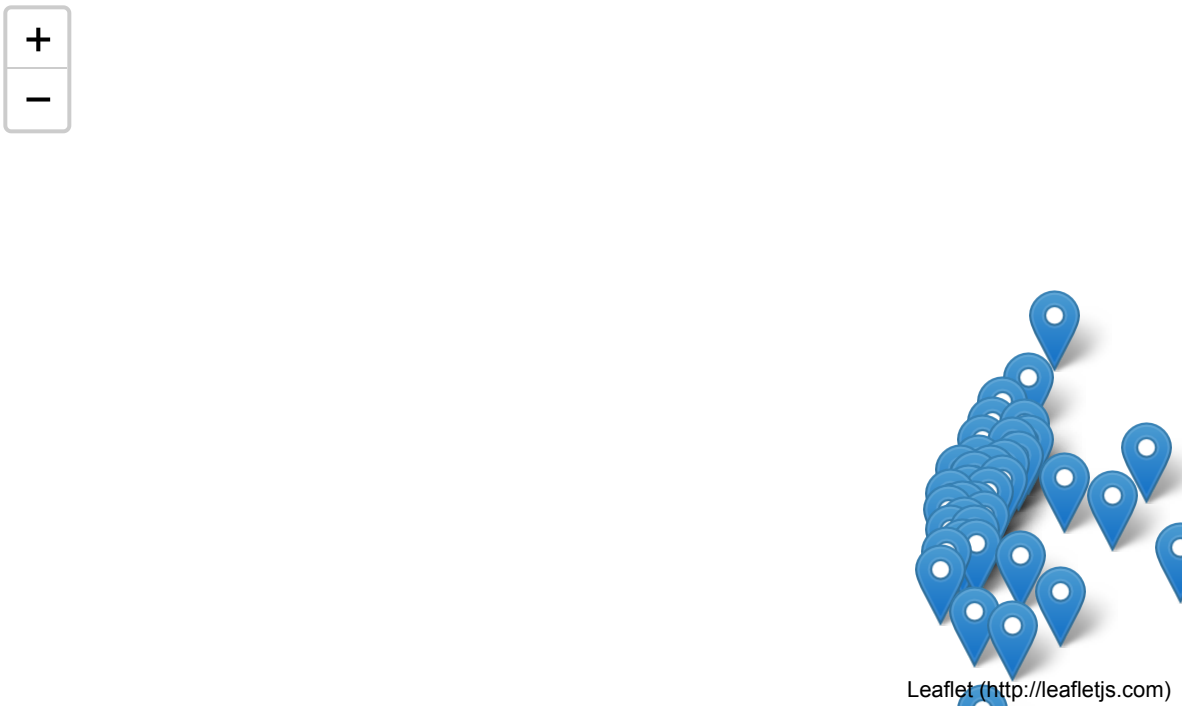
The main objective was to find a optimal min. distance(Which roughly estimates to the radius of a cluster) between the clusters which we got was 40. For the 50 clusters you can observe that there are two clusters with only 0.3 miles apart from each other. so we choose 40 clusters for solve the further problem

In [42]:
```
# Getting 40 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(c
oords)
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame
_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
```

Plotting the cluster centers:

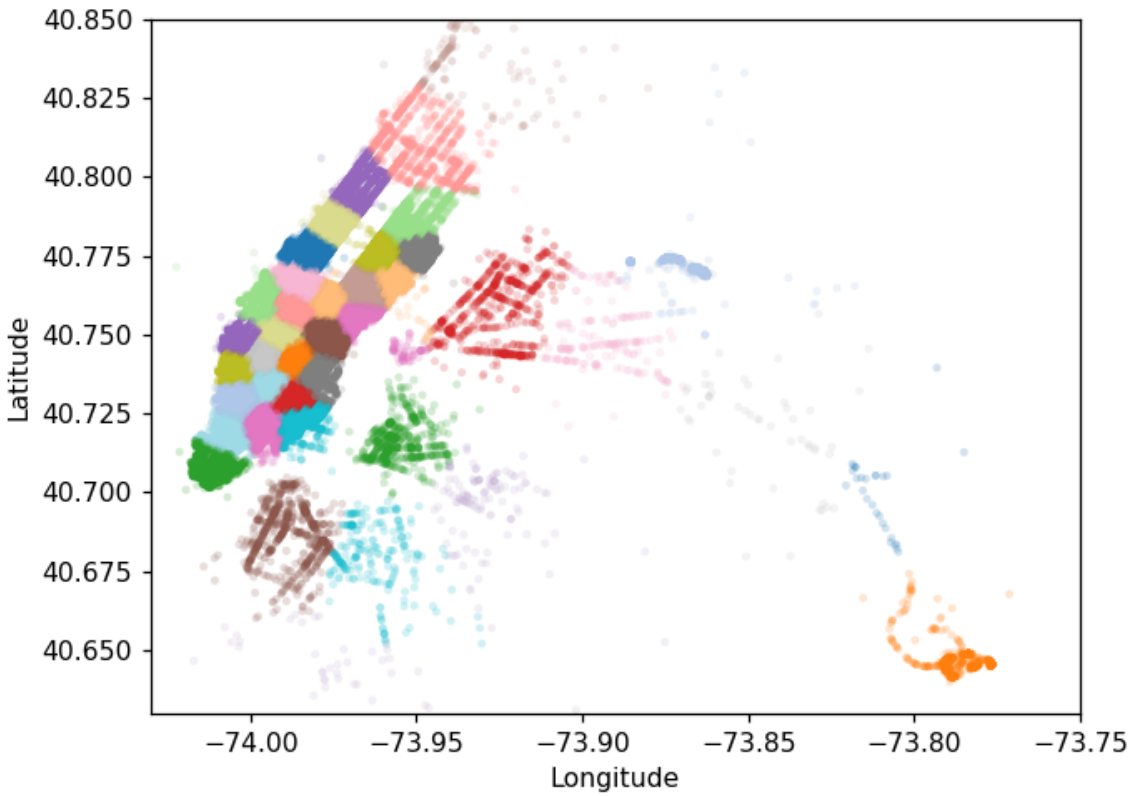
In [43]:
```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])), popup=(
str(cluster_centers[i][0])+str(cluster_centers[i][1]))).add_to(map_osm)
map_osm
```

Out[43]:



Plotting the clusters:

In [44]:
```
#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.v
alues[:100000], s=10, lw=0,
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.2
)
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [45]:
```
# Example - 1420070400 : 2015-01-01 00:00:00

def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,1433
116800],\
                  [1451606400,1454284800,1456790400,1459468800,1462060800,14
64739200]]
    start_pickup_unix=unix_times[year-2015][month-1]
    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 [46]:
```
# clustering, making pickup bins and grouping by pickup cluster and pickup bin
s
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_distan
ce']].groupby(['pickup_cluster','pickup_bins']).count()
```

In [47]:
```
# we add two more columns 'pickup_cluster'(to which cluster it belogns to)
# and 'pickup_bins' (to which 10min interval the trip belongs to)
jan_2015_frame.head()
```

Out[47]:

	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 [48]:
```
# here 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 divide whole months time into 10min intrav
els 24*31*60/10 =4464bins)
jan_2015_groupby.head()
```

Out[48]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**1**	105
	2	199
	3	208
	4	141
	5	155

In [49]:
```
# upto now we cleaned data and prepared data for the year 2015,
# now do the same operations for months Jan, Feb, March of 2016
# 1. get the dataframe which inlcudes only required colums
# 2. adding trip times, speed, unix time stamp of pickup_time
# 4. remove the outliers based on trip_times, speed, trip_duration, total_amou
nt
# 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 'pickuo_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_duratio
ns)

    print ("Estimating clusters..")
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(f
rame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.pre
dict(frame_with_durations_outliers_removed_2016[['pickup_latitude', 'pickup_lo
ngitude']])

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

Smoothing

In [50]:
```
# Gets the unique bins where pickup values are present for each each reigion

# for each cluster region we will collect all the indices of 10min intravels i
n which the pickups are happened
# we got an observation that there are some pickpbins that doesnt have any pic
kups
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 [51]:
```
# for every month we get all indices of 10min intravels in which atleast one p
ickup got happened

#jan
jan_2015_unique = return_unq_pickup_bins(jan_2015_frame)
jan_2016_unique = return_unq_pickup_bins(jan_2016_frame)

#feb
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [52]:
```
# 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:  41
---------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1986
---------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  30
---------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  355
---------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  38
---------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  154
---------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  35
---------------------------------------------------------------
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:  118
---------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  41
---------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  26
---------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  45
---------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  43
---------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  29
---------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  27
---------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  32
---------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  41
---------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  59
---------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  1191
---------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1358
---------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  54
---------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  30
---------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  30
---------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  164
---------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  36
---------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  42
---------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  32
---------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  215
---------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  37
```

```
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1181
------------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  43
------------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  45
------------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  43
------------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  322
------------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
```

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 [53]:
```
# 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 10
min intravel
# there wont be any value if there are no picksups.
# values: number of unique bins

# for every 10min intravel(pickup_bin) we will check it is there in our unique
 bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 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 [54]:
```
# 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 10 min intravel
# there wont be any value if there are no picksups.
# values: number of unique bins

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        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/resolved
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup bin if it exists
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin value which has a 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 missing,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/((right_hand_limit-i)+2)*1.0
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
```

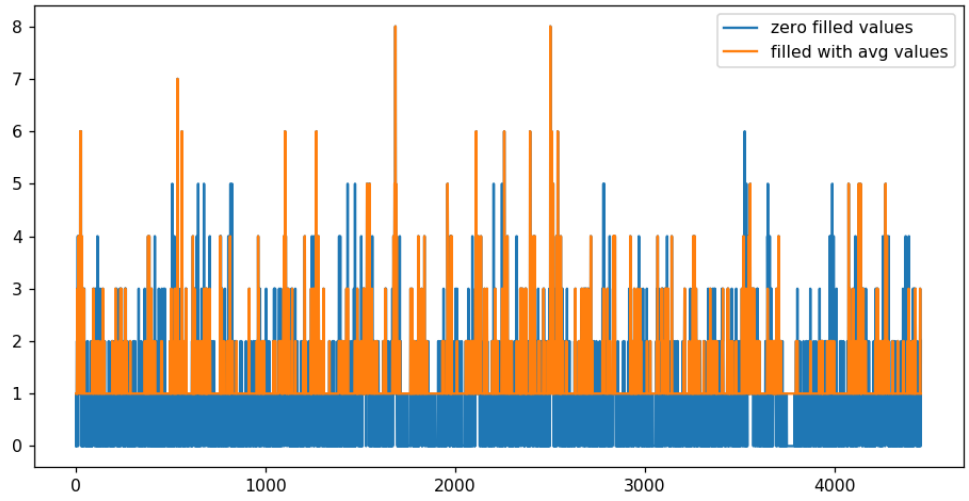
```
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found
 to be missing,hence we have no left-limit here
                    right_hand_limit=0
                    for j in range(i,4464):
            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

In [55]:
```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number o
f pickups that are happened
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015
_unique)

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_
unique)
```

In [56]:
```
# 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 (lengt
h of the jan_2015_fill)
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

```
number of 10min intravels among all the clusters  178560
```

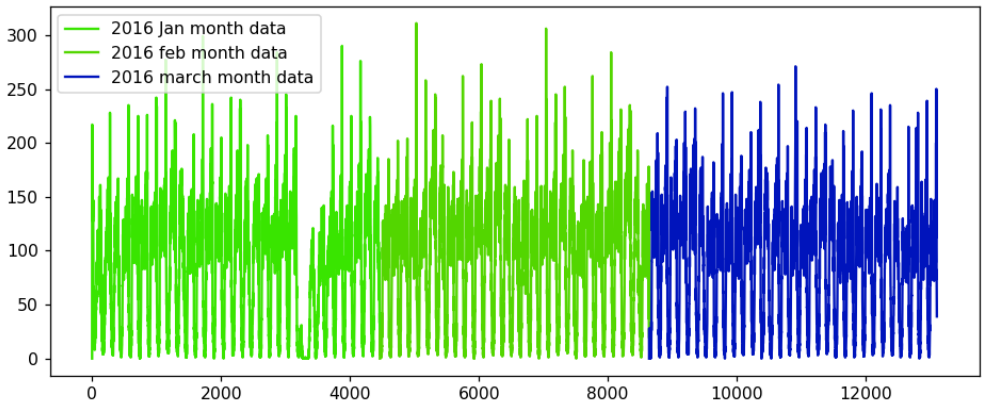
In [57]:
```
# Smoothing vs Filling
# sample plot that shows two variations of filling missing values
# we have taken the number of pickups for cluster region 2
plt.figure(figsize=(10,5))
plt.plot(jan_2015_fill[4464:8920], label="zero filled values")
plt.plot(jan_2015_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```

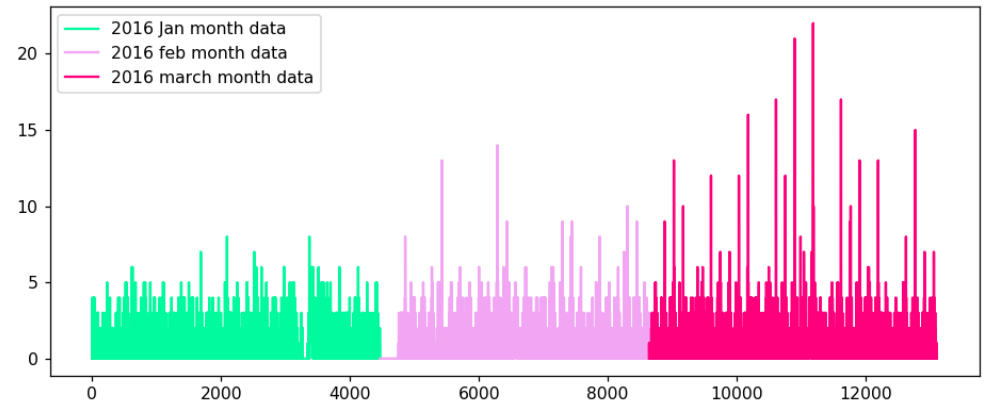


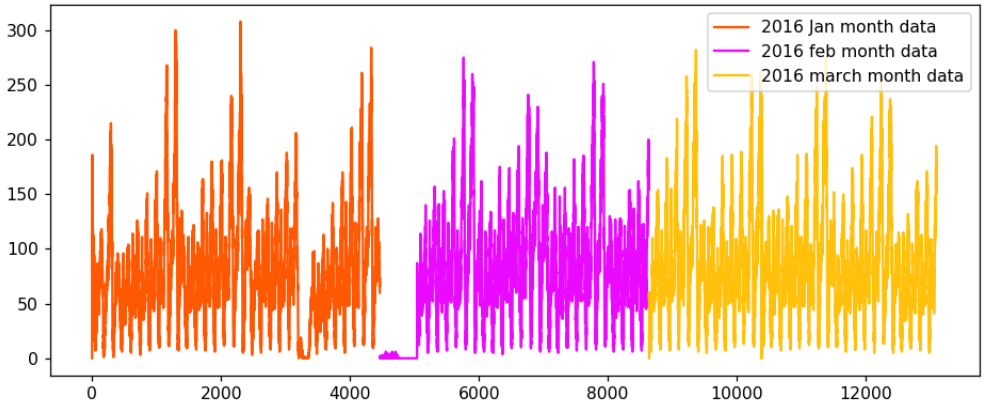
In [58]:
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are fill
ed with zero
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_
unique)
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_20
16_unique)
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_20
16_unique)
mar_2016_smooth = fill_missing(mar_2016_groupby['trip_distance'].values,mar_20
16_unique)

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

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

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

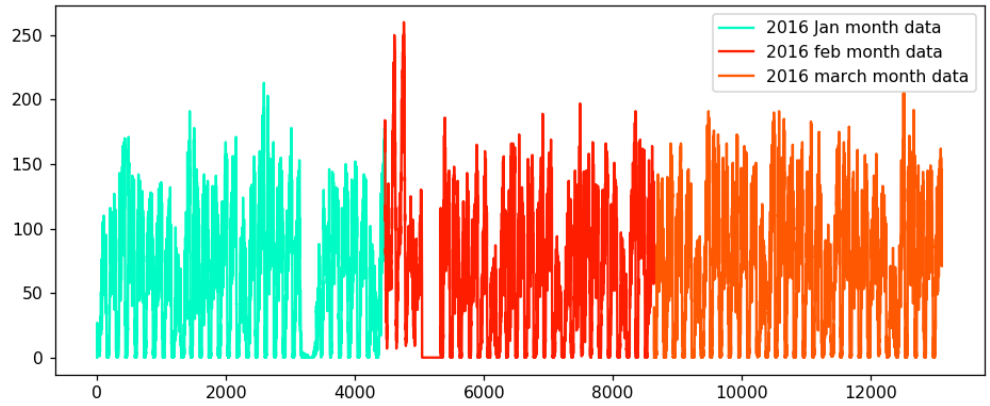
Time series and Fourier Transforms

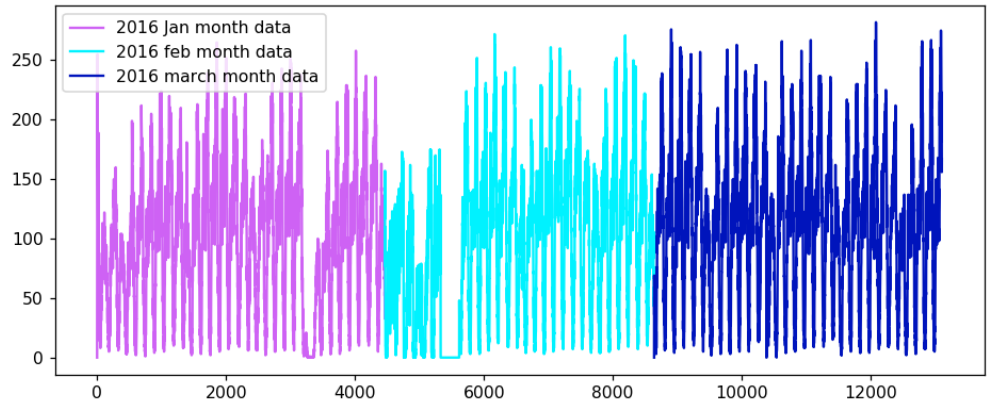
In [59]:
```
def uniqueish_color():

    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='20
16 Jan month data')
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), labe
l='2016 feb month data')
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='20
16 march month data')
    plt.legend()
    plt.show()
```

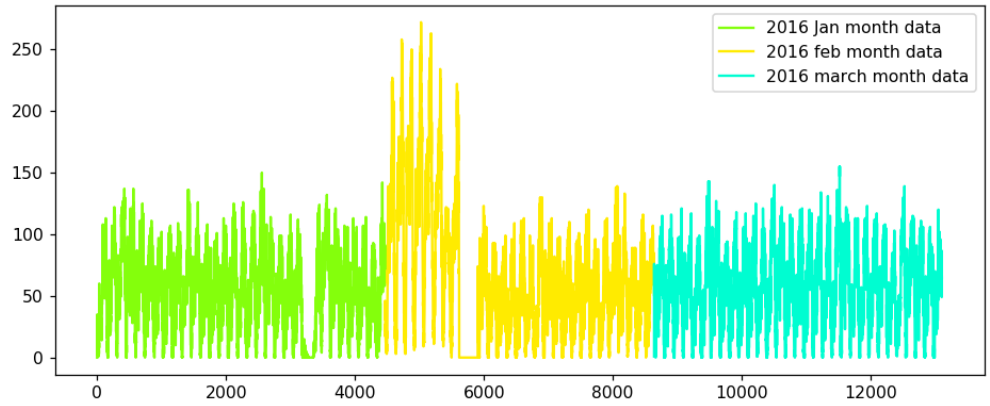

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

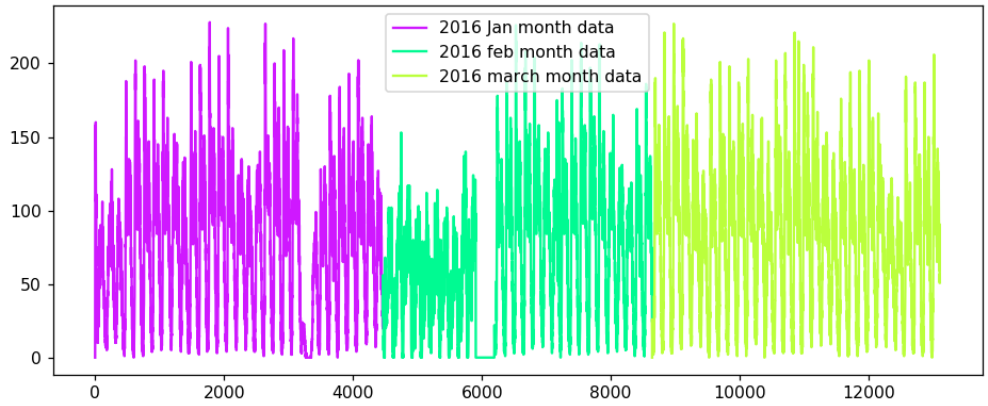

2016 Jan month data
2016 feb month data
2016 march month data
0
5
10
15
20
0
2000
4000
6000
8000
10000
12000

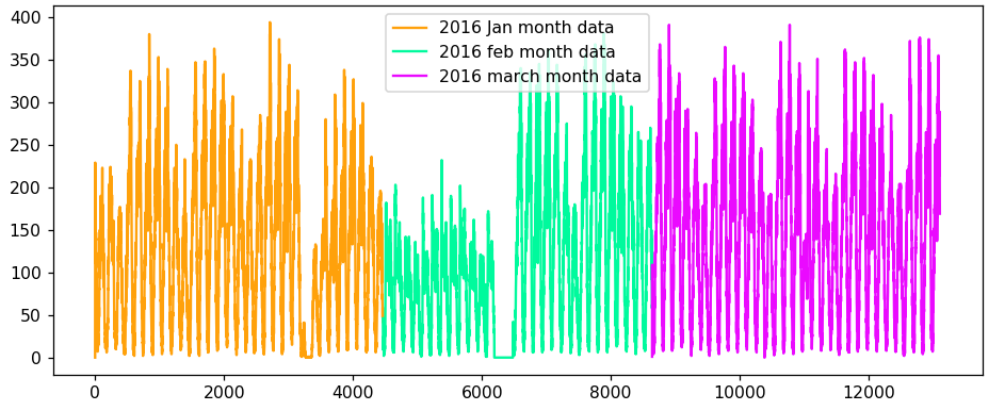

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

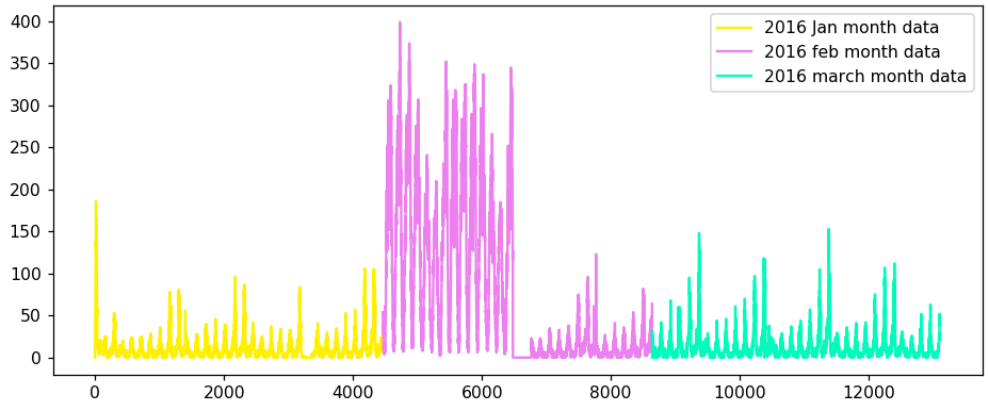

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

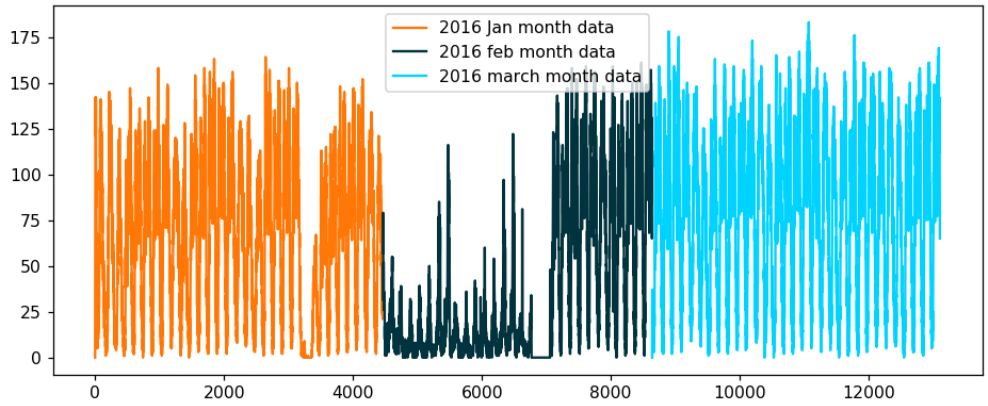

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

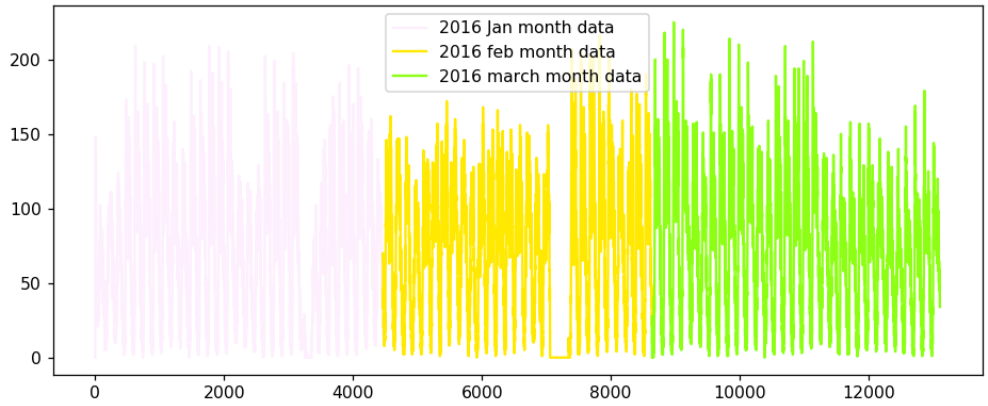

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

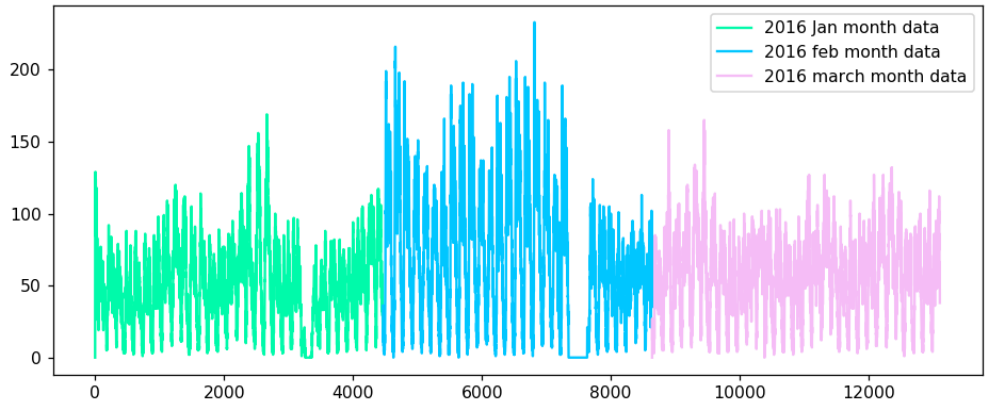

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

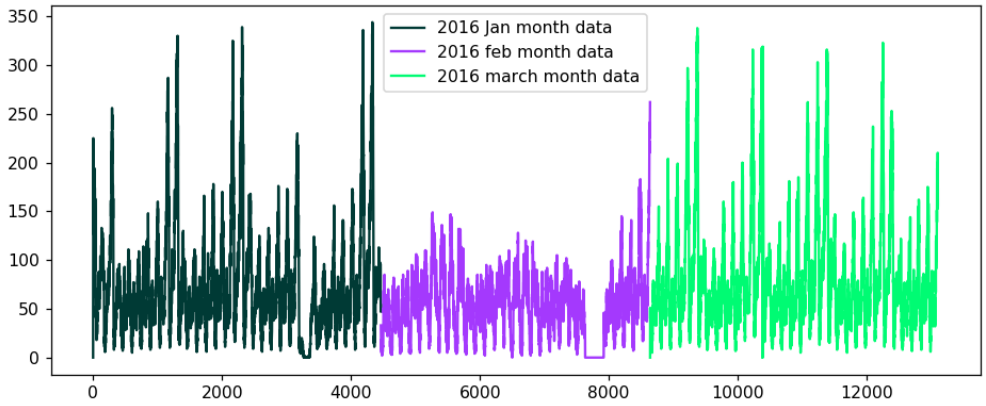

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
12000

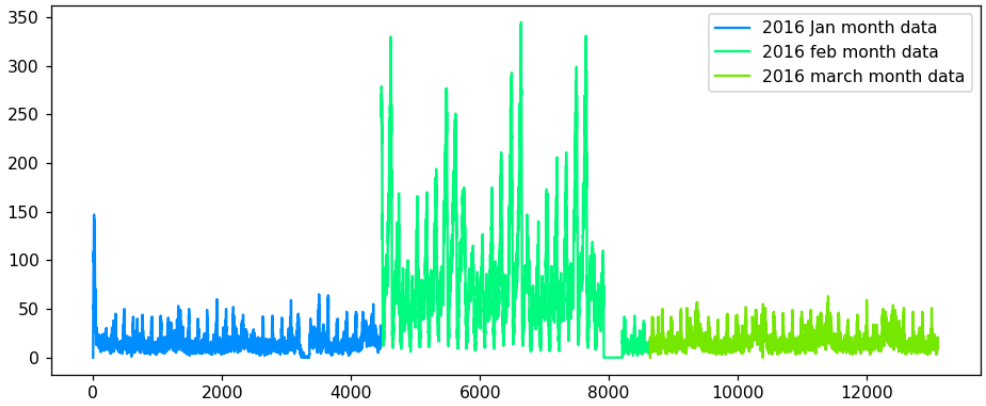

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
12000

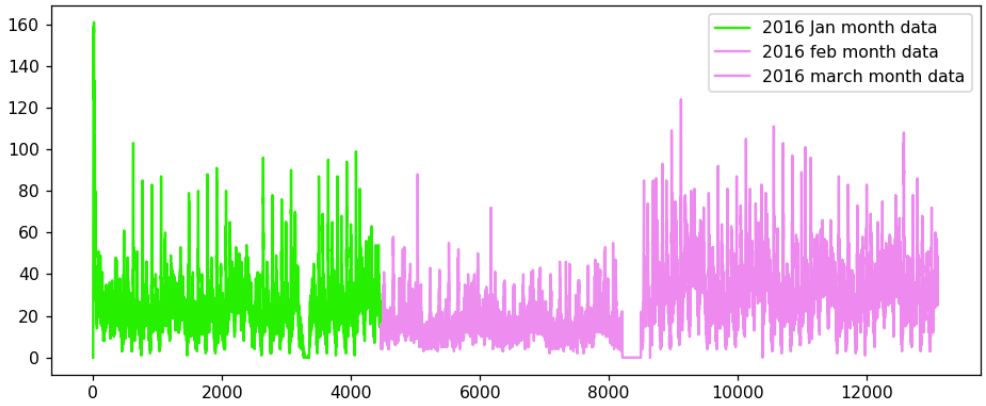

2016 Jan month data
2016 feb month data
2016 march month data
0
25
50
75
100
125
150
175
0
2000
4000
6000
8000
10000
12000

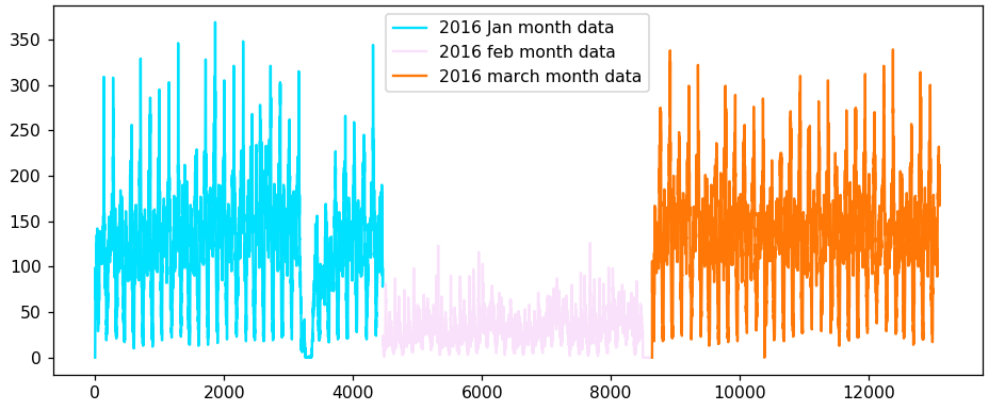

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

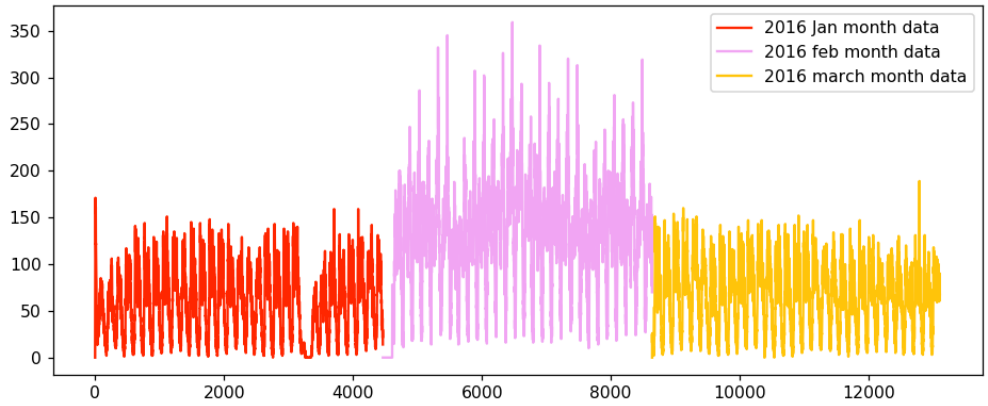

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

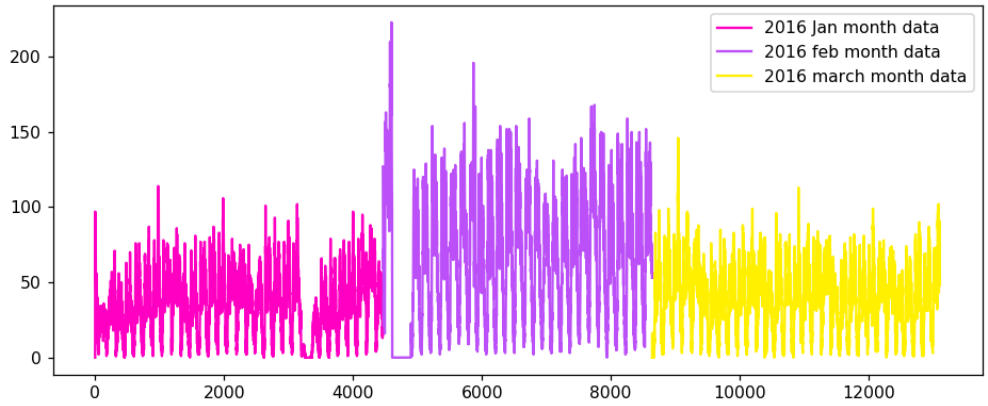

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

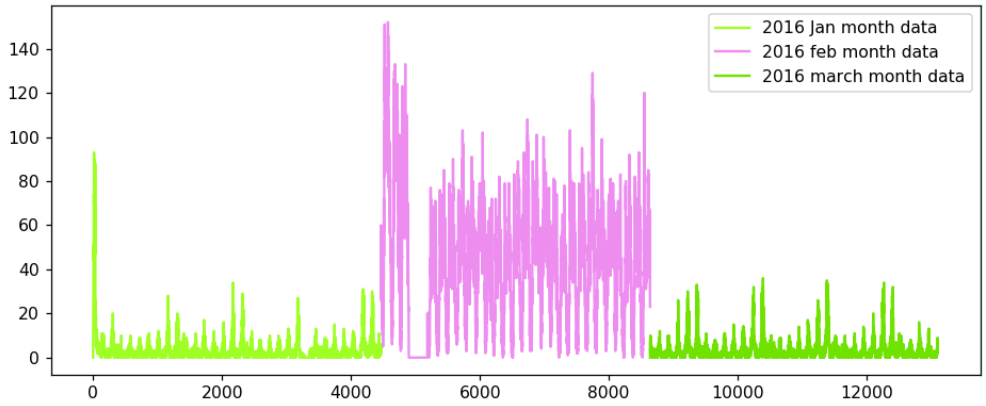

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

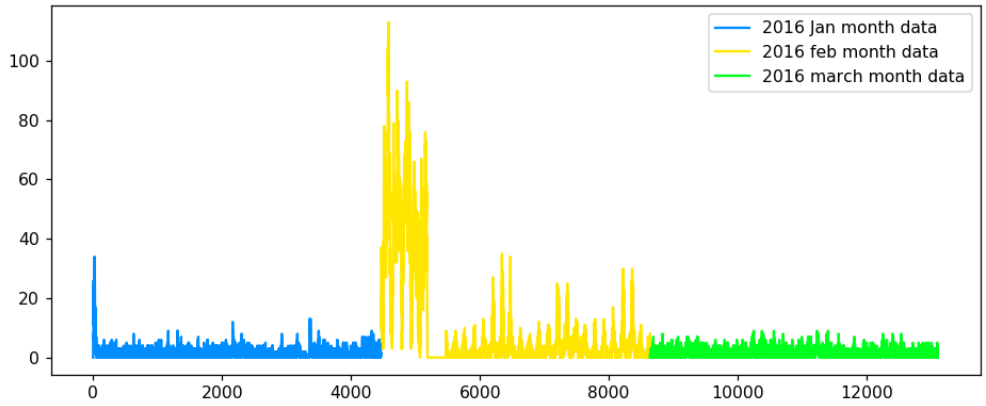

2016 Jan month data
2016 feb month data
2016 march month data
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000
10000
12000

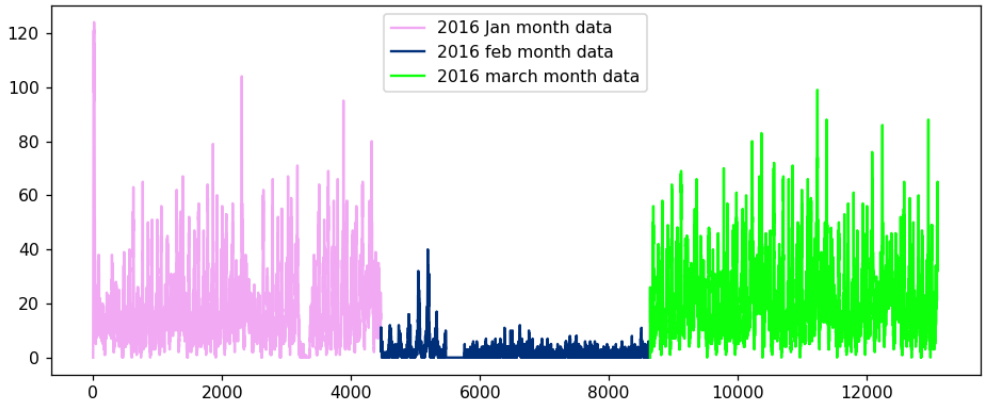

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

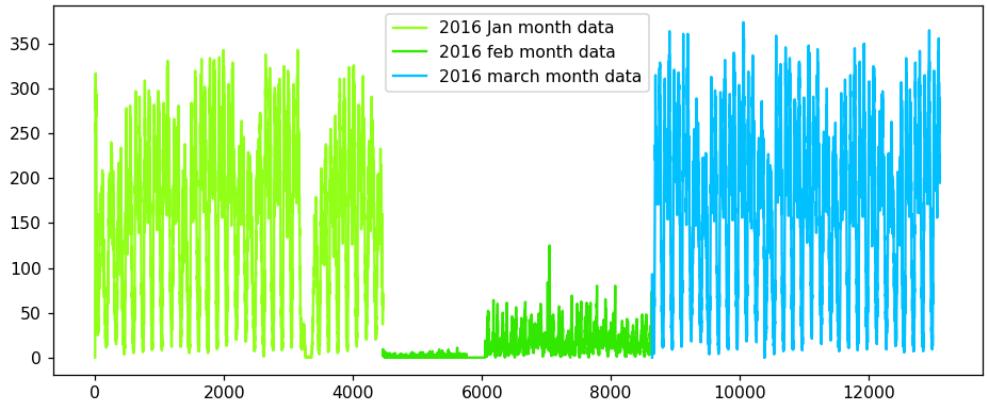

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

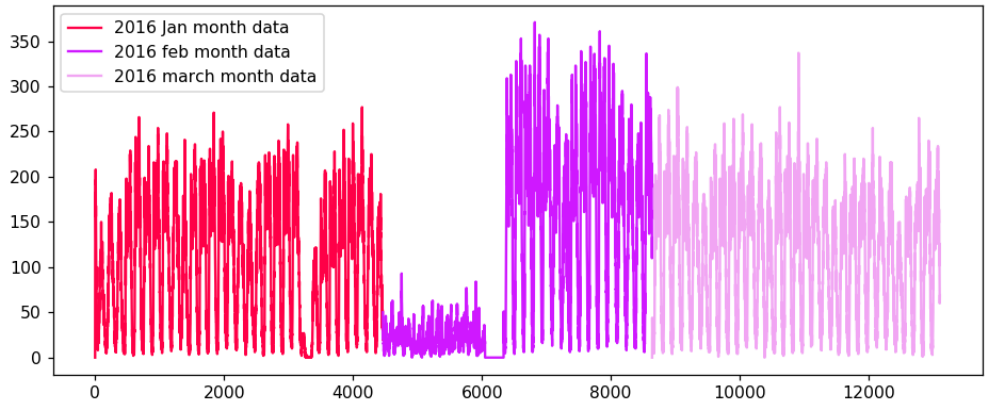

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

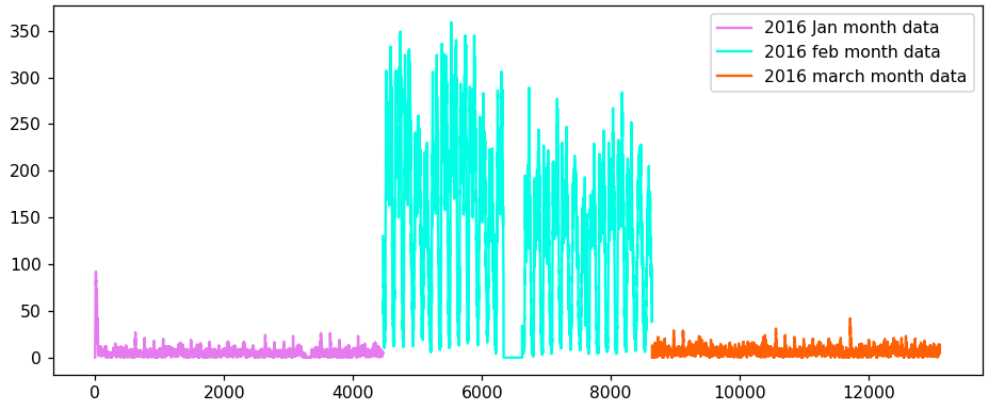

2016 Jan month data
2016 feb month data
2016 march month data
0
20
40
60
80
100
120
140
0
2000
4000
6000
8000
10000
12000

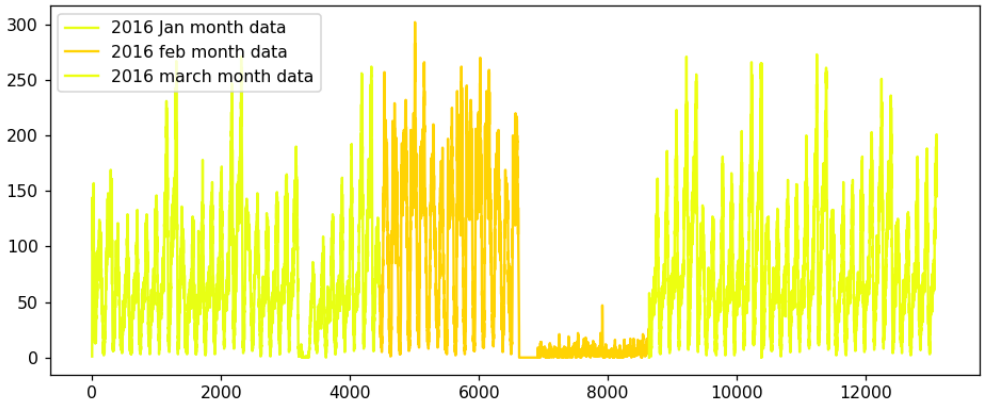

2016 Jan month data
2016 feb month data
2016 march month data
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
12000

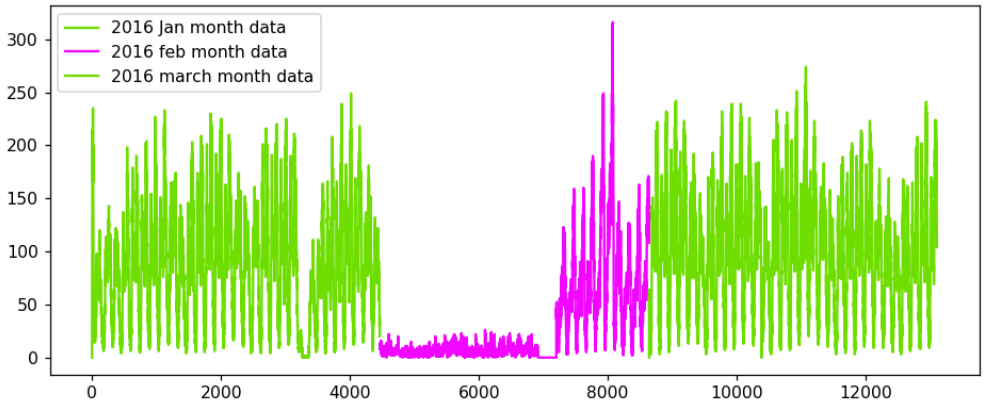

2016 Jan month data
2016 feb month data
2016 march month data
0
20
40
60
80
100
120
0
2000
4000
6000
8000
10000
12000

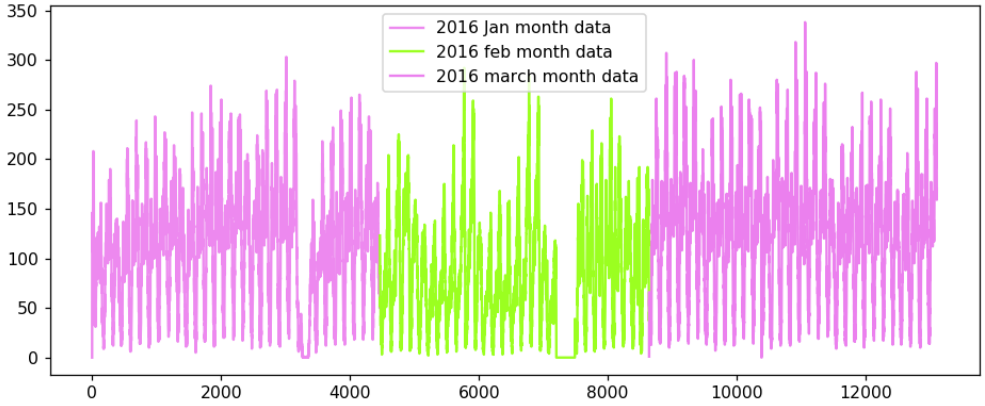

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

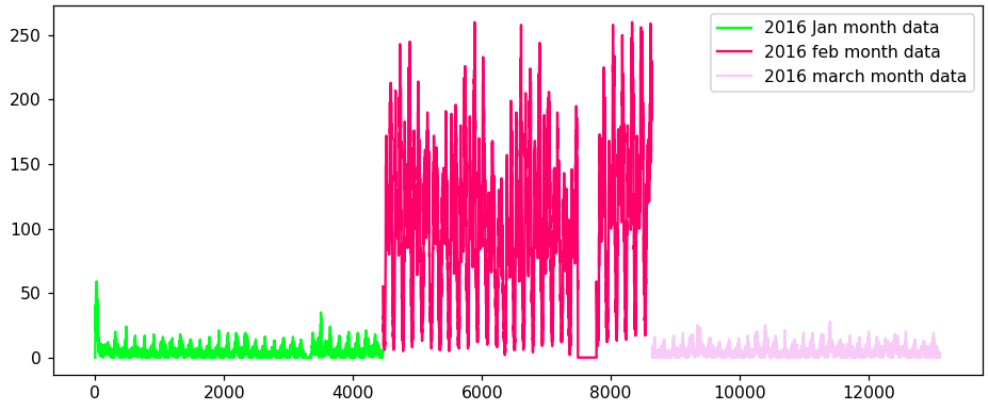

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

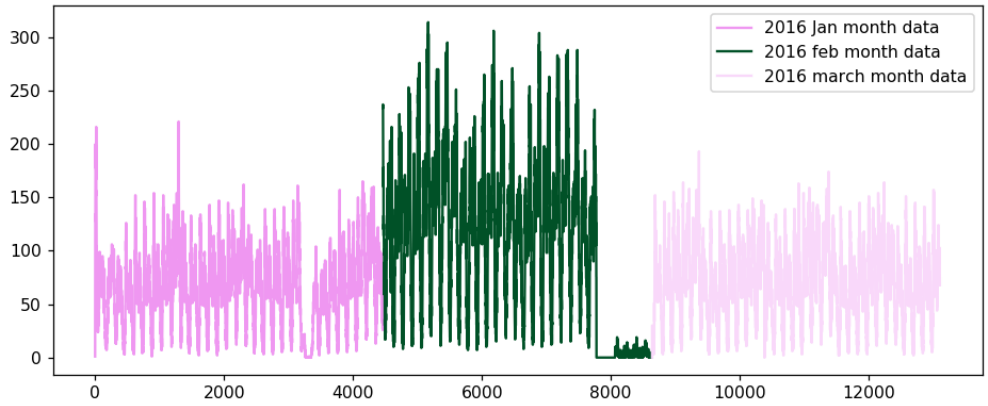

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

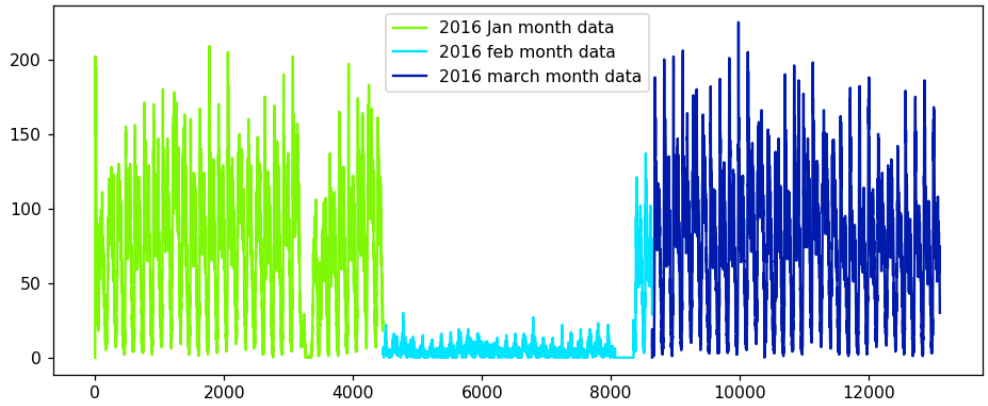

2016 Jan month data
2016 feb month data
2016 march month data
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

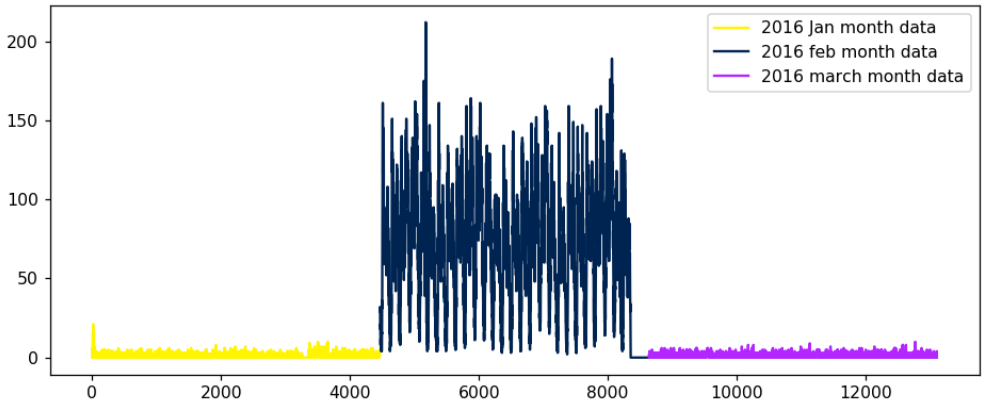

2016 Jan month data
2016 feb month data
2016 march month data
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

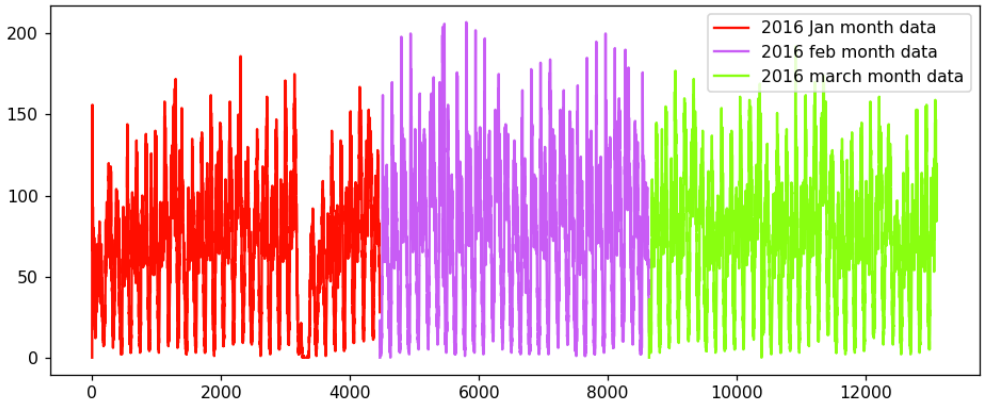

2016 Jan month data
2016 feb month data
2016 march month data
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

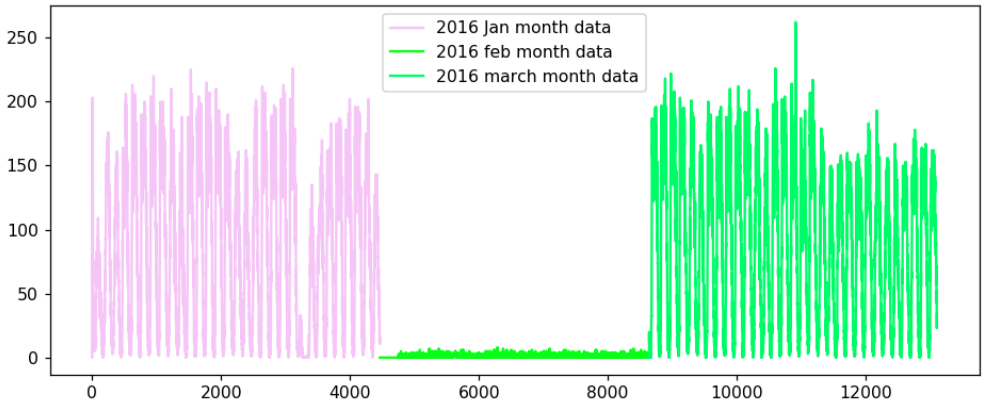

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

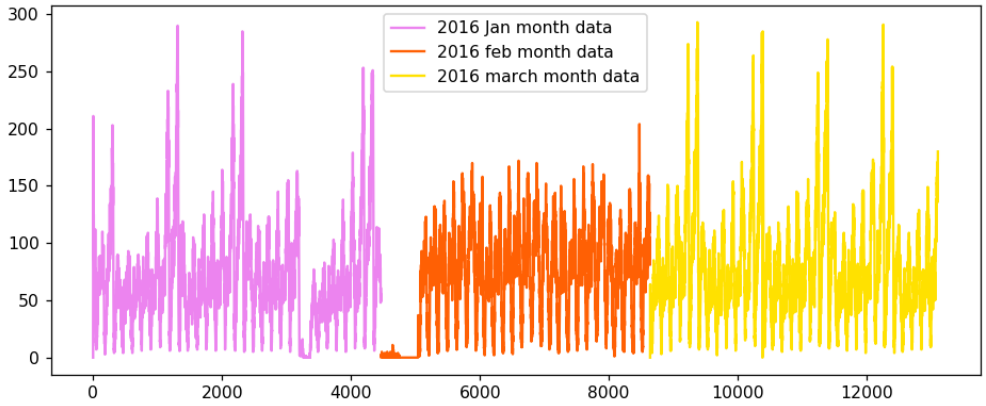

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

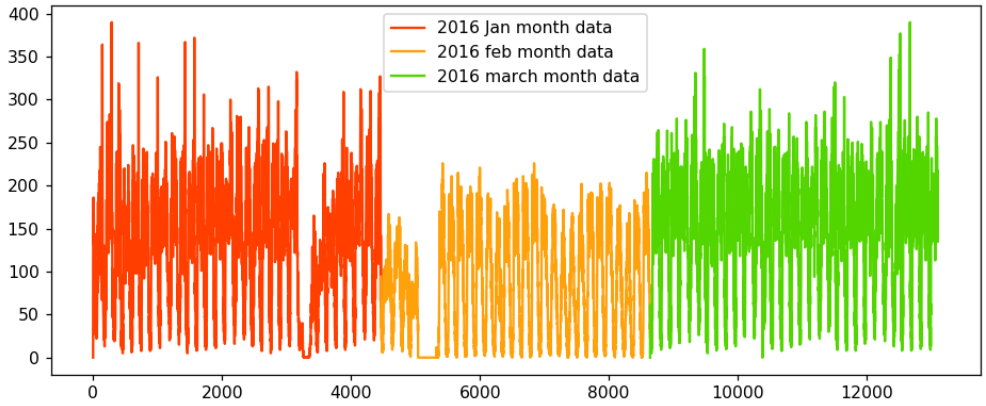

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

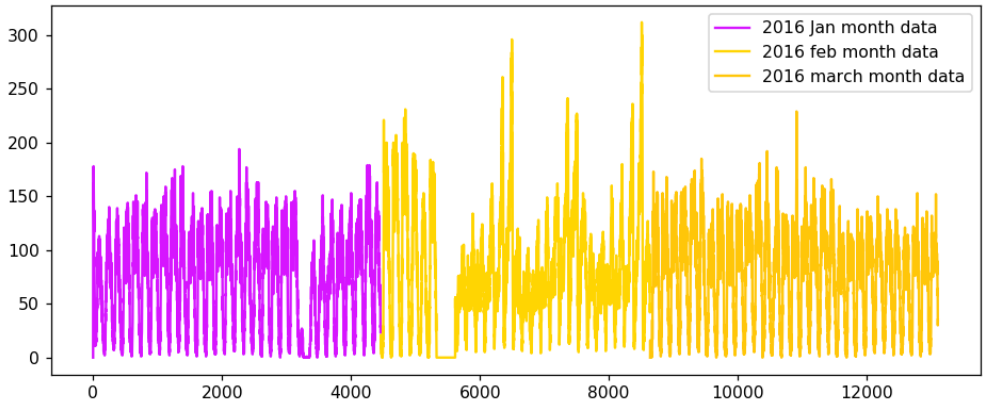

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

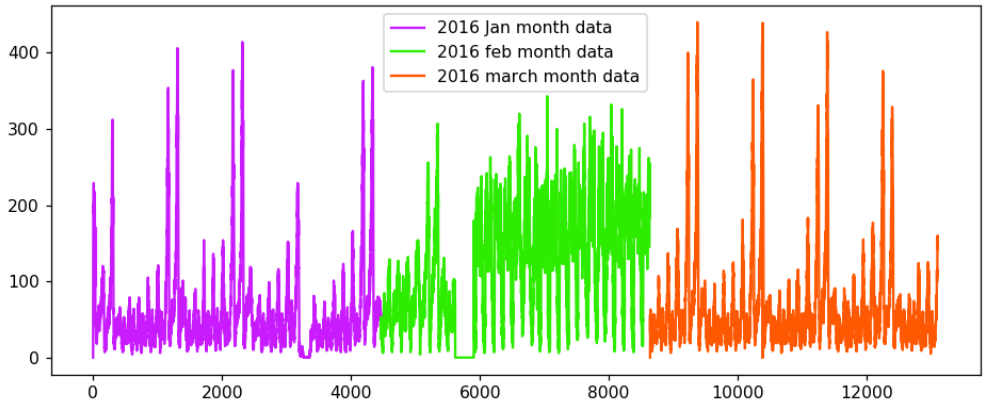

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

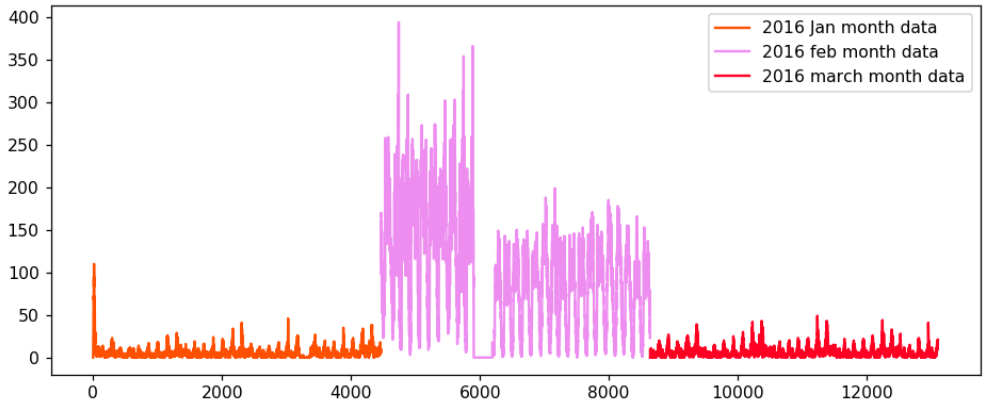

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

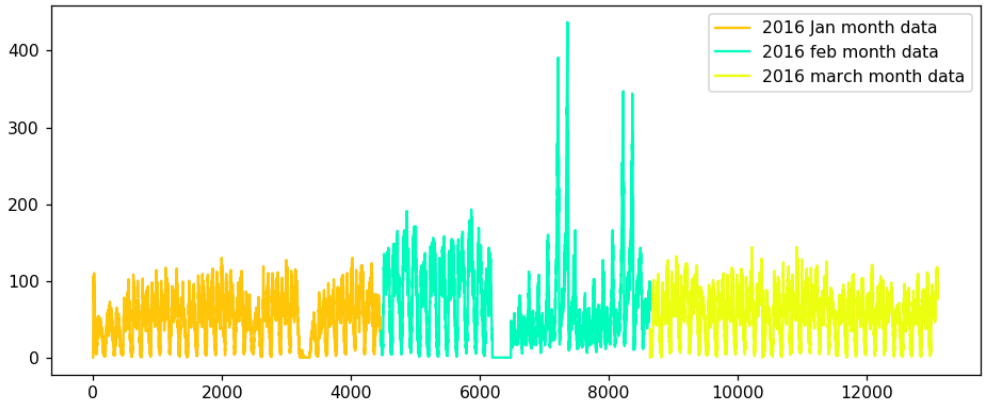

2016 Jan month data
2016 feb month data
2016 march month data
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

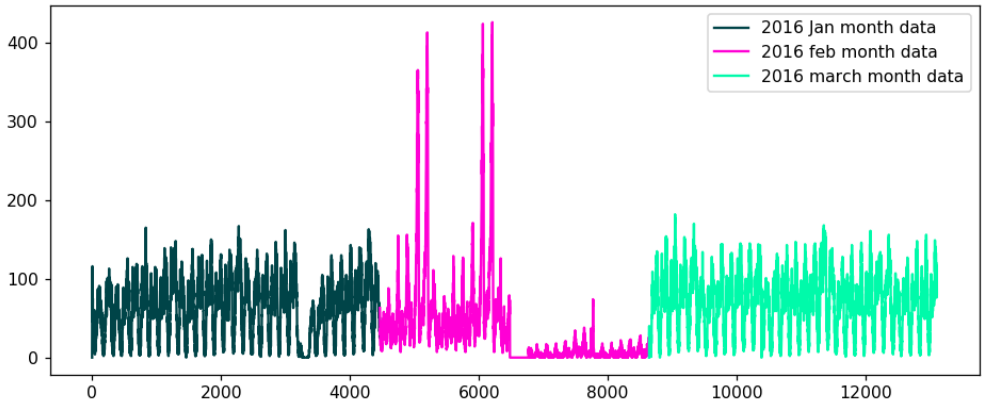

2016 Jan month data
2016 feb month data
2016 march month data
400
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000


2016 Jan month data
2016 feb month data
2016 march month data
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

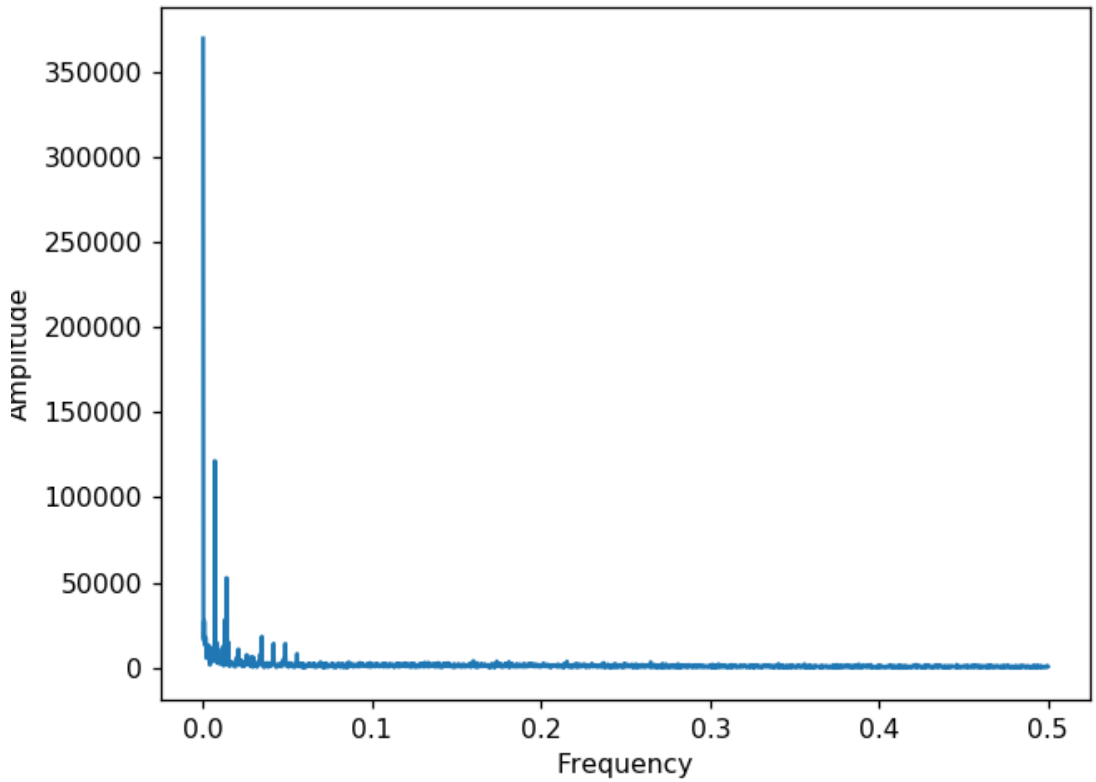

2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000


2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
12000


2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000



In [60]:
```
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
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()
```



In [61]:
```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) value
s 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
```

Modelling: Baseline Models

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

1. Using Ratios of the 2016 data to the 2015 data 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

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 - $R_t = (R_{t-1} + R_{t-2} + R_{t-3} .... R_{t-n})/n$

In [62]:
```
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_ra
tio))
        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(r
atios['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 $P_t = (P_{t-1} + P_{t-2} + P_{t-3} .... P_{t-n})/n$

In [63]:
```
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'].value
s)[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(r
atios['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 $P_t = P_{t-1}$

Weighted Moving Averages

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

Weighted Moving Averages using Ratio Values -

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

In [64]:
```
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_ra
tio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted
_ratio)-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

    ratios['WA_R_Predicted'] = predicted_values
    ratios['WA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(r
atios['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 -
$P_t = (N * P_{t-1} + (N-1) * P_{t-2} + (N-2) * P_{t-3} .... 1 * P_{t-n})/(N * (N+1)/2)$

In [65]:
```
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'].value
s)[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(r
atios['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 $P_t = (2 * P_{t-1} + P_{t-2})/3$

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 infinetly 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 [66]:
```
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_ra
tio))
        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['Ratio
s'].values)[i])

    ratios['EA_R1_Predicted'] = predicted_values
    ratios['EA_R1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(r
atios['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 [67]:
```
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'].value
s)[i],1))))
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Pre
diction'].values)[i]))

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

In [68]:
```
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')
```

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 [101]:
```
print ("Error Metric Matrix (Forecasting Methods) - MAPE & MSE")
print ("--------------------------------------------------------------------
----------------------------------------")
print ("Moving Averages (Ratios) -                              MAPE: ",mean_er
r[0],"       MSE: ",median_err[0])
print ("Moving Averages (2016 Values) -                         MAPE: ",mean_er
r[1],"       MSE: ",median_err[1])
print ("--------------------------------------------------------------------
----------------------------------------")
print ("Weighted Moving Averages (Ratios) -                     MAPE: ",mean_er
r[2],"       MSE: ",median_err[2])
print ("Weighted Moving Averages (2016 Values) -                MAPE: ",mean_er
r[3],"       MSE: ",median_err[3])
print ("--------------------------------------------------------------------
----------------------------------------")
print ("Exponential Moving Averages (Ratios) -                  MAPE: ",mean_er
r[4],"       MSE: ",median_err[4])
print ("Exponential Moving Averages (2016 Values) -             MAPE: ",mean_er
r[5],"       MSE: ",median_err[5])
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------
--------------------------------------
Moving Averages (Ratios) -                              MAPE:  0.1821155173392
136        MSE:  400.0625504032258
Moving Averages (2016 Values) -                         MAPE:  0.1429284968697
5506        MSE:  174.84901993727598
--------------------------------------------------------------------------------
--------------------------------------
Weighted Moving Averages (Ratios) -                     MAPE:  0.1784869254376
018        MSE:  384.01578741039424
Weighted Moving Averages (2016 Values) -                MAPE:  0.1355108843618
2082        MSE:  162.46707549283155
--------------------------------------------------------------------------------
--------------------------------------
Exponential Moving Averages (Ratios) -                  MAPE:  0.1778355019486
1494        MSE:  378.34610215053766
Exponential Moving Averages (2016 Values) -             MAPE:  0.1350915263669
572        MSE:  159.73614471326164
```

Plese 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:-
$P_t^{'} = \alpha * P_{t-1} + (1 - \alpha) * P_{t-1}^{'}$ i.e Exponential Moving Averages using 2016 Values

Regression Models

Train-Test Split

Before we start predictions using the tree based regression models we take 3 months of 2016 pickup data and split it such that for every region we have 70% data in train and 30% in test, ordered date-wise for every region

In [70]:
```
# Preparing data to be split into train and test, The below prepares data in c
umulative 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 which represents the number of pickups
# that are happened for three months in 2016 data

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

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

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


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


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

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

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

```
tsne_feature = [0]*number_of_time_stamps
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 numbe
r of pickups that are happened in last 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+
4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104],
 [x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time
_stamps] for r in range(0,len(regions_cum[i])-number_of_time_stamps)]))
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [71]:
```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*l
en(tsne_weekday[0]) == 40*13099 == len(output)*len(output[0])
```

Out[71]: True

In [72]:
```
# Getting the predictions of exponential moving averages to be used as a featu
re in cumulative form

# upto now we computed 8 features for every data point that starts from 50th m
in of the day
# 1. cluster center lattitude
# 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 giv
es us the best error
# we will try to add the same exponential weighted moving avarage at t as a fe
ature 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 e
ach 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], [x
5,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 [73]:
```
# 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
```

In [74]:
```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) f
or 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(0,4
0)]
```

In [75]:
```
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_fe
atures[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_featu
res[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 [76]:
```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) f
or 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]
```

In [77]:
```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestam
ps) 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]
```

In [78]:
```
# 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 [79]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

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

In [80]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

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

In [81]:
```
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1']
df_train = pd.DataFrame(data=train_new_features, columns=columns)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon
df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg

print(df_train.shape)
```

```
(366760, 9)
```

In [82]:
```
# 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
print(df_test.shape)
```

```
(157200, 9)
```

In [83]:
```
df_test.head()
```

Out[83]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg
0	118	106	104	93	102	40.776228	-73.982119	4	100
1	106	104	93	102	101	40.776228	-73.982119	4	100
2	104	93	102	101	120	40.776228	-73.982119	4	114
3	93	102	101	120	131	40.776228	-73.982119	4	125
4	102	101	120	131	164	40.776228	-73.982119	4	152

Using Linear Regression

In [84]:
```
# default paramters
# sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, c
opy_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 o
f the prediction.
# set_params(**params)  Set the parameters of this estimator.

from sklearn.linear_model import LinearRegression
lr_reg=LinearRegression().fit(df_train, tsne_train_output)

y_pred = lr_reg.predict(df_test)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = lr_reg.predict(df_train)
lr_train_predictions = [round(value) for value in y_pred]
```

Using Random Forest Regressor

In [85]:
```
# default paramters
# sklearn.ensemble.RandomForestRegressor(n_estimators=10, criterion='mse', max
_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=None, verbose=0, warm_start=False)

# some of methods of RandomForestRegressor()
# apply(X)      Apply trees in the forest to X, return leaf indices.
# decision_path(X)      Return the decision path in the forest
# fit(X, y[, sample_weight])    Build a forest of trees from the training set
 (X, y).
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict regression target for X.
# score(X, y[, sample_weight])  Returns the coefficient of determination R^2 o
f the prediction.

regr1 = RandomForestRegressor(max_features='sqrt',min_samples_leaf=4,min_sampl
es_split=3,n_estimators=40, n_jobs=-1)
regr1.fit(df_train, tsne_train_output)
```

Out[85]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=4, min_samples_split=3,
           min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [86]:
```
y_pred = regr1.predict(df_test)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr1.predict(df_train)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [87]:
```
#feature importances based on analysis using random forest
print (df_train.columns)
print (regr1.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg'],
      dtype='object')
[0.0411536  0.06153569 0.06834625 0.18420542 0.21076391 0.00416572
 0.00265193 0.00162884 0.42554863]
```

Using XgBoost Regressor

In [88]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# default paramters
# xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, silen
t=True, objective='reg:linear',
# booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_d
elta_step=0, subsample=1, colsample_bytree=1,
# colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_sco
re=0.5, random_state=0, seed=None,
# missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stoppin
g_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE:
 This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance

x_model = xgb.XGBRegressor(
 learning_rate =0.1,
 n_estimators=1000,
 max_depth=3,
 min_child_weight=3,
 gamma=0,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=0.8,nthread=4)
x_model.fit(df_train, tsne_train_output)
```

Out[88]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.8, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=3, missing=None, n_estimators=1000,
       n_jobs=1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=200, reg_lambda=200, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
```

In [95]:
```
#predicting with our trained XG-Boost regressor
# the parameters that we got above are found using grid search

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

In [99]:
```
#feature importances
x_model.get_booster().get_score(importance_type='weight')
```

Out[99]:
```
{'exp_avg': 788,
 'ft_1': 978,
 'ft_2': 1023,
 'ft_3': 860,
 'ft_4': 722,
 'ft_5': 1032,
 'lon': 593,
 'lat': 584,
 'weekday': 171}
```

Calculating the error metric values for various models

In [91]:
```
train_mape=[]
test_mape=[]

train_mape.append((mean_absolute_error(tsne_train_output,df_train['ft_1'].valu
es))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].v
alues))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_prediction
s))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output, xgb_train_prediction
s))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output, lr_train_predictions
))/(sum(tsne_train_output)/len(tsne_train_output)))

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values
))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].val
ues))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions
))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))
/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, lr_test_predictions))/
(sum(tsne_test_output)/len(tsne_test_output)))
```

Error Metric Matrix

In [103]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------
--------------------------------")
print ("Baseline Model -                              Train: ",train_mape[0],"
     Test: ",test_mape[0])
print ("Exponential Averages Forecasting -          Train: ",train_mape[1],"
     Test: ",test_mape[1])
print ("Linear Regression -                           Train: ",train_mape[4],"
     Test: ",test_mape[4])
print ("Random Forest Regression -                    Train: ",train_mape[2],"
     Test: ",test_mape[2])
print ("XgBoost Regression -                          Train: ",train_mape[3],"
     Test: ",test_mape[3])
print ("--------------------------------------------------------------------
--------------------------------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------
---------------------------
Baseline Model -                              Train:  0.14005275878666593
  Test:  0.13653125704827038
Exponential Averages Forecasting -          Train:  0.13289968436017227
  Test:  0.12936180420430524
Linear Regression -                           Train:  0.13331572016045437
  Test:  0.1291202994009687
Random Forest Regression -                    Train:  0.09174106169550536
  Test:  0.12718529831147335
XgBoost Regression -                          Train:  0.12939582809998681
  Test:  0.12686179464732844
--------------------------------------------------------------------------------
---------------------------
```

Conclusion

- Of all the models XGBoost performed well with 12% MAPE.
- At the same time we have observed that baseline models itself are also very powerful in this particular time-series problem. So, therefore, we shall not neglect our baseline predictions as well.
- We can use other feature engineering techniques to reduce the error less than 12%.