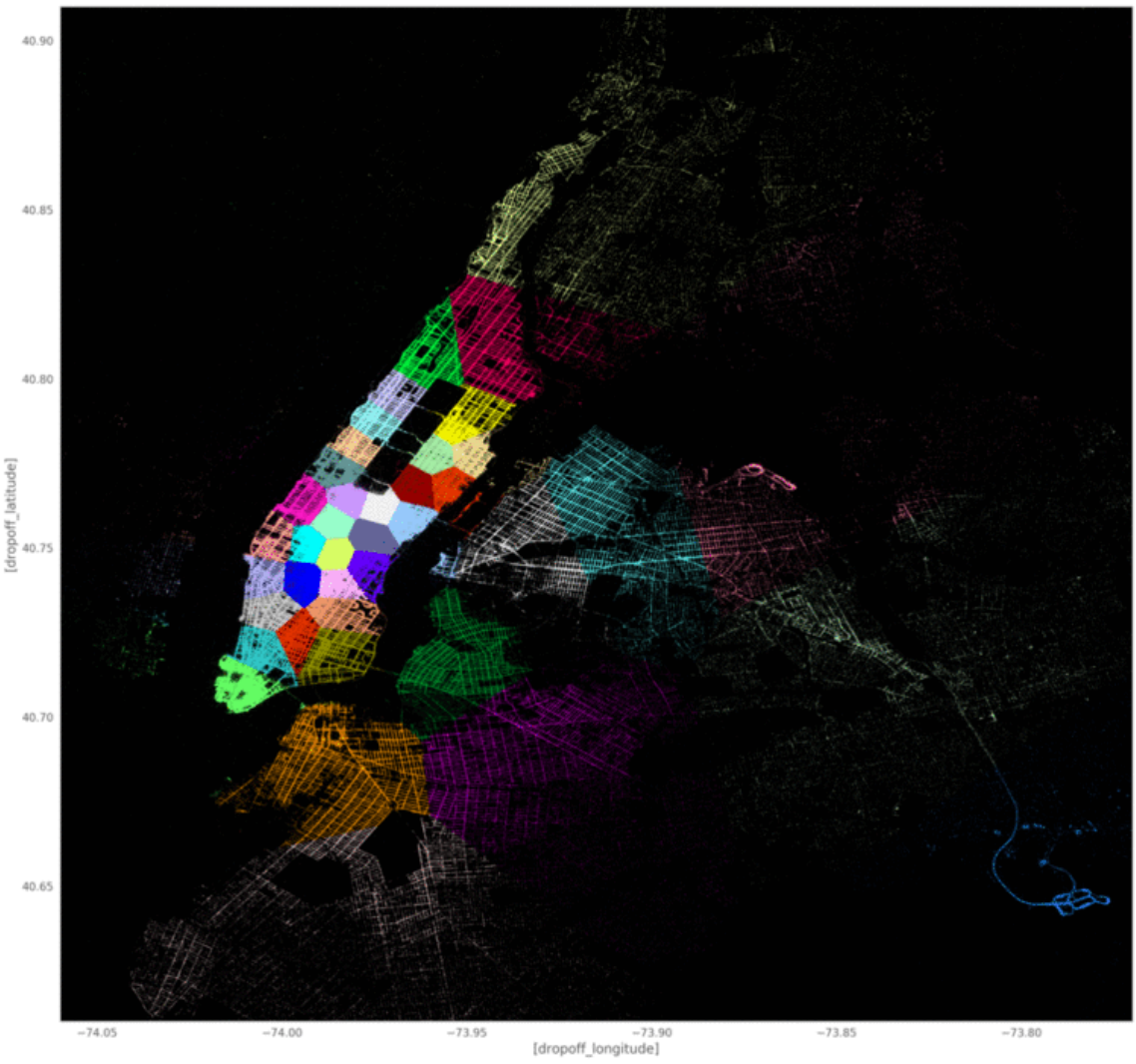
Taxi demand prediction in New York City



In [1]:

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

import pandas as pd#pandas to create small dataframes

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

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

import time #Convert to unix time

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

# matplotlib: used to plot graphs
import matplotlib
# matplotlib.use('nbagg') : matplotlib uses this protocall which makes plots more u
matplotlib.use('nbagg')
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots

# this lib is used while we calculate the stight line distance between two (lat,lon
import gpxpy.geo #Get the haversine distance

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

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

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

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

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

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

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	1. 62G	11135470	19
yellow_tripdata_2016-07	884Mb	10294080	17
yellow_tripdata_2016-08	854Mb	9942263	17
yellow_tripdata_2016-09	870Mb	10116018	17
yellow_tripdata_2016-10	933Mb	10854626	17
yellow_tripdata_2016-11	868Mb	10102128	17
yellow_tripdata_2016-12	897Mb	10449408	17

yellow_tripdata_2015-01	1.84Gb	12748986	19
yellow_tripdata_2015-02	1.81Gb	12450521	19
yellow_tripdata_2015-03	1.94Gb	13351609	19
yellow_tripdata_2015-04	1.90Gb	13071789	19
yellow_tripdata_2015-05	1.91Gb	13158262	19
yellow_tripdata_2015-06	1.79Gb	12324935	19
yellow_tripdata_2015-07	1.68Gb	11562783	19
yellow_tripdata_2015-08	1.62Gb	11130304	19
yellow_tripdata_2015-09	1.63Gb	11225063	19
yellow_tripdata_2015-10	1.79Gb	12315488	19
yellow_tripdata_2015-11	1.65Gb	11312676	19
yellow_tripdata_2015-12	1.67Gb	11460573	19

In [2]:

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

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

In [3]:

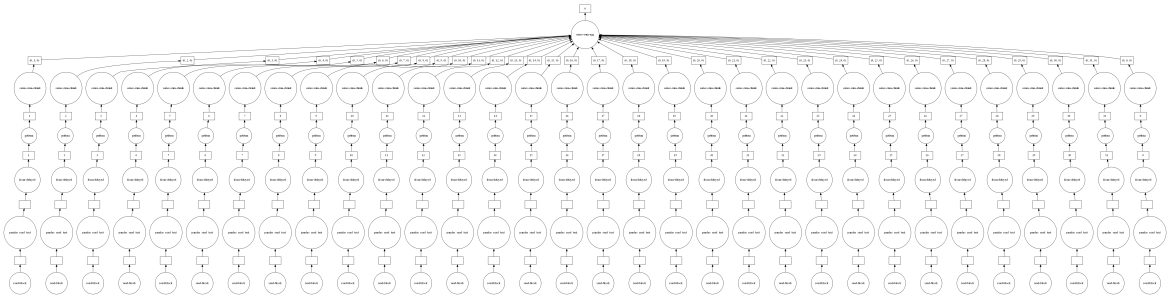
```
# However unlike Pandas, operations on dask.dataframes don't trigger immediate comp
# instead they add key-value pairs to an underlying Dask graph. Recall that in the
# circles are operations and rectangles are results.

# to see the visulaization you need to install graphviz
# pip3 install graphviz if this doesnt work please check the install_graphviz.jpg i

month.visualize()
```

Out[3]:

In [4]:

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

Out[4]:



Features in the dataset:

```
<tr>
    <td>Dropoff_longitude</td>
    <td>Longitude where the meter was disengaged.</td>
</tr>
<tr>
    <td>Dropoff_ latitude</td>
    <td>Latitude where the meter was disengaged.</td>
</tr>
<tr>
    <td>Payment_type</td>
    <td>A numeric code signifying how the passenger paid for the trip.
    <ol>
        <li> Credit card </li>
        <li> Cash </li>
        <li> No charge </li>
        <li> Dispute</li>
        <li> Unknown </li>
        <li> Voided trip</li>
    </ol>
    </td>
</tr>
<tr>
    <td>Fare_amount</td>
    <td>The time-and-distance fare calculated by the meter.</td>
</tr>
<tr>
    <td>Extra</td>
    <td>Miscellaneous extras and surcharges. Currently, this only includes.
the $0.50 and $1 rush hour and overnight charges.</td>
</tr>
<tr>
    <td>MTA_tax</td>
    <td>0.50 MTA tax that is automatically triggered based on the metered r
ate in use.</td>
</tr>
<tr>
    <td>Improvement_surcharge</td>
    <td>0.30 improvement surcharge assessed trips at the flag drop. the imp
rovement surcharge began being levied in 2015.</td>
</tr>
<tr>
    <td>Tip_amount</td>
    <td>Tip amount – This field is automatically populated for credit card
 tips.Cash tips are not included.</td>
</tr>
<tr>
    <td>Tolls_amount</td>
    <td>Total amount of all tolls paid in trip.</td>
</tr>
<tr>
    <td>Total_amount</td>
    <td>The total amount charged to passengers. Does not include cash tips.
```

```
</td>
</tr>
```

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. 1. Standard rate 2. JFK 3. Newark 4. Nassau or Westchester 5. Negotiated fare 6. Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip

ML Problem Formulation

Time-series forecasting and Regression

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

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

Performance metrics

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

Data Cleaning

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

In [3]:

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

Out[3]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pic
0	2.0	2015-01-15 19:05:39	2015-01-15 19:23:42	1.0	1.59	
1	1.0	2015-01-10 20:33:38	2015-01-10 20:53:28	1.0	3.30	
2	1.0	2015-01-10 20:33:38	2015-01-10 20:43:41	1.0	1.80	
3	1.0	2015-01-10 20:33:39	2015-01-10 20:35:31	1.0	0.50	
4	1.0	2015-01-10 20:33:39	2015-01-10 20:52:58	1.0	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 [4]:

```
# Plotting pickup cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlie
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_latit
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitude >=

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

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

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

# we will spot only first 100 outliers on the map, plotting all the outliers will t
sample_locations = outlier_locations.head(10000)
for i,j in sample_locations.iterrows():
    if int(j['pickup_latitude']) != 0:
        folium.Marker(list((j['pickup_latitude'],j['pickup_longitude']))).add_to(ma
map_osm
```

Out[4]:

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

2. Dropoff Latitude & Dropoff Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 (https://www.flickr.com/places/info/2459115) that New York is bounded by the location 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 dropoffs which are within New York.

In [5]:

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

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

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

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

# we will spot only first 100 outliers on the map, plotting all the outliers will t
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
```

Out[5]:

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.**

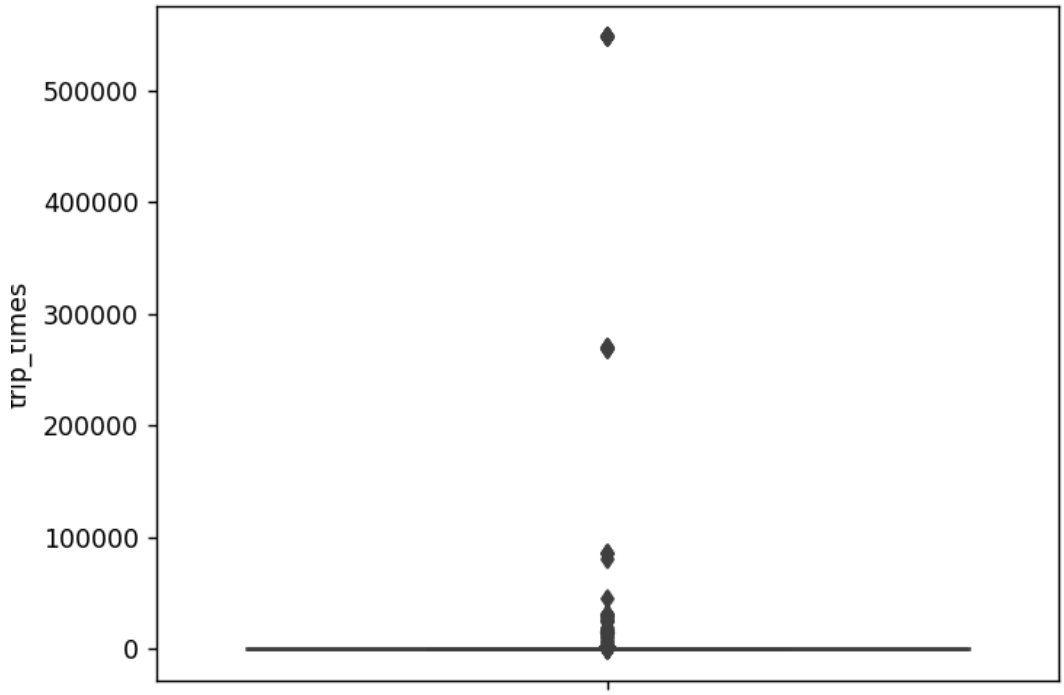
In [6]:

```
#The timestamps are converted to unix so as to get duration(trip-time) & speed alsc

# in out data we have time in the formate "YYYY-MM-DD HH:MM:SS" we convert thiss st
# https://stackoverflow.com/a/27914405
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetuple



# we return a data frame which contains the columns
# 1.'passenger_count' : self explanatory
# 2.'trip_distance' : self explanatory
# 3.'pickup_longitude' : self explanatory
# 4.'pickup_latitude' : self explanatory
# 5.'dropoff_longitude' : self explanatory
# 6.'dropoff_latitude' : self explanatory
# 7.'total_amount' : total fair that was paid
# 8.'trip_times' : duration of each trip
# 9.'pickup_times : pickup time converted into unix time
# 10.'Speed' : velocity of each trip
def return_with_trip_times(month):
    duration = month[['tpep_pickup_datetime','tpep_dropoff_datetime']].compute()
    #pickups and dropoffs to unix time
    duration_pickup = [convert_to_unix(x) for x in duration['tpep_pickup_datetime']
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'].
    #calculate duration of trips
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)

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

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

    return new_frame

# print(frame_with_durations.head())
#   passenger_count  trip_distance   pickup_longitude    pickup_latitude dropoff_lon
#   1                   1.59       -73.993896           40.750111       -73.974785
#   1                    3.30        -74.001648          40.724243       -73.994415
#   1                    1.80        -73.963341          40.802788       -73.951820
#   1                    0.50        -74.009087          40.713818       -74.004326
#   1                    3.00        -73.971176          40.762428       -74.004181
frame_with_durations = return_with_trip_times(month)
```

In [7]:

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

```
<IPython.core.display.Javascript object>
```



In [6]:

```
#calculating 0-100th percentile to find a the correct percentile value for removal
for i in range(0,100,10):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
0 percentile value is -1200.0
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.316666666666666
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 23.466666666666665
100 percentile value is  548555.6333333333
```

In [7]:

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

```
90 percentile value is 23.466666666666665
91 percentile value is 24.366666666666667
92 percentile value is 25.4
93 percentile value is 26.566666666666666
94 percentile value is 27.95
95 percentile value is 29.6
96 percentile value is 31.716666666666665
97 percentile value is 34.5
98 percentile value is 38.733333333333334
99 percentile value is 46.8
100 percentile value is  548555.6333333333
```

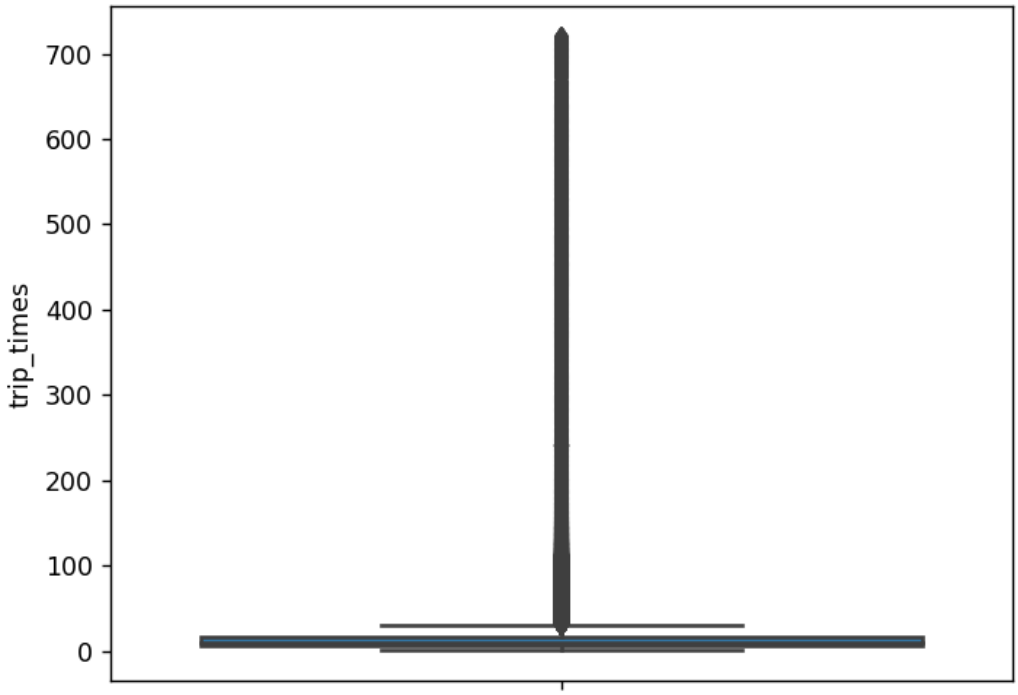
In [8]:

```
#removing data based on our analysis and TLC regulations
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_times
```

In [18]:

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

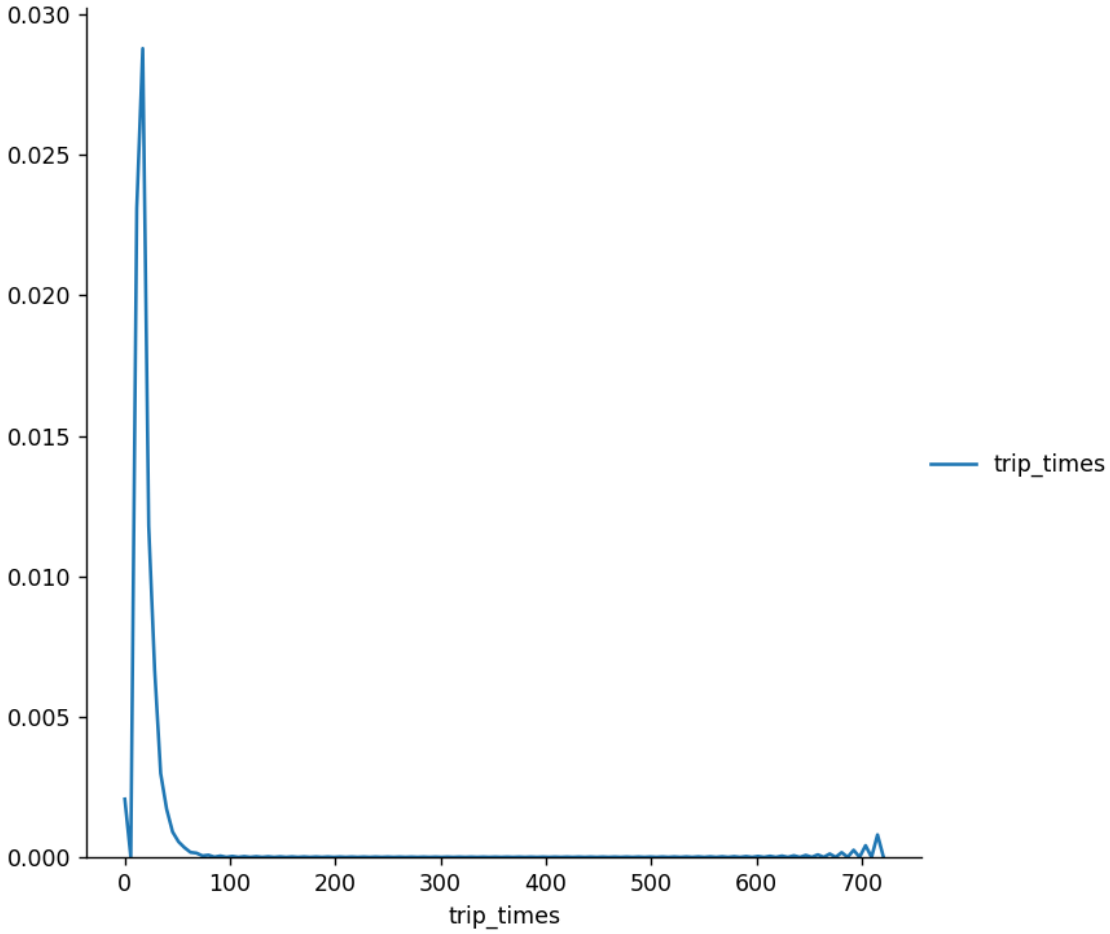
<IPython.core.display.Javascript object>



In [19]:

```
#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();
```

```
<IPython.core.display.Javascript object>
```



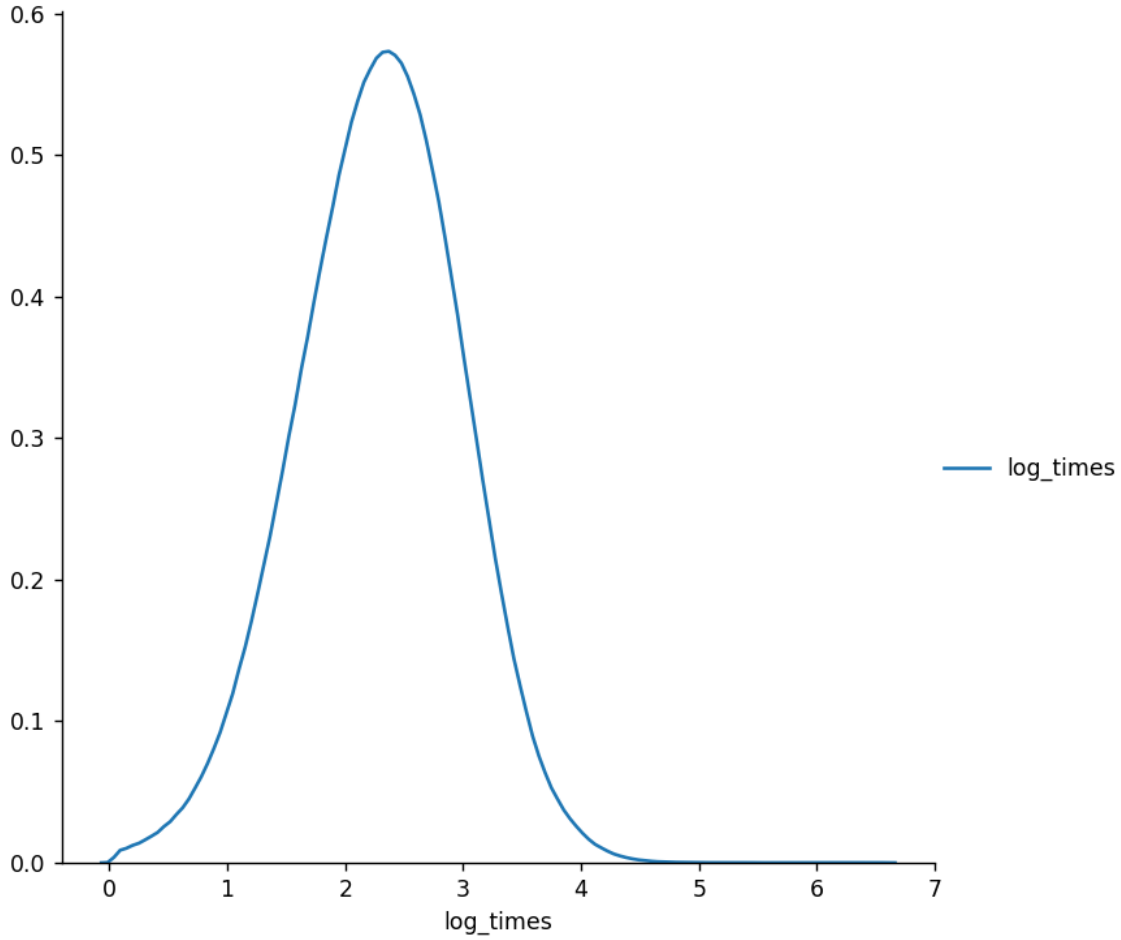
In [9]:

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

In [21]:

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

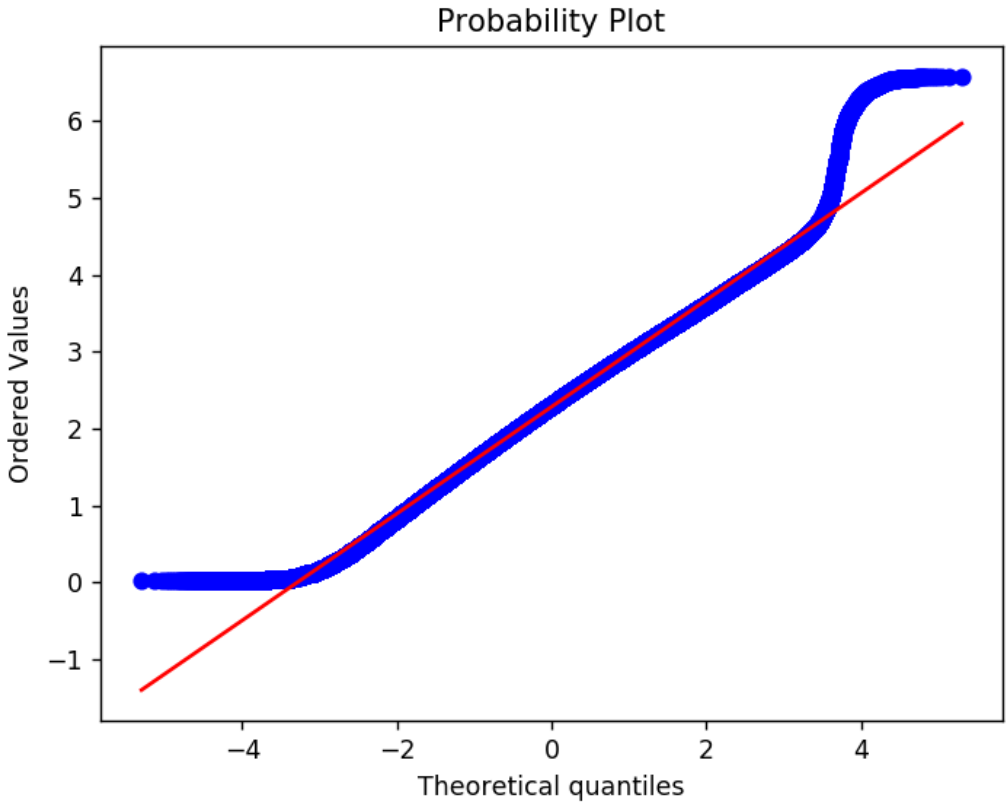
```
<IPython.core.display.Javascript object>
```



In [23]:

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

```
<IPython.core.display.Javascript object>
```

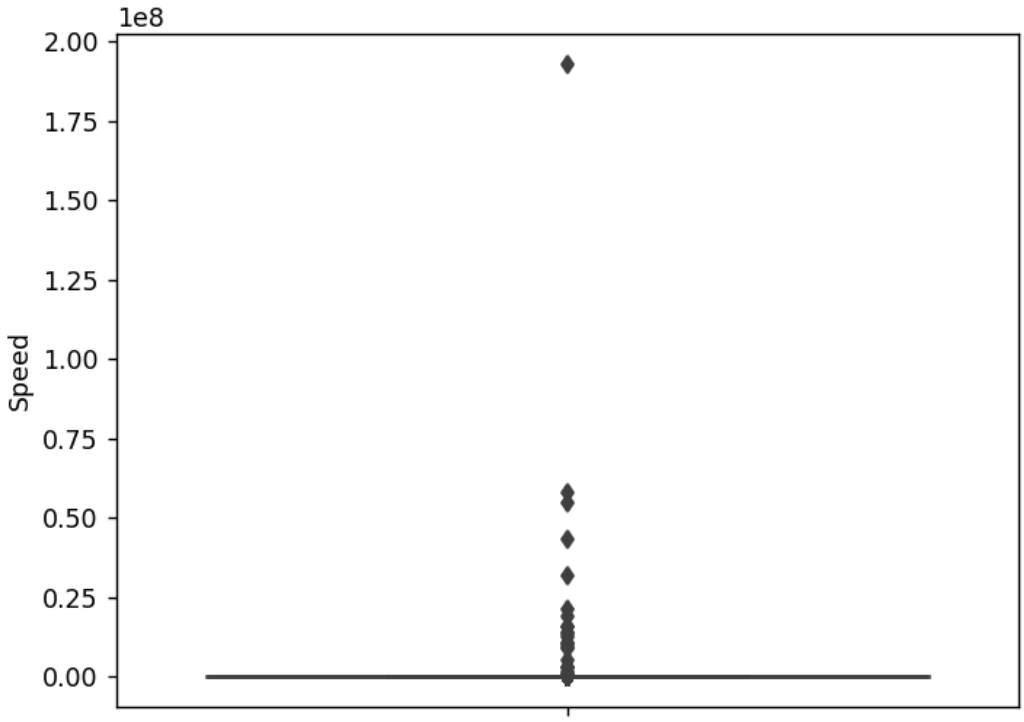


4. Speed

In [10]:

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

```
<IPython.core.display.Javascript object>
```



In [11]:

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

```
0 percentile value is 0.0
10 percentile value is 6.4
20 percentile value is 7.797833935018051
30 percentile value is 8.91640866873065
40 percentile value is 9.967130214917825
50 percentile value is 11.054739652870493
60 percentile value is 12.272727272727273
70 percentile value is 13.783375314861463
80 percentile value is 15.949367088607593
90 percentile value is 20.170212765957448
100 percentile value is  192857142.85714284
```

In [12]:

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

```
90 percentile value is 20.170212765957448
91 percentile value is 20.897097625329817
92 percentile value is 21.732283464566926
93 percentile value is 22.7027027027027
94 percentile value is 23.822904368358916
95 percentile value is 25.161290322580644
96 percentile value is 26.78659035159444
97 percentile value is 28.82058613295211
98 percentile value is 31.57377049180328
99 percentile value is 35.73529411764706
100 percentile value is  192857142.85714284
```

In [13]:

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

```
99.0 percentile value is 35.73529411764706
99.1 percentile value is 36.29530201342282
99.2 percentile value is 36.89750692520775
99.3 percentile value is 37.57178841309823
99.4 percentile value is 38.314014752370916
99.5 percentile value is 39.159263271939324
99.6 percentile value is 40.140350877192986
99.7 percentile value is 41.32442748091603
99.8 percentile value is 42.85579937304075
99.9 percentile value is 45.299777942264996
100 percentile value is  192857142.85714284
```

In [11]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0) &
```

In [12]:

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

Out[12]:

```
12.437081883528638
```

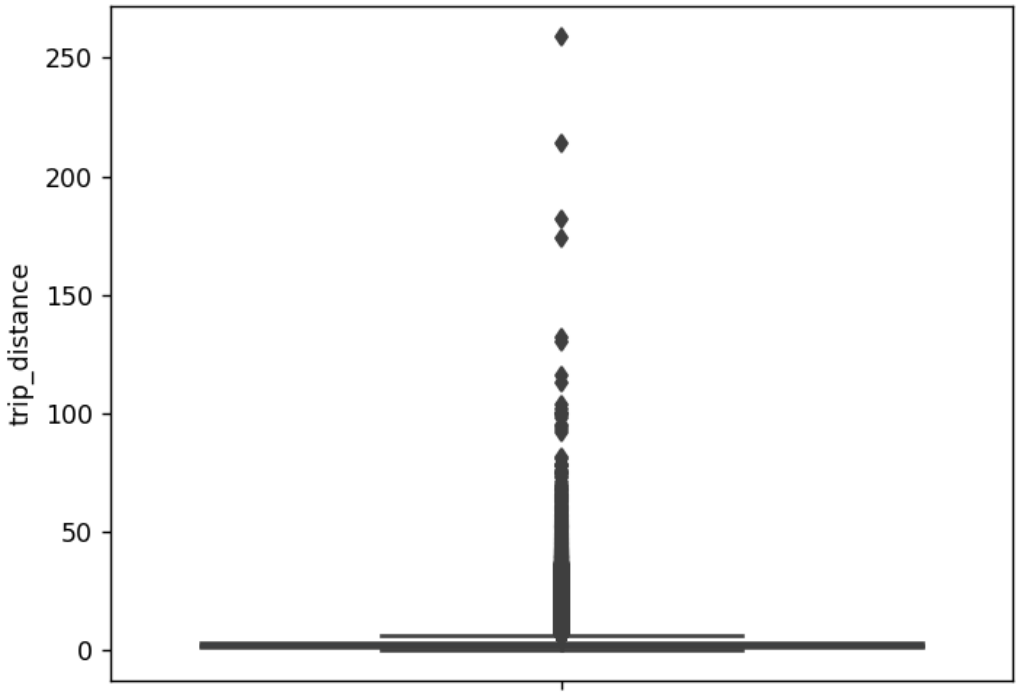
The avg speed in Newyork speed is 12.45miles/hr, so a cab driver can travel 2 miles per 10min on avg.

4. Trip Distance

In [30]:

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

```
<IPython.core.display.Javascript object>
```



In [16]:

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

```
0 percentile value is 0.01
10 percentile value is 0.66
20 percentile value is 0.9
30 percentile value is 1.1
40 percentile value is 1.38
50 percentile value is 1.69
60 percentile value is 2.07
70 percentile value is 2.6
80 percentile value is 3.59
90 percentile value is 5.96
100 percentile value is  258.9
```

In [17]:

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

```
90 percentile value is 5.96
91 percentile value is 6.44
92 percentile value is 7.06
93 percentile value is 7.83
94 percentile value is 8.71
95 percentile value is 9.6
96 percentile value is 10.6
97 percentile value is 12.1
98 percentile value is 16.0
99 percentile value is 18.16
100 percentile value is  258.9
```

In [18]:

```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,9
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 18.16
99.1 percentile value is 18.35
99.2 percentile value is 18.59
99.3 percentile value is 18.82
99.4 percentile value is 19.12
99.5 percentile value is 19.5
99.6 percentile value is 19.94
99.7 percentile value is 20.5
99.8 percentile value is 21.21
99.9 percentile value is 22.57
100 percentile value is  258.9
```

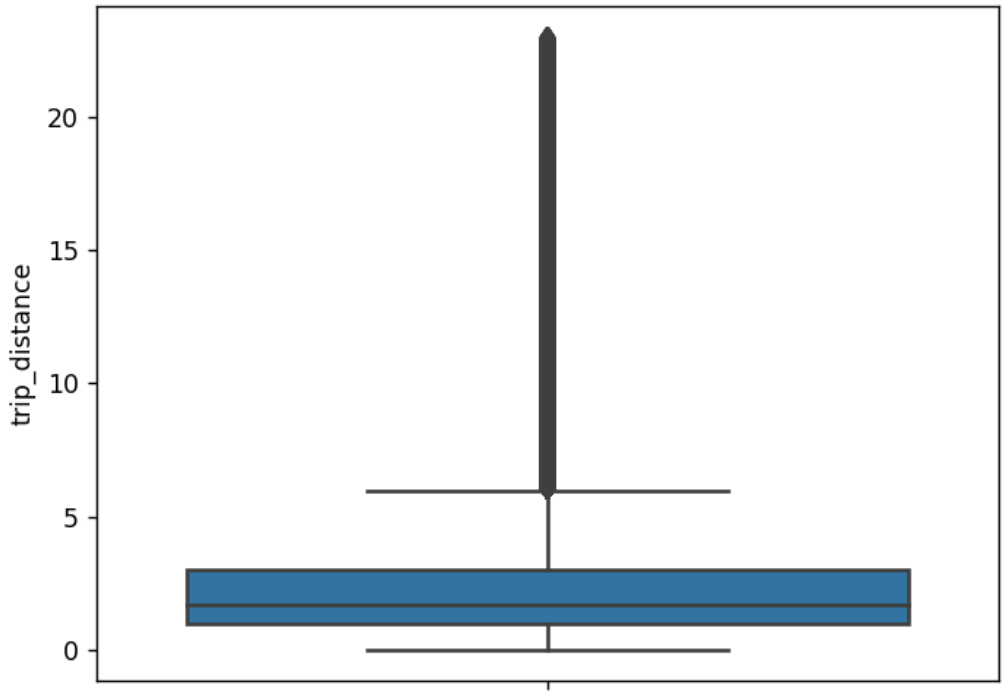
In [13]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_dista
```

In [35]:

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

<IPython.core.display.Javascript object>

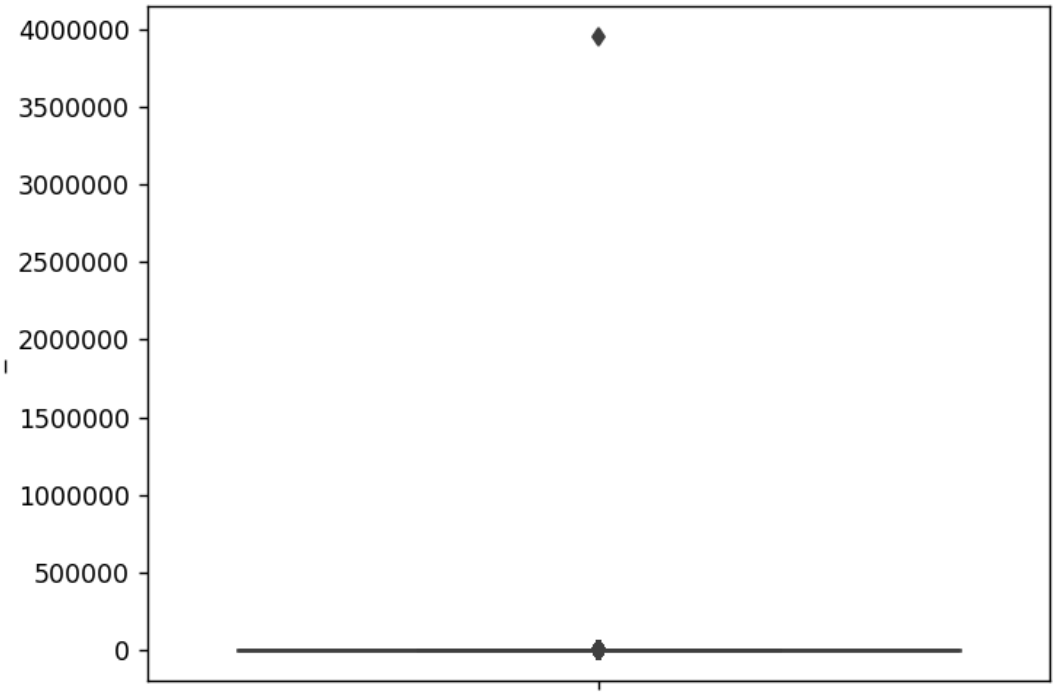


5. Total Fare

In [36]:

```
# up to now we have removed the outliers based on trip durations, cab speeds, and t
# lets try if there are any outliers in based on the total_amount
# box-plot showing outliers in fare
sns.boxplot(y="total_amount", data =frame_with_durations_modified)
plt.show()
```

```
<IPython.core.display.Javascript object>
```



In [20]:

```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80,9
for i in range(0,100,10):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is -242.55
10 percentile value is 6.31
20 percentile value is 7.8
30 percentile value is 8.8
40 percentile value is 9.8
50 percentile value is 11.16
60 percentile value is 12.8
70 percentile value is 14.8
80 percentile value is 18.3
90 percentile value is 25.8
100 percentile value is  nan
```

In [21]:

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

```
90 percentile value is 25.8
91 percentile value is 27.3
92 percentile value is 29.3
93 percentile value is 31.8
94 percentile value is 34.8
95 percentile value is 38.5
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  nan
```

In [22]:

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

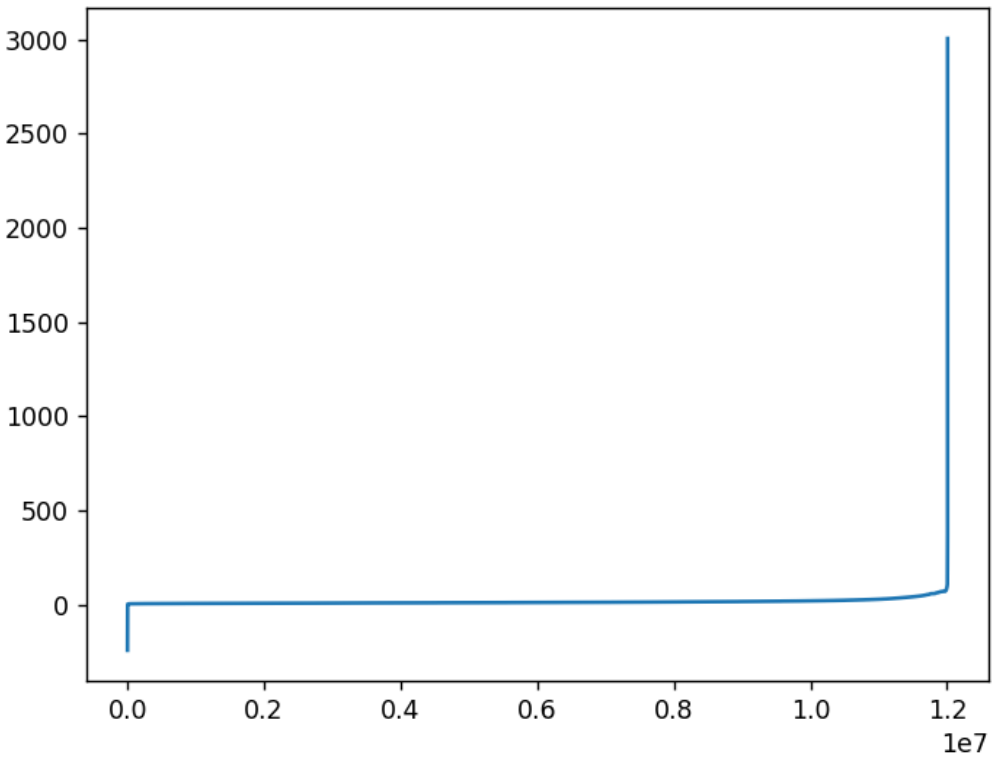
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value is 88.3
100 percentile value is  nan
```

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 [42]:

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

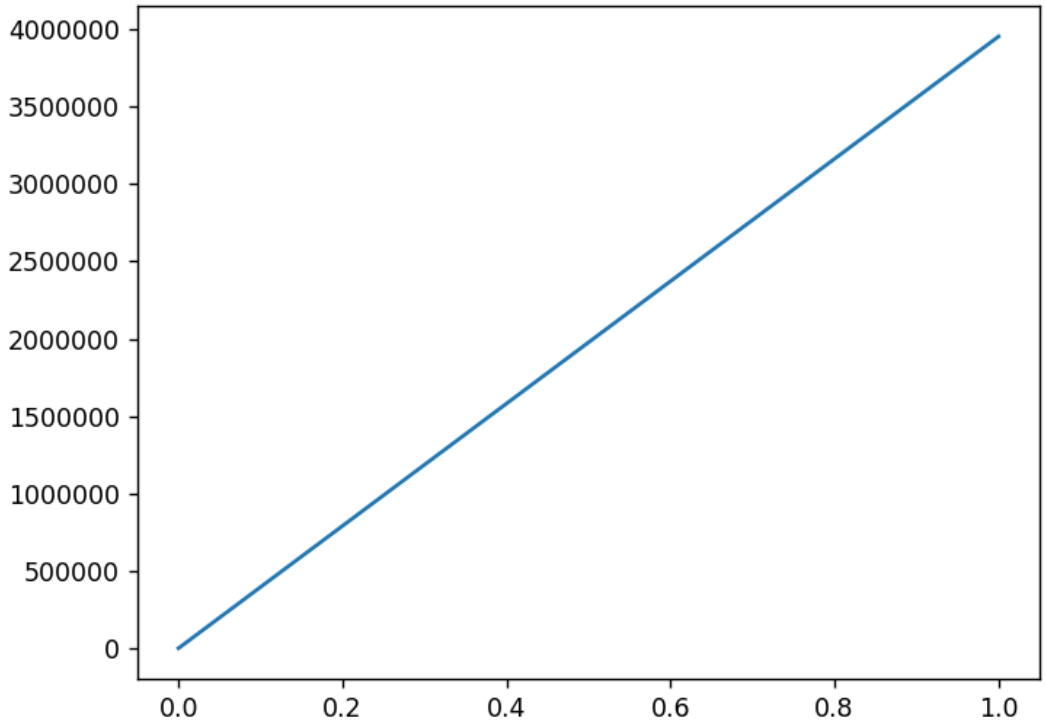
```
<IPython.core.display.Javascript object>
```



In [43]:

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

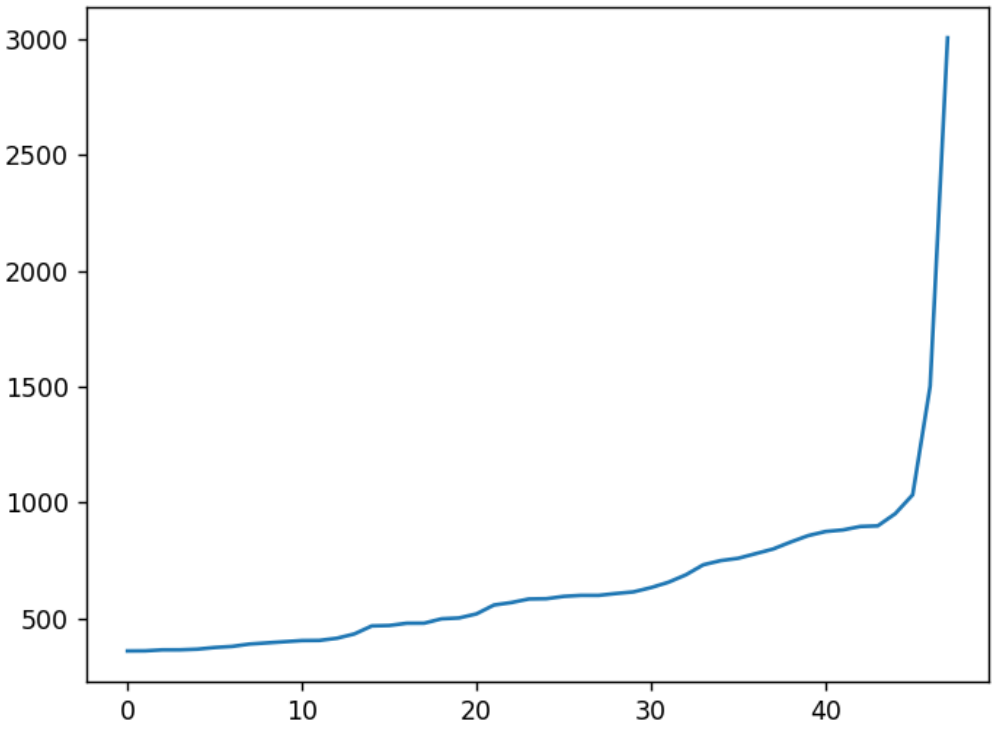
```
<IPython.core.display.Javascript object>
```



In [44]:

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

<IPython.core.display.Javascript object>



Remove all outliers/erronous points.

In [14]:

```
#removing all outliers based on our univariate analysis above
def remove_outliers(new_frame):


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


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


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

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

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


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dro
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup

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

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

In [15]:

```
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(fram
```

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12096245
Number of outlier coordinates lying outside NY boundaries: 278687
Number of outliers from trip times analysis: 22580
Number of outliers from trip distance analysis: 87766
Number of outliers from speed analysis: 23181
Number of outliers from fare analysis: 4987
Total outliers removed 358382
---
fraction of data points that remain after removing outliers 0.9703724
585604871
```

Data-preperation

Clustering/Segmentation

In [16]:

```
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitud
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], clus
                min_dist = min(min_dist,distance/(1.60934*1000))
                if (distance/(1.60934*1000)) <= 2:
                    nice_points +=1
                else:
                    wrong_points += 1
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_
    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 a
re more number of cluster regions
#that are close to any cluster center
# and make sure that the minimum inter cluster should not be very less
for increment in range(10, 100, 10):
    cluster_centers, cluster_len = find_clusters(increment)
    find_min_distance(cluster_centers, cluster_len)
```

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 8.0
Min inter-cluster distance =  0.8566670328678423
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 15.0
Min inter-cluster distance =  0.5521277391731461
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 6.0
```

```
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 23.0
Min inter-cluster distance =  0.4992108777389083
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 9.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 31.0
Min inter-cluster distance =  0.43949991451300807
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 38.0
Min inter-cluster distance =  0.24291338260858703
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 46.0
Min inter-cluster distance =  0.337315075632416
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 54.0
Min inter-cluster distance =  0.27917742578341287
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 20.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 60.0
Min inter-cluster distance =  0.2746213115844946
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distan
ce < 2): 24.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-dista
nce > 2): 66.0
Min inter-cluster distance =  0.05331995897070649
---
```

Inference:

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

In [17]:

```
# if check for the 50 clusters you can observe that there are two clusters with onl
# so we choose 40 clusters for solve the further problem

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

Plotting the cluster centers:

In [18]:

```
# 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(c
map_osm
```

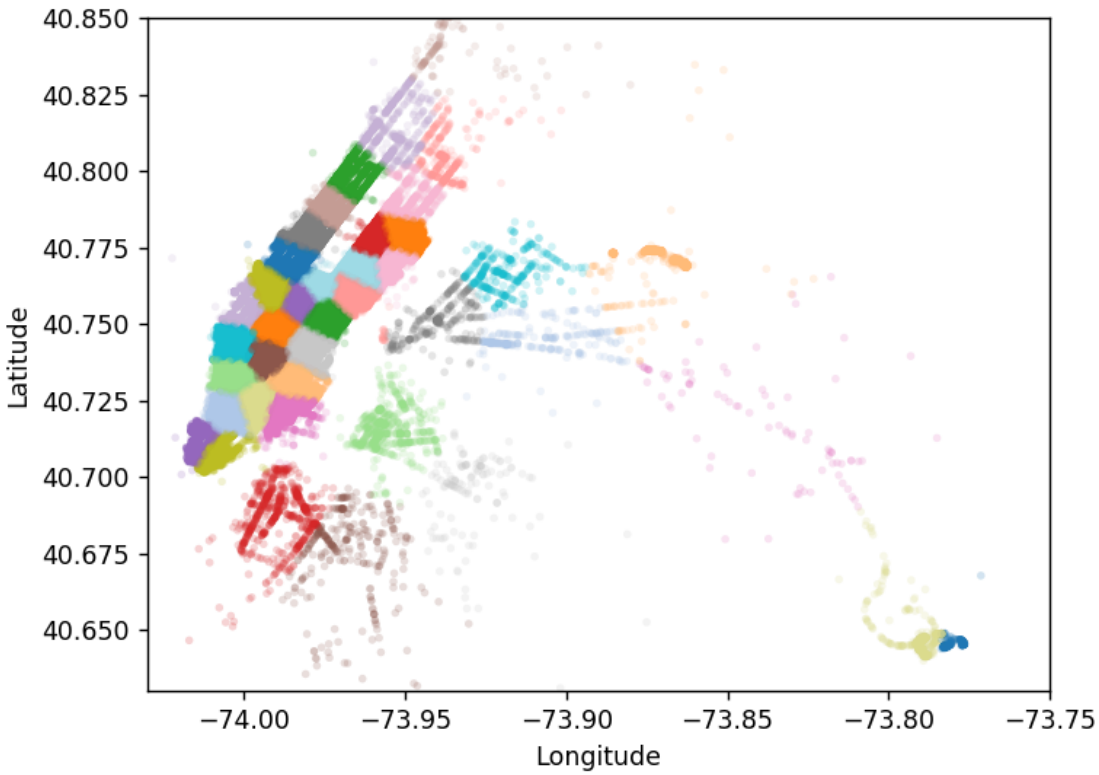
Out[18]:

Plotting the clusters:

In [19]:

```
#Visualising the clusters on a map
def plot_clusters(frame):
    city_long_border = (-74.03, -73.75)
    city_lat_border = (40.63, 40.85)
    fig, ax = plt.subplots(ncols=1, nrows=1)
    ax.scatter(frame.pickup_longitude.values[:100000], frame.pickup_latitude.values
               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)
```

```
<IPython.core.display.Javascript object>
```



Time-binning

In [20]:

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

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

def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,143311680
                   [1451606400,1454284800,1456790400,1459468800,1462060800,1464739

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

In [21]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with
jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2015)
jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance']]
```

In [22]:

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

Out[22]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
0	1.0	1.59	-73.993896	40.750111	-73.974785	
1	1.0	3.30	-74.001648	40.724243	-73.994415	
2	1.0	1.80	-73.963341	40.802788	-73.951820	
3	1.0	0.50	-74.009087	40.713818	-74.004326	
4	1.0	3.00	-73.971176	40.762428	-74.004181	

In [23]:

```
# hear the trip_distance represents the number of pickups that are happend in that
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels 24
jan_2015_groupby.head()
```

Out[23]:

		trip_distance
pickup_cluster	**pickup_bins**	
	33	170
	34	272
0	**35**	215
	36	137
	37	155

In [24]:

```
# upto now we cleaned data and prepared data for the month 2015,

# now do the same operations for months Jan, Feb, March of 2016
# 1. get the dataframe which 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_amount
# 5. add pickup_cluster to each data point
# 6. add pickup_bin (index of 10min intravel to which that trip belongs to)
# 7. group by data, based on 'pickup_cluster' and '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_durations)

    print ("Estimating clusters..")
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,mon
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip

    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
```

Smoothing

In [25]:

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

# for each cluster region we will collect all the indices of 10min intravels in whi
# we got an observation that there are some pickpbins that doesnt have any pickups
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,40):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [26]:

```
# for every month we get all indices of 10min intravels in which atleast one pickup

#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 [27]:

```
# 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: ",44
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  21
1
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  80
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  13
5
------------------------------------------------------------
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:  32
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  3
3
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  1
18
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  6
0
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  5
9
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  5
8
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  2
7
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  6
3
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  3
1
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  4
4
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  6
4
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  1
```

```
81
-----------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  3
6
-----------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  4
8
-----------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  2
86
-----------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  3
7
-----------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  6
41
-----------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  3
5
-----------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  4
2
-----------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  4
7
-----------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  5
9
-----------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1
173
-----------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  3
7
-----------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  4
6
-----------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  3
4
-----------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  4
94
-----------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  3
5
-----------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  4
1
-----------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  7
7
-----------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  3
6
-----------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  4
9
-----------------------------------------------------------
```

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 [28]:

```
# Fills a value of zero for every bin where no pickup data is present
# the count_values: number pickps that are happened in each region for each 10min i
# 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 [29]:

```
# Fills a value of zero for every bin where no pickup data is present
# the count_values: number pickps that are happened in each region for each 10min i
# 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 di
# 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
                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
            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
                            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
                        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 valu
                        smoothed_value=(count_values[ind-1]+count_values[ind])*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 b
                    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
                    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 [30]:

```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pic
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015_uniq

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uniqu
```

In [31]:

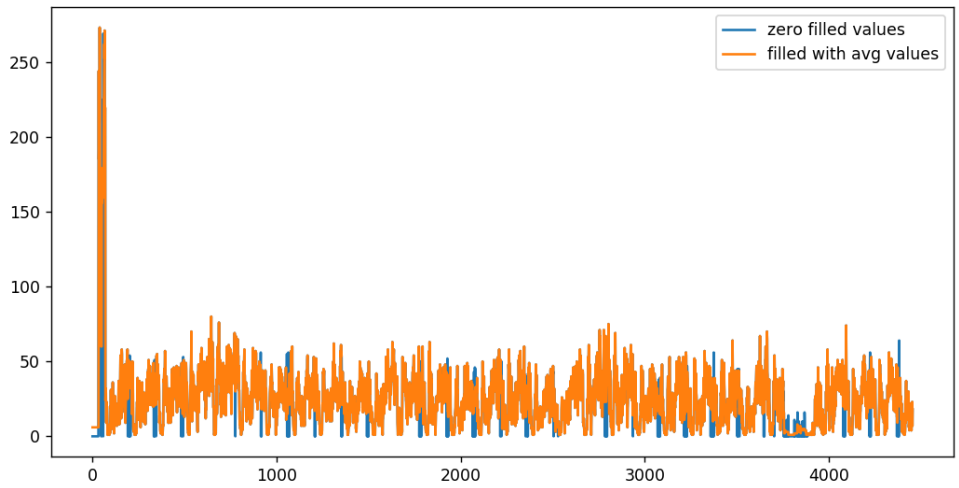
```
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*30*60/10 = 4320
# for each cluster we will have 4464 values, therefore 40*4464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

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

In [63]:

```
# 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()
```

```
<IPython.core.display.Javascript object>
```



In [0]:

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

# Ans: consider we have data of some month in 2015 jan 1st, 10 _ _ _ 20, i.e there
# 10st 10min intravel, 0 pickups happened in 2nd 10mins intravel, 0 pickups happene
# and 20 pickups happened in 4th 10min intravel.
# in fill_missing method we replace these values like 10, 0, 0, 20
# where as in smoothing method we replace these values as 6,6,6,6,6, if you can che
# that are happened in the first 40min are same in both cases, but if you can obser
# wheen you are using smoothing we are looking at the future number of pickups whic

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

In [32]:

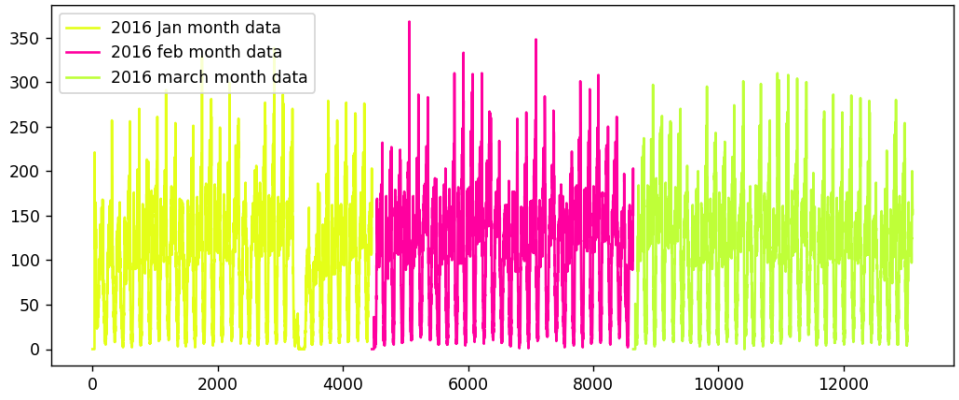
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled wi
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uniqu
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_un
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_un
mar_2016_smooth = fill_missing(mar_2016_groupby['trip_distance'].values,mar_2016_un

# Making list of all the values of pickup data in every bin for a period of 3 month
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 valu
# 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:41

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

Time series and Fourier Transforms

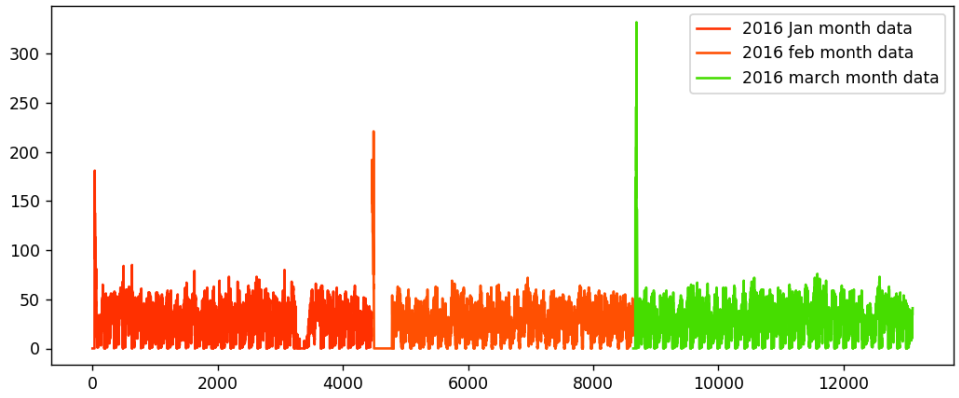
In [42]:

```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40):
    plt.figure(figsize=(10,4))
    plt.plot(first_x,regions_cum[i][:4464], color=uniqueish_color(), label='2016 Ja
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), label='20
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='2016 ma
    plt.legend()
    plt.show()
```

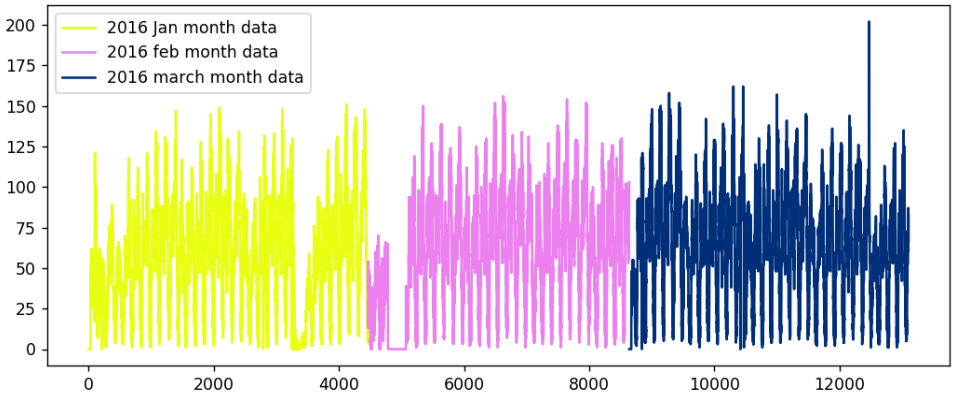
<IPython.core.display.Javascript object>



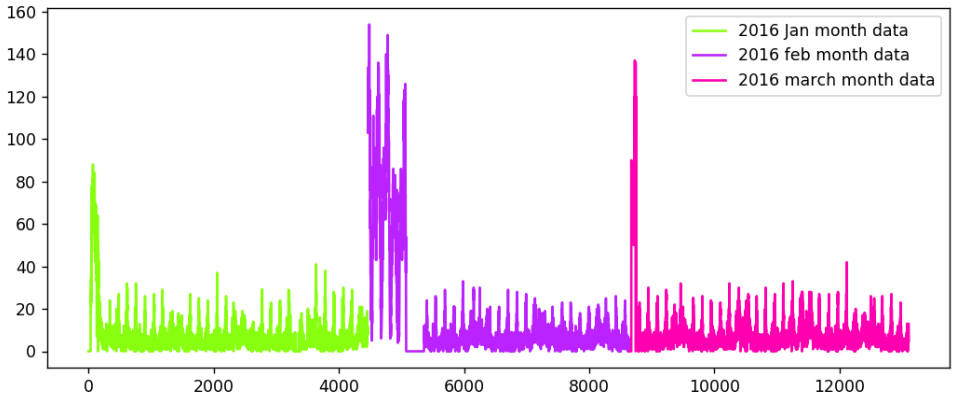
<IPython.core.display.Javascript object>



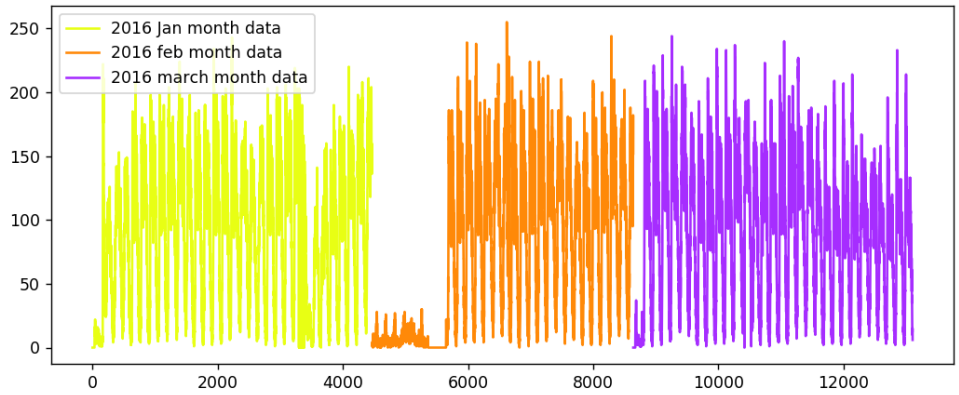
```
<IPython.core.display.Javascript object>
```



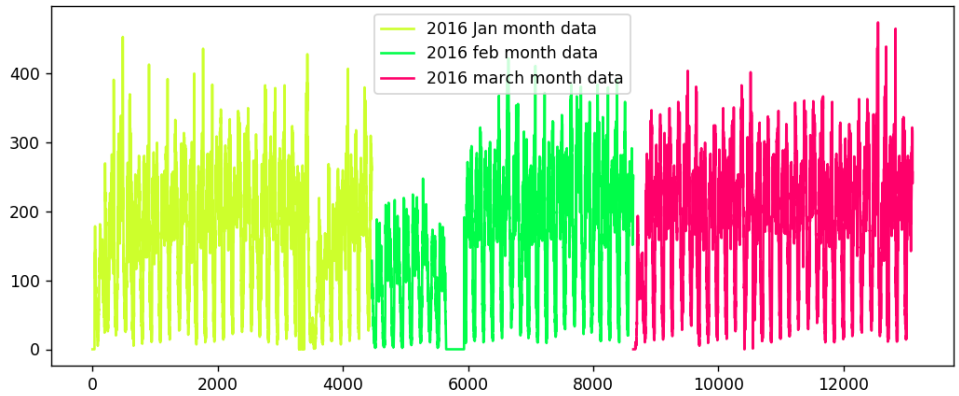

```
<IPython.core.display.Javascript object>
```



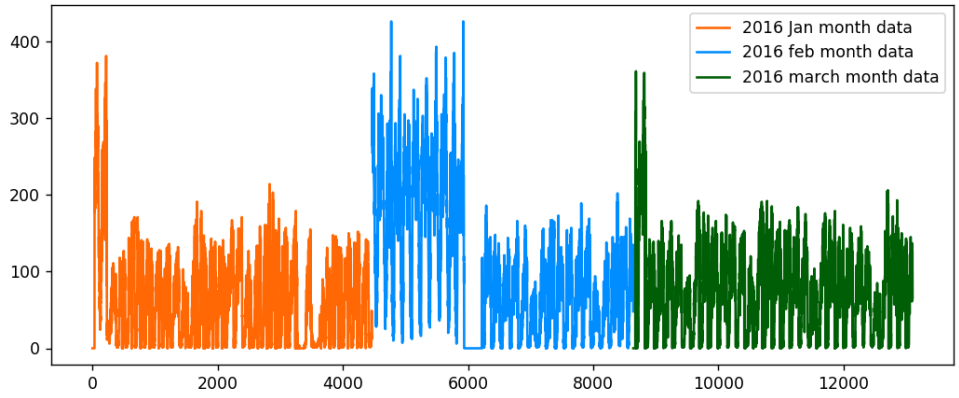

```
<IPython.core.display.Javascript object>
```



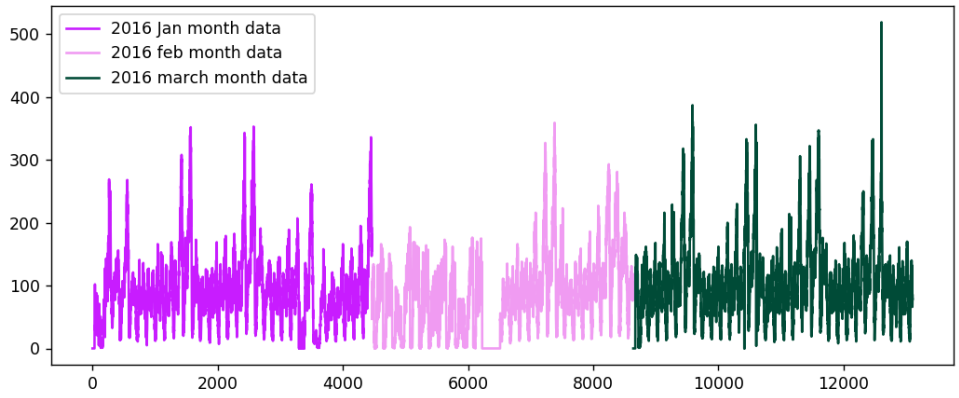
<IPython.core.display.Javascript object>



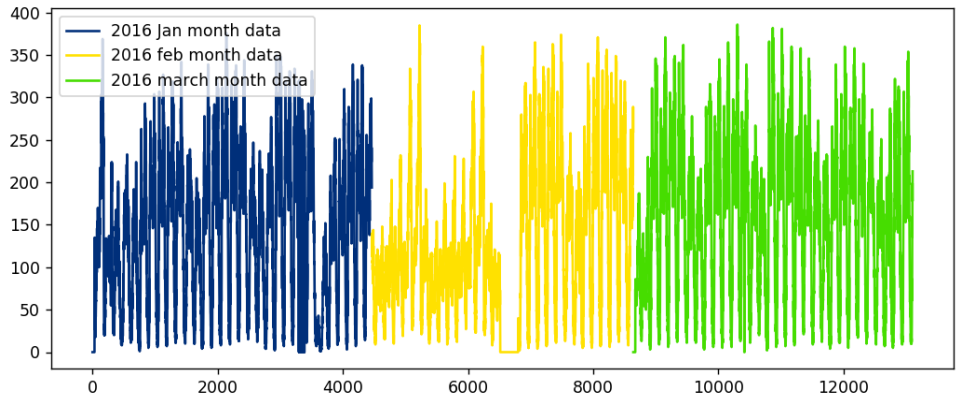
<IPython.core.display.Javascript object>



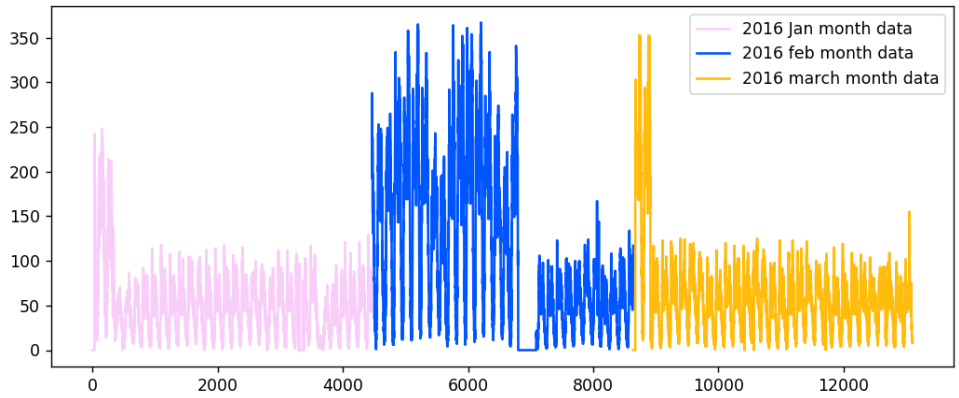

```
<IPython.core.display.Javascript object>
```



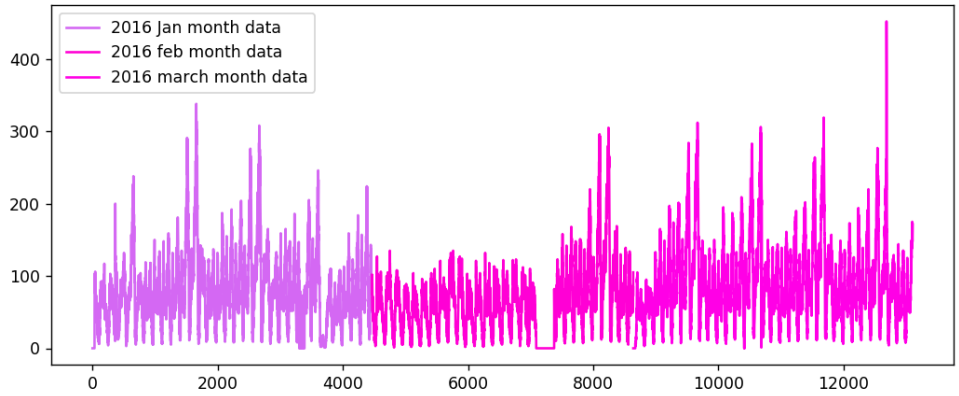

```
<IPython.core.display.Javascript object>
```



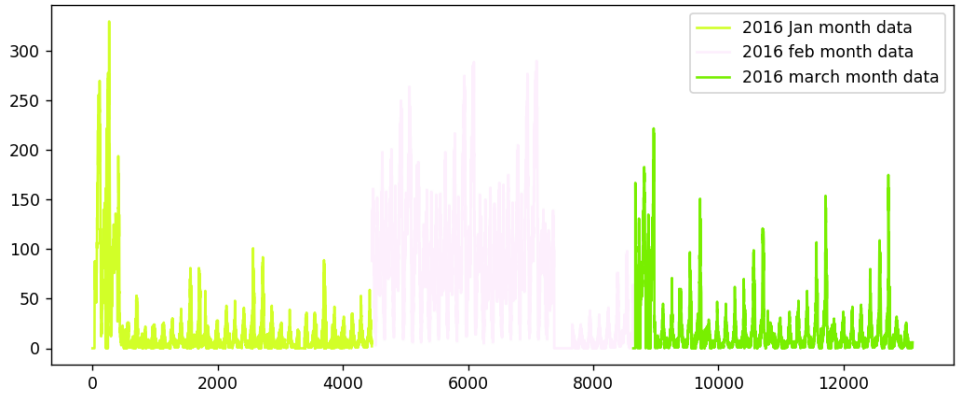
```
<IPython.core.display.Javascript object>
```



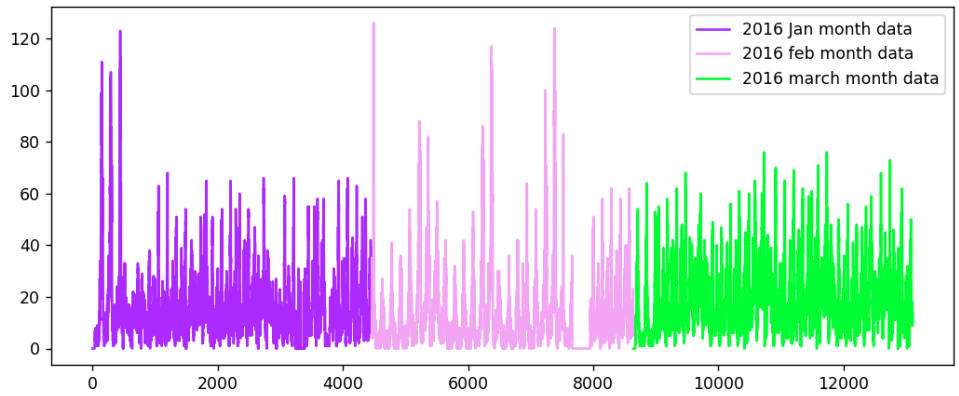

```
<IPython.core.display.Javascript object>
```



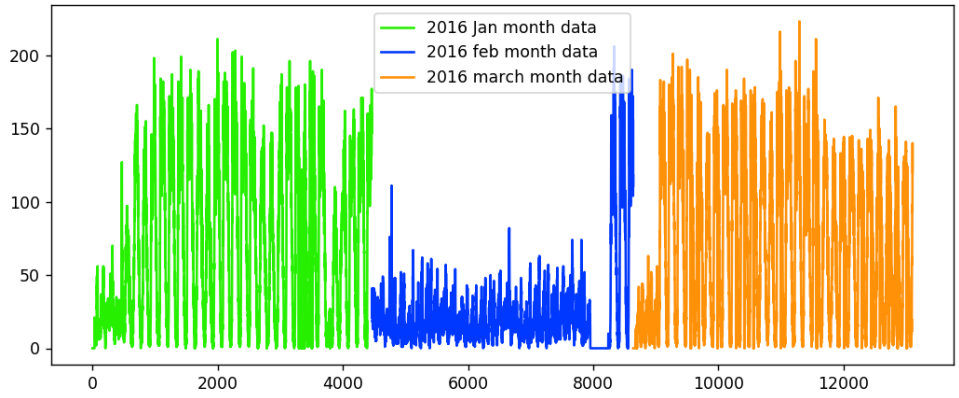

```
<IPython.core.display.Javascript object>
```



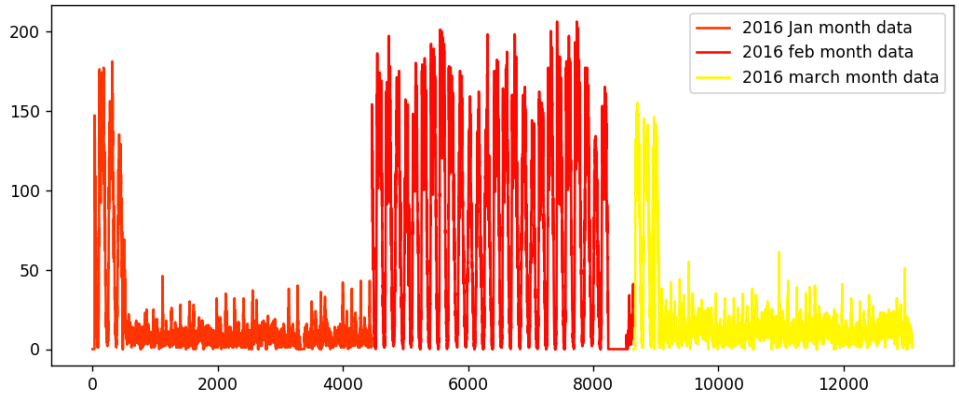
<IPython.core.display.Javascript object>



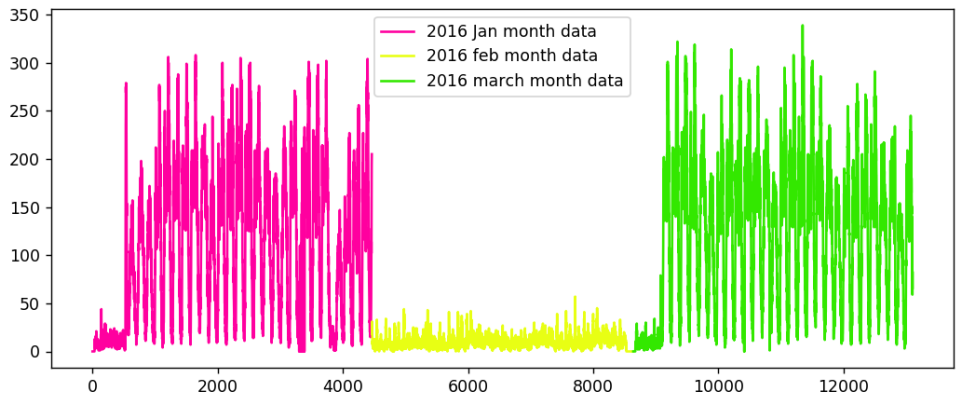
<IPython.core.display.Javascript object>



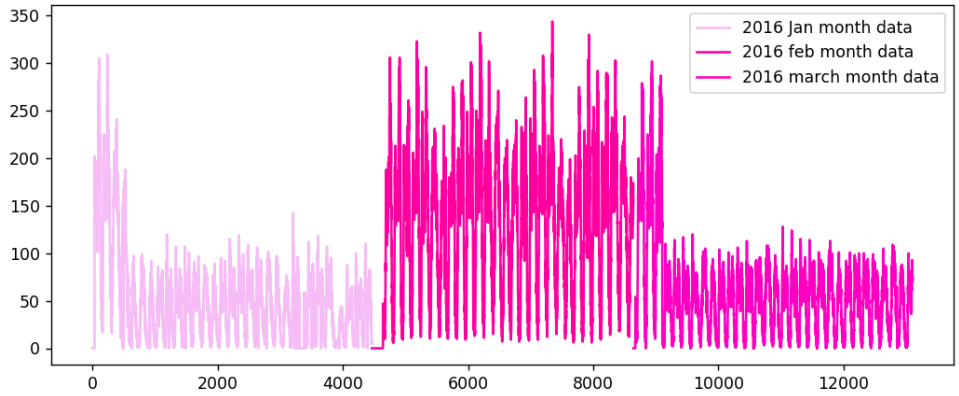

```
<IPython.core.display.Javascript object>
```



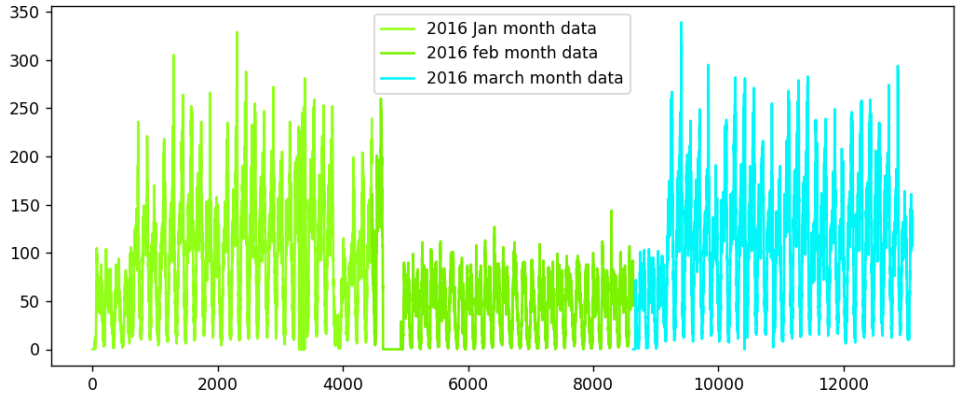

```
<IPython.core.display.Javascript object>
```



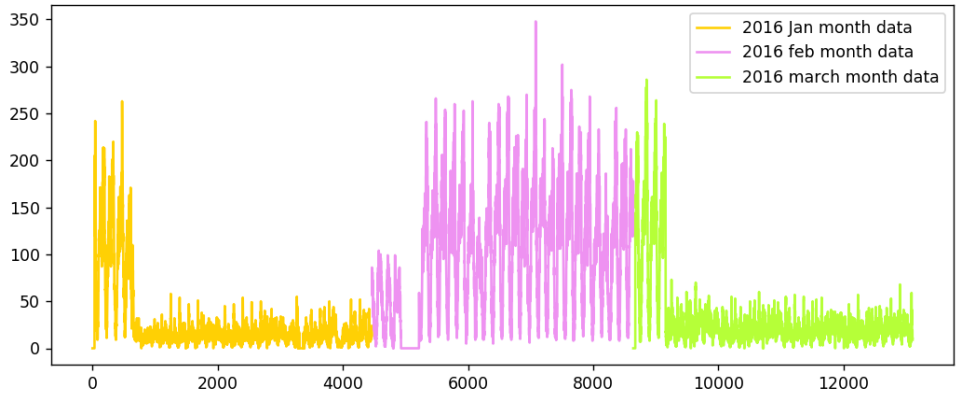
```
<IPython.core.display.Javascript object>
```



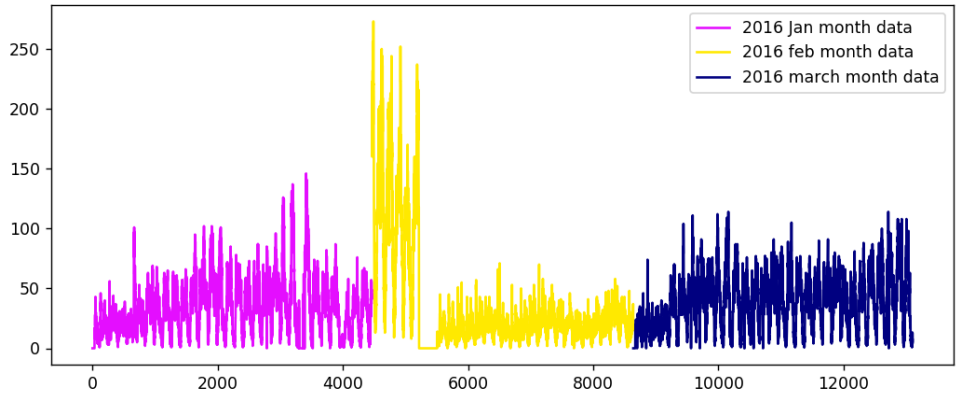

```
<IPython.core.display.Javascript object>
```



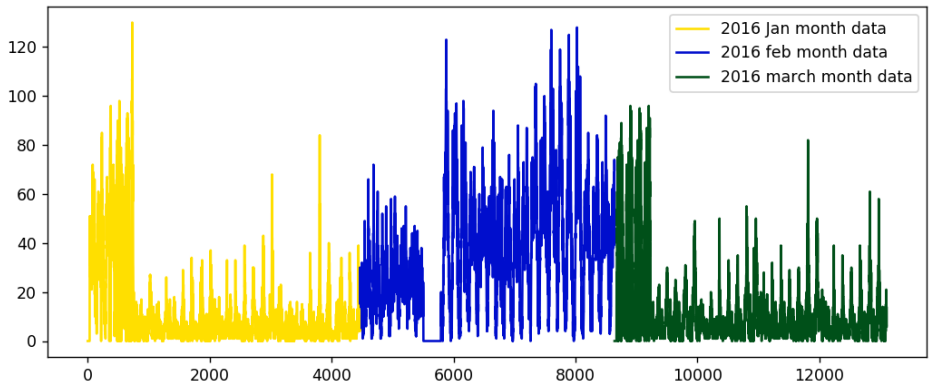

```
<IPython.core.display.Javascript object>
```



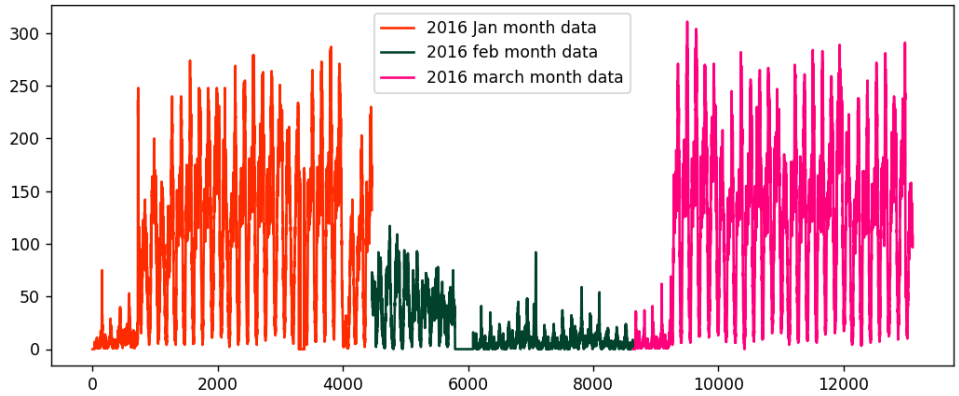
<IPython.core.display.Javascript object>



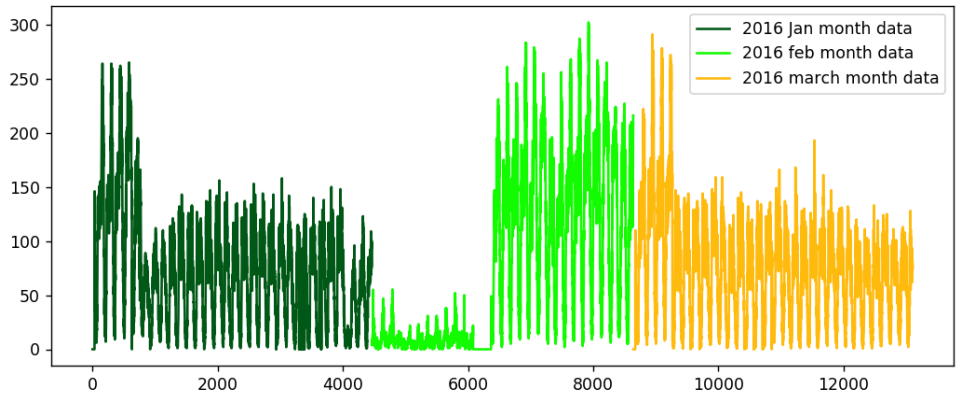
<IPython.core.display.Javascript object>



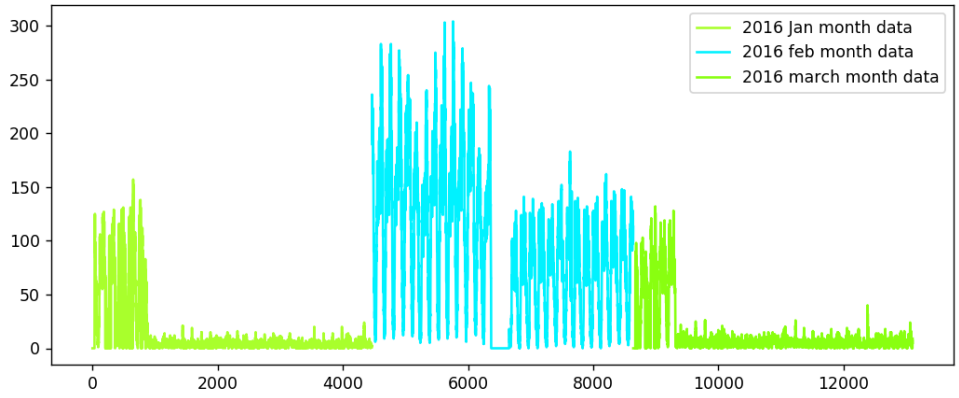
<IPython.core.display.Javascript object>



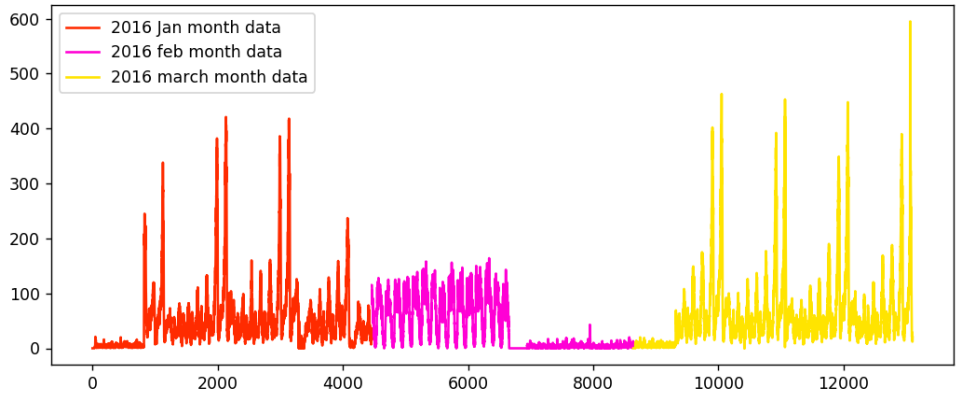
<IPython.core.display.Javascript object>



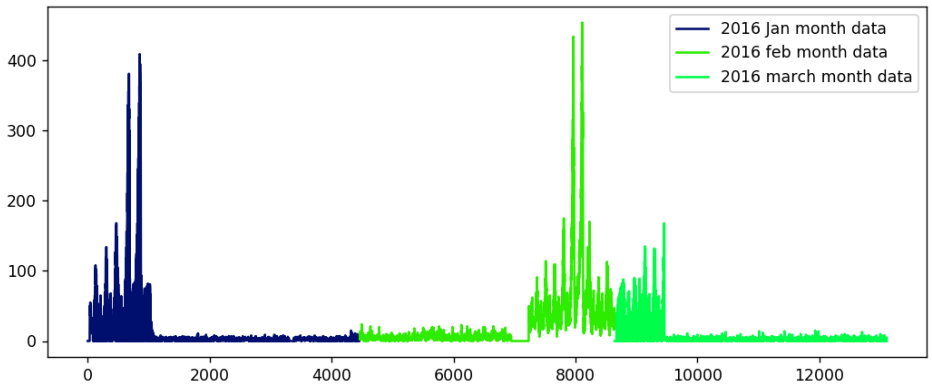
<IPython.core.display.Javascript object>



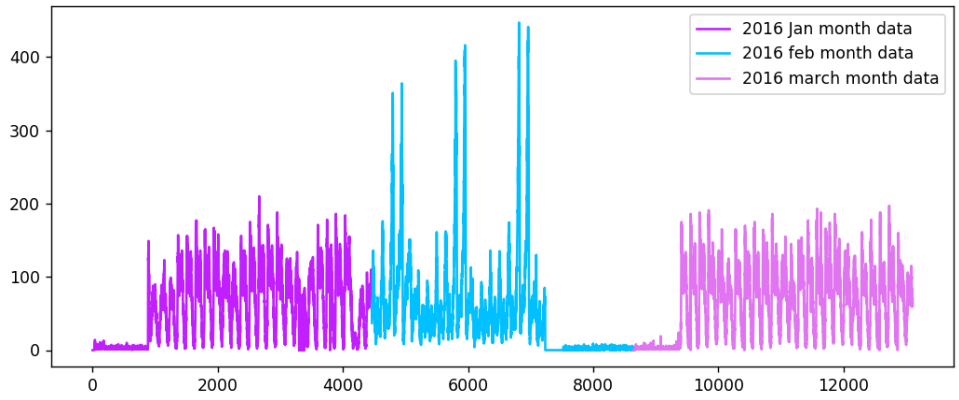
<IPython.core.display.Javascript object>



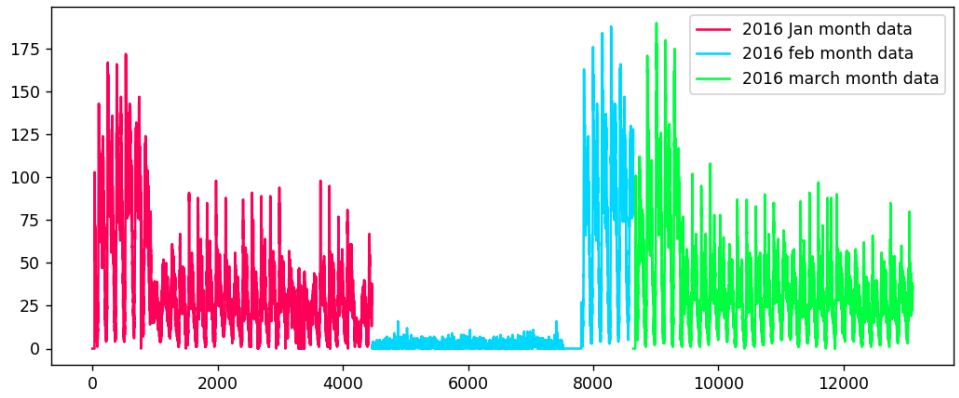
<IPython.core.display.Javascript object>



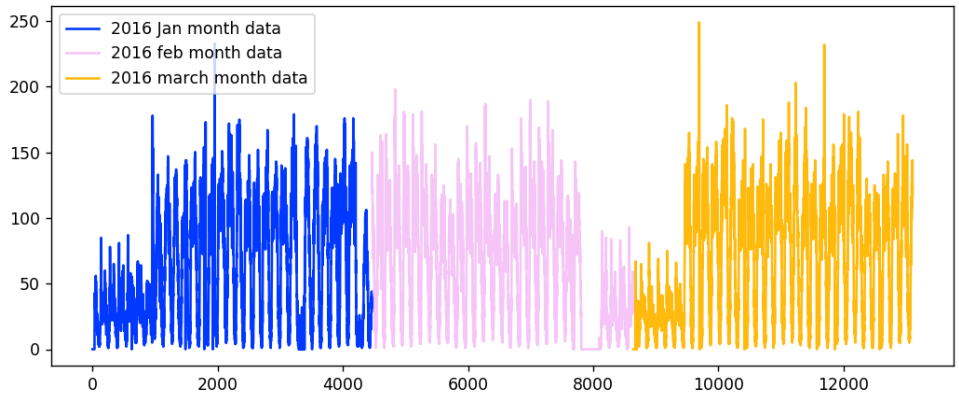

```
<IPython.core.display.Javascript object>
```



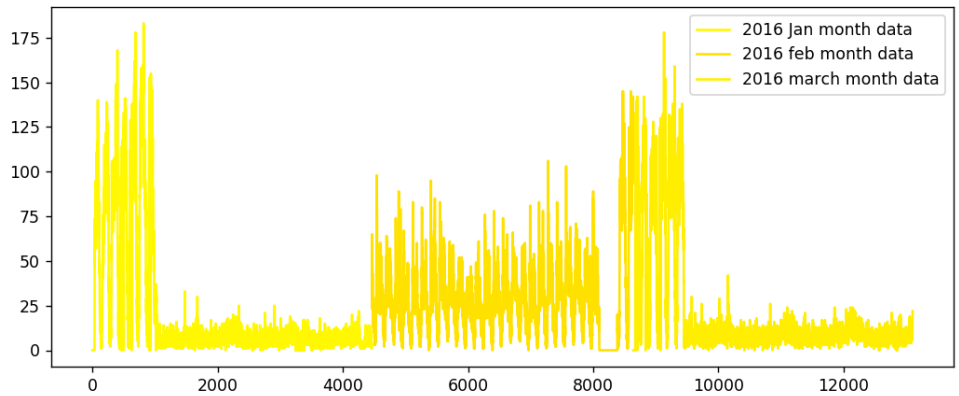

```
<IPython.core.display.Javascript object>
```



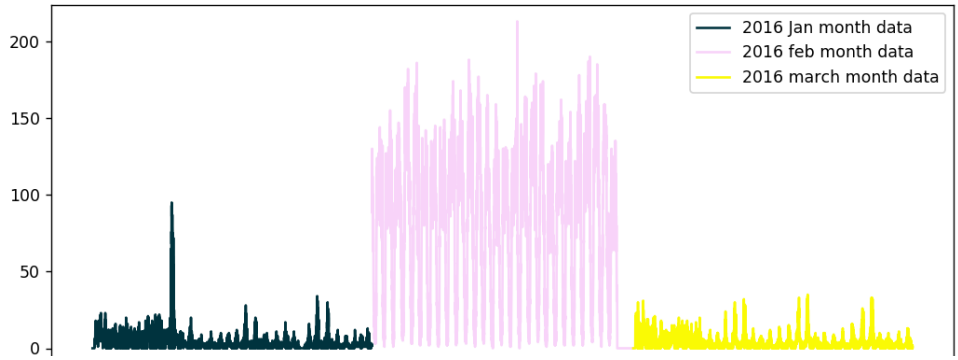
```
<IPython.core.display.Javascript object>
```



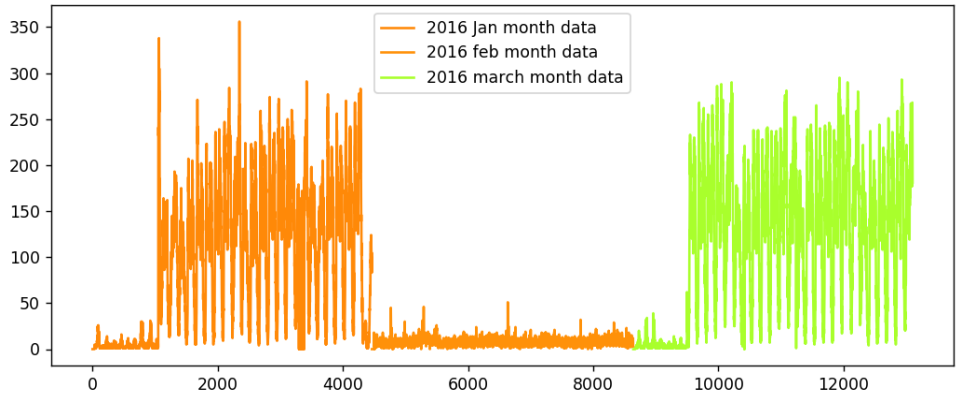

```
<IPython.core.display.Javascript object>
```



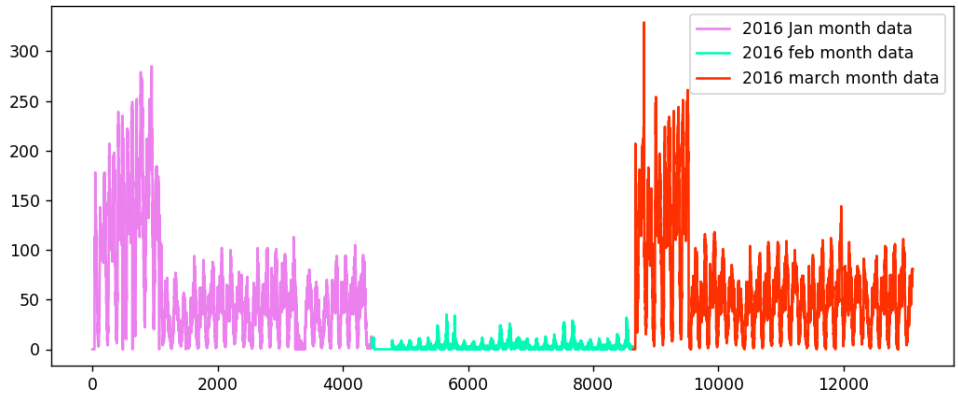

```
<IPython.core.display.Javascript object>
```



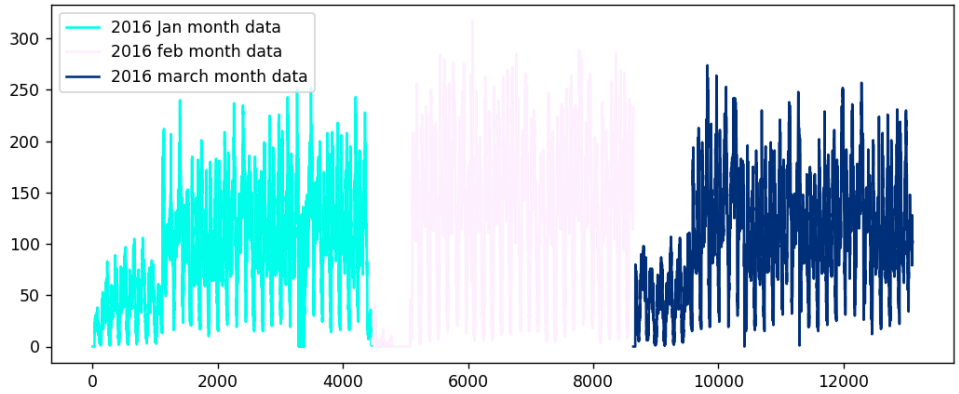

```
<IPython.core.display.Javascript object>
```



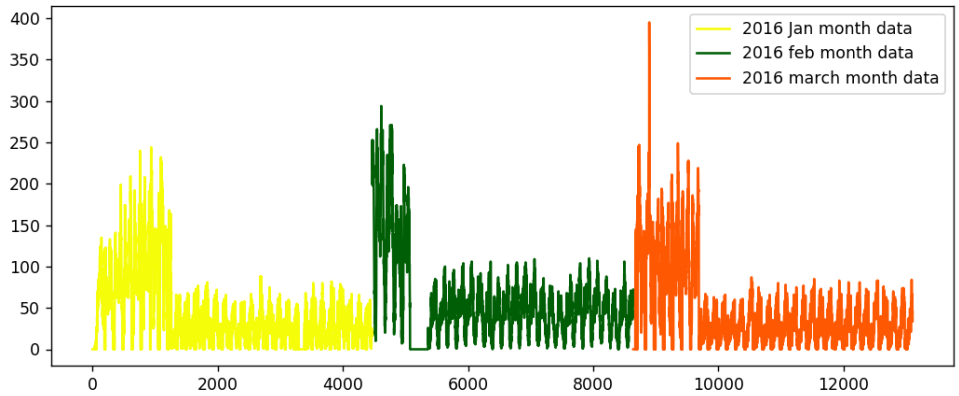

```
<IPython.core.display.Javascript object>
```



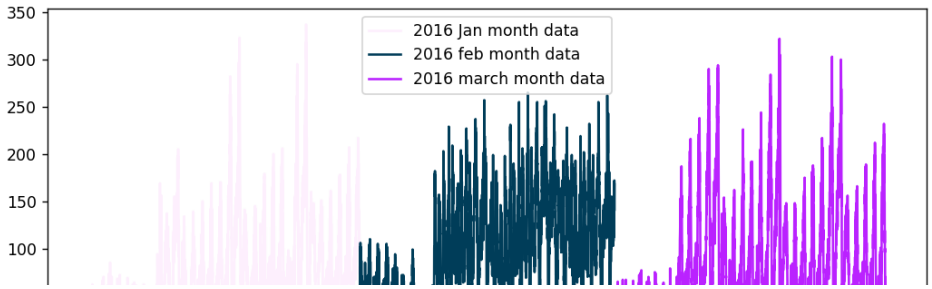
```
<IPython.core.display.Javascript object>
```



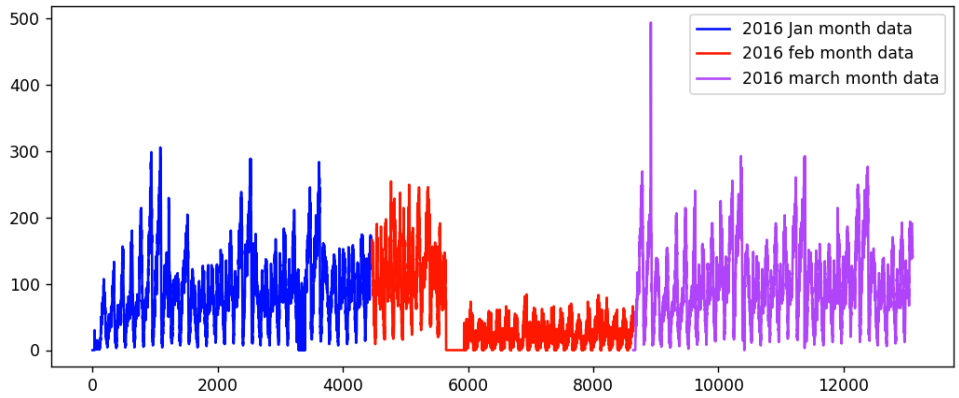

```
<IPython.core.display.Javascript object>
```



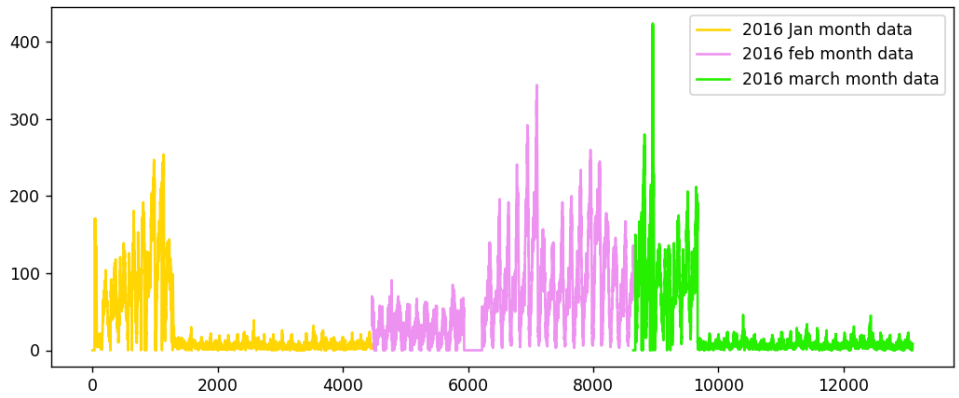

```
<IPython.core.display.Javascript object>
```



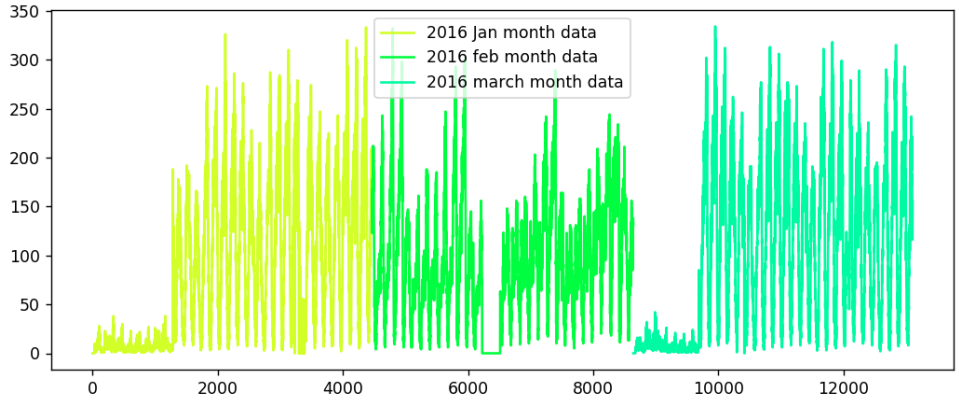

```
<IPython.core.display.Javascript object>
```



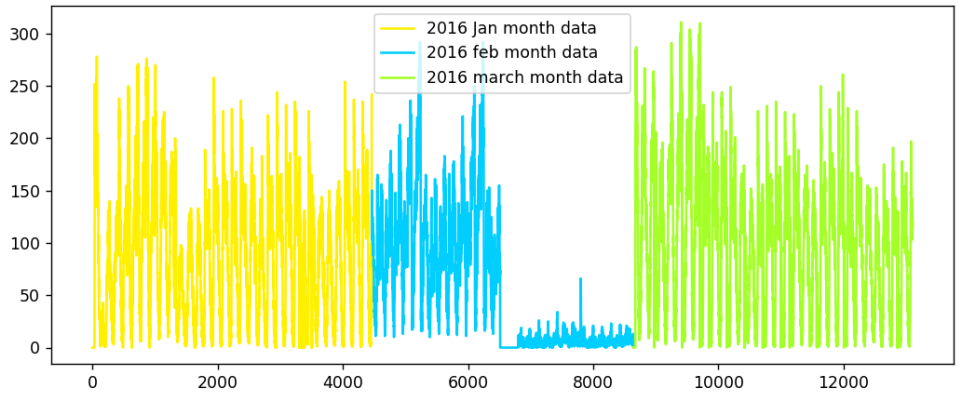

```
<IPython.core.display.Javascript object>
```



```
<IPython.core.display.Javascript object>
```



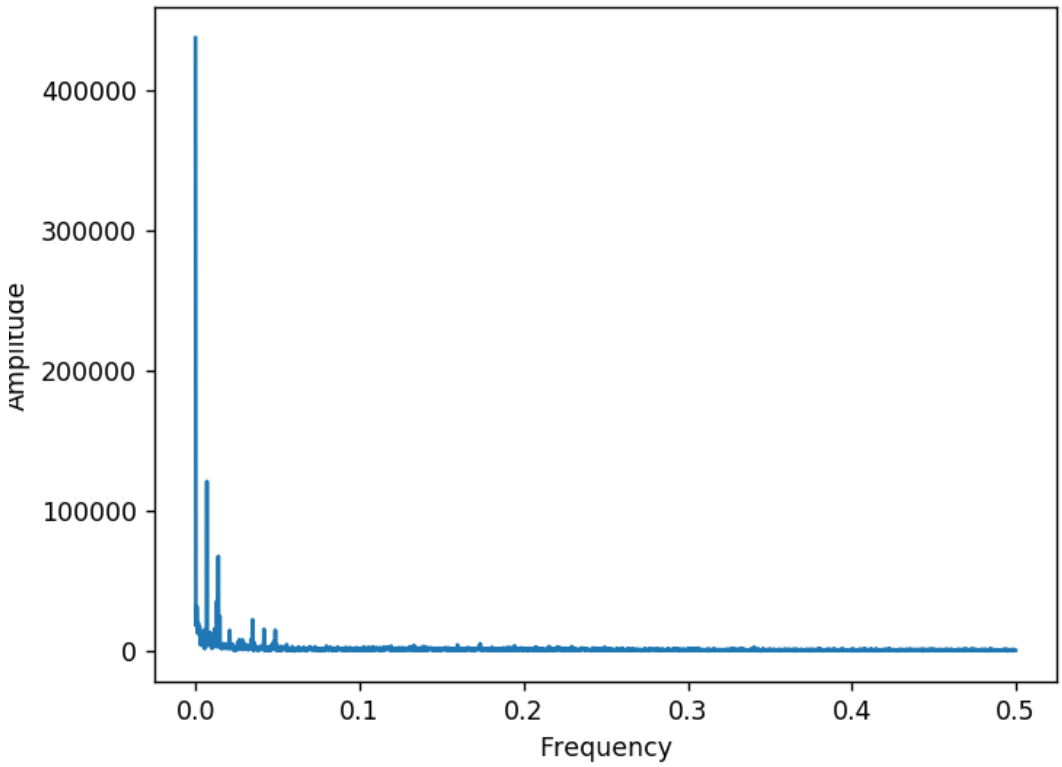

```
<IPython.core.display.Javascript object>
```



In [33]:

```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generat
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated
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()
```

```
<IPython.core.display.Javascript object>
```



In [34]:

```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) values as
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 [35]:

```
def MA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    error=[]
    predicted_values=[]
    window_size=3
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_rati
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


    ratios['MA_R_Predicted'] = predicted_values
    ratios['MA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios
    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 [36]:

```
def MA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=1
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i]
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_size
        else:
            predicted_value=int(sum((ratios['Prediction'].values)[0:(i+1)])/(i+1))

    ratios['MA_P_Predicted'] = predicted_values
    ratios['MA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios
    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 -

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

In [37]:

```
def WA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_rati
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

    ratios['WA_R_Predicted'] = predicted_values
    ratios['WA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios
    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 -

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

In [38]:

```
def WA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,4464*40):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i]
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Prediction'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

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

    ratios['WA_P_Predicted'] = predicted_values
    ratios['WA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios
    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 [39]:

```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_rati
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].va

    ratios['EA_R1_Predicted'] = predicted_values
    ratios['EA_R1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios
    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 [40]:

```
def EA_P1_Predictions(ratios,month):
    predicted_value= (ratios['Prediction'].values)[0]
    alpha=0.3
    error=[]
    predicted_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i]
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Predicti

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

In [41]:

```
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 [42]:

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------
---------------------------------
Moving Averages (Ratios) -                              MAPE:  0.24386
001037861377       MSE:  1667.3306731630823
Moving Averages (2016 Values) -                         MAPE:  0.15478
490556194502        MSE:  275.152105734767
--------------------------------------------------------------------
---------------------------------
Weighted Moving Averages (Ratios) -                     MAPE:  0.24225
742751082818       MSE:  1389.410635080645
Weighted Moving Averages (2016 Values) -                MAPE:  0.14728
692354640144       MSE:  242.79684699820788
--------------------------------------------------------------------
---------------------------------
Exponential Moving Averages (Ratios) -               MAPE:  0.24252931
634713543       MSE:  1331.1140737007167
Exponential Moving Averages (2016 Values) -          MAPE:  0.14690650
765358507       MSE:  239.8966565860215
```

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 [43]:

```
# Preparing data to be split into train and test, The below prepares data in cumula
# 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 valu
# that are happened for three months in 2016 data

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

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

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


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


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

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

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


tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13099)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13099)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of
    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,x
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stam
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [44]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(ts
```

Out[44]:

True

In [45]:

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

# upto now we computed 8 features for every data point that starts from 50th min of
# 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 gives us
# we will try to add the same exponential weighted moving avarage at t as a feature
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.3

# it is a temporary array that store exponential weighted moving avarage for each 1
# for each cluster it will get reset
# for every cluster it contains 13104 values
predicted_values=[]

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=[]
```

Fourier Features (Amplitude and Frequencies) - Adding new Features

In [46]:

```
amplitude = []
frequency = []
for i in range(40):
    amp  = np.abs(np.fft.fft(regions_cum[i][0:13104]))#amplitude calculation
    freq = np.abs(np.fft.fftfreq(13104, 1)) #frequencies calculation
    amp_indices = np.argsort(-amp)[1:]    #sorting amplitude
    amp_values = []
    freq_values = []
    for j in range(0, 9, 2):   #taking top 5 amplitudes and frequencies
        amp_values.append(amp[amp_indices[j]])
        freq_values.append(freq[amp_indices[j]])
    for k in range(13104):    #those top 5 frequencies and amplitudes are same for
        amplitude.append(amp_values)
        frequency.append(freq_values)
```

we have obtained 5 frequencies corresponding to their highest amplitude values and 5 amplitudes fourier features. We will encorporate these to our final feature set.

In [47]:

```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 mont
# 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 [48]:

```
#for train and test data fourier frequencies
train_frequencies = [frequency[i*13099:(13099*i+9169)] for i in range(0,40)]
test_frequencies = [frequency[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

In [49]:

```
#for train and test data fourier amplitudes
train_amplitudes = [amplitude[i*13099:(13099*i+9169)] for i in range(0,40)]
test_amplitudes = [amplitude[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

In [50]:

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

In [51]:

```
print("Number of data clusters",len(train_features), "Number of data points in tria
print("Number of data clusters",len(train_features), "Number of data points in test
```

```
Number of data clusters 40 Number of data points in trian data 9169 E
ach data point contains 5 features
Number of data clusters 40 Number of data points in test data 3930 Ea
ch data point contains 5 features
```

In [52]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for ou
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 [53]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) f
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 [54]:

```
# the above contains values in the form of list of lists (i.e. list of values of ea
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])
train_freq=[]
test_freq=[]
train_amp=[]
test_amp=[]
for i in range(0,40):
    train_freq.extend(train_frequencies[i])
    test_freq.extend(test_frequencies[i])
    train_amp.extend(train_amplitudes[i])
    test_amp.extend(test_amplitudes[i])
```

In [55]:

```
#stacking new features,frequencies and amplitudes horizontally using hstack.
train_brand_new_features=np.hstack((train_new_features,train_freq,train_amp))
test_brand_new_features=np.hstack((test_new_features,test_freq,test_amp))
```

In [56]:

```
# 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 [57]:

```
# 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 [58]:

```
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1','frequency1','frequency2','frequency3
df_train = pd.DataFrame(data=train_brand_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, 19)
```

In [59]:

```
# Preparing the data frame for our train data
df_test = pd.DataFrame(data=test_brand_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, 19)
```

In [60]:

```
df_test.head()
```

Out[60]:

	ft_5	ft_4	ft_3	ft_2	ft_1	frequency1	frequency2	frequency3	frequency4	frequ
0	156.0	162.0	142.0	117.0	139.0	0.006944	0.013889	0.012897	0.000992	0.03
1	162.0	142.0	117.0	139.0	150.0	0.006944	0.013889	0.012897	0.000992	0.03
2	142.0	117.0	139.0	150.0	168.0	0.006944	0.013889	0.012897	0.000992	0.03
3	117.0	139.0	150.0	168.0	161.0	0.006944	0.013889	0.012897	0.000992	0.03
4	139.0	150.0	168.0	161.0	131.0	0.006944	0.013889	0.012897	0.000992	0.03

In [61]:

```
df_train.head()
```

Out[61]:

	ft_5	ft_4	ft_3	ft_2	ft_1	frequency1	frequency2	frequency3	frequency4	frequency5	
0	0.0	0.0	0.0	0.0	0.0	0.006944	0.013889	0.012897	0.000992	0.034722	3
1	0.0	0.0	0.0	0.0	0.0	0.006944	0.013889	0.012897	0.000992	0.034722	3
2	0.0	0.0	0.0	0.0	0.0	0.006944	0.013889	0.012897	0.000992	0.034722	3
3	0.0	0.0	0.0	0.0	0.0	0.006944	0.013889	0.012897	0.000992	0.034722	3
4	0.0	0.0	0.0	0.0	0.0	0.006944	0.013889	0.012897	0.000992	0.034722	3

Using Linear Regression

Hyperparameter Tuning using GridSearch

In [62]:

```
#hyperparameter tuning using GridSearch
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV
model=SGDRegressor(loss='squared_loss',penalty='l2')
alpha=[10**-4,10**-3,10**-2,10**-1,1,10,100,1000]
param_grid={"alpha":alpha}
clf = GridSearchCV(model,param_grid, scoring = "neg_mean_absolute_error", cv=10)
clf.fit(df_train,tsne_train_output)
clf.best_params_
```

Out[62]:

```
{'alpha': 100}
```

In [63]:

```
#applying linear regression with alpha=100
model = SGDRegressor(loss = "squared_loss", penalty = "l2", alpha = 100)
model.fit(df_train, tsne_train_output)
pred_tr = model.predict(df_train)
pred_test = model.predict(df_test)

train_MAPE_lr = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_ou
train_MSE_lr = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_lr = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_out
test_MSE_lr = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for linear regression =',train_MAPE_lr)
print('Mean absolute percentage test error for linear regression =',test_MAPE_lr)
print('-'*50)
print('Mean Squared train error for linear regression =',train_MSE_lr)
print('Mean Squared test error for linear regression =',test_MSE_lr)
```

```
Mean absolute percentage train error for linear regression = 4.035042
2747360896e+17
Mean absolute percentage test error for linear regression = 3.6098592
73327205e+17
--------------------------------------------------
Mean Squared train error for linear regression = 6.697666787610942e+3
8
Mean Squared test error for linear regression = 6.698706777507761e+38
```

Using Random Forest Regressor

Hyperparameter Tuning using RandomSearch

In [64]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform

param_dist = {"n_estimators":[10,20,50,100,150,200,250,300],
              "max_depth": [2,5,10,20,50,100,200,250],
              "min_samples_split": sp_randint(120,190),
              "min_samples_leaf": sp_randint(25,65)
             }

clf = RandomForestRegressor(random_state=25,n_jobs=-1)

model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=3,scoring='neg_mean_absolute_error',

model.fit(df_train,tsne_train_output)
model.best_estimator_
```

Out[64]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=51, min_samples_split=135,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
           oob_score=False, random_state=25, verbose=0, warm_start=Fa
lse)
```

In [65]:

```
model=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=51, min_samples_split=135,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
           oob_score=False, random_state=25, verbose=0, warm_start=False)
model.fit(df_train,tsne_train_output)
pred_tr = model.predict(df_train)
pred_test = model.predict(df_test)

train_MAPE_rf = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_ou
train_MSE_rf = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_rf = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_out
test_MSE_rf = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for Random Forest Regressor =',train_MA
print('Mean absolute percentage test error for Random Forest Regressor =',test_MAPE
print('-'*50)
print('Mean Squared train error for Random Forest Regressor =',train_MSE_rf)
print('Mean Squared test error for Random Forest Regressor=',test_MSE_rf)
```

```
Mean absolute percentage train error for Random Forest Regressor = 0.
13400491278880125
Mean absolute percentage test error for Random Forest Regressor = 0.1
3014137467079873
--------------------------------------------------
Mean Squared train error for Random Forest Regressor = 188.1494947615
8353
Mean Squared test error for Random Forest Regressor= 197.190078419331
14
```

In [66]:

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

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'frequency1', 'frequen
cy2',
       'frequency3', 'frequency4', 'frequency5', 'Amplitude1', 'Ampli
tude2',
       'Amplitude3', 'Amplitude4', 'Amplitude5', 'lat', 'lon', 'weekd
ay',
       'exp_avg'],
      dtype='object')
[8.09201635e-04 5.63542183e-04 6.51438089e-04 5.08579588e-04
 1.05553525e-03 6.88924676e-05 1.07263969e-04 1.16531360e-04
 1.36025453e-04 1.15516168e-04 2.74605915e-04 1.53836478e-04
 1.11270449e-04 1.45127417e-04 1.61234842e-04 1.32279350e-04
 2.33479347e-04 1.90279714e-04 9.94465360e-01]
```

Using XgBoost Regressor

Hyperparameter Tuning using RandomSearch

In [76]:

```
import xgboost as xgb
clf = xgb.XGBRegressor()
param_dist = {"n_estimators":[10,20,50,100,150,200,250,300],
              "max_depth": [2,5,10,20,50,100,200,250]
              }
model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=3,scoring='neg_mean_absolute_error',
model.fit(df_train,tsne_train_output)
model.best_estimator_
```

```
[13:49:14] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[13:53:45] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[13:58:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:02:36] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:02:38] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:02:40] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:02:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:03:33] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:04:24] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:05:15] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:05:55] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:06:38] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:07:19] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:18:23] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:29:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[14:41:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
```

Out[76]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=5, min_child_weight=1, missing=None, n_estimators=20
0,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=
0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

In [77]:

```
model=xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=5, min_child_weight=1, missing=None, n_estimators=200,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
model.fit(df_train,tsne_train_output)
pred_tr = model.predict(df_train)

pred_test = model.predict(df_test)

train_MAPE_xgb = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_o
train_MSE_xgb = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_xgb = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_ou
test_MSE_xgb = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for XGB Regressor =',train_MAPE_xgb)
print('Mean absolute percentage test error for XGB Regressor =',test_MAPE_xgb)
print('-'*50)
print('Mean Squared train error for XGB Regressor =',train_MSE_xgb)
print('Mean Squared test error for XGB Regressor=',test_MSE_xgb)
```

```
[14:42:59] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
Mean absolute percentage train error for XGB Regressor = 0.1362328162
7876688
Mean absolute percentage test error for XGB Regressor = 0.13026013847
165532
--------------------------------------------------
Mean Squared train error for XGB Regressor = 175.7993202004048
Mean Squared test error for XGB Regressor= 196.17801635115413
```

In [78]:

```
#feature importances based on analysis using XGBOOST
print (df_train.columns)
print (model.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'frequency1', 'frequen
cy2',
       'frequency3', 'frequency4', 'frequency5', 'Amplitude1', 'Ampli
tude2',
       'Amplitude3', 'Amplitude4', 'Amplitude5', 'lat', 'lon', 'weekd
ay',
       'exp_avg'],
      dtype='object')
[1.1116869e-03 1.2815341e-03 2.9206248e-03 4.0263333e-03 5.9577376e-0
3
 7.3607726e-04 1.0995131e-03 9.2012819e-04 1.4610766e-03 1.1338480e-0
3
 1.7893878e-03 1.5352806e-03 1.3215462e-03 1.5450796e-03 1.6677270e-0
3
 8.7168580e-04 1.5809995e-03 1.3727670e-03 9.6766698e-01]
```

Calculating the error metric values for various models

In [103]:

```
from prettytable import PrettyTable
x = PrettyTable()
x = PrettyTable(["Models", "MAPE train %", "MAPE test %"])
x.add_row(['Linear Regression','403.5','360.9'])
x.add_row(['Random Forest','13.40','13.01'])
x.add_row(['XGBOOST','13.62','13.02'])
print(x)
```

```
+-------------------+--------------+-------------+
|       Models      | MAPE train % | MAPE test % |
+-------------------+--------------+-------------+
| Linear Regression |    403.5     |    360.9    |
|   Random Forest   |    13.40     |    13.01    |
|      XGBOOST      |    13.62     |    13.02    |
+-------------------+--------------+-------------+
```

Best MAPE test score is 13.01 for RandomForest. After adding new features - fourier features amplitude and frequencies we will try more feature engineering to reduce MAPE<12%

Triple Exponential Smoothing

Exponential smoothing is a time series forecasting method for univariate data.

This method is based on three smoothing equations stationary components, trend and seasonal.

alpha= data smoothing factor (0<alpha<1)

beta= Trend Smoothing factor (0<beta<1)

gama= seasonal change smoothing factor (0<gama<1)

In [79]:

```
#https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-i
#Triple Exponential Smoothing a.k.a Holt-Winters Method
#initial trend
#Exponential smoothing is a time series forecasting method for univariate data.
def initial_trend(series, slen):  #computing the average of trend averages across s
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen

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


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

In [80]:

```
alpha,beta,gama,slen = 0.2,.2,0.3,24
triple_exp_smoothing = []
for i in range(len(train_features)):
    triple_exp_smoothing.append(triple_exponential_smoothing(regions_cum[i][0:13104
```

In [81]:

```
tsne_train_triple_exp_smooth = [i[:9169] for i in triple_exp_smoothing]
tsne_test_triple_exp_smooth = [i[9169:] for i in triple_exp_smoothing]
```

In [82]:

```
#flattening
train_triple_exp_smooth=sum(tsne_train_triple_exp_smooth,[])
test_triple_exp_smooth=sum(tsne_test_triple_exp_smooth,[])
```

In [83]:

```
df_train['triple_exp_smooth']=train_triple_exp_smooth
df_test['triple_exp_smooth']=test_triple_exp_smooth[:157200]
```

In [86]:

```
#dropping columns of fourier frequencies and amplitudes
df_train=df_train.drop(['frequency1','frequency2','frequency3','frequency4','freque
df_train.head()
```

Out[86]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp_smooth
0	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	6.653589
1	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	5.679604
2	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	4.714477
3	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	1.762075
4	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	1.932747

In [87]:

```
#dropping columns of fourier frequencies and amplitudes
df_test=df_test.drop(['frequency1','frequency2','frequency3','frequency4','frequenc
df_test.head()
```

Out[87]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp_smoo
0	156.0	162.0	142.0	117.0	139.0	40.7699	-73.984164	4	134	133.3748
1	162.0	142.0	117.0	139.0	150.0	40.7699	-73.984164	4	145	144.3811
2	142.0	117.0	139.0	150.0	168.0	40.7699	-73.984164	4	161	138.5503
3	117.0	139.0	150.0	168.0	161.0	40.7699	-73.984164	4	161	127.3183
4	139.0	150.0	168.0	161.0	131.0	40.7699	-73.984164	4	140	141.1427

In [88]:

```
#Mean
df_train['mean'] = df_train[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].mean(axis=1)
df_test['mean'] = df_test[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].mean(axis=1)
```

In [89]:

```
#Variance
df_train['variance'] = df_train[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].var(axis=
df_test['variance'] = df_test[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].var(axis=1)
```

In [90]:

```
#skewness
df_train['skewness'] = df_train[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].skew(axis
df_test['skewness'] = df_test[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].skew(axis=1
```

In [91]:

```
#kurtosis
df_train['kurtosis'] = df_train[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].kurtosis(
df_test['kurtosis'] = df_test[['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1']].kurtosis(ax
```

In [92]:

```
df_train.head()
```

Out[92]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp_smooth	mea
0	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	6.653589	0
1	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	5.679604	0
2	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	4.714477	0
3	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	1.762075	0
4	0.0	0.0	0.0	0.0	0.0	40.7699	-73.984164	4	0	1.932747	0

In [93]:

```
df_test.head()
```

Out[93]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp_smoo
0	156.0	162.0	142.0	117.0	139.0	40.7699	-73.984164	4	134	133.3748
1	162.0	142.0	117.0	139.0	150.0	40.7699	-73.984164	4	145	144.3811
2	142.0	117.0	139.0	150.0	168.0	40.7699	-73.984164	4	161	138.5503
3	117.0	139.0	150.0	168.0	161.0	40.7699	-73.984164	4	161	127.3183
4	139.0	150.0	168.0	161.0	131.0	40.7699	-73.984164	4	140	141.1427

Linear Regression

Hyperparameter using GridSearch

In [97]:

```
#hyperparameter tuning using GridSearch
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV
model=SGDRegressor(loss='squared_loss')
alpha=[10**-4,10**-3,10**-2,10**-1,1,10,100,1000]
param_grid={"alpha":alpha}
clf = GridSearchCV(model,param_grid, scoring = "neg_mean_absolute_error", cv=10)
clf.fit(df_train,tsne_train_output)
clf.best_params_
```

Out[97]:

```
{'alpha': 1000}
```

In [98]:

```
#applying linear regression with alpha=1000
model = SGDRegressor(loss = "squared_loss", alpha =1000)
model.fit(df_train, tsne_train_output)
pred_tr = model.predict(df_train)
pred_test = model.predict(df_test)

train_MAPE_lr = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_ou
train_MSE_lr = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_lr = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_out
test_MSE_lr = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for linear regression =',train_MAPE_lr)
print('Mean absolute percentage test error for linear regression =',test_MAPE_lr)
print('-'*50)
print('Mean Squared train error for linear regression =',train_MSE_lr)
print('Mean Squared test error for linear regression =',test_MSE_lr)
```

```
Mean absolute percentage train error for linear regression = 43923778
597.45205
Mean absolute percentage test error for linear regression = 371030060
19.94596
--------------------------------------------------
Mean Squared train error for linear regression = 1.6558289484130186e+
26
Mean Squared test error for linear regression = 1.1702102020216539e+2
6
```

RandomForest

Hyperparamter tuning using RandomSearch

In [99]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform

param_dist = {"n_estimators":[10,20,50,100,150,200,250,300],
              "max_depth": [2,5,10,20,50,100,200,250],
              "min_samples_split": sp_randint(120,190),
              "min_samples_leaf": sp_randint(25,65)
             }

clf = RandomForestRegressor(random_state=25,n_jobs=-1)

model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=3,scoring='neg_mean_absolute_error',

model.fit(df_train,tsne_train_output)
model.best_estimator_
```

Out[99]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=250,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=28, min_samples_split=175,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
           oob_score=False, random_state=25, verbose=0, warm_start=Fa
lse)
```

In [100]:

```
model=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=250,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=28, min_samples_split=175,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
           oob_score=False, random_state=25, verbose=0, warm_start=False)
model.fit(df_train,tsne_train_output)
pred_tr = model.predict(df_train)
pred_test = model.predict(df_test)

train_MAPE_rf = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_ou
train_MSE_rf = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_rf = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_out
test_MSE_rf = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for Random Forest Regressor =',train_MA
print('Mean absolute percentage test error for Random Forest Regressor =',test_MAPE
print('-'*50)
print('Mean Squared train error for Random Forest Regressor =',train_MSE_rf)
print('Mean Squared test error for Random Forest Regressor=',test_MSE_rf)
```

```
Mean absolute percentage train error for Random Forest Regressor = 0.
09777779825107251
Mean absolute percentage test error for Random Forest Regressor = 0.0
9761746607190208
--------------------------------------------------
Mean Squared train error for Random Forest Regressor = 87.57764281269
21
Mean Squared test error for Random Forest Regressor= 102.002166781637
63
```

In [101]:

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

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekda
y',
       'exp_avg', 'triple_exp_smooth', 'mean', 'variance', 'skewnes
s',
       'kurtosis'],
      dtype='object')
[2.56451642e-04 3.07633571e-04 3.77933121e-04 3.09917687e-04
 1.53599065e-03 3.18927202e-04 3.09921143e-04 5.25734985e-05
 8.95013287e-03 9.85768456e-01 1.21837933e-03 3.80049776e-04
 1.08921082e-04 1.04712358e-04]
```

XGBOOST

Hyperparameter tuning using RandomSearch

In [102]:

```
import xgboost as xgb
clf = xgb.XGBRegressor()
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15)
              }
model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=3,scoring='neg_mean_absolute_error',
model.fit(df_train,tsne_train_output)
model.best_estimator_
```

```
[15:06:33] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:08:12] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:09:56] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:11:44] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:13:32] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:15:21] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:17:04] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:18:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:19:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:20:05] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:21:21] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:22:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:23:59] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:25:12] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:26:22] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
[15:27:31] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
```

Out[102]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=10, min_child_weight=1, missing=None, n_estimators=1
09,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=
0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

In [104]:

```
model=xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=10, min_child_weight=1, missing=None, n_estimators=109,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
model.fit(df_train,tsne_train_output)
pred_tr = model.predict(df_train)
pred_test = model.predict(df_test)

train_MAPE_xgb = mean_absolute_error(tsne_train_output, pred_tr)/ (sum(tsne_train_o
train_MSE_xgb = mean_squared_error(tsne_train_output, pred_tr)

test_MAPE_xgb = mean_absolute_error(tsne_test_output, pred_test)/ (sum(tsne_test_ou
test_MSE_xgb = mean_squared_error(tsne_test_output, pred_test)

print('Mean absolute percentage train error for XGB Regressor =',train_MAPE_xgb)
print('Mean absolute percentage test error for XGB Regressor =',test_MAPE_xgb)
print('-'*50)
print('Mean Squared train error for XGB Regressor =',train_MSE_xgb)
print('Mean Squared test error for XGB Regressor=',test_MSE_xgb)
```

```
[15:29:54] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/s
rc/objective/regression_obj.cu:152: reg:linear is now deprecated in f
avor of reg:squarederror.
Mean absolute percentage train error for XGB Regressor = 0.0882084791
4286391
Mean absolute percentage test error for XGB Regressor = 0.09486347584
770169
--------------------------------------------------
Mean Squared train error for XGB Regressor = 62.45763901711435
Mean Squared test error for XGB Regressor= 93.02083027462237
```

In [105]:

```
#feature importances based on analysis using XGBOOST
print (df_train.columns)
print (model.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekda
y',
       'exp_avg', 'triple_exp_smooth', 'mean', 'variance', 'skewnes
s',
       'kurtosis'],
      dtype='object')
[5.2623980e-04 7.6007395e-04 1.0500749e-03 1.1454030e-03 2.7519972e-0
3
 1.1958419e-03 1.3165005e-03 7.6908473e-04 2.2495540e-02 9.6312594e-0
1
 2.4609298e-03 1.1126547e-03 6.6581374e-04 6.2392623e-04]
```

In [106]:

```
from prettytable import PrettyTable
x = PrettyTable()
x = PrettyTable(["Models", "MAPE train %", "MAPE test %"])
x.add_row(['Linear Regression','4392377859745.2','3710300601994.5'])
x.add_row(['Random Forest','9.77','9.76'])
x.add_row(['XGBOOST','8.82','9.48'])
print(x)
```

```
+-------------------+-----------------+-----------------+
|       Models      |   MAPE train %  |   MAPE test %   |
+-------------------+-----------------+-----------------+
| Linear Regression | 4392377859745.2 | 3710300601994.5 |
|   Random Forest   |       9.77      |       9.76      |
|      XGBOOST      |       8.82      |       9.48      |
+-------------------+-----------------+-----------------+
```

Please notice that - after adding triple exponential smoothing and some features the best mean absolute percentage error is reduced to 9.48 for XGBOOST Regressor.

Observations

1. Adding 5 fourier features of amplitude and freqencies, we got MAPE = 13.01.(using Randomforest Regressor model).

2. Using triple exponential smoothing, we reduced the MAPE to even frther to 9.48, using XGBoost.