Taxi demand prediction in New York City

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
!pip install gpxpy
```

```
Requirement already satisfied: gpxpy in /usr/local/lib/python3.6/dist-packages (1.4.0
```

```
#Importing Libraries
# pip3 install graphviz
#pip3 install dask
#pip3 install toolz
#pip3 install cloudpickle
# https://www.youtube.com/watch?v=ieW3G7ZzRZ0
# https://github.com/dask/dask-tutorial
# please do go through this python notebook: https://github.com/dask/dask-tutorial/blob/ma
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 user int
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) pairs
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\\mingw64\\bi
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) Th
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
number of medallions issued by the TLC. You access this mode of transportation by standing in th
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 F
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 bor
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

Data Collection

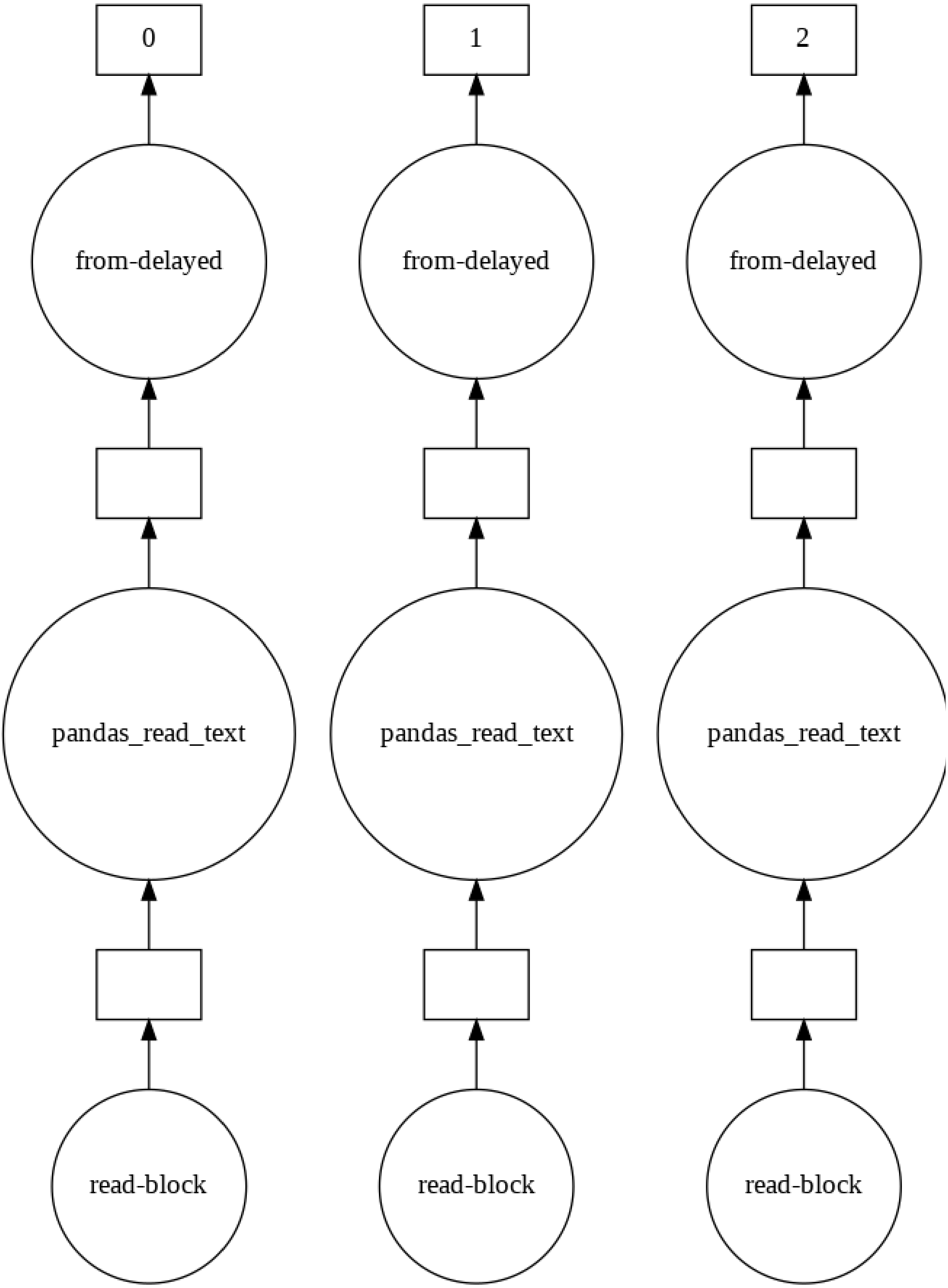
We Have collected all yellow taxi trips data from jan-2015 to dec-2016(Will be using only 2015 dat

file name	file name size	number of records	number of f
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

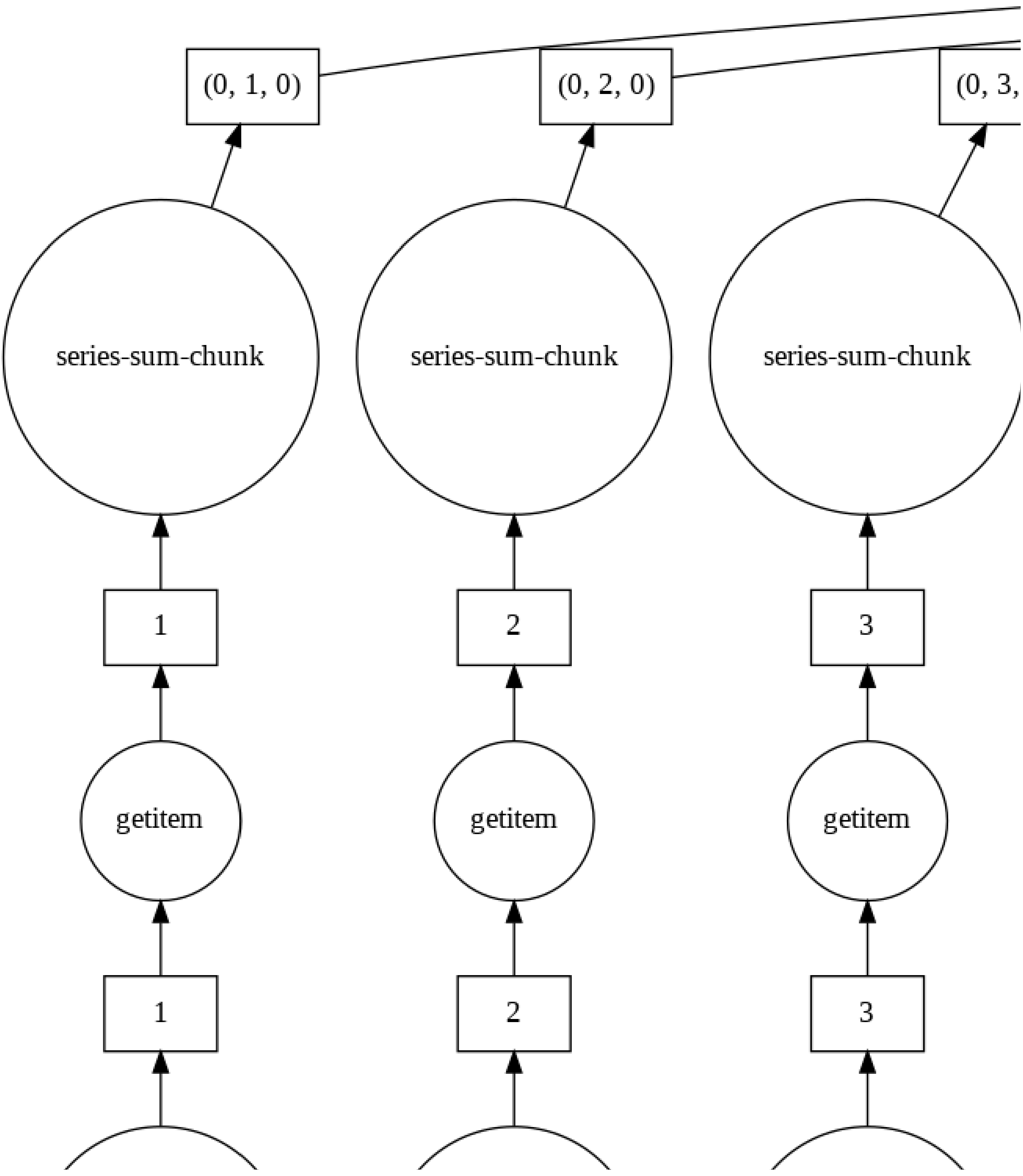
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2015-01.csv')
print(month.columns)
```

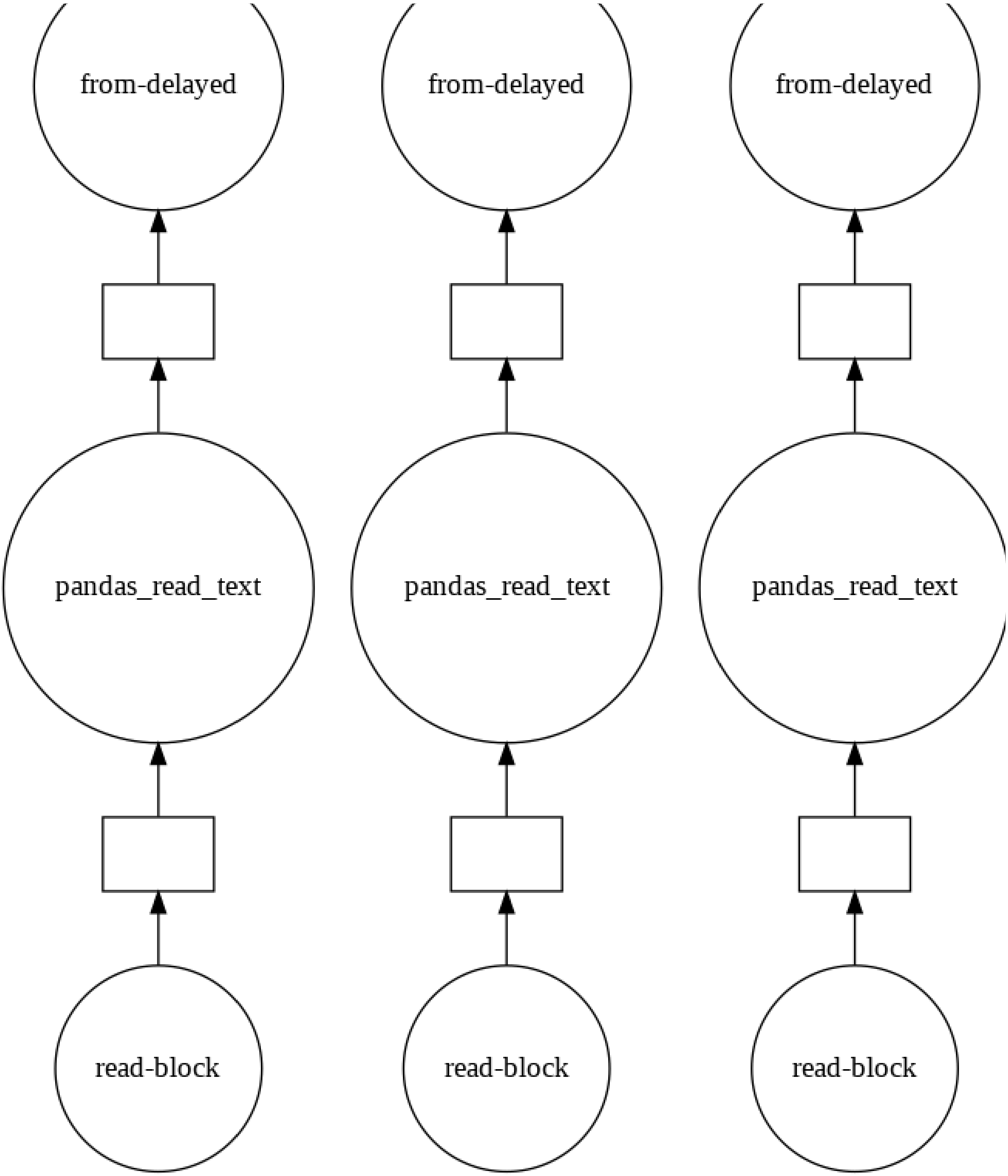
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax',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

```
# However unlike Pandas, operations on dask.dataframes don't trigger immediate computation
# instead they add key-value pairs to an underlying Dask graph. Recall that in the diagram
# 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 in the d
month.visualize()
```




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


(0, 1, 0)
(0, 2, 0)
(0, 3,
series-sum-chunk
series-sum-chunk
series-sum-chunk
1
2
3
getitem
getitem
getitem
1
2
3



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 h
    </tr>
    <tr>
        <td>MTA_tax</td>
        <td>0.50 MTA tax that is automatically triggered based on the metered rate in use.</td>
    </tr>
    <tr>
        <td>Improvement_surcharge</td>
        <td>0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge begar
    </tr>
    <tr>
        <td>Tip_amount</td>
        <td>Tip amount – This field is automatically populated for credit card tips.Cash tips are not i
    </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	
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

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 su

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

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 r

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

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

1. Pickup Latitude and Pickup Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bound (40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are nc with pickups which originate within New York.

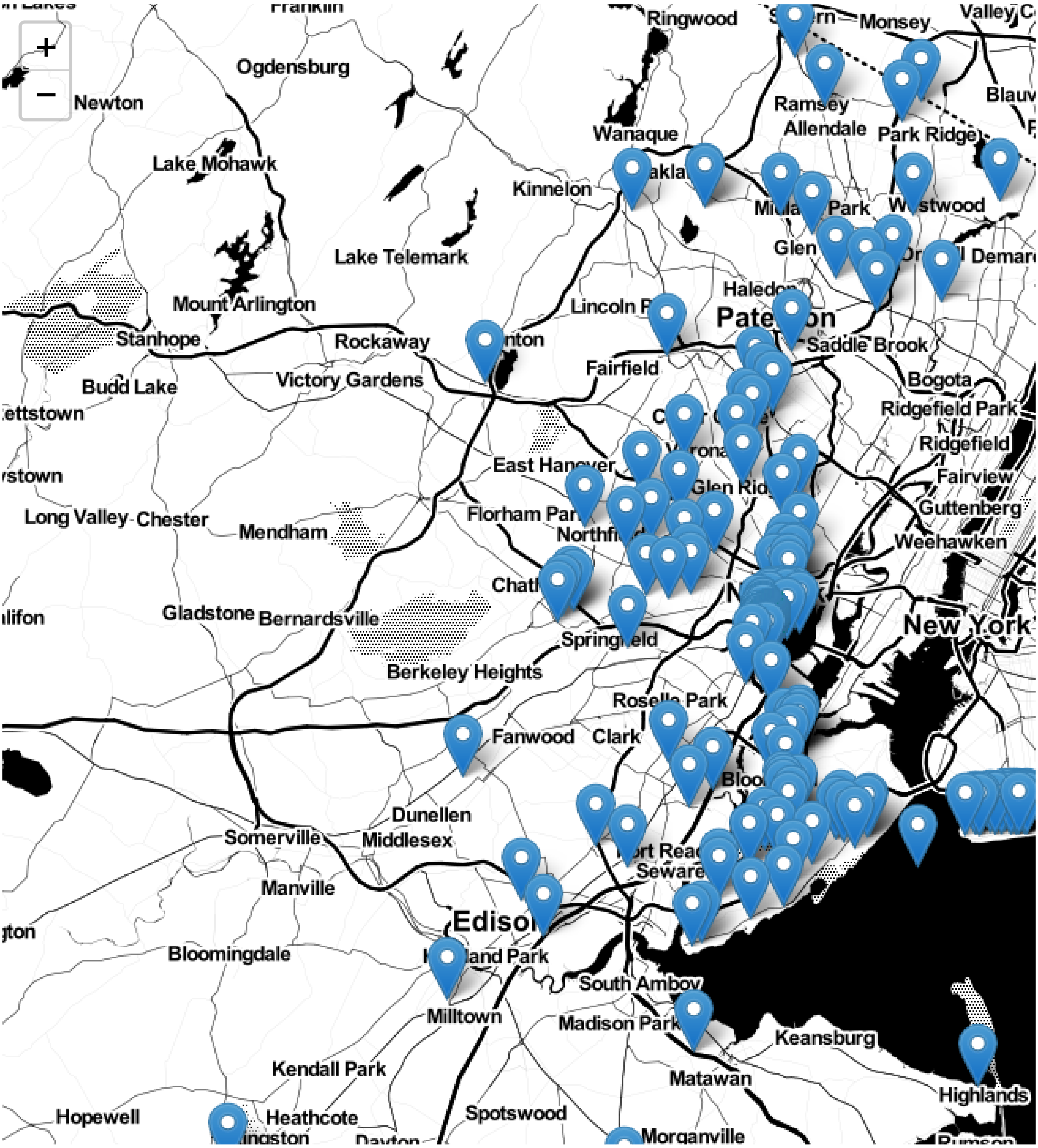
```
# Plotting pickup cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locat
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_latitude <=
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitude >= 40.917

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

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

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

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

Observation:- As you can see above that there are some points just outside the boundary but there
Mexico or Canada

2. Dropoff Latitude & Dropoff Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bound
(40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are nc
with dropoffs which are within New York.

```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locat
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitude <
                    (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.9

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

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

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

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



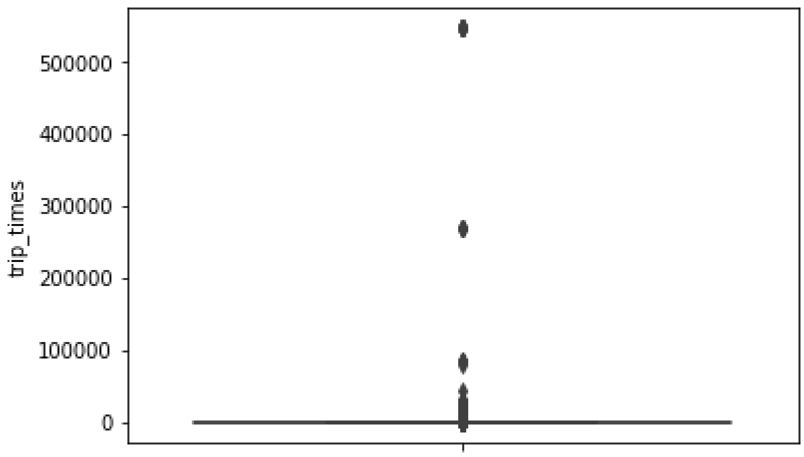
Observation:- The observations here are similar to those obtained while analysing pickup latitude

3. Trip Durations:

According to NYC Taxi & Limousine Commision Regulations **the maximum allowed trip duration ir**

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

# in out data we have time in the formate "YYYY-MM-DD HH:MM:SS" we convert thiss sting to
# https://stackoverflow.com/a/27914405
```

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



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

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

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

    return new_frame

# print(frame_with_durations.head())
#  passenger_count  trip_distance pickup_longitude  pickup_latitude dropoff_longitude drop
#   1                  1.59       -73.993896         40.750111     -73.974785        40.7
#   1                 3.30      -74.001648        40.724243     -73.994415        40.75910
#   1                 1.80      -73.963341        40.802788       -73.951820         40.824
#   1                 0.50      -74.009087        40.713818     -74.004326        40.71998
#   1                 3.00      -73.971176        40.762428     -74.004181        40.74265
frame_with_durations = return_with_trip_times(month)


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




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

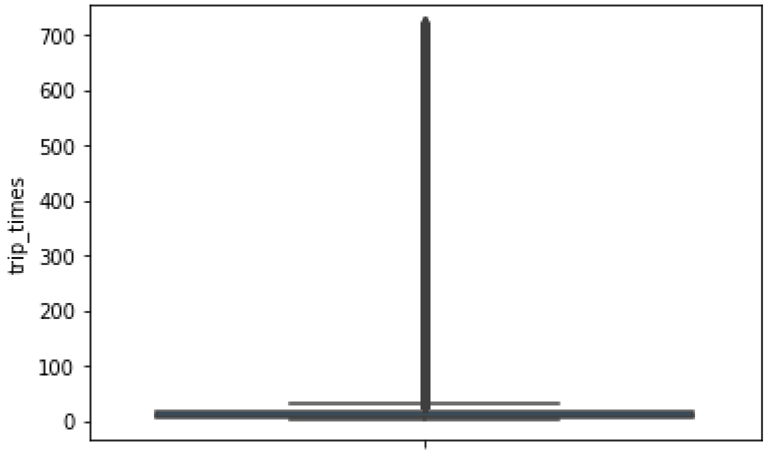
```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333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

```
#looking further from the 99th percecntile
for i in range(0,10):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

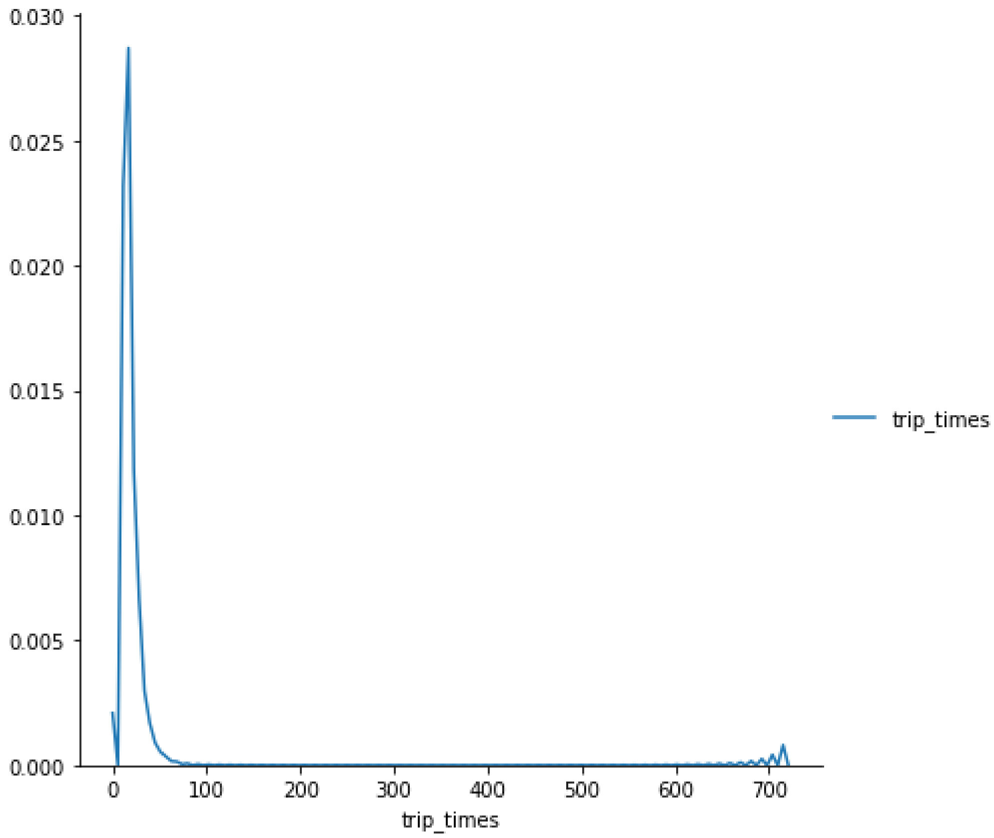
```
0 percentile value is -1211.0166666666667
1 percentile value is 1.2166666666666666
2 percentile value is 1.8833333333333333
3 percentile value is 2.2666666666666666
4 percentile value is 2.5833333333333335
5 percentile value is 2.8333333333333335
6 percentile value is 3.066666666666667
7 percentile value is 3.2666666666666666
8 percentile value is 3.466666666666667
9 percentile value is 3.65
100 percentile value is  548555.6333333333
```

```
#removing data based on our analysis and TLC regulations
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_times>1) & (
```

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



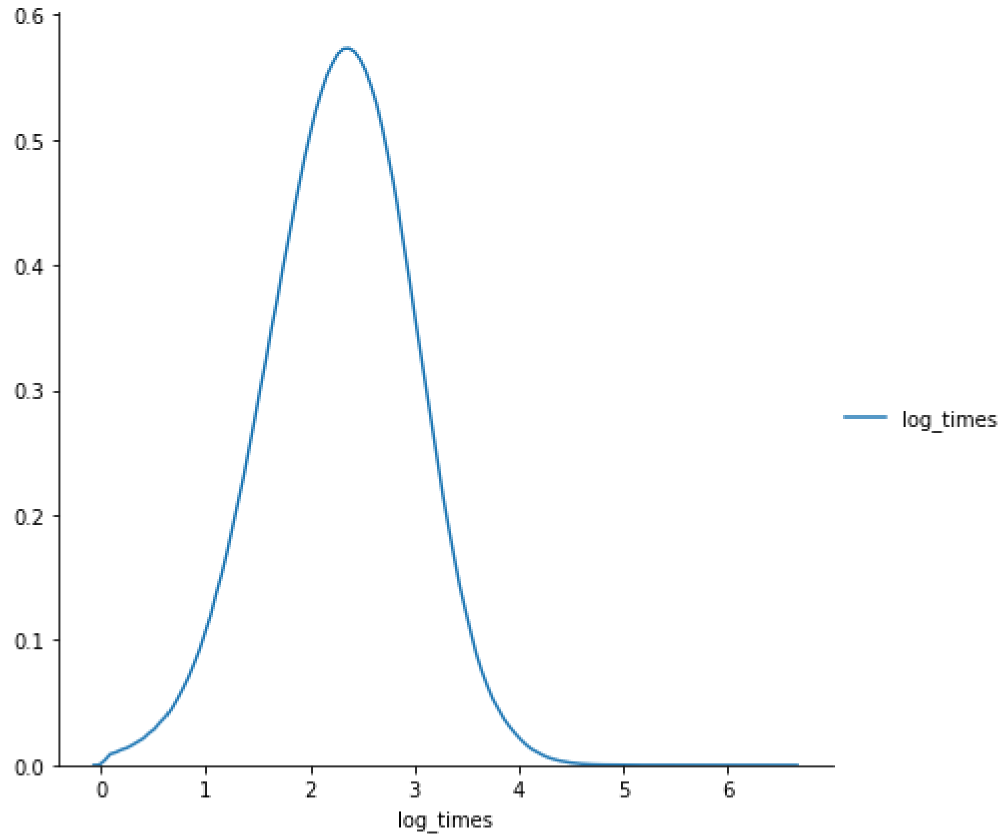

```
#pdf of trip-times after removing the outliers
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```



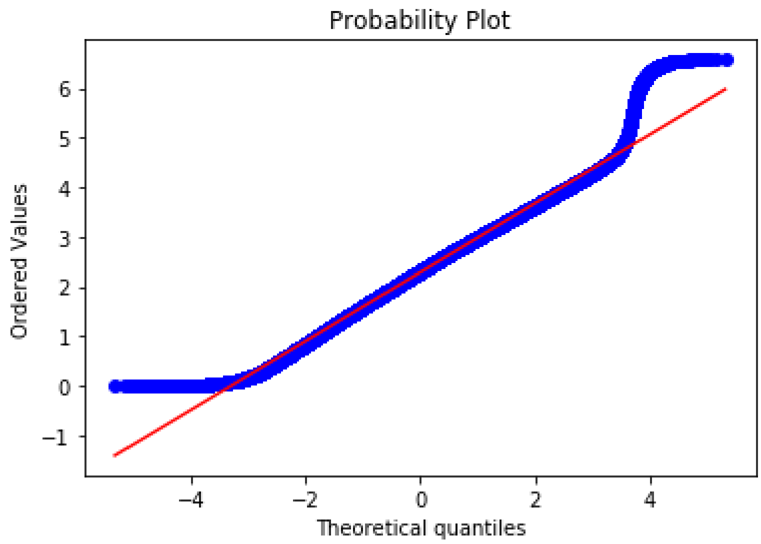

```
#converting the values to log-values to chec for log-normal
import math
frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_durations_modi
```

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

```
        .add_legend();
plt.show();
```



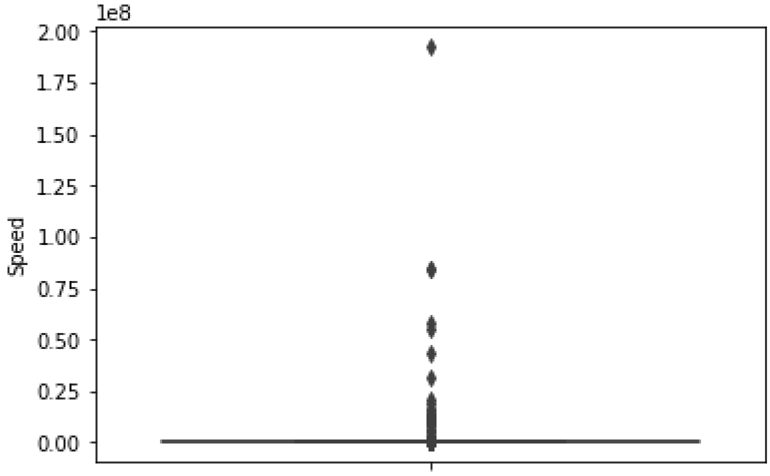

```
#Q-Q plot for checking if trip-times is log-normal
from scipy import stats
stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

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

```
plt.show()
```




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

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

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

```
90 percentile value is 20.186915887850468
91 percentile value is 20.91645569620253
92 percentile value is 21.752988047808763
93 percentile value is 22.721893491124263
94 percentile value is 23.844155844155843
95 percentile value is 25.182552504038775
96 percentile value is 26.80851063829787
97 percentile value is 28.84304932735426
98 percentile value is 31.591128254580514
99 percentile value is 35.7513566847558
100 percentile value is  192857142.85714284
```

```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,9
for i in np.arange(0.0, 1.0, 0.1):
```

```
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

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

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

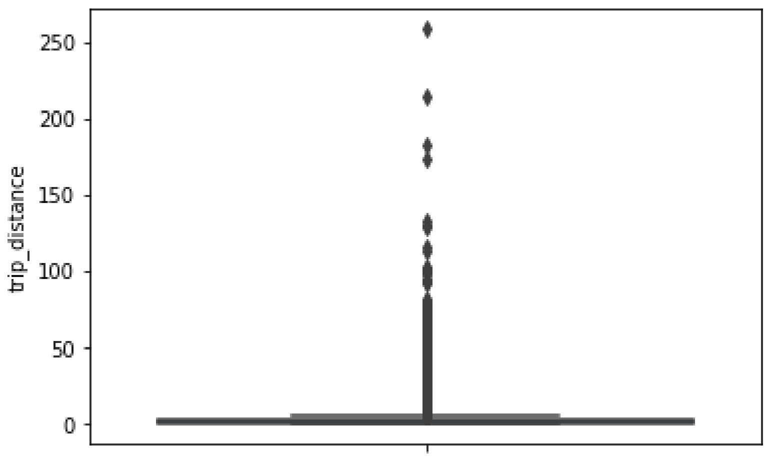
```
#avg.speed of cabs in New-York
sum(frame_with_durations_modified["Speed"]) / float(len(frame_with_durations_modified["Spe
```

```
12.450173996027528
```

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

4. Trip Distance

```
# 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()
```




```
#calculating trip distance values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
```

```
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

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

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

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

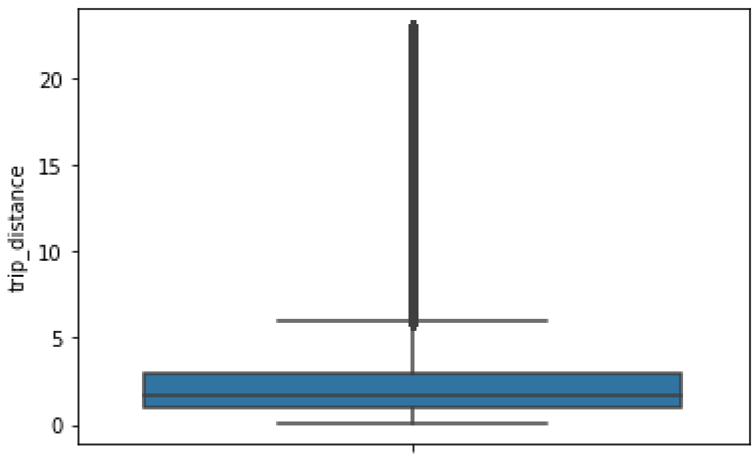
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

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

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

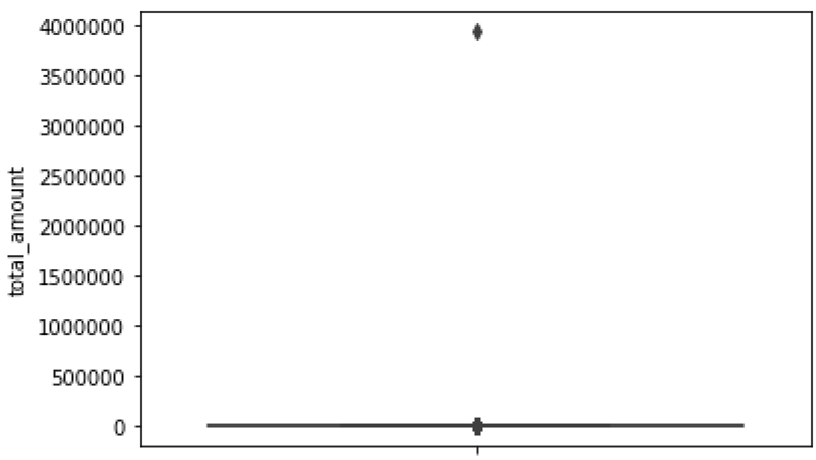
```
#box-plot after removal of outliers
```

```
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

```
# up to now we have removed the outliers based on trip durations, cab speeds, and trip dis
# 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()
```




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

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

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

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

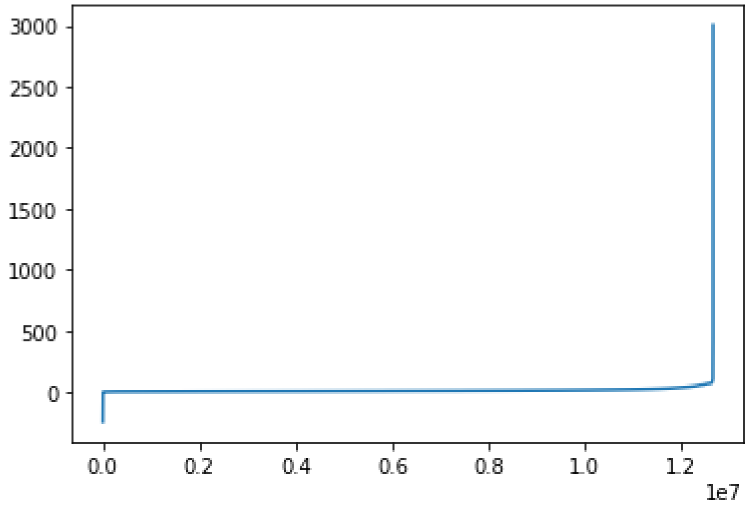
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

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

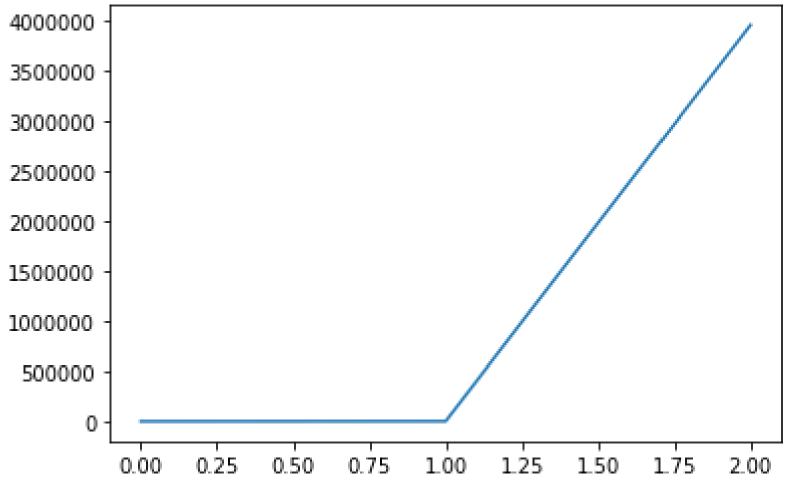
Observation:- As even the 99.9th percentile value doesnt look like an outlier,as there is not much c
99.9th percentile, we move on to do graphical analyis

```
#below plot shows us the fare values(sorted) to find a sharp increase to remove those valu
# plot the fare amount excluding last two values in sorted data
```

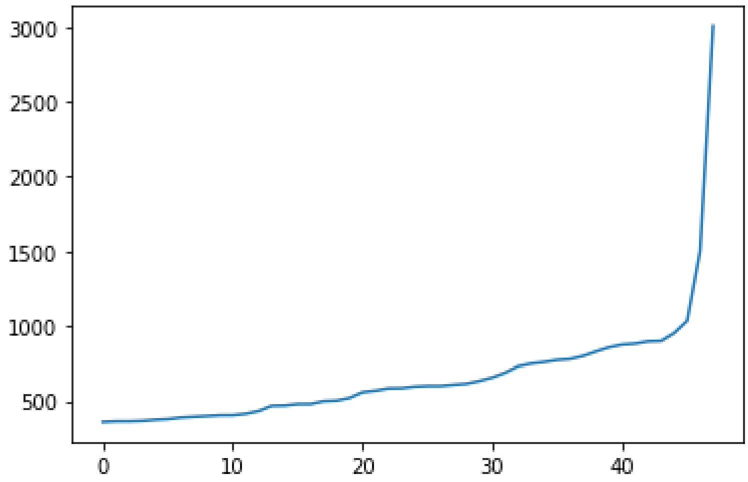
```
plt.plot(var[:-2])
plt.show()
```




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




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



Remove all outliers/erronous points.

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


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


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


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

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

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


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_lo
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitu
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitud
```

```
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitu

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

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


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

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

Data-preperation

Clustering/Segmentation

```
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']].va
neighbours=[]

def find_min_distance(cluster_centers, cluster_len):
    nice_points = 0
    wrong_points = 0
    less2 = []
    more2 = []
    min_dist=1000
    for i in range(0, cluster_len):
        nice_points = 0
        wrong_points = 0
        for j in range(0, cluster_len):
            if j!=i:
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0], cluster_cen
                min_dist = min(min_dist,distance/(1.60934*1000))
                if (distance/(1.60934*1000)) <= 2:
                    nice_points +=1
                else:
                    wrong_points += 1
```

```
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters within

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fit(c
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_du
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

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

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 8.0
Min inter-cluster distance =  1.0945442325142662
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 16.0
Min inter-cluster distance =  0.7131298007388065
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 22.0
Min inter-cluster distance =  0.5185088176172186
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 32.0
Min inter-cluster distance =  0.5069768450365043
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 38.0
Min inter-cluster distance =  0.36536302598358383
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 46.0
Min inter-cluster distance =  0.34704283494173577
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 54.0
Min inter-cluster distance =  0.30502203163245994
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 62.0
Min inter-cluster distance =  0.292203245317388
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 69.0
Min inter-cluster distance =  0.18257992857033273
---
```

Inference:

- The main objective was to find a optimal min. distance(Which roughly estimates to the radiu got was 40

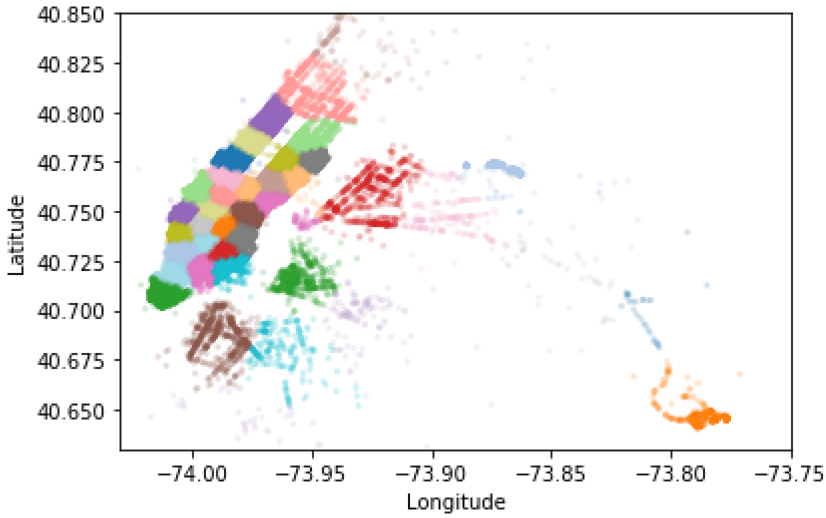
```
# if check for the 50 clusters you can observe that there are two clusters with only 0.3 m
# so we choose 40 clusters for solve the further problem

# 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_durati
```

Plotting the cluster centers:

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

Plotting the clusters:

```
#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[:10000
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.2)
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

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

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

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

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


# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durati
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']].groupb


# 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()
```

ip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	t
1.59	-73.993896	40.750111	-73.974785	40.750618	
3.30	-74.001648	40.724243	-73.994415	40.759109	
1.80	-73.963341	40.802788	-73.951820	40.824413	
0.50	-74.009087	40.713818	-74.004326	40.719986	
3.00	-73.971176	40.762428	-74.004181	40.742653	

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

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

```
# 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_with_du
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(frame_w

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_no,y
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip_distan

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-
month_feb_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-
month_mar_2016 = dd.read_csv('/content/drive/My Drive/Data_Notebooks/yellow_tripdata_2016-

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

```
Return with trip times..
Remove outliers..
Number of pickup records =  10906858
Number of outlier coordinates lying outside NY boundaries: 214677
Number of outliers from trip times analysis: 27190
Number of outliers from trip distance analysis: 79742
Number of outliers from speed analysis: 21047
Number of outliers from fare analysis: 4991
Total outliers removed 297784
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  11382049
Number of outlier coordinates lying outside NY boundaries: 223161
Number of outliers from trip times analysis: 27670
Number of outliers from trip distance analysis: 81902
Number of outliers from speed analysis: 22437
Number of outliers from fare analysis: 5476
Total outliers removed 308177
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  12210952
Number of outlier coordinates lying outside NY boundaries: 232444
Number of outliers from trip times analysis: 30868
Number of outliers from trip distance analysis: 87318
Number of outliers from speed analysis: 23889
Number of outliers from fare analysis: 5859
Total outliers removed 324635
---
Estimating clusters..
Final groupbying..
```

Smoothing

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

# for each cluster region we will collect all the indices of 10min intravels in which the
# 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


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

#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)


# for each cluster number of 10min intravels with 0 pickups
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",4464 - le
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1985
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  354
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  153
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  117
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  58
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  1190
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1357
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  53
------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  163
------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  35
------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  214
------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1180
```

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

there are two ways to fill up these values

- Fill the missing value with 0's
- Fill the missing values with the avg values
    - Case 1:(values missing at the start)
      Ex1: _ _ _ x =>ceil(x/4), ceil(x/4), ceil(x/4), ceil(x/4)
      Ex2: _ _ x => ceil(x/3), ceil(x/3), ceil(x/3)
    - Case 2:(values missing in middle)
      Ex1: x _ _ y => ceil((x+y)/4), ceil((x+y)/4), ceil((x+y)/4), ceil((x+y)/4)
      Ex2: x _ _ _ y => ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5), ceil((x+y)/5)
    - Case 3:(values missing at the end)
      Ex1: x _ _ _ => ceil(x/4), ceil(x/4), ceil(x/4), ceil(x/4)
      Ex2: x _ => ceil(x/2), ceil(x/2)

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 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


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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed
# 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/resolv
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.0/((right
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missi
```

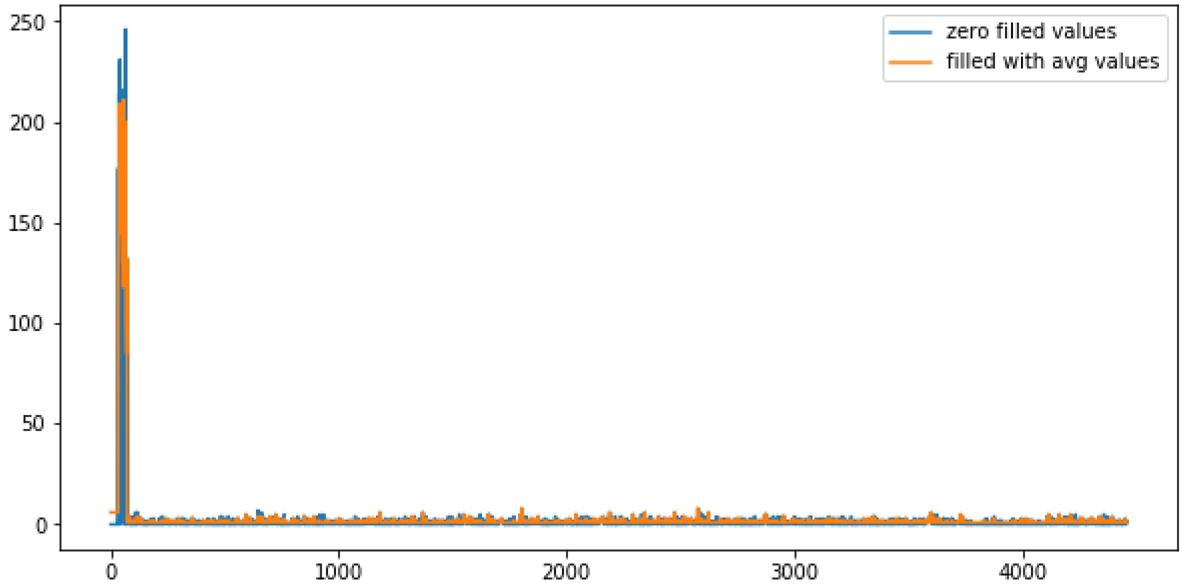
```
                right_hand_limit=0
                for j in range(i,4464):
                    if  j not in values[r]:
                        continue
                    else:
                        right_hand_limit=j
                        break
                smoothed_value=count_values[ind]*1.0/((right_hand_limit-i)+1)*1.0
                for j in range(i,right_hand_limit+1):
                        smoothed_bins.append(math.ceil(smoothed_value))
                repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

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

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

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

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

```
# 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()
```

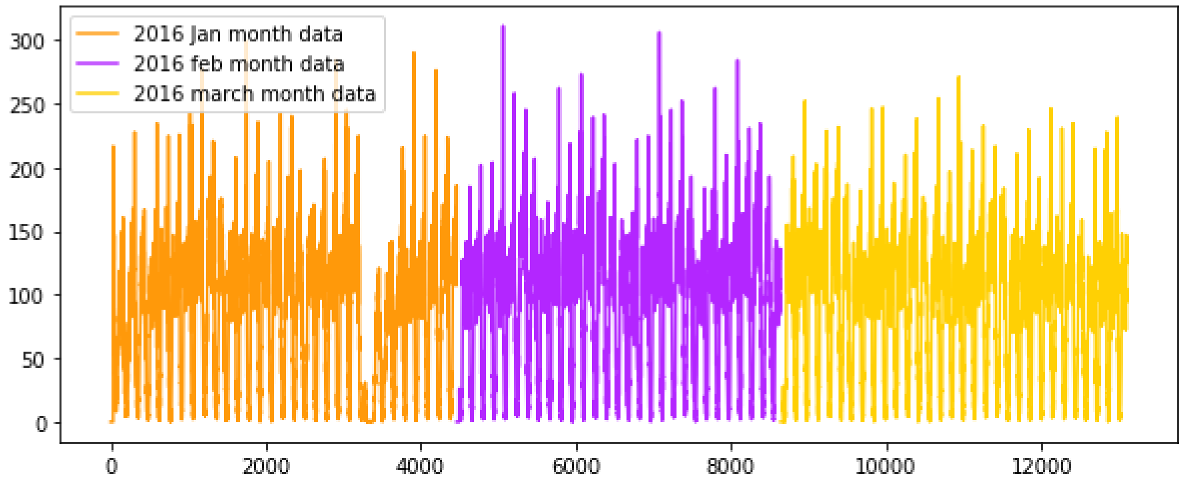


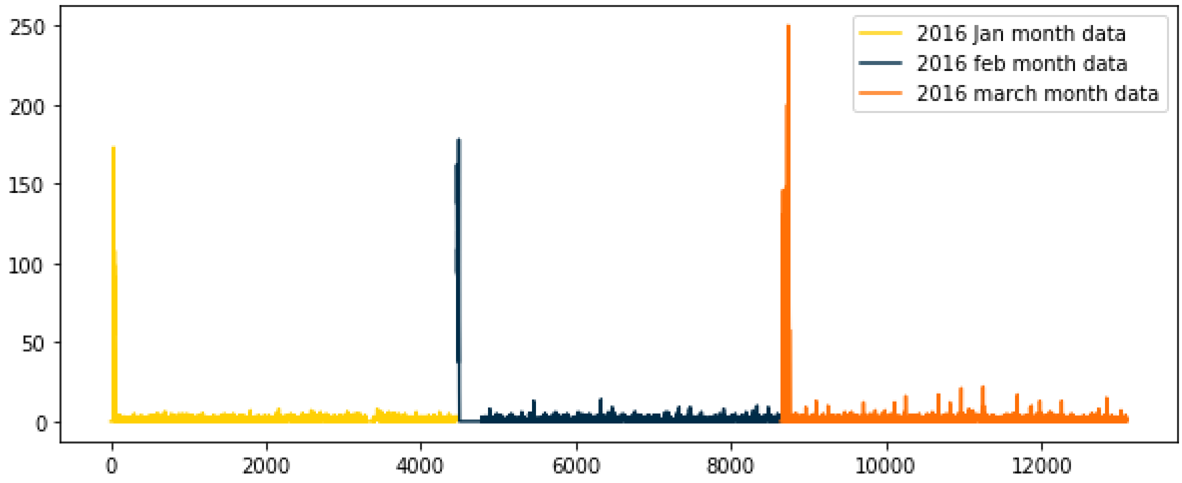

```
# why we choose, these methods and which method is used for which data?

# Ans: consider we have data of some month in 2015 jan 1st, 10 _ _ _ 20, i.e there are 10
# 10st 10min intravel, 0 pickups happened in 2nd 10mins intravel, 0 pickups happened in 3r
# 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 check the
# that are happened in the first 40min are same in both cases, but if you can observe that
# wheen you are using smoothing we are looking at the future number of pickups which might

# 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.


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

# Making list of all the values of pickup data in every bin for a period of 3 months and s
regions_cum = []

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

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

for i in range(0,40):
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_smooth[4176*i:4176*(i+1
```

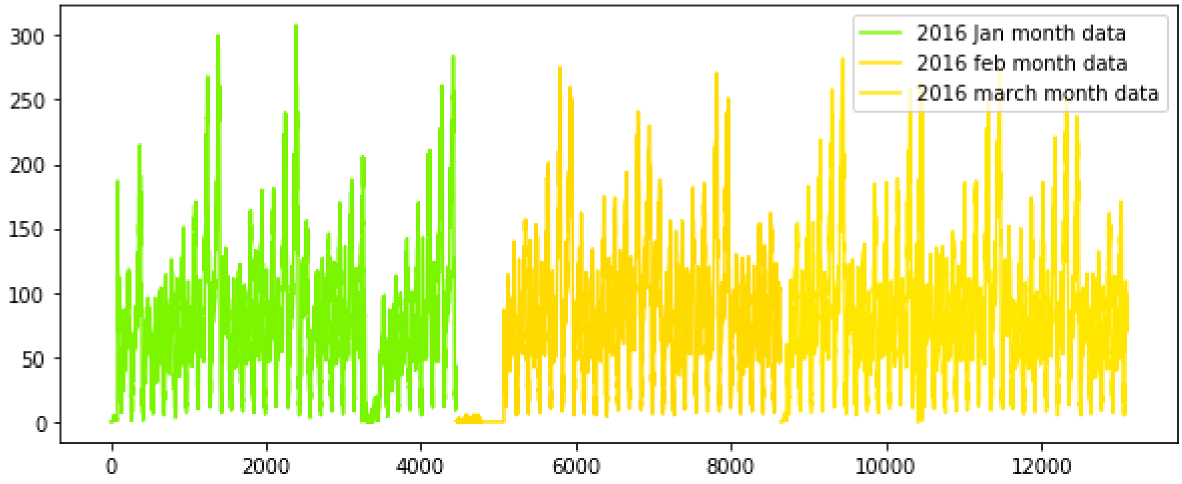
```
# print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 13104
```

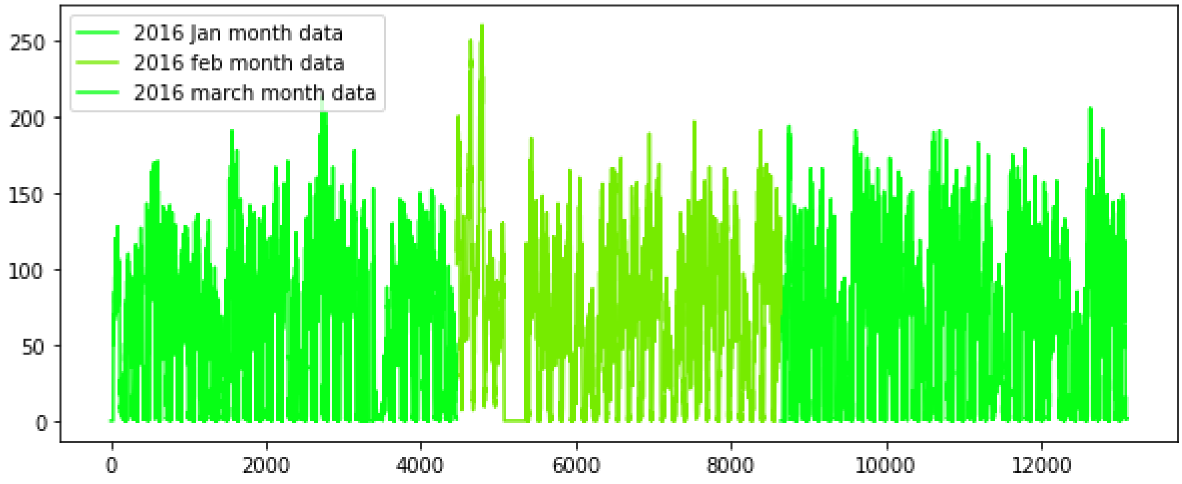
Time series and Fourier Transforms

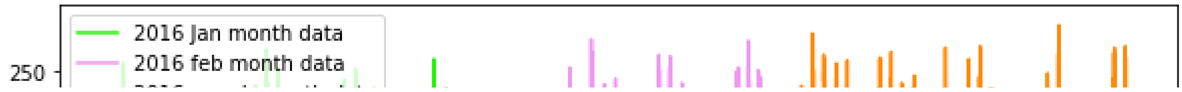
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40):
    plt.figure(figsize=(10,4))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

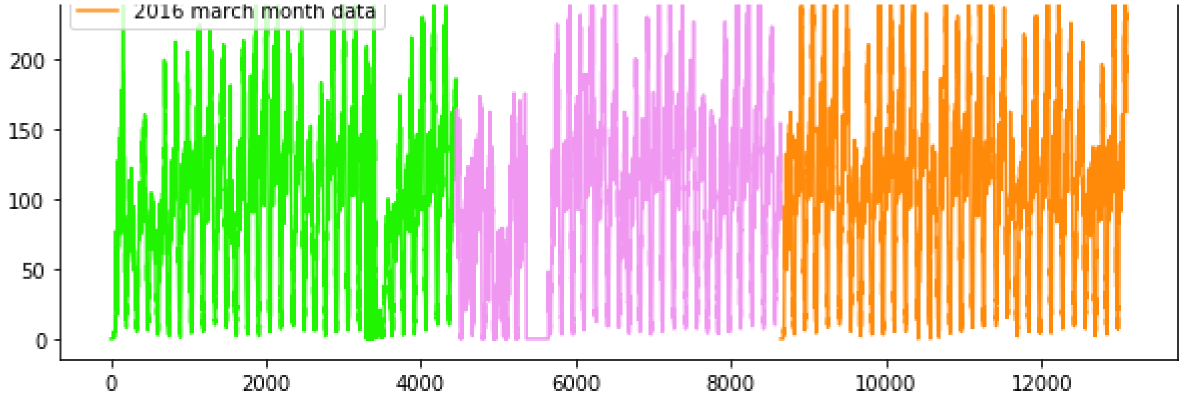

2016 Jan month data
2016 feb month data
2016 march month data

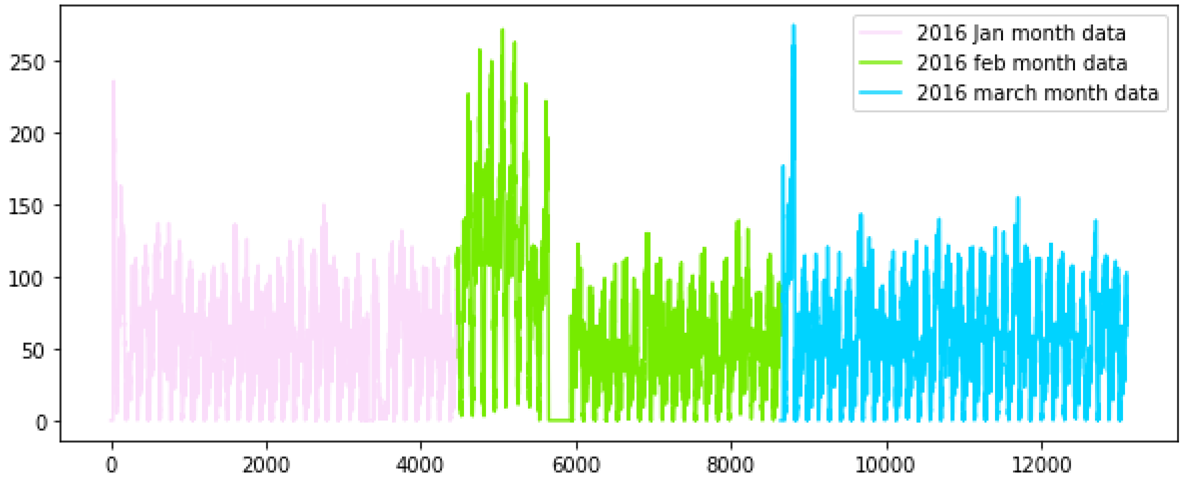

2016 Jan month data
2016 feb month data
2016 march month data

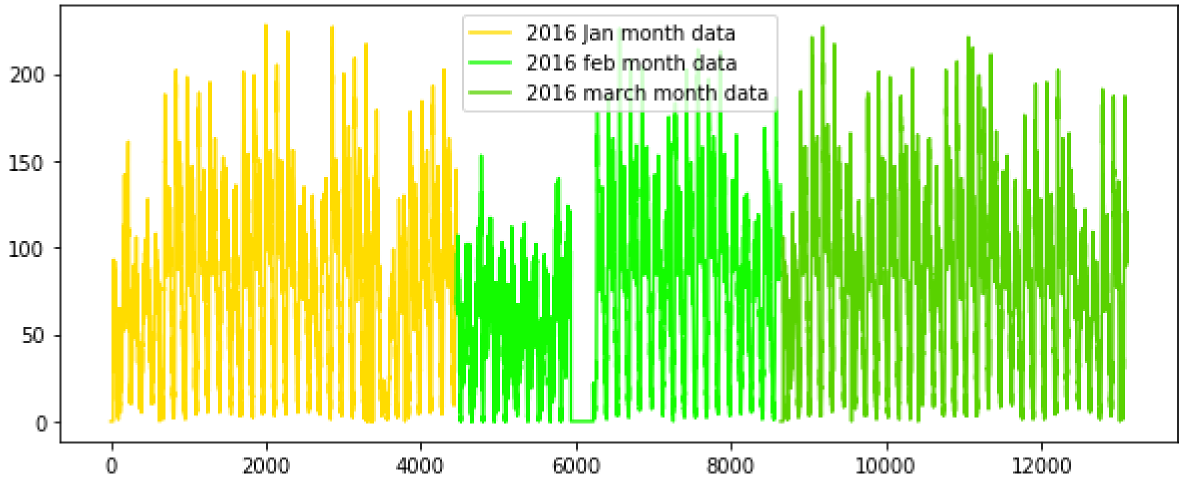

2016 Jan month data
2016 feb month data
2016 march month data

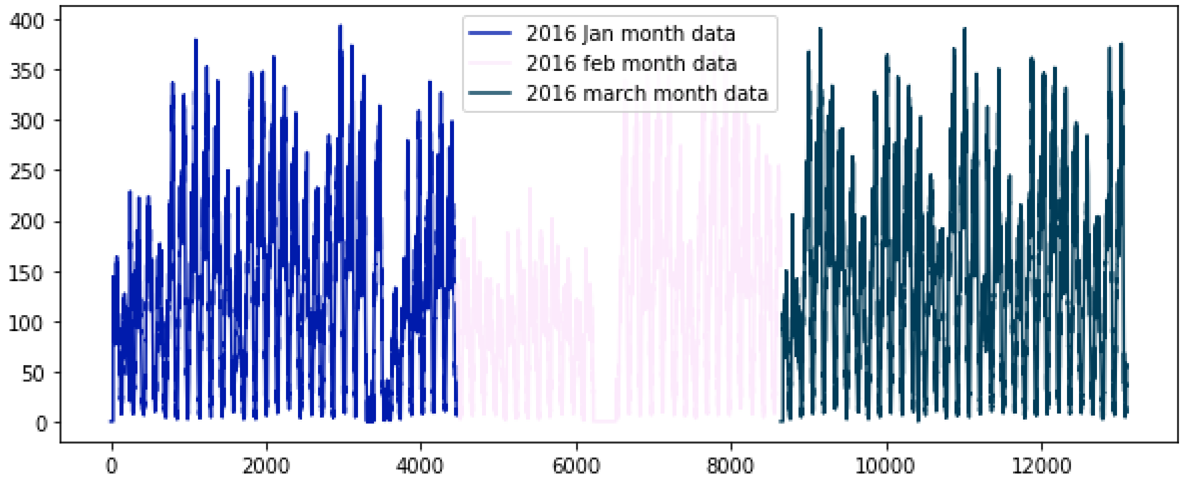

2016 Jan month data
2016 feb month data
2016 march month data

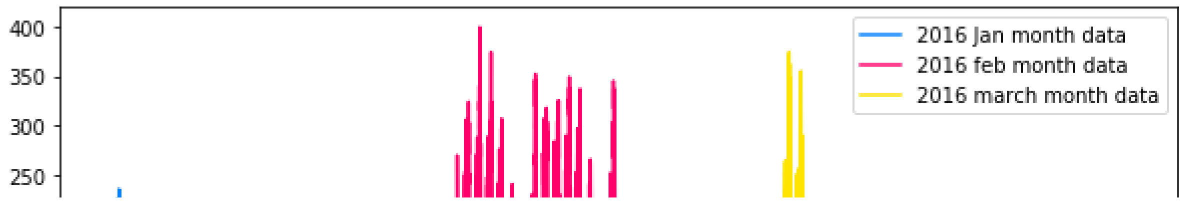

2016 Jan month data
2016 feb month data

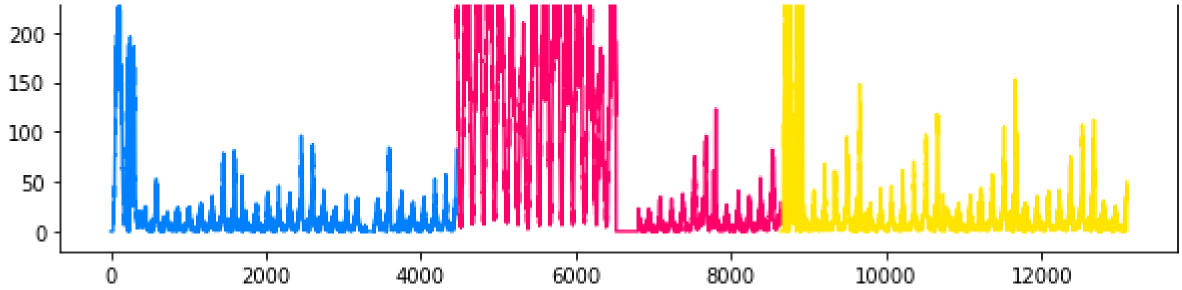

2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

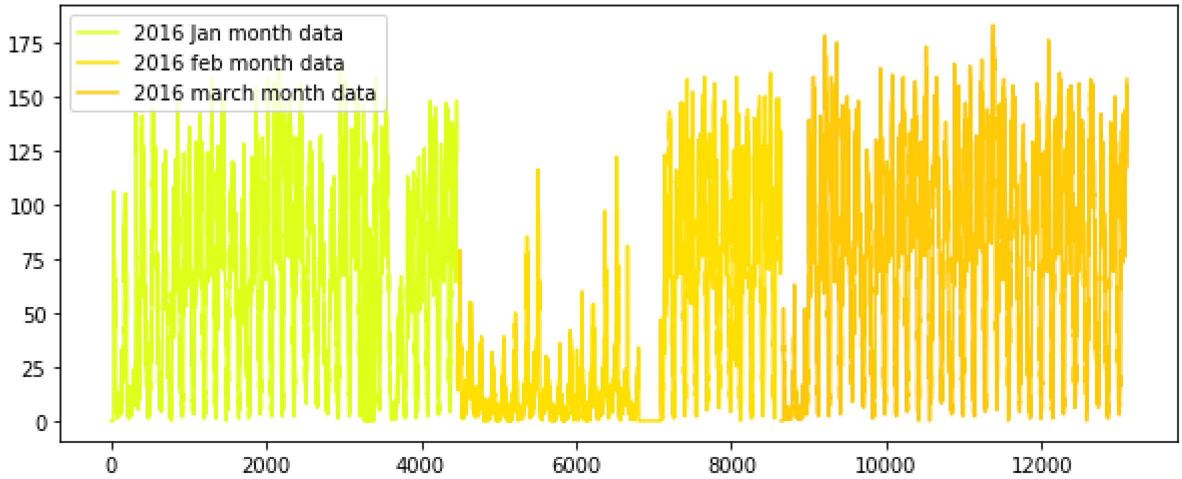

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
12000

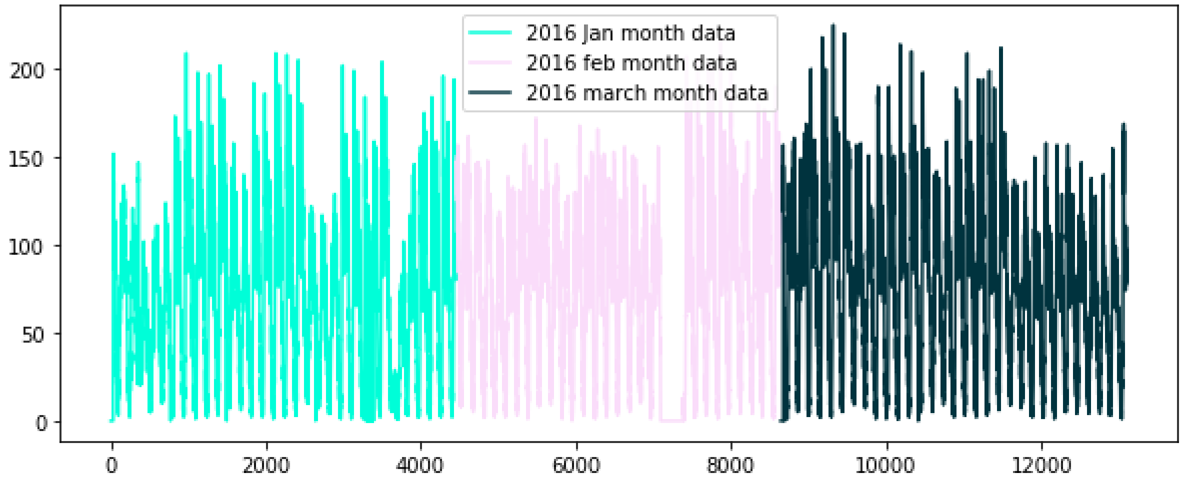

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

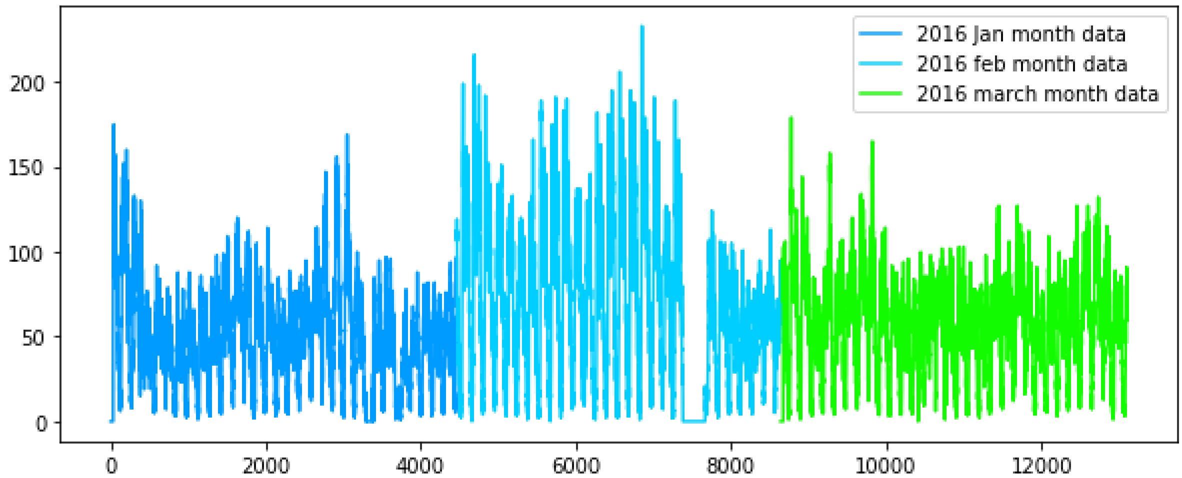

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
400
0
2000
4000
6000
8000
10000
12000

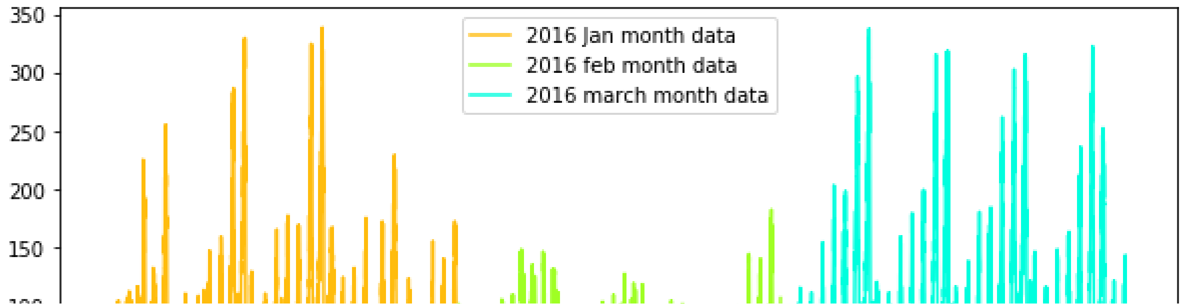

2016 Jan month data
2016 feb month data
2016 march month data
250
300
350
400

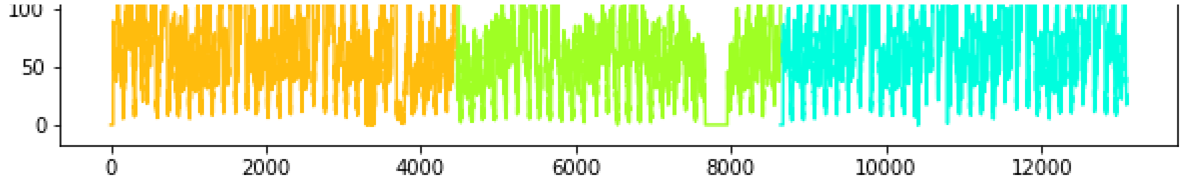


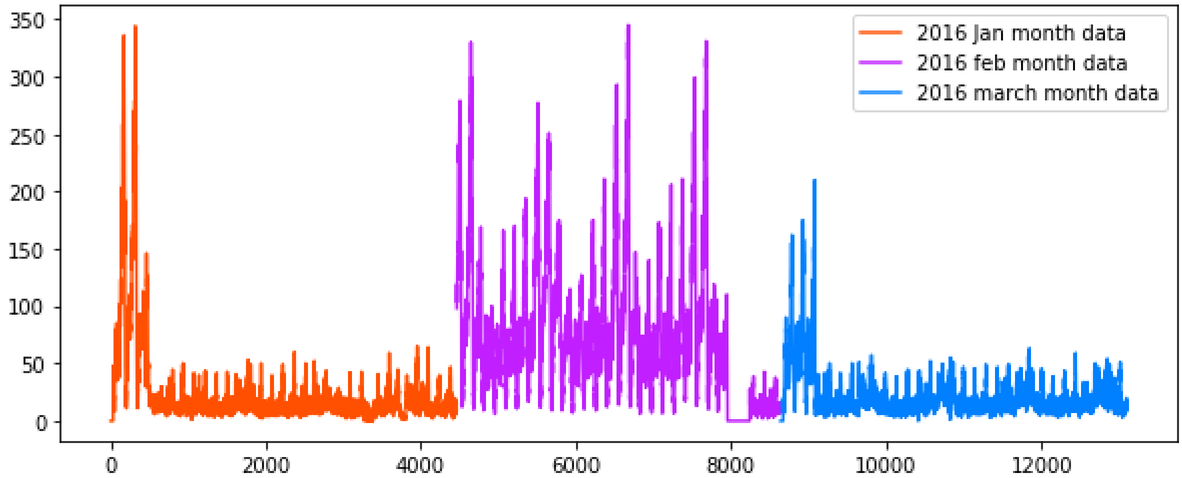

2016 Jan month data
2016 feb month data
2016 march month data

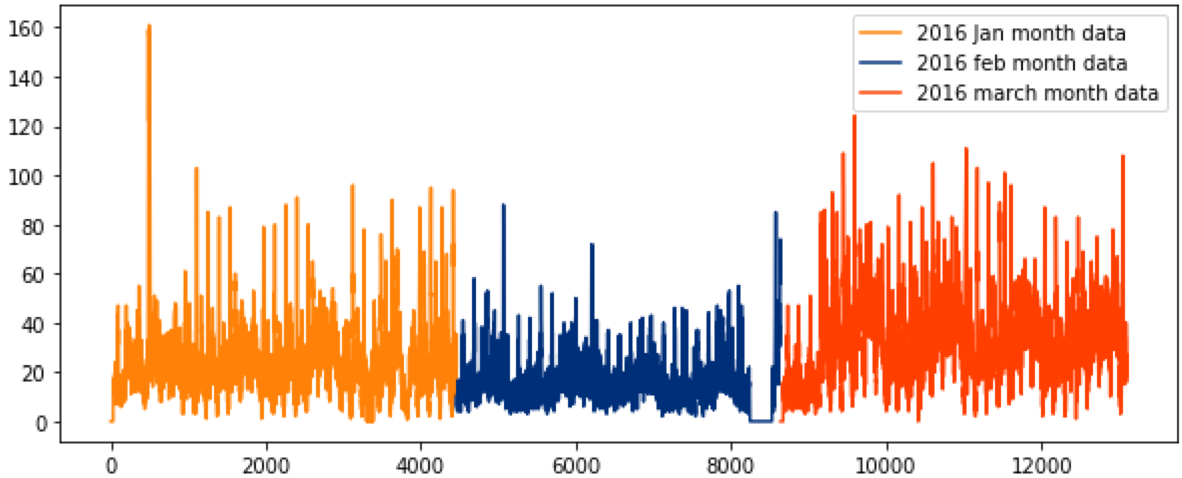

2016 Jan month data
2016 feb month data
2016 march month data

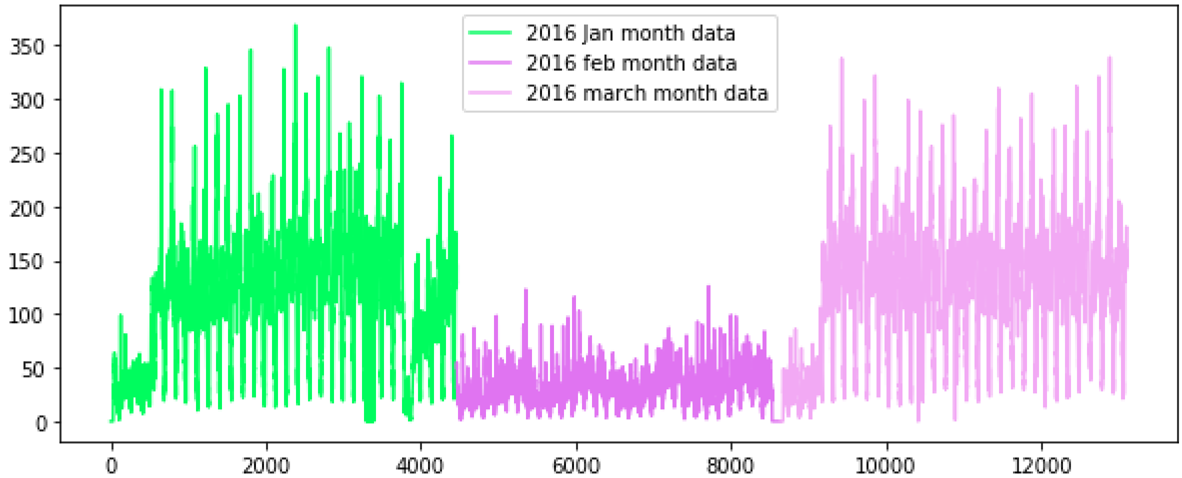

2016 Jan month data
2016 feb month data
2016 march month data

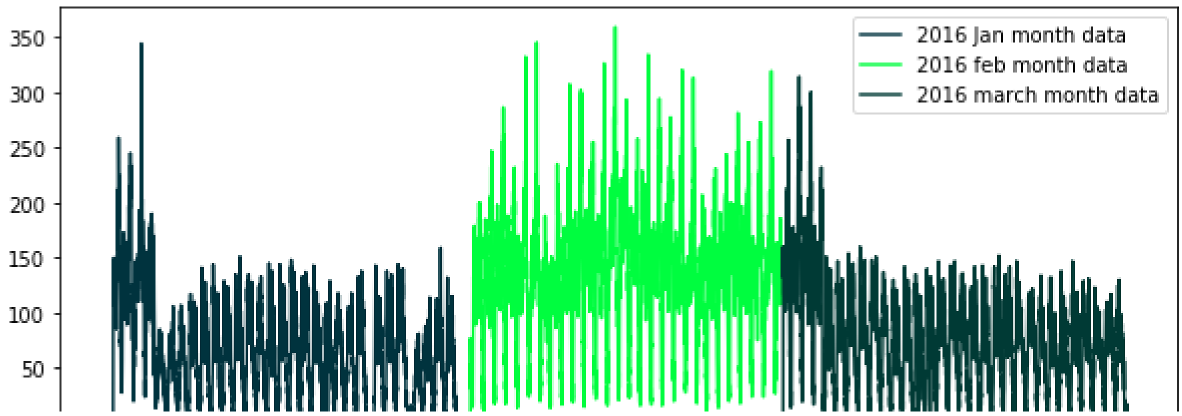

2016 Jan month data
2016 feb month data
2016 march month data

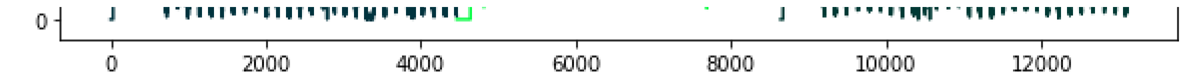

100
50
0
0 2000 4000 6000 8000 10000 12000

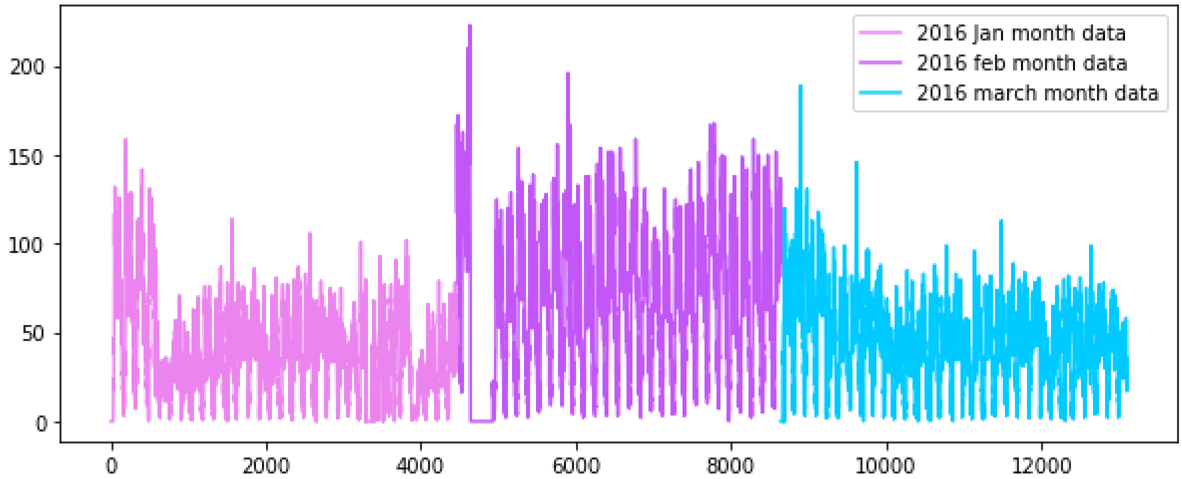

2016 Jan month data
2016 feb month data
2016 march month data
350 300 250 200 150 100 50 0
0 2000 4000 6000 8000 10000 12000

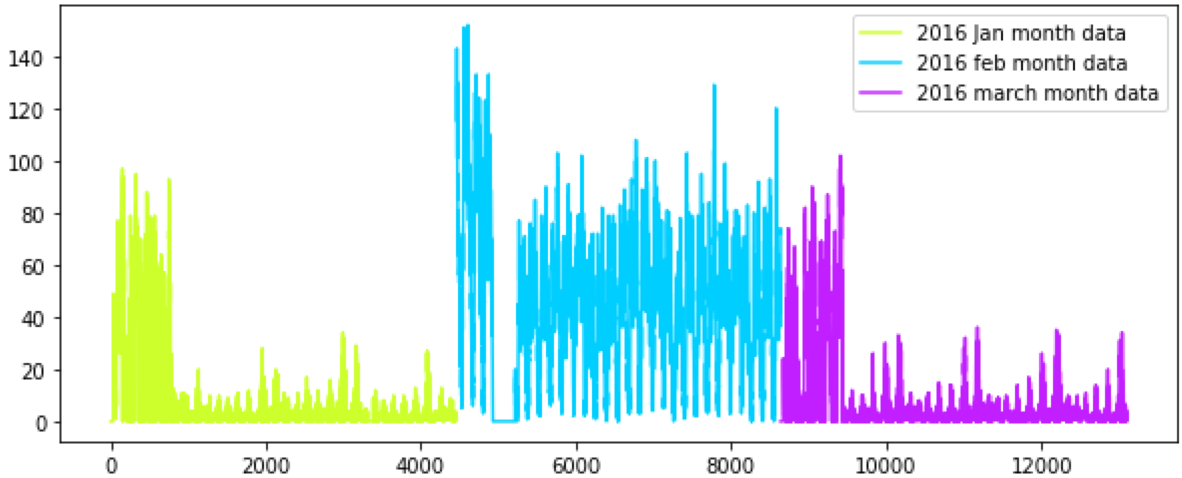

2016 Jan month data
2016 feb month data
2016 march month data
160 140 120 100 80 60 40 20 0
0 2000 4000 6000 8000 10000 12000

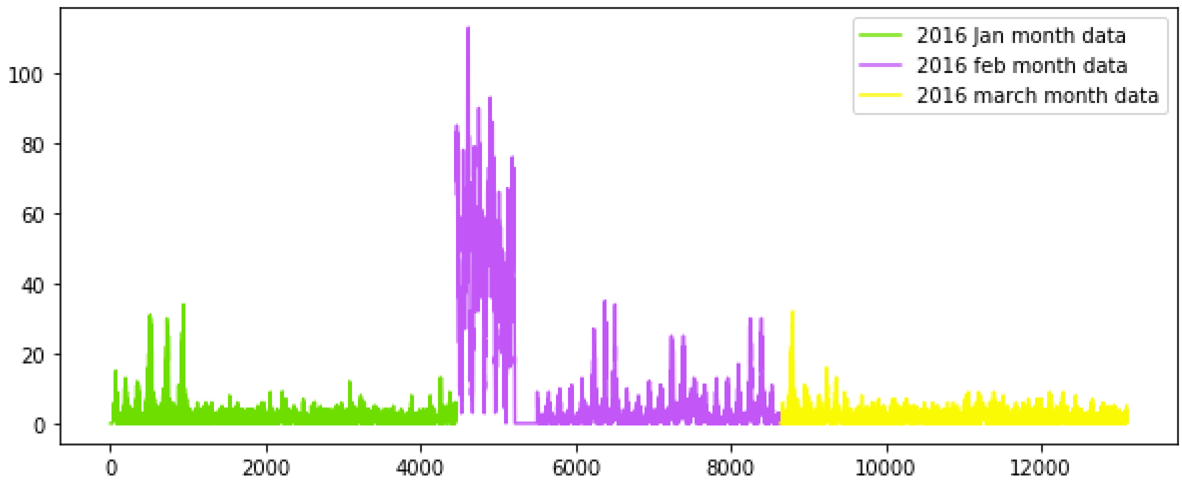

2016 Jan month data
2016 feb month data
2016 march month data
350 300 250 200 150 100 50 0
0 2000 4000 6000 8000 10000 12000

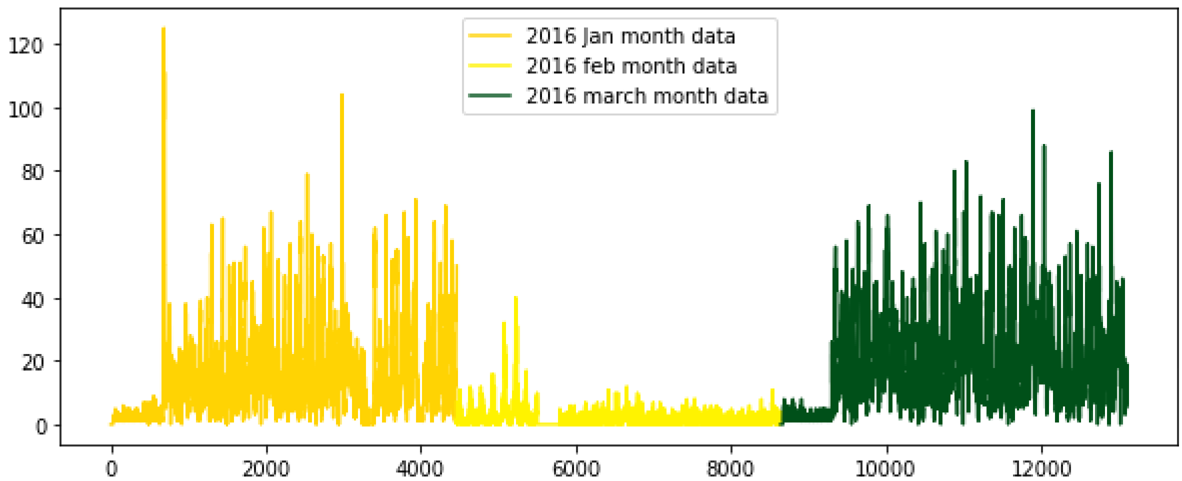

2016 Jan month data
2016 feb month data
2016 march month data
350 300 250 200 150 100 50

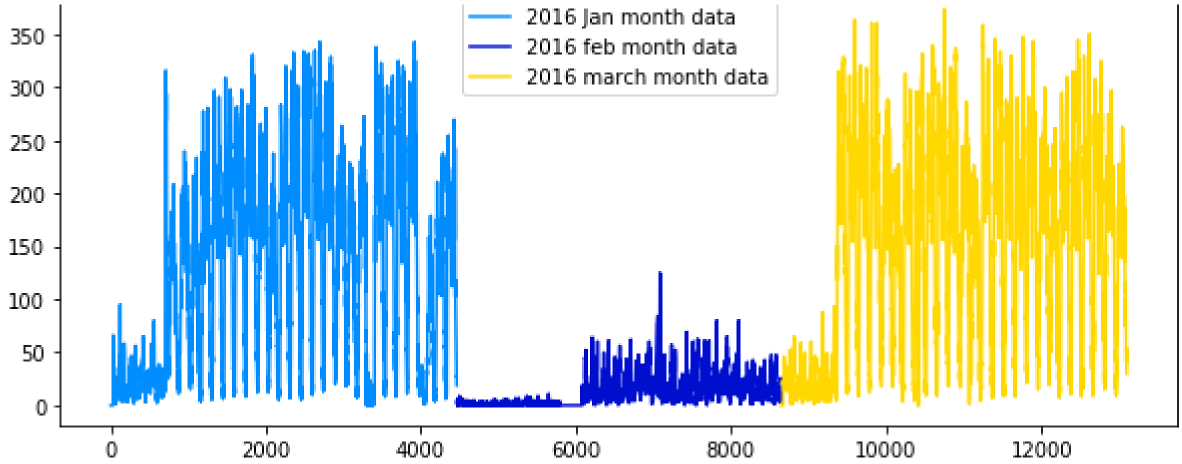


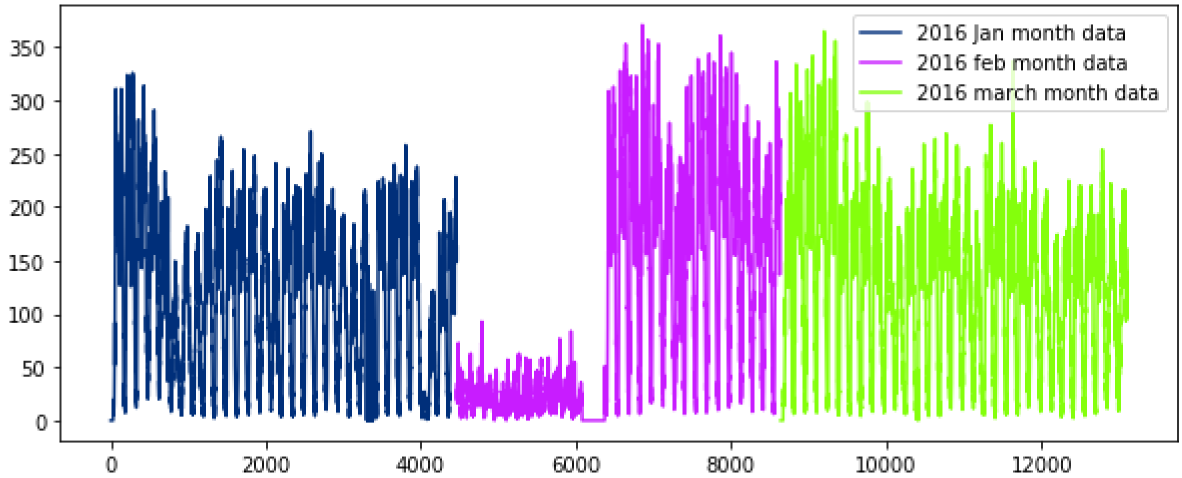

2016 Jan month data
2016 feb month data
2016 march month data

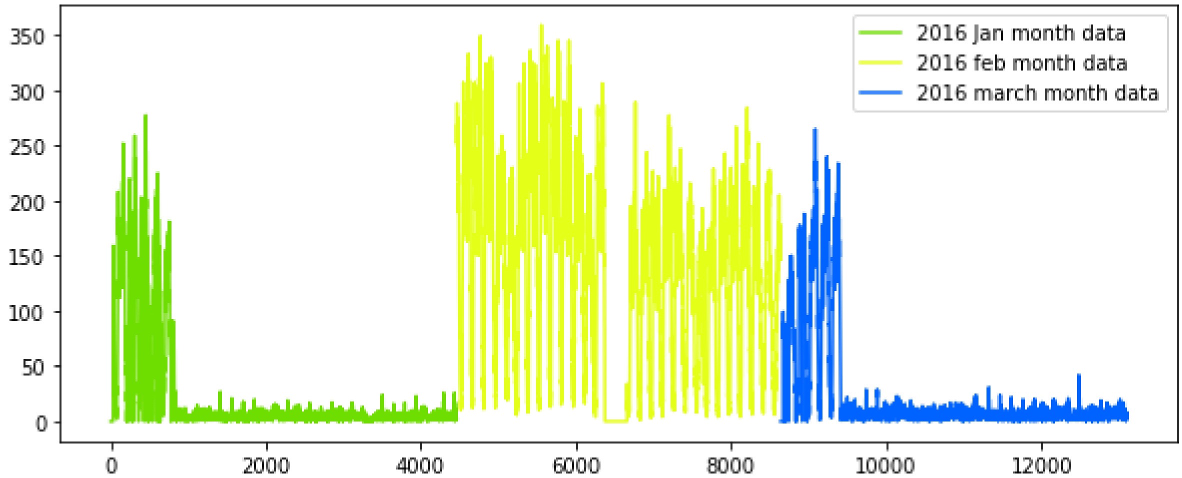

2016 Jan month data
2016 feb month data
2016 march month data

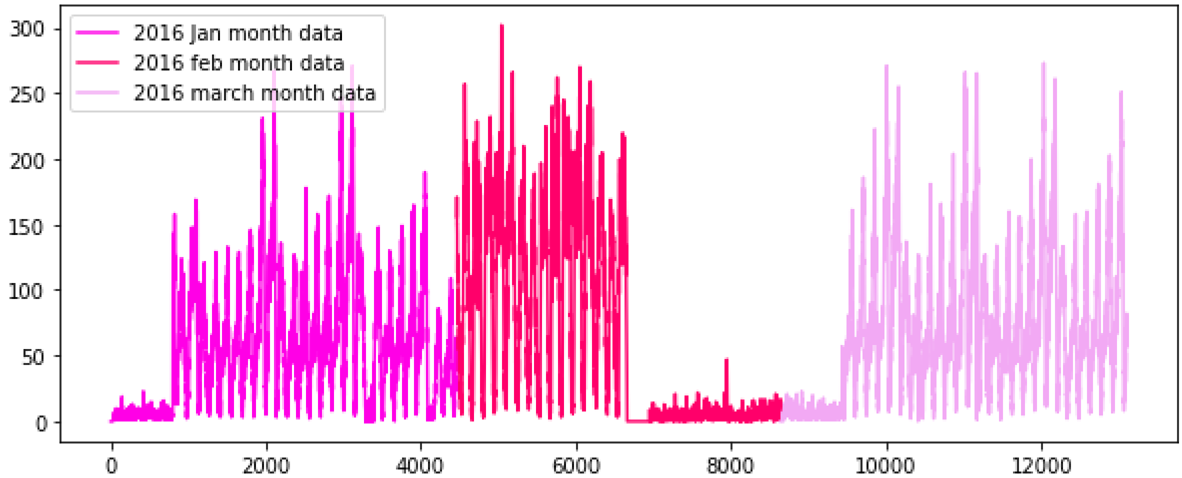

2016 Jan month data
2016 feb month data
2016 march month data


2016 Jan month data
2016 feb month data
2016 march month data

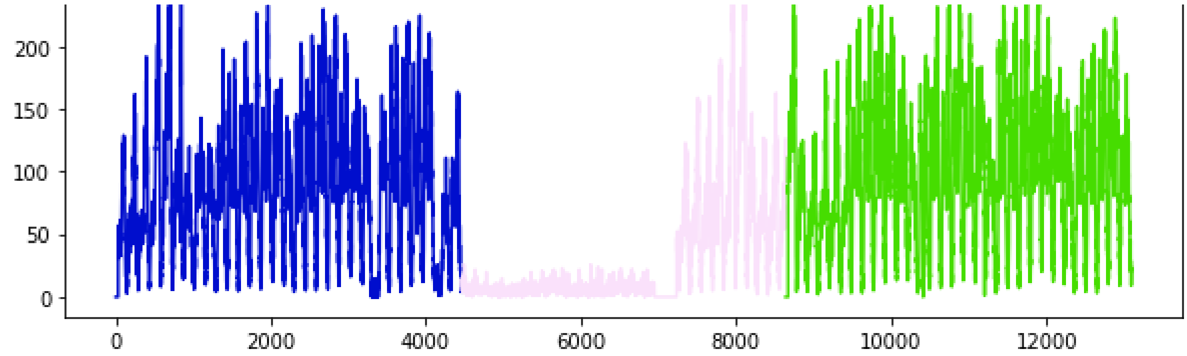

2016 Jan month data
2016 feb month data
2016 march month data

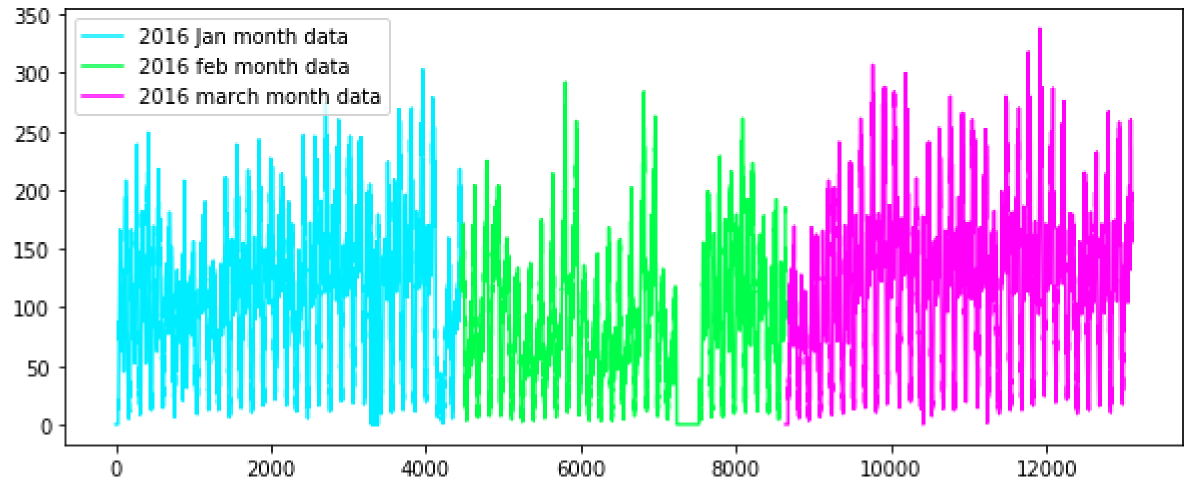

2016 Jan month data
2016 feb month data
2016 march month data

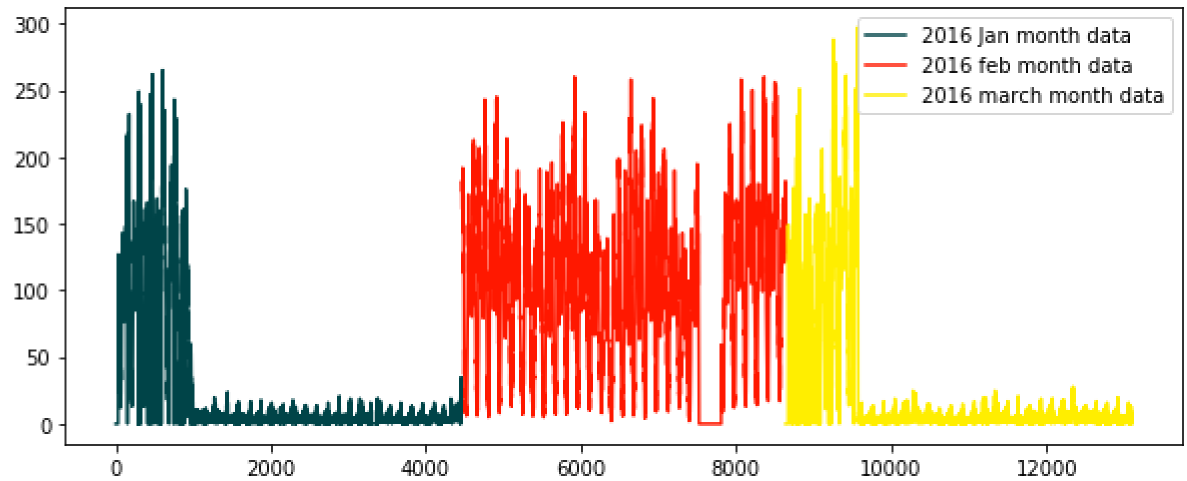

2016 Jan month data
2016 feb month data
2016 march month data

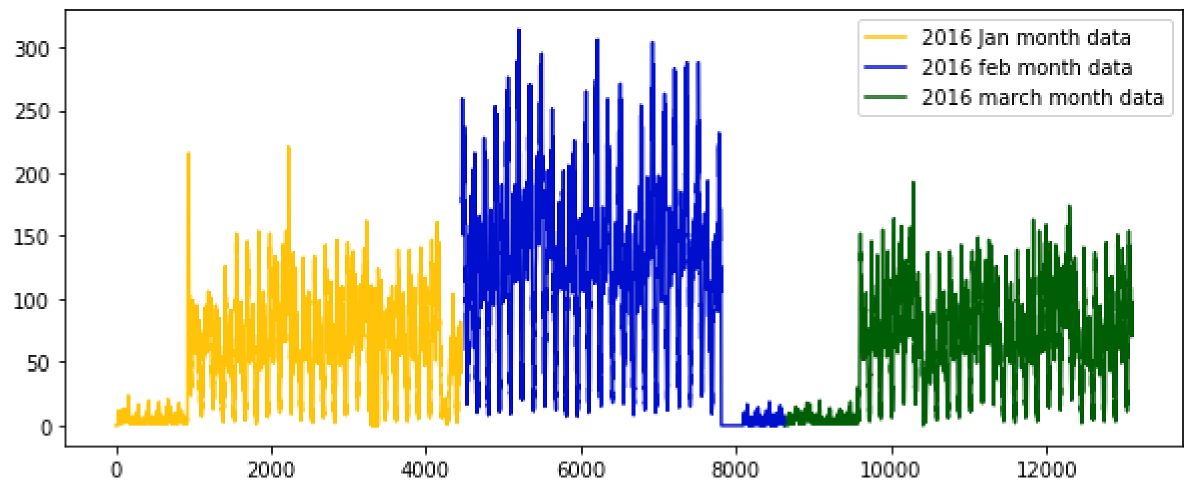

2016 Jan month data
2016 feb month data
2016 march month data

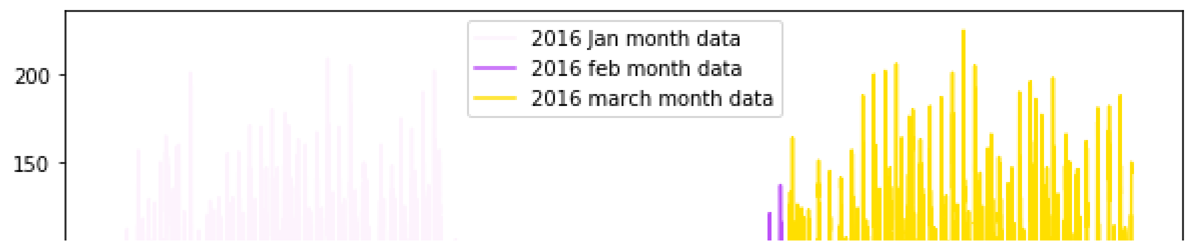

2016 Jan month data
2016 feb month data
2016 march month data

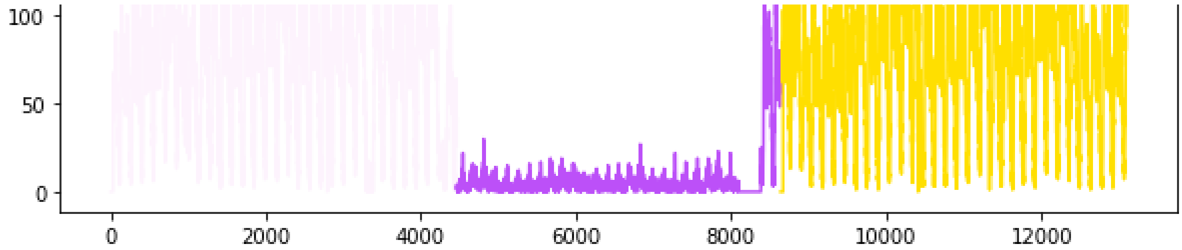

0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

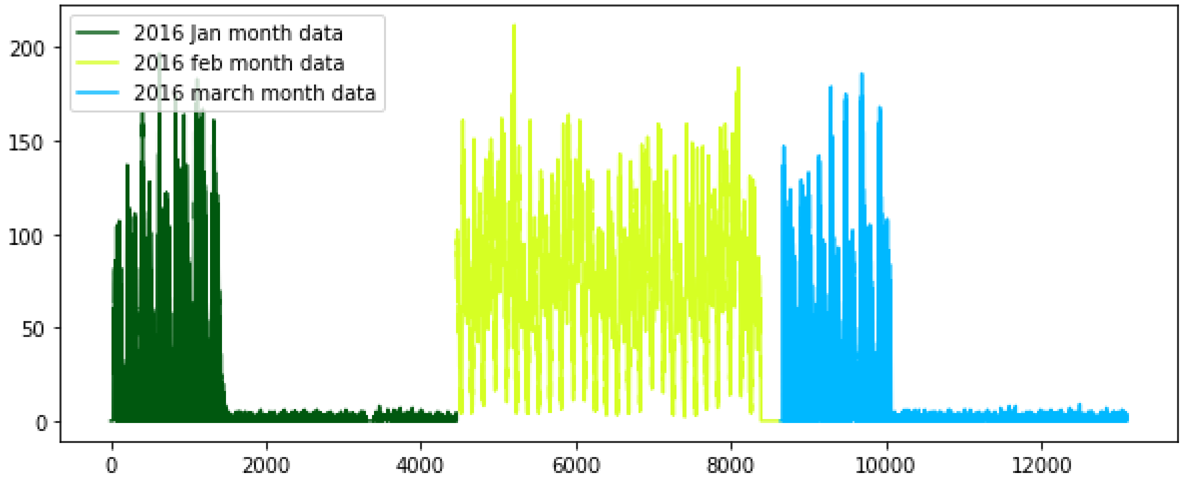

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

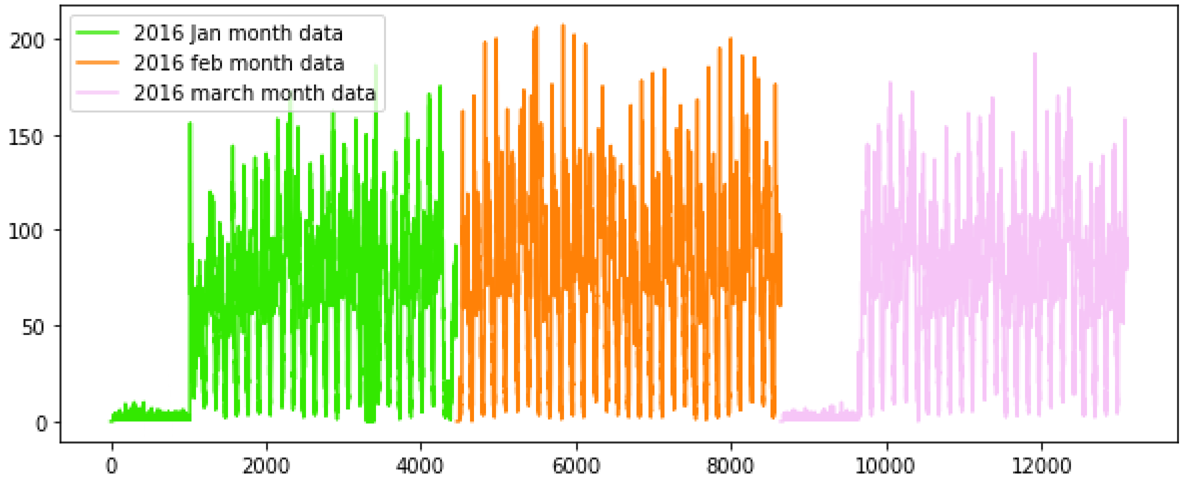

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

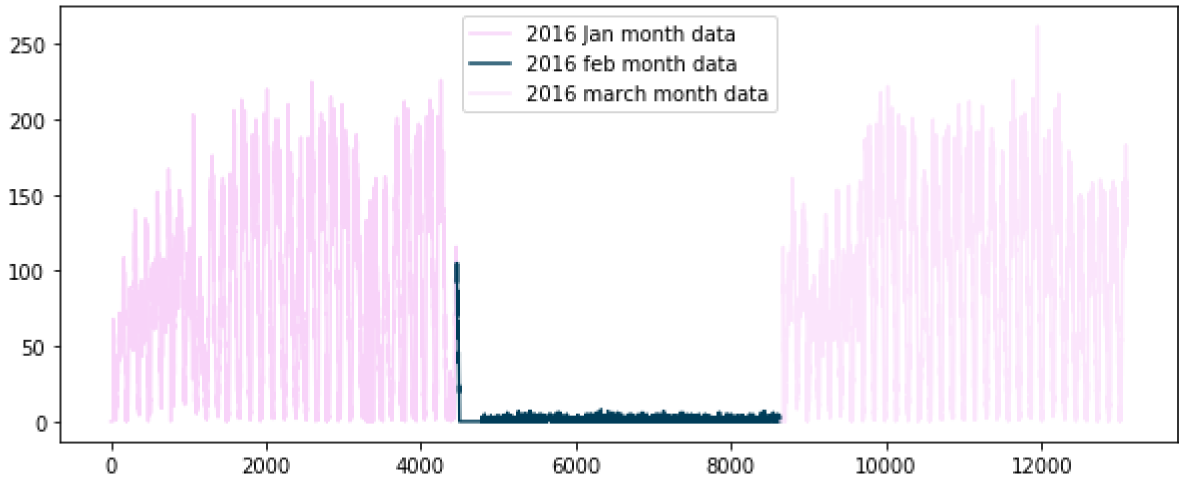

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

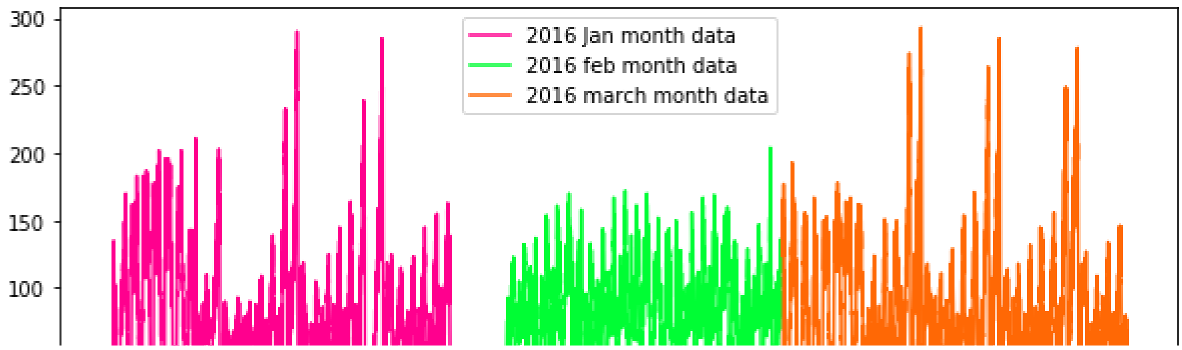

2016 Jan month data
2016 feb month data
2016 march month data
150
200

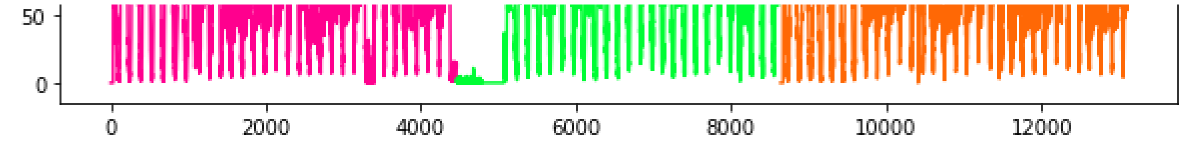


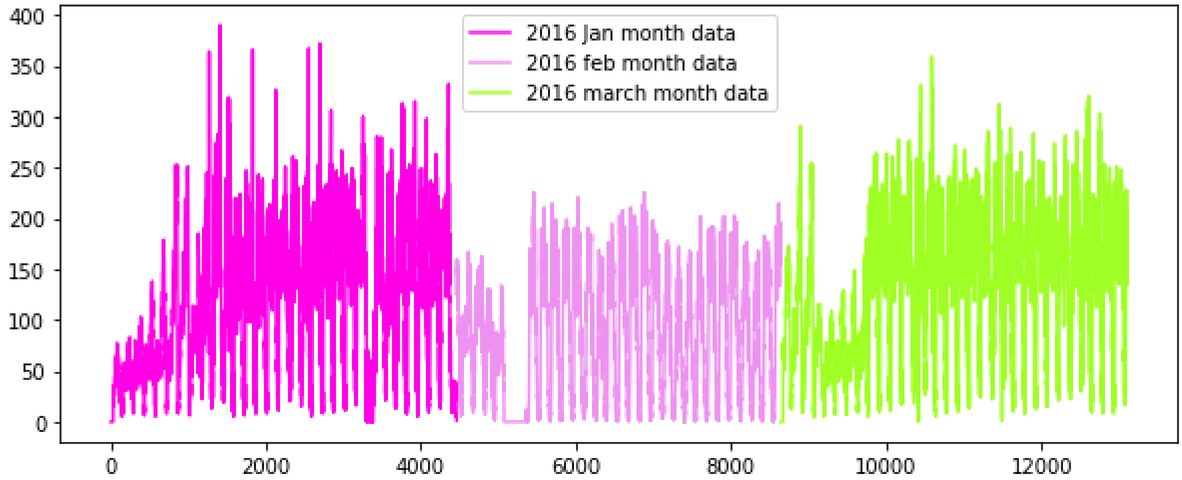

2016 Jan month data
2016 feb month data
2016 march month data

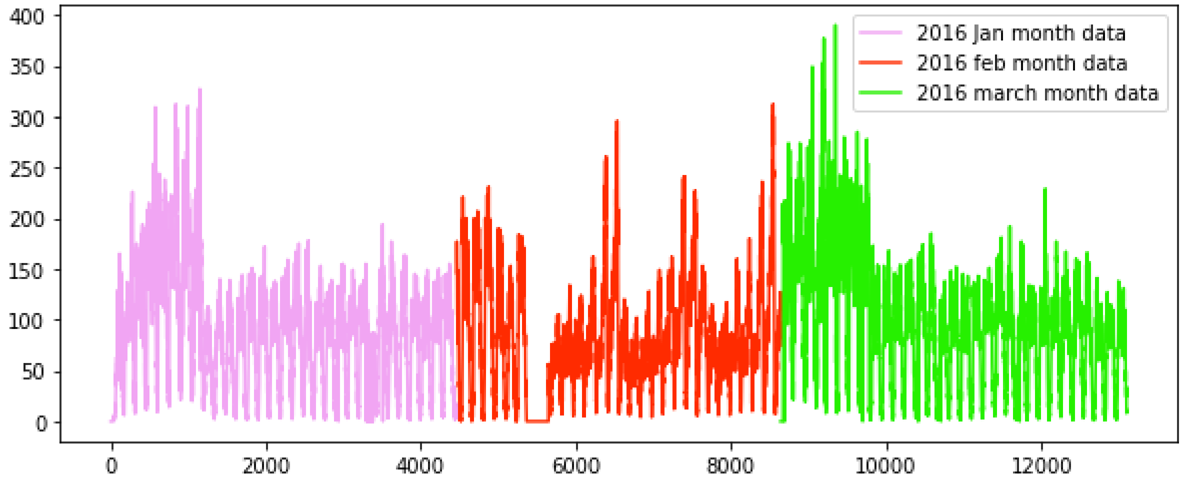

2016 Jan month data
2016 feb month data
2016 march month data

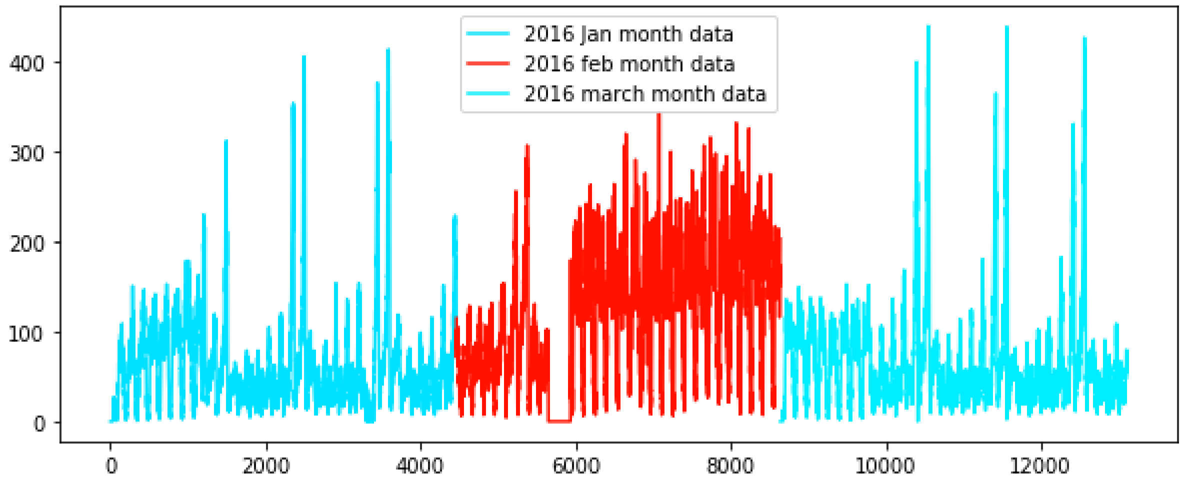

2016 Jan month data
2016 feb month data
2016 march month data

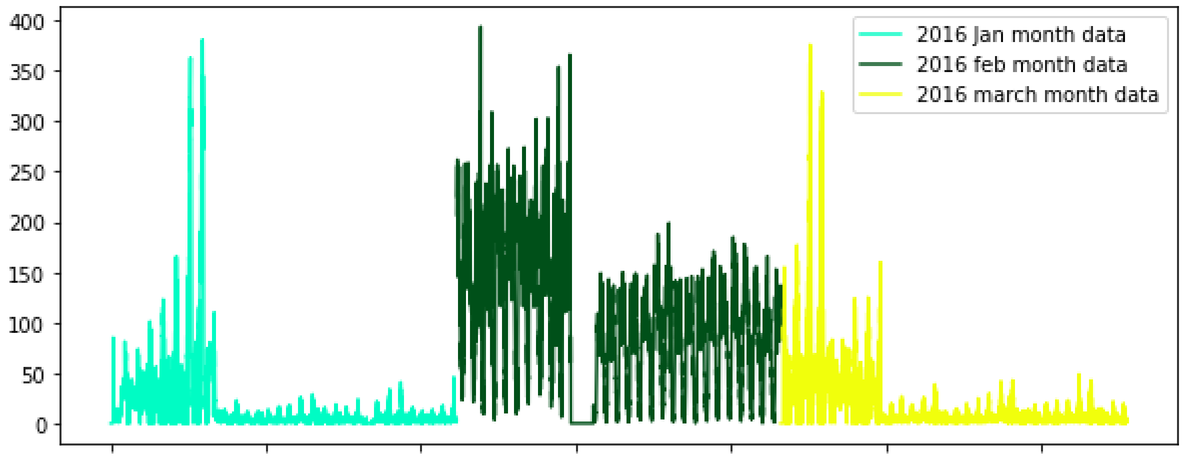

2016 Jan month data
2016 feb month data
2016 march month data

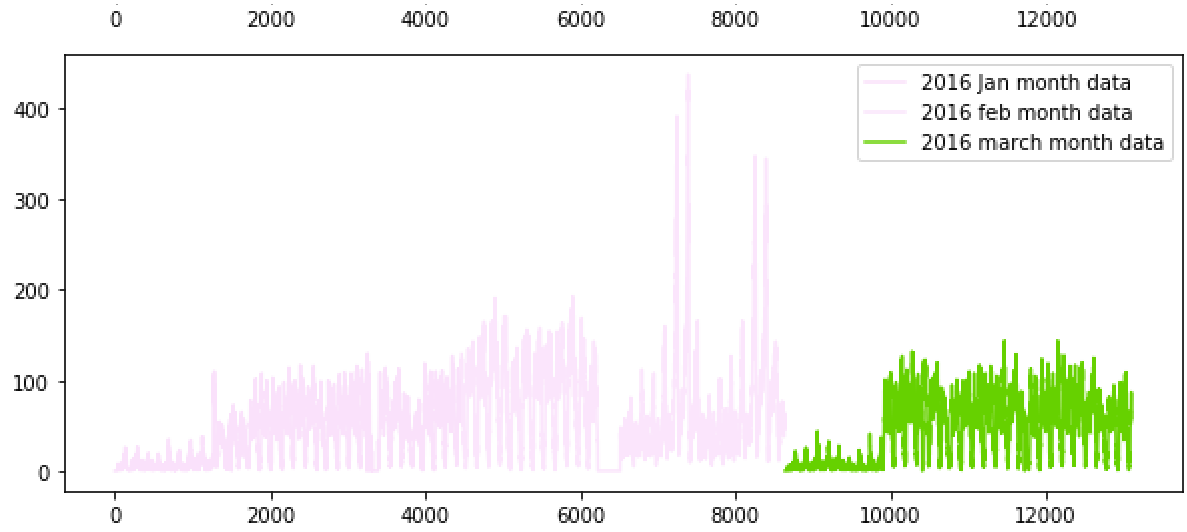


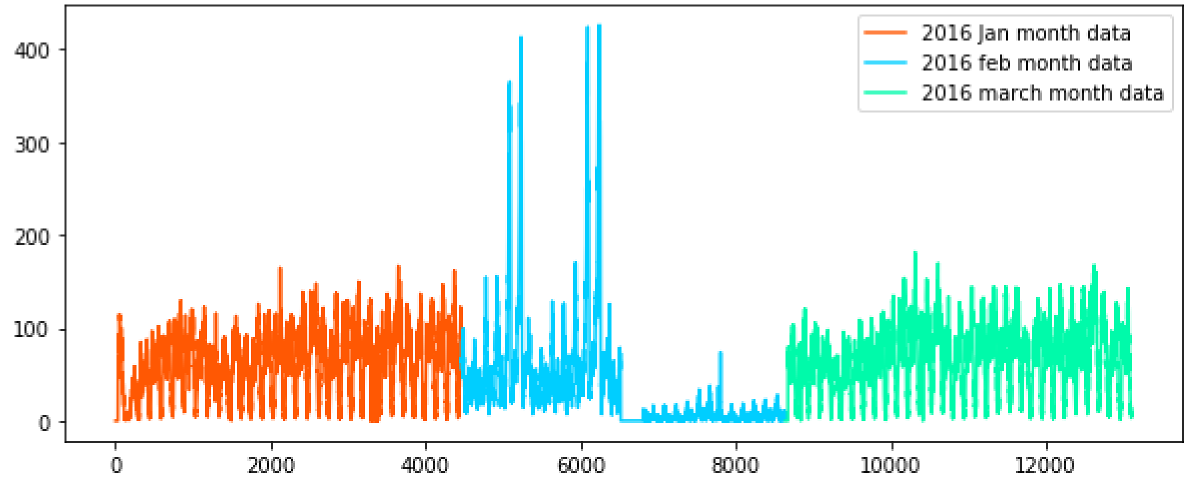

2016 Jan month data
2016 feb month data
2016 march month data


2016 Jan month data
2016 feb month data
2016 march month data


2016 Jan month data
2016 feb month data
2016 march month data


2016 Jan month data
2016 feb month data
2016 march month data

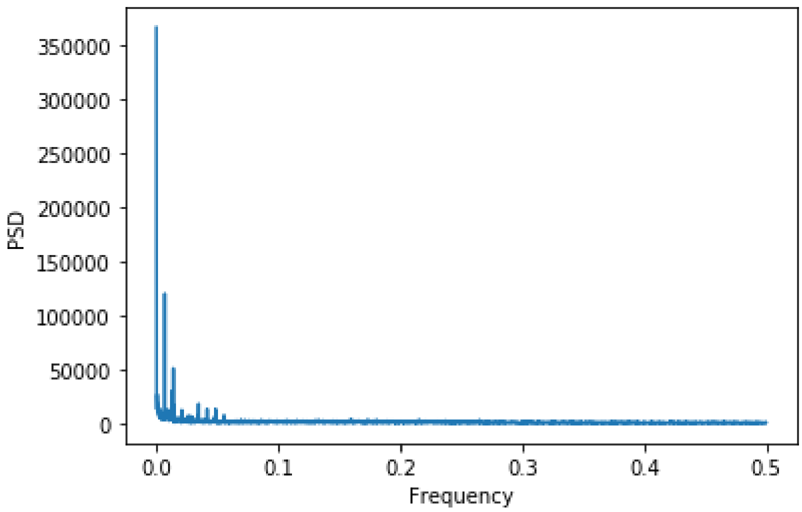





```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generated/nump
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated/numpy.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

```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) values as jan-201
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
```

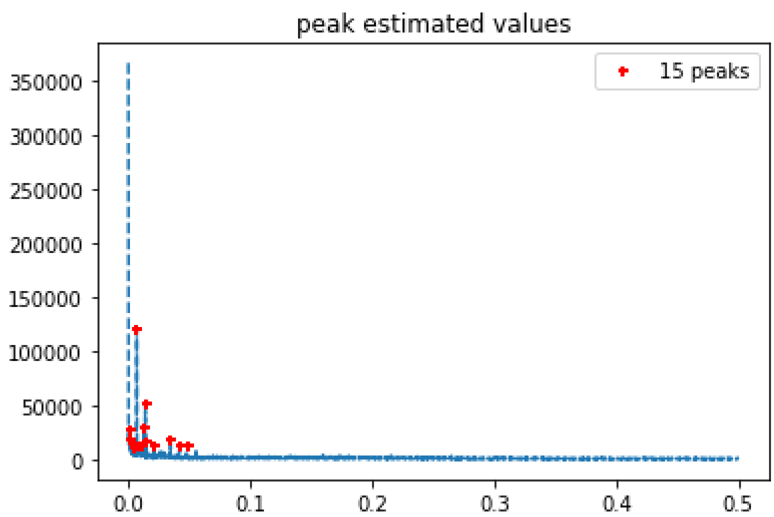
```
def p_freq(freq,Y1):
    '''The Amplitude spectrum in frequency domian is a complex space
       so take absolute values of amplitude i.e PSD.

       The amplitude values are symmetric with y axis acting as the mirror so half of the
       frequency space is sufficient to record all the frequency peaks'''
    n = len(freq) # x is freq

    f = np.abs(freq)[:int(n/2)]
    a = np.abs(Y1)[:int(n/2)]

    return f,a



freq_val, amp_val = p_freq(freq,Y)

plt.figure()
plt.plot(freq_val, amp_val )
plt.xlabel("Frequency")
plt.ylabel("PSD")
plt.show()
```




```
def get_amp(amp_values,t):
    '''returns incices of the peaks'''
    indices = peakutils.indexes(amp_values, thres=t, min_dist=1,thres_abs=True)
    return indices


!pip install PeakUtils
import peakutils
from peakutils.plot import plot as pplot
```

```
Collecting PeakUtils
  Downloading https://files.pythonhosted.org/packages/0a/11/6416c8aebba4d5f73e23e1f07
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Installing collected packages: PeakUtils
Successfully installed PeakUtils-1.3.3
```

```
t = 10000 #threshold
ind = get_amp(amp_val,t)

plt.figure()
pplot(freq_val, amp_val, ind)
plt.title('peak estimated values')
plt.show()

print('extracted peaks \n',amp_val[ind])
```




```
extracted peaks
 [ 26825.59458603  17517.0751569   14061.37132052  10161.41784112
 120325.55631502  12733.81926209  12103.83527708  13365.60746262
  30094.62607722  51086.06596691  16843.52145572  12419.84572283
  18176.89474404  13463.97304533  13364.86000587]
```

```
def freq_psd(months):
    '''Discrete frequency transformation using fast fourier tranform'''
    '''Each cluster is transformed and processed separtly'''
    '''Returns top 5 amp and corresponding freq values for each cluster'''
    psds = []
    freqs = []
    for i in range(40):
        amp = np.fft.fft(months[i][:]) # returns complex values
        fre = np.fft.fftfreq(1304,1)

        freq,amp = p_freq(fre,amp)

        t1=10000 # peak threshold
        amp_index = get_amp(amp,t1)

        # sorting decending order , returns indices
        sorted_index = np.argsort(-(amp[amp_index]))
```

```
        top5 = sorted_index[0:5]

        top5_amp = list(amp[top5])
        top5_freq = list(fre[top5])

        psds.append(top5_amp)
        freqs.append(top5_freq)
    return psds,freqs


smoothened_year_16 = []
for i in range(0,40):
    smoothened_year_16.append(jan_2016_smooth[4464*i:4464*(i+1)]  \
                              +feb_2016_smooth[4176*i:4176*(i+1)] \
                              +mar_2016_smooth[4464*i:4464*(i+1)])


psds,frequencies = freq_psd(smoothened_year_16)


print('no. of clusters',len(psds))
print('no. of top values',len(psds[0]))
```

```
no. of clusters 40
no. of top values 5
```

Modelling: Baseline Models

Now we get into modelling in order to forecast the pickup densities for the months of Jan, Feb anc
multiple models with two variations

1. Using Ratios of the 2016 data to the 2015 data i.e $R_t = P^{2016}_t / P^{2015}_t$
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 p

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

```
def MA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    error=[]
    predicted_values=[]
    window_size=3
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
```

```
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(rat
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])/window
        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['Predi
    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 tl the best results using Moving Averages using previous Ratio values therefore we get $R_t = (R_{t-1}$

Next we use the Moving averages of the 2016 values itself to predict the future value using $P_t =$

```
def MA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=1
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_size:(i+1)]
        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['Predi
    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 tl 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 likely to be similar to the latest values and less similar to the older values. Weighted Averages con relationship giving the highest weight while computing the averages to the latest previous value ar older ones

Weighted Moving Averages using Ratio Values -
$R_t = (N * R_{t-1} + (N-1) * R_{t-2} + (N-2) * R_{t-3} \ldots 1 * R_{t-n})/(N * (N+1)/2)$

```
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_ratio)-(rat
        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['Predi
    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 tl
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 -
$P_t = (N * P_{t-1} + (N-1) * P_{t-2} + (N-2) * P_{t-3} \ldots 1 * P_{t-n})/(N * (N+1)/2)$

```
def WA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
```

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

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

    ratios['WA_P_Predicted'] = predicted_values
    ratios['WA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Predi
    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 tl
the best results using Weighted Moving Averages using previous 2016 values therefore we get P_t

Exponential Weighted Moving Averages

https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average Through weighted av
giving higher weights to the latest value and decreasing weights to the subsequent ones but we st
scheme as there are infinetly many possibilities in which we can assign weights in a non-increasin
window-size. To simplify this process we use Exponential Moving Averages which is a more logica
same time also using an optimal window-size.

In exponential moving averages we use a single hyperparameter alpha (α) which is a value betw
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
10 days prior before we predict the value for the current iteration. Also the weights are assigned us
of prior values being considered, hence from this it is implied that the first or latest value is assign
exponentially decreasing for the subsequent values.

$$R_t' = \alpha * R_{t-1} + (1 - \alpha) * R_{t-1}'$$

```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
```

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

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

$$P_t^{'} = \alpha * P_{t-1} + (1-\alpha) * P_{t-1}^{'}$$

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

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

```
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 Perc** average how good is our model with predictions and **MSE (Mean Squared Error)** is also used so th well our forecasting model performs with outliers so that we make sure that there is not much of a actual value

```
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],"      MSE
print ("Exponential Moving Averages (2016 Values) -         MAPE: ",mean_err[5],"      MSE
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
-------------------------------------------------------------------------------------
Moving Averages (Ratios) -                              MAPE:  0.22785156353133512
Moving Averages (2016 Values) -                         MAPE:  0.15583458712025738
-------------------------------------------------------------------------------------
Weighted Moving Averages (Ratios) -                     MAPE:  0.22706529144871415
Weighted Moving Averages (2016 Values) -                MAPE:  0.1479482182992932
-------------------------------------------------------------------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.2275474636148534      M
Exponential Moving Averages (2016 Values) -         MAPE:  0.1475381297798153      M
```

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:- *I* Exponential Moving Averages using 2016 Values

Feature engineering to reduce MAPE value by using -Holt-Wi

The number of exponential smoothings depends on the number of features that you want to use ir is level, trend, and seasonality. For all three, it's usually triple exponential smoothing

```
#Code from: https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-pa
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
        return sum / slen

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

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)*(smoot
            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

#Initializing the Holt-Winters method: the variables have been initialised after reading t
# https://robjhyndman.com/hyndsight/hw-initialization/

alpha = 0.2
beta = 0.1
gamma = 0.1
season = 24

#Cluster features for all points
predicted_values_HW =[]
predicted_list_HW = []
for r in range(0,40):
    predicted_values_HW = triple_exponential_smoothing(smoothened_year_16[r][0:13104], sea
    predict_list_HW.append(predict_values_HW[5:])
```

Regression Models

Train-Test Split

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

```
# Preparing data to be split into train and test, The below prepares data in cumulative fo
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*31*60/10 = 4464
# regions_cum: it will contain 40 lists, each list will contain 4464+4176+4464 values whic
# that are happened for three months in 2016 data
# print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 12960
# we take number of pickups that are happened in last 5 10min intravels

number_of_time_stamps = 5

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


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


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

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

# its an numbpy array, of shape (523960, 5)
# each row corresponds to an entry in out data
# for the first row we will have [f0,f1,f2,f3,f4] fi=number of pickups happened in i+1th 1
# 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 pickups
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

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

```
True
```

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

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

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

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x131
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    for i in range(0,13104):
        if i==0:
            predicted_value= smoothened_year_16[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(smoothened_year_16[r][i]
    predict_list.append(predicted_values[5:])
    predicted_values=[]

#frequencies and amplitudes are same for all the points a cluster
psd_feature = [0]*40
freq_feature = [0]*40
```

```
for c in range(40):
    psd = []
    freq = []

    for k in range(13104):
        psd.append(psds[c])
        freq.append(frequencies[c])

    psd_feature[c]=psd
    freq_feature[c]=freq


# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 months of 2
# 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
```

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our train
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)]


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

```
Number of data clusters 40 Number of data points in trian data 9169 Each data point c
Number of data clusters 40 Number of data points in test data 3930 Each data point co
```

```
psd_train = [psd_feature[i][5:9169+5] for i in range(40)]
psd_test = [psd_feature[i][9169+5:] for i in range(40)]


freq_train = [freq_feature[i][5:9169+5] for i in range(40)]
freq_test = [freq_feature[i][9169+5:] for i in range(40)]


train_psds = sum(psd_train, [])
test_psds  = sum(psd_test, [])

train_freqs = sum(freq_train, [])
test_freqs   = sum(freq_test, [])


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

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


# the above contains values in the form of list of lists (i.e. list of values of each regi
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])


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

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


# 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,[])
tsne_test_HW = sum(tsne_test_flat_HW,[])


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

```
(366760, 10)
```

```
# Preparing the data frame for our train data
df_test = pd.DataFrame(data=test_new_features, columns=columns)
df_test['lat'] = tsne_test_lat
df_test['lon'] = tsne_test_lon
df_test['weekday'] = tsne_test_weekday
df_test['exp_avg'] = tsne_test_exp_avg
df_test['triple_exp'] = tsne_test_flat_HW
print(df_test.shape)
```

```
(157200, 10)
```

```
df_test.head()
```

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	triple_exp
0	143	145	119	113	124	40.776228	-73.982119	4	121	115.279892
1	145	119	113	124	121	40.776228	-73.982119	4	120	117.445892
2	119	113	124	121	131	40.776228	-73.982119	4	127	111.143612
3	113	124	121	131	110	40.776228	-73.982119	4	115	113.343013
4	124	121	131	110	116	40.776228	-73.982119	4	115	128.924512

```
df_train_n =  pd.DataFrame()
```

```
psd_train=pd.DataFrame(train_psds)
psd_test=pd.DataFrame(test_psds)
freq_train=pd.DataFrame(train_freqs)
freq_test=pd.DataFrame(test_freqs)
```

```
print(psd_train.shape)
print(psd_test.shape)
print(freq_train.shape)
print(freq_test.shape)
```

```
(366760, 5)
(157200, 5)
(366760, 5)
(157200, 5)
```

```
psd_train =psd_train.fillna(method='ffill')
psd_test = psd_test.fillna(method='ffill')
```

```
freq_train =freq_train.fillna(method='ffill')
freq_test = freq_test.fillna(method='ffill')
```

```
df_train_n=pd.concat([psd_train,freq_train,df_train],axis=1)
```

```
df_test_n=pd.concat([psd_test,freq_test,df_test],axis=1)
```

```
print(df_train_n.shape)
print(df_test_n.shape)
```

```
(366760, 20)
(157200, 20)
```

```
from sklearn.preprocessing import  StandardScaler
scalar =  StandardScaler()
df_train_new = scalar.fit_transform(df_train_n)
df_test_new = scalar.transform(df_test_n)
```

Using Linear Regression

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


def LR_reg(df_train,df_test, train_output):


    LR = linear_model.SGDRegressor(loss="squared_loss")


    alpha =  [0.00001,0.000001,0.000002,0.000005]
    itera = [300,400,500,600]

    param = {"alpha": alpha, "max_iter":itera}
    best_model = GridSearchCV(LR, param_grid= param, scoring = "neg_mean_absolute_error",n

    best_model.fit(df_train, train_output)

    y_pred =  best_model.best_estimator_.predict(df_train)
    lr_train_predictions = [round(value) for value in y_pred]

    y_pred = best_model.best_estimator_.predict(df_test)
    lr_test_predictions = [round(value) for value in y_pred]

    print(best_model.best_params_)

    return lr_train_predictions, lr_test_predictions


lr_train_predictions,lr_test_predictions = LR_reg(df_train_new,df_test_new, tsne_train_out
```

```
{'alpha': 1e-06, 'max_iter': 300}
```

```
train_mape_lr= (mean_absolute_error(tsne_train_output, lr_train_predictions))/(sum(tsne_tr
test_mape_lr= (mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_test_
print(train_mape_lr)
print(test_mape_lr)
```

0.10388363427466933

Using Random Forest Regressor

```
from sklearn.model_selection import RandomizedSearchCV
def RF_regsn(df_train,df_test,train_output):
    n_est = [200,400,600,800]
    max_dep =[10,13,16,19]
    min_split = [8,10,12,15]
    start = [False]
    param = {'n_estimators':n_est ,'max_depth': max_dep,'min_samples_split':min_split
            ,'warm_start':start }

    RF_reg = RandomForestRegressor(max_features='sqrt', n_jobs=4)

    model_2 =  RandomizedSearchCV(RF_reg, param_distributions= param, scoring = "neg_mean_

    model_2.fit(df_train, train_output)

    y_pred = model_2.best_estimator_.predict(df_test)
    rndf_test_predictions = [round(value) for value in y_pred]
    y_pred = model_2.best_estimator_.predict(df_train)
    rndf_train_predictions = [round(value) for value in y_pred]
    print(model_2.best_params_)
    return rndf_train_predictions,rndf_test_predictions
```

```
rndf_train_predictions,rndf_test_predictions = RF_regsn(df_train_new,df_test_new,tsne_trai
```

```
{'warm_start': False, 'n_estimators': 600, 'min_samples_split': 8, 'max_depth': 19}
```

```
train_mape_RF=(mean_absolute_error(tsne_train_output,rndf_train_predictions))/(sum(tsne_tr
test_mape_RF= (mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_tes
print(train_mape_RF)
print(test_mape_RF)
```

```
0.06818674066872511
0.08659775096354515
```

Using XgBoost Regressor

```
def xg_reg(df_train,df_test,train_output):

    c_param={'learning_rate' :[0.001,0.01,0.1,0.2],
      'n_estimators':[100,200,500,800],
      'max_depth':[5,7,8,10]}

    xreg= xgb.XGBRegressor(nthread = 4)
    model3 = RandomizedSearchCV(xreg, param_distributions= c_param, scoring = "neg_mean_ab

    model3.fit(df_train, train_output)
```

```
    y_pred = model3.predict(df_test)
    xgb_test_predictions = [round(value) for value in y_pred]
    y_pred = model3.predict(df_train)
    xgb_train_predictions = [round(value) for value in y_pred]
    print(model3.best_params_)

    return xgb_train_predictions,xgb_test_predictions
```

```
xgb_train_predictions,xgb_test_predictions=xg_reg(df_train_new,df_test_new,tsne_train_outp
```

```
[17:18:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:20:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:23:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:26:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:28:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:31:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:33:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:35:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:37:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:40:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:43:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:47:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:51:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:51:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:51:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:51:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:54:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:57:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[17:59:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:01:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:03:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:04:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:05:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:05:20] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:05:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:05:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:06:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:06:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:07:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:09:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[18:10:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
{'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1}
```

```
train_mape_xgb=(mean_absolute_error(tsne_train_output, xgb_train_predictions))/(sum(tsne_t
test_mape_xgb= (mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_tes
print(train_mape_xgb)
print(test_mape_xgb)
```

```
0.08753173751798735
0.0869364767184682
```

Calculating the error metric values for various models

```
train_mape=[]
test_mape=[]
```

```
train_mape.append((mean_absolute_error(tsne_train_output,df_train['ft_1'].values))/(sum(ts
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].values))/(sum
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_predictions))/(sum(tsn
train_mape.append((mean_absolute_error(tsne_train_output, xgb_train_predictions))/(sum(tsn
train_mape.append((mean_absolute_error(tsne_train_output, lr_train_predictions))/(sum(tsne

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/(sum(tsne
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(sum(t
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_t
test_mape.append((mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_te
```

Error Metric Matrix

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

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------
Baseline Model -                  Train:  0.14870666996426116 Test:  0.14225522601041
Exponential Averages Forecasting -Train:  0.14121603560900353 Test:  0.13490049942819
--------------------------------------------------------------------------------------
MAPE for models after feature engineering
--------------------------------------------------------------------------------------
Linear Regression -           Train:  0.10388363427466933 Test:  0.0983382496206547
Random Forest Regression -    Train:  0.06818674066872511 Test:  0.08659775096354515
XgBoost Regression -          Train:  0.08753173751798735 Test:  0.0869364767184682
--------------------------------------------------------------------------------------
```

Conclusion:

There is no significant reduction in MAPE value by considering top 5 amplitude and frequency valu

So we have used feature engineering i.e Holt-winters method and sucessfully brought down MAPE

Both models i.e Random forest regressor and XGBoost regressor have performed well but Randon

But Train and Test MAPE for XGBoost regressor is almost similar so we can say it is a better mode