Taxi demand prediction in New York City

In [0]:

```
!pip install gpxpy
```

```
Collecting gpxpy
  Downloading https://files.pythonhosted.org/packages/6e/d3/ce52e67771929de455e76655365a4935a2f369f76dfb0d70c20a308ec463/gpxpy-1.3.5.tar.gz (105kB)
     |████████████████████████████████| 112kB 5.0MB/s
Building wheels for collected packages: gpxpy
  Building wheel for gpxpy (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/d2/f0/5e/b8e85979e66efec3eaa0e47fbc5274db99fd1a07befd1b2aa4
Successfully built gpxpy
Installing collected packages: gpxpy
Successfully installed gpxpy-1.3.5
```

In [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/master/07_dataframe.ipynb
import dask.dataframe as dd#similar to pandas
from tqdm import tqdm
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 intractive like zoom in and zoom out
```

```python
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 in miles
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\\bin'
# os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

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

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

%matplotlib inline
```

Data Information

https://drive.google.com/drive/u/0/folders/1okdoeVcz6peAgJRrBC1Hnx1o7zD3OhYE

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

In [0]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googlea
%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2
.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive/
```

Data Collection

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

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	1. 62G	11135470	19
yellow_tripdata_2016-07	884Mb	10294080	17
yellow_tripdata_2016-08	854Mb	9942263	17

yellow_tripdata_2016-09	870Mb	10116018	17
yellow_tripdata_2016-10	933Mb	10854626	17
yellow_tripdata_2016-11	868Mb	10102128	17
yellow_tripdata_2016-12	897Mb	10449408	17
yellow_tripdata_2015-01	1.84Gb	12748986	19
yellow_tripdata_2015-02	1.81Gb	12450521	19
yellow_tripdata_2015-03	1.94Gb	13351609	19
yellow_tripdata_2015-04	1.90Gb	13071789	19
yellow_tripdata_2015-05	1.91Gb	13158262	19
yellow_tripdata_2015-06	1.79Gb	12324935	19
yellow_tripdata_2015-07	1.68Gb	11562783	19
yellow_tripdata_2015-08	1.62Gb	11130304	19
yellow_tripdata_2015-09	1.63Gb	11225063	19
yellow_tripdata_2015-10	1.79Gb	12315488	19
yellow_tripdata_2015-11	1.65Gb	11312676	19
yellow_tripdata_2015-12	1.67Gb	11460573	19

In [0]:

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

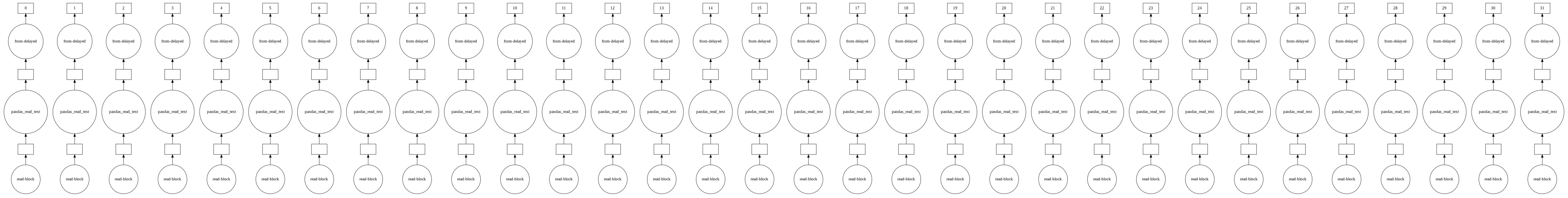
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RateCodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax',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

In [0]:

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

# to see the visulaization you need to install graphviz
```

```
# pip3 install graphviz if this doesnt work please check the install_graphviz.jpg in the drive
month.visualize()
```

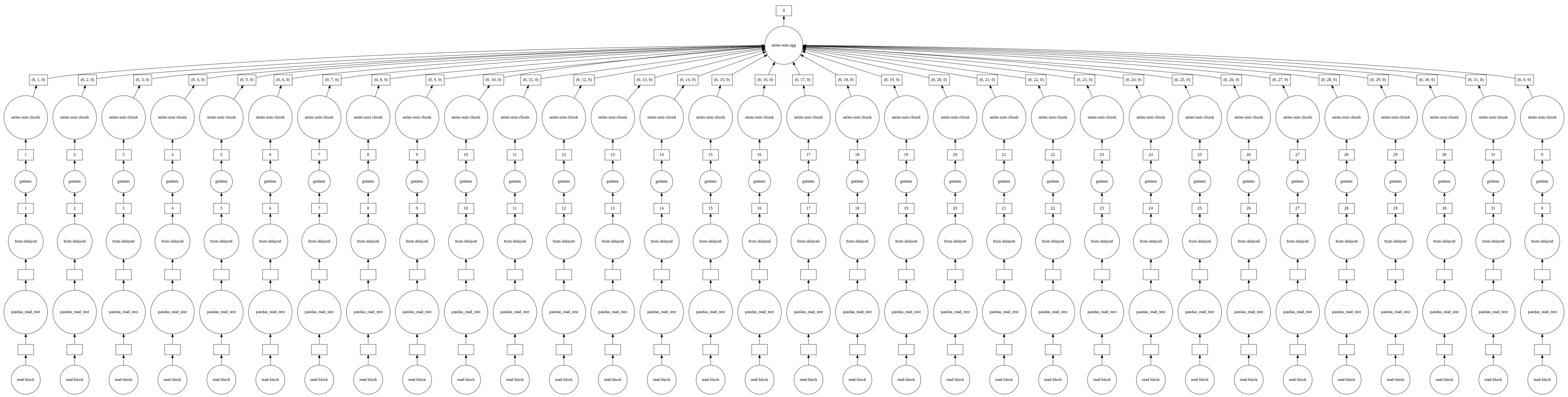
Out[0]:



In [0]:

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

Out[0]:



Features in the dataset:

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.

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

ML Problem Formulation

Time-series forecasting and Regression

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

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

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

Performance metrics

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

Data Cleaning

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

In [0]:

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

Out[0]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RateCodeID	store_and_fwd_flag	dropoff_longitude	dro
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	-73.993896	40.750111	1	N	-73.974785	40.
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30	-74.001648	40.724243	1	N	-73.994415	40.
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80	-73.963341	40.802788	1	N	-73.951820	40.
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50	-74.009087	40.713818	1	N	-74.004326	40.
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00	-73.971176	40.762428	1	N	-74.004181	40.

1. Pickup Latitude and Pickup Longitude

It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bounded by the location cordinates(lat,long) - (40.4774, -74.2589) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are not considered by us as we are only concerned with pickups which originate within New York.

In [0]:

```
# 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_locations
outlier_locations = month[((month.pickup_longitude <= -74.2589) | (month.pickup_latitude <= 40.4774)| \
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitude >= 40.9176))]

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

```
indeepth knowledge on these maps and plots

tamen Toner')

l the outliers will take more time


gitude'])))).add_to(map_osm)
```

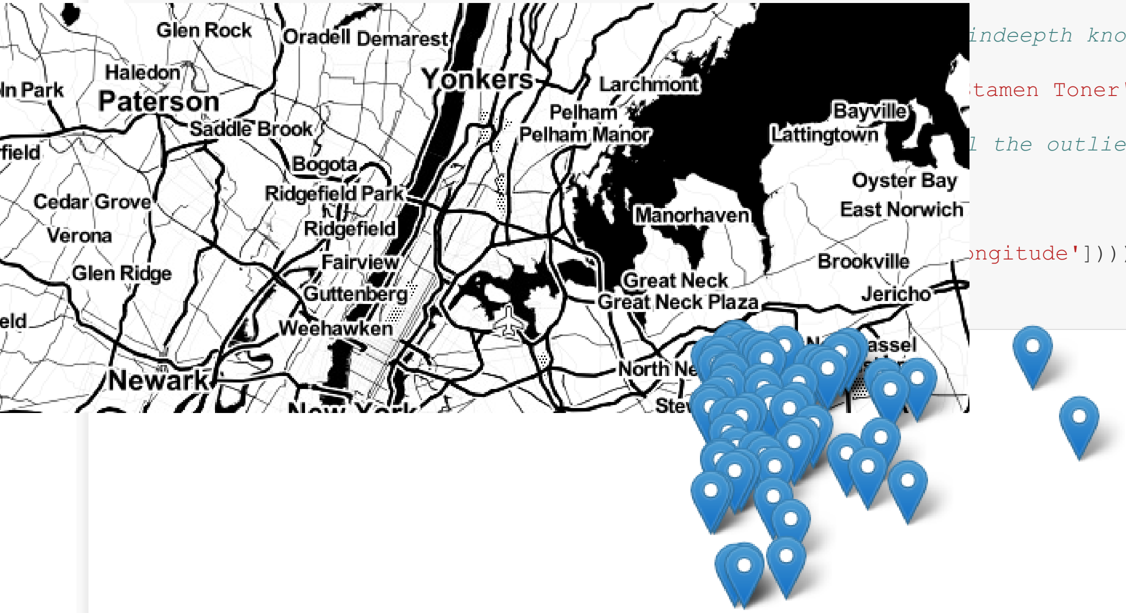
Observation:- As you can see above that there are some points just outside the boundary but there are a few that are in either South america, Mexico or Canada

2. Dropoff Latitude & Dropoff Longitude

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

In [0]:

```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locations
outlier_locations = month[((month.dropoff_longitude <= -74.2589) | (month.dropoff_latitude <= 40.4774)| \
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.9176))]

# creating a map with the a base location
# read more about the folium here: http://folium.readthedocs.io/en/latest/quickstart.html
indeepth knowledge on these maps and plots
tamen Toner')
l the outliers will take more time
ngitude']))).add_to(map_osm)
```



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

3. Trip Durations:

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

In [0]:

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

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



# we return a data frame which contains the columns
# 1.'passenger_count' : self explanatory
```

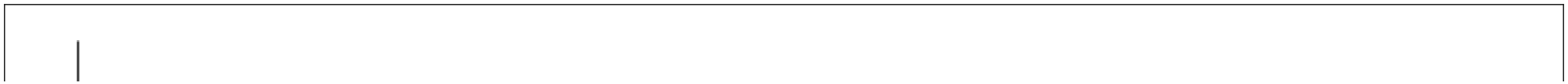
```python
# 2.'trip_distance' : self explanatory
# 3.'pickup_longitude' : self explanatory
# 4.'pickup_latitude' : self explanatory
# 5.'dropoff_longitude' : self explanatory
# 6.'dropoff_latitude' : self explanatory
# 7.'total_amount' : total fair that was paid
# 8.'trip_times' : duration of each trip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

    #append durations of trips and speed in miles/hr to a new dataframe
    new_frame = month[['passenger_count','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','total_amount']
].compute()

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

    return new_frame

# print(frame_with_durations.head())
#  passenger_count trip_distance pickup_longitude pickup_latitude dropoff_longitude dropoff_latitude total_amount trip_times pickup_times Speed
#   1                  1.59      -73.993896        40.750111      -73.974785       40.750618            17.05      18.050000 1.421329e+09 5.285319
#   1                3.30     -74.001648       40.724243    -73.994415       40.759109            17.80    19.833333 1.420902e+09 9.983193
#   1                1.80      -73.963341       40.802788      -73.951820       40.824413             10.80    10.050000 1.420902e+09 10.746269
#   1                0.50     -74.009087       40.713818     -74.004326        40.719986             4.80     1.866667 1.420902e+09 16.071429
#   1                3.00     -73.971176       40.762428     -74.004181       40.742653            16.30    19.316667 1.420902e+09 9.318378
```

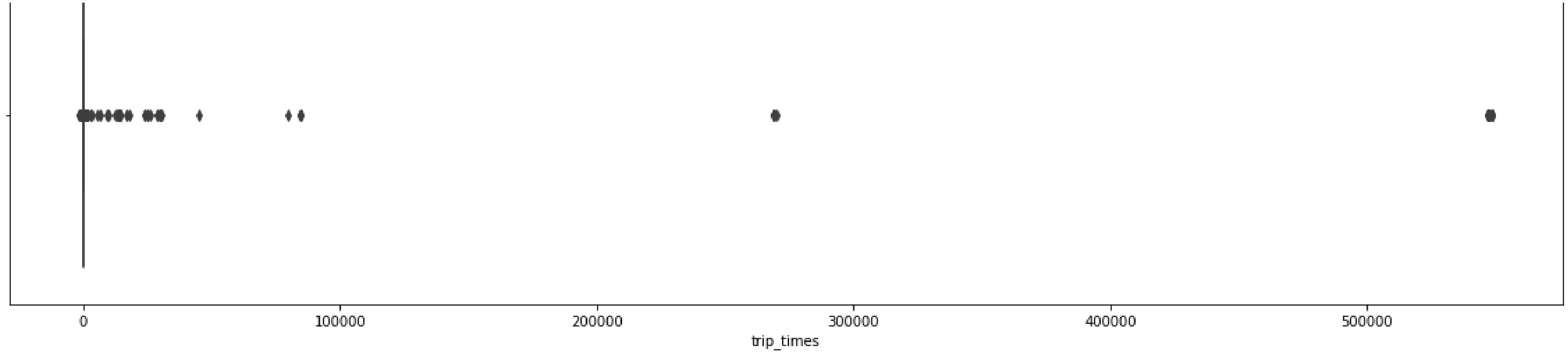
In [0]:

```python
frame_with_durations = return_with_trip_times(month)
```

In [0]:

```python
# the skewed box plot shows us the presence of outliers
plt.figure(figsize=(20,5))
sns.boxplot(y="trip_times", data =frame_with_durations, orient='h')
plt.show()
```





In [0]:

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

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

In [0]:

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

```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
```

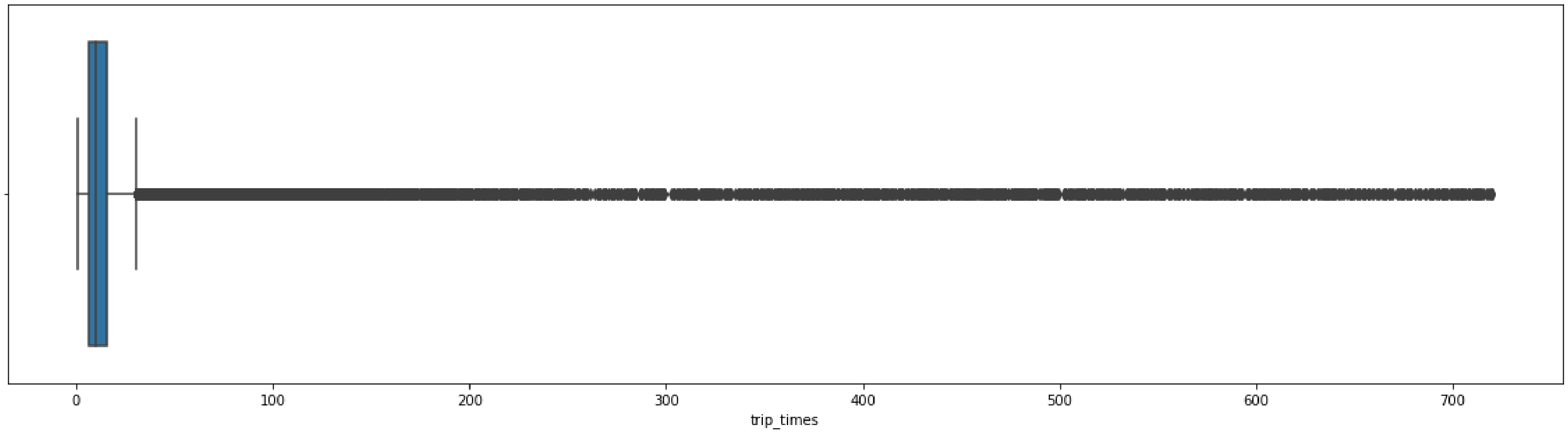
```
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98 percentile value is 38.71666666666667
99 percentile value is 46.75
100 percentile value is  548555.6333333333
```

In [0]:

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

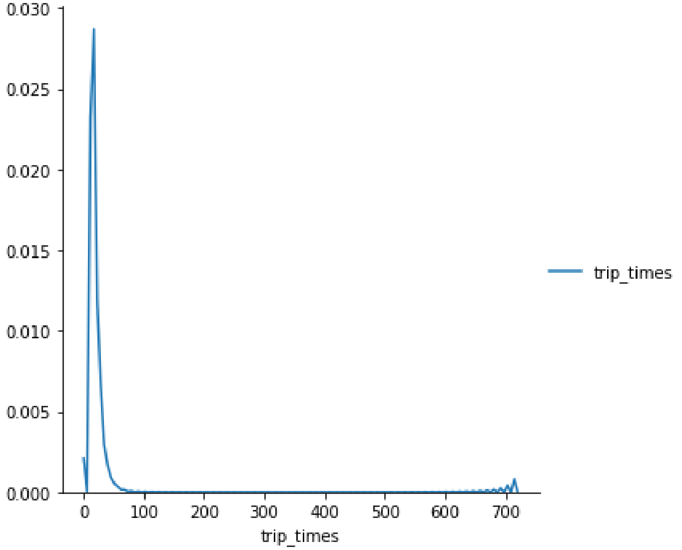
In [0]:

```
#box-plot after removal of outliers
plt.figure(figsize=(20,5))
sns.boxplot(y="trip_times", data =frame_with_durations_modified, orient='h')
plt.show()
```



In [0]:

```
#pdf of trip-times after removing the outliers
sns.FacetGrid(frame_with_durations_modified,size=5) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```

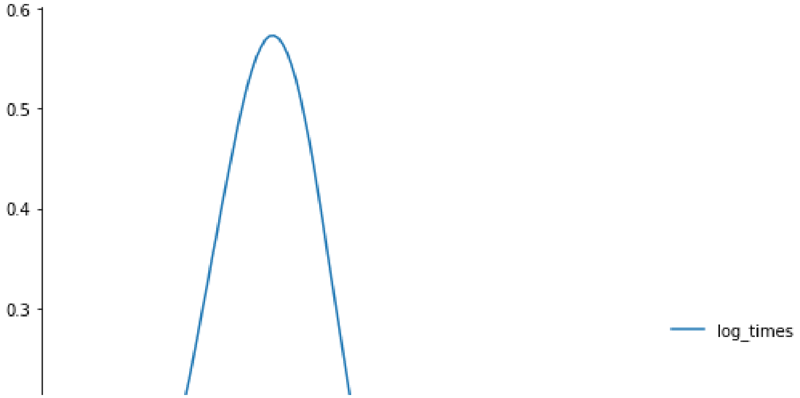


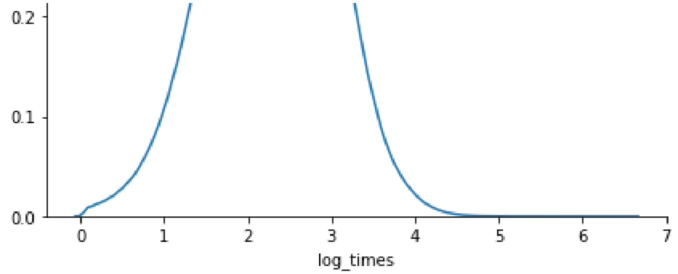
In [0]:

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

In [0]:

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



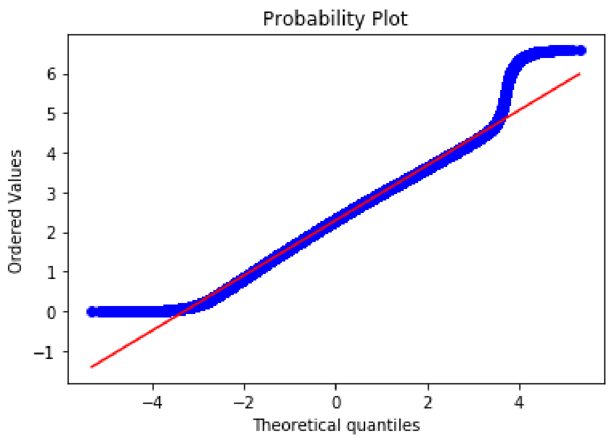


In [0]:

```
import scipy
```

In [0]:

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

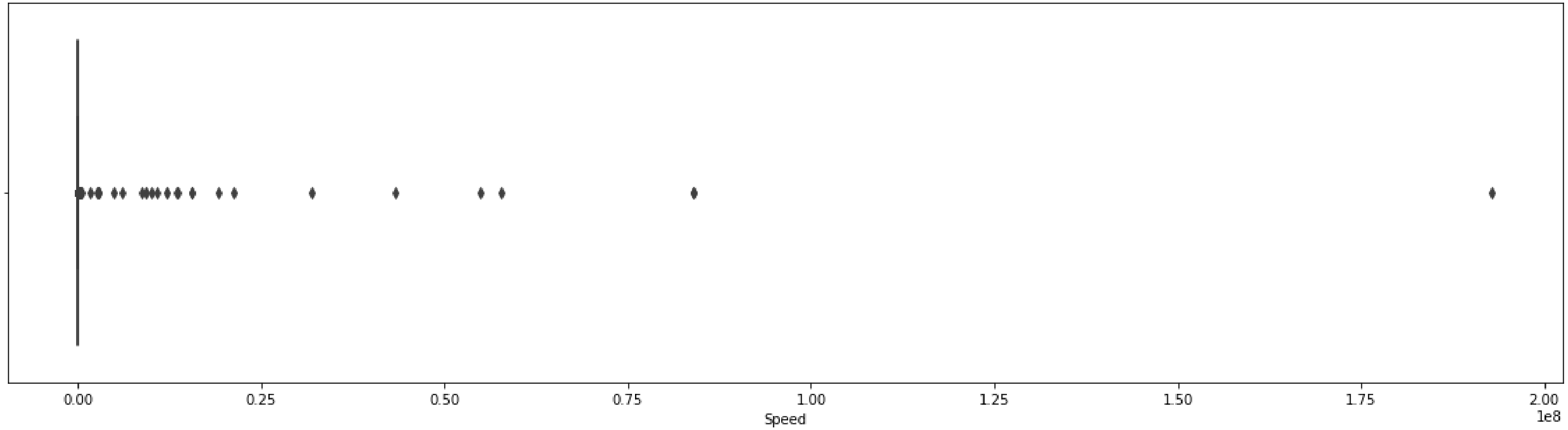


4. Speed

In [0]:

```
# check for any outliers in the data after trip duration outliers removed
# box-plot for speeds with outliers
frame_with_durations_modified['Speed'] = 60*(frame_with_durations_modified['trip_distance']/frame_with_durations_modified['trip_times'])
plt.figure(figsize=(20,5))
sns.boxplot(y="Speed", data =frame_with_durations_modified, orient='h')
plt.show()
```

```
plt.show()
```



In [0]:

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

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

In [0]:

```
#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])
```

```
print("100 percentile value is ",var[-1])
```

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

In [0]:

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

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

In [0]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0) & (frame_with_durations.Speed<45.31)]
```

In [0]:

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

Out[0]:

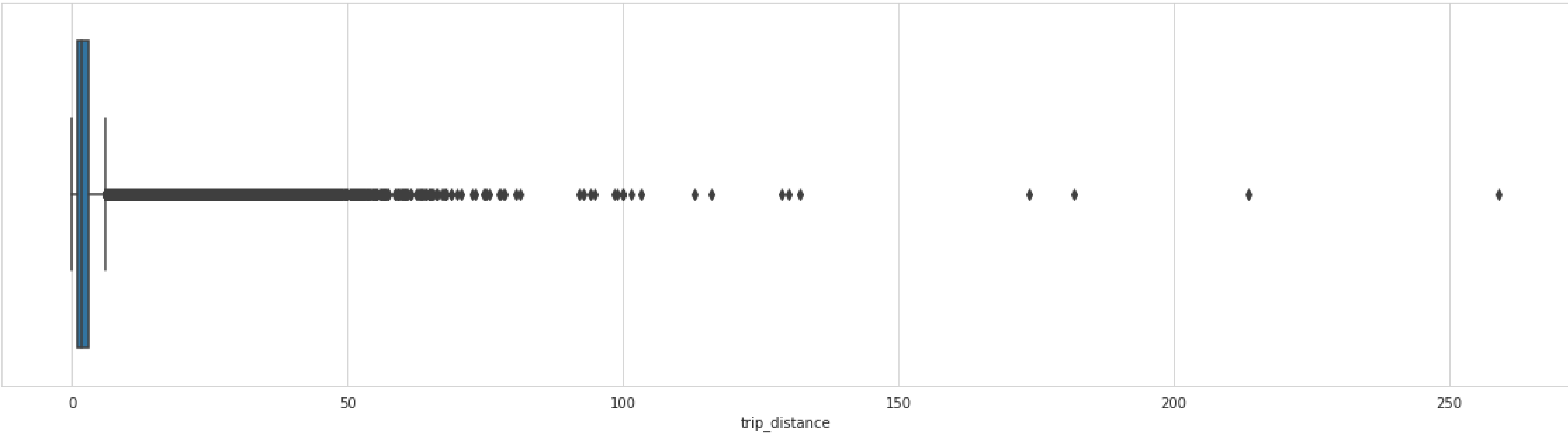
```
12.450173996027528
```

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

4. Trip Distance

In [0]:

```
# 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
plt.figure(figsize=(20,5))
sns.boxplot(y="trip_distance", data =frame_with_durations_modified, orient='h')
plt.show()
```



In [0]:

```
#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
```

```
50 percentile value is 1.69
60 percentile value is 2.07
70 percentile value is 2.6
80 percentile value is 3.6
90 percentile value is 5.97
100 percentile value is  258.9
```

In [0]:

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

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

In [0]:

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

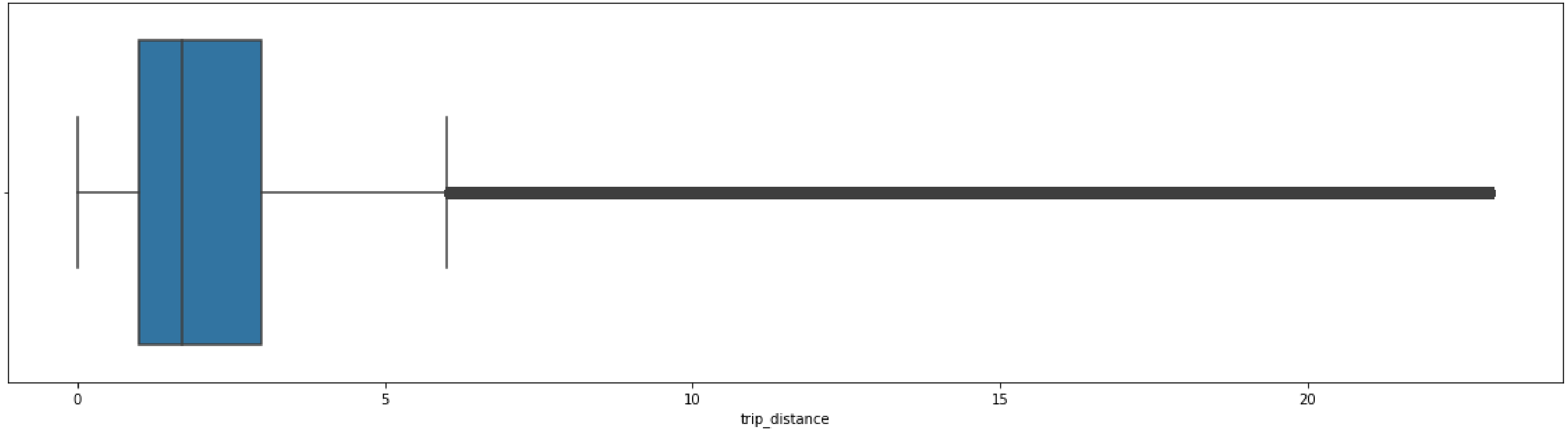
```
99.0 percentile value is 18.17
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.83
99.4 percentile value is 19.13
99.5 percentile value is 19.5
99.6 percentile value is 19.96
99.7 percentile value is 20.5
99.8 percentile value is 21.22
99.9 percentile value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

In [0]:

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

In [0]:

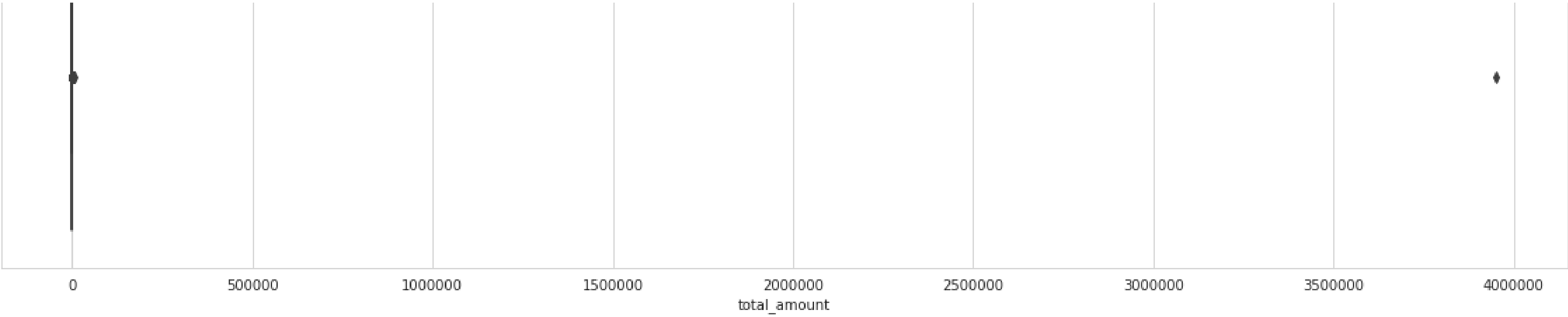
```
#box-plot after removal of outliers
plt.figure(figsize=(20,5))
sns.boxplot(y="trip_distance", data = frame_with_durations_modified, orient='h')
plt.show()
```



5. Total Fare

In [0]:

```
# up to now we have removed the outliers based on trip durations, cab speeds, and trip distances
# lets try if there are any outliers in based on the total_amount
# box-plot showing outliers in fare
plt.figure(figsize=(20,5))
sns.boxplot(y="total_amount", data =frame_with_durations_modified, orient='h')
plt.show()
```



In [0]:

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

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

In [0]:

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

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

```
95 percentile value is 38.53
96 percentile value is 42.6
97 percentile value is 48.13
98 percentile value is 58.13
99 percentile value is 66.13
100 percentile value is  3950611.6
```

In [0]:

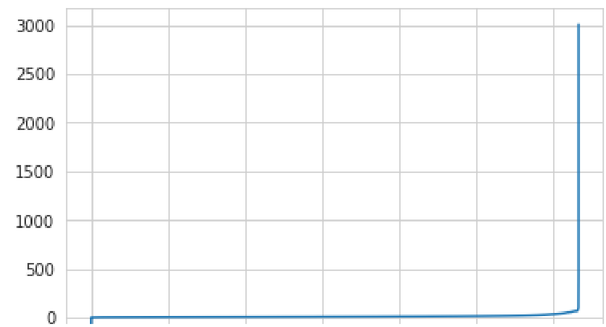
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

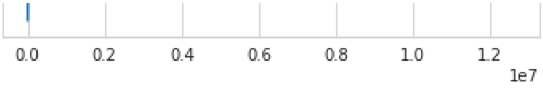
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value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

Observation:- As even the 99.9th percentile value doesnt look like an outlier,as there is not much difference between the 99.8th percentile and 99.9th percentile, we move on to do graphical analyis

In [0]:

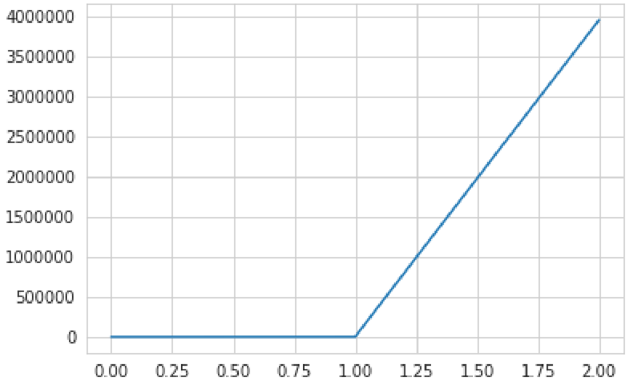
```
#below plot shows us the fare values(sorted) to find a sharp increase to remove those values as outliers
# plot the fare amount excluding last two values in sorted data
plt.plot(var[:-2])
plt.show()
```





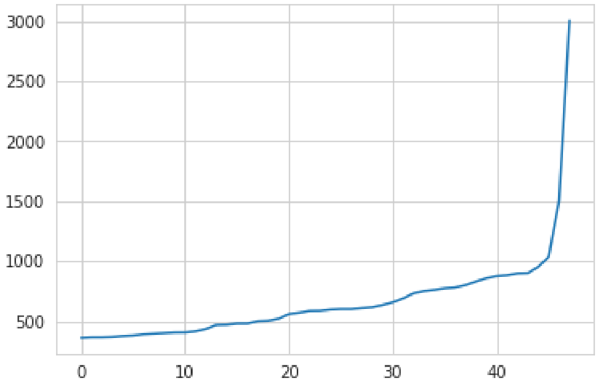
In [0]:

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



In [0]:

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



Remove all outliers/erronous points.

In [0]:

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


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


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


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

    temp_frame = new_frame[(new_frame.Speed <= 45.31) & (new_frame.Speed >= 0)]
    e = temp_frame.shape[0]
    print ("Number of outliers from speed analysis:",(a-e))

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


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

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

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

In [0]:

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

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12748986
Number of outlier coordinates lying outside NY boundaries: 293919
Number of outliers from trip times analysis: 23889
Number of outliers from trip distance analysis: 92597
Number of outliers from speed analysis: 36690
Number of outliers from fare analysis: 5275
Total outliers removed 377910
---
fraction of data points that remain after removing outliers 0.9703576425607495
```

Data-preperation

Clustering/Segmentation

In [0]:

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

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

```
        less2.append(nice_points)
        more2.append(wrong_points)
#     neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,\
           "\nAvg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):", np.ceil(sum(less2)/len(less2)),\
           "\nAvg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):", np.ceil(sum(more2)/len(more2)),\
           "\nMin inter-cluster distance = ",min_dist,"\n---")

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000).fit(coords)
#     frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude',
'pickup_longitude']])
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len
```

In [0]:

```
# 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 =  1.075305864558758
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 15.0
Min inter-cluster distance =  0.5106919257768725
---
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 =  0.45186320297366733
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 10.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 30.0
Min inter-cluster distance =  0.4171071263921082
---
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 =  0.38257408102024554
---
On choosing a cluster size of  60
```

```
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 15.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 45.0
Min inter-cluster distance =  0.27336589578272946
---
On choosing a cluster size of  70
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): 49.0
Min inter-cluster distance =  0.19351464665794352
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 25.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 55.0
Min inter-cluster distance =  0.18341410437555683
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 25.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 65.0
Min inter-cluster distance =  0.12881210269210389
---
```

Inference:

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

In [0]:

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

# Getting 40 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=30, batch_size=10000).fit(coords)
```

In [0]:

```
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude
']])
```

Plotting the cluster centers:

In [0]:

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

```
map_osm
```

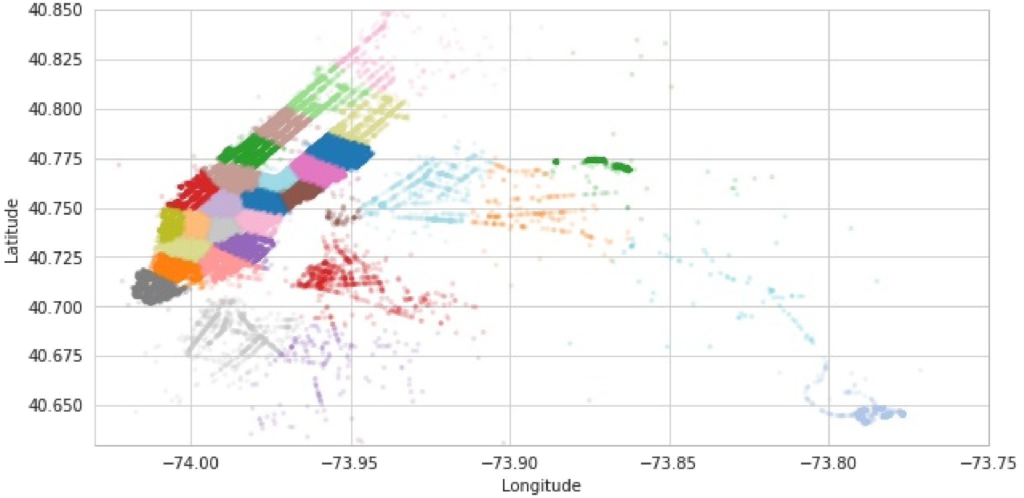
Out[0]:

Plotting the clusters:

In [0]:

In [0]:

```
#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, figsize=(10,5))
    ax.scatter(frame.pickup_longitude.values[:100000], frame.pickup_latitude.values[:100000], s=10, lw=0,
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.2)
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [0]:

```
#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
```

```
# 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= frame['pickup_times'].values
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,1433116800],\
                  [1451606400,1454284800,1456790400,1459468800,1462060800,1464739200]]

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

In [0]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude
']])

jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed, 1, 2015)

jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cluster','pickup_bins']).count()
```

In [0]:

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

Out[0]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pickup_times	Speed	pickup_cluster	pickup_bi
0	1	1.59	-73.993896	40.750111	-73.974785	40.750618	17.05	18.050000	1.421349e+09	5.285319	5	2130
1	1	3.30	-74.001648	40.724243	-73.994415	40.759109	17.80	19.833333	1.420922e+09	9.983193	25	1419
2	1	1.80	-73.963341	40.802788	-73.951820	40.824413	10.80	10.050000	1.420922e+09	10.746269	9	1419
3	1	0.50	-74.009087	40.713818	-74.004326	40.719986	4.80	1.866667	1.420922e+09	16.071429	10	1419
4	1	3.00	-73.971176	40.762428	-74.004181	40.742653	16.30	19.316667	1.420922e+09	9.318378	0	1419

In [0]:

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

Out[0]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**0**	120
	1	230
	2	238
	3	203
	4	156

In [0]:

```
# 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_durations_outliers_removed[['pickup_latitude', 'pickup_longi
tude']])
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed_2016[['pickup_latitude', '
pickup_longitude']])

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_no,year_no)
```

```
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cluster','pickup_bins']).count()

    return final_updated_frame, final_groupby_frame

month_jan_2016 = dd.read_csv('gdrive/My Drive/CoLab/NYC Taxi Demand/yellow_tripdata_2016-01.csv')
# month_feb_2016 = dd.read_csv('gdrive/My Drive/CoLab/NYC Taxi Demand/yellow_tripdata_2016-02.csv')
# month_mar_2016 = dd.read_csv('gdrive/My Drive/CoLab/NYC Taxi Demand/yellow_tripdata_2016-03.csv')
```

In [0]:

```
jan_2016_frame, jan_2016_groupby = datapreparation(month_jan_2016,kmeans,1,2016)
```

```
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: 31018
Number of outliers from fare analysis: 4991
Total outliers removed 297784
---
Estimating clusters..
Final groupbying..
```

In [0]:

```
print('Amount of data retained is:', np.round(((10906858-297784)/10906858)*100.0,3),'%')
```

```
Amount of data retained is: 97.27 %
```

In [0]:

```
# feb_2016_frame, feb_2016_groupby = datapreparation(month_feb_2016,kmeans,2,2016)
jan_2016_frame.head(5)
```

Out[0]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pickup_times	Speed	pickup_cluster	pickup_bi
5	2	5.52	-73.980118	40.743050	-73.913490	40.763142	20.3	18.50	1.451606e+09	17.902703	15	0
6	2	7.45	-73.994057	40.719990	-73.966362	40.789871	27.3	26.75	1.451606e+09	16.710280	2	0
7	1	1.20	-73.979424	40.744614	-73.992035	40.753944	10.3	11.90	1.451606e+09	6.050420	15	0
8	1	6.00	-73.947151	40.791046	-73.920769	40.865578	19.3	11.20	1.451606e+09	32.142857	4	0
9	1	3.21	-73.998344	40.723896	-73.995850	40.688400	12.8	11.10	1.451606e+09	17.351351	2	0

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pickup_times	Speed	pickup_cluster	pickup_bi

In [0]:

```
# mar_2016_frame, mar_2016_groupby = datapreparation(month_mar_2016,kmeans,3,2016)
jan_2016_groupby.head(5)
```

Out[0]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**0**	108
	1	202
	2	182
	3	174
	4	152

In [0]:

```
jan_2016_frame.shape
```

Out[0]:

```
(10609074, 12)
```

Smoothing

In [0]:

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

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

In [0]:

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

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

# #feb
# feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

# #march
# mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [0]:

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

```
for the  0 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  146
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  52
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  35
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  23
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  221
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  115
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  33
```

```
----------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  35
----------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  48
----------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  25
----------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  609
----------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  28
----------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  49
----------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  57
----------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  638
----------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  35
----------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  29
----------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  39
----------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  56
----------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  28
----------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  37
----------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  29
----------------------------------------------------------------
```

there are two ways to fill up these values

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

In [0]:

```
# Fills a value of zero for every bin where no pickup data is present
# the count values: number pickps that are happened in each region for each 10min intravel
```

```
# 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(30):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [0]:

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

# for every 10min interval(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the above markdown cell)
# we finally return smoothed data
def smoothing(count_values, values):
    ind = 0
    repeat = 0
    smoothed_region = []
    for r in range(0, 30):
        smoothed_bin = []
        for t1 in range(4464):
            if repeat != 0:   #This will ensure that we shall not fill the pickup values again which we already filled by smoothing
                repeat -= 1
            else:
                if t1 in values[r]:
                    smoothed_bin.append(count_values[ind])
                    ind += 1
                else:
                    if t1 == 0:
                  #CASE-1:Pickups missing in the beginning
                        for t2 in range(t1, 4464):
                            if t2 not in values[r]:
                                continue
```

```
                    else:
                        right_hand_limit = t2
                        smoothed_value = (count_values[ind]*1.0)/((right_hand_limit + 1)*1.0)
                        for i in range(right_hand_limit + 1):
                            smoothed_bin.append(math.ceil(smoothed_value))
                        ind += 1
                        repeat = right_hand_limit - t1

                if t1 != 0:
                    right_hand_limit = 0
                    for t2 in range(t1, 4464):
                        if t2 not in values[r]:
                            continue
                        else:
                            right_hand_limit = t2
                            break
                    if right_hand_limit == 0:
                #CASE-2: Pickups missing in the end
                        smoothed_value = (count_values[ind-1]*1.0)/(((4464 - t1)+1)*1.0)
                        del smoothed_bin[-1]
                        for i in range((4464 - t1)+1):
                            smoothed_bin.append(math.ceil(smoothed_value))
                        repeat = (4464 - t1) - 1
                #CASE-3: Pickups missing in middle of two values
                    else:
                        smoothed_value = ((count_values[ind-1] + count_values[ind])*1.0)/(((right_hand_limit - t1)+2)*1.0)
                        del smoothed_bin[-1]
                        for i in range((right_hand_limit - t1)+2):
                            smoothed_bin.append(math.ceil(smoothed_value))
                        ind += 1
                        repeat = right_hand_limit - t1
        smoothed_region.extend(smoothed_bin)
    return smoothed_region
```

In [0]:

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

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

In [0]:

```
def getNoOfZeros(values):
    return np.nonzero(np.array(values) == 0)[0].size
```

In [0]:

```
print("number of 10min intravels with zero pickups in filled missing data:",getNoOfZeros(jan_2015_fill))
```

number of 10min intravels with zero pickups in filled missing data: 2587

In [0]:

```
print("number of 10min intravels with zero pickups in smoothed data:",getNoOfZeros(jan_2015_smooth))
```

number of 10min intravels with zero pickups in smoothed data: 0

In [0]:

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

number of 10min intravels among all the clusters 133920

In [0]:

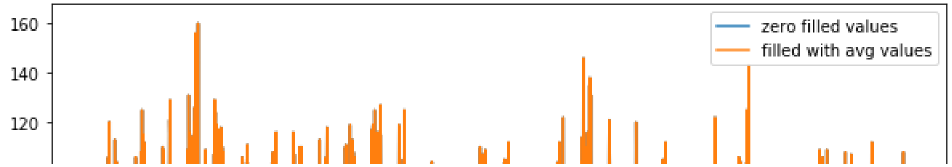
```
4464-8920
```

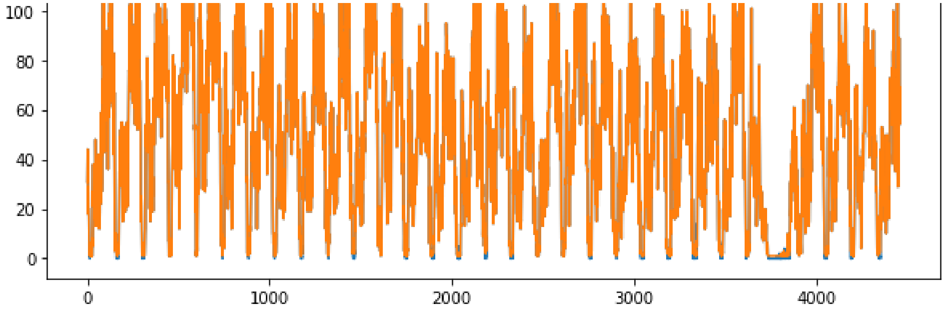
Out[0]:

-4456

In [0]:

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





In [0]:

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

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

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

In [0]:

```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled with zero

# jan_2016_fill = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)

jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)
```

In [0]:

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

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

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

```
# that are happened for three months in 2016 data
for i in tqdm(range(30)):
  regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)])
```

```
100%|██████████| 30/30 [00:00<00:00, 4256.16it/s]
```

In [0]:

```
print(len(regions_cum))

print(len(regions_cum[0]))
```

```
30
4464
```

In [0]:

```
import scipy.fftpack as fftpack
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html
from scipy.signal import find_peaks
from scipy import stats
```

In [0]:

```
fft_features = []

# get real part of fft : https://stackoverflow.com/a/29545236

jan_freq = fftpack.rfftfreq(4464)[1:10:2]

for i in tqdm(range(0,30)):
  #January
  pick_ups = jan_2016_smooth[4464*i:4464*(i+1)]

  jan_fft = fftpack.rfft(pick_ups)[1:6]

  for _ in range(4464-5):

    fft_features.append( np.append(jan_fft, jan_freq) )
```

```
100%|██████████| 30/30 [00:00<00:00, 55.65it/s]
```

In [0]:

```
fft_features = np.array(fft_features)
fft_features.shape
```

Out[0]:

```
(133770, 10)
```

In [0]:

```
fft_features[:5][0]
```

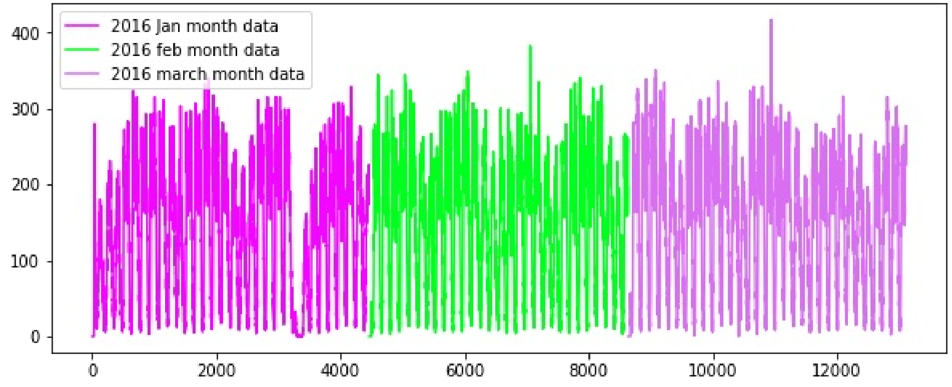
Out[0]:

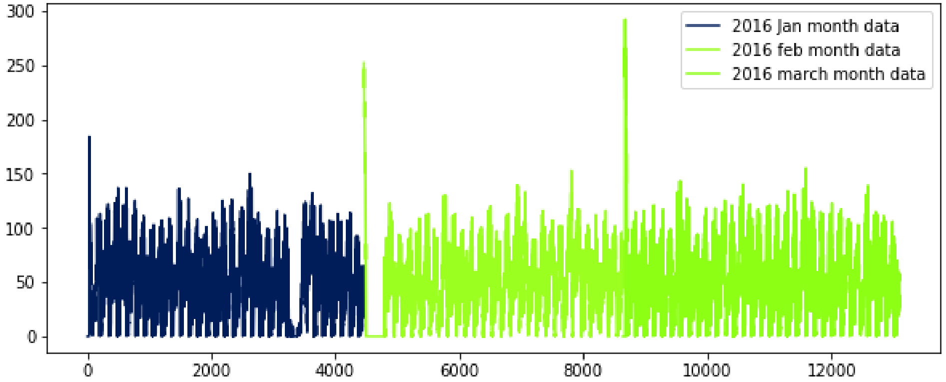
```
array([-2.72649078e+04, -1.69148474e+04,  1.14583045e+04,  7.43482986e+03,
       -8.65243959e+03,  2.24014337e-04,  4.48028674e-04,  6.72043011e-04,
        8.96057348e-04,  1.12007168e-03])
```

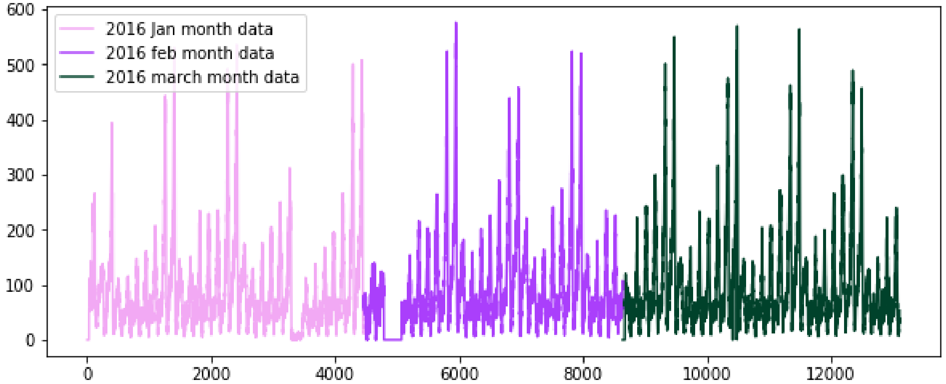
Time series and Fourier Transforms

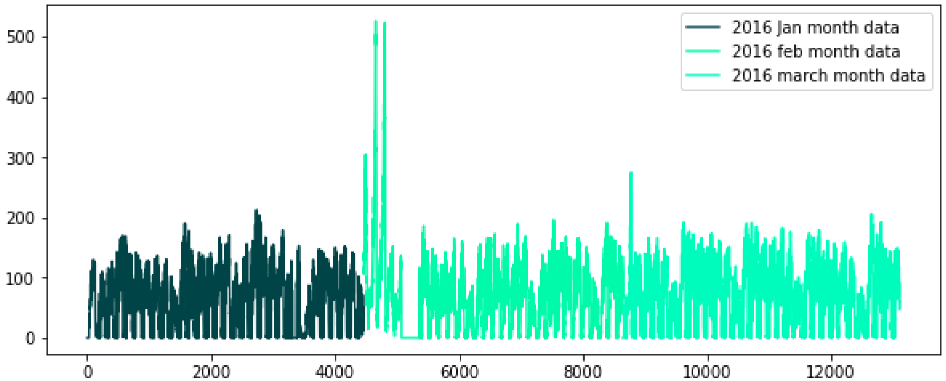
In [0]:

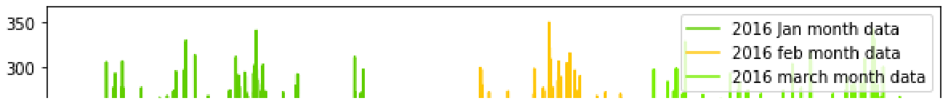
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40):
    plt.figure(figsize=(10,4))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

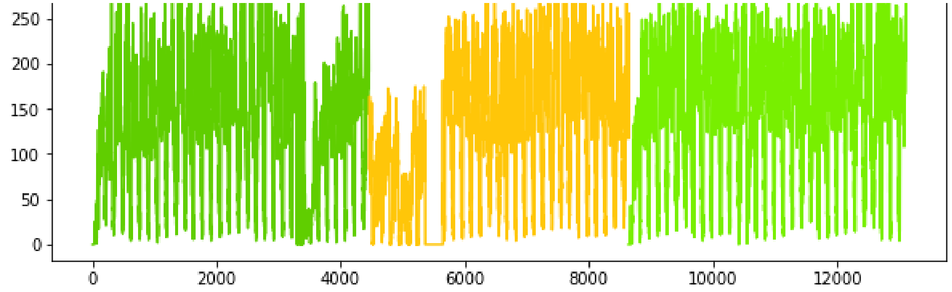


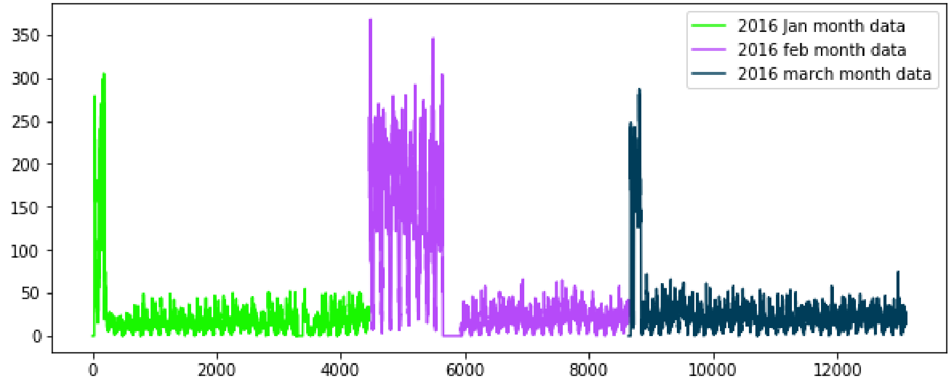

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

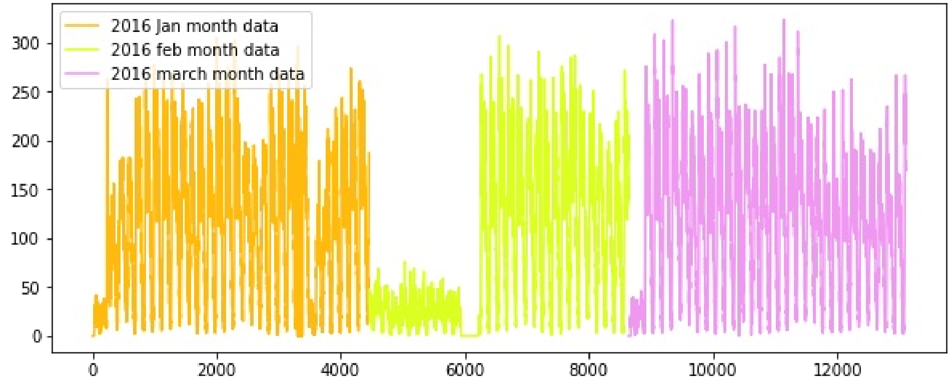

2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
500
600
0
2000
4000
6000
8000
10000
12000

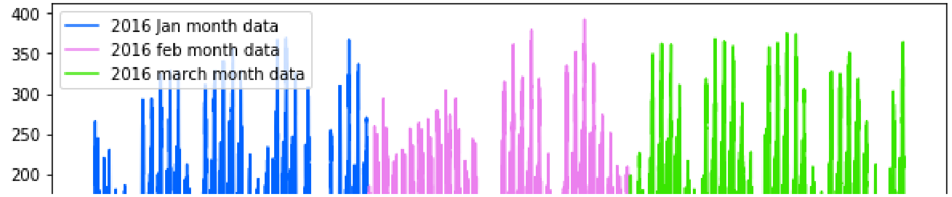

2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000

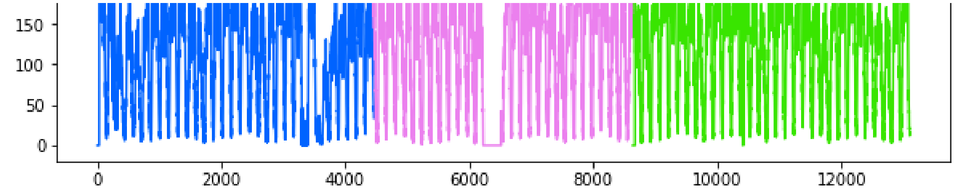

2016 Jan month data
2016 feb month data
2016 march month data
300
350

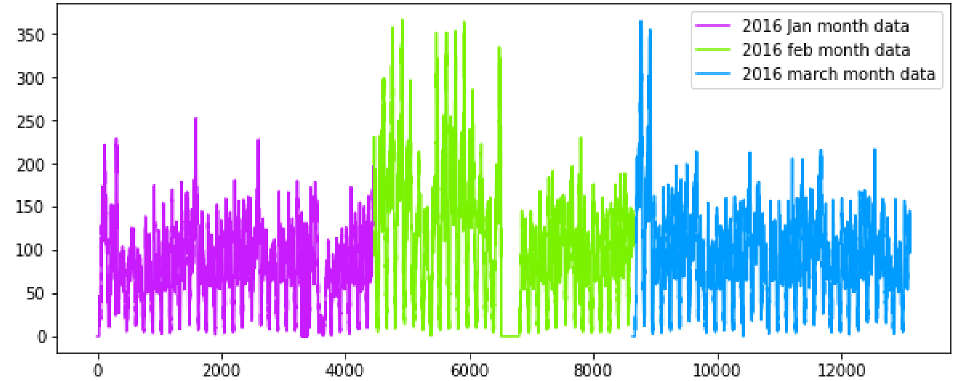


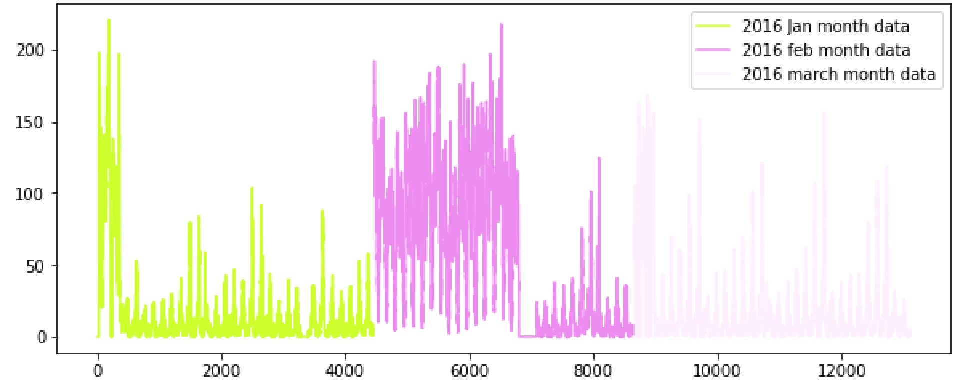

2016 Jan month data
2016 feb month data
2016 march month data

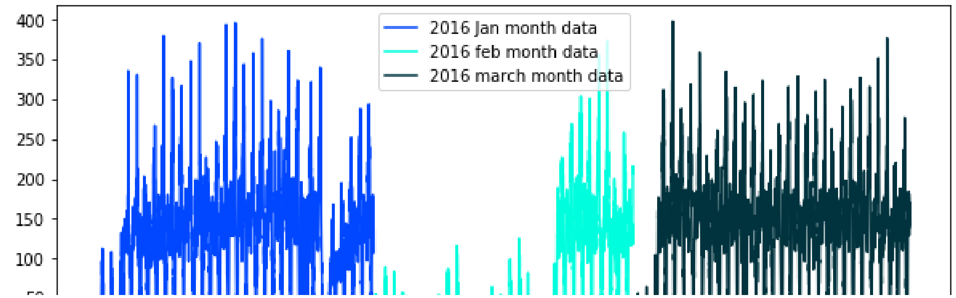

2016 Jan month data
2016 feb month data
2016 march month data

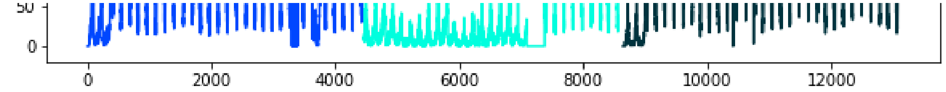

2016 Jan month data
2016 feb month data
2016 march month data

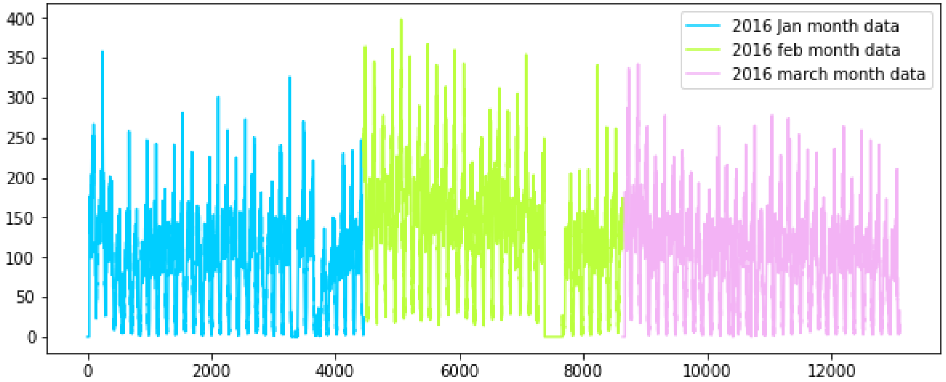

0
50
100
150
0
2000
4000
6000
8000
10000
12000

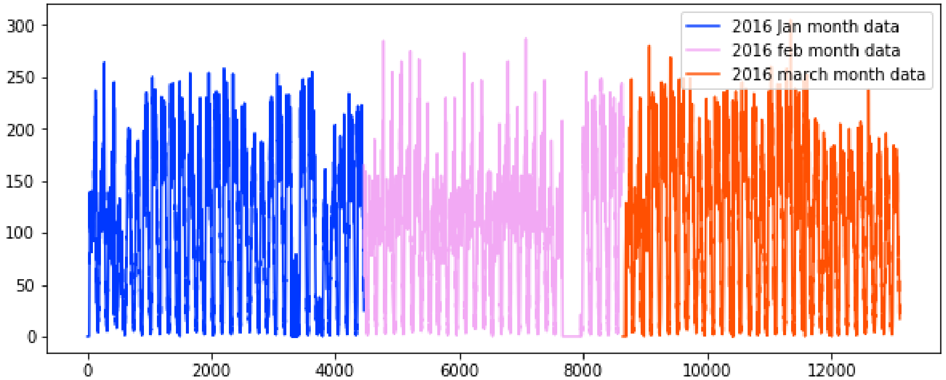

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

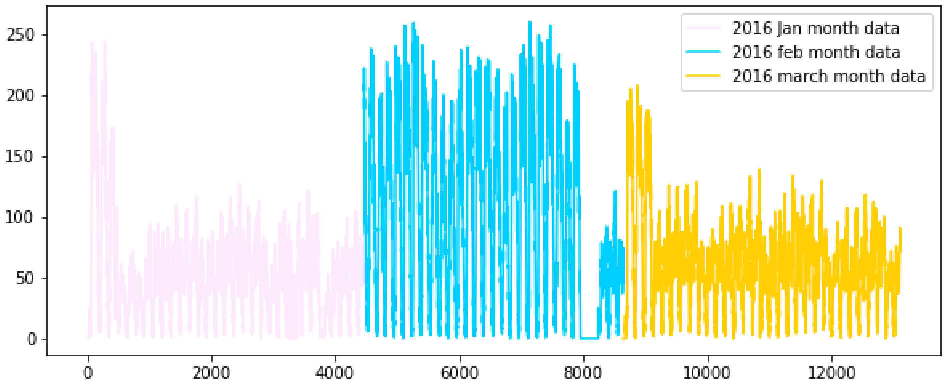

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

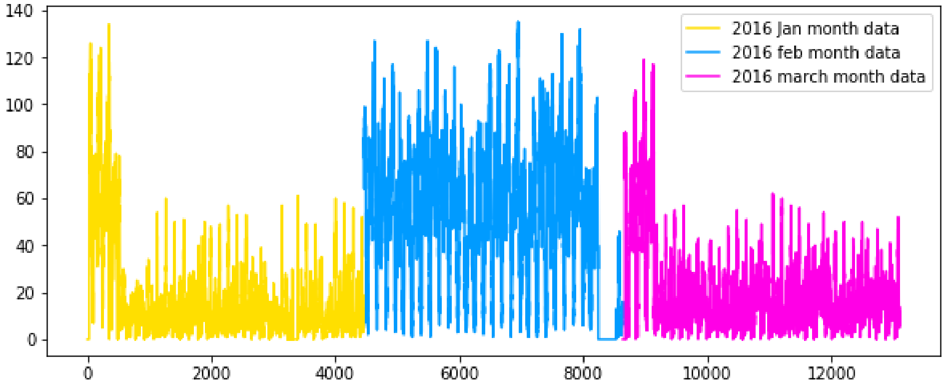

2016 Jan month data
2016 feb month data
2016 march month data
100
150
200
250
300
350
400

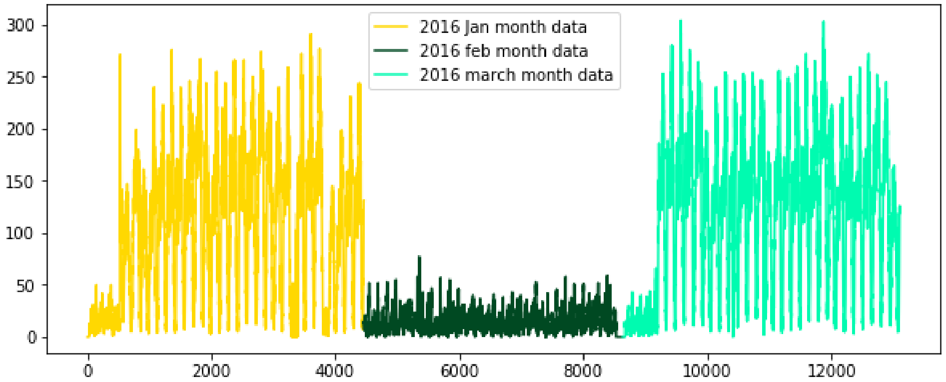

50
0
0
2000
4000
6000
8000
10000
12000

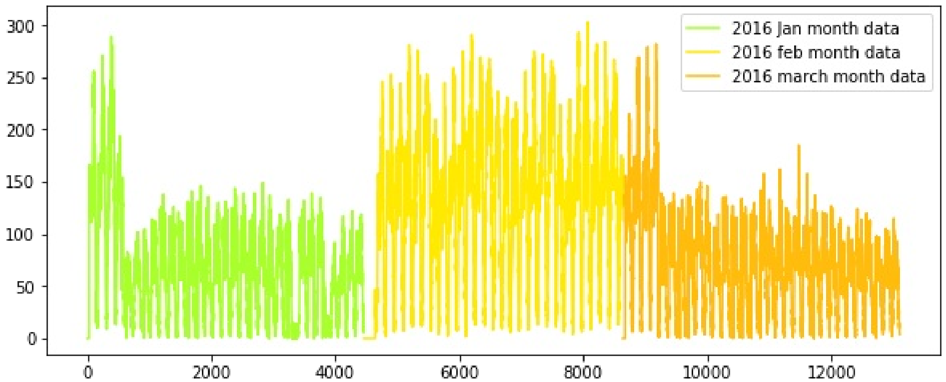

2016 Jan month data
2016 feb month data
2016 march month data
400
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000


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

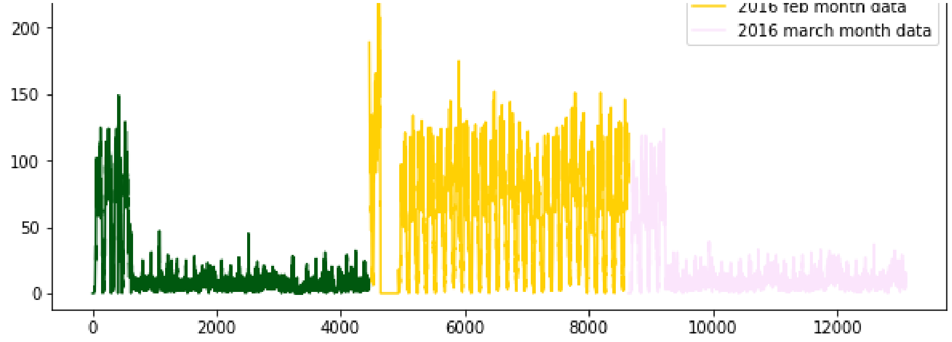

2016 Jan month data
2016 feb month data
2016 march month data
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

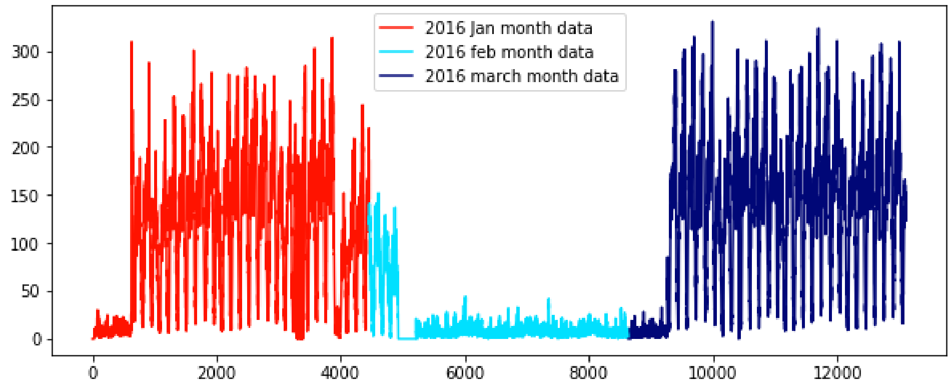

2016 Jan month data
2016 feb month data
2016 march month data

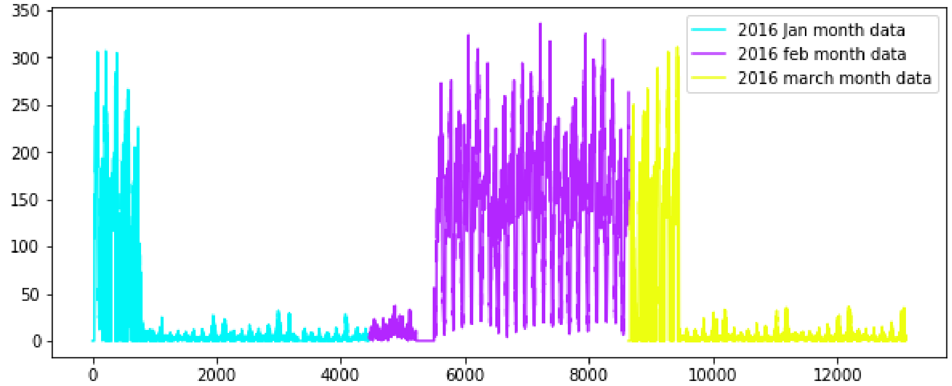

2016 Jan month data
2016 feb month data
2016 march month data

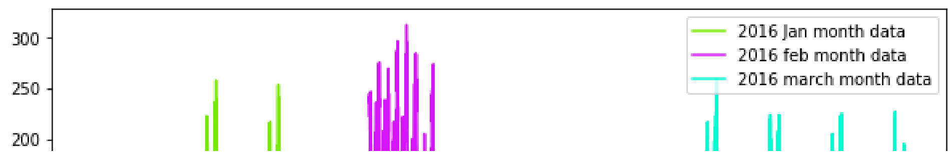

2016 Jan month data
2016 feb month data
2016 march month data

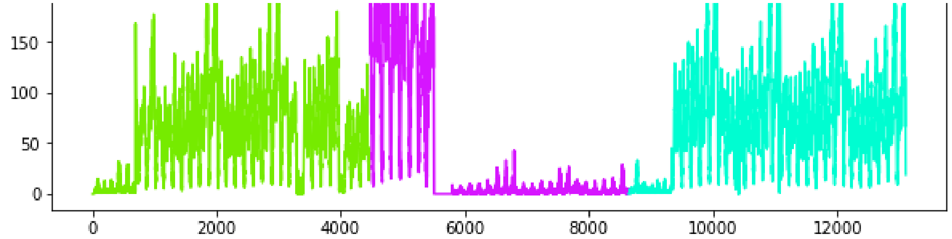

2016 Jan month data

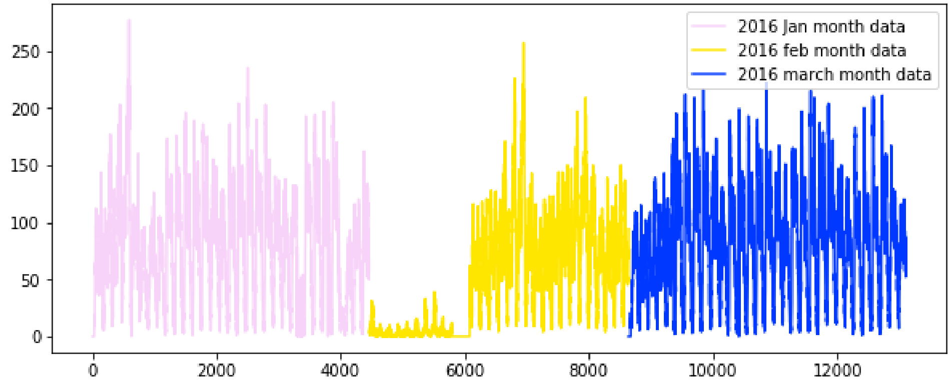

2016 feb month data
2016 march month data

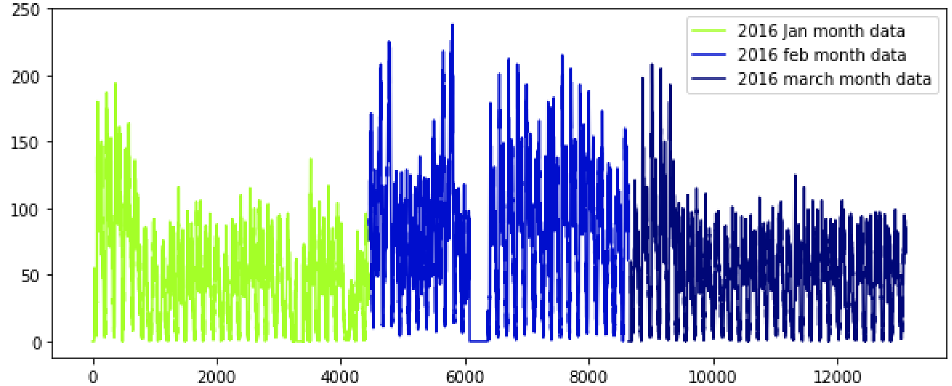

2016 Jan month data
2016 feb month data
2016 march month data

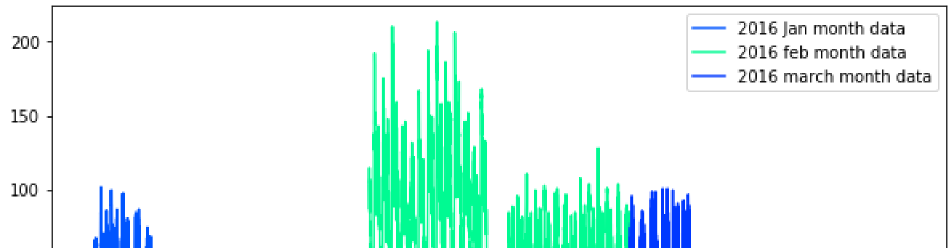

2016 Jan month data
2016 feb month data
2016 march month data

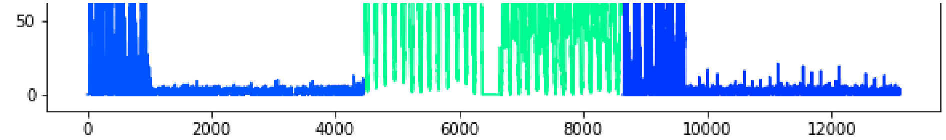

2016 Jan month data
2016 feb month data
2016 march month data

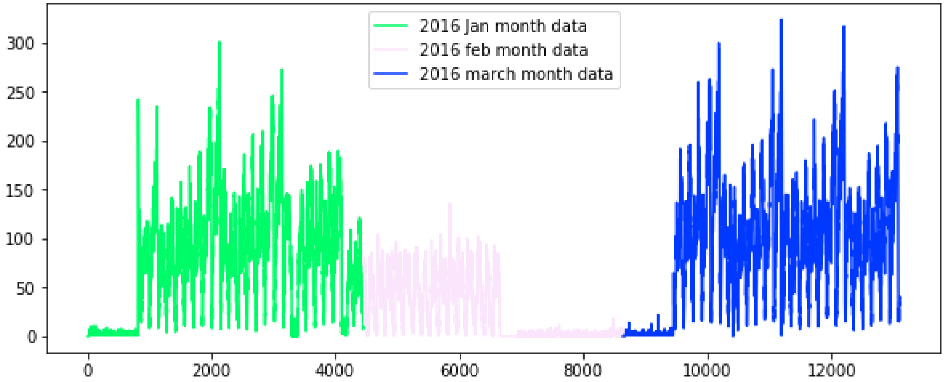


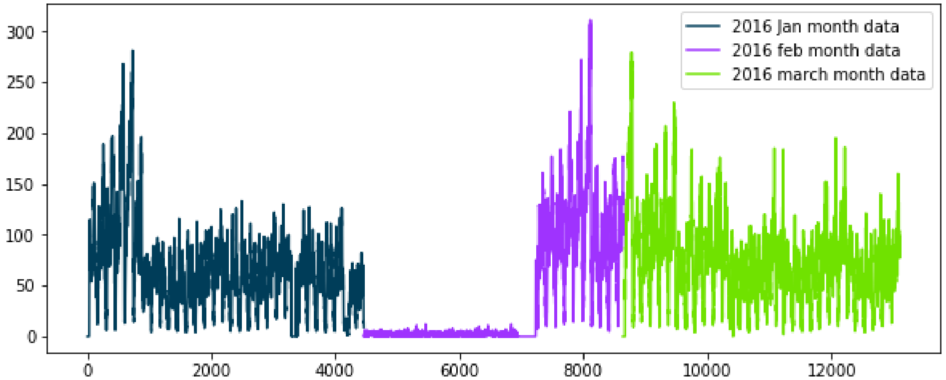

2016 Jan month data
2016 feb month data
2016 march month data

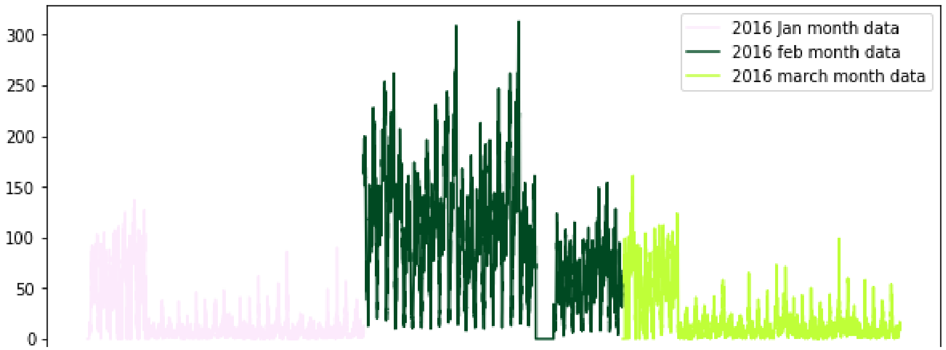

2016 Jan month data
2016 feb month data
2016 march month data


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

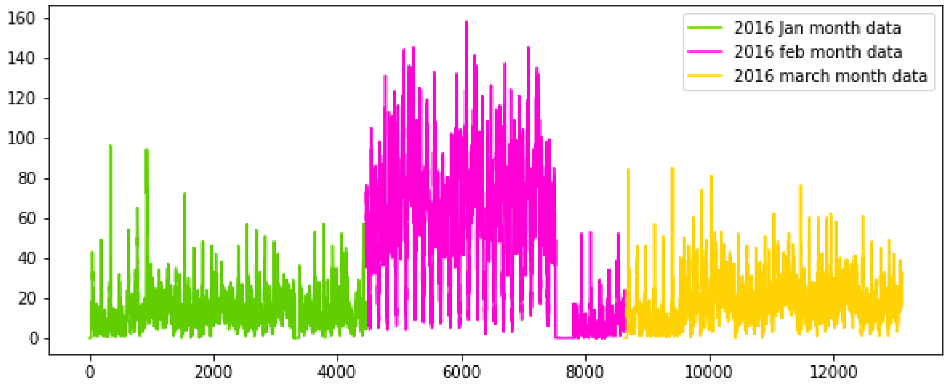


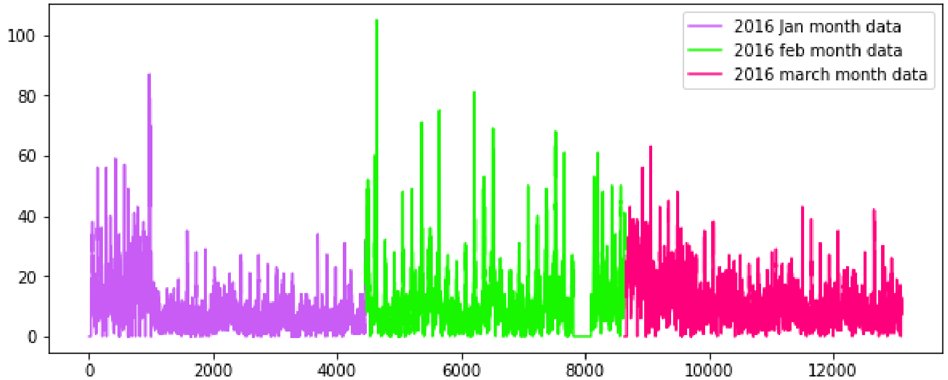

2016 Jan month data
2016 feb month data
2016 march month data

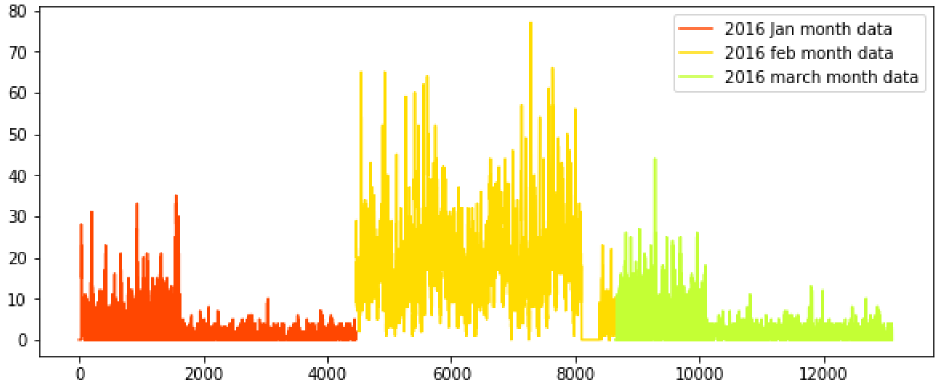

2016 Jan month data
2016 feb month data
2016 march month data

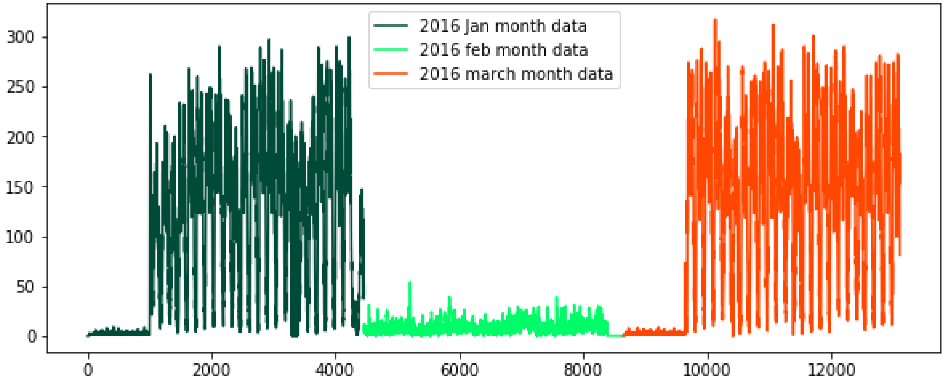

2016 Jan month data
2016 feb month data
2016 march month data

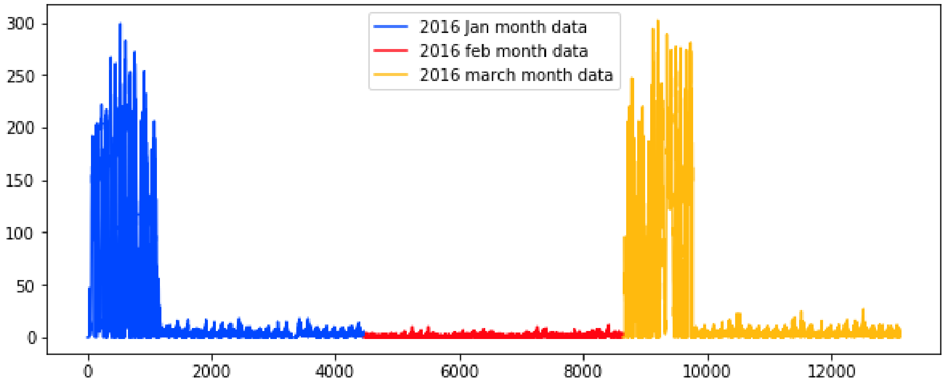

0
2000
4000
6000
8000
10000
12000

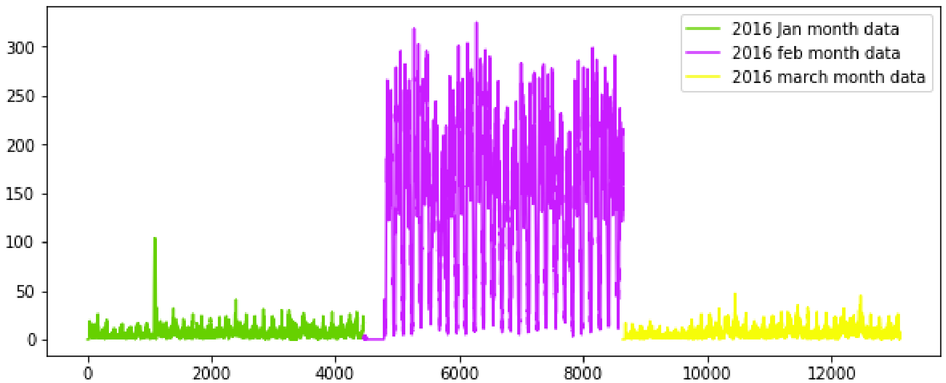

2016 Jan month data
2016 feb month data
2016 march month data
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000
10000
12000


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

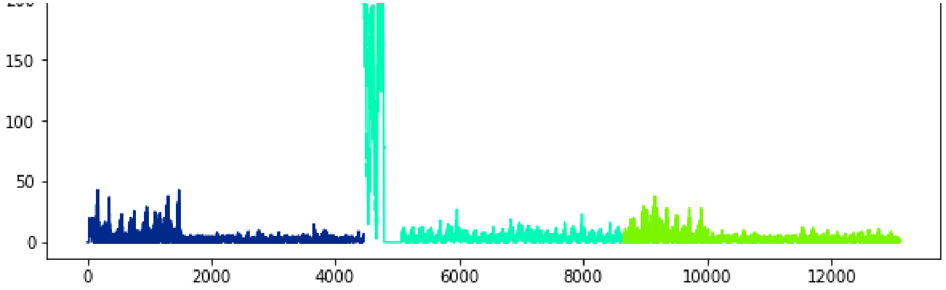

2016 Jan month data
2016 feb month data
2016 march month data
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000
12000

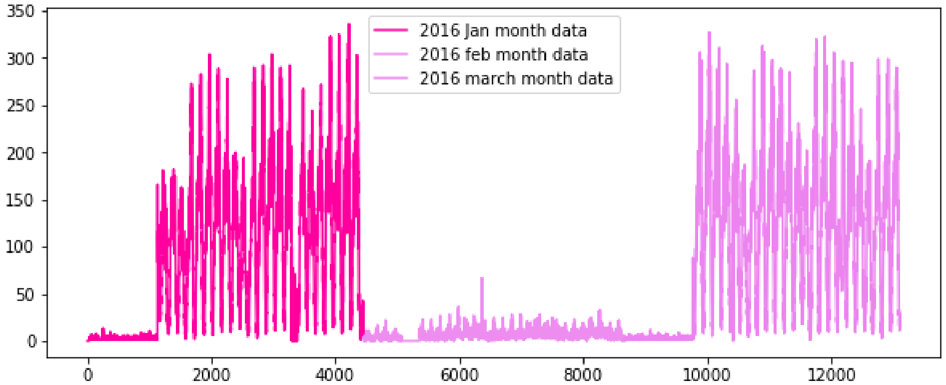

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

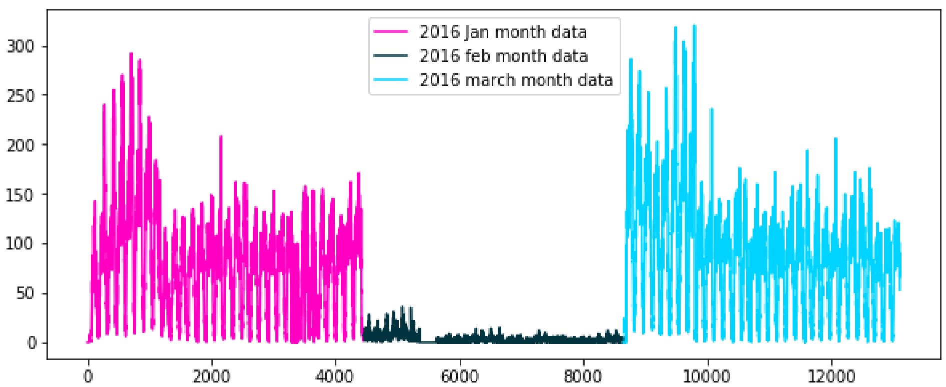

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

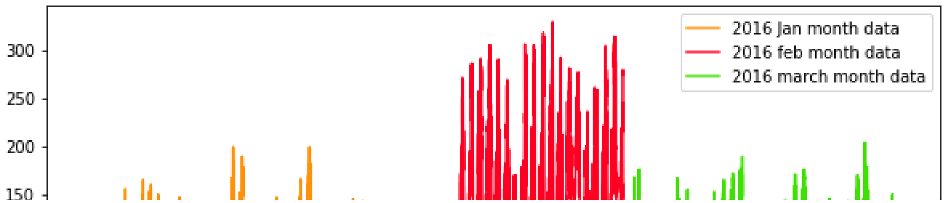

2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

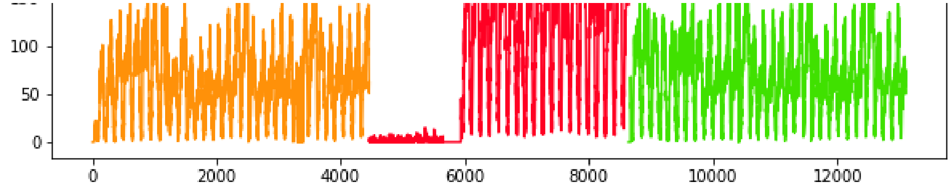

2016 Jan month data
2016 feb month data
2016 march month data
250
200


150
100
50
0
0
2000
4000
6000
8000
10000
12000

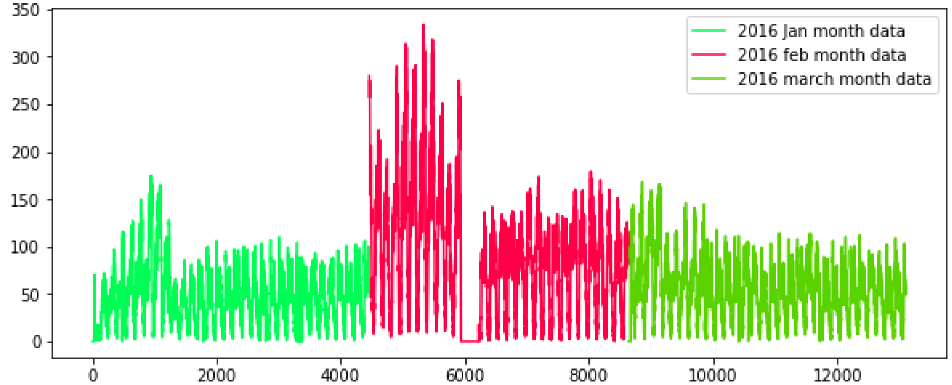

2016 Jan month data
2016 feb month data
2016 march month data
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000


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

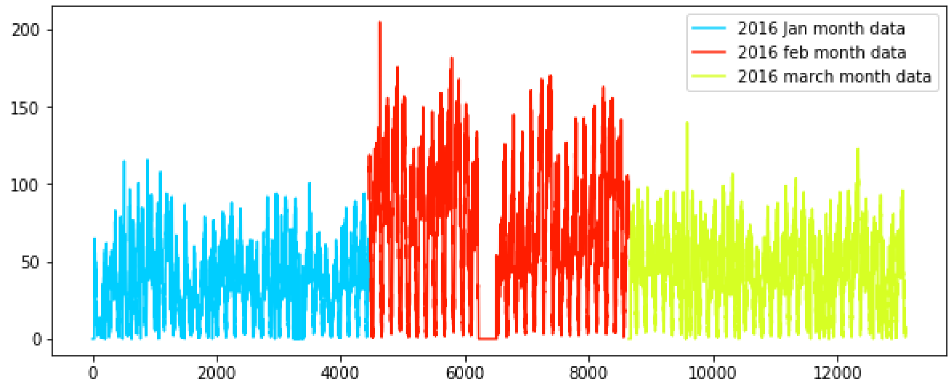

2016 Jan month data
2016 feb month data
2016 march month data
300
250
200
150



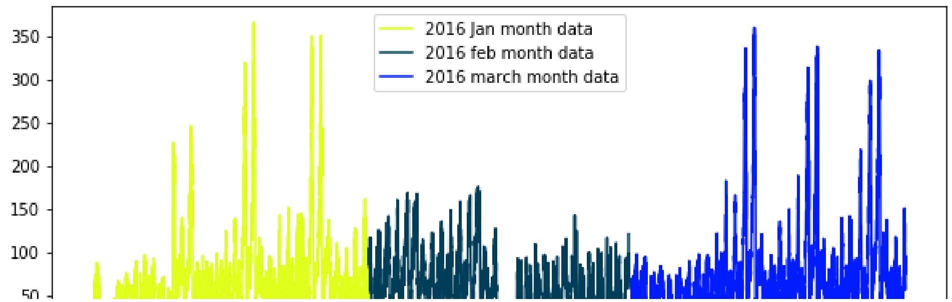
100
50
0
0
2000
4000
6000
8000
10000
12000



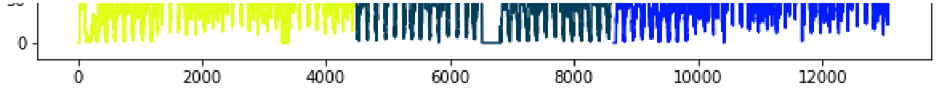
350
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data



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

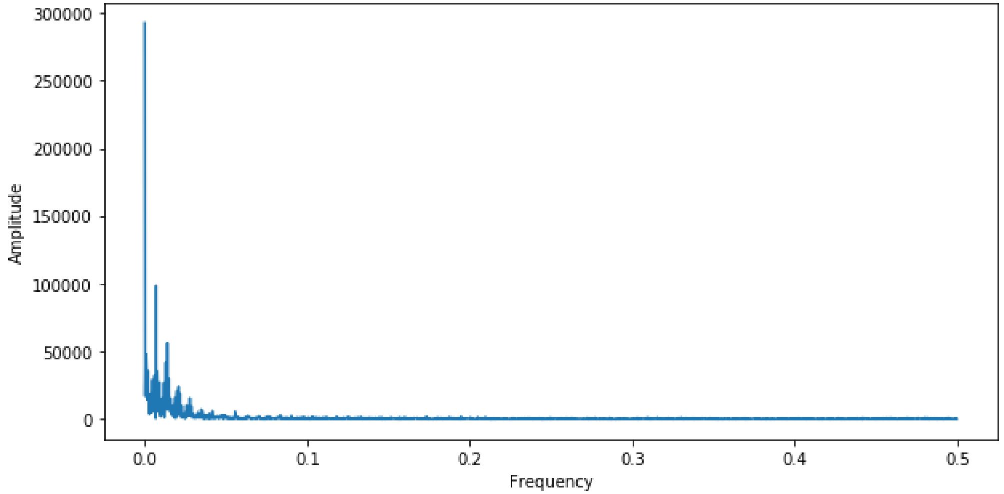


350
300
250
200
150
100
2016 Jan month data
2016 feb month data
2016 march month data



In [0]:

```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html
freq = np.fft.fftfreq(4460, 1)
n = len(freq)
plt.figure(figsize=(10,5))
plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



In [0]:

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

Modelling: Baseline Models

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

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

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

In [0]:

```
def MA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    error=[]
    predicted_values=[]
    window_size=3
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])/window_size
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


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

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

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

In [0]:

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

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

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

Weighted Moving Averages

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

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

In [0]:

```python
def WA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
```

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

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

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

Weighted Moving Averages using Previous 2016 Values - $P_t = (N * P_{t-1} + (N-1) * P_{t-2} + (N-2) * P_{t-3}. \dots 1 * P_{t-n})/(N * (N+1)/2)$

In [0]:

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

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

```
            sum_of_coeff+=j
        predicted_value=int(sum_values/sum_of_coeff)

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

For the above the Hyperparameter is the window-size (n) which is tuned manually and it is found that the window-size of 2 is optimal for getting the best results using Weighted Moving Averages using previous 2016 values therefore we get $P_t = (2 * P_{t-1} + P_{t-2})/3$

Exponential Weighted Moving Averages

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

In exponential moving averages we use a single hyperparameter alpha (α) which is a value between 0 & 1 and based on the value of the hyperparameter alpha the weights and the window sizes are configured.
For eg. If $\alpha = 0.9$ then the number of days on which the value of the current iteration is based is~ $1/(1 - \alpha) = 10$ i.e. we consider values 10 days prior before we predict the value for the current iteration. Also the weights are assigned using $2/(N + 1) = 0.18$,where N = number of prior values being considered, hence from this it is implied that the first or latest value is assigned a weight of 0.18 which keeps exponentially decreasing for the subsequent values.

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

In [0]:

```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Prediction'].values)[i],1))))
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].values)[i])

    ratios['EA_R1_Predicted'] = predicted_values
```

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

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

In [0]:

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

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

In [0]:

```
mean_err=[0]*10
median_err=[0]*10
ratios_jan,mean_err[0],median_err[0]=MA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[1],median_err[1]=MA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[2],median_err[2]=WA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[3],median_err[3]=WA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[4],median_err[4]=EA_R1_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[5],median_err[5]=EA_P1_Predictions(ratios_jan,'jan')
```

Comparison between baseline models

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

In [0]:

In [0]:

```
from prettytable import PrettyTable

pt_base_models = PrettyTable()
pt_base_models.field_names = ['Model', 'MAPE', 'MSE']

pt_no_fft = PrettyTable()
pt_no_fft.field_names = ['Model', 'Train MAPE', 'Test MAPE']

pt_fft_other = PrettyTable()
pt_fft_other.field_names = ['Model', 'Train MAPE', 'Test MAPE']
```

In [0]:

```
pt_base_models.add_row(['Moving Averages(Ratios)\nMoving Averages(2016 Values)', str(mean_err[0])+'\n'+str(mean_err[1]), str(median_err[0])+'\n'+str(median_err[1])])
pt_base_models.add_row(['','',''])
pt_base_models.add_row(['Weighted Moving Averages(Ratios)\nWeighted Moving Averages(2016 Values)', str(mean_err[2])+'\n'+str(mean_err[3]), str(median_err[2])+'\n'+str(median_err[3])])
pt_base_models.add_row(['','',''])
pt_base_models.add_row(['Exponential Moving Averages(Ratios)\nExponential Moving Averages(2016 Values)', str(mean_err[4])+'\n'+str(mean_err[5]), str(median_err[4])+'\n'+str(median_err[5])])
```

In [0]:

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------------------------------
Moving Averages (Ratios) -                             MAPE:  0.1632303629892675       MSE:  538.6893891875746
Moving Averages (2016 Values) -                        MAPE:  0.1270124989230917        MSE:  236.3680630227001
--------------------------------------------------------------------------------------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.16046320348034146       MSE:  525.8311379928315
Weighted Moving Averages (2016 Values) -               MAPE:  0.1216380430563497       MSE:  223.38190710872163
--------------------------------------------------------------------------------------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.15992950939921804       MSE:  516.9143667861409
Exponential Moving Averages (2016 Values) -         MAPE:  0.1214696023422968       MSE:  222.33956093189965
```

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

From the above matrix it is inferred that the best forecasting model for our prediction would be:- $P_t^{'} = \alpha * P_{t-1} + (1-\alpha) * P_{t-1}^{'}$ i.e Exponential Moving Averages using 2016 Values

Regression Models

Train-Test Split

Before we start predictions using the tree based regression models we take January of 2016 pickup data and split it such that for every region we have 80% data in train and 20% in test, ordered date-wise for every region

In [0]:

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

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

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

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


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


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

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

```
# 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 [f1,f2,f3,f4,f5] fi=number of pickups happened in i+1th 10min intravel(bin)
# the second row will have [f1,f2,f3,f4,f5]
# the third row will have [f2,f3,f4,f5,f6]
# and so on...
tsne_feature = []


tsne_feature = [0]*number_of_time_stamps
for i in range(0,30):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*4459)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*4459)
    # jan 1st 2016 is Friday, so we start our day from 5: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of pickups that are happened in last 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+5)%7) for k in range(5,4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for r in range(0,len(regions_cum[i])-number_of_time_stamps)]))

    output.append(regions_cum[i][5:])

tsne_feature = tsne_feature[1:]
```

In [0]:

```
mean_med_peaks_feats = [0]*3

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

    cum = []

    for r in range(0,len(regions_cum[i])-number_of_time_stamps):

      picks = regions_cum[i][r:r+number_of_time_stamps]

      cum.append([np.mean(picks), np.median(picks), find_peaks(picks)[0].size])

    mean_med_peaks_feats = np.vstack((mean_med_peaks_feats, cum))


mean_med_peaks_feats = mean_med_peaks_feats[1:]
```

```
100%|██████████| 30/30 [00:08<00:00,  3.77it/s]
```

In [0]:

```
133770/30
```

Out[0]:

```
4459.0
```

In [0]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_weekday[0]) == 30*4459 == len(output)*len(output[0])
```

Out[0]:

```
True
```

In [0]:

```
tsne_feature[:5]
```

Out[0]:

```
array([[108, 202, 182, 174, 152],
       [202, 182, 174, 152, 178],
       [182, 174, 152, 178, 172],
       [174, 152, 178, 172, 156],
       [152, 178, 172, 156, 130]])
```

DataPrep

In [0]:

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from datetime import datetime
```

In [0]:

```
holt_exp = []

for r in range(0,30):
  start = datetime.now()

  predicted_value = ExponentialSmoothing(regions_cum[r], seasonal_periods=4, trend='add', seasonal='add',).fit().fittedvalues

  holt_exp.append(predicted_value.astype(np.int64)[5:])

  print('{} : {}'.format(r, datetime.now()-start))
```

```
0 : 0:00:44.778242
```

```
1 : 0:01:22.074369
2 : 0:00:57.730067
3 : 0:00:52.785740
4 : 0:01:16.834581
5 : 0:01:05.281389
6 : 0:00:35.359353
7 : 0:01:08.440183
8 : 0:00:34.811765
9 : 0:02:19.719125
10 : 0:00:30.741994
11 : 0:01:23.832093
12 : 0:01:06.242445
13 : 0:00:33.683749
14 : 0:00:56.394602
15 : 0:01:44.796497
16 : 0:00:46.211646
17 : 0:01:00.126469
18 : 0:01:01.197081
19 : 0:01:06.528152
20 : 0:00:37.903262
21 : 0:00:30.357664
22 : 0:01:26.781773
23 : 0:02:18.257933
24 : 0:02:03.371294
25 : 0:00:33.774475
26 : 0:01:03.024825
27 : 0:00:50.922404
28 : 0:00:42.689623
29 : 0:00:31.058868
```

In [0]:

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

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

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

```
predicted_values=[]

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], .. 40 lsits]

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

alpha=0.3

predicted_values=[]

for r in tqdm(range(0,30)):

  for i in range(5,4464):

    if i==5:
      predicted_value = int(np.mean(regions_cum[r][:5]))
      predicted_values.append(predicted_value)
      continue

    predicted_values.append(predicted_value)

    predicted_value = int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
```

In [0]:

```
len(predicted_values)
```

Out[0]:

133770

In [0]:

```
predicted_values[:10]
```

Out[0]:

[163, 163, 169, 159, 138, 136, 147, 143, 147, 140]

In [0]:

```
output[0][:10]
```

Out[0]:

```
[178, 172, 156, 130, 136, 153, 142, 150, 137, 144]
```

In [0]:

```
# train, test split : 80% 20% split
# Before we start predictions using the tree based regression models we take January of 2016 pickup data
# and split it such that for every region we have 80% data in train and 20% in test,
# ordered date-wise for every region
print("# of train data points :", int(133770*0.8))
print("# of test data points :", int(133770*0.2))
```

```
# of train data points : 107016
# of test data points : 26754
```

In [0]:

```
print("# of train data points for a single cluster:", int(4459*0.8))
print("# of test data points for a single cluster:", int(4459*0.2))
```

```
# of train data points for a single cluster: 3567
# of test data points for a single cluster: 891
```

In [0]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
train_features =  [tsne_feature[i*4459:(4459*i+3567)] for i in range(0,30)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(4459*i)+3567:4459*(i+1)] for i in range(0,30)]
```

In [0]:

```
fft_train_features = [fft_features[i*4459:(4459*i+3567)] for i in range(0,30)]

fft_test_features = [fft_features[(4459*i)+3567:4459*(i+1)] for i in range(0,30)]
```

In [0]:

```
other_train_features = [mean_med_peaks_feats[i*4459:(4459*i+3567)] for i in range(0,30)]
```

```
other_test_features = [mean_med_peaks_feats[(4459*i)+3567:4459*(i+1)] for i in range(0,30)]
```

In [0]:

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

```
Number of data clusters 30 Number of data points in trian data 3567 Each data point contains 5 features
Number of data clusters 30 Number of data points in test data 892 Each data point contains 5 features
```

In [0]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
tsne_train_flat_lat = [i[:3567] for i in tsne_lat]
tsne_train_flat_lon = [i[:3567] for i in tsne_lon]
tsne_train_flat_weekday = [i[:3567] for i in tsne_weekday]
tsne_train_flat_output = [i[:3567] for i in output]
tsne_train_flat_holt = [i[:3567] for i in holt_exp]
tsne_train_flat_exp_avg = [predicted_values[i*4459:(4459*i+3567)] for i in range(30)]
```

In [0]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) for our test data
tsne_test_flat_lat = [i[3567:] for i in tsne_lat]
tsne_test_flat_lon = [i[3567:] for i in tsne_lon]
tsne_test_flat_weekday = [i[3567:] for i in tsne_weekday]
tsne_test_flat_output = [i[3567:] for i in output]
tsne_test_flat_holt = [i[3567:] for i in holt_exp]
tsne_test_flat_exp_avg = [predicted_values[(4459*i)+3567:4459*(i+1)] for i in range(30)]
```

In [0]:

```
# the above contains values in the form of list of lists (i.e. list of values of each region), here we make all of them in one list
train_new_features = []
for i in range(0,30):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,30):
    test_new_features.extend(test_features[i])
```

In [0]:

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

```
for i in range(0,30):
    fft_train_new_features.extend(fft_train_features[i])
fft_test_new_features = []
for i in range(0,30):
    fft_test_new_features.extend(fft_test_features[i])
```

In [0]:

```
other_train_new_features = []
for i in range(0,30):
    other_train_new_features.extend(other_train_features[i])
other_test_new_features = []
for i in range(0,30):
    other_test_new_features.extend(other_test_features[i])
```

In [0]:

```
np.array(train_new_features).shape
```

Out[0]:

```
(107010, 5)
```

In [0]:

```
np.array(fft_train_new_features).shape
```

Out[0]:

```
(107010, 10)
```

In [0]:

```
np.array(other_train_new_features).shape
```

Out[0]:

```
(107010, 3)
```

In [0]:

```
np.array(tsne_train_flat_output).shape
```

Out[0]:

```
(30, 3567)
```

In [0]:

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

tsne_train_lat = sum(tsne_train_flat_lat, [])
tsne_train_lon = sum(tsne_train_flat_lon, [])
tsne_train_weekday = sum(tsne_train_flat_weekday, [])
tsne_train_output = sum(tsne_train_flat_output, [])
tsne_train_exp_avg = sum(tsne_train_flat_exp_avg,[])
tsne_train_holt = sum(np.array(tsne_train_flat_holt).tolist(),[])
```

In [0]:

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

tsne_test_lat = sum(tsne_test_flat_lat, [])
tsne_test_lon = sum(tsne_test_flat_lon, [])
tsne_test_weekday = sum(tsne_test_flat_weekday, [])
tsne_test_output = sum(tsne_test_flat_output, [])
tsne_test_exp_avg = sum(tsne_test_flat_exp_avg,[])
tsne_test_holt = sum(np.array(tsne_test_flat_holt).tolist(),[])
```

DataFrame

In [0]:

```
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1','freq_1','freq_2','freq_3','freq_4','freq_5','amp_1','amp_2','amp_3','amp_4','amp_5','mean','median','
# of peaks']

df_train = pd.DataFrame(data=np.hstack((train_new_features, fft_train_new_features, other_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['holt_exp'] = tsne_train_holt

print(df_train.shape)
```

```
(107010, 23)
```

In [0]:

```
print(df_train.shape)
print(len(tsne_train_output))
```

```
(107010, 23)
107010
```

In [0]:

```
df_train.head(5)
```

Out[0]:

	ft_5	ft_4	ft_3	ft_2	ft_1	freq_1	freq_2	freq_3	freq_4	freq_5	amp_1	amp_2	amp_3	amp_4	amp_5	mean	median	# of peaks	lat
0	108.0	202.0	182.0	174.0	152.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	163.6	174.0	1.0	40.7624
1	202.0	182.0	174.0	152.0	178.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	177.6	178.0	0.0	40.7624
2	182.0	174.0	152.0	178.0	172.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	171.6	174.0	1.0	40.7624
3	174.0	152.0	178.0	172.0	156.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	166.4	172.0	1.0	40.7624
4	152.0	178.0	172.0	156.0	130.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	157.6	156.0	1.0	40.7624

In [0]:

```
# Preparing the data frame for our train data
df_test = pd.DataFrame(data=np.hstack((test_new_features, fft_test_new_features, other_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['holt_exp'] = tsne_test_holt
print(df_test.shape)
```

```
(26760, 23)
```

In [0]:

```
df_test.head(5)
```

Out[0]:

	ft_5	ft_4	ft_3	ft_2	ft_1	freq_1	freq_2	freq_3	freq_4	freq_5	amp_1	amp_2	amp_3	amp_4	amp_5	mean	median	# of peaks	lat
0	167.0	206.0	207.0	201.0	229.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	202.0	206.0	1.0	40.7624
1	206.0	207.0	201.0	229.0	220.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	212.6	207.0	2.0	40.7624
2	207.0	201.0	229.0	220.0	234.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	218.2	220.0	1.0	40.7624
3	201.0	229.0	220.0	234.0	235.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	223.8	229.0	1.0	40.7624
4	229.0	220.0	234.0	235.0	228.0	-27264.907807	-16914.84737	11458.304518	7434.829856	-8652.439594	0.000224	0.000448	0.000672	0.000896	0.00112	229.2	229.0	1.0	40.7624

In [0]:

```
df_train_temp = df_train.copy()

df_train_temp['output'] = tsne_train_output

df_train_temp.to_csv('NYC_Train.csv', index=False)
```

In [0]:

```
df_test_temp = df_test.copy()

df_test_temp['output'] = tsne_test_output

df_test_temp.to_csv('NYC_Test.csv', index=False)
```

Using Linear Regression

In [0]:

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

# some of methods of LinearRegression()
```

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

from sklearn.linear_model import LinearRegression
lr_reg=LinearRegression().fit(df_train.drop(['holt_exp'], axis=1), tsne_train_output)

y_pred = lr_reg.predict(df_test.drop(['holt_exp'], axis=1))
lr_test_predictions = [round(value) for value in y_pred]

y_pred = lr_reg.predict(df_train.drop(['holt_exp'], axis=1))
lr_train_predictions = [round(value) for value in y_pred]
```

Using Random Forest Regressor

In [0]:

```
# Training a hyper-parameter tuned random forest regressor on our train data
# find more about LinearRegression function here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
# -------------------------
# default paramters
# sklearn.ensemble.RandomForestRegressor(n_estimators=10, criterion='mse', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False)

# some of methods of RandomForestRegressor()
# apply(X) Apply trees in the forest to X, return leaf indices.
# decision_path(X) Return the decision path in the forest
# fit(X, y[, sample_weight]) Build a forest of trees from the training set (X, y).
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict regression target for X.
# score(X, y[, sample_weight]) Returns the coefficient of determination R^2 of the prediction.
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

regr1 = RandomForestRegressor(max_features='sqrt',min_samples_leaf=4,min_samples_split=3,n_estimators=40, n_jobs=-1)
regr1.fit(df_train.drop(['holt_exp'], axis=1), tsne_train_output)
```

Out[0]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='sqrt', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                      min_samples_leaf=4, min_samples_split=3,
                      min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=-1,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False)
```

In [0]:

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

# the models regr1 is already hyper parameter tuned
# the parameters that we got above are found using grid search

y_pred = regr1.predict(df_test.drop(['holt_exp'], axis=1))
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr1.predict(df_train.drop(['holt_exp'], axis=1))
rndf_train_predictions = [round(value) for value in y_pred]
```

In [0]:

```
#feature importances based on analysis using random forest
print (np.array(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'freq_1', 'freq_2', 'freq_3',
       'freq_4', 'freq_5', 'amp_1', 'amp_2', 'amp_3', 'amp_4', 'amp_5', 'mean',
       'median', '# of peaks', 'lat', 'lon', 'weekday', 'exp_avg']))
print (np.round(regr1.feature_importances_, 3))
```

```
['ft_5' 'ft_4' 'ft_3' 'ft_2' 'ft_1' 'freq_1' 'freq_2' 'freq_3' 'freq_4'
 'freq_5' 'amp_1' 'amp_2' 'amp_3' 'amp_4' 'amp_5' 'mean' 'median'
 '# of peaks' 'lat' 'lon' 'weekday' 'exp_avg']
[0.043 0.091 0.065 0.108 0.254 0.009 0.034 0.001 0.005 0.002 0.    0.
 0.    0.    0.    0.092 0.107 0.    0.001 0.007 0.001 0.18 ]
```

Using XgBoost Regressor

In [0]:

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

# find more about XGBRegressor function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#module-xgboost.sklearn
# -------------------------
# default paramters
# xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective='reg:linear',
# booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1,
# colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None,
# missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
```

```
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-ensembles/
# -----------------------

x_model = xgb.XGBRegressor(
 learning_rate =0.1,
 n_estimators=1000,
 max_depth=3,
 min_child_weight=3,
 gamma=0,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=0.8,nthread=4)
x_model.fit(df_train.drop(['holt_exp'], axis=1), tsne_train_output)
```

Out[0]:

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

In [0]:

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

y_pred = x_model.predict(df_test.drop(['holt_exp'], axis=1))
xgb_test_predictions = [round(value) for value in y_pred]
y_pred = x_model.predict(df_train.drop(['holt_exp'], axis=1))
xgb_train_predictions = [round(value) for value in y_pred]
```

In [0]:

```
#feature importances
print(np.array(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'freq_1', 'freq_2', 'freq_3',
       'freq_4', 'freq_5', 'amp_1', 'amp_2', 'amp_3', 'amp_4', 'amp_5', 'mean',
       'median', '# of peaks', 'lat', 'lon', 'weekday', 'exp_avg']))
print(np.round(x_model.feature_importances_, 3))
```

```
['ft_5' 'ft_4' 'ft_3' 'ft_2' 'ft_1' 'freq_1' 'freq_2' 'freq_3' 'freq_4'
 'freq_5' 'amp_1' 'amp_2' 'amp_3' 'amp_4' 'amp_5' 'mean' 'median'
 '# of peaks' 'lat' 'lon' 'weekday' 'exp_avg']
```

```
[0.001 0.    0.002 0.004 0.1   0.001 0.001 0.001 0.001 0.    0.    0.
 0.    0.    0.    0.001 0.001 0.    0.001 0.001 0.    0.885]
```

Calculating the error metric values for various models

In [0]:

```
train_mape=[]
test_mape=[]

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

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

In [0]:

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

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------
Baseline Model -                             Train:  0.1280437758477308       Test:  0.12219816254958768
Exponential Averages Forecasting -           Train:  0.12246580855928174       Test:  0.11710512044138537
Linear Regression -                         Train:  0.12240021520408155       Test:  0.11644387874447713
Random Forest Regression -                   Train:  0.08564116841540079       Test:  0.11307145897012695
XgBoost Regression -                         Train:  0.11686143512232376       Test:  0.11390345609864785
```

Error Metric Matrix

In [0]:

```
print ("My Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------------------------------")
print ("Baseline Model -                             Train: ",train_mape[0],"      Test: ",test_mape[0])
```

```
print ("Baseline Model -                            Train: ",train_mape[0],"      Test: ",test_mape[0])
print ("Exponential Averages Forecasting -          Train: ",train_mape[1],"      Test: ",test_mape[1])
print ("Linear Regression -                        Train: ",train_mape[4],"      Test: ",test_mape[4])
print ("Random Forest Regression -                  Train: ",train_mape[2],"     Test: ",test_mape[2])
print ("XgBoost Regression -                        Train: ",train_mape[3],"      Test: ",test_mape[3])
print ("---------------------------------------------------------------------------------------------------------")
# pt_no_fft.clear_rows()
pt_no_fft.add_row(["Exponential Averages Forecasting", round(train_mape[1],3), round(test_mape[1], 3)])
pt_no_fft.add_row(["Linear Regression", round(train_mape[4],3), round(test_mape[4],3) ])
pt_no_fft.add_row(["Random Forest Regression", round(train_mape[2],3), round(test_mape[2],3)])
pt_no_fft.add_row(["XgBoost Regression", round(train_mape[3],3), round(test_mape[3],3)])
```

```
My Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------
Baseline Model -                             Train:  0.1280437758477308       Test:  0.12219816254958768
Exponential Averages Forecasting -           Train:  0.12246580855928174       Test:  0.11710512044138537
Linear Regression -                         Train:  0.12240021520408155       Test:  0.11644387874447713
Random Forest Regression -                   Train:  0.08564116841540079      Test:  0.11307145897012695
XgBoost Regression -                         Train:  0.11686143512232376       Test:  0.11390345609864785
--------------------------------------------------------------------------------------------------------
```

Assignments

In [0]:

```
'''
Task 1: Incorporate Fourier features as features into Regression models and measure MAPE. <br>

Task 2: Perform hyper-parameter tuning for Regression models.
        2a. Linear Regression: Grid Search
        2b. Random Forest: Random Search
        2c. Xgboost: Random Search
Task 3: Explore more time-series features using Google search/Quora/Stackoverflow
to reduce the MAPE to < 12%
'''
```

Out[0]:

```
'\nTask 1: Incorporate Fourier features as features into Regression models and measure MAPE. <br>\n\nTask 2: Perform hyper-parameter tuning for Reg
ression models.\n        2a. Linear Regression: Grid Search\n        2b. Random Forest: Random Search \n        2c. Xgboost: Random Search\nTask 3:
Explore more time-series features using Google search/Quora/Stackoverflow\nto reduce the MAPE to < 12%\n'
```

Task 2, 3:

2a. Linear Regression : Grid Search

In [0]:

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

In [0]:

```
parameters = { 'alpha':[ 10**-6, 10**-4, 10**-2, 10**-1, 10**0, 10, 10**2, 10**4, 10**6] }
```

In [0]:

```
clf = linear_model.SGDRegressor('huber')
search = GridSearchCV(clf, parameters, cv=5, verbose=10)
```

In [0]:

```
search.fit(df_train, tsne_train_output)
```

In [0]:

```
search.best_params_
```

Out[0]:

```
{'alpha': 0.1}
```

In [0]:

```
y_pred = search.best_estimator_.predict(df_test)
lr_test_predictions = [int(value) for value in y_pred]

y_pred = search.best_estimator_.predict(df_train)
lr_train_predictions = [int(value) for value in y_pred]
```

In [0]:

```
print('Train MAPE:',(mean_absolute_error(tsne_train_output, lr_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))
print('Test MAPE:',(mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
```

```
Train MAPE: 0.12021031351815147
Test MAPE: 0.11489749664696444
```

In [0]:

```
pt_fft_other.add_row(['Linear Regression', np.round(0.12021031351815147, 3), np.round(0.11489749664696444, 3)])
```

2b. Random Forest : Random Search

In [0]:

```
import scipy.stats as st
from sklearn.model_selection import RandomizedSearchCV
```

In [0]:

```
# http://danielhnyk.cz/how-to-use-xgboost-in-python/
one_to_left = st.beta(10, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "n_estimators": st.randint(3, 40),
    "max_depth": st.randint(3, 40),
    'min_samples_leaf': st.randint(2, 10),
    'min_samples_split': st.randint(2, 10)
}
```

In [0]:

```
clf = RandomForestRegressor(max_features='sqrt')
search = RandomizedSearchCV(clf, params, verbose=10)
```

In [0]:

```
search.fit(df_train, tsne_train_output)
```

In [0]:

```
search.best_params_
```

Out[0]:

```
{'max_depth': 38,
 'min_samples_leaf': 6,
 'min_samples_split': 7,
 'n_estimators': 26}
```

In [0]:

```
y_pred = search.best_estimator_.predict(df_test)
rndf_test_predictions = [round(value) for value in y_pred]
```

```
y_pred = search.best_estimator_.predict(df_train)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [0]:

```
print('Train MAPE:',(mean_absolute_error(tsne_train_output, rndf_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))
print('Test MAPE:',(mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
```

```
Train MAPE: 0.093307030076362
Test MAPE: 0.11140093070422238
```

In [0]:

```
pt_fft_other.add_row(['Random Forest Regressor', np.round(0.093307030076362, 3), np.round(0.11140093070422238, 3)])
```

2C. Xgboost: Random Search

In [0]:

```
params = {
    "n_estimators": st.randint(3, 40),
    "max_depth": st.randint(3, 40),
    "learning_rate": st.uniform(0.05, 0.4),
    "colsample_bytree": one_to_left,
    "subsample": one_to_left,
    "gamma": st.uniform(0, 10),
    'reg_alpha': from_zero_positive,
    "min_child_weight": from_zero_positive,
}
```

In [0]:

```
reg = xgb.XGBRegressor(nthreads=-1)
search = RandomizedSearchCV(reg, params, n_jobs=4, verbose=10)
```

In [0]:

```
search.fit(df_train, tsne_train_output)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:   41.4s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:   58.4s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:  2.4min
```

```
[Parallel(n_jobs=4)]: Done  27 out of  30 | elapsed:  3.5min remaining:   23.6s
[Parallel(n_jobs=4)]: Done  30 out of  30 | elapsed:  3.6min finished
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None, nthreads=-1,
                                          objective='reg:linear',
                                          random_stat...
                                        'min_child_weight': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7efc19282668>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7efc195c3748>,
                                        'reg_alpha': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7efc19282668>,
                                        'subsample': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7efc192826a0>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=10)
```

In [0]:

```
search.best_params_
```

Out[0]:

```
{'colsample_bytree': 0.9934608311890654,
 'gamma': 2.380137843495951,
 'learning_rate': 0.17137402641765975,
 'max_depth': 24,
 'min_child_weight': 51.94031257832054,
 'n_estimators': 31,
 'reg_alpha': 57.69169205248272,
 'subsample': 0.8758934133375538}
```

In [0]:

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

In [0]:

```
print('Train MAPE:',(mean_absolute_error(tsne_train_output, xgb_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))
print('Test MAPE:',(mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
```

```
Train MAPE: 0.10229380204287129
Test MAPE: 0.11071790897777173
```

In [0]:

```
pt_fft_other.add_row(['XGBRegressor', np.round(0.10229380204287129, 3), np.round(0.11071790897777173, 3)])
```

Comparision of Models:

In [0]:

```
print('Comparision of Baseline Models:\n')
print(pt_base_models)
```

```
Comparision of Base Models:
+-------------------------------------------+---------------------+---------------------+
|                   Model                   |         MAPE        |         MSE         |
+-------------------------------------------+---------------------+---------------------+
|          Moving Averages(Ratios)          |  0.1632303629892675 | 538.6893891875746   |
|        Moving Averages(2016 Values)       |  0.1270124989230917 | 236.3680630227001   |
|                                           |                     |                     |
|      Weighted Moving Averages(Ratios)     | 0.16046320348034146 | 525.8311379928315   |
|   Weighted Moving Averages(2016 Values)   |  0.1216380430563497 | 223.38190710872163 |
|                                           |                     |                     |
|    Exponential Moving Averages(Ratios)    | 0.15992950939921804 | 516.9143667861409   |
| Exponential Moving Averages(2016 Values) |  0.1214696023422968 | 222.33956093189965 |
+-------------------------------------------+---------------------+---------------------+
```

In [0]:

```
print('With FFT features, Simple Exponential smoothing(SES) and already given hyperparameters:\n')
print(pt_no_fft)
```

```
With FFT features, Simple Exponential smoothing(SES) and already given hyperparameters:

+-----------------------------------+------------+-----------+
|               Model               | Train MAPE | Test MAPE |
+-----------------------------------+------------+-----------+
| Exponential Averages Forecasting |   0.122    |   0.117   |
|         Linear Regression         |   0.122    |   0.116   |
|      Random Forest Regression     |   0.086    |   0.113   |
|         XgBoost Regression        |   0.117    |   0.114   |
+-----------------------------------+------------+-----------+
```

In [0]:

In [6]:

```
print('With FFT features, Simple Exponential smoothing(SES), Triple Exponential Smoothing(Holt Winter) and HyperParameter tuning:\n')
print(pt_fft_other)
```

```
With FFT features, Simple Exponential smoothing(SES), Triple Exponential Smoothing(Holt Winter) and HyperParameter tuning:

+-------------------------+------------+-----------+
|          Model          | Train MAPE | Test MAPE |
+-------------------------+------------+-----------+
|    Linear Regression    |    0.12    |    0.12   |
| Random Forest Regressor |   0.093    |   0.111   |
|       XGBRegressor      |   0.102    |   0.111   |
+-------------------------+------------+-----------+
```

Steps:

1. As a first step I've gone through some internet references for getting some idea on NYC taxi commission and then downloaded the dataset of pickups for **January 2015, January 2016** from here.
2. Started exploring the data so that I get some understanding what to be used for featurization.
3. Based on the various columns available started plotting the boxplot, Kernel density estimates and getting the percentiles so that I can get bounds of inlier points. Using these bounds removed the outliers from data with **97.03%** of data retained.
4. Used the same bounds of January 2015 for January 2016 as January 2015 is used for training and added 10 minute time bins for regions in the data.
5. Based on geographical co-ordinates of January 2015 assigned the co-ordinates to the 10 minute pickup bins of January 2016 using **KMeans** and grouped the pickups bin based on the **cluster Ids**. With minimum distance between the regions of **0.5 miles** and maximum of **2 miles**.
6. Since there might be cases of no pickups in last **n** pickup bins first filled those empty pickups with **zeros** and then smoothed it by giving the **average** of values in pickup bins window for January 2015 data but only filled the missing values for Janauary 2016.
7. Then started exploring the some new features for the time series data and found features like,
    - **mean,**
    - **median,**
    - **# of peaks,**
    - **minimum,**
    - **maximum** etc...
8. Along with these features also added **Fourier features** to the data and this didn't helped much for reducing the **MAPE** which is used metric fro this problem.
9. Read many blogs on internet and found some interesting topics like smoothing the data i.e giving more imporatnce to the recent data as compared to past data,

    - **Simple exponential smoothing(SES) :** https://youtu.be/Fqge2HDH2Co
    - **Double exponential smoothing(Holt) :** https://youtu.be/DUyZl-abnNM
    - **Triple exponential smoothing(Holt Winter) :** https://youtu.be/mrLiC1biciY
10. Found the above featurizations very much relevant to the time series for forecasting using the previous values and incorporated them in my dataset.
11. Did the hyperparameter tuning as follows,

    - **Linear Regression ->> Grid Search**,
    - **RandomForestRegressor ->> Randomized Search**, and
    - **XGBoost Regressor ->> Randomized Search**.

    and got the best paramters for each of the model above and used these paramters for evaluation of test data and got the best paramiters for each of the model above and used these paramiters for evaluation of test data.

12. With use of **Simple exponential smoothing(SES)**, **Triple exponential smoothing(Holt Winter)** combined with **FFT features** I was able to reduce the **MAPE** to **11.1%** as compared to simple baseline models like **Simple moving averages, Weighted moving averages** of **MAPE = 12.14%**.

    - **Simple exponential smoothing(SES)** + **Triple exponential smoothing(Holt Winter)** + **FFT features ->> 11.1%**
    - **Simple moving averages, Weighted moving averages, Exponential moving averages ->> 12.7%, 12.16%, 12.14%**.

13. At the last compared the **MAPE** of all the models in the tables as shown above.