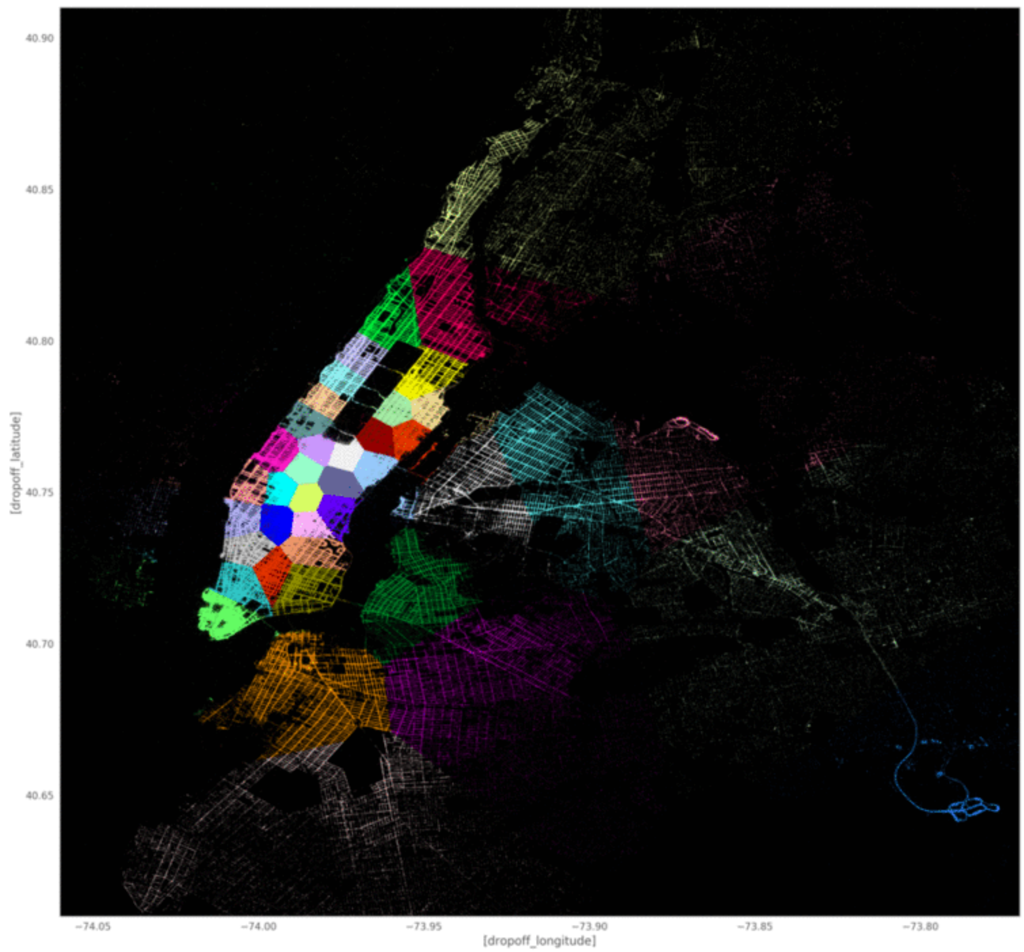
Taxi demand prediction in New York City



In [2]:
```
!pip install gpxpy
```

```
Collecting gpxpy
  Downloading https://files.pythonhosted.org/packages/9e/96/d9fdd462f17ef54fd64fe3f5e870c88e35686c740dcaed6460d048b43566/gpxpy-1.4.0.tar.gz (https://files.pythonhosted.org/packages/9e/96/d9fdd462f17ef54fd64fe3f5e870c88e35686c740dcaed6460d048b43566/gpxpy-1.4.0.tar.gz) (105kB)
     |████████████████████████████████| 112kB 2.8MB/s eta 0:00:01
Building wheels for collected packages: gpxpy
  Building wheel for gpxpy (setup.py) ... done
  Created wheel for gpxpy: filename=gpxpy-1.4.0-cp36-none-any.whl size=42813 sha256=5a911a384a97e99e02e7f24dfae7552e3e5b393f41d4f55ff596828ce6ff11c8
  Stored in directory: /root/.cache/pip/wheels/77/d7/ee/cb4d7a151ce924c35e681377fb90a0b882f55bfd3c2c586739
Successfully built gpxpy
Installing collected packages: gpxpy
Successfully installed gpxpy-1.4.0
```

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-tut
import dask.dataframe as dd#similar to pandas

import pandas as pd#pandas to create small dataframes

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

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

import time #Convert to unix time

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

# matplotlib: used to plot graphs
import matplotlib
# matplotlib.use('nbagg') : matplotlib uses this protocall which makes plots
#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 (l
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
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")
```

In [0]:
```
from google.colab import drive
drive.mount('/gdrive')
```

```
Mounted at /gdrive
```

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

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

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19

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

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
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&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

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

In [0]:

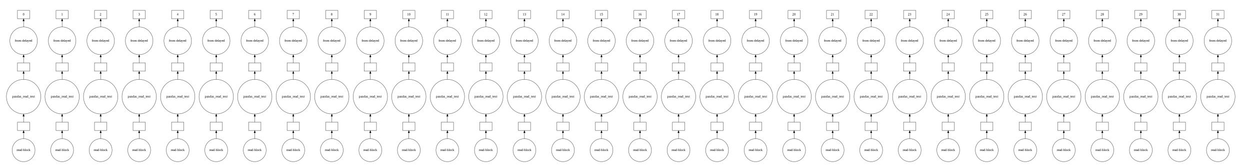
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_da
month = dd.read_csv('drive/My Drive/NYTD/yellow_tripdata_2015-01.csv')
print(month.columns)
```

```
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 immediat
# instead they add key-value pairs to an underlying Dask graph. Recall that i
# 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
month.visualize()
```

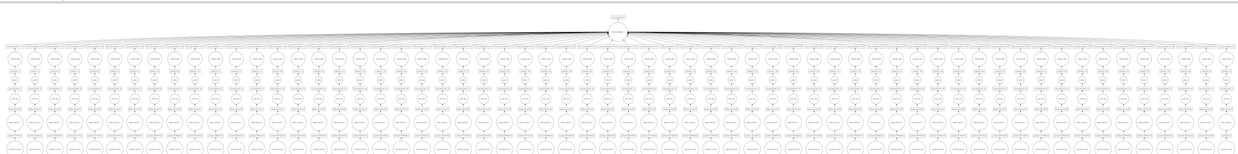
Out[15]:



In [0]:

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

Out[98]:



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 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
start = datetime.datetime.now()
print(month.head(5))
print('time taken:',datetime.datetime.now() - start)
```

```
   VendorID tpep_pickup_datetime  ... improvement_surcharge  total_amount
0         2  2015-01-15 19:05:39  ...                   0.3         17.05
1         1  2015-01-10 20:33:38  ...                   0.3         17.80
2         1  2015-01-10 20:33:38  ...                   0.3         10.80
3         1  2015-01-10 20:33:39  ...                   0.3          4.80
4         1  2015-01-10 20:33:39  ...                   0.3         16.30

[5 rows x 19 columns]
time taken: 0:00:03.256042
```

1. Pickup Latitude and Pickup Longitude

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

In [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 = month[((month.pickup_longitude <= -74.15) | (month.pickup
                        (month.pickup_longitude >= -73.7004) | (month.pickup_latit
#outlier_locations.compute()

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

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

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

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

Out[10]:

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

2. Dropoff Latitude & Dropoff Longitude

It is inferred from the source [https://www.flickr.com/places/info/2459115 (https://www.flickr.com/places/info/2459115)](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 = month[((month.dropoff_longitude <= -74.15) | (month.dropo
                    (month.dropoff_longitude >= -73.7004) | (month.dropoff_lat

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

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

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

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

Out[11]:

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) & spee

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



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

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

    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 dropo
#   1                  1.59       -73.993896          40.750111       -73.9
#   1                   3.30        -74.001648         40.724243       -73.9
#   1                   1.80        -73.963341         40.802788       -73.9
#   1                   0.50        -74.009087         40.713818       -74.0
#   1                   3.00        -73.971176         40.762428       -74.0
frame_with_durations = return_with_trip_times(month)
```

In [0]:
```
# the skewed box plot shows us the presence of outliers
sns.boxplot(y="trip_times", data =frame_with_durations)
plt.show()
```

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

In [0]:
```
#calculating 0-100th percentile to find a the correct percentile value for re
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  548555.633333
```

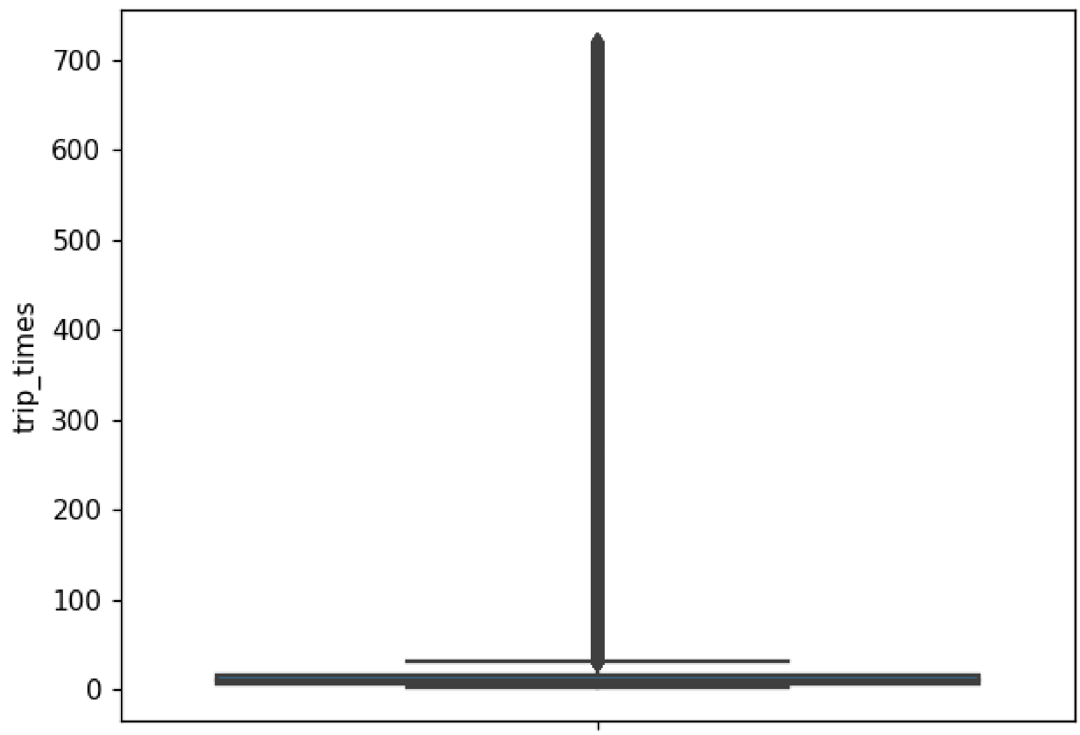
In [0]:
```
#looking further from the 99th percecntile
for i in range(90,100):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100
print ("100 percentile value is ",var[-1])
```

```
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  548555.633333
```

In [0]:
```
#removing data based on our analysis and TLC regulations
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip
```

In [0]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.show()
```

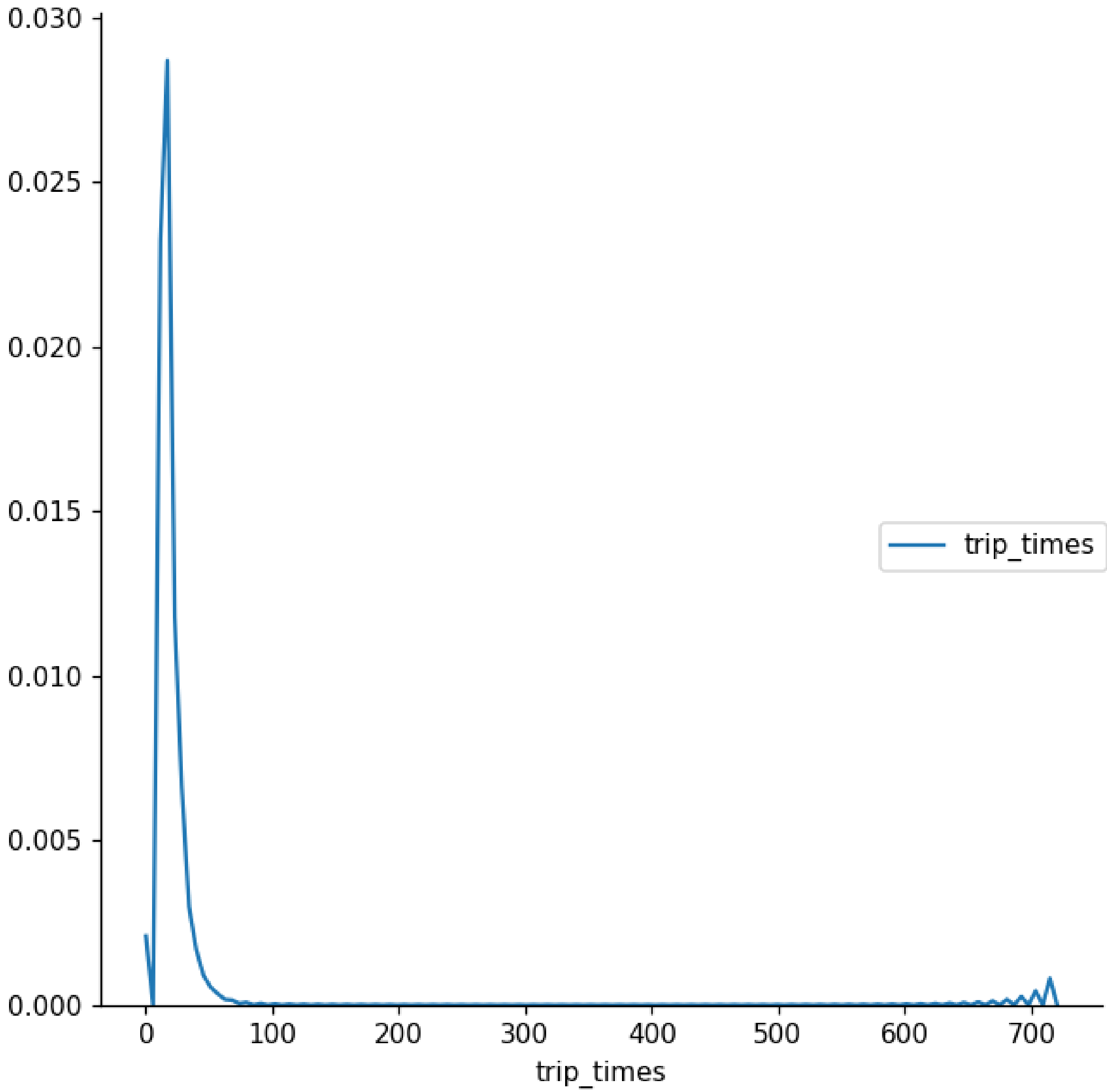
<IPython.core.display.Javascript object>



In [0]:

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

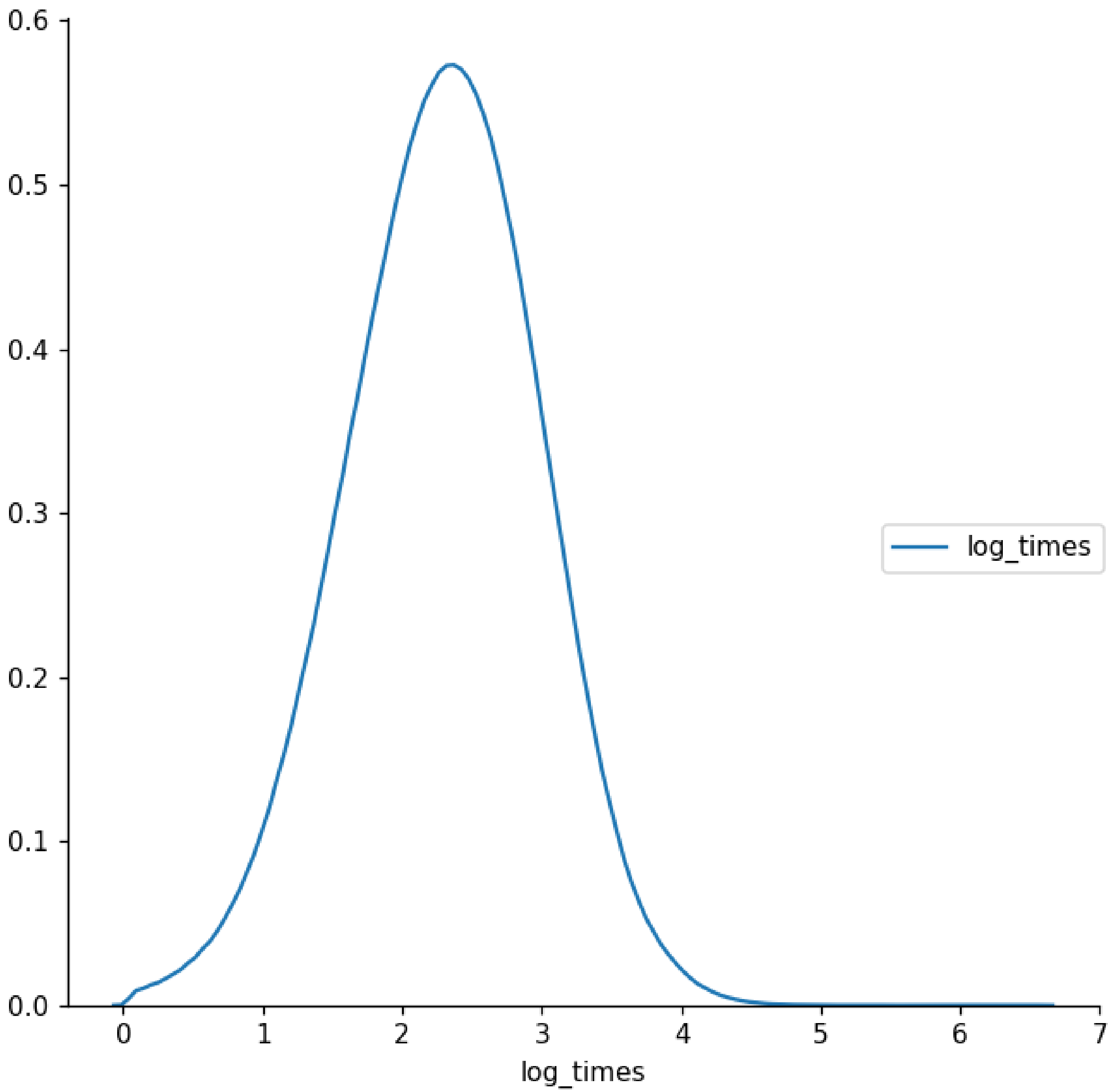
```
<IPython.core.display.Javascript object>
```



In [0]:
```python
#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_d
```

In [0]:
```python
#pdf of log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```

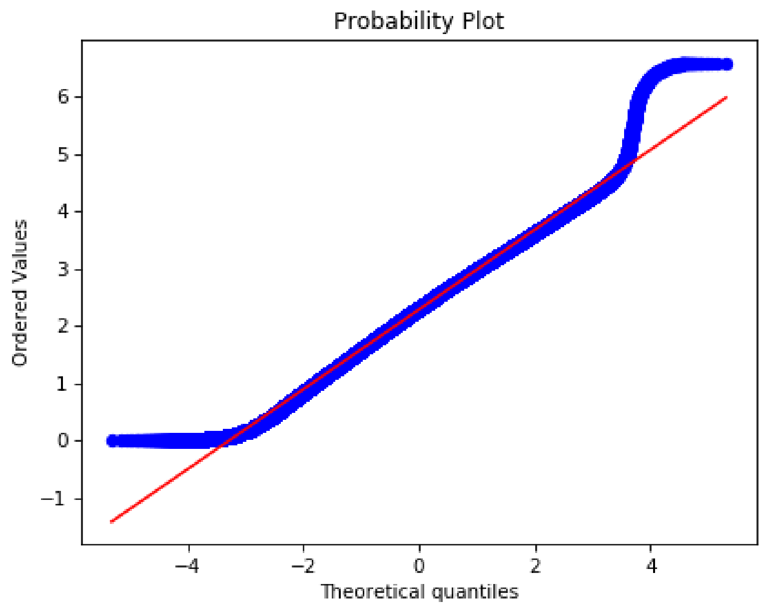
```
<IPython.core.display.Javascript object>
```



In [0]:

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

<IPython.core.display.Javascript object>

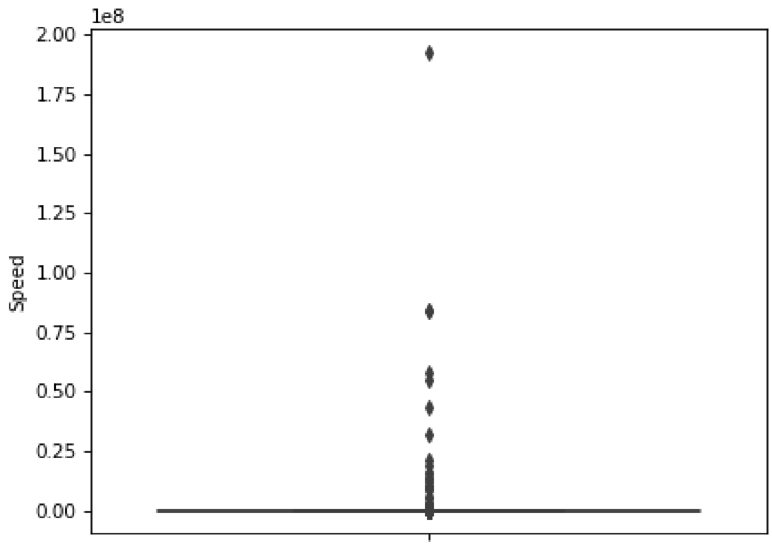


4. Speed

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['t
sns.boxplot(y="Speed", data =frame_with_durations_modified)
plt.show()
```

<IPython.core.display.Javascript object>



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  192857142.857
```

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])
```

```
90 percentile value is 20.186915887850468
91 percentile value is 20.91645569620253
92 percentile value is 21.752988047808763
93 percentile value is 22.721893491124263
94 percentile value is 23.844155844155843
95 percentile value is 25.182552504038775
96 percentile value is 26.80851063829787
97 percentile value is 28.84304932735426
98 percentile value is 31.591128254580514
99 percentile value is 35.7513566847558
100 percentile value is  192857142.857
```

In [0]:
```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.
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+
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  192857142.857
```

In [0]:
```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Spee
```

In [0]:
```
#avg.speed of cabs in New-York
sum(frame_with_durations_modified["Speed"]) / float(len(frame_with_durations_
```

Out[17]:
```
12.450173996027528
```

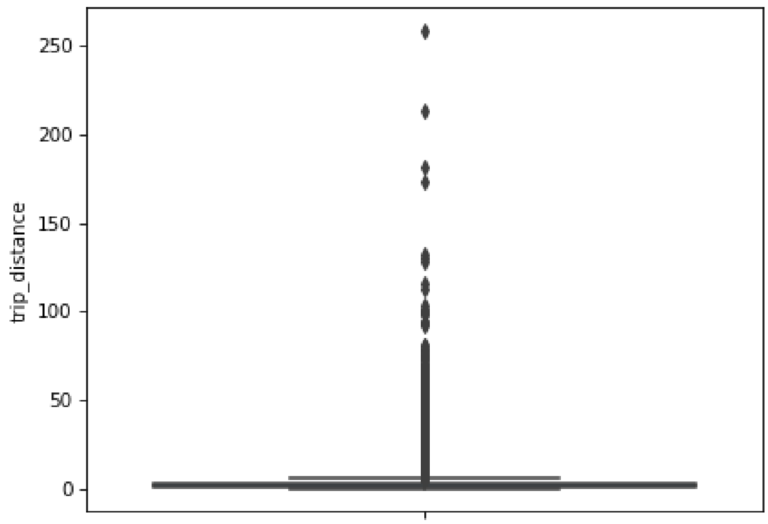
The avg speed in Newyork speed is 12.45miles/hr, so a cab driver can travel 2 miles per 10min on avg.

4. Trip Distance

In [0]:

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

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



In [0]:

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

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

In [0]:
```
#calculating trip distance values at each percntile 90,91,92,93,94,95,96,97,9
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
```

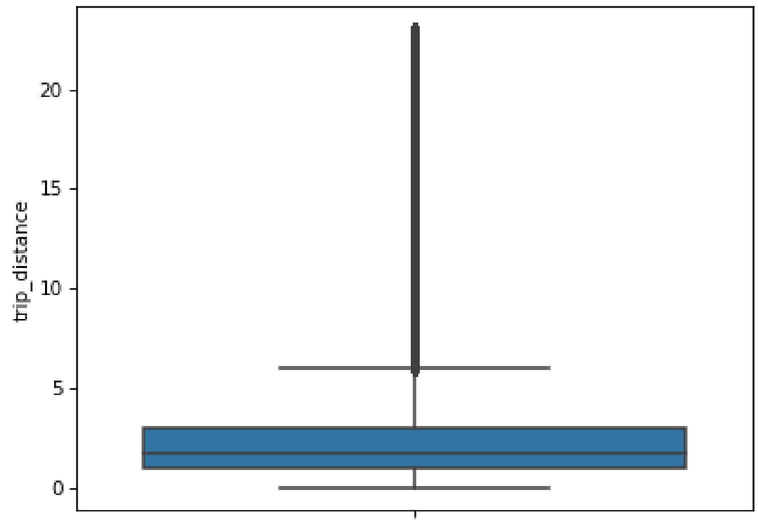
In [0]:
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+
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
```

In [0]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```

<IPython.core.display.Javascript object>

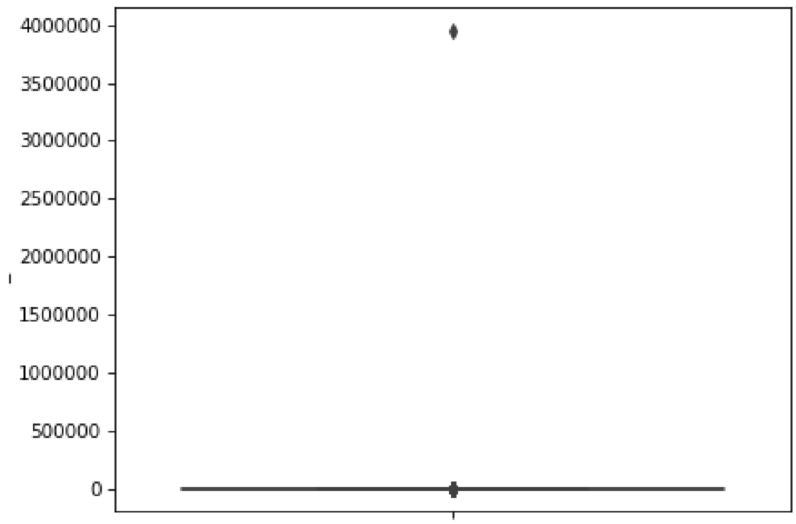


5. Total Fare

In [0]:

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

<IPython.core.display.Javascript object>



In [0]:

```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,7
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,
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 99.0,99.1,99.2,99.3,9
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+
print("100 percentile value is ",var[-1])
```

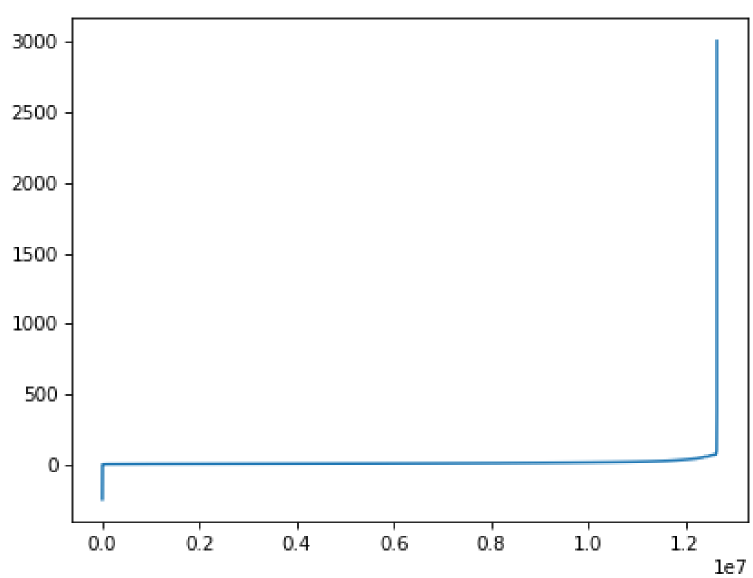
```
99.0 percentile value is 68.13
99.1 percentile value is 69.13
99.2 percentile value is 69.6
99.3 percentile value is 69.73
99.4 percentile value is 69.73
99.5 percentile value is 69.76
99.6 percentile value is 72.46
99.7 percentile value is 72.73
99.8 percentile value is 80.05
99.9 percentile value is 95.55
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]:
```
del frame_with_durations_modified
```

In [0]:
```
#below plot shows us the fare values(sorted) to find a sharp increase to remo
# plot the fare amount excluding last two values in sorted data
plt.plot(var[:-2])
plt.show()
```

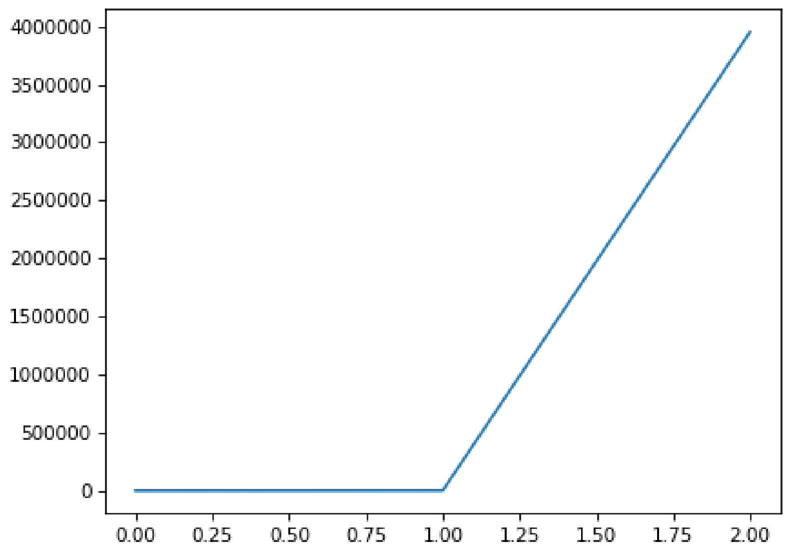
```
<IPython.core.display.Javascript object>
```



In [0]:

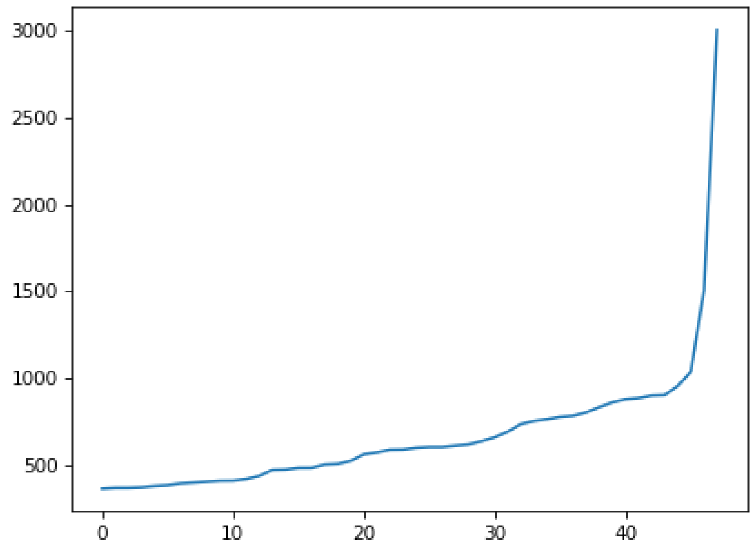
```
# a very sharp increase in fare values can be seen
# plotting last three total fare values, and we can observe there is share in
plt.plot(var[-3:])
plt.show()
```

<IPython.core.display.Javascript object>



In [0]:
```
#now looking at values not including the last two points we again find a dras
# we plot last 50 values excluding last two values
plt.plot(var[-50:-2])
plt.show()
```

<IPython.core.display.Javascript object>



Remove all outliers/erronous points.

In [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_fr
                        (new_frame.dropoff_latitude >= 40.5774) & (new_frame.d
                        ((new_frame.pickup_longitude >= -74.15) & (new_frame.p
                        (new_frame.pickup_longitude <= -73.7004) & (new_frame.
    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
    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_di
    d = temp_frame.shape[0]
    print ("Number of outliers from trip distance analysis:",(a-d))

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

    temp_frame = new_frame[(new_frame.total_amount <1000) & (new_frame.total_
    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_fra
                        (new_frame.dropoff_latitude >= 40.5774) & (new_frame.d
                        ((new_frame.pickup_longitude >= -74.15) & (new_frame.p
                        (new_frame.pickup_longitude <= -73.7004) & (new_frame.

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_dis
    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_a

    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(le
```

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

Data-preperation

Clustering/Segmentation

In [0]:

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

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

# we need to choose number of clusters so that, there are more number of clus
#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 =  1.0933194607372518
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
16.0
Min inter-cluster distance =  0.7123318236197774
---
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 =  0.5179286172497254
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
9.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
31.0
Min inter-cluster distance =  0.5064095487015858
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
38.0
Min inter-cluster distance =  0.36495419250817024
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
46.0
Min inter-cluster distance =  0.346654501371586
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
6.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
54.0
Min inter-cluster distance =  0.30468071844965394
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
62.0
Min inter-cluster distance =  0.29187627608454664
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2
1.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
69.0
Min inter-cluster distance =  0.18237562550345013
---
```

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]:
```
coords = jan_2015_frame[['pickup_latitude', 'pickup_longitude']].values
```

In [0]:
```
# if check for the 50 clusters you can observe that there are two clusters wi
# so we choose 40 clusters for solve the further problem

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

Plotting the cluster centers:

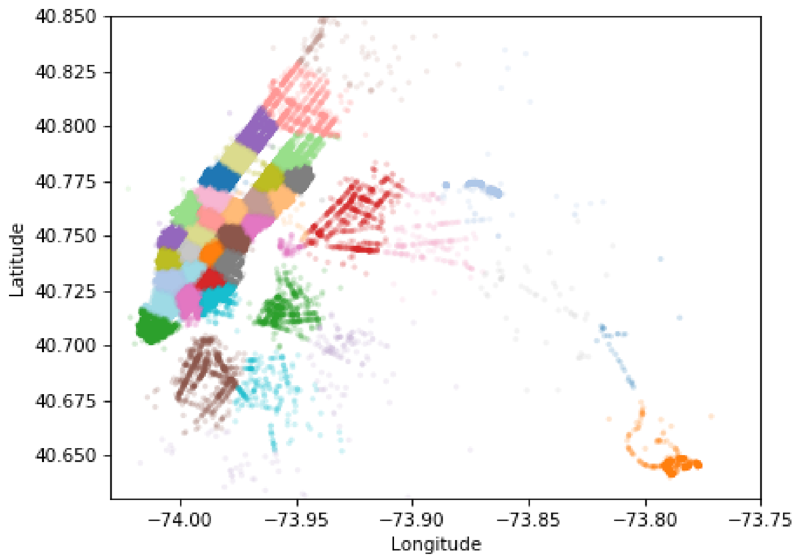
In [0]:
```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])),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

Out[48]:

Plotting the clusters:

In [0]:
```python
#Visualising the clusters on a map
def plot_clusters(frame):
    city_long_border = (-74.03, -73.75)
    city_lat_border = (40.63, 40.85)
    fig, ax = plt.subplots(ncols=1, nrows=1)
    ax.scatter(frame.pickup_longitude.values[:100000], frame.pickup_latitude.
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```

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



Time-binning

In [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
# 1456790400 : 2016-03-01 00:00:00
# 1459468800 : 2016-04-01 00:00:00
# 1462060800 : 2016-05-01 00:00:00
# 1464739200 : 2016-06-01 00:00:00

def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,143
                    [1451606400,1454284800,1456790400,1459468800,1462060800,1

    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
    tenminutewise_binned_unix_pickup_times=[(int((i-start_pickup_unix)/600)+3
    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 bi
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(fram
jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2015
jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_dista
```

In [0]:
```
del jan_2015_frame
del jan_2015_groupby
```

In [0]:
```
jan_2015_frame.to_csv('jan_2015_frame.csv',index = False)
jan_2015_groupby.to_csv('jan_2015_groupby.csv',index = False)
```

In [0]:
```
#del frame_with_durations
del frame_with_durations_outliers_removed
del frame_with_durations
```

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[14]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
0	1	1.59	-73.993896	40.750111	-73.974785	40.75(
1	1	3.30	-74.001648	40.724243	-73.994415	40.75ç
2	1	1.80	-73.963341	40.802788	-73.951820	40.82‹
3	1	0.50	-74.009087	40.713818	-74.004326	40.71ç
4	1	3.00	-73.971176	40.762428	-74.004181	40.74ž

In [0]:

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

Out[31]:

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

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_amo
# 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_durati

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

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

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('drive/My Drive/NYTD/yellow_tripdata_2016-01.csv
month_feb_2016 = dd.read_csv('drive/My Drive/NYTD/yellow_tripdata_2016-02.csv
month_mar_2016 = dd.read_csv('drive/My Drive/NYTD/yellow_tripdata_2016-03.csv
```

Type *Markdown* and LaTeX: α^2

In [0]:

```
jan_2016_frame,jan_2016_groupby = datapreparation(month_jan_2016,kmeans,1,201
feb_2016_frame,feb_2016_groupby = datapreparation(month_feb_2016,kmeans,2,201
mar_2016_frame,mar_2016_groupby = datapreparation(month_mar_2016,kmeans,3,201
```

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

In [0]:

```
jan_2016_frame.to_csv('jan_2016_frame.csv',index = False)
jan_2016_groupby.to_csv('jan_2016_groupby.csv',index = False)
feb_2016_frame.to_csv('feb_2016_frame.csv',index = False)
feb_2016_groupby.to_csv('feb_2016_groupby.csv',index = False)
mar_2016_frame.to_csv('march_2016_frame.csv',index = False)
mar_2016_groupby.to_csv('march_2016_groupby.csv',index = False)
```

Type *Markdown* and LaTeX: α^2

In [0]:

```
del month_jan_2016
del month_feb_2016
del month_mar_2016
```

Smoothing

In [0]:

```
jan_2015_frame = pd.read_csv('drive/My Drive/NYTD/jan_2015_frame.csv')
feb_2016_frame = pd.read_csv('drive/My Drive/NYTD/feb_2016_frame.csv')
march_2016_frame = pd.read_csv('drive/My Drive/NYTD/march_2016_frame.csv')
jan_2016_frame = pd.read_csv('drive/My Drive/NYTD/jan_2016_frame.csv')
jan_2015_groupby = pd.read_csv('drive/My Drive/NYTD/jan_2015_groupby.csv')
jan_2016_groupby = pd.read_csv('drive/My Drive/NYTD/jan_2016_groupby.csv')
feb_2016_groupby = pd.read_csv('drive/My Drive/NYTD/feb_2016_groupby.csv')
march_2016_groupby = pd.read_csv('drive/My Drive/NYTD/march_2016_groupby.csv'
```

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
# we got an observation that there are some pickpbins that doesnt have any pi
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,40):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [0]:

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

#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)
```

In [0]:

```
#march
mar_2016_unique = return_unq_pickup_bins(march_2016_frame)
```

In [11]:

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

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

```
----------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  31
----------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  214
----------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  36
----------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  41
----------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1180
----------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  42
----------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  44
----------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  43
----------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  39
----------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  42
----------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  36
----------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  321
----------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  36
----------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  43
----------------------------------------------------------
```

there are two ways to fill up these values

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

In [0]:

```python
# 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 1
# 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 uniqu
# if it is there we will add the count_values[index] to smoothed data
# if not we add 0 to the smoothed data
# we finally return smoothed data
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,40):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [0]:

```python
# 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 1
# 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 uniqu
# 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
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigio
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already v
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value o
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two know
                        smoothed_value=(count_values[ind-1]+count_values[ind]
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are foun
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]:
                            continue
                        else:
                            right_hand_limit=j
                            break
```

```
                    smoothed_value=count_values[ind]*1.0/((right_hand_limit-i
                    for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                    repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions

```

In [0]:

```
jan_2015_groupby = pd.read_csv('drive/My Drive/NYTD/jan_2015_groupby.csv')
```

In [0]:

```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_201

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015
```

Type *Markdown* and LaTeX: α^2

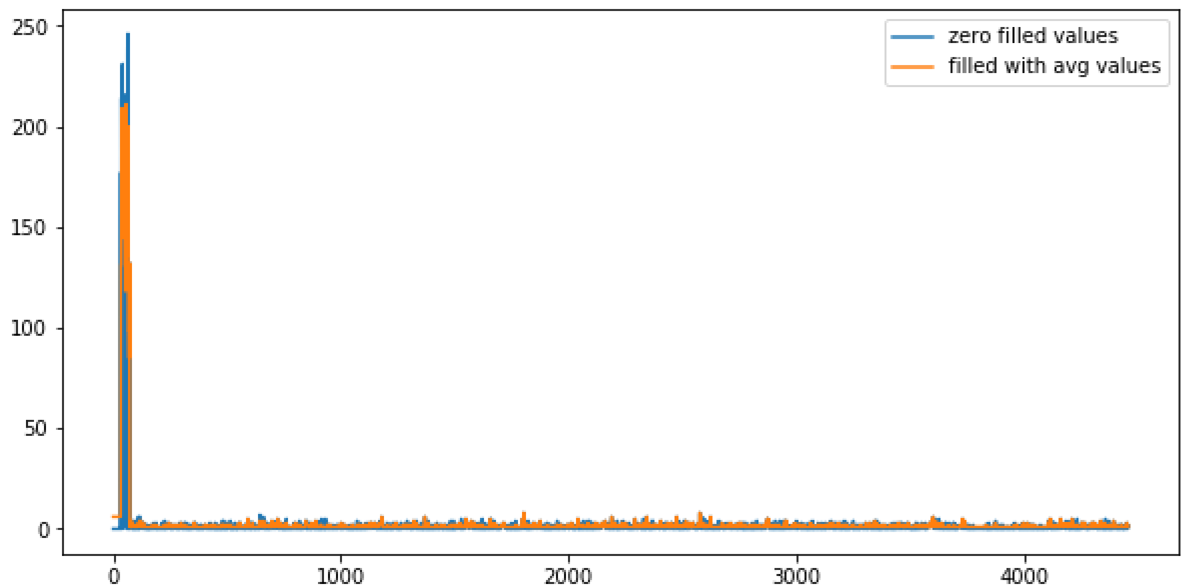
In [16]:

```
# 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 (leng
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

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

In [17]:

```
# 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]:

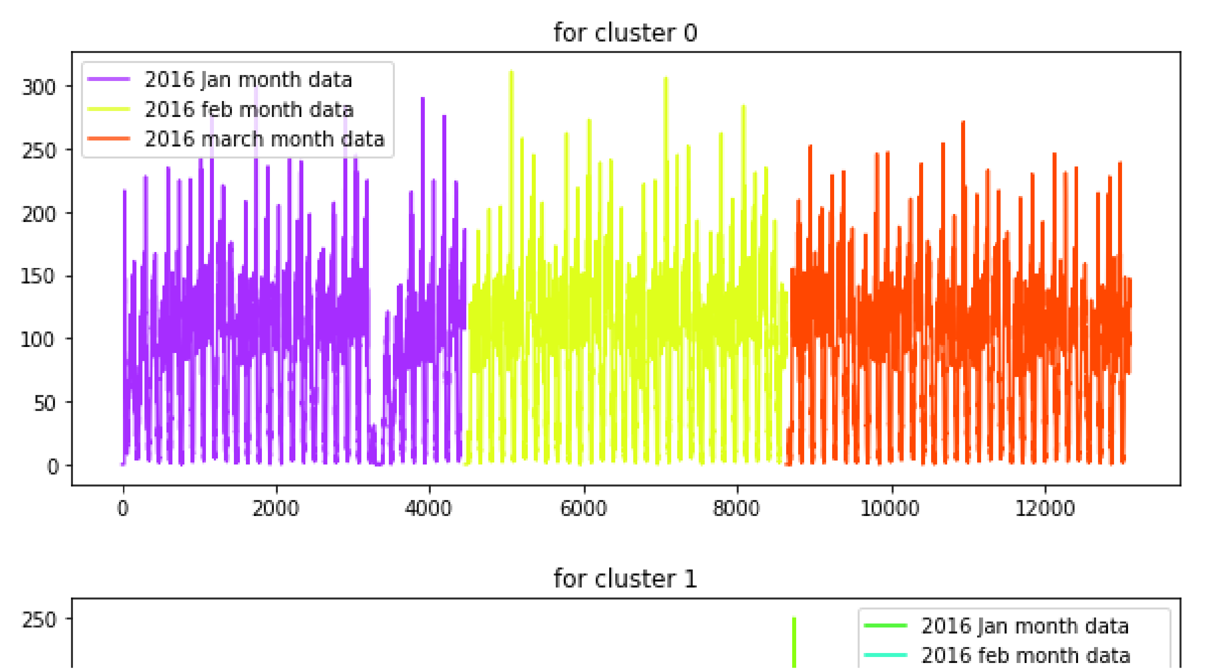
```
# why we choose, these methods and which method is used for which data?

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

# 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]:

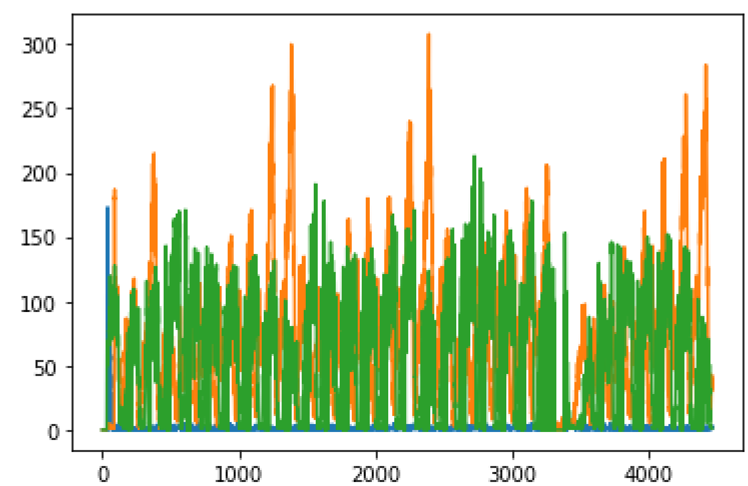
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are fil
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2
mar_2016_smooth = fill_missing(march_2016_groupby['trip_distance'].values,mar

# Making list of all the values of pickup data in every bin for a period of 3
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+446
# that are happened for three months in 2016 data

for i in range(0,40):
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_smooth[417

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

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='2
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), lab
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='2
    plt.title('for cluster '+ str(i))
    plt.legend()
    plt.show()
```



In [0]:
```
plt.plot(regions_cum[1][0:4464])
plt.plot(regions_cum[2][0:4464])
plt.plot(regions_cum[3][0:4464])

plt.show()
```

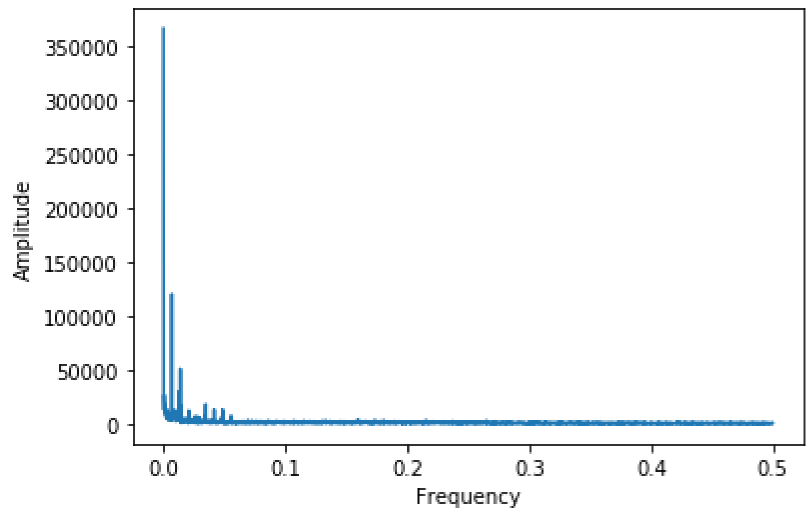


In [0]:
```
len(regions_cum)
```

Out[50]: 40

In [0]:
```
%matplotlib inline
```

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/g
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/gen
freq = np.fft.fftfreq(4460, 1)
n = len(freq)
plt.figure()
plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



In [0]:
```
  #Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) va
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*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_r
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicte
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(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(
    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]:
```
def MA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=1
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].valu
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-windo
        else:
            predicted_value=int(sum((ratios['Prediction'].values)[0:(i+1)])/(

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

```
def WA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_r
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicte
        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(
    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*40):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].valu
        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(
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

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

Exponential Weighted Moving Averages

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

In exponential moving averages we use a single hyperparameter alpha (α) which is a value between 0 & 1 and based on the value of the hyperparameter alpha the weights and the window sizes are configured.
For eg. If $\alpha = 0.9$ then the number of days on which the value of the current iteration is based is~ $1/(1-\alpha) = 10$ i.e. we consider values 10 days prior before we predict the value for the current

iteration. Also the weights are assigned using $2/(N+1) = 0.18$,where N = number of prior values being considered, hence from this it is implied that the first or latest value is assigned a weight of 0.18 which keeps exponentially decreasing for the subsequent values.

$$R_t^{\prime} = \alpha * R_{t-1} + (1-\alpha) * R_{t-1}^{\prime}$$

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*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_r
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicte
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratic

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

$$P_t^{\prime} = \alpha * P_{t-1} + (1-\alpha) * P_{t-1}^{\prime}$$

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*40):
        if i%4464==0:
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].valu
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Pr

    ratios['EA_P1_Predicted'] = predicted_values
    ratios['EA_P1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(
    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')
```

Type *Markdown* and LaTeX: α^2

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]:
```
print ("Error Metric Matrix (Forecasting Methods) - MAPE & MSE")
print ("--------------------------------------------------------------------------
print ("Moving Averages (Ratios) -                             MAPE: ",mean_e
print ("Moving Averages (2016 Values) -                        MAPE: ",mean_e
print ("--------------------------------------------------------------------------
print ("Weighted Moving Averages (Ratios) -                    MAPE: ",mean_e
print ("Weighted Moving Averages (2016 Values) -               MAPE: ",mean_e
print ("--------------------------------------------------------------------------
print ("Exponential Moving Averages (Ratios) -              MAPE: ",mean_err[
print ("Exponential Moving Averages (2016 Values) -         MAPE: ",mean_err[
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------
-------------------------
Moving Averages (Ratios) -                             MAPE:  0.182115517339
MSE:  400.0625504032258
Moving Averages (2016 Values) -                        MAPE:  0.14292849687
MSE:  174.84901993727598
--------------------------------------------------------------------------------
-------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.178486925438
MSE:  384.01578741039424
Weighted Moving Averages (2016 Values) -               MAPE:  0.135510884362
MSE:  162.46707549283155
--------------------------------------------------------------------------------
-------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.177835501949
MSE:  378.34610215053766
Exponential Moving Averages (2016 Values) -         MAPE:  0.135091526367
MSE:  159.73614471326164
```

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

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

Regression Models

Top fourier features(frequencies and amplitudes)

Train-Test Split

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

In [0]:

```
#now we will calculate the top frequencies and amplitudes

#all the fourier features
final_df = pd.DataFrame(columns = ["Amplitude1","Amplitude2","Amplitude3","Am
for i in range(0,40):


  #we will finally append the values in dataframe
    jan_df = pd.DataFrame()
    feb_df = pd.DataFrame()
    march_df = pd.DataFrame()

   #we will calculate fft for each month
  #as length of regions_cum is where it holds data for each of the clusters

  #for month of january

    january_data = regions_cum[i][0:4464]
    jan_df['Amplitude'] = sorted(np.fft.fft(january_data),reverse = True)[0:5
    jan_df['Frequency'] = sorted(np.fft.fftfreq(4464,1),reverse = True)[0:5]
    jan_df = jan_df.reset_index(drop = True).T
    jan_list = []
  #jan_final = fourier(jan_df,4464)


  #feb data
    february_data = regions_cum[i][4464:4464 + 4176]
    feb_df['Amplitude'] = sorted(np.fft.fft(february_data),reverse = True)[0:
    feb_df['Frequency'] = sorted(np.fft.fftfreq(4464,1),reverse = True)[0:5]
    feb_df = feb_df.reset_index(drop = True).T
    feb_list = []
  #feb_final = fourier(feb_df,4176)

  #march data
    march_data = regions_cum[i][4464+4176:4464+4176+4464]
    march_df['Amplitude'] = sorted(np.fft.fft(march_data),reverse = True)[0:5
    march_df['Frequency'] = sorted(np.fft.fftfreq(4464,1),reverse = True)[0:5
    march_df = march_df.reset_index(drop = True).T
    march_list = []


    jan_list = []
    feb_list = []
    march_list = []
    for i in range(0,5):

        jan_list.append(float(fr_am_jan_sorted[i]['Frequency']))
        jan_list.append(float(fr_am_jan_sorted[i]['Amplitude']))

        feb_list.append(float(fr_am_feb_sorted[i]['Frequency']))
        feb_list.append(float(fr_am_feb_sorted[i]['Amplitude']))

        march_list.append(float(fr_am_mar_sorted[i]['Frequency']))
        march_list.append(float(fr_am_mar_sorted[i]['Amplitude']))

    new_jan = pd.DataFrame([fr_am_list_jan]*4464)
```

```
    new_feb = pd.DataFrame([fr_am_list_feb]*4176)
    new_mar = pd.DataFrame([fr_am_list_mar]*4464)

    new_jan.columns = ["Amplitude1","Amplitude2","Amplitude3","Amplitude4","A
    new_feb.columns = ["Amplitude1","Amplitude2","Amplitude3","Amplitude4","A
    new_mar.columns = ["Amplitude1","Amplitude2","Amplitude3","Amplitude4","A


    final_df = final_df.append(fr_am_new_jan, ignore_index=True)
    final_df = final_df.append(fr_am_new_feb, ignore_index=True)
    final_df = final_df.append(fr_am_new_mar, ignore_index=True)
```

In [0]:

```
final_df = fourier_features_df
```

In [22]:

```
final_df.head()
```

Out[22]:

	Amplitude1	Amplitude2	Amplitude3	Amplitude4	Amplitude5	Freq1	Freq2	Fre
0	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.499552	0.4993
1	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.499552	0.4993
2	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.499552	0.4993
3	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.499552	0.4993
4	367173.0	94490.188858	94490.188858	14349.849101	14349.849101	0.499776	0.499552	0.4993

In [23]:
```
## Feature engineering part for the data
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps)

f_train = pd.DataFrame(columns=['Amplitude1','Amplitude2','Amplitude3','Ampli
f_test = pd.DataFrame(columns=['Amplitude1','Amplitude2','Amplitude3','Amplit
#Train data
for i in range(40):
    f_train = f_train.append(final_df[i*13099 : 13099*i + 9169])

#Test data
for i in range(40):
    f_test = f_test.append(final_df[i*13099 + 9169 : 13099*(i+1)])

#Reset all the indexes in train and test data
f_train.reset_index(inplace=True)
f_test.reset_index(inplace = True)

fourier_train_df=f_train.drop(labels="index", axis=1)
fourier_test_df=f_test.drop(labels="index", axis=1)

print("Shape of the fourier transformed train dataframe: ",fourier_train_df.s
print("Shape of the fourier transformed test dataframe: ",fourier_test_df.sha
fourier_test_df.head()

```

```
Shape of the fourier transformed train dataframe:  (366760, 10)
Shape of the fourier transformed test dataframe:  (157200, 10)
```

Out[23]:

	Amplitude1	Amplitude2	Amplitude3	Amplitude4	Amplitude5	Freq1	Freq2	Fre
0	387761.0	91160.781939	91160.781939	17509.351171	17509.351171	0.499776	0.499552	0.4993
1	387761.0	91160.781939	91160.781939	17509.351171	17509.351171	0.499776	0.499552	0.4993
2	387761.0	91160.781939	91160.781939	17509.351171	17509.351171	0.499776	0.499552	0.4993
3	387761.0	91160.781939	91160.781939	17509.351171	17509.351171	0.499776	0.499552	0.4993
4	387761.0	91160.781939	91160.781939	17509.351171	17509.351171	0.499776	0.499552	0.4993

Time Series data feature engineering

In this part I have considered two different parts ,first one is High band pass filter and Wavelet Denposing while the second part is Holt's Trend Model adn smoothing

Part 1:High Band pass filer + Wavelet Denoising

for understanding about the wavelet denosing part I have referred to following blogs and kernels whcih have explained the concepts and code fluently and in an excellent manner

1.[https://www.kaggle.com/jackvial/dwt-signal-denoising (https://www.kaggle.com/jackvial/dwt-signal-denoising)](https://www.kaggle.com/jackvial/dwt-signal-denoising)

2.[https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/ (https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/)](https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/)

3.[https://www.kaggle.com/theoviel/fast-fourier-transform-denoising (https://www.kaggle.com/theoviel/fast-fourier-transform-denoising)](https://www.kaggle.com/theoviel/fast-fourier-transform-denoising)

The high amplitude, unstable parts of the signal (with high peaks and valleys) are the actual signal we are looking for. The medium amplitude patches of the signal (between the high amplitude regions) represent the unnecessary noise and artificial impulse, and the wavelet method seems to do better at removing these patches. This denoising illustrated in the image at the beginning of the kernel.

In [0]:

```
import os
import gc
from numpy.fft import *
import pywt
from statsmodels.robust import mad
import scipy
from scipy import signal
from scipy.signal import butter, deconvolve
import warnings
warnings.filterwarnings('ignore')
```

In [0]:

```
SIGNAL_LEN = 4464 + 4176 + 4464 #signal length corresponds to the length size
SAMPLE_RATE = 40000 #this is the sampling rate we are considering here
```

In [0]:

```
signals = []
for i in range(40):
    signals.append(np.array(regions_cum[i][0:4464+4176+4464]))


```

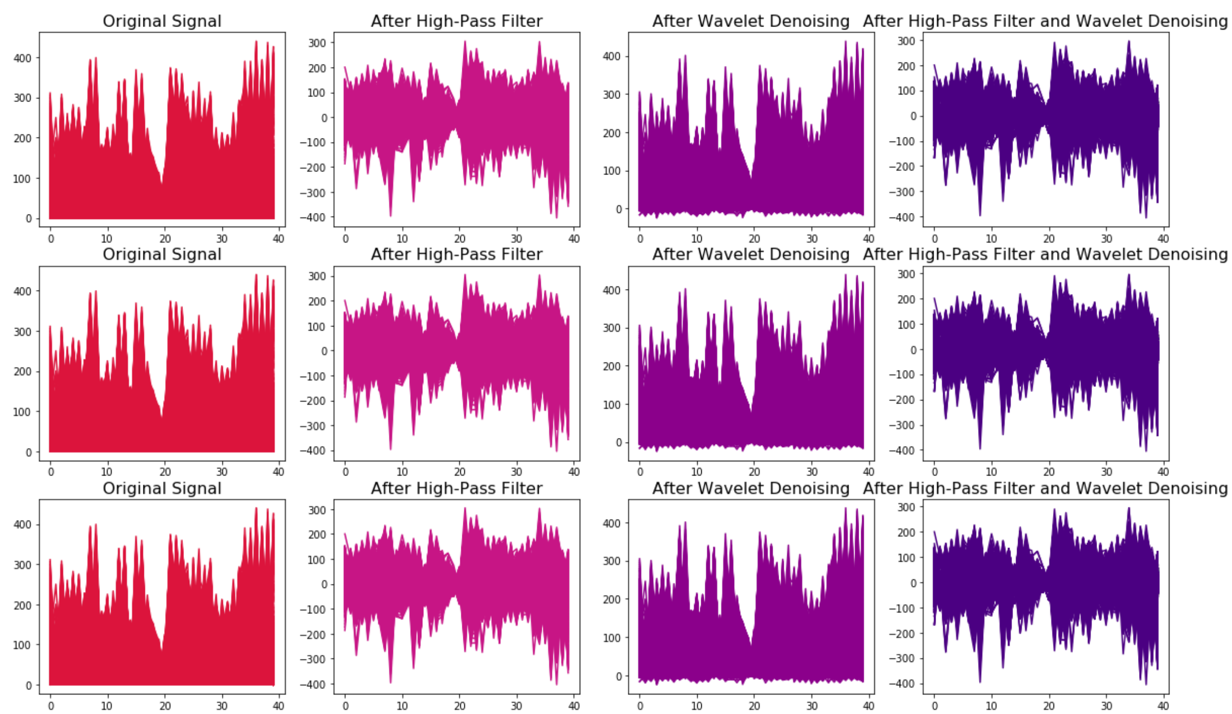
In [0]:

```
def maddest(d, axis=None):
    """The mean absolute deviation
    This calculates the mean of the absolute values of the deviations of the
    of the time series. It is a measure of entropy or disorder in the time se
    The greater the MAD value, the more disorderly and unpredictable the time

    return np.mean(np.absolute(d - np.mean(d, axis)), axis)

#====================================================================
def high_pass_filter(x, low_cutoff=1000, SAMPLE_RATE=SAMPLE_RATE):
    """
    From @randxie https://github.com/randxie/Kaggle-VSB-Baseline/blob/master/
    Modified to work with scipy version 1.1.0 which does not have the fs para
    """

    # nyquist frequency is half the sample rate https://en.wikipedia.org/wiki
    nyquist = 0.5 * SAMPLE_RATE
    norm_low_cutoff = low_cutoff / nyquist

    # Fault pattern usually exists in high frequency band. According to liter
    sos = butter(10, Wn=[norm_low_cutoff], btype='highpass', output='sos')
    filtered_sig = signal.sosfilt(sos, x)

    return filtered_sig

#=============================================================================
#after passing from the filter we will finally denoise the signal
def denoise_signal(x, wavelet='db4', level=1):
    """
    1. Adapted from waveletSmooth function found here:
    http://connor-johnson.com/2016/01/24/using-pywavelets-to-remove-high-freq
    2. Threshold equation and using hard mode in threshold as mentioned
    in section '3.2 denoising based on optimized singular values' from paper
    http://dspace.vsb.cz/bitstream/handle/10084/133114/VAN431_FEI_P1807_1801V
    """

    # Decompose to get the wavelet coefficients
    coeff = pywt.wavedec(x, wavelet, mode="per")

    # Calculate sigma for threshold as defined in http://dspace.vsb.cz/bitstr
    # As noted by @harshit92 MAD referred to in the paper is Mean Absolute De
    sigma = (1/0.6745) * maddest(coeff[-level])

    # Calculate the univeral threshold
    uthresh = sigma * np.sqrt(2*np.log(len(x)))
    coeff[1:] = (pywt.threshold(i, value=uthresh, mode='hard') for i in coeff

    # Reconstruct the signal using the thresholded coefficients
    return pywt.waverec(coeff, wavelet, mode='per')
```

In [0]:
```
#storing the denosied signals for all the regions
denoised_signals = []
for i in range(40):
    recon_signal = denoise_signal(high_pass_filter(signals[i], low_cutoff=100
    denoised_signals.append(recon_signal)
```

In [0]:

```python
#plotting for filter,denosing and then both of them combined
fig, ax = plt.subplots(nrows=3, ncols=4, figsize=(20, 12))

ax[0, 0].plot(signals[0], 'crimson')
ax[0, 0].set_title('Original Signal', fontsize=16)
ax[0, 1].plot(high_pass_filter(signals[0], low_cutoff=10000, SAMPLE_RATE=4000
ax[0, 1].set_title('After High-Pass Filter', fontsize=16)
ax[0, 2].plot(denoise_signal(signals[0]), 'darkmagenta')
ax[0, 2].set_title('After Wavelet Denoising', fontsize=16)
ax[0, 3].plot(denoise_signal(high_pass_filter(signals[0], low_cutoff=10000, S
ax[0, 3].set_title('After High-Pass Filter and Wavelet Denoising', fontsize=1

ax[1, 0].plot(signals[1], 'crimson')
ax[1, 0].set_title('Original Signal', fontsize=16)
ax[1, 1].plot(high_pass_filter(signals[1], low_cutoff=10000, SAMPLE_RATE=4000
ax[1, 1].set_title('After High-Pass Filter', fontsize=16)
ax[1, 2].plot(denoise_signal(signals[1]), 'darkmagenta')
ax[1, 2].set_title('After Wavelet Denoising', fontsize=16)
ax[1, 3].plot(denoise_signal(high_pass_filter(signals[1], low_cutoff=10000, S
ax[1, 3].set_title('After High-Pass Filter and Wavelet Denoising', fontsize=1

ax[2, 0].plot(signals[2], 'crimson')
ax[2, 0].set_title('Original Signal', fontsize=16)
ax[2, 1].plot(high_pass_filter(signals[2], low_cutoff=10000, SAMPLE_RATE=4000
ax[2, 1].set_title('After High-Pass Filter', fontsize=16)
ax[2, 2].plot(denoise_signal(signals[2]), 'darkmagenta')
ax[2, 2].set_title('After Wavelet Denoising', fontsize=16)
ax[2, 3].plot(denoise_signal(high_pass_filter(signals[2], low_cutoff=10000, S
ax[2, 3].set_title('After High-Pass Filter and Wavelet Denoising', fontsize=1

plt.show()
```



Part 2: Holt’s Linear Trend Model

It is an extension of simple exponential smoothing to allow forecasting of data with a trend. This method takes into account the trend of the dataset. The forecast function in this method is a function of level and trend. First, lets visualize the trend, seasonality and error in the series and then decompose the time series in four parts.

Observed, which is the original time series. Trend, which shows the trend in the time series, i.e., increasing or decreasing behaviour of the time series. Seasonal, which tells us about the seasonality in the time series. Residual, which is obtained by removing any trend or seasonality in the time series

For understaning about it and the code snippet I have used is:

1.https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/ (https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/)

2.https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-ii/ (https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-ii/)

3.https://www.kaggle.com/lampubhutia/timeseries-modelling-predicting-traffic-growth (https://www.kaggle.com/lampubhutia/timeseries-modelling-predicting-traffic-growth)

In [24]:
```
def initial_trend(series, slen):
    #here series is the signal we want
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
        return sum / slen
#=========================================================
def initial_seasonal_components(series, slen):
    seasonals = {}
    season_averages = []
    n_seasons = int(len(series)/slen)
    # compute season averages
    for j in range(n_seasons):
        season_averages.append(sum(series[slen*j:slen*j+slen])/float(slen))
    # compute initial values
    for i in range(slen):
        sum_of_vals_over_avg = 0.0
        for j in range(n_seasons):
            sum_of_vals_over_avg += series[slen*j+i]-season_averages[j]
        seasonals[i] = sum_of_vals_over_avg/n_seasons
    return seasonals
#=======================================================================
def triple_exponential_smoothing(series, slen, alpha, beta, gamma, n_preds):
    result = []
    seasonals = initial_seasonal_components(series, slen)
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
            result.append(series[0])
            continue
        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append((smooth + m*trend) + seasonals[i%slen])
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%sl
            result.append(smooth+trend+seasonals[i%slen])
    return result

#Holt Winters initialization of variables: # https://robjhyndman.com/hyndsigh
alpha = 0.3
beta = 0.15
gamma = 0.2
season_len = 24

#Prepare the features for all points for all clusters
predict_values_three =[]
predict_final = []
for r in range(0,40):
    predict_values_three = triple_exponential_smoothing(regions_cum[r][0:131€
    predict_final.append(predict_values_three[5:])
```

Out[24]:
```
[0,
 -0.15001182844932864,
 0.04973103441697209,
 0.1047856693471457,
 0.2207239432930307,
 -0.13916905851025202,
 0.0798387775307404,
 -0.26116276794373344,
 0.09459909749222546,
 0.17075822779303143]
```

Using the final features

In [0]:
```

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

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


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

# 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 w
# it is list of lists
tsne_weekday = []

# its an numbpy array, of shape (523960, 5)
# each row corresponds to an entry in out data
# for the first row we will have [f0,f1,f2,f3,f4] fi=number of pickups happen
# the second row will have [f1,f2,f3,f4,f5]
# the third row will have [f2,f3,f4,f5,f6]
# and so on...
tsne_feature = []
tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13099)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13099)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4
    # our prediction start from 5th 10min intravel since we need to have numb
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104],
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_tim
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [26]:
```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take
# and split it such that for every region we have 70% data in train and 30% i
# ordered date-wise for every region
print("size of train data :", int(13099*0.7))
print("size of test data :", int(13099*0.3))
```

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

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

In [0]:
```
# Getting the predictions of exponential moving averages to be used as a feat

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

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

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

In [41]:
```
print("Number of data clusters",len(train_features), "Number of data points i
print("Number of data clusters",len(train_features), "Number of data points i
```

```
Number of data clusters 40 Number of data points in trian data 9169 Each data p
oint contains 5 features
Number of data clusters 40 Number of data points in test data 3930 Each data po
int contains 5 features
```

In [0]:
```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps)
#Extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) f
tsne_train_flat_lat = [i[:9169] for i in tsne_lat]
tsne_train_flat_lon = [i[:9169] for i in tsne_lon]
tsne_train_flat_weekday = [i[:9169] for i in tsne_weekday]
tsne_train_flat_output = [i[:9169] for i in output]
tsne_train_flat_exp_avg = [i[:9169] for i in predict_list]
tsne_train_flat_triple_exp = [i[:9169] for i in predict_list_three]



# extracting the rest of the timestamp values i.e 30% of 12956 (total timesta
#Extracting the rest of the timestamp values i.e 30% of 12956 (total timestam
tsne_test_flat_lat = [i[9169:] for i in tsne_lat]
tsne_test_flat_lon = [i[9169:] for i in tsne_lon]
tsne_test_flat_weekday = [i[9169:] for i in tsne_weekday]
tsne_test_flat_output = [i[9169:] for i in output]
tsne_test_flat_exp_avg = [i[9169:] for i in predict_list]
#tsne_test_flat_double_exp = [i[9169:] for i in predict_list_two]
tsne_test_flat_triple_exp = [i[9169:] for i in predict_list_three]


# the above contains values in the form of list of lists (i.e. list of values
train_new_features = []
for i in range(0,40):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,40):
    test_new_features.extend(test_features[i])



# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]
tsne_train_lat = sum(tsne_train_flat_lat, [])
tsne_train_lon = sum(tsne_train_flat_lon, [])
tsne_train_weekday = sum(tsne_train_flat_weekday, [])
tsne_train_output = sum(tsne_train_flat_output, [])
tsne_train_exp_avg = sum(tsne_train_flat_exp_avg,[])
tsne_train_flat_triple_exp = sum(tsne_train_flat_triple_exp,[])

```

In [48]:
```
# Preparing the data frame for our train data
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1']
df_train = pd.DataFrame(data=train_new_features, columns=columns)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon
df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg
#df_train['double_exp'] = tsne_train_flat_double_exp
df_train['triple_exp'] = tsne_train_flat_triple_exp

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

```
(366760, 10)
```

In [0]:
```
#Mering the train and test df with fourier features train and test df
df_train = pd.concat([df_train, fourier_train_df], axis = 1)
df_test = pd.concat([df_test, fourier_test_df], axis = 1)
```

In [0]:
```
#Save the final dataframes along with output
df_test.to_csv("df_test.csv", index=None)
df_train.to_csv("df_train.csv", index=None)

import pickle
with open('tsne_train_output.pkl', 'wb') as file:
    pickle.dump(tsne_train_output, file)
with open('tsne_test_output.pkl', 'wb') as file:
    pickle.dump(tsne_test_output, file)
```

Machine Learning Models

We wiil use Linear Regression,Random Forest regression and XGBoost regression and tune the parameters using GridSearch Cross validation

Model1: Using Linear Regression and Tuning

In [0]:

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

#declaring the parameters to be tuned
#Declaring parameters
params = {'alpha':[0.0001,0.001,0.01,0.1],
          'penalty':['l2','l1'],
          'fit_intercept':[True,False]}

#Using SGRRegressor with squared loss, in order to fine tune he hyperparamete
start =datetime.datetime.now()
print('Hyperparameter tuning: \n')
model= SGDRegressor(loss='squared_loss', l1_ratio=0.15, max_iter=1000, tol=0.
                    verbose=0, epsilon=0.1, random_state=None, learning_rate=
                    validation_fraction=0.1, n_iter_no_change=5, warm_start=F
g_search =GridSearchCV(model, params, scoring='neg_mean_absolute_error',cv=3,
g_search.fit(df_train, tsne_train_output)
print('Time taken to perform Hyperparameter tuning :',datetime.datetime.now()

#Getting the best hyperparameter tuned model
best_model=g_search.best_estimator_
print("Best estimator: ",best_model)

#Fitting the best model to our training data
best_model.fit(df_train, tsne_train_output)
```

```
Hyperparameter tuning: 

Time taken to perform Hyperparameter tuning : 0:34:04.541462
Best estimator:  SGDRegressor(alpha=0.001, average=False, early_stopping=False,
epsilon=0.1,
             eta0=0.01, fit_intercept=False, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

Out[71]:

```
SGDRegressor(alpha=0.001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=False, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [65]:
```
from sklearn.linear_model import SGDRegressor
best_est = SGDRegressor(alpha=0.001, average=False, early_stopping=False, eps
             eta0=0.01, fit_intercept=False, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=Non
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
best_est.fit(df_train,tsne_train_output)
```

Out[65]:
```
SGDRegressor(alpha=0.001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=False, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [0]:
```
#Using the best model to make predictions
pred_test = (best_est.predict(df_test))
pred_train = (best_est.predict(df_train))
pred_train = [round(i) for i in pred_train]#rounding up for integer value
pred_test = [round(i) for i in pred_test]

```

Using Random Forest Regressor

In [8]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint #for uniform discrete random va
from scipy.stats import uniform

param_dist = {"n_estimators":sp_randint[500,600,800,1000,1200,1500,2000],#num
              "max_depth": sp_randint(2,9),#the depth of the the trees
              "min_samples_leaf": sp_randint(20,65)} #number of leafs
start = datetime.datetime.now()
clf = RandomForestRegressor(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=10,scoring='neg_mean_absolute_
rf_random.fit(df_train,tsne_train_output)
print('total time taken for computation is:',datetime.datetime.now() - start)
```

```
time taken to perform the hyperparameter tuning: 1:12:29.091710
```

In [52]:
```
model = RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=4,
           max_features='auto', max_leaf_nodes=100,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=25, min_samples_split=7,
           min_weight_fraction_leaf=0.0, n_estimators=1500, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)

model.fit(df_train,tsne_train_output)
```

Out[52]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=4, max_features='auto', max_leaf_nodes=100,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=25,
                      min_samples_split=7, min_weight_fraction_leaf=0.0,
                      n_estimators=1500, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [0]:
```
#Using the best model to make predictions
y_pred = model.predict(df_test)
rf_test_predictions = [round(value) for value in y_pred] #rounding the values
y_pred = model.predict(df_train)
rf_train_predictions = [round(value) for value in y_pred]
```

Using XgBoost Regressor

In [9]:
```
from xgboost import XGBRegressor
#Declaring parameters
params = {'learning_rate':[0.1,0.01,0.001,0.0001],
          'n_estimators':[250,500,750,1000,1500,2000,3000],
          'subsample':[0.6,0.7,0.8,0.9],
          'min_child_weight':[3,5,7,9],
          'reg_lambda':[100,200,300,400],
          'reg_alpha':[100,200,300, 400],
          'max_depth': [3,4,5,6,7,9],
          'colsample_bytree':[0.6,0.7,0.8],
          'gamma':[0,0.5,1]}

#Tuning hyperparameters
start =datetime.now()
print('Hyperparameter tuning: \n')
model= XGBRegressor(random_state=0,n_jobs=-1)
rsearch = RandomizedSearchCV(model,params,n_iter=20,scoring='neg_mean_absolut
rsearch.fit(df_train, tsne_train_output)
print('Time taken to perform Hyperparameter tuning :',datetime.now()-start)

#Getting the best hyperparameter tuned model
best_model=rsearch.best_estimator_
print("Best estimator: ",best_model)

#Fitting the best model to our training data
#best_model.fit(df_train, tsne_train_output)
```

In [58]:
```
best_xgb = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1
       colsample_bytree=0.8, gamma=0.5, learning_rate=0.1,
       max_delta_step=0, max_depth=9, min_child_weight=9, missing=None,
       n_estimators=1500, n_jobs=-1, nthread=None, objective='reg:linear',
       random_state=0, reg_alpha=100, reg_lambda=200, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.9)

best_xgb.fit(df_train,tsne_train_output)
```

Out[58]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0.5,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=9, min_child_weight=9, missing=None, n_estimators=1500,
             n_jobs=-1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=100, reg_lambda=200, scale_pos_weight=1, seed=None,
             silent=True, subsample=0.9, verbosity=1)
```

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

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'].val
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_predictic
train_mape.append((mean_absolute_error(tsne_train_output, xgb_train_predictic
train_mape.append((mean_absolute_error(tsne_train_output, pred_train))/(sum(t

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].value
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].va
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions)
test_mape.append((mean_absolute_error(tsne_test_output,pred_test))/(sum(tsne_
```

In [70]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("-------------------------------------------------------------------
print ("Baseline Model -                            Train: ",train_mape[0],"
print ("Exponential Averages Forecasting -          Train: ",train_mape[1],"
print ("Linear Regression -                        Train: ",train_mape[4],"
print ("Random Forest Regression -                  Train: ",train_mape[2],"
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------
-------------------------
Baseline Model -                             Train:  0.14870666996426116
Test:  0.14225522601041551
Exponential Averages Forecasting -           Train:  0.14121603560900353
Test:  0.13490049942819257
Linear Regression -                         Train:  6.0207618934980696e+16
Test:  5.87620067570374e+16
Random Forest Regression -                   Train:  0.10861209273180993      T
est:  0.10484934300782964
```

Error Metric Matrix

In [71]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("-------------------------------------------------------------------
print ("Baseline Model -                            Train: ",train_mape[0],"
print ("Exponential Averages Forecasting -          Train: ",train_mape[1],"
print ("Linear Regression -                        Train: ",train_mape[4],"
print ("Random Forest Regression -                  Train: ",train_mape[2],"
print ("XgBoost Regression -                        Train: ",train_mape[3],"
print ("-------------------------------------------------------------------
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------
-------------------------
Baseline Model -                             Train:  0.14870666996426116
Test:  0.14225522601041551
Exponential Averages Forecasting -           Train:  0.14121603560900353
Test:  0.13490049942819257
Linear Regression -                         Train:  6.0207618934980696e+16
Test:  5.87620067570374e+16
Random Forest Regression -                   Train:  0.10861209273180993      T
est:  0.10484934300782964
XgBoost Regression -                         Train:  0.07017864663012166
Test:  0.0875584710259383
--------------------------------------------------------------------------------
-------------------------
```

In this case study the business problem we have undertaken was to predict the number of pickups in a region for a 10 min interval,the interpretability of the model is not important here from a perspective of end user,the error metric we have chosen here is mean absolute percentage error.

Then we clustered the data using segmentation i.e by using the latitude and longitude of pickups and then broke the regions using daily pickups in 10 min interval ,the strategy we used here for clustering the data was that we wanted minimum inter cluster distance to be 0.5 and maximum cluster distance to be 2 miles as that's how much a taxi travels with average speed in a 10 min interval,also we smoothed the data as for intervals where number of pickups tend to be zero we imputated the average over interval values

If we notice carefully some 10 minute intervals has 0 pick ups. 0 is not very relevant when we train the model. So instead of keeping those values as it is, we will smooth them. The process is as follows. We have time binned regions of 10 minute intervals. Let's take three continuos intervals and let's say each of the intervals have 50,100 and 0 pickups respectively. Instead of taking 0 as the third value we will take the take the average of all the values and distribute it in the three regions as 50,50,50. In this way, each 30 minute interval will have the same number of pickups even though the pickups values changes in each of the three 10 minute interval.

Time Series: In the below graph, we can see repeated patterns like each day the number of cabs are highest during the office hours. It's least during midnight and increases as the day progresses. It decreases during noon time. Then again starts increasing during evening time. We will explore all such data dependendies using Fourier transformed features.

Fourier transform generally means decomposition of a wave into sum of multiple sine waves. Whenever there is are repeated patterns in data, we can leverage the most out of them by using Fourier transform. Using fourier transform we can decompose any given waveform(or a function) into it's constituent frequencies. The graph obtained after fourier transform will have unique frequencies and amplitudes corresonding to their most frequent occurences in the original wave.

Each fourier transformed features are basically multiple sine waves. Each sine wave has a specific time period, frequency and amplitude. From the time period we can easily get the frequencies for each of the sine waves. We are basically converting the time series data from time domain to frequency domain. We will plot these frequencies and it's corresponding amplitudes in a fourier transformed graph. The X axis will represent the frequencies of the sine waves and the Y axis will represent the correspondin amplitudes.

Then we used High band pass filter + Wavenet denosing along with Holt's exponential smoothening as part of feature extracting process.

I achieved the best MAPE with XGBRegressor - 0.0875