Taxi demand prediction in New York City

In [1]:

```
import dask.dataframe as dd#similar to pandas

import pandas as pd#pandas to create small dataframes

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

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

import time #Convert to unix time

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

# matplotlib: used to plot graphs
import matplotlib
# matplotlib.use('nbagg') : matplotlib uses this protocall which makes plots more user
 intractive like zoom in and zoom out
matplotlib.use('nbagg')
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots

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

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

# download migwin: https://mingw-w64.org/doku.php/download/mingw-builds
# install it in your system and keep the path, migw_path ='installed path'
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mingw64
\\bin'
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

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

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

```
C:\Users\Shamim Ahmed\Anaconda3\lib\site-packages\sklearn\ensemble\weight_
boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an internal
NumPy module and should not be imported. It will be removed in a future Nu
mPy release.
  from numpy.core.umath_tests import inner1d
```

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

We Have collected all yellow taxi trips data of jan-2016.

In [4]:

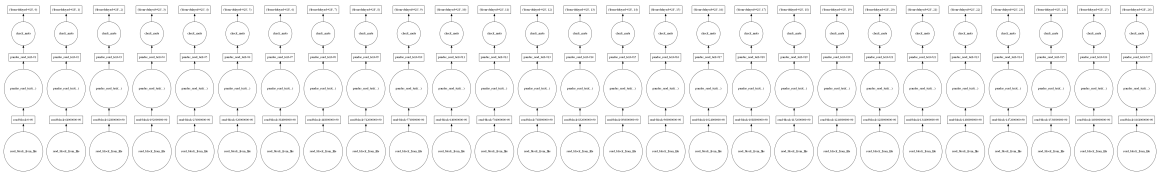
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('yellow_tripdata_2016-01.csv')
print(month.columns)
```

```
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_amou
nt',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [6]:

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

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

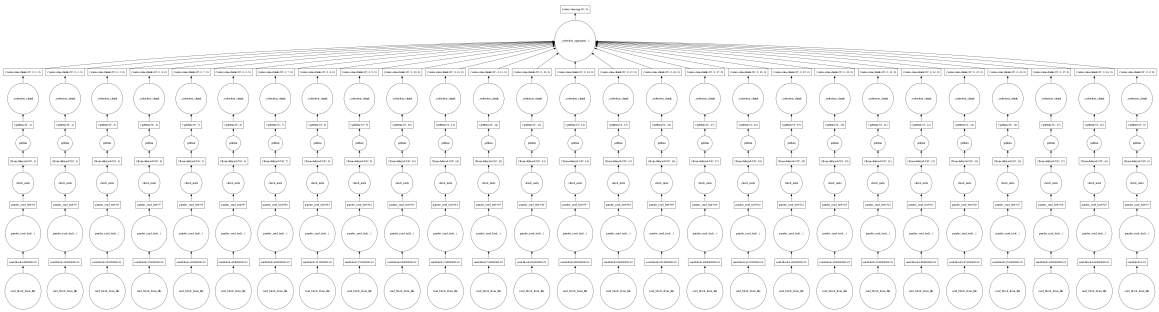
Out[6]:



In [7]:

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

Out[7]:



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.

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 [5]:

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

Out[5]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
0	2	2016-01-01 00:00:00	2016-01-01 00:00:00	2	1.10
1	2	2016-01-01 00:00:00	2016-01-01 00:00:00	5	4.90
2	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	10.5
3	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	4.75
4	2	2016-01-01 00:00:00	2016-01-01 00:00:00	3	1.76

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 [8]:

```
# Plotting pickup cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locations
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_latitude <= 40.5774)| \
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitude >= 40.9176))]

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

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

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

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

Out[8]:

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

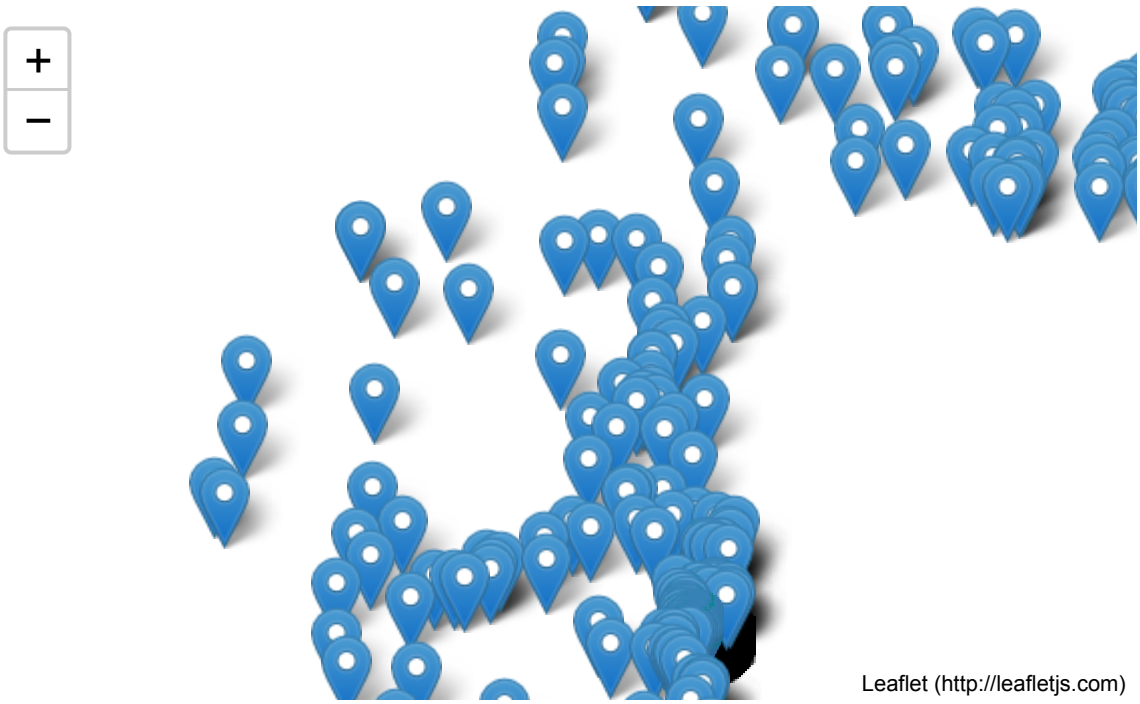
2. Dropoff Latitude & Dropoff Longitude

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

In [9]:

```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locations
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitude <= 40.5774)| \
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.9176))]

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

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

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

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

Out[9]:



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.**

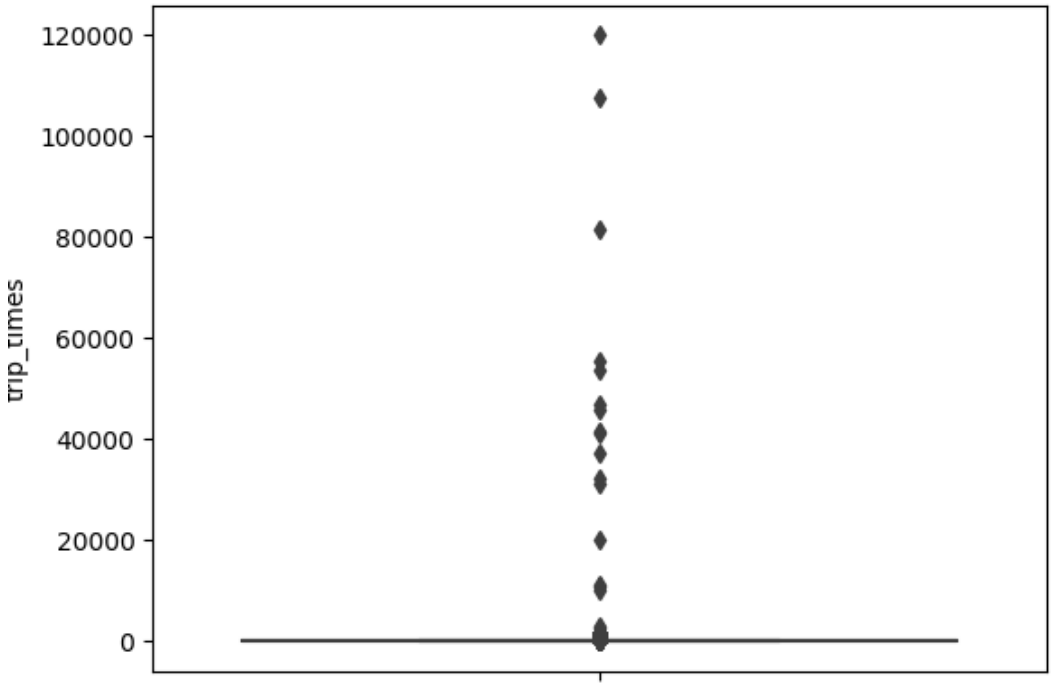
In [10]:

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

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



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

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

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

    return new_frame

# print(frame_with_durations.head())
#  passenger_count      trip_distance   pickup_longitude        pickup_latitude dropoff
_longitude      dropoff_latitude        total_amount    trip_times      pickup_times
Speed
#   1                 1.59          -73.993896              40.750111       -73.974
785             40.750618               17.05            18.050000      1.421329e+09
5.285319
#   1                 3.30            -74.001648              40.724243       -73.994
415             40.759109               17.80           19.833333       1.420902e+09
9.983193
#   1                 1.80            -73.963341              40.802788       -73.951
820             40.824413               10.80           10.050000       1.420902e+09
10.746269
#   1                 0.50            -74.009087              40.713818       -74.004
326             40.719986               4.80            1.866667        1.420902e+09
16.071429
#   1                 3.00            -73.971176              40.762428       -74.004
```

```
181             40.742653               16.30           19.316667       1.420902e+09
9.318378
frame_with_durations = return_with_trip_times(month)
```

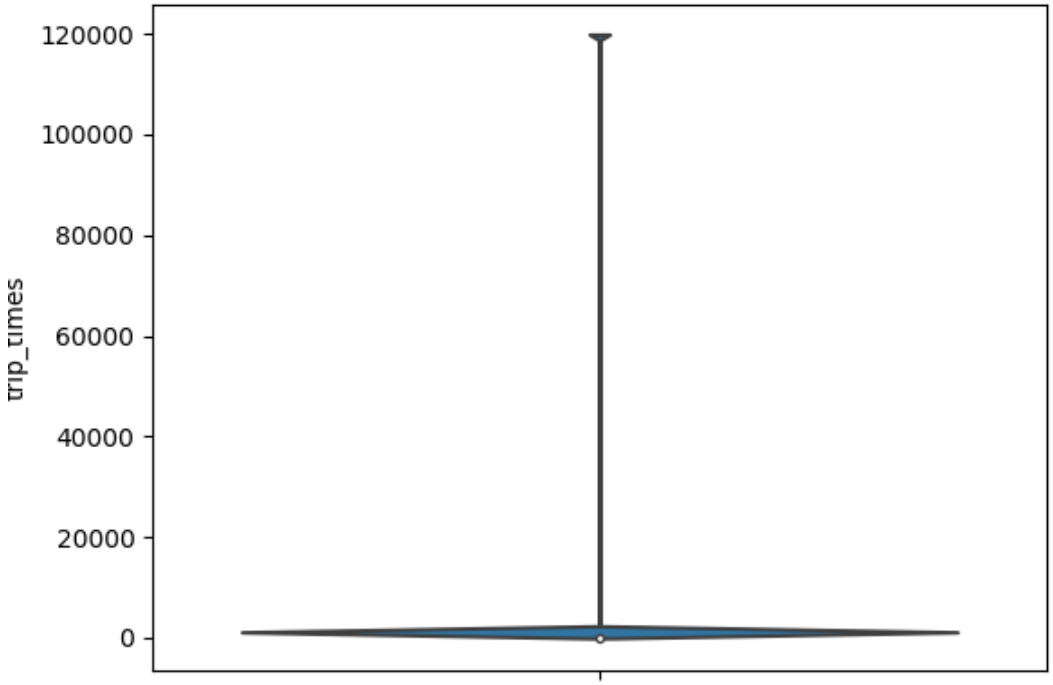
In [12]:

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



In [15]:

```
sns.violinplot(y="trip_times", data =frame_with_durations)
plt.show()
```



In [13]:

```
#calculating 0-100th percentile to find a the correct percentile value for removal of o
utliers
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 -43.95
10 percentile value is 3.933333333333333
20 percentile value is 5.566666666666666
30 percentile value is 7.1
40 percentile value is 8.683333333333334
50 percentile value is 10.466666666666667
60 percentile value is 12.55
70 percentile value is 15.2
80 percentile value is 18.933333333333334
90 percentile value is 25.516666666666666
100 percentile value is  119912.7
```

In [14]:

```
#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])
```

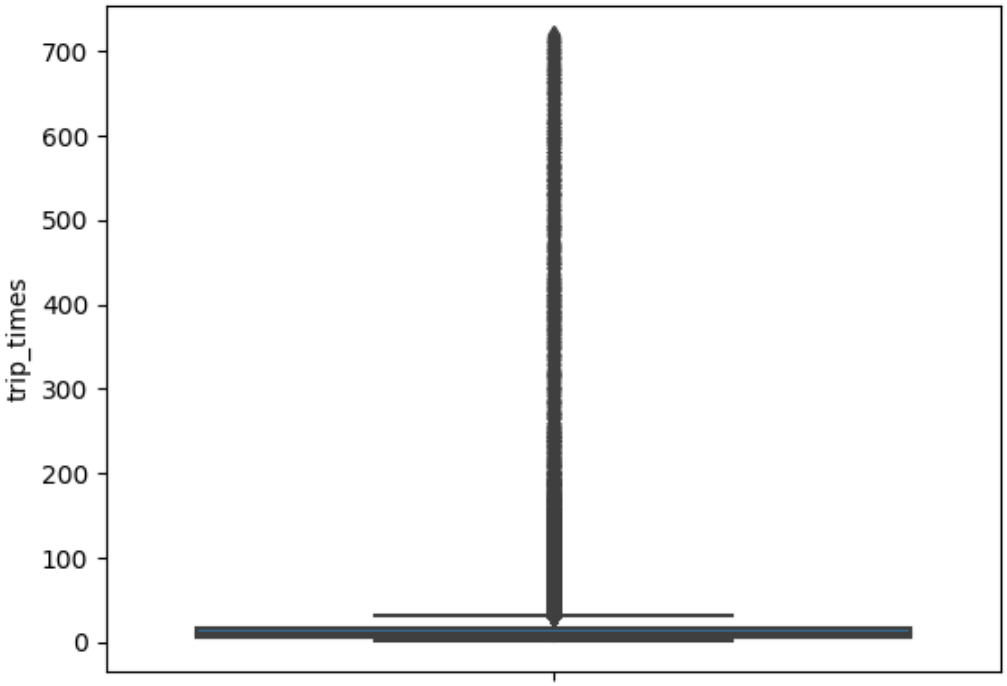
```
90 percentile value is 25.516666666666666
91 percentile value is 26.55
92 percentile value is 27.733333333333334
93 percentile value is 29.116666666666667
94 percentile value is 30.733333333333334
95 percentile value is 32.7
96 percentile value is 35.233333333333334
97 percentile value is 38.666666666666664
98 percentile value is 43.916666666666664
99 percentile value is 53.9
100 percentile value is  119912.7
```

In [15]:

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

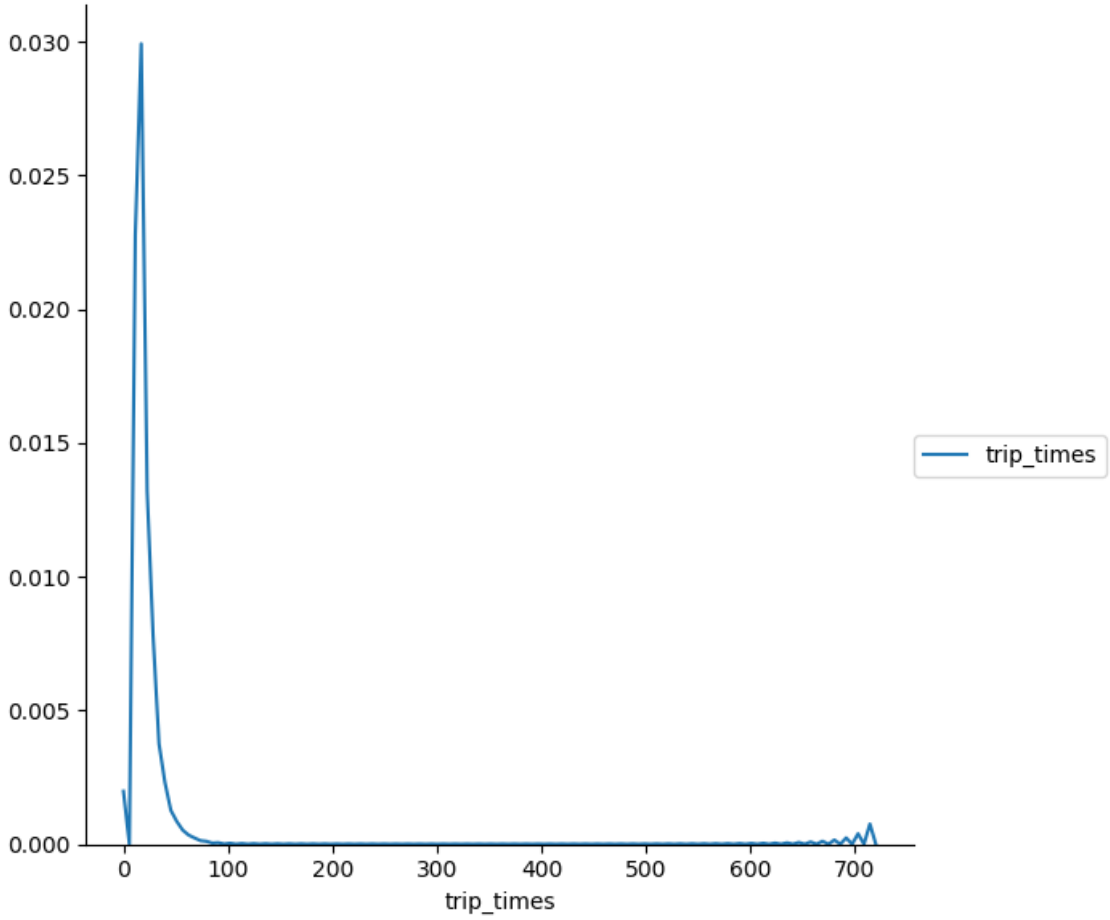
In [16]:

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



In [21]:

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

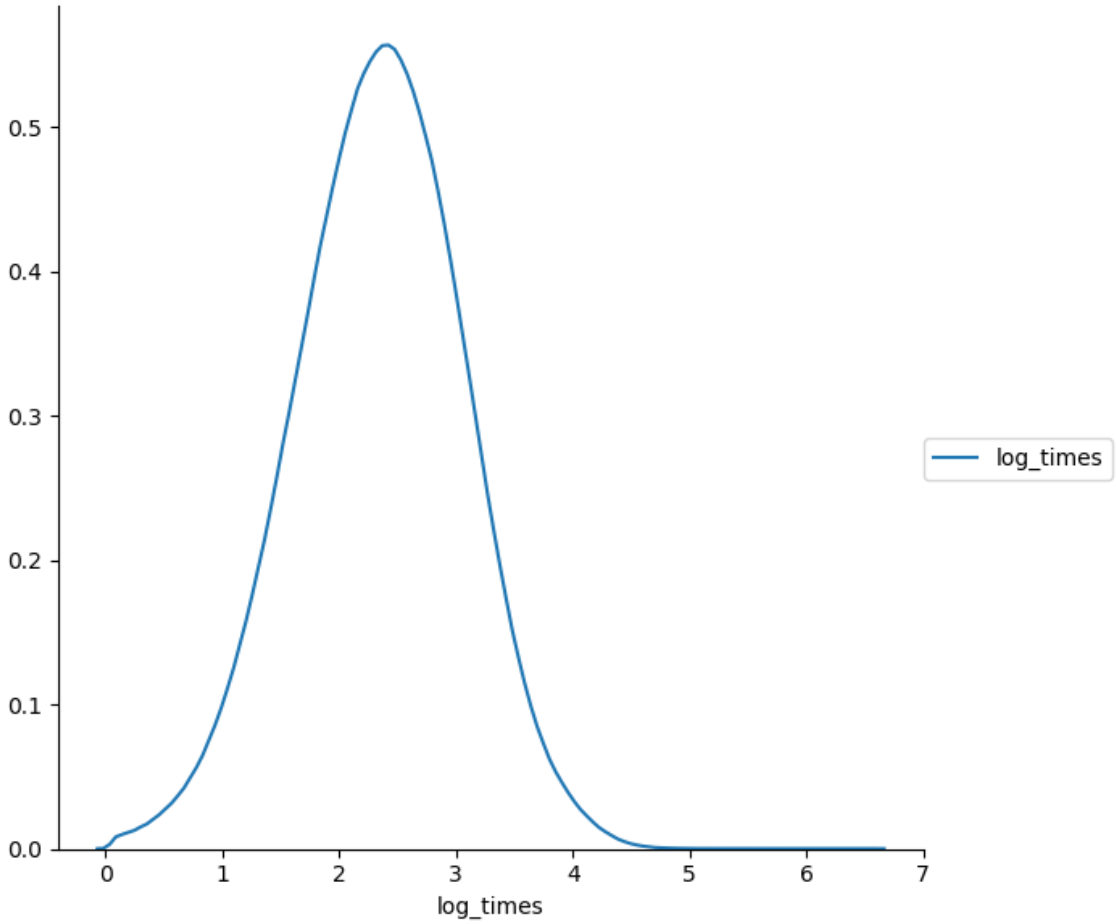


In [17]:

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

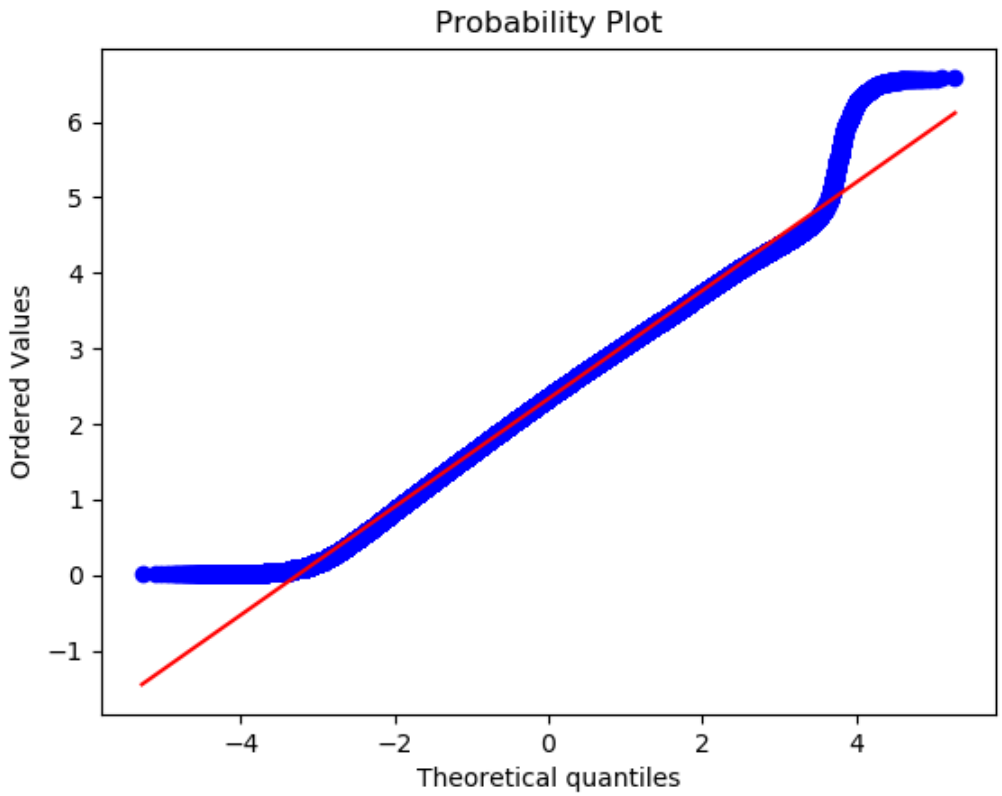
In [18]:

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



In [19]:

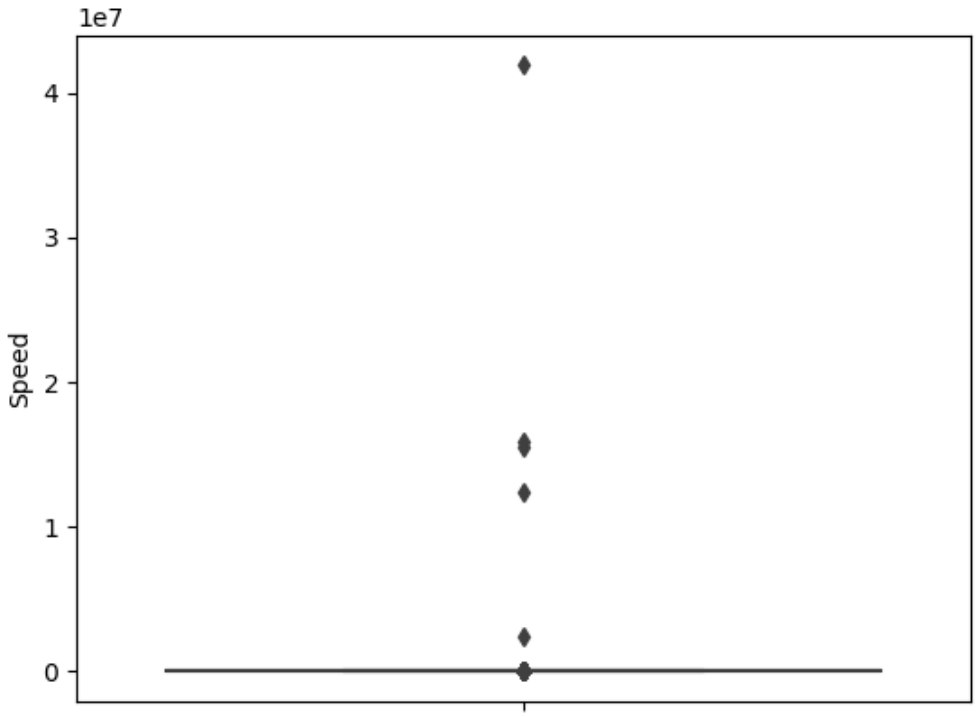
```
#Q-Q plot for checking if trip-times is log-normal
import scipy
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [20]:

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



In [21]:

```
#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 6.011787819253438
20 percentile value is 7.390029325513196
30 percentile value is 8.49689440993789
40 percentile value is 9.545454545454543
50 percentile value is 10.638522427440636
60 percentile value is 11.87948350071736
70 percentile value is 13.432835820895523
80 percentile value is 15.6734693877551
90 percentile value is 20.035906642728904
100 percentile value is  41917233.8028169
```

In [22]:

```
#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 20.035906642728904
91 percentile value is 20.77922077922078
92 percentile value is 21.62544169611307
93 percentile value is 22.594142259414227
94 percentile value is 23.720930232558143
95 percentile value is 25.043478260869566
96 percentile value is 26.641366223908914
97 percentile value is 28.652097902097903
98 percentile value is 31.304347826086957
99 percentile value is 35.33428165007113
100 percentile value is  41917233.8028169
```

In [23]:

```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.
8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100
))]))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 35.33428165007113
99.1 percentile value is 35.89123867069487
99.2 percentile value is 36.49769585253456
99.3 percentile value is 37.17507418397626
99.4 percentile value is 37.91878172588833
99.5 percentile value is 38.762376237623755
99.6 percentile value is 39.768642447418735
99.7 percentile value is 41.019230769230774
99.8 percentile value is 42.63212435233161
99.9 percentile value is 45.163636363636364
100 percentile value is  41917233.8028169
```

In [24]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0) & (fr
ame_with_durations.Speed<45.1636)]
```

In [25]:

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

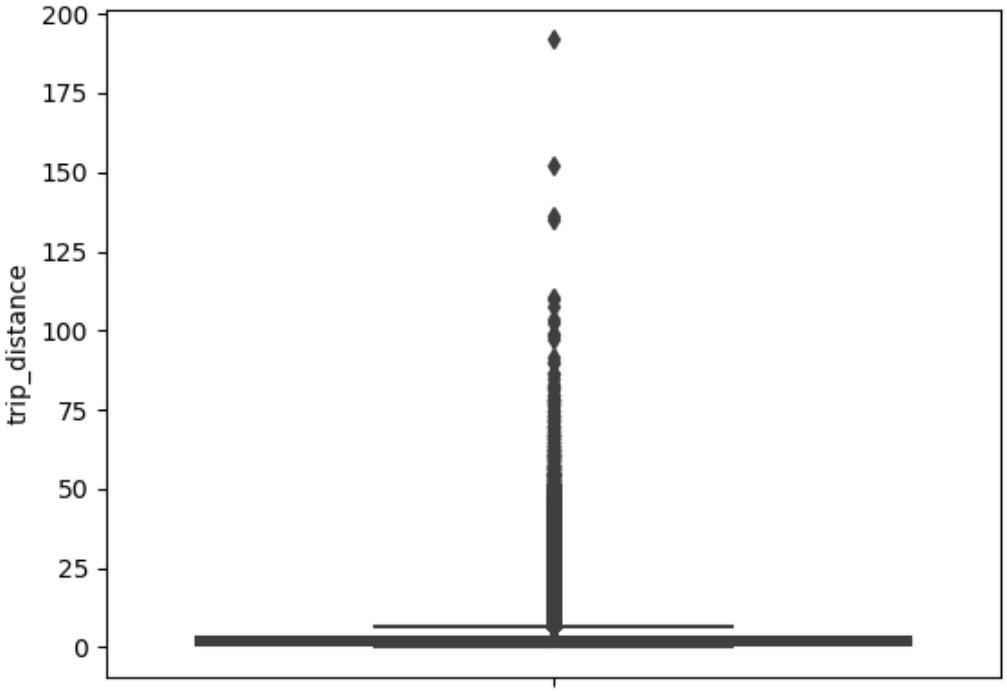
Out[25]:

```
12.077450908637793
```

The avg speed in Newyork speed is 12.077miles/hr, so a cab driver can travel 2 miles per 10min on avg.

In [26]:

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



In [27]:

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

```
0 percentile value is 0.01
10 percentile value is 0.66
20 percentile value is 0.9
30 percentile value is 1.1
40 percentile value is 1.38
50 percentile value is 1.69
60 percentile value is 2.08
70 percentile value is 2.64
80 percentile value is 3.7
90 percentile value is 6.5
100 percentile value is  191.9
```

In [28]:

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

```
90 percentile value is 6.5
91 percentile value is 7.1
92 percentile value is 7.88
93 percentile value is 8.7
94 percentile value is 9.48
95 percentile value is 10.3
96 percentile value is 11.4
97 percentile value is 13.5
98 percentile value is 16.97
99 percentile value is 18.59
100 percentile value is  191.9
```

In [29]:

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

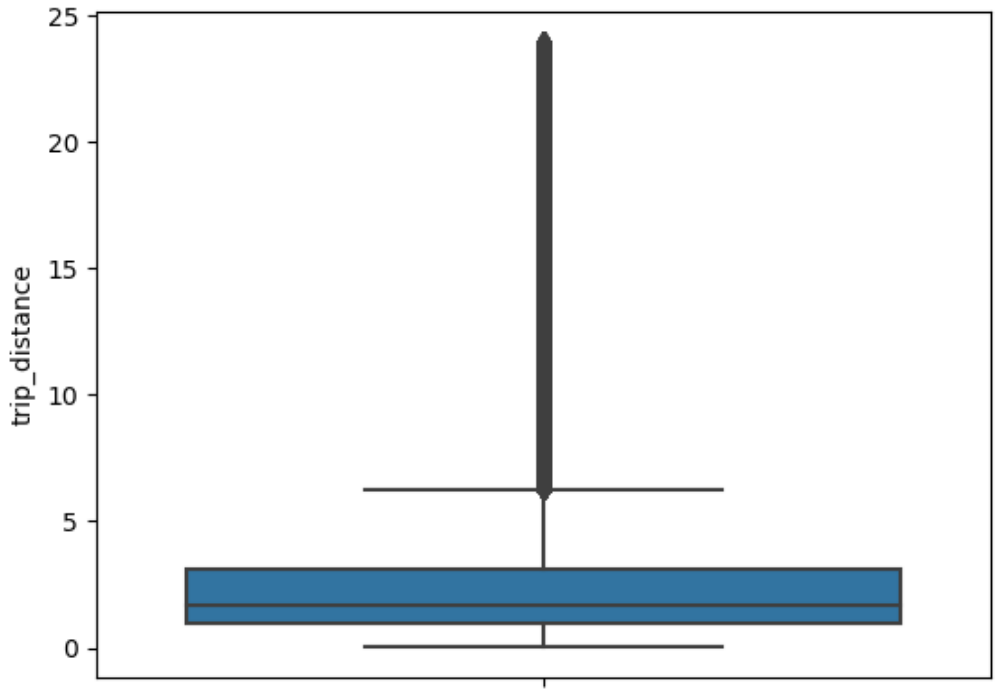
```
99.0 percentile value is 18.59
99.1 percentile value is 18.8
99.2 percentile value is 19.01
99.3 percentile value is 19.3
99.4 percentile value is 19.61
99.5 percentile value is 20.0
99.6 percentile value is 20.5
99.7 percentile value is 21.0
99.8 percentile value is 21.78
99.9 percentile value is 23.89
100 percentile value is  191.9
```

In [30]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_distance>
0) & (frame_with_durations.trip_distance<24)]
```

In [31]:

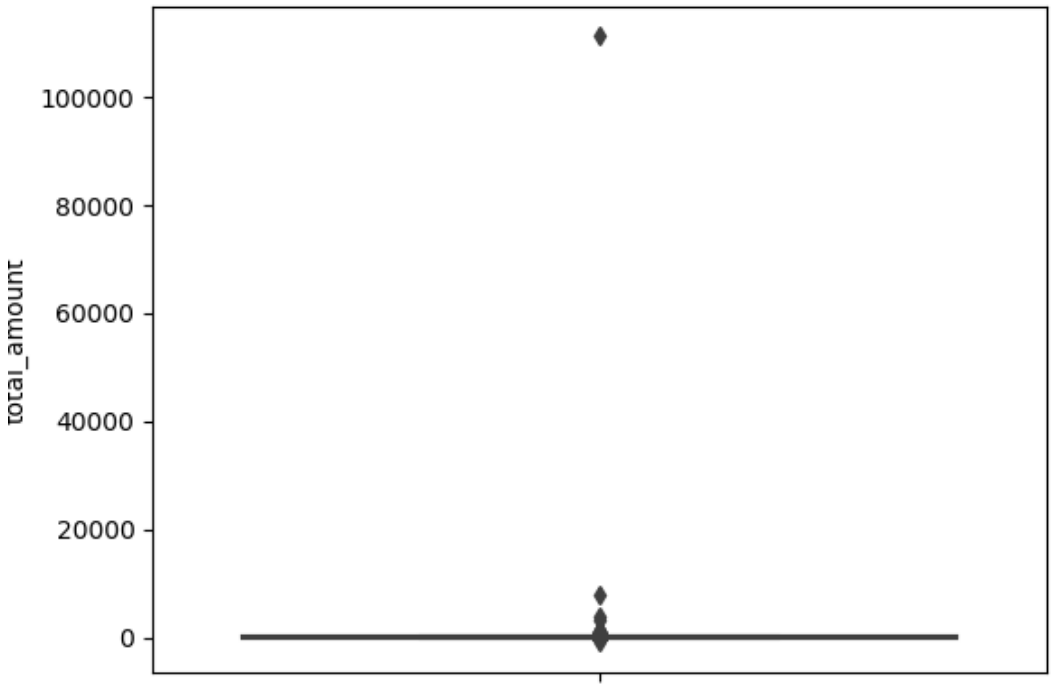
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [32]:

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



In [33]:

```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80,90,10
0
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 -958.4
10 percentile value is 6.8
20 percentile value is 7.8
30 percentile value is 8.8
40 percentile value is 10.3
50 percentile value is 11.62
60 percentile value is 13.3
70 percentile value is 15.36
80 percentile value is 19.3
90 percentile value is 28.08
100 percentile value is  111271.65
```

In [34]:

```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,98,99,1
00
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 28.08
91 percentile value is 30.14
92 percentile value is 32.54
93 percentile value is 35.38
94 percentile value is 38.8
95 percentile value is 42.36
96 percentile value is 46.8
97 percentile value is 52.8
98 percentile value is 58.8
99 percentile value is 69.99
100 percentile value is  111271.65
```

In [35]:

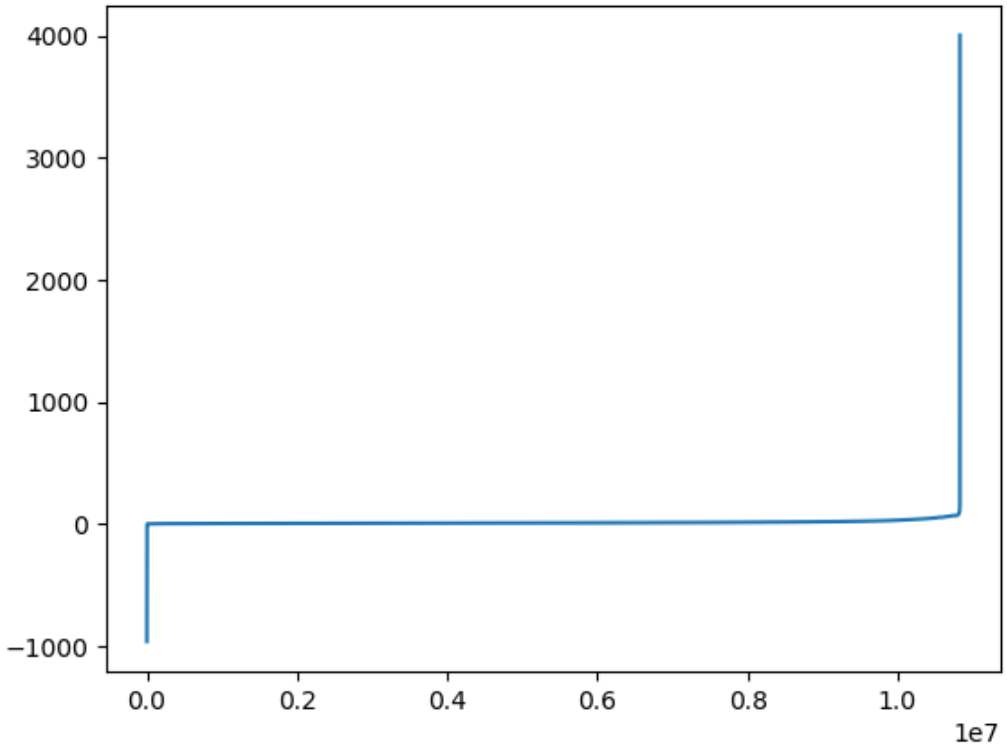
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,9
9.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100
))]))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 69.99
99.1 percentile value is 69.99
99.2 percentile value is 70.01
99.3 percentile value is 70.01
99.4 percentile value is 70.01
99.5 percentile value is 70.01
99.6 percentile value is 72.89
99.7 percentile value is 72.96
99.8 percentile value is 79.3
99.9 percentile value is 93.36
100 percentile value is  111271.65
```

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

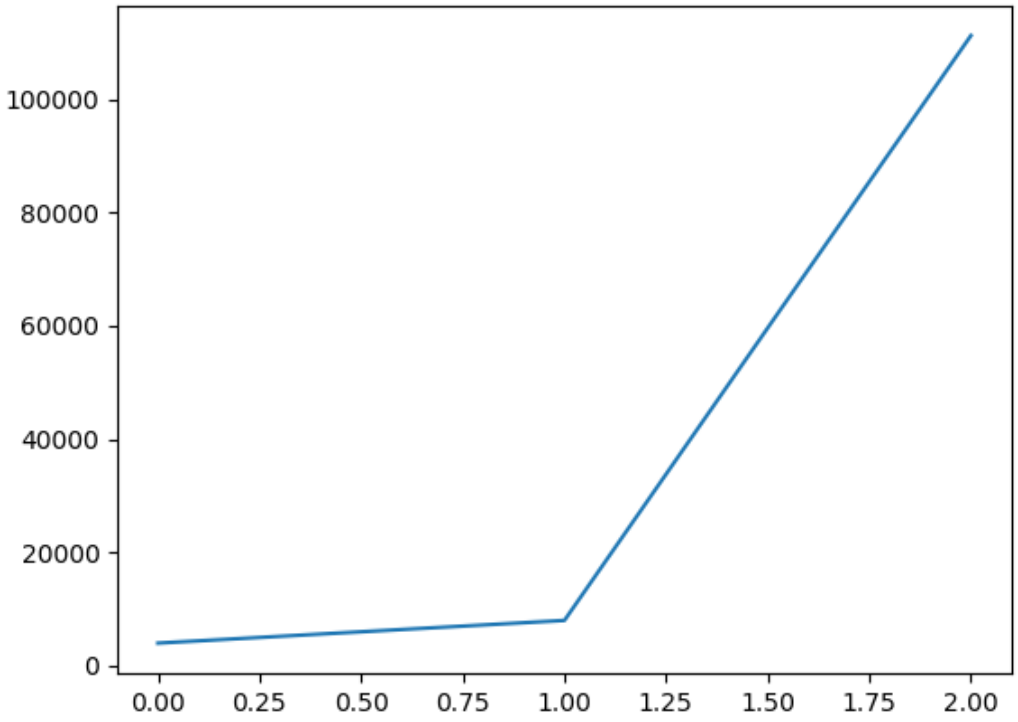
In [36]:

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



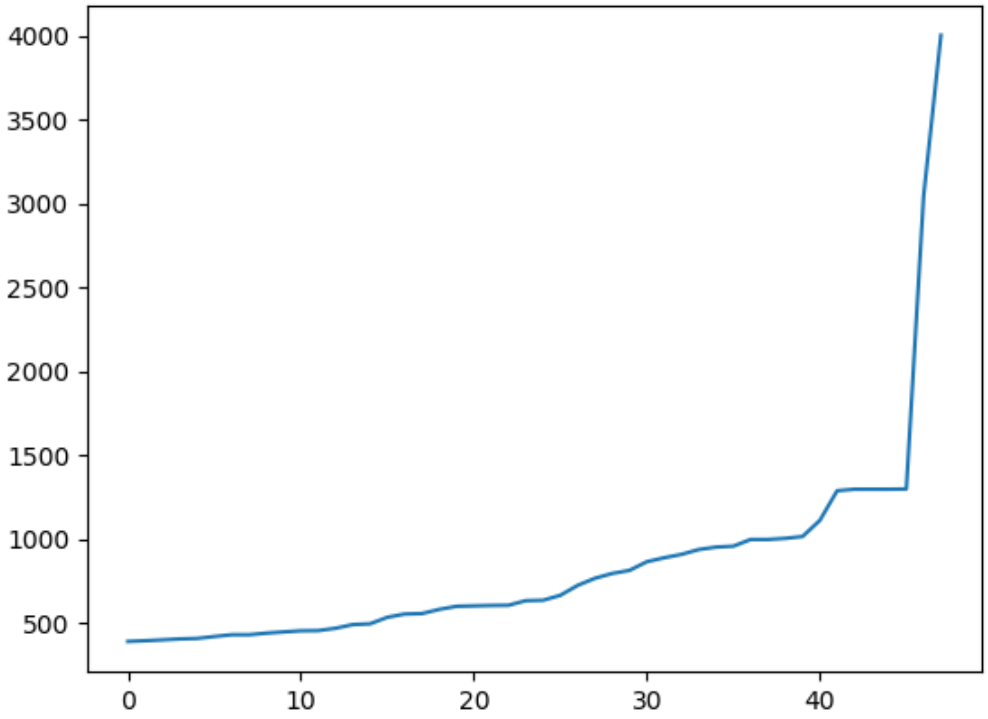
In [37]:

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



In [38]:

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



Remove all outliers/erronous points.

In [ ]:

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


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


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


    temp_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 2
4)]
    d = temp_frame.shape[0]
    print ("Number of outliers from trip distance analysis:",(a-d))

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

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


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff
_longitude <= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_lat
itude <= 40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_lati
tude >= 40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_lat
itude <= 40.9176))]

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 24
)]
    new_frame = new_frame[(new_frame.Speed < 45.16) & (new_frame.Speed > 0)]
    new_frame = new_frame[(new_frame.total_amount <1000) & (new_frame.total_amount >0)]

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

In [40]:

```
print ("Removing outliers in the month of Jan-2016")
print ("----")
frame_with_durations_outliers_removed = remove_outliers(frame_with_durations)
print("fraction of data points that remain after removing outliers", float(len(frame_wi
th_durations_outliers_removed))/len(frame_with_durations))
```

```
Removing outliers in the month of Jan-2016
----
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: 76825
Number of outliers from speed analysis: 21047
Number of outliers from fare analysis: 4991
```

In [42]:

```
# Saving in pickle file
frame_with_durations_outliers_removed.to_pickle('pickles/frame_with_durations_outliers_
removed')
```

Data-preperation

Clustering/Segmentation

In [33]:

```
# loading pickle file
frame_with_durations_outliers_removed = pd.read_pickle('pickles/frame_with_durations_ou
tliers_removed')
```

In [34]:

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

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

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fi
t(coords)
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with
_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

# we need to choose number of clusters so that, there are more number of cluster region
s
#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 =  0.9789656998887113
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 15.0
Min inter-cluster distance =  0.753506573441802
---
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 =  0.4996848900177776
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 10.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 30.0
Min inter-cluster distance =  0.39027384084042077
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 38.0
Min inter-cluster distance =  0.37148421681854055
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 15.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 45.0
Min inter-cluster distance =  0.28149955081065253
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 52.0
Min inter-cluster distance =  0.18085758995776036
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 20.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 60.0
Min inter-cluster distance =  0.17300636125508823
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 25.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 65.0
```

```
Min inter-cluster distance =  0.19254623756729097
---
```

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 [35]:

```
# if check for the 50 clusters you can observe that there are two clusters with only 0.
3 miles apart from each other
# so we choose 40 clusters for solve the further problem

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

Plotting the cluster centers:

In [36]:

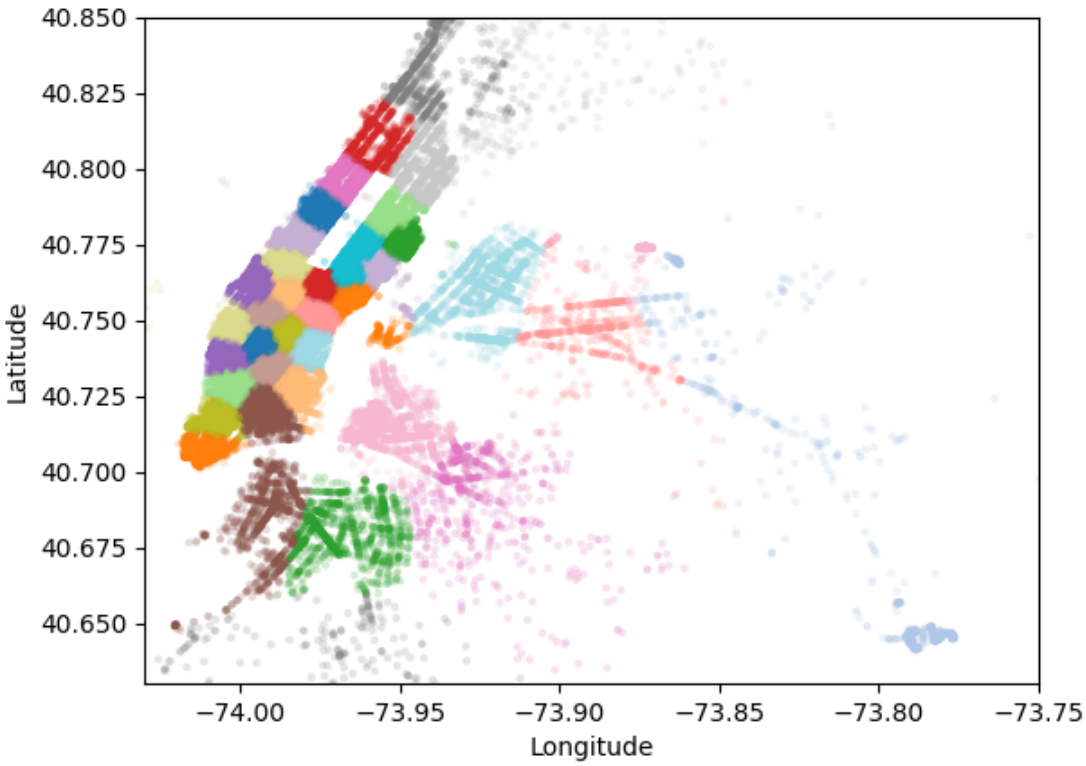
```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])), popup=(str(clust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

Out[36]:

Plotting the clusters:

In [37]:

```
#Visualising the clusters on a map
def plot_clusters(frame):
    city_long_border = (-74.03, -73.75)
    city_lat_border = (40.63, 40.85)
    fig, ax = plt.subplots(ncols=1, nrows=1)
    ax.scatter(frame.pickup_longitude.values[:100000], frame.pickup_latitude.values[:10
0000], s=10, lw=0,
               c=frame.pickup_cluster.values[:100000], cmap='tab20', alpha=0.2)
    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [64]:

```
## In binning we are converting the timestamps to 10 minute bins.
## We convert our data to clusters using the kmeans center points and 10 min bins.

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

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

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

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

In [ ]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_dur
ations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
jan_2016_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2016)
jan_2016_groupby = jan_2016_frame[['pickup_cluster','pickup_bins','trip_distance']].gro
upby(['pickup_cluster','pickup_bins']).count()
```

In [ ]:

```
# 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_2016_frame.head()
```

In [ ]:

```
# hear the trip_distance represents the number of pickups that are happend in that part
icular 10min intravel
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels 24*31*
60/10 =4464bins)
jan_2016_groupby.head()
```

In [ ]:

```
jan_2016_frame.to_pickle('pickles/jan_2016_frame')
jan_2016_groupby.to_pickle('pickles/jan_2016_groupby')
```

Smoothing

In [ ]:

```
jan_2016_frame = pd.read_pickle('pickles/jan_2016_frame')
jan_2016_groupby = pd.read_pickle('pickles/jan_2016_groupby')
```

In [ ]:

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

# for each cluster region we will collect all the indices of 10min intravels in which t
he pickups are happened
# we got an observation that there are some pickpbins that doesnt have any pickups
def return_unq_pickup_bins(frame):
    values = []
    for i in range(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 [ ]:

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

jan_2016_unique = return_unq_pickup_bins(jan_2016_frame)
```

In [ ]:

```
# for each cluster number of 10min intravels with 0 pickups
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",4464 -
len(set(jan_2016_unique[i])))
    print('-'*60)
```

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 [ ]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 0 to the smoothed data
# we finally return smoothed data
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,40):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [ ]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discus
sed in the above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already visited/res
olved
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pick
up bin if it exists
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pic
kup-bin value which has a pickup value
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be miss
ing,hence we have no right-limit here
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.0/((ri
ght_hand_limit-i)+2)*1.0
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be mi
ssing,hence we have no left-limit here
                    right_hand_limit=0
                    for j in range(i,4464):
```

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

In [ ]:

```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pickups
that are happened
jan_2016_fill = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)

#Smoothing Missing values of Jan-2015
jan_2016_smooth = smoothing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)
```

In [ ]:

```
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*30*60/10 = 4320
# for each cluster we will have 4464 values, therefore 40*4464 = 178560 (length of the
 jan_2015_fill)
print("number of 10min intravels among all the clusters ",len(jan_2016_fill))
```

In [ ]:

```
# 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_2016_fill[4464:8920], label="zero filled values")
plt.plot(jan_2016_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```

In [ ]:

```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled with z
ero
#jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_uniqu
e

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

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

for i in range(0,40):
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)])
    # print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 13104
```

In [ ]:

```
len(jan_2016_unique)
```

In [ ]:

```
jan_2016_groupby['trip_distance'].shape
```

Time series and Fourier Transforms

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

```
len(jan_2016_smooth)
```

Modelling: Baseline Models

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

1. Using Ratios of the 2016 data to the 2015 data i.e $R_t = P_t^{2016} / P_t^{2015}$
2. Using Previous known values of the 2016 data itself to predict the future values

Simple Moving Averages

The First Model used is the Moving Averages Model which uses the previous n values in order to predict the next value

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

In [ ]:

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


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

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

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

In [39]:

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

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

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

Weighted Moving Averages

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

Weighted Moving Averages using Ratio Values -

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

In [40]:

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

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

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

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

In [41]:

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

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

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

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

Exponential Weighted Moving Averages

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

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

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

In [42]:

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

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

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

In [43]:

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

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

In [44]:

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

Comparison between baseline models

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

In [45]:

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------
------------------------------
Moving Averages (Ratios) -                              MAPE:  0.0       MS
E:  0.0
Moving Averages (2016 Values) -                         MAPE:  0.1425637079
1772408        MSE:  173.31759072580644
--------------------------------------------------------------------------------
------------------------------
Weighted Moving Averages (Ratios) -                     MAPE:  0.0       MS
E:  0.0
Weighted Moving Averages (2016 Values) -                MAPE:  0.1352966729
8832504       MSE:  160.39991039426522
--------------------------------------------------------------------------------
------------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.0       MSE:
0.0
Exponential Moving Averages (2016 Values) -         MAPE:  0.1351395967755
6814       MSE:  158.959811827957
```

In [46]:

```
## https://github.com/anirudhpillai16/Time-Series-Analysis/blob/master/Time%20Series%20
Analytics%20Vidhya.ipynb
```

Regression Models

Train-Test Split

Before we start predictions using the tree based regression models we take JAN 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 [47]:

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

4464

In [48]:

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

Out[48]:

4464

In [49]:

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

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

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

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


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


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

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

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


tsne_feature = [0]*number_of_time_stamps
for i in range(0,20):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*4459)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*4459)
```

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

In [50]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_w
eekday[0]) == 20*4459 == len(output)*len(output[0])
```

Out[50]:

True

In [51]:

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

# upto now we computed 8 features for every data point that starts from 50th min of the
day
# 1. cluster center lattitude
# 2. cluster center longitude
# 3. day of the week
# 4. f_t_1: number of pickups that are happened previous t-1th 10min intravel
# 5. f_t_2: number of pickups that are happened previous t-2th 10min intravel
# 6. f_t_3: number of pickups that are happened previous t-3th 10min intravel
# 7. f_t_4: number of pickups that are happened previous t-4th 10min intravel
# 8. f_t_5: number of pickups that are happened previous t-5th 10min intravel

# from the baseline models we said the exponential weighted moving avarage gives us the
best error
# we will try to add the same exponential weighted moving avarage at t as a feature to
 our data
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.3

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

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x
13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], .. 40 lsits]
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,20):
    for i in range(0,4464):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

Fourier Transformation Features:

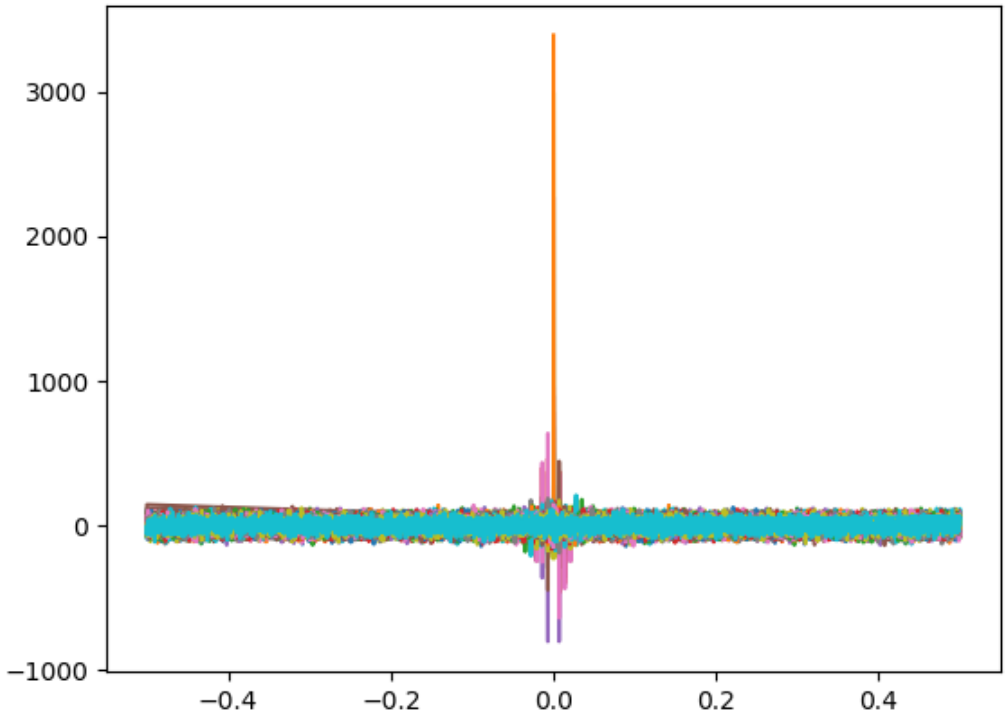
FFT (Fast Fourier Transform) refers to a way the discrete Fourier Transform (DFT) can be calculated efficiently, by using symmetries in the calculated terms. The symmetry is highest when n is a power of 2, and the transform is therefore most efficient for these sizes.

The DFT is defined, with the conventions used in this implementation, in the documentation for the numpy.fft module.

In this example, real input has an FFT which is Hermitian, i.e., symmetric in the real part and anti-symmetric in the imaginary part

In [100]:

```
import matplotlib.pyplot as plt
t =  np.asarray(regions_cum)
sp = np.fft.fft(np.sin(t))
freq = np.fft.fftfreq(t.shape[-1])
plt.plot(freq, sp.T.real, freq, sp.T.imag)
plt.show()
```



In [285]:

```
#freq.shape
```

Out[285]:

(4464,)

In [286]:

```
a = np.abs(sp.T)
index = np.argsort(-a)[1:]
my_freq = []
my_sp = []
for i in range(0,40):
    sp = []
    frequency = []
    for i in range(0,9,2):
        sp.append(a[index[i]])
        frequency.append(freq[index[i]])
    for j in range(4459):
        my_sp.append(sp)
        my_freq.append(frequency)
```

In [53]:

```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 1 months o
f 2016 pickup data
# and split it such that for every` region we have 70% data in train and 30% in test,
# ordered date-wise for every region
print("size of train data :", int(4459*0.7))
print("size of test data :", int(4459*0.3))
```

```
size of train data : 3121
size of test data : 1337
```

In [54]:

```
# extracting first 3121 timestamp values i.e 70% of 4459 (total timestamps) for our tra
ining data
train_features =  [tsne_feature[i*4459:(4459*i+3121)] for i in range(0,20)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(4459*(i))+3121:4459*(i+1)] for i in range(0,20)]
```

In [55]:

```
'''
train_freq = [my_freq[4459*i:(4459*i+3121)] for i in range(0,40)]
train_sp = [my_sp[4459*i:(4459*i+3121)] for i in range(0,40)]

test_freq = [my_freq[4459*i + 3121: 4459*(i+1)] for i in range(0,40)]
test_sp = [my_sp[4459*i:4459*(i+1)] for i in range(0,40)]

'''
```

Out[55]:

```
'\ntrain_freq = [my_freq[4459*i:(4459*i+3121)] for i in range(0,40)]\ntrai
n_sp = [my_sp[4459*i:(4459*i+3121)] for i in range(0,40)]\n\ntest_freq =
[my_freq[4459*i + 3121: 4459*(i+1)] for i in range(0,40)]\ntest_sp = [my_s
p[4459*i:4459*(i+1)] for i in range(0,40)]\n\n'
```

In [56]:

```
train_top_freq = [freq[4459*i:(4459*i+3121)] for i in range(0,20)]
train_top_amp = [sp[4459*i:(4459*i+3121)] for i in range(0,20)]
test_top_freq = [freq[4459*i + 3121: 4459*(i+1)] for i in range(0,20)]
test_top_amp = [sp[4459*i:4459*(i+1)] for i in range(0,20)]
```

In [58]:

```
print("Number of data clusters",len(train_features), "Number of data points in trian da
ta", len(train_features[0]), "Each data point contains", len(train_features[0][0]),"fea
tures")
print("Number of data clusters",len(train_features), "Number of data points in test dat
a", len(test_features[0]), "Each data point contains", len(test_features[0][0]),"featur
es")
```

```
Number of data clusters 20 Number of data points in trian data 3121 Each d
ata point contains 5 features
Number of data clusters 20 Number of data points in test data 1338 Each da
ta point contains 5 features
```

In [59]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our tr
aining data
tsne_train_flat_lat = [i[:3121 ] for i in tsne_lat]
tsne_train_flat_lon = [i[:3121 ] for i in tsne_lon]
tsne_train_flat_weekday = [i[:3121 ] for i in tsne_weekday]
tsne_train_flat_output = [i[:3121 ] for i in output]
tsne_train_flat_exp_avg = [i[:3121 ] for i in predict_list]
```

In [60]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) for o
ur test data
tsne_test_flat_lat = [i[3121:] for i in tsne_lat]
tsne_test_flat_lon = [i[3121:] for i in tsne_lon]
tsne_test_flat_weekday = [i[3121:] for i in tsne_weekday]
tsne_test_flat_output = [i[3121:] for i in output]
tsne_test_flat_exp_avg = [i[3121:] for i in predict_list]
```

In [61]:

```
# the above contains values in the form of list of lists (i.e. list of values of each r
egion), here we make all of them in one list
train_new_features = []
for i in range(0,20):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,20):
    test_new_features.extend(test_features[i])
```

In [62]:

```
train_new_features[0]
```

Out[62]:

```
array([ 29,  29, 120, 158, 148], dtype=int64)
```

In [295]:

```
train_freq[33][2][2]
```

Out[295]:

```
array([0.00492832, 0.00448029, 0.00403226, 0.00582437, 0.00067204,
       0.00649642, 0.00112007, 0.00806452, 0.0015681 , 0.00716846,
       0.0047043 , 0.0078405 , 0.00425627, 0.00201613, 0.00358423,
       0.00224014, 0.00560036, 0.00044803, 0.00246416, 0.0062724 ,
       0.        , 0.00380824, 0.00022401, 0.00672043, 0.00828853,
       0.0031362 , 0.00851254, 0.00604839, 0.00089606, 0.00694444,
       0.00268817, 0.00134409, 0.00739247, 0.00761649, 0.00291219,
       0.00515233, 0.00537634, 0.00336022, 0.00179211, 0.00873656])
```

In [290]:

```
train_new_freq = []
for i in range(0,40):
    l = []
    for j in range(3121):

        for k in range(5):
            l.append((train_freq[i][j][k]))
            train_new_freq.append(l)
        l=[]
```

In [303]:

```
print(train_new_freq[7][2])
```

```
[0.00492832 0.00448029 0.00403226 0.00582437 0.00067204 0.00649642
 0.00112007 0.00806452 0.0015681  0.00716846 0.0047043  0.0078405
 0.00425627 0.00201613 0.00358423 0.00224014 0.00560036 0.00044803
 0.00246416 0.0062724  0.         0.00380824 0.00022401 0.00672043
 0.00828853 0.0031362  0.00851254 0.00604839 0.00089606 0.00694444
 0.00268817 0.00134409 0.00739247 0.00761649 0.00291219 0.00515233
 0.00537634 0.00336022 0.00179211 0.00873656]
```

In [312]:

```
test_new_freq = []
for i in range(0,40):
    l = []
    for j in range(1338):
        for k in range(5):
            l.append((test_freq[i][j][k]))
            test_new_freq.append(lis)
        l = []
```

In [ ]:

```
import math
train_new_sp = []
for i in range(0,40):
    l = []
    for j in range(3121):
        for k in range(5):
            l.append(math.log(train_sp[i][j][k]))
            train_new_sp.append(l)
        l = []
```

In [ ]:

```
test_new_sp = []
for i in range(0,40):
    l = []
    for j in range(3121):
        for k in range(5):
            l.append(math.log(test_sp[i][j][k]))
            test_new_sp.append(l)
        l = []
```

In [64]:

```
columns_freq = ['freq_1','freq_2','freq_3','freq_4','freq_5']
columns_sp = ['sp_1','sp_2','sp_3','sp_4','sp_5']
df_freq_train = pd.DataFrame(train_new_freq,columns = columns_freq)
df_sp_train = pd.DataFrame(train_new_sp,columns = columns_amp)
df_freq_test = pd.DataFrame(test_new_freq,columns = columns_freq)
df_sp_test = pd.DataFrame(test_new_sp,columns = columns_amp)
```

In [65]:

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

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

In [66]:

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

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

In [67]:

```
# 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 = pd.merge(df_train,df_freq_train,left_index = True,right_index = True)
df_train = pd.merge(df_train,df_sp_train,left_index = True,right_index = True)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon

df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg

print(df_train.shape)
```

```
(62420, 19)
```

In [68]:

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

```
(26760, 19)
```

In [69]:

```
df_train.head()
```

Out[69]:

	ft_5	ft_4	ft_3	ft_2	ft_1	freq_1	freq_2	freq_3	freq_4	freq_5	s
0	29	29	120	158	148	0.00689	0.007136	0.014026	0.000246	0.007874	11.390
1	29	120	158	148	143	0.00689	0.007136	0.014026	0.000246	0.007874	11.390
2	120	158	148	143	171	0.00689	0.007136	0.014026	0.000246	0.007874	11.390
3	158	148	143	171	171	0.00689	0.007136	0.014026	0.000246	0.007874	11.390
4	148	143	171	171	162	0.00689	0.007136	0.014026	0.000246	0.007874	11.390

In [70]:

```
df_train['lat'].to_pickle('pickles/df_train[lat]')
df_train['lon'].to_pickle('pickles/df_train[lon]')
df_train['weekday'].to_pickle('pickles/df_train[weekday]')
df_train['exp_avg'].to_pickle('pickles/df_train[exp_avg]')

df_test['lat'].to_pickle('pickles/df_test[lat]')
df_test['lon'].to_pickle('pickles/df_test[lon]')
df_test['weekday'].to_pickle('pickles/df_test[weekday]')
df_test['exp_avg'].to_pickle('pickles/df_test[exp_avg]')
```

In [71]:

```
df_train['lat'] = pd.read_pickle('pickles/df_train[lat]')
df_train['lon'] = pd.read_pickle('pickles/df_train[lon]')
df_train['weekday'] = pd.read_pickle('pickles/df_train[weekday]')
df_train['exp_avg'] = pd.read_pickle('pickles/df_train[exp_avg]')

df_test['lat'] = pd.read_pickle('pickles/df_test[lat]')
df_test['lon'] = pd.read_pickle('pickles/df_test[lon]')
df_test['weekday'] = pd.read_pickle('pickles/df_test[weekday]')
df_test['exp_avg'] = pd.read_pickle('pickles/df_test[exp_avg]')
```

MODEL:

Linear Regression:

In [72]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
train_sc = sc.fit_transform(df_train)
test_sc = sc.transform(df_test)

from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV , RandomizedSearchCV

alpha  = [10**i for i in range(-8,9)]
param_grid = {'alpha':alpha}

sgd = SGDRegressor(loss = 'squared_loss')
grid = GridSearchCV(sgd , param_grid , cv = 10,scoring = 'neg_mean_absolute_error')
grid.fit(df_train,tsne_train_output)
alp  = grid.best_params_['alpha']

clf = SGDRegressor(alpha = alp,loss = 'squared_loss')
clf.fit(train_sc,tsne_train_output)

y_pred_test = clf.predict(test_sc)
y_pred_test = [round(value) for value in y_pred_test]

y_pred_train = clf.predict(train_sc)
y_pred_train = [round(value) for value in y_pred_train]


from sklearn.metrics import mean_squared_error , mean_absolute_error
train_mse_sgd = mean_squared_error(tsne_train_output,y_pred_train )
train_mpe_sgd = mean_absolute_error(tsne_train_output,y_pred_train )/(sum(tsne_train_ou
tput)/len(tsne_train_output))

test_mse_sgd = mean_squared_error(tsne_test_output,y_pred_test )
test_mpe_sgd = mean_absolute_error(tsne_test_output,y_pred_test )/(sum(tsne_test_output
)/len(tsne_test_output))
```

Random Forest:

In [ ]:

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

'''
param_rf = {'n_estimators':[40,150,200,600,800]}
clf_rf = RandomForestRegressor()
grid_rf = GridSearchCV(clf_rf,param_rf,scoring = 'neg_mean_absolute_error',cv = 3)
grid_rf.fit(df_train,tsne_train_output)
n_est_rf = grid_rf.best_params_['n_estimators']

clf_opt_rf = RandomForestRegressor(n_estimators = n_est_rf)
clf_opt_rf.fit(df_train,tsne_train_output)

'''
########################

n_estimators = [100, 200]
max_depth = [10, 50, 100]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

clf = RandomForestRegressor()

rf_random = RandomizedSearchCV(clf, param_distributions=random_grid, n_iter = 100, cv =
3, verbose=2,  n_jobs = -1)

rf_random.fit(df_train,tsne_train_output)
```

In [82]:

```
n_est_rf = rf_random.best_params_['n_estimators']

clf_opt_rf = RandomForestRegressor(n_estimators = n_est_rf)
clf_opt_rf.fit(df_train,tsne_train_output)
```

Out[82]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e)
```

In [83]:

```
y_pred_test = clf_opt_rf.predict(df_test)
y_pred_train = clf_opt_rf.predict(df_train)

train_mse_rf = mean_squared_error(tsne_train_output,y_pred_train )
train_mpe_rf = mean_absolute_error(tsne_train_output,y_pred_train )/(sum(tsne_train_out
put)/len(tsne_train_output))

test_mse_rf = mean_squared_error(tsne_test_output,y_pred_test )
test_mpe_rf = mean_absolute_error(tsne_test_output,y_pred_test )/(sum(tsne_test_output)
/len(tsne_test_output))
```

XGBOOST:

In [ ]:

```
clf = xgb.XGBRegressor(
        eval_metric = 'rmse',
        nthread = 4,
        eta = 0.1,
        num_boost_round = 80,
        max_depth = 5,
        subsample = 0.5,
        colsample_bytree = 1.0,
        silent = 1,
        )
parameters = {
        'num_boost_round': [10, 25, 50],
        'eta': [0.05, 0.1, 0.3],
        'max_depth': [3, 4, 5],
        'subsample': [0.9, 1.0],
        'colsample_bytree': [0.9, 1.0],
    }

clf_opt_xgb = RandomizedSearchCV(clf, param_distributions=parameters, n_iter = 100, cv
= 2, verbose=2,  n_jobs = -1)


clf_opt_xgb.fit(df_train,tsne_train_output)
```

In [89]:

```
best_parameters, score, _ = max(clf_opt_xgb.grid_scores_, key=lambda x: x[1])
print('RandomizedSearchCV Results: ')
print(score)

for param_name in sorted(best_parameters.keys()):
    print("%s: %r" % (param_name, best_parameters[param_name]))


y_pred_test = clf_opt_xgb.predict(df_test)
y_pred_train = clf_opt_xgb.predict(df_train)

train_mse_xgb = mean_squared_error(tsne_train_output,y_pred_train )
train_mpe_xgb = mean_absolute_error(tsne_train_output,y_pred_train)/(sum(tsne_train_out
put)/len(tsne_train_output))

test_mse_xgb = mean_squared_error(tsne_test_output,y_pred_test)
test_mpe_xgb = mean_absolute_error(tsne_test_output,y_pred_test)/(sum(tsne_test_output)
/len(tsne_test_output))
```

```
RandomizedSearchCV Results:
0.9466201632390374
colsample_bytree: 0.9
eta: 0.1
max_depth: 3
num_boost_round: 10
subsample: 0.9
```

Calculating the error metric values for various models:

In [90]:

```
train_mape=[]
test_mape=[]

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

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/(sum(t
sne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(su
m(tsne_test_output)/len(tsne_test_output)))
```

Error Metric Matrix:

In [99]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)")
print ("--------------------------------------------------------------------------------
------------------------")
print ("Baseline Model -                              Train(%): ",train_mape[0]*100,"
Test(%): ",test_mape[0]*100)
print ("Exponential Averages Forecasting -            Train(%): ",train_mape[1]*100,"
Test(%): ",test_mape[1]*100)
print ("Linear Regression -                          Train(%): ",train_mpe_sgd*100 ,"
Test(%): ",(test_mpe_sgd)*100)
print ("Random Forest Regression -                    Train(%): ",train_mpe_rf*100,"
Test(%): ",test_mpe_rf*100)
print ("XgBoost Regression -                          Train(%): ",train_mpe_xgb*100,"
Test(%): ",test_mpe_xgb*100)
print ("--------------------------------------------------------------------------------
------------------------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)
--------------------------------------------------------------------------
-----------------------------
Baseline Model -                              Train(%):  13.85566139386993
Test(%):  13.842551135646787
Exponential Averages Forecasting -            Train(%):  13.096604735655026
Test(%):  13.035643399831986
Linear Regression -                          Train(%):  13.013864287817775
Test(%):  13.005240499863064
Random Forest Regression -                    Train(%):  4.821183540122485
Test(%):  11.423901436673699
XgBoost Regression -                          Train(%):  11.05647682349867
Test(%):  11.163352294464584
--------------------------------------------------------------------------
-----------------------------
```

Conclusion and Steps followed:

1. Initially we have done EDA and outlier removal on several features and reframed the data based on the desired boundaries. we considered Jan 2016 data for our operations.
2. We tried to get various clusters based on the data we have got. The clusters are considered for further opeartions.
3. Next we have built various baseline models like Exponential averages forcasting on jan 2016 data and tried to get the MAPE(mean absolute percentage error) and MSE(mean squared error) metric and observed which model performed the best.
4. Then we tried to add some fourier transform features (which we got from scipy library) and merged them with the existing five values to perform our model better. The existing features like exponential average forecasting feature.
5. After that we tried to tune the hyperparameters using gridsearch and randomsaerch on various models like linear regression(Gridsearch), random forest(random search) and xgboost (random search).
6. We observed that our Random Forest regression model was overfitting as the error difference between test and train was huge.
7. The XGboost model performed the best, as we could reduce the MAPE lesser than the baseline model by nearly 2%.