Taxi demand prediction in New York City

In [1]:

```
#Importing Libraries
# pip3 install graphviz
#pip3 install dask
#pip3 install toolz
#pip3 install cloudpickle
# https://www.youtube.com/watch?v=ieW3G7ZzRZ0
# https://github.com/dask/dask-tutorial
# please do go through this python notebook: https://github.com/dask/dask-tutorial/blob/mas
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 inline
# matplotlib: used to plot graphs
import matplotlib
# matplotlib.use('nbagg') : matplotlib uses this protocall which makes plots more user intr
matplotlib.use('nbagg')
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots

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

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

# download migwin: https://mingw-w64.org/doku.php/download/mingw-builds
# install it in your system and keep the path, migw_path ='installed path'
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mingw64\\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")
```

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

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

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

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

In [2]:

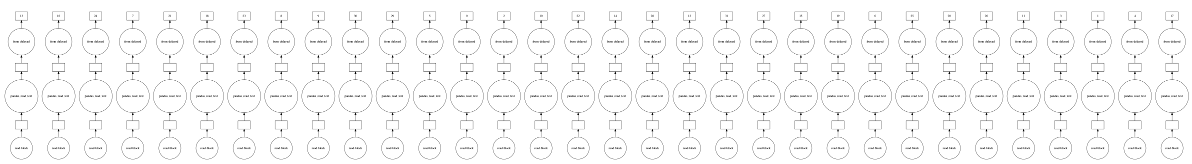
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('yellow_tripdata_2015-01.csv')
print(month.columns)
```

```
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_amoun
t',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [3]:

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

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

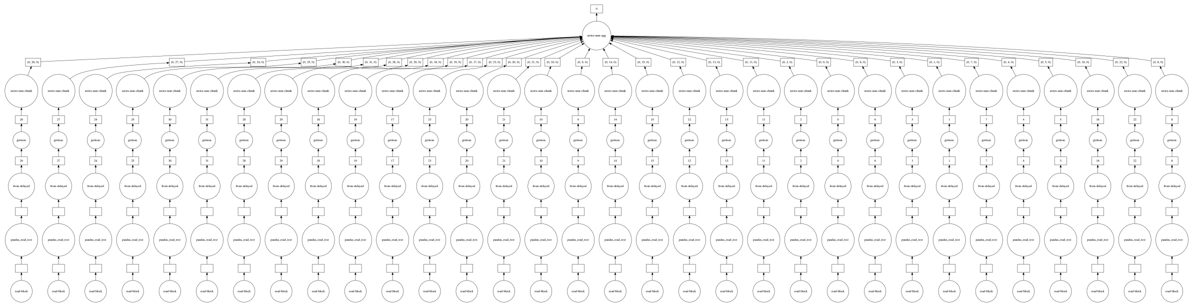
Out[3]:



In [4]:

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

Out[4]:



Features in the dataset:

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

```
    <td>The total amount charged to passengers. Does not include cash tips.</td>
</tr>
```

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

ML Problem Formulation

Time-series forecasting and Regression

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

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

Performance metrics

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

Data Cleaning

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

In [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_distance	picku
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30	
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80	
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50	
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00	

1. Pickup Latitude and Pickup Longitude

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

In [6]:

```
# 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_locati
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_latitude <= 4
                   (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 m

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
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[6]:

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

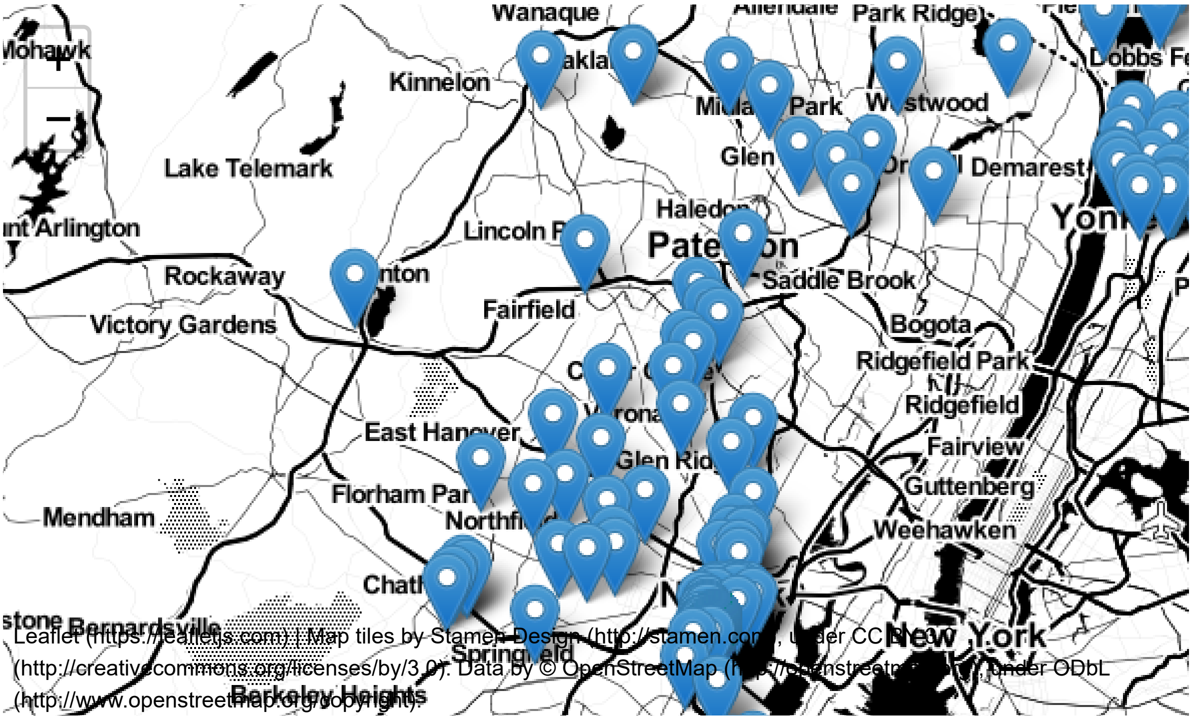
2. Dropoff Latitude & Dropoff Longitude

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 [7]:

```
# 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_locati
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitude <=
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 40.91

# 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 m

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
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[7]:



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

3. Trip Durations:

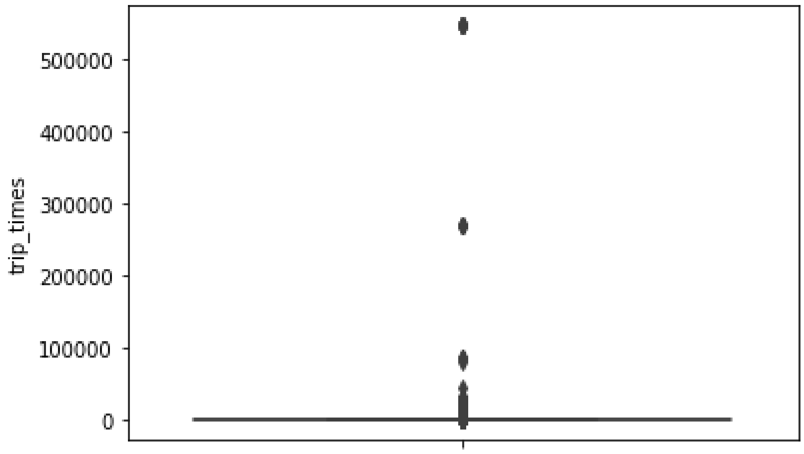
According to NYC Taxi & Limousine Commision Regulations **the maximum allowed trip duration in a 24 hour interval is 12 hours.**

In [8]:

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

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



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

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

    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
#   1                  1.59       -73.993896          40.750111       -73.974785
#   1                   3.30        -74.001648          40.724243       -73.994415
#   1                   1.80        -73.963341          40.802788       -73.951820
#   1                   0.50        -74.009087          40.713818       -74.004326
#   1                   3.00        -73.971176          40.762428       -74.004181
frame_with_durations = return_with_trip_times(month)
```

In [9]:

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



In [10]:

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

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

In [11]:

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

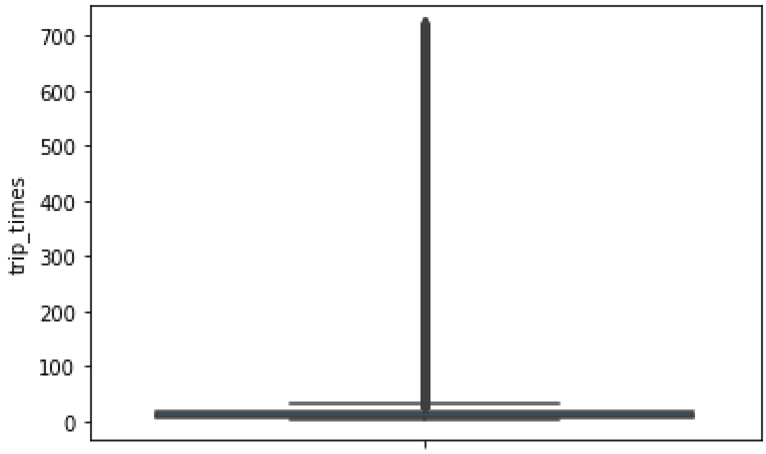
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98 percentile value is 38.71666666666667
99 percentile value is 46.75
100 percentile value is  548555.6333333333
```

In [12]:

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

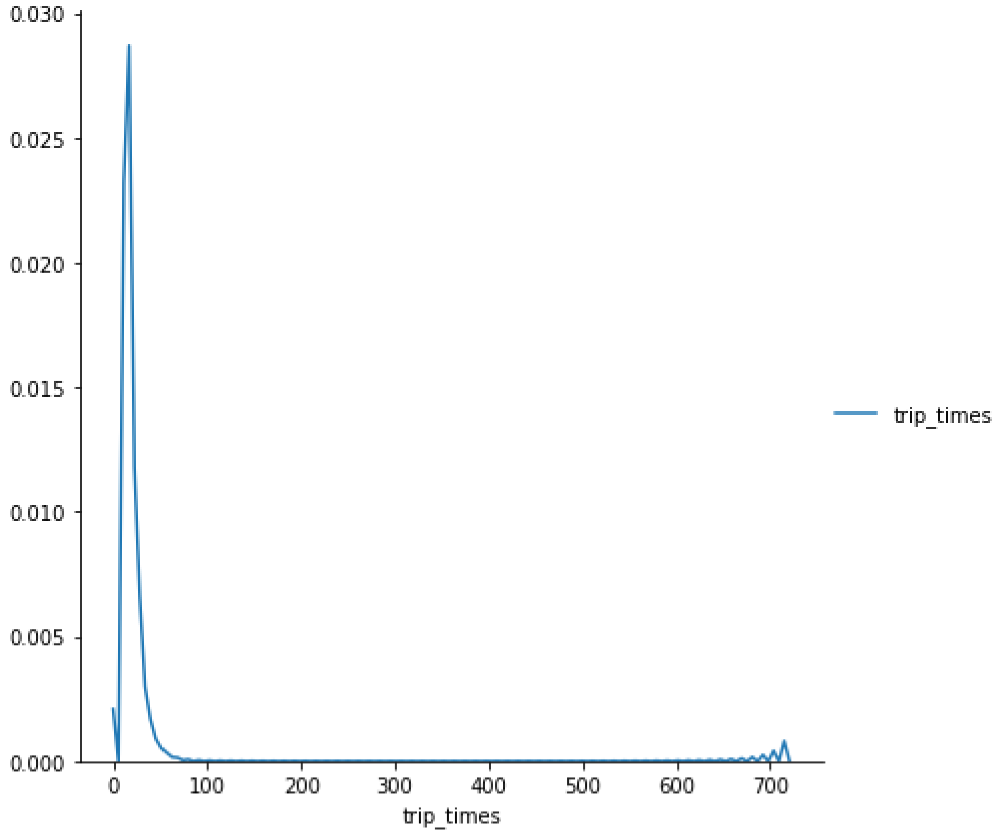
In [13]:

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



In [14]:

```
#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();
```

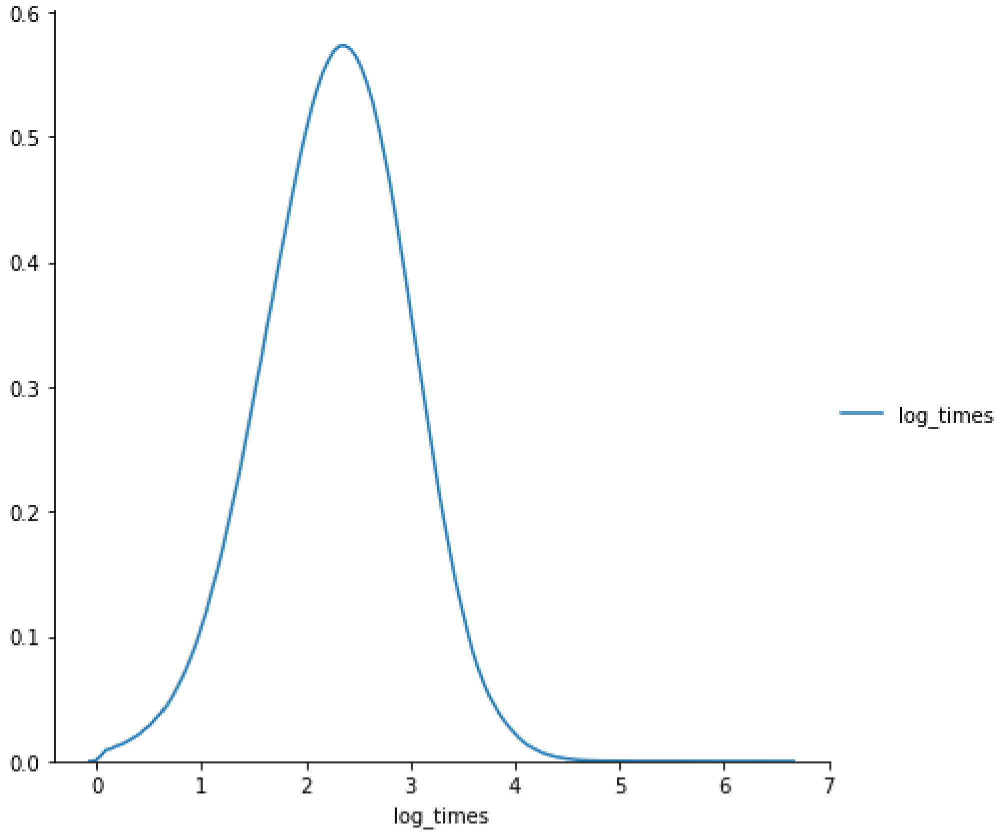


In [15]:

```
#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_modif
```

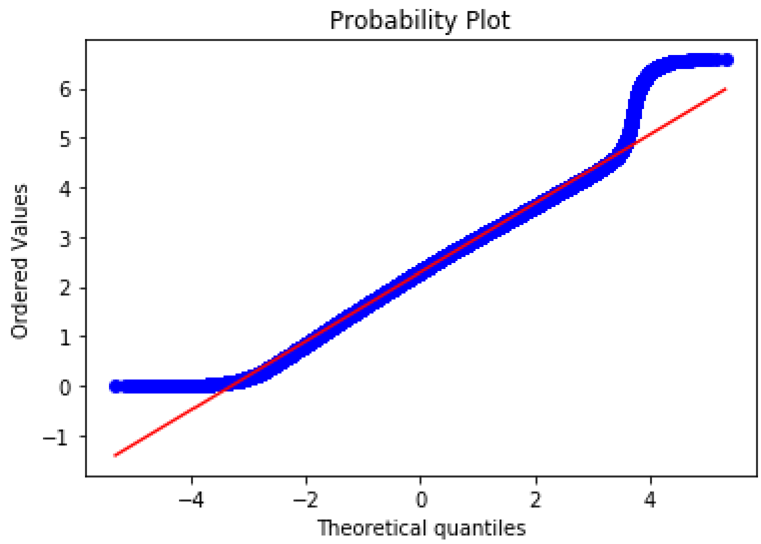
In [16]:

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



In [17]:

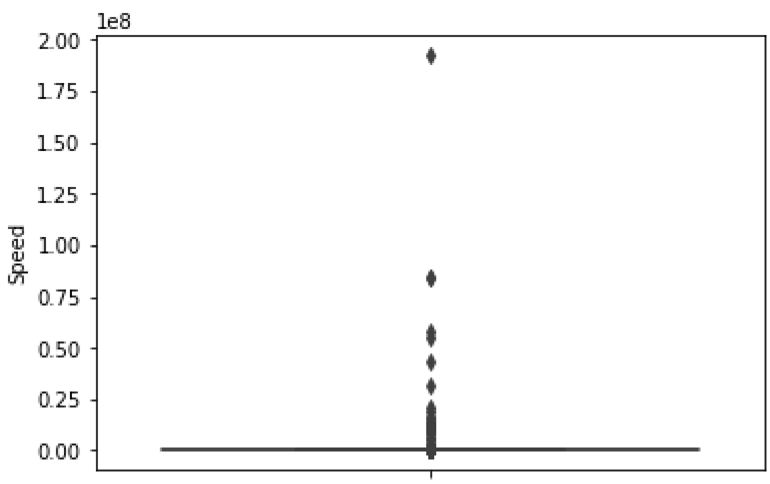
```
import scipy

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



4. Speed

In [18]:

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



In [19]:

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

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

In [20]:

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

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

In [21]:

```
#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
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

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

In [22]:

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

In [23]:

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

Out[23]:

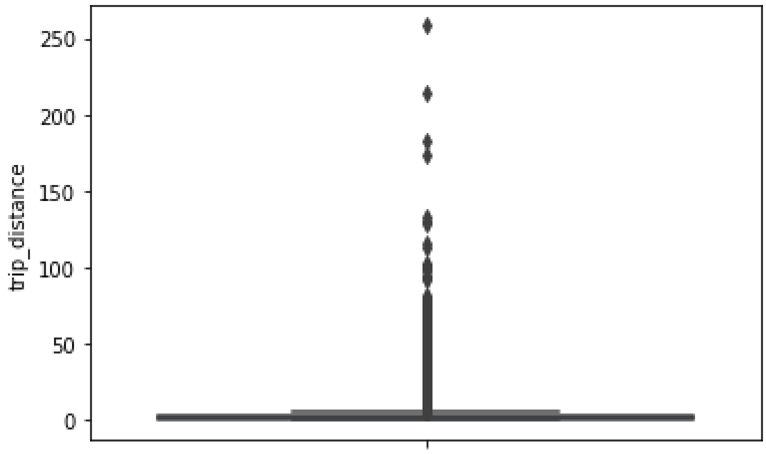
```
12.450173996027528
```

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 [24]:

```
# 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 [25]:

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

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

In [26]:

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

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

In [27]:

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

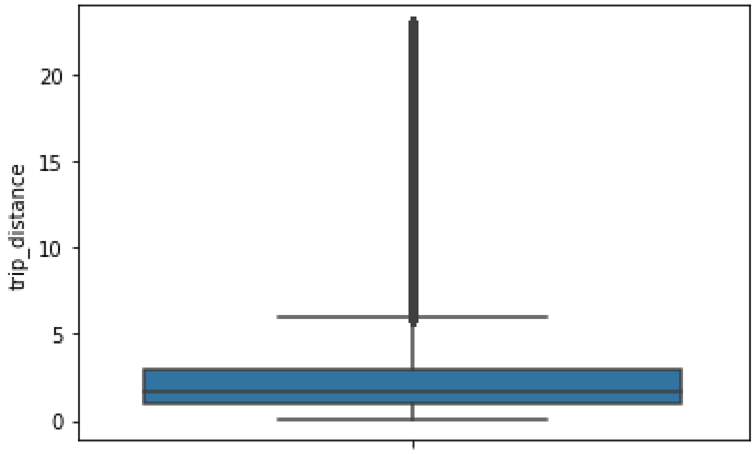
```
99.0 percentile value is 18.17
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.83
99.4 percentile value is 19.13
99.5 percentile value is 19.5
99.6 percentile value is 19.96
99.7 percentile value is 20.5
99.8 percentile value is 21.22
99.9 percentile value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

In [28]:

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

In [29]:

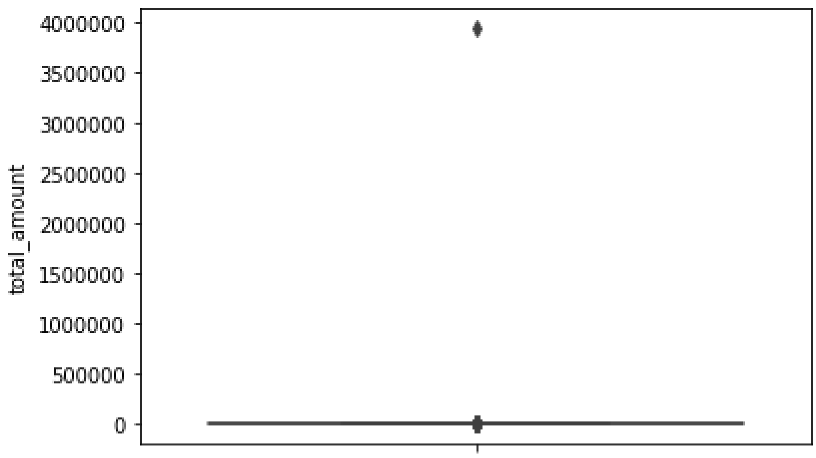
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [30]:

```
# up to now we have removed the outliers based on trip durations, cab speeds, and trip dist
# 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 [31]:

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

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

In [32]:

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

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

In [33]:

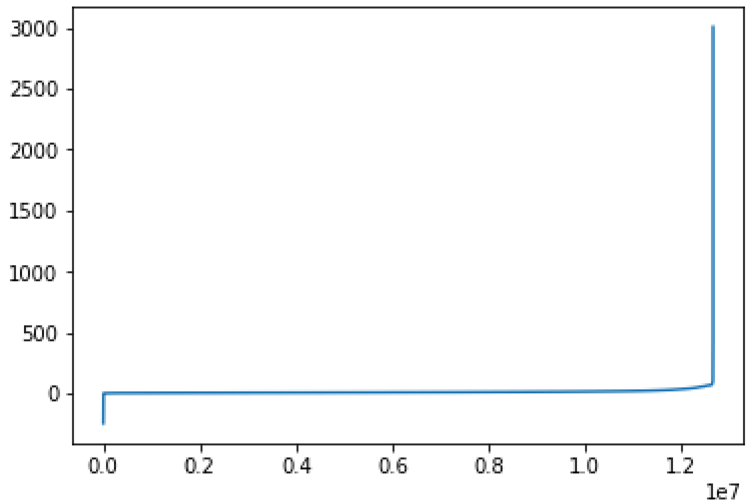
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

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

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

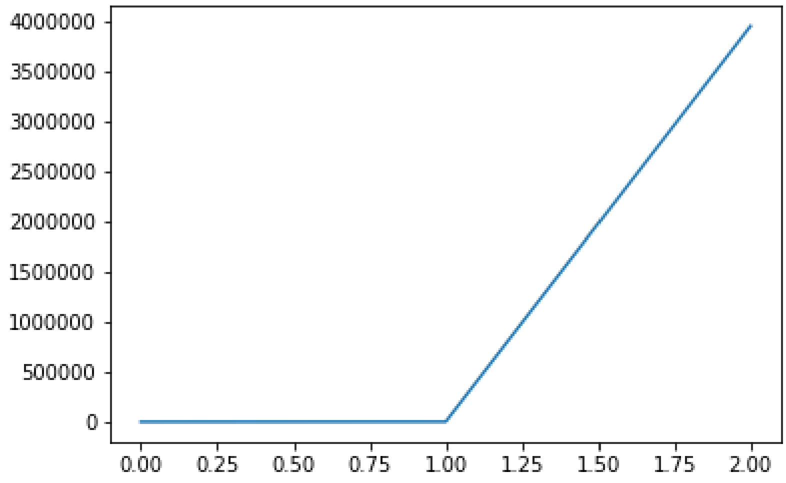
In [34]:

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



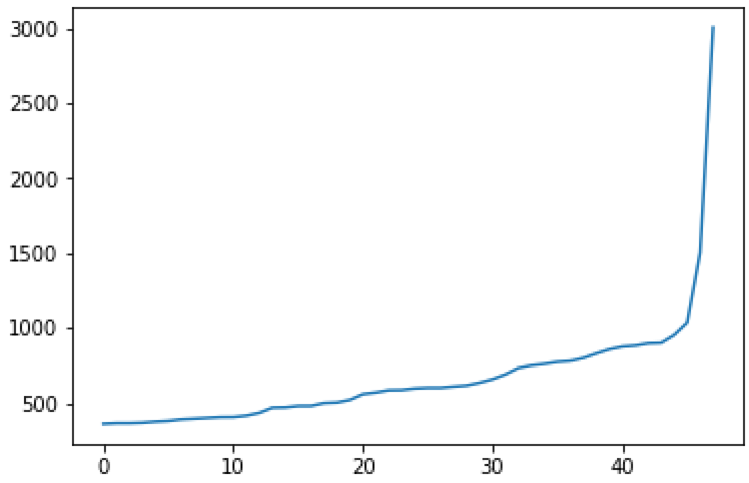
In [35]:

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



In [36]:

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



Remove all outliers/erronous points.

In [37]:

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


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


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


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

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

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


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_lon
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitud
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitude
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitud

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

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

In [38]:

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

```
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 0.97035764256074
95
```

Data-preperation

Clustering/Segmentation

In [39]:

```
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']].val
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_cent
                min_dist = min(min_dist,distance/(1.60934*1000))
                if (distance/(1.60934*1000)) <= 2:
                    nice_points +=1
                else:
                    wrong_points += 1
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters within t

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

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

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 8.0
Min inter-cluster distance =  1.0945442325142543
---
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 =  0.7131298007387813
---
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 =  0.5185088176172206
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 32.0
Min inter-cluster distance =  0.5069768450363973
---
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 =  0.365363025983595
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 46.0
Min inter-cluster distance =  0.34704283494187155
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 54.0
Min inter-cluster distance =  0.30502203163244707
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 62.0
Min inter-cluster distance =  0.29220324531738534
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 69.0
Min inter-cluster distance =  0.18257992857034985
---
```

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 [40]:

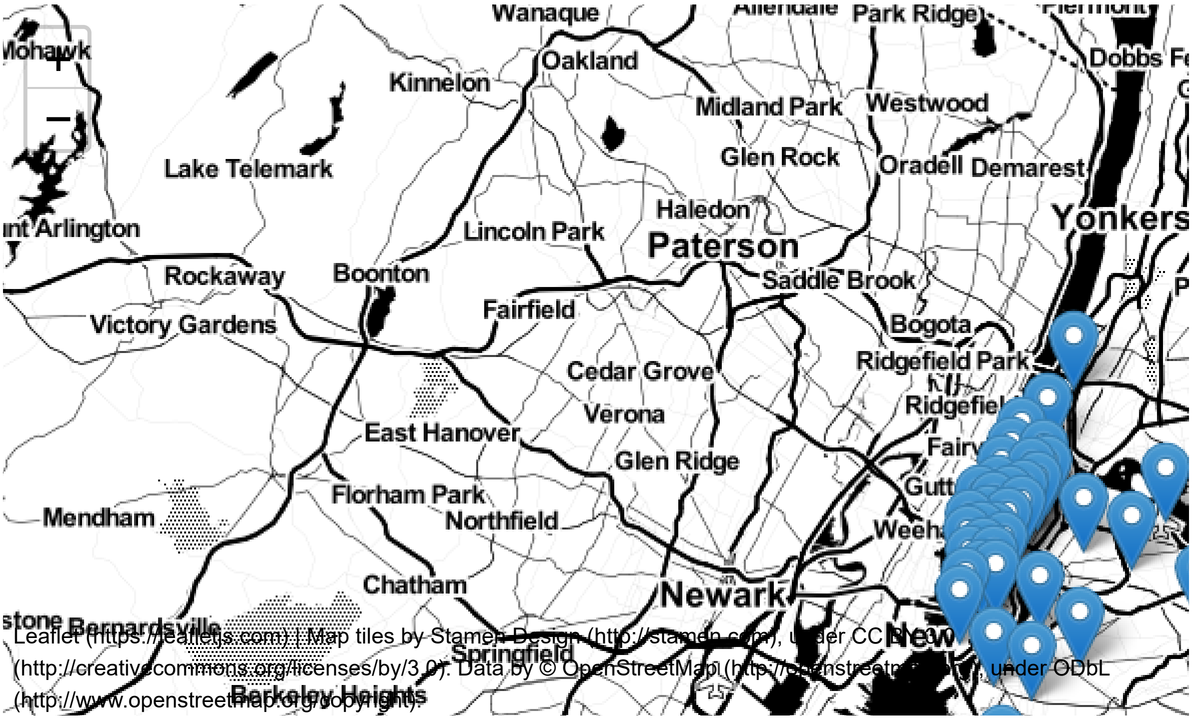
```
# if check for the 50 clusters you can observe that there are two clusters with only 0.3 mi
# so we choose 40 clusters for solve the further problem

# 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_duratio
```

Plotting the cluster centers:

In [41]:

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

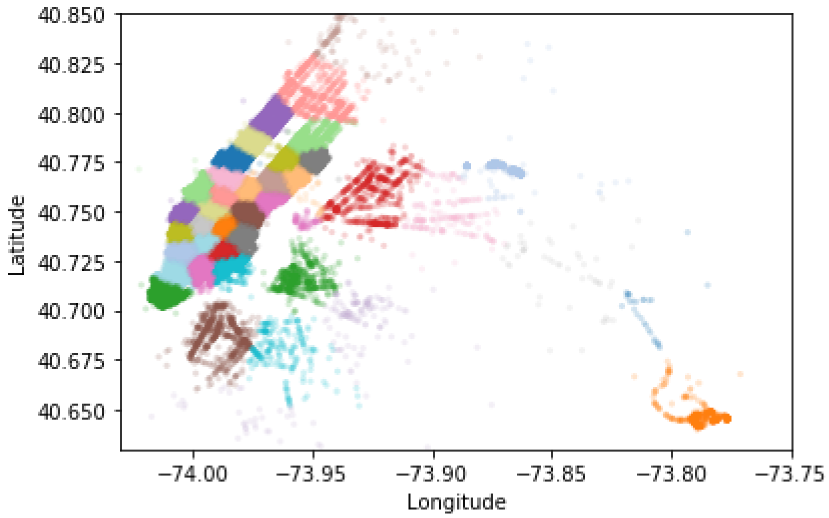
Out[41]:



Plotting the clusters:

In [42]:

```
#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[:100000
               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 [43]:

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

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

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

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

In [44]:

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

In [45]:

```
# 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[45]:

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

In [46]:

```
# hear the trip_distance represents the number of pickups that are happend in that particul
# 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*60/1
jan_2015_groupby.head()
```

Out[46]:

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

In [47]:

```
!wget --header="Host: s3.amazonaws.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
```

```
--2020-01-17 22:28:10--  https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_t
ripdata_2016-01.csv (https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripd
ata_2016-01.csv)
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.27.134
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.27.134|:443... conn
ected.
HTTP request sent, awaiting response... 416 Requested Range Not Satisfiable

    The file is already fully retrieved; nothing to do.
```

In [48]:

```
!wget --header="Host: s3.amazonaws.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
```

```
--2020-01-17 21:44:45--  https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_t
ripdata_2016-02.csv (https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripd
ata_2016-02.csv)
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.45.110
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.45.110|:443... conn
ected.
HTTP request sent, awaiting response... 416 Requested Range Not Satisfiable

    The file is already fully retrieved; nothing to do.
```

In [49]:

```
!wget --header="Host: s3.amazonaws.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
```

```
--2020-01-17 21:44:45--  https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_t
ripdata_2016-03.csv (https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripd
ata_2016-03.csv)
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.45.110
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.45.110|:443... conn
ected.
HTTP request sent, awaiting response... 416 Requested Range Not Satisfiable

    The file is already fully retrieved; nothing to do.
```

In [48]:

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

# now do the same operations for months Jan, Feb, March of 2016
# 1. get the dataframe which inlcudes only required colums
# 2. adding trip times, speed, unix time stamp of pickup_time
# 4. remove the outliers based on trip_times, speed, trip_duration, total_amount
# 5. add pickup_cluster to each data point
# 6. add pickup_bin (index of 10min intravel to which that trip belongs to)
# 7. group by data, based on 'pickup_cluster' and 'pickuo_bin'

# Data Preparation for the months of Jan,Feb and March 2016
def datapreparation(month,kmeans,month_no,year_no):

    print ("Return with trip times..")

    frame_with_durations = return_with_trip_times(month)

    print ("Remove outliers..")
    frame_with_durations_outliers_removed = remove_outliers(frame_with_durations)

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

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

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('yellow_tripdata_2016-01.csv')
month_feb_2016 = dd.read_csv('yellow_tripdata_2016-02.csv')
month_mar_2016 = dd.read_csv('yellow_tripdata_2016-03.csv')

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

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

Smoothing

In [49]:

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

# for each cluster region we will collect all the indices of 10min intravels in which the p
# 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 [50]:

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

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

#feb
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [51]:

```
# 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
    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 [52]:

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

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

In [53]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already visited/resolve
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup b
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing,
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.0/((right_
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missin
                    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 [54]:

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

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

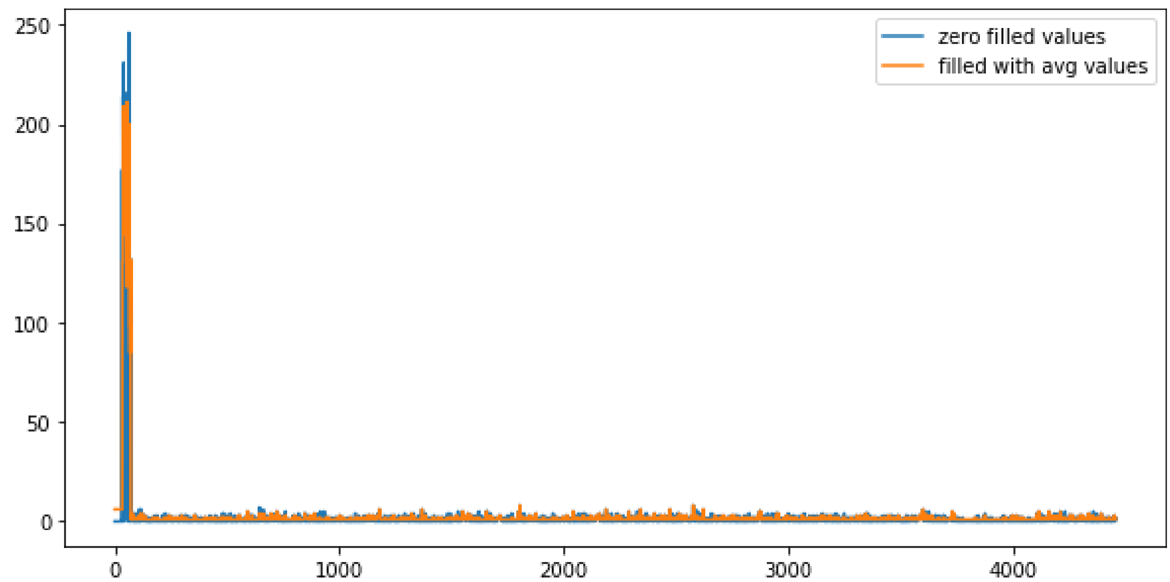
In [55]:

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

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

In [56]:

```
# 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 [57]:

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

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

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

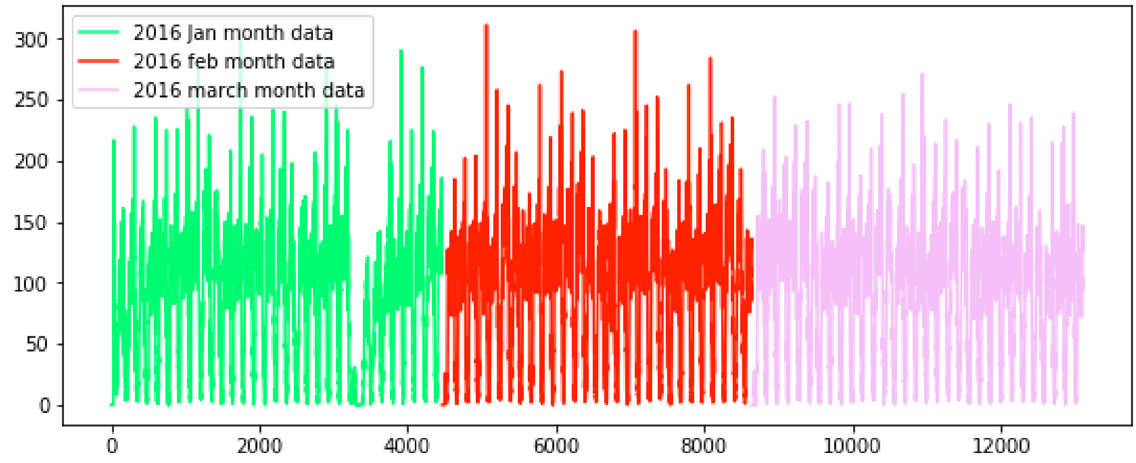
In [58]:

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

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

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

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

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

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

Time series and Fourier Transforms

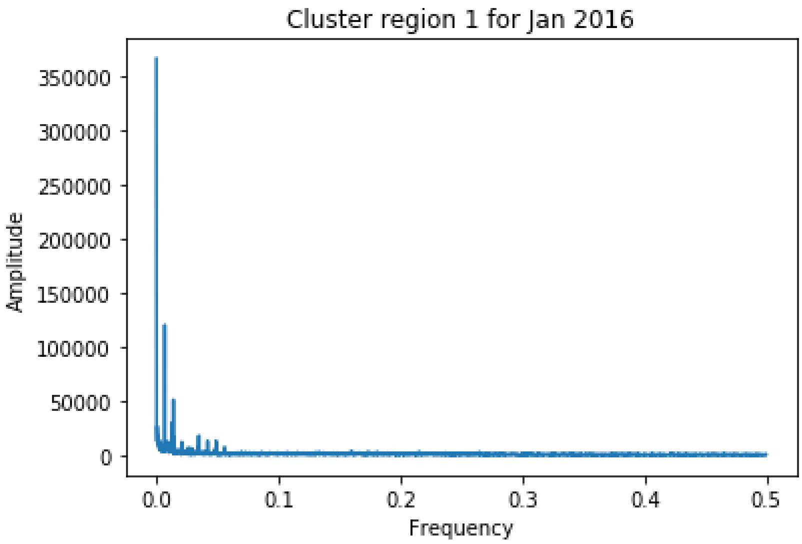
In [59]:

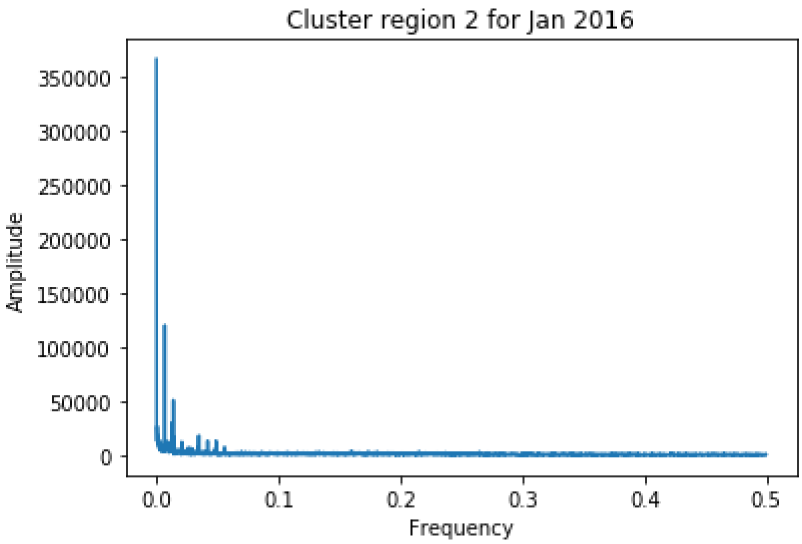
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40):
    plt.figure(figsize=(10,4))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

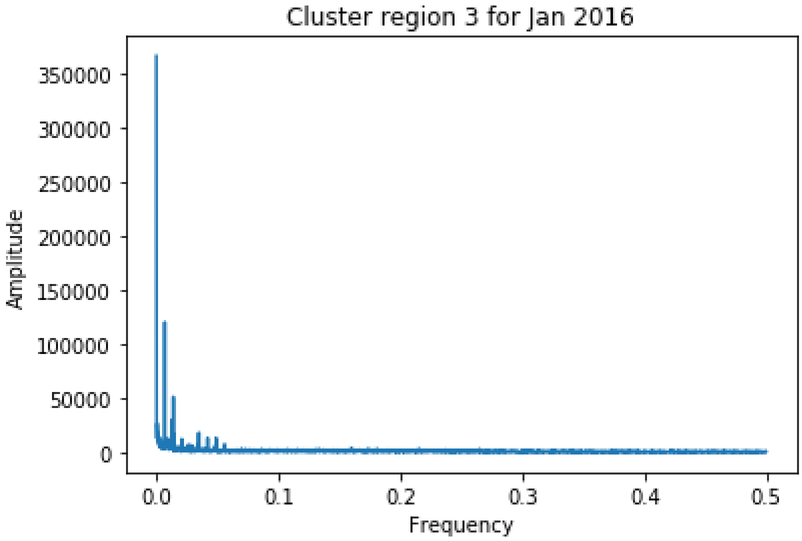


In [60]:

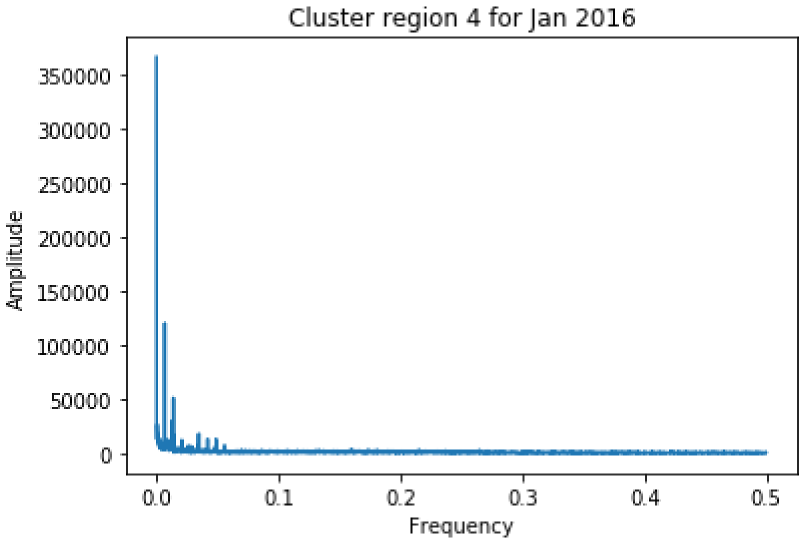
```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generated/numpy
for i in range(40):
    Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated/numpy.f
    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.title("Cluster region {} for Jan 2016".format(i+1))
    plt.show()
```



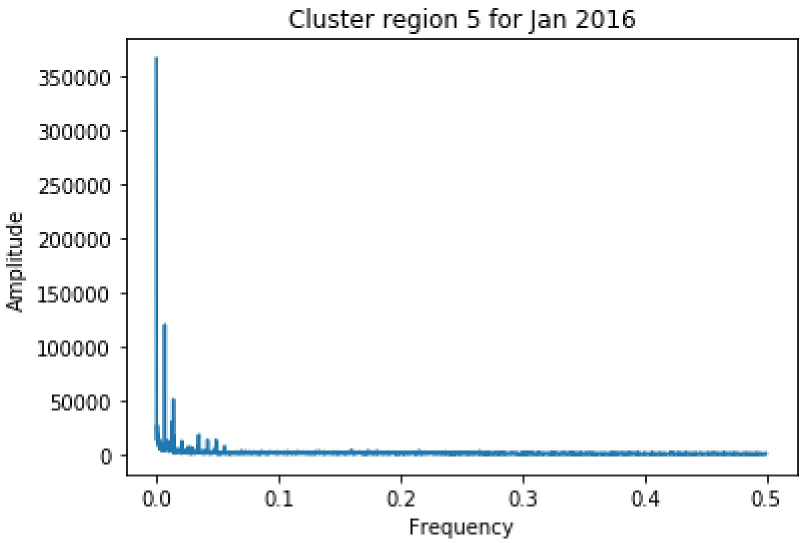




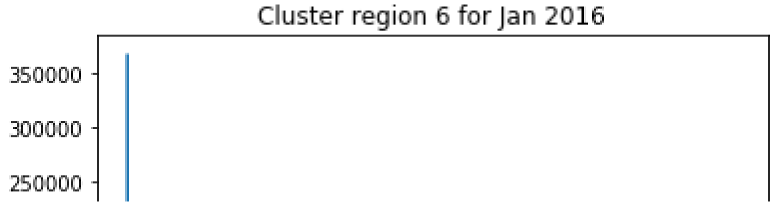
Cluster region 3 for Jan 2016
Amplitude
Frequency

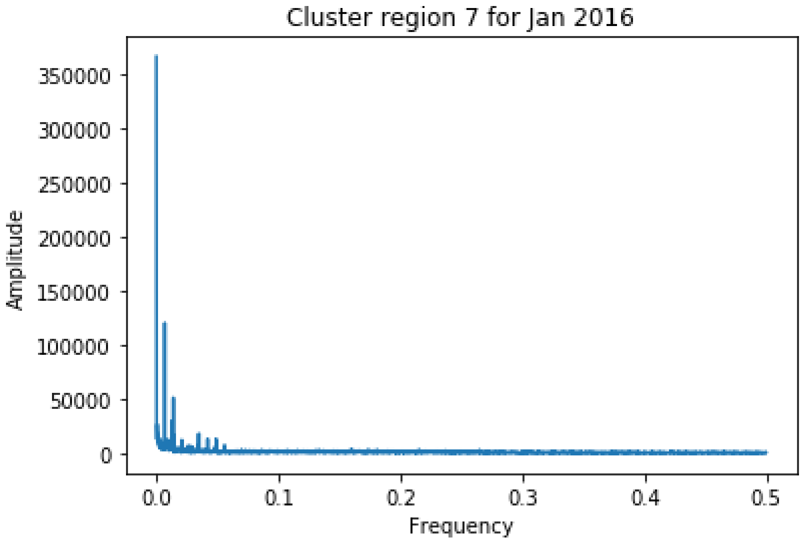


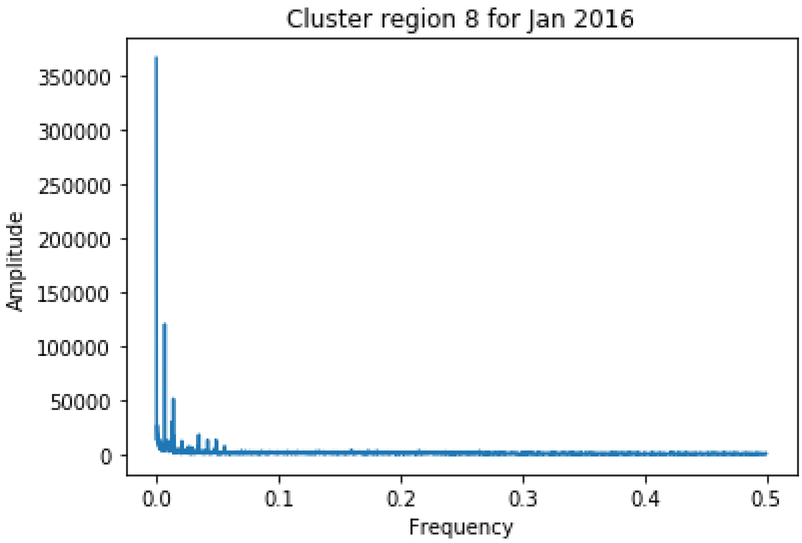
Cluster region 4 for Jan 2016
Amplitude
Frequency

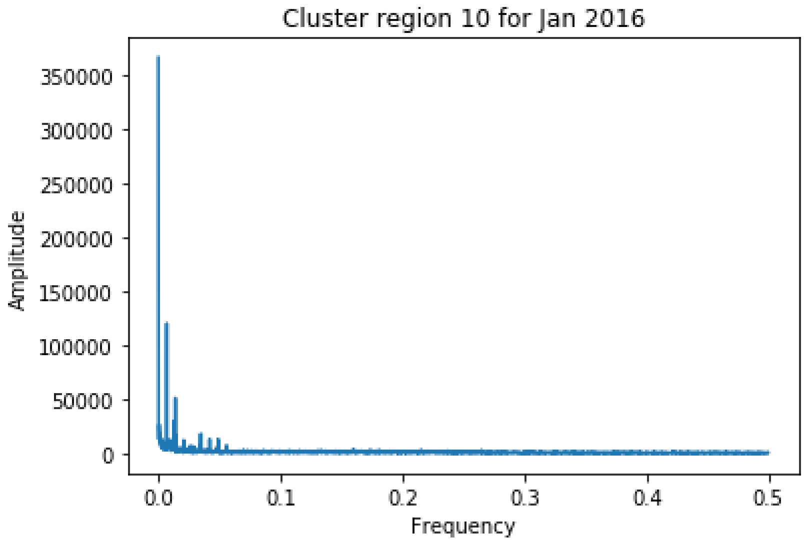


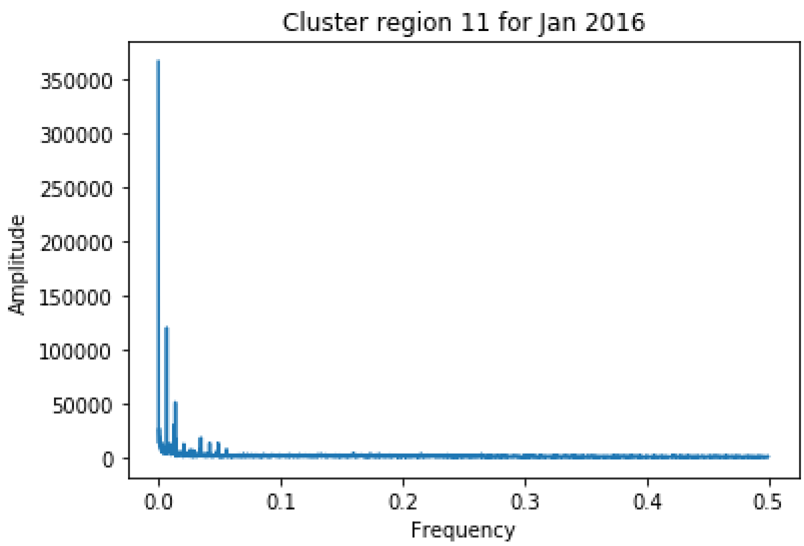
Cluster region 5 for Jan 2016
Amplitude
Frequency

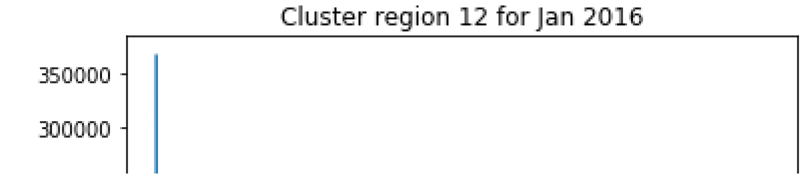

Cluster region 6 for Jan 2016
350000
300000
250000

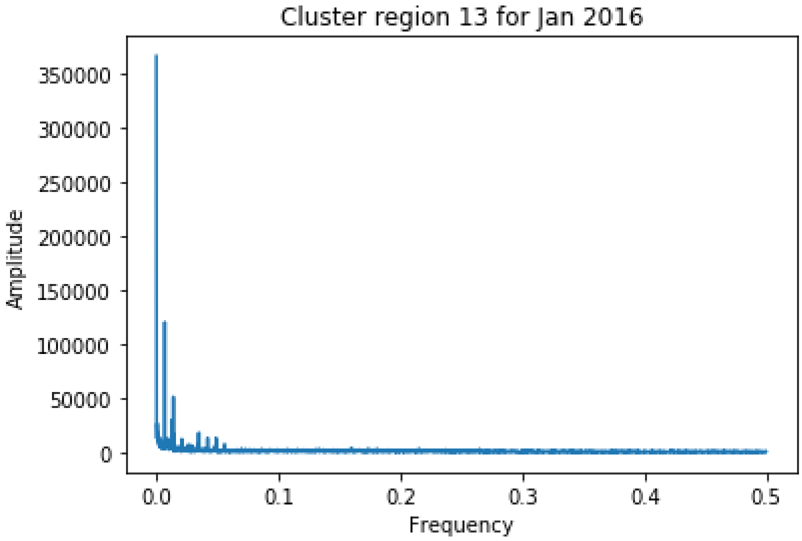

Cluster region 7 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

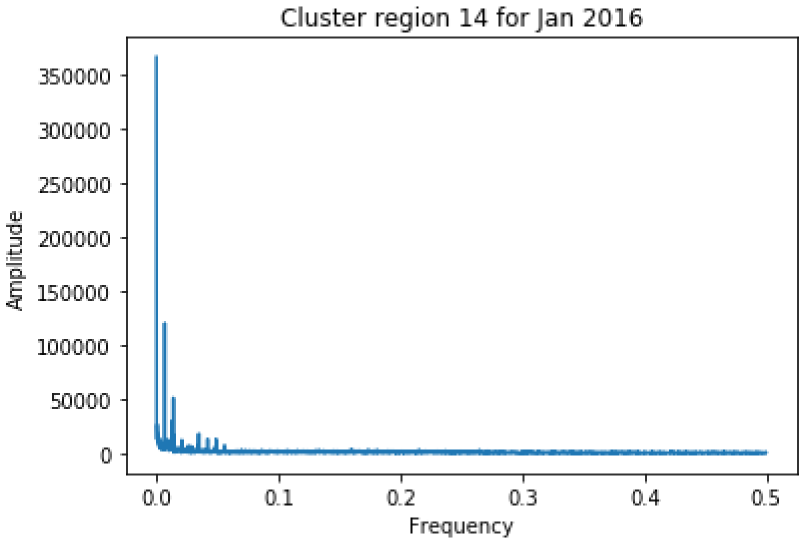

Cluster region 8 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

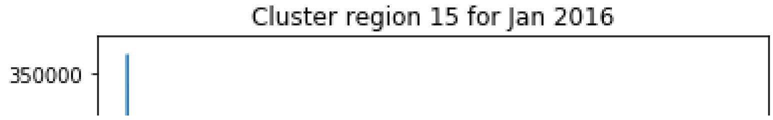

Cluster region 10 for Jan 2016
350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

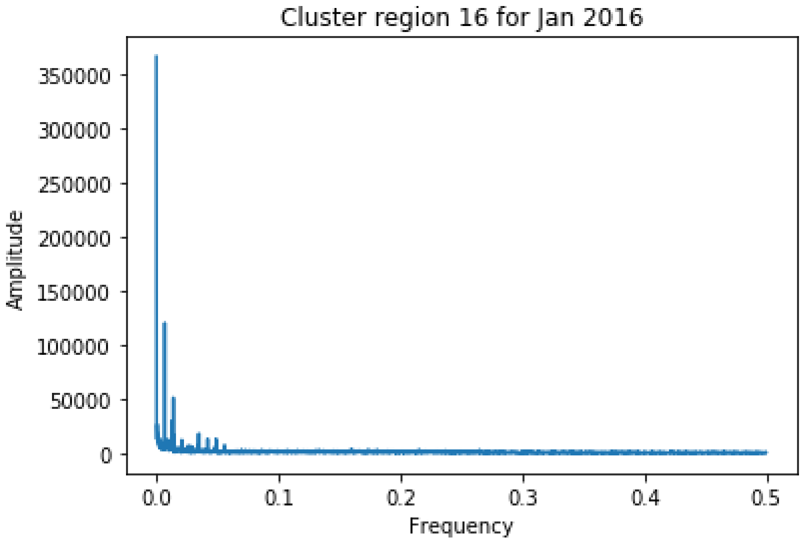

Cluster region 11 for Jan 2016
350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

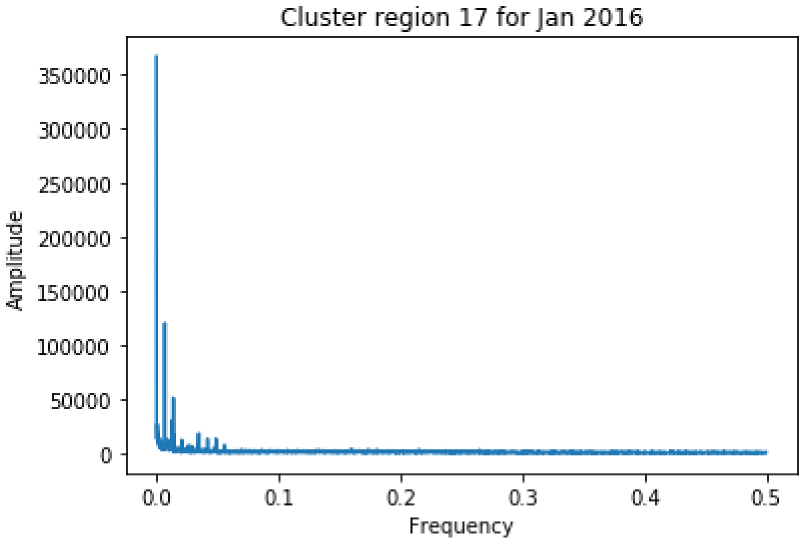

Cluster region 12 for Jan 2016
350000
300000

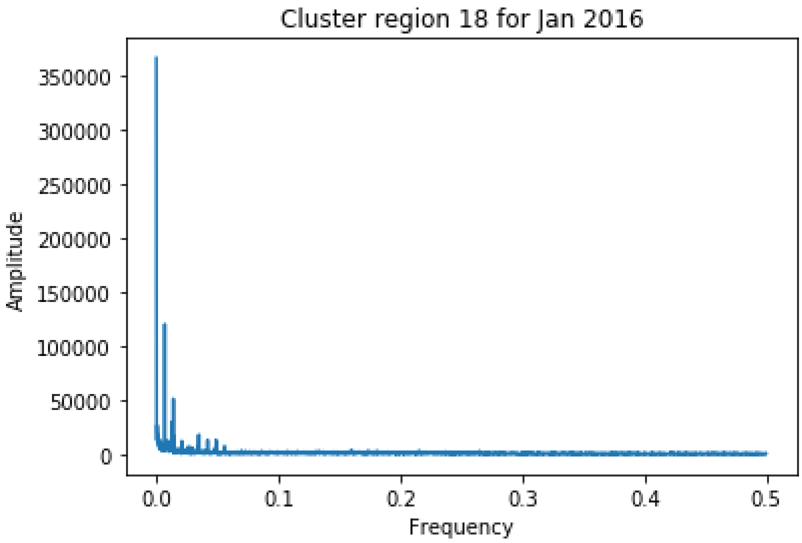

Cluster region 13 for Jan 2016
Amplitude
Frequency

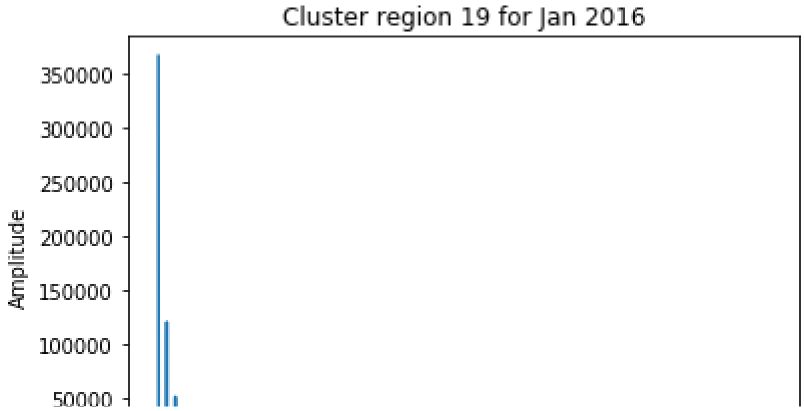

Cluster region 14 for Jan 2016
Amplitude
Frequency

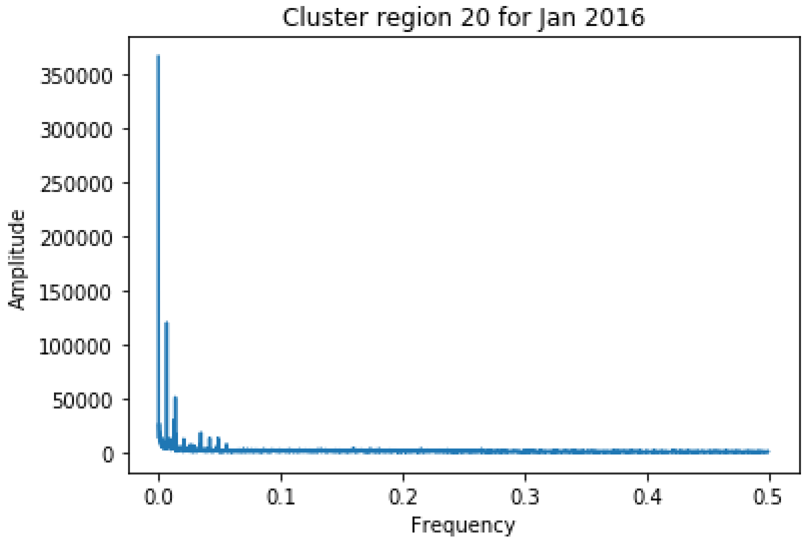

Cluster region 15 for Jan 2016
350000

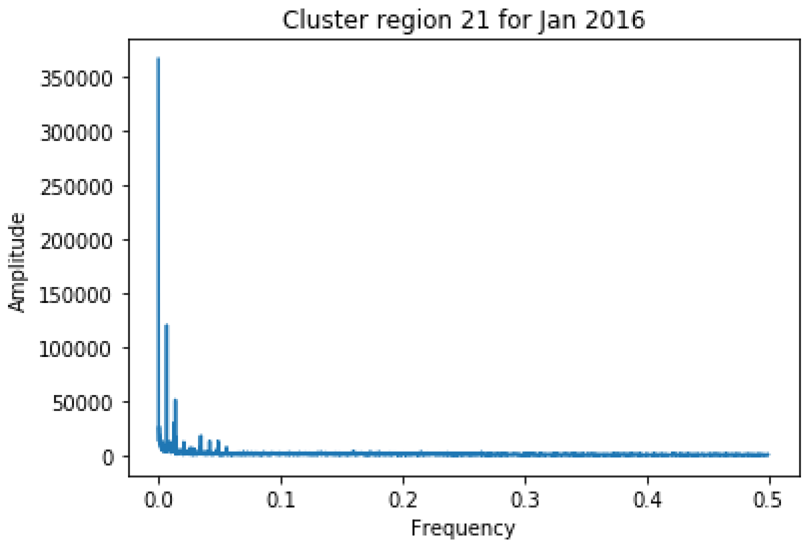

Cluster region 16 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

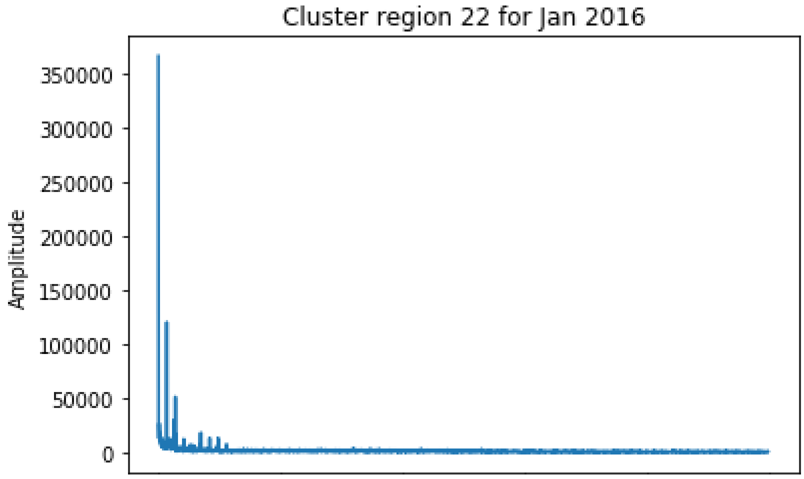

Cluster region 17 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

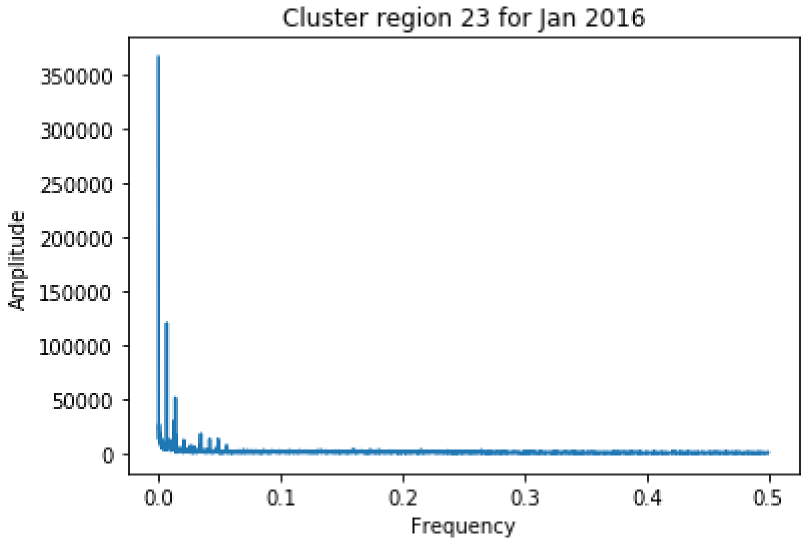

Cluster region 18 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

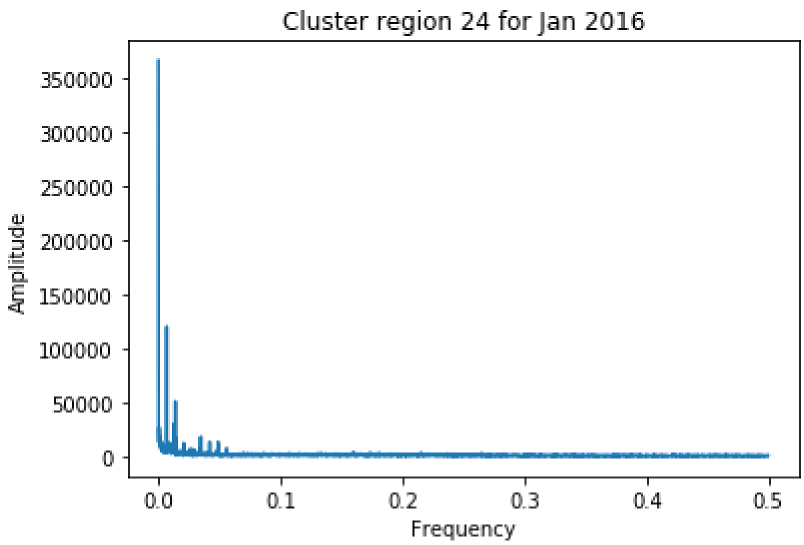

Cluster region 19 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000

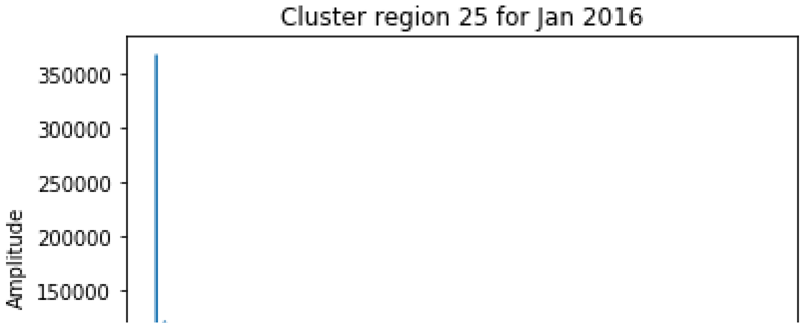

Cluster region 20 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

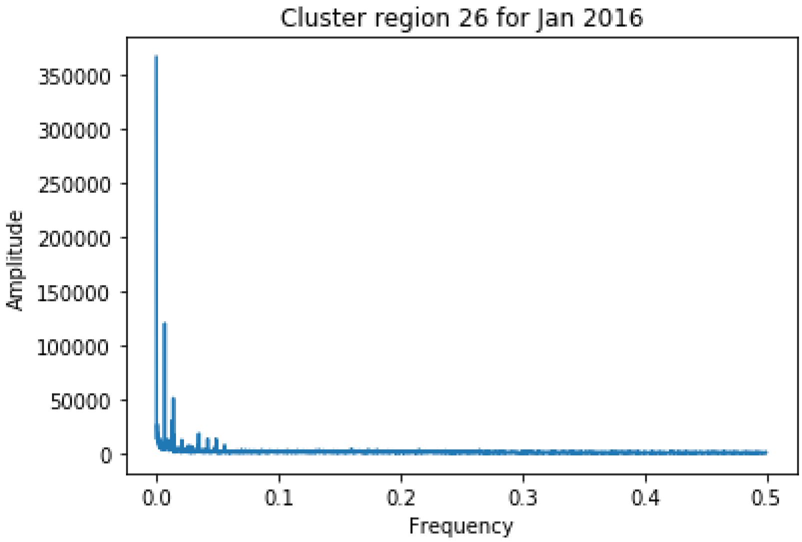

Cluster region 21 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

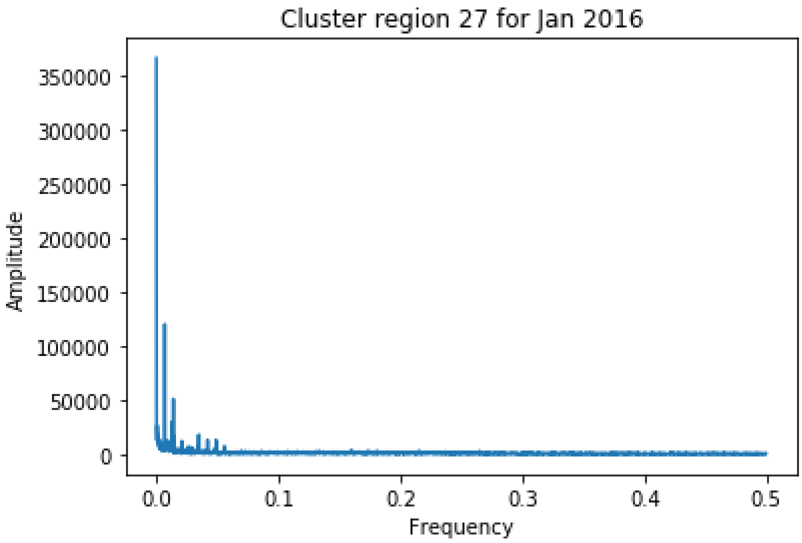

Cluster region 22 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0

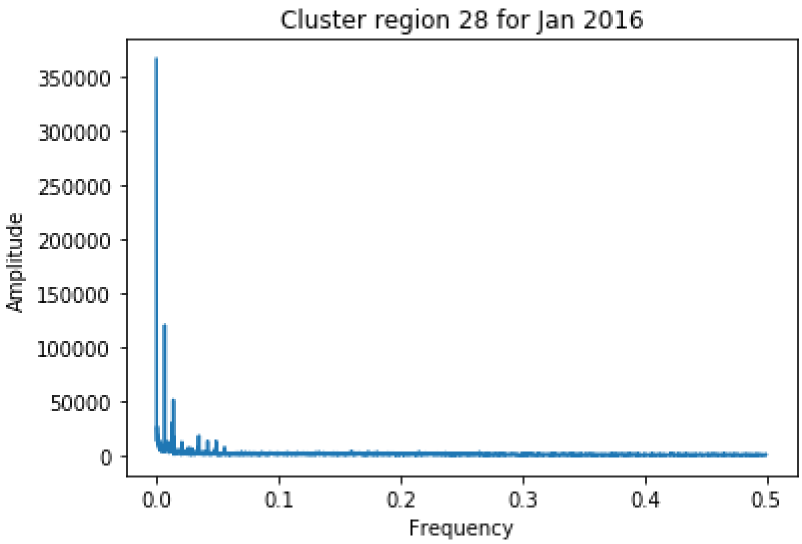

Cluster region 23 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Cluster region 24 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

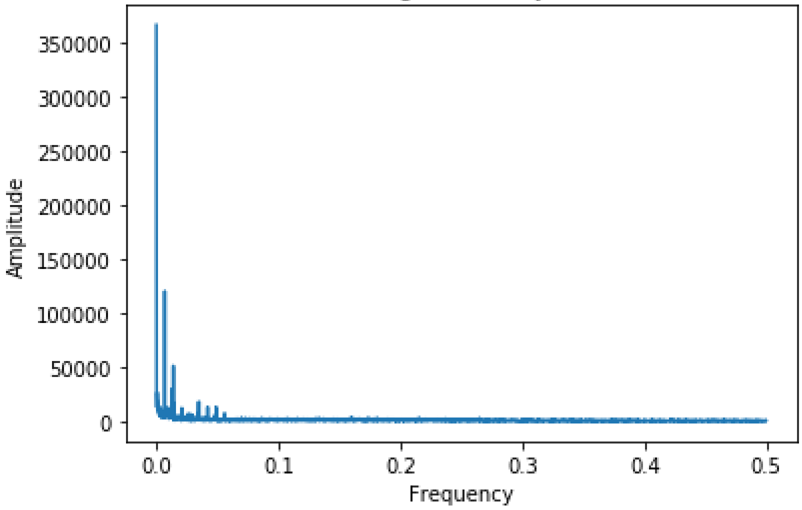

Cluster region 25 for Jan 2016
Amplitude
350000
300000
250000
200000
150000


Cluster region 26 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

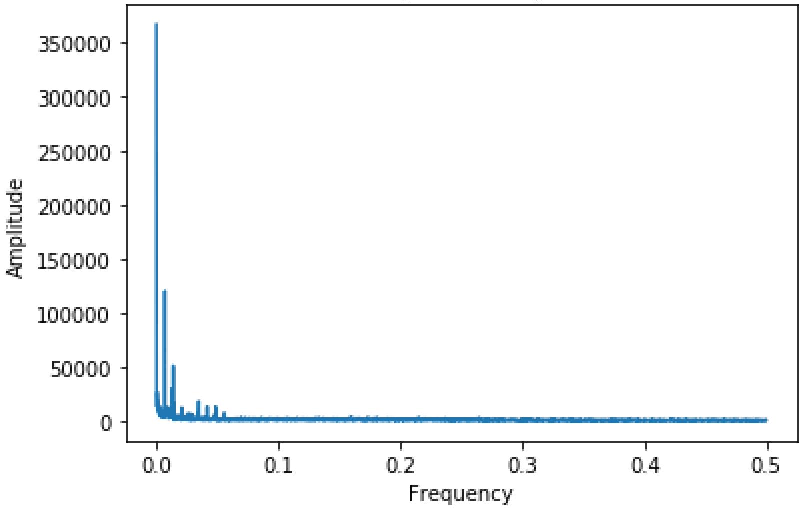

Cluster region 27 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5


Cluster region 28 for Jan 2016
Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

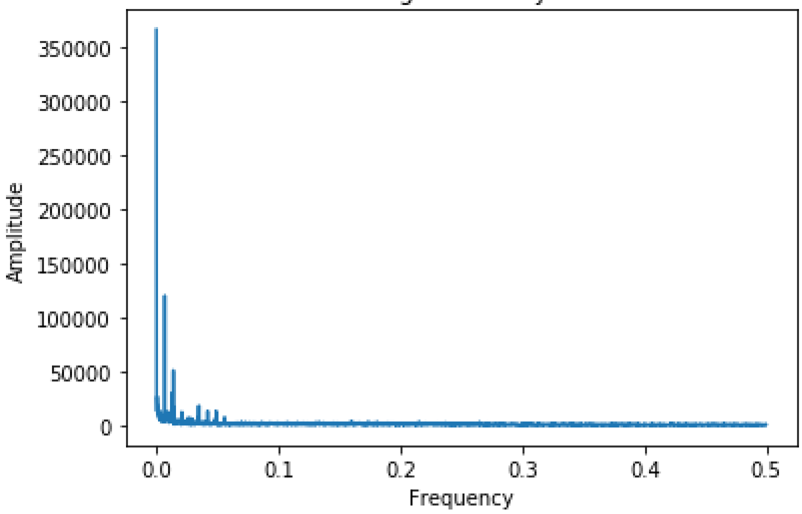
Cluster region 29 for Jan 2016

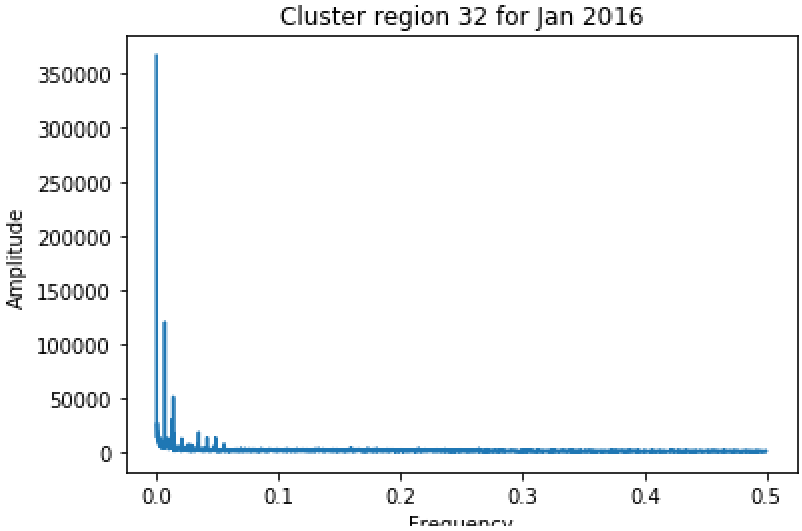


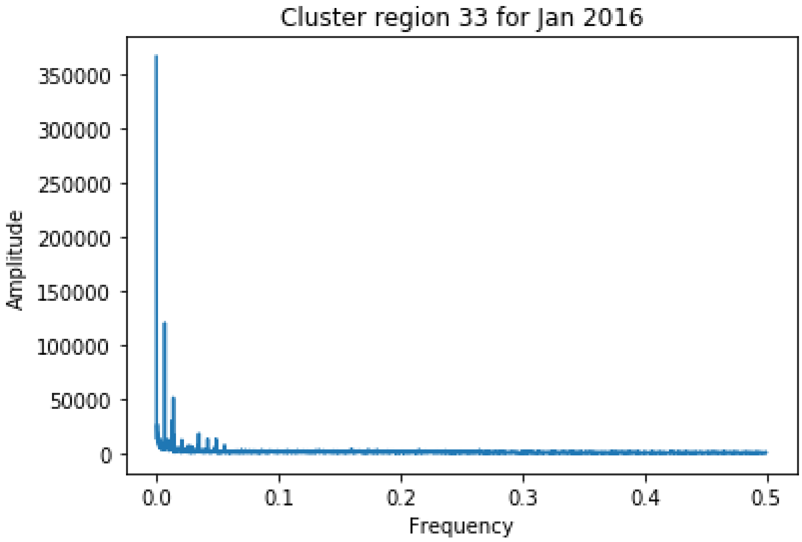
Cluster region 30 for Jan 2016

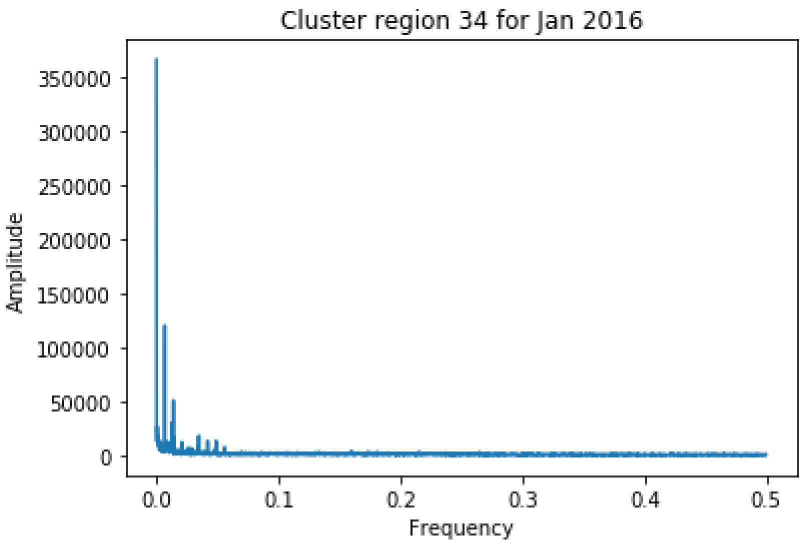


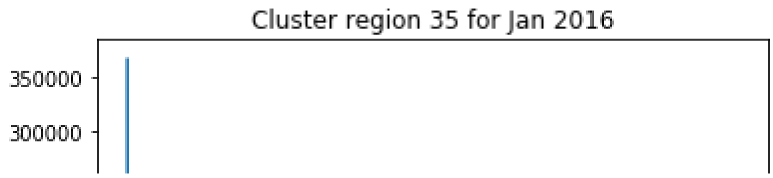
Cluster region 31 for Jan 2016

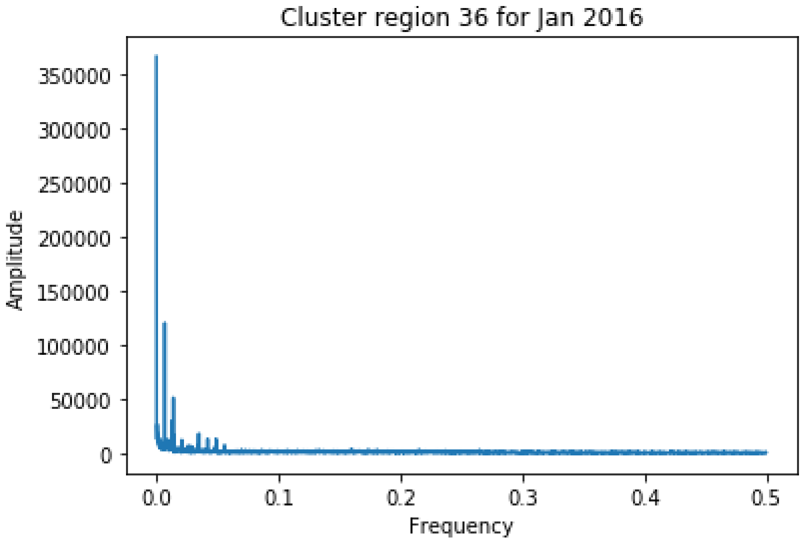


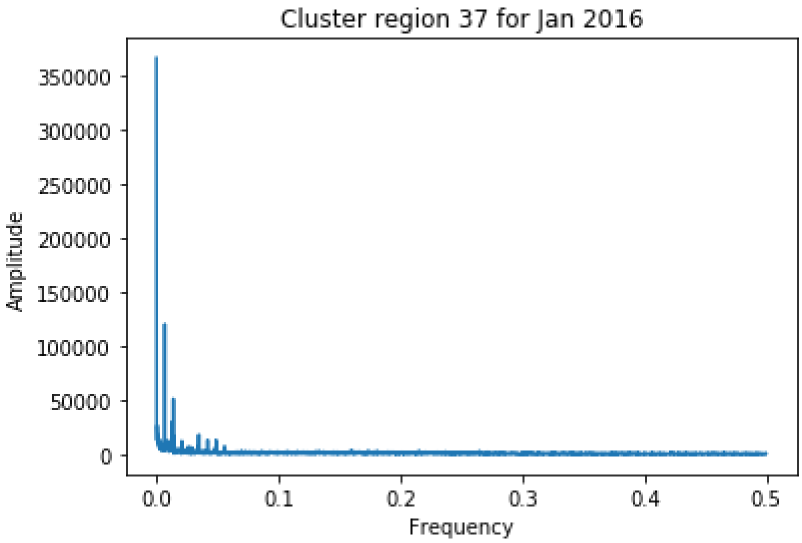

Cluster region 32 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

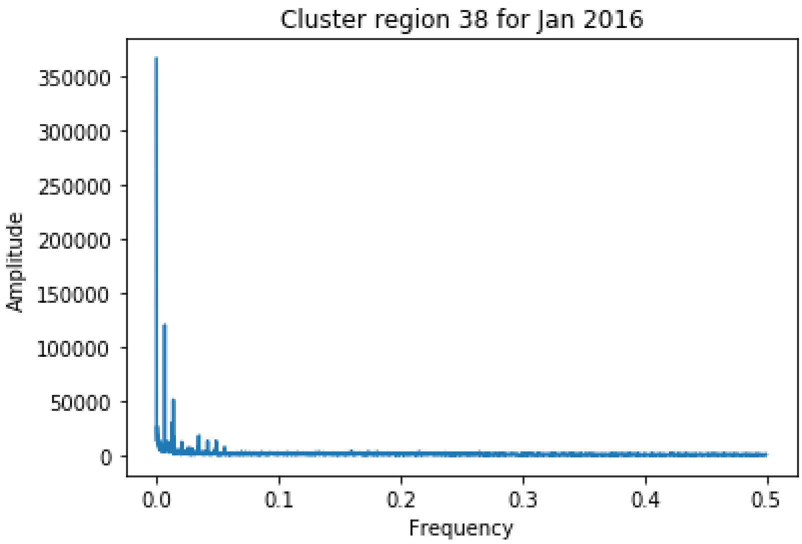

Cluster region 33 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

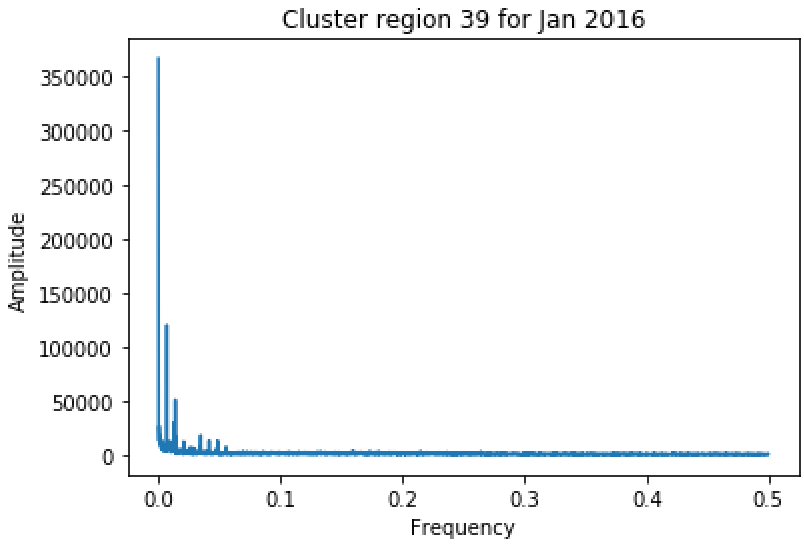

Cluster region 34 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

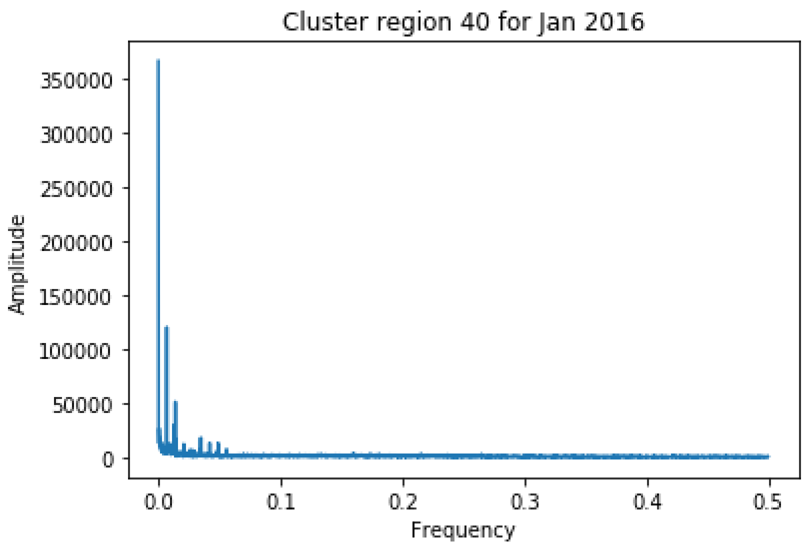

Cluster region 35 for Jan 2016
350000
300000


Cluster region 36 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Cluster region 37 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Cluster region 38 for Jan 2016
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency





In [61]:

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

Modelling: Baseline Models

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

1. Using Ratios of the 2016 data to the 2015 data i.e $R_t = P_t^{2016}/P_t^{2015}$
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 - $R_t = (R_{t-1} + R_{t-2} + R_{t-3} \dots R_{t-n})/n$

In [62]:

```
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)-(rati
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])/window_
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


    ratios['MA_R_Predicted'] = predicted_values
    ratios['MA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Predic
    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 $P_t = (P_{t-1} + P_{t-2} + P_{t-3} \dots P_{t-n})/n$

In [63]:

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

    ratios['MA_P_Predicted'] = predicted_values
    ratios['MA_P_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Predic
    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 [64]:

```
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)-(rati
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

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

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

Weighted Moving Averages using Previous 2016 Values -

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

In [65]:

```
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['Predic
    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 [66]:

```
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)-(rati
        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['Predic
    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 [67]:

```
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'].val

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

In [68]:

```
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 [69]:

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
----------------------------------------------------------------------------
----------------------------
Moving Averages (Ratios) -                             MAPE:  0.227851563531
33512       MSE:  1196.2953853046595
Moving Averages (2016 Values) -                        MAPE:  0.155834587120
25738        MSE:  254.66309363799283
----------------------------------------------------------------------------
----------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.227065291448
71415       MSE:  1053.083529345878
Weighted Moving Averages (2016 Values) -               MAPE:  0.147948218299
2932       MSE:  224.81054547491038
----------------------------------------------------------------------------
----------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.227547463614853
4       MSE:  1019.3071012544802
Exponential Moving Averages (2016 Values) -         MAPE:  0.147538129779815
3       MSE:  222.35159610215055
```

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

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

Regression Models

Train-Test Split

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

In [70]:

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

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

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

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


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


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

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

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


tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13099)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13099)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of pickups
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x1
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [71]:

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

Out[71]:

True

In [72]:

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

# 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 bes
# we will try to add the same exponential weighted moving avarage at t as a feature to our
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.3

# it is a temporary array that store exponential weighted moving avarage for each 10min int
# 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..x1310
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    for i in range(0,13104):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

In [73]:

```
fr_am_final = pd.DataFrame(columns= ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5',
for r in range(0,40):
    YJan = np.fft.fft(np.array(regions_cum[r][0:4464]))
    freqJan = np.fft.fftfreq((4464), 1)


    YFeb = np.fft.fft(np.array(regions_cum[r])[4464:(4176+4464)])
    freqFeb = np.fft.fftfreq((4176), 1)


    YMar = np.fft.fft(np.array(regions_cum[r])[(4176+4464):(4176+4464+4464)])
    freqMar = np.fft.fftfreq((4464), 1)

    fr_am_jan = pd.DataFrame()
    fr_am_feb = pd.DataFrame()
    fr_am_mar = pd.DataFrame()

    fr_am_jan['Frequency'] = freqJan
    fr_am_jan['Amplitude'] = YJan
    fr_am_feb['Frequency'] = freqFeb
    fr_am_feb['Amplitude'] = YFeb
    fr_am_mar['Frequency'] = freqMar
    fr_am_mar['Amplitude'] = YMar

    fr_am_list_jan = []
    fr_am_list_feb = []
    fr_am_list_mar = []

    fr_am_jan_sorted = fr_am_jan.sort_values(by=["Amplitude"], ascending=False)[:5].reset_i
    fr_am_feb_sorted = fr_am_feb.sort_values(by=["Amplitude"], ascending=False)[:5].reset_i
    fr_am_mar_sorted = fr_am_mar.sort_values(by=["Amplitude"], ascending=False)[:5].reset_i

    for i in range(0,5):
        fr_am_list_jan.append(float(fr_am_jan_sorted[i]['Frequency']))
        fr_am_list_jan.append(float(fr_am_jan_sorted[i]['Amplitude']))

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

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

    fr_am_new_jan = pd.DataFrame([fr_am_list_jan]*4464)
    fr_am_new_feb = pd.DataFrame([fr_am_list_feb]*4176)
    fr_am_new_mar = pd.DataFrame([fr_am_list_mar]*4464)

    fr_am_new_jan.columns = ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','a_5',]
    fr_am_new_feb.columns = ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','a_5',]
    fr_am_new_mar.columns = ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','a_5',]


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

In [74]:

```
fr_am_final.drop(['f_1'],axis=1,inplace=True)
fr_am_final = fr_am_final.fillna(0)
```

In [75]:

```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 months of 20
# and split it such that for every region we have 70% data in train and 30% in test,
# ordered date-wise for every region
print("size of train data :", int(13099*0.7))
print("size of test data :", int(13099*0.3))
```

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

In [76]:

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

In [77]:

```
fr_am_final_train = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','
fr_am_final_test = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5','a
for i in range(0,40):
    fr_am_final_train = fr_am_final_train.append(fr_am_final[i*13099:(13099*i+9169)] )
fr_am_final_train.reset_index(inplace=True)
for i in range(0,40):
    fr_am_final_test = fr_am_final_test.append(fr_am_final[(13099*(i))+9169:13099*(i+1)])
fr_am_final_test.reset_index(inplace=True)
```

In [78]:

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

```
Number of data clusters 40 Number of data points in trian data 9169 Each dat
a point contains 5 features
Number of data clusters 40 Number of data points in test data 3930 Each data
point contains 5 features
```

In [79]:

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

In [80]:

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

In [81]:

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

In [82]:

```
# 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 [83]:

```
# 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 [84]:

```
# 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

print(df_train.shape)
```

```
(366760, 9)
```

In [85]:

```
# 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
print(df_test.shape)
```

```
(157200, 9)
```

In [86]:

```
df_test.head()
```

Out[86]:

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

In [89]:

```
df_train.to_csv('new_features_train.csv', index=False)
```

In [90]:

```
df_test.to_csv('new_features_test.csv', index=False)
```

In [87]:

```
df_test_lm = pd.concat([df_test, fr_am_final_test], axis=1)
df_train_lm = pd.concat([df_train, fr_am_final_train], axis=1)

print(df_test_lm.shape)
print(df_train_lm.shape)
```

```
(157200, 19)
(366760, 19)
```

In [92]:

```
df_test_lm.to_csv('fft_test.csv',index=True)
```

In [93]:

```
df_train_lm.to_csv('fft_train.csv',index=True)
```

In [88]:

```
df_test_lm.head()
```

Out[88]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	index	a_1	
0	143	145	119	113	124	40.776228	-73.982119	4	121	9169	387761.0	0.006
1	145	119	113	124	121	40.776228	-73.982119	4	120	9170	387761.0	0.006
2	119	113	124	121	131	40.776228	-73.982119	4	127	9171	387761.0	0.006
3	113	124	121	131	110	40.776228	-73.982119	4	115	9172	387761.0	0.006
4	124	121	131	110	116	40.776228	-73.982119	4	115	9173	387761.0	0.006

Using Linear Regression

Hyperparameter tuning for Linear regression

In [89]:

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

lr=LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False
for i in tqdm(parameters):
    grid = GridSearchCV(lr,parameters, cv=2)

grid.fit(df_train_lm, tsne_train_output)

print(grid.best_estimator_)
print(grid.best_params_)
```

```
100%|██████████| 3/3 [00:00<00:00, 14648.33it/s]

LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None,
         normalize=True)
{'copy_X': True, 'fit_intercept': False, 'normalize': True}
```

In [90]:

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

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

from sklearn.linear_model import LinearRegression
lr=LinearRegression(copy_X =True,normalize=True,fit_intercept=False).fit(df_train_lm, tsne_

y_pred = lr.predict(df_test_lm)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = lr.predict(df_train_lm)
lr_train_predictions = [round(value) for value in y_pred]
```

Using Random Forest Regressor

In [91]:

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

regr1 = RandomForestRegressor()

param_dist = {"max_depth": [3,5,7,9],
              "max_features": ['sqrt' , 'log2' ],
              "min_samples_split": sp_randint(2,5,11),
              "min_samples_leaf": sp_randint(1,4,11),
              "n_estimators":[35,40,45,50,55,60]
              }

# run randomized search
n_iter_search = 20
for i in tqdm(param_dist):
    random_search = RandomizedSearchCV(regr1, param_distributions=param_dist,n_iter=n_iter_

    random_search.fit(df_train_lm, tsne_train_output)
```

```
100%|██████████| 5/5 [37:50<00:00, 454.06s/it]
```

In [92]:

```
random_search.best_params_
```

Out[92]:

```
{'max_depth': 9,
 'max_features': 'log2',
 'min_samples_leaf': 13,
 'min_samples_split': 13,
 'n_estimators': 60}
```

In [93]:

```
random_search.best_score_
```

Out[93]:

```
0.9456052921297403
```

In [94]:

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

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

regr1 = RandomForestRegressor(max_features='log2',min_samples_leaf=13,min_samples_split=13,
regr1.fit(df_train_lm, tsne_train_output)
```

Out[94]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=9,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=14, min_samples_split=14,
           min_weight_fraction_leaf=0.0, n_estimators=60, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [95]:

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

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

y_pred = regr1.predict(df_test_lm)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr1.predict(df_train_lm)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [96]:

```
#feature importances based on analysis using random forest
print (df_train_lm.columns)
print (regr1.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg', 'index', 'a_1', 'f_2', 'a_2', 'f_3', 'a_3', 'f_4', 'a_4',
       'f_5', 'a_5'],
      dtype='object')
[8.78686717e-02 8.23928256e-02 1.58246176e-01 1.49019712e-01
 1.57397058e-01 1.33454649e-03 5.13094367e-04 4.42612224e-05
 3.24307383e-01 5.26899506e-04 1.97862568e-02 4.98742175e-05
 8.01393915e-03 1.72379003e-04 7.31715605e-03 1.01336164e-04
 1.08435470e-03 1.30615514e-04 1.69346052e-03]
```

Using XgBoost Regressor

In [97]:

```
import xgboost as xgb

param = {"max_depth": [3, 4,5],
              "min_child_weight": [3, 4,5,6],
              "gamma":[0,0.1,0.2],
              "colsample_bytree":[0.7,0.8,0.9],
              "nthread":[3,4,5],'learning_rate': [ 0.01, 0.1, 0.2]
              }
n_iter_search = 10
x_model = xgb.XGBRegressor()

for i in tqdm(param):
    random_xgb = RandomizedSearchCV(x_model, param_distributions=param,n_iter=n_iter_search

    random_xgb.fit(df_train_lm, tsne_train_output)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]

[23:32:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 17%|█▋        | 1/6 [02:47<13:57, 167.41s/it]

[23:35:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 33%|███▎      | 2/6 [05:35<11:10, 167.71s/it]

[23:37:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 50%|█████     | 3/6 [08:25<08:25, 168.42s/it]

[23:40:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 67%|██████▋   | 4/6 [11:18<05:39, 169.72s/it]

[23:43:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 83%|████████▎ | 5/6 [14:05<02:48, 168.72s/it]

[23:46:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

100%|██████████| 6/6 [16:54<00:00, 169.03s/it]
```

In [98]:

```
random_xgb.best_score_
```

Out[98]:

```
-8.629914599005286
```

In [99]:

```
random_xgb.best_params_
```

Out[99]:

```
{'colsample_bytree': 0.9,
 'gamma': 0.1,
 'learning_rate': 0.1,
 'max_depth': 4,
 'min_child_weight': 6,
 'nthread': 4}
```

In [100]:

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

# find more about XGBRegressor function here http://xgboost.readthedocs.io/en/latest/python
# -------------------------
# default paramters
# xgboost.XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objec
# booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0,
# colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, rando
# missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This functio
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regr
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what
# -----------------------

x_model = xgb.XGBRegressor(
 learning_rate =0.1,
 n_estimators=1000,
 max_depth=4,
 min_child_weight=6,
 gamma=0,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=0.9,nthread=4)
x_model.fit(df_train_lm, tsne_train_output)
```

```
[23:46:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

Out[100]:

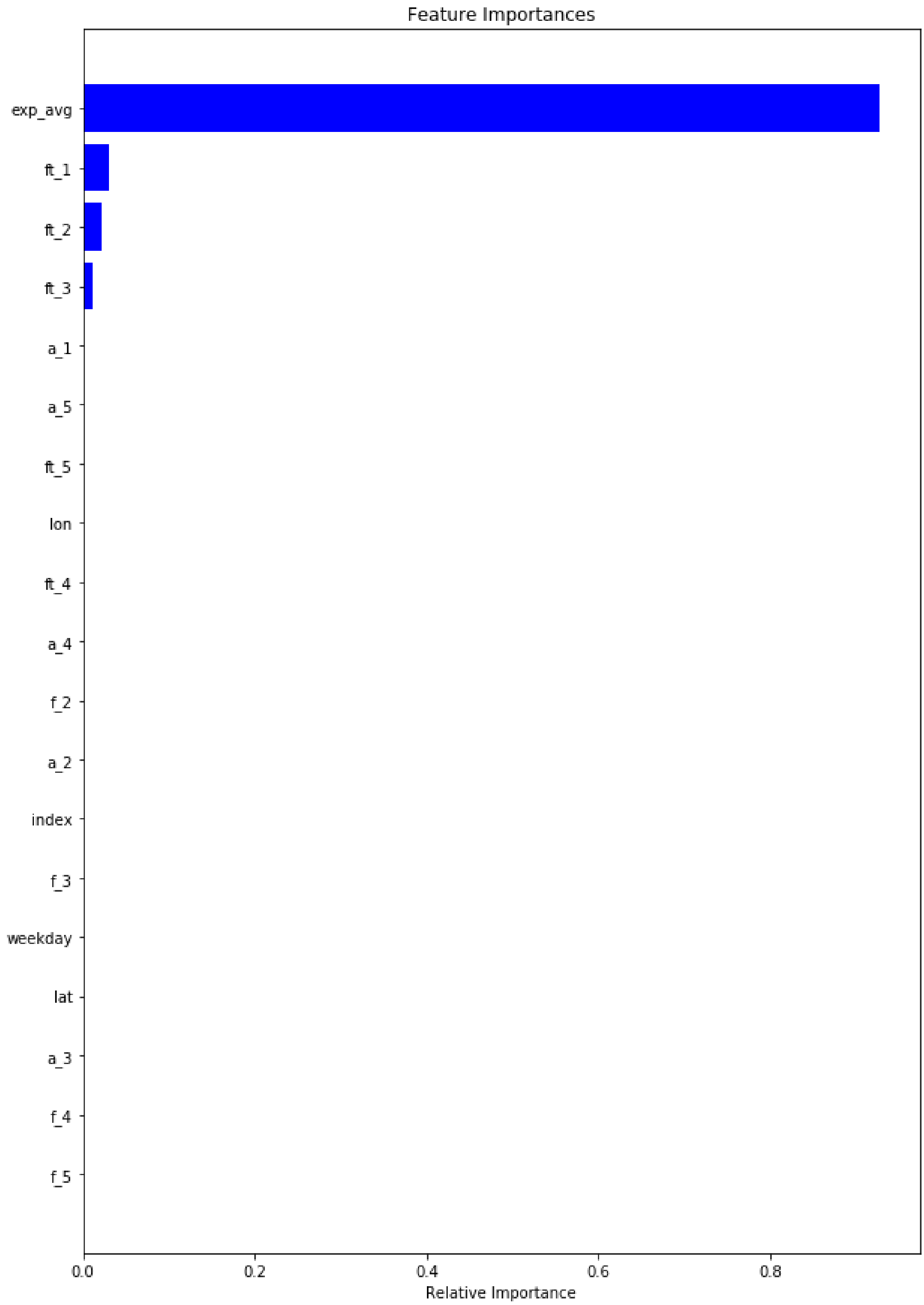
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.9, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=6, missing=None, n_estimators=1000,
       n_jobs=1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=200, reg_lambda=200, scale_pos_weight=1, seed=None,
       silent=None, subsample=0.8, verbosity=1)
```

In [101]:

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

y_pred = x_model.predict(df_test_lm)
xgb_test_predictions = [round(value) for value in y_pred]
y_pred = x_model.predict(df_train_lm)
xgb_train_predictions = [round(value) for value in y_pred]
```

In [102]:

```
#feature importances
features = df_train_lm.columns
importances = x_model.feature_importances_
indices = (np.argsort(importances))[-19:]
plt.figure(figsize=(10,15))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Calculating the error metric values for various models

In [103]:

```
train_mape=[]
test_mape=[]

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

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/(sum(tsne_
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(sum(ts
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_t
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_te
test_mape.append((mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_tes
```

In [104]:

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

```
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:  0.13781520088635468
Test:  0.13258126782951432
Random Forest Regression -                   Train:  0.13933052052284461
Test:  0.13426255167894308
```

Error Metric Matrix

In [105]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("---------------------------------------------------------------------------------------
print ("Baseline Model -                                Train: ",train_mape[0],"      Test: ",
print ("Exponential Averages Forecasting -              Train: ",train_mape[1],"      Test: ",
print ("Linear Regression -                            Train: ",train_mape[4],"      Test: ",t
print ("Random Forest Regression -                      Train: ",train_mape[2],"     Test: ",t
print ("XgBoost Regression -                            Train: ",train_mape[3],"      Test: ",
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:  0.14210483617977826
Test:  0.13468579269091166
Random Forest Regression -                   Train:  0.13933052052284461
Test:  0.13426255167894308
XgBoost Regression -                         Train:  0.13781520088635468
Test:  0.13258126782951432
---------------------------------------------------------------------------
----------------------------
```

Assignments

Task 1: Incorporate Fourier features as features into Regression models and measure MAPE.

Task 2: Perform hyper-parameter tuning for Regression models. 2a. Linear Regression: Grid Search 2b. Random Forest: Random Search 2c. Xgboost: Random Search

Task 3: Explore more time-series features using Google search/Quora/Stackoverflow to reduce the MAPE to < 12%

Double Exponentional Smoothing

[https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-ii/ (https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-ii/)](https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-ii/)

In [106]:

```
def double_exponential_smoothing(series, alpha, beta):
    result = [series[0]]
    for n in range(1, len(series)+1):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # we are forecasting
            value = result[-1]
        else:
            value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result
```

In [107]:

```
alpha = 0.3
beta = 0.15
predict_values_double =[]
predict_list_double = []
for r in range(0,40):
    predict_values_double = double_exponential_smoothing(regions_cum[r][0:13104], alpha, be
    predict_list_double.append(predict_values_double[5:])
```

Triple Exponentional Smoothing

[https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/ (https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/)](https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/)

In [108]:

```
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen

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

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

In [109]:

```
alpha = 0.3
beta = 0.15
gamma = 0.2
season_len = 24

predict_values_triple =[]
predict_list_triple = []
for r in range(0,40):
    predict_values_triple = triple_exponential_smoothing(regions_cum[r][0:13104], season_le
    predict_list_triple.append(predict_values_triple[5:])
```

In [110]:

```
tsne_train_flat_double_exp = [i[:9169] for i in predict_list_double]
tsne_train_flat_triple_exp = [i[:9169] for i in predict_list_triple]
tsne_test_flat_double_exp = [i[9169:] for i in predict_list_double]
tsne_test_flat_triple_exp = [i[9169:] for i in predict_list_triple]
```

In [111]:

```
tsne_train_double_exp = sum(tsne_train_flat_double_exp,[])
tsne_train_triple_exp = sum(tsne_train_flat_triple_exp,[])
```

In [112]:

```
df_train['Double_exponential'] = tsne_train_double_exp
df_train['Triple_exponential'] = tsne_train_triple_exp
print(df_train.shape)
```

```
(366760, 11)
```

In [113]:

```
tsne_test_double_exp = sum(tsne_test_flat_double_exp,[])
tsne_test_triple_exp = sum(tsne_test_flat_triple_exp,[])
```

In [114]:

```
len(tsne_test_exp_avg)
```

Out[114]:

```
157200
```

In [115]:

```
tsne_test_double_exp = tsne_test_double_exp[0:157200]
```

In [116]:

```
df_test['Double_exponential'] = tsne_test_double_exp
df_test['Triple_exponential'] = tsne_test_triple_exp
print(df_test.shape)
```

```
(157200, 11)
```

In [117]:

```
df_test.head()
```

Out[117]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	Double_exponential	Tr
0	143	145	119	113	124	40.776228	-73.982119	4	121	121.077767	
1	145	119	113	124	121	40.776228	-73.982119	4	120	123.844168	
2	119	113	124	121	131	40.776228	-73.982119	4	127	118.857661	
3	113	124	121	131	110	40.776228	-73.982119	4	115	117.038511	
4	124	121	131	110	116	40.776228	-73.982119	4	115	123.998373	

In [107]:

```
df_train.to_csv('exp_d_train.csv',index=True)
df_test.to_csv('exp_d_test.csv',index=True)
```

model on Exponentinal Smoothing

On Linear regression

In [108]:

```
df_train = pd.read_csv('exp_d_train.csv',index_col=0)
```

In [109]:

```
df_train.head()
```

Out[109]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	Double_exponential	Tr
0	0	0	0	0	0	40.776228	-73.982119	4	0	0.0	
1	0	0	0	0	0	40.776228	-73.982119	4	0	0.0	
2	0	0	0	0	0	40.776228	-73.982119	4	0	0.0	
3	0	0	0	0	0	40.776228	-73.982119	4	0	0.0	
4	0	0	0	0	0	40.776228	-73.982119	4	0	0.0	

In [110]:

```
df_test = pd.read_csv('exp_d_test.csv',index_col=0)
```

In [111]:

```
df_test.head()
```

Out[111]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	Double_exponential	Tr
0	143	145	119	113	124	40.776228	-73.982119	4	121	121.077767	
1	145	119	113	124	121	40.776228	-73.982119	4	120	123.844168	
2	119	113	124	121	131	40.776228	-73.982119	4	127	118.857661	
3	113	124	121	131	110	40.776228	-73.982119	4	115	117.038511	
4	124	121	131	110	116	40.776228	-73.982119	4	115	123.998373	

In [118]:

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



lr_exp=LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False
for i in tqdm(parameters):
    grid = GridSearchCV(lr_exp,parameters, cv=2)

grid.fit(df_train, tsne_train_output)

print(grid.best_estimator_)
print(grid.best_params_)
```

```
100%|██████████| 3/3 [00:00<00:00, 4128.25it/s]

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Tru
e)
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

In [119]:

```
lr_exp=LinearRegression(copy_X =True,normalize=True,fit_intercept=True).fit(df_train, tsne_

y_pred = lr_exp.predict(df_test)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = lr_exp.predict(df_train)
lr_train_predictions = [round(value) for value in y_pred]
```

On Random Forest Regressor

In [120]:

```
regr_exp = RandomForestRegressor()

param_dist = {"max_depth": [3,5,7,9],
              "max_features": ['sqrt' , 'log2' ],
              "min_samples_split": sp_randint(2,5,11),
              "min_samples_leaf": sp_randint(1,4,11),
              "n_estimators":[35,40,45,50,55,60]
              }

# run randomized search
n_iter_search = 20
for i in tqdm(param_dist):
    random_search_exp = RandomizedSearchCV(regr_exp, param_distributions=param_dist,n_iter=

    random_search_exp.fit(df_train, tsne_train_output)
```

```
100%|██████████| 5/5 [37:33<00:00, 450.72s/it]
```

In [121]:

```
random_search_exp.best_score_
```

Out[121]:

```
0.9794293943595431
```

In [122]:

```
random_search_exp.best_params_
```

Out[122]:

```
{'max_depth': 9,
 'max_features': 'log2',
 'min_samples_leaf': 12,
 'min_samples_split': 14,
 'n_estimators': 40}
```

In [123]:

```
regr_exp = RandomForestRegressor(max_features='log2',min_samples_leaf=12,min_samples_split=
regr_exp.fit(df_train, tsne_train_output)
```

Out[123]:

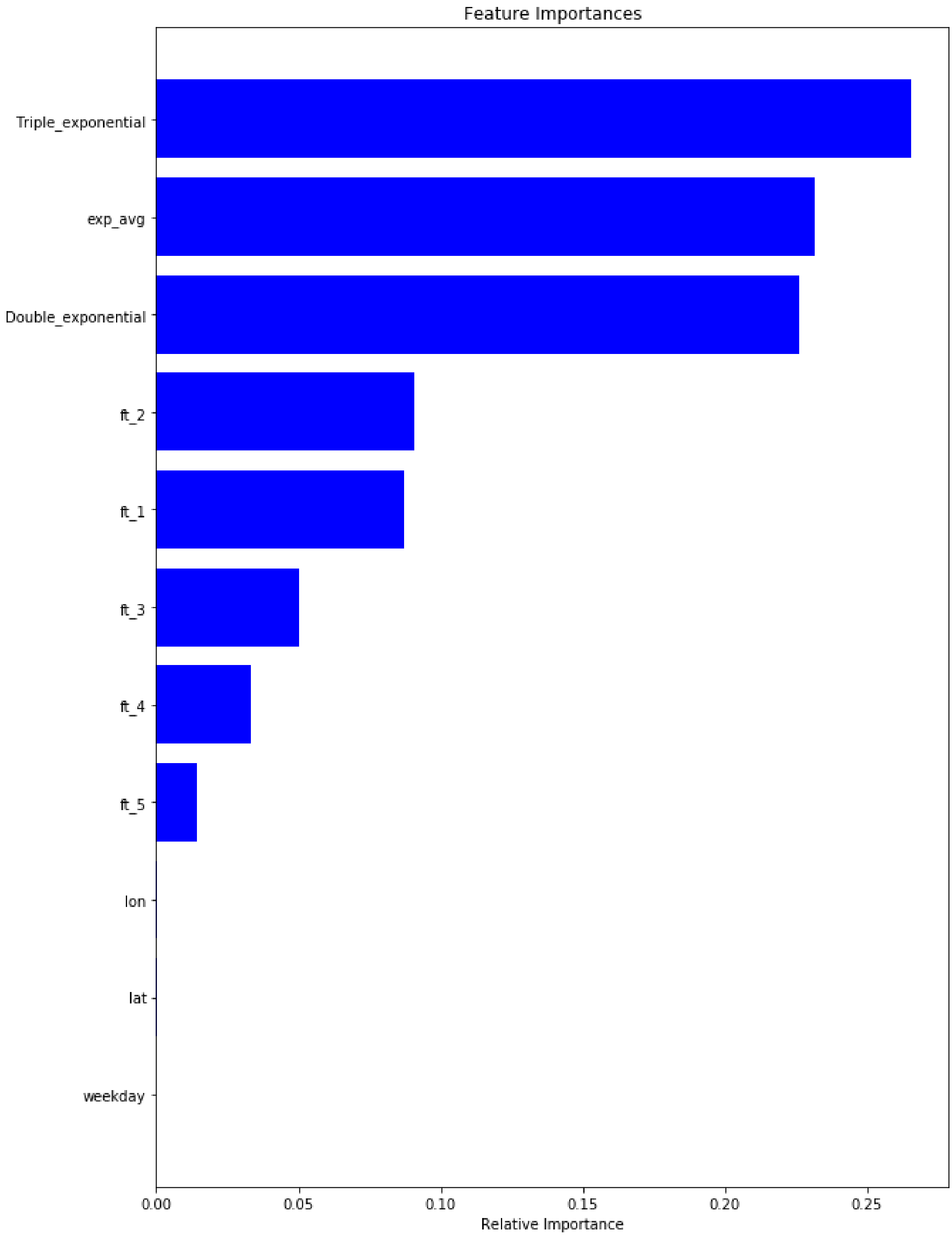
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=9,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=12, min_samples_split=14,
           min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [124]:

```
y_pred = regr_exp.predict(df_test)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr_exp.predict(df_train)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [125]:

```
#feature importances
features = df_train.columns
importances = regr_exp.feature_importances_
indices = (np.argsort(importances))[-11:]
plt.figure(figsize=(10,15))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



On XGBoost Regressor

In [126]:

```
param = {"max_depth": [3, 4,5],
              "min_child_weight": [3, 4,5,6],
              "gamma":[0,0.1,0.2],
              "colsample_bytree":[0.7,0.8,0.9],
              "nthread":[3,4,5],'learning_rate': [ 0.01, 0.1, 0.2]
              }
n_iter_search = 10
x_model = xgb.XGBRegressor()

for i in tqdm(param):
    random_xgb_exp = RandomizedSearchCV(x_model, param_distributions=param,n_iter=n_iter_se

    random_xgb_exp.fit(df_train, tsne_train_output)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]

[00:29:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 17%|█▋        | 1/6 [04:14<21:14, 254.91s/it]

[00:34:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 33%|███▎      | 2/6 [08:50<17:24, 261.15s/it]

[00:38:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 50%|█████     | 3/6 [13:01<12:54, 258.07s/it]

[00:43:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 67%|██████▋   | 4/6 [17:27<08:41, 260.54s/it]

[00:47:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

 83%|████████▎ | 5/6 [21:36<04:17, 257.04s/it]

[00:51:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.

100%|██████████| 6/6 [26:05<00:00, 260.99s/it]
```

In [127]:

```
random_xgb_exp.best_params_
```

Out[127]:

```
{'colsample_bytree': 0.9,
 'gamma': 0,
 'learning_rate': 0.1,
 'max_depth': 5,
 'min_child_weight': 3,
 'nthread': 4}
```

In [128]:

```
xgb_exp = xgb.XGBRegressor(
 learning_rate = random_xgb.best_estimator_.learning_rate,
 n_estimators= random_xgb.best_estimator_.n_estimators,
 max_depth=random_xgb.best_estimator_.max_depth,
 min_child_weight=random_xgb.best_estimator_.min_child_weight,
 gamma=random_xgb.best_estimator_.gamma,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=random_xgb.best_estimator_.colsample_bytree,nthread=random_xgb.best_estim
xgb_exp.fit(df_train, tsne_train_output)
```

```
[00:52:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

Out[128]:

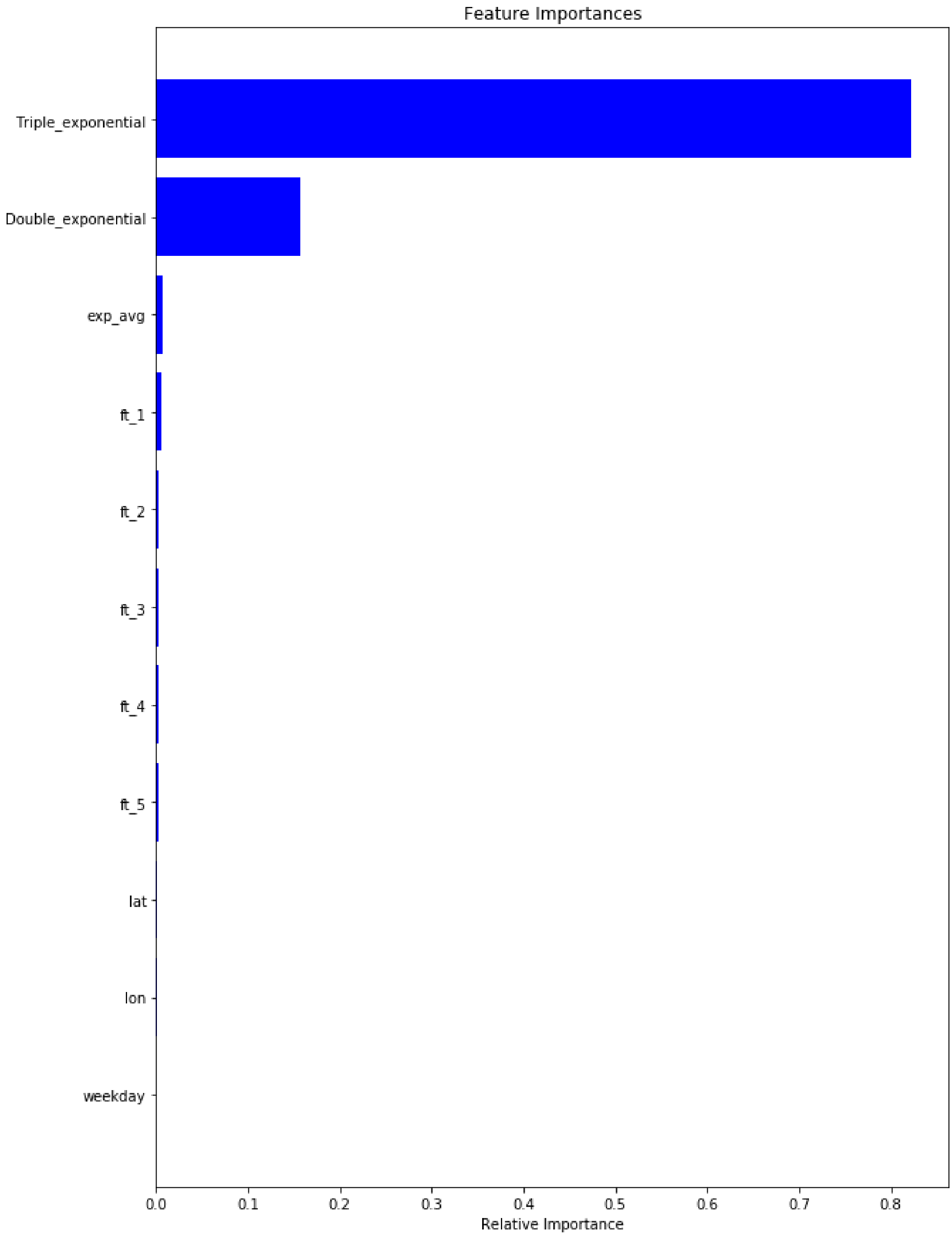
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.9, gamma=0.1,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=4, min_child_weight=6, missing=None, n_estimators=100,
       n_jobs=1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=200, reg_lambda=200, scale_pos_weight=1, seed=None,
       silent=None, subsample=0.8, verbosity=1)
```

In [129]:

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

In [130]:

```
#feature importances
features = df_train.columns
importances = xgb_exp.feature_importances_
indices = (np.argsort(importances))[-11:]
plt.figure(figsize=(10,15))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



In [131]:

```
train_mape=[]
test_mape=[]

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

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/(sum(tsne_
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(sum(ts
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(tsne_t
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsne_te
test_mape.append((mean_absolute_error(tsne_test_output, lr_test_predictions))/(sum(tsne_tes
```

In [132]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------------------
print ("Baseline Model -                                Train: ",train_mape[0],"      Test: ",
print ("Exponential Averages Forecasting -              Train: ",train_mape[1],"      Test: ",
print ("Linear Regression -                            Train: ",train_mape[4],"      Test: ",t
print ("Random Forest Regression -                      Train: ",train_mape[2],"     Test: ",t
print ("XgBoost Regression -                            Train: ",train_mape[3],"      Test: ",
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:  0.10392341326296872
Test:  0.09678896150975777
Random Forest Regression -                   Train:  0.08963402121268307
Test:  0.1801144526502323
XgBoost Regression -                         Train:  0.09071616200136332
Test:  0.11534471351844328
----------------------------------------------------------------------------
----------------------------
```

Both Linear Regression and Xgboost performed well with MAPE of 0.096 and 0.115 respectively.

Triple Exponentinal features has the highest importance in Random Forest and XGBoost Regression models.

Between the train and test models in Random forest we see large differences in the value which shows Overfitting model.

In [ ]: