Taxi demand prediction in New York City

Predict the taxi demand for yellow cabs with the location in next 10 minutes for new york city.

This python notebook is to develop machine learning model to predict the taxi demand for yellow cabs in new york city with the data provided by the Taxi & Limousine Commission for yellow cabs. Based on the data, machine learning model predicts the pickup demand of cabs in 10 minutes time frame. In this python notebook different machine learning model have been trained and accuracy is tested.

Data Overview

- pick-up and drop-off dates/times,
- pick-up and drop-off locations,
- trip distances,
- itemized fares,
- rate types,
- payment types,
- driver-reported passenger counts

With the given data first, we will do the data cleaning and convert data into the required format.

To divide new york city into the region so that prediction can be done region vise, we will use K-means algorithm.

Feature importance is an important part for any of the machine learning problem. Here we will use below baseline model by generating feature with ratio and previous value at a time (t-1) and will calculate Mean Absolute Percentage Error.

- Moving Averages
- Weighted Moving Averages
- Exponential Moving Averages

Along with that, we will use below regression model by selecting best hyper-parameter with the help of different technique depending on hype parameter to predict the taxi demand.

- Linear Regression with GridSearch
- Random Forest Regressor with Random search
- XgBoost Regressor with Random search

Objective: By comparing the different model we will select the best model to predict the Yellow Taxi demand which helps the taxi drivers.

This project is developed with the help of videos and basic code provided by appliedaicourse.com.

In [95]:
```python
import warnings
warnings.filterwarnings("ignore")
import os
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mi
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']
import datetime
import time
import numpy as np
import gpxpy.geo
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
matplotlib.use('nbagg')
import dask.dataframe as dd
import pandas as pd
import scipy
import folium
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
import scipy
```

Data Information

Data is downloaded from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml (http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml) (2016 data)

In [7]:
```python
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_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

Features in the dataset:

Field Name	Description

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

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 [8]:
```
#table below shows few datapoints along with all our features
month.head(5)
```

Out[8]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_l
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	-7
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30	-7
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80	-7
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50	-7
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00	-7

1. Pickup Latitude and Pickup Longitude

As per the 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).

In [9]:
```
outlier_locations = month[((month.pickup_longitude <= -74.15) | (month.pickup_lat
                   (month.pickup_longitude >= -73.7004) | (month.pickup_latitude

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

Out[9]:

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

In [10]:
```
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_l
                    (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitud
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
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_t
map_osm
```

Out[10]:

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

3. Trip Durations:

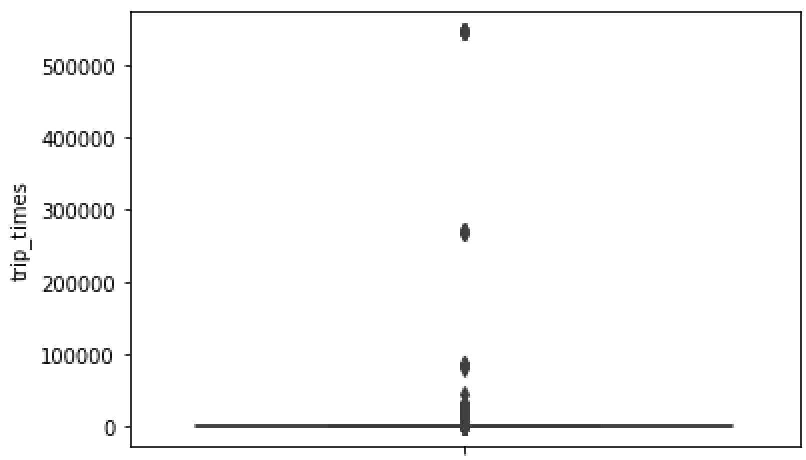
We assume that, the maximum allowed trip duration in a 24 hour interval is 12 hours.

In [11]:

```
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetup
def return_with_trip_times(month):
    duration = month[['tpep_pickup_datetime','tpep_dropoff_datetime']].compute()
    duration_pickup = [convert_to_unix(x) for x in duration['tpep_pickup_datetime
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)
    new_frame = month[['passenger_count','trip_distance','pickup_longitude','pick
    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
frame_with_durations = return_with_trip_times(month)
```

In [12]:

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



In [13]:

```
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])
```

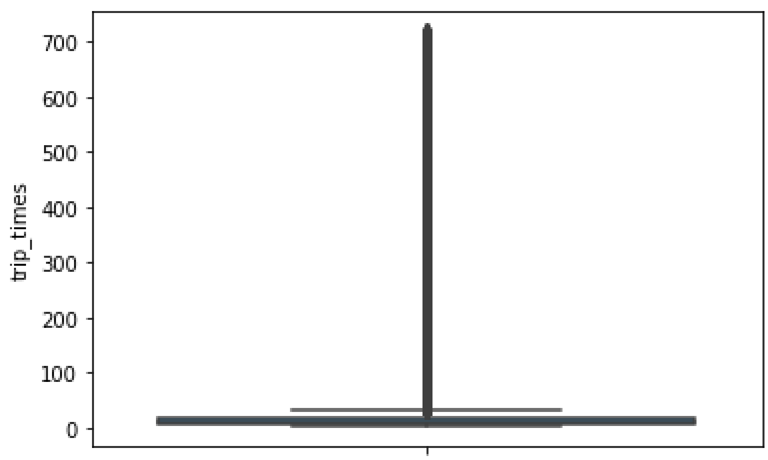
```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333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

In [14]:
```
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])
```

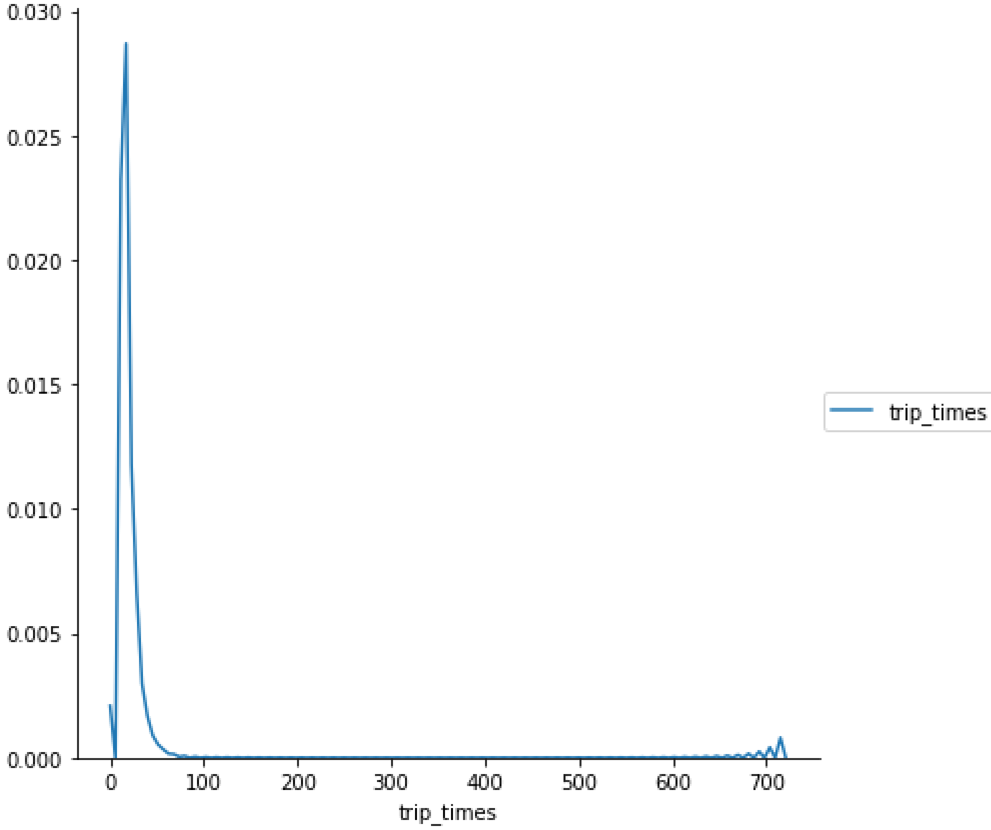
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98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

In [15]:
```
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_tim
```

In [16]:
```
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.show()
```

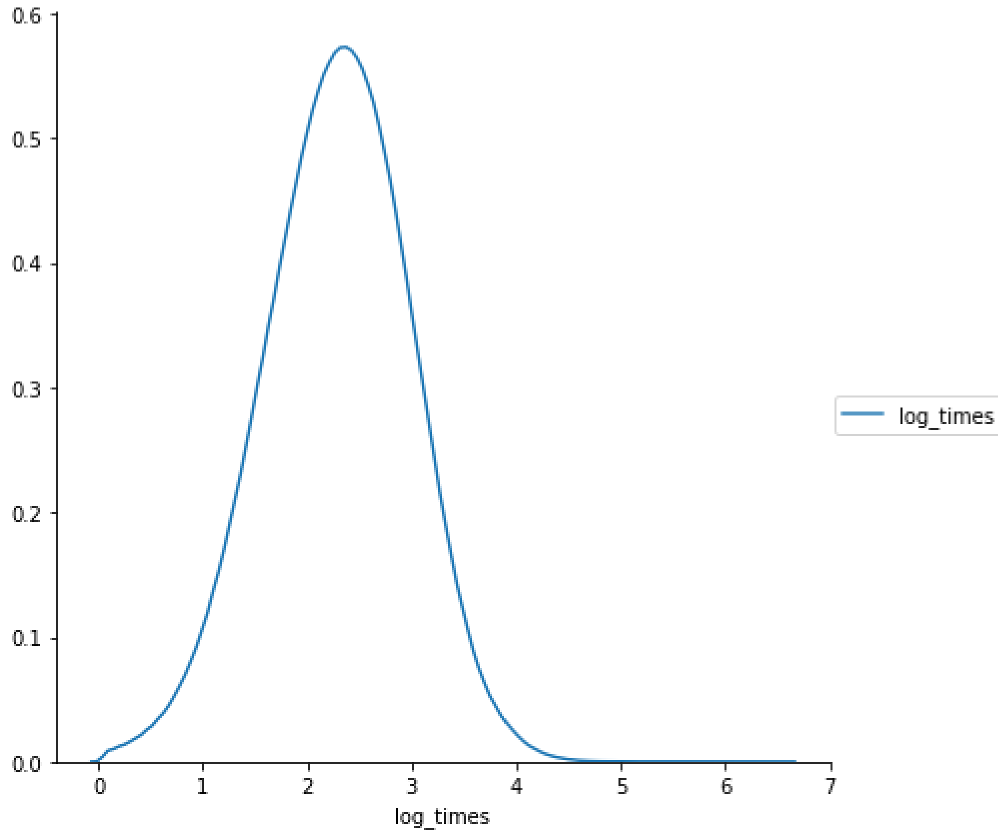


In [17]:
```
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```

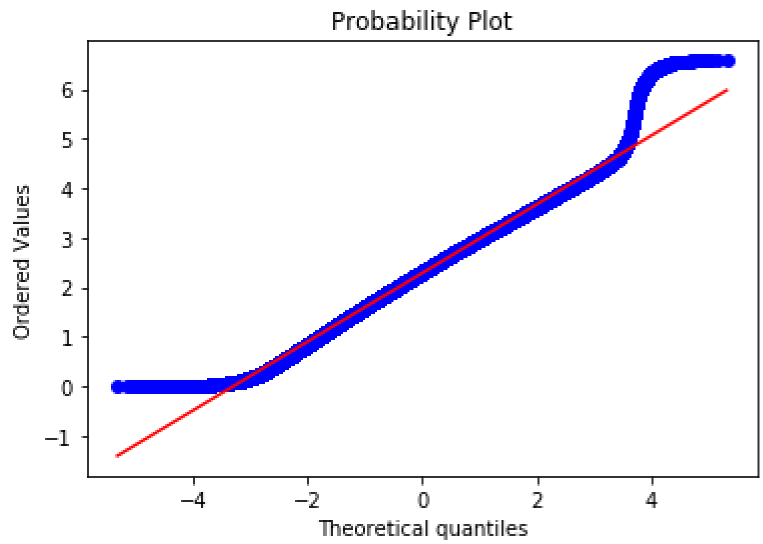


In [18]:
```
import math
frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_durat
```

In [19]:
```
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```



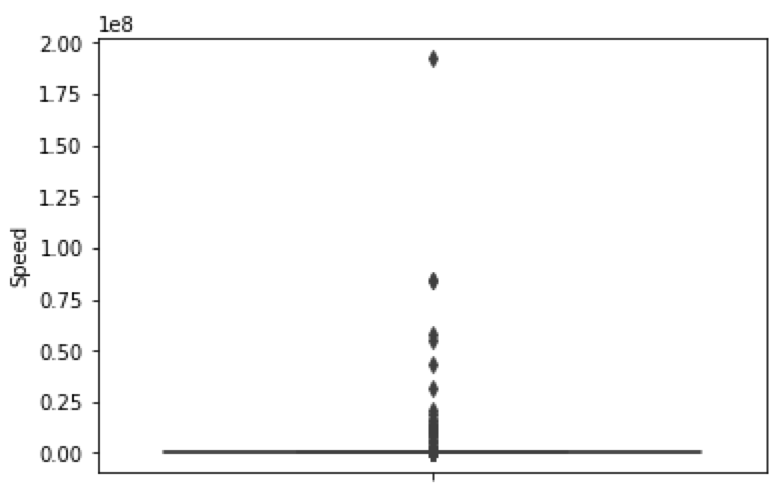
In [20]:
```
#Q-Q plot for checking if trip-times is log-normal
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



Both tails in out of line in Q-Q plot which indicate that trip time which we have converted in logarithmic function is not following Gaussian distribution

4. Speed

In [21]:
```
frame_with_durations_modified['Speed'] = 60*(frame_with_durations_modified['trip_
sns.boxplot(y="Speed", data =frame_with_durations_modified)
plt.show()
```



In [22]:
```
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 [23]:
```
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 [24]:
```
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)/1
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 [25]:
```
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0)
```

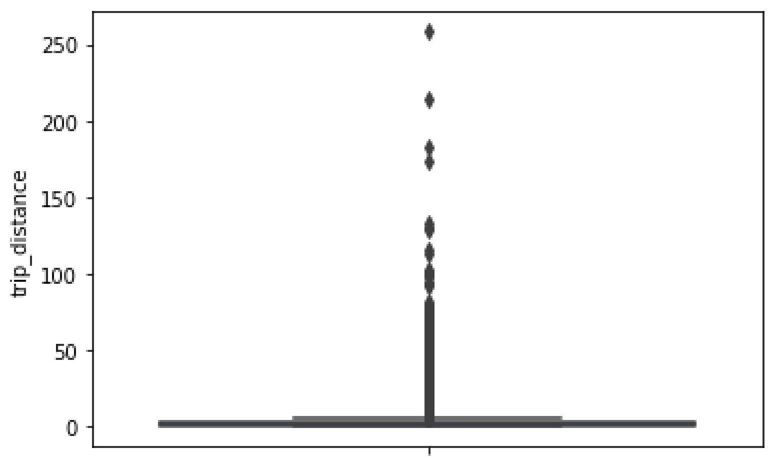
In [26]:
```
sum(frame_with_durations_modified["Speed"]) / float(len(frame_with_durations_modi
```

Out[26]:
```
12.450173996028015
```

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 [27]:
```
sns.boxplot(y="trip_distance", data =frame_with_durations_modified)
plt.show()
```



In [28]:
```
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 [29]:

```
#calculating trip distance values at each percntile 90,91,92,93,94,95,96,97,98,99
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 [30]:

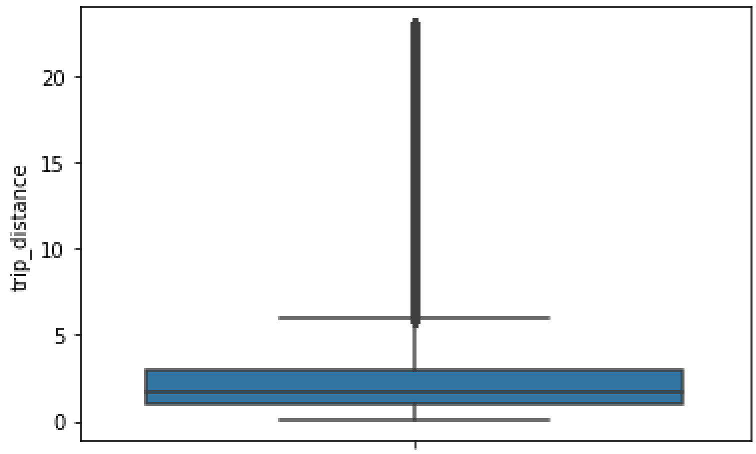
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations_modified["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/1
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 [31]:

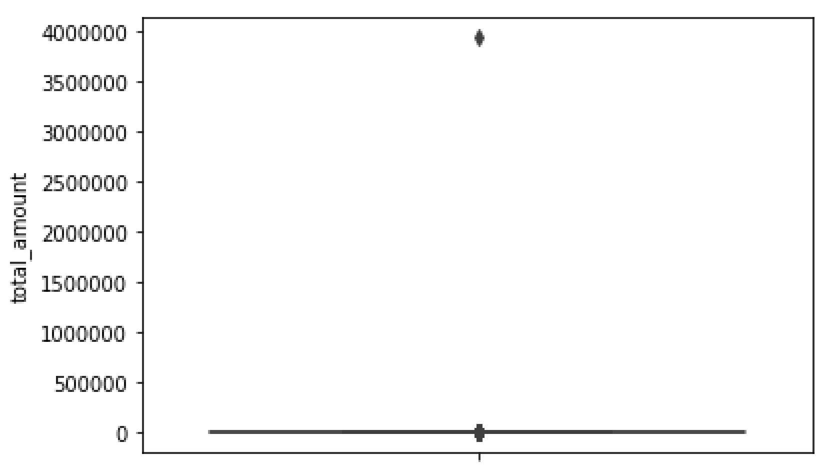
```
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_dis
```

In [32]:
```
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [33]:
```
sns.boxplot(y="total_amount", data =frame_with_durations_modified)
plt.show()
```



In [34]:
```
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 [35]:
```
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])
```

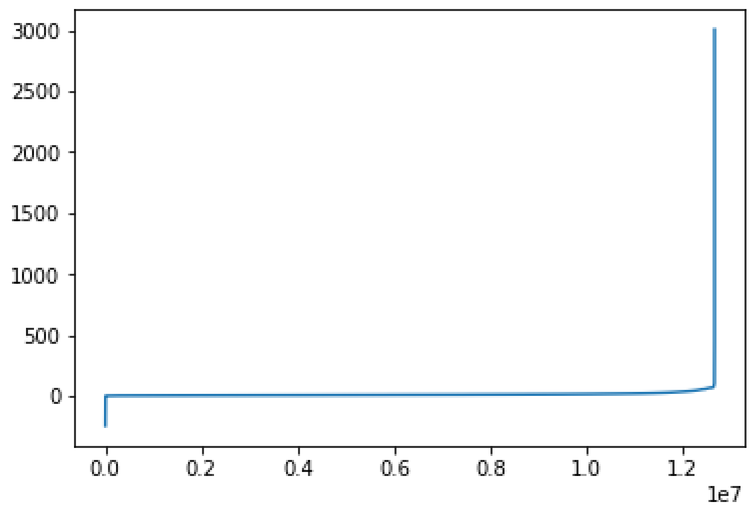
```
90 percentile value is 25.8
91 percentile value is 27.3
92 percentile value is 29.3
93 percentile value is 31.8
94 percentile value is 34.8
95 percentile value is 38.53
96 percentile value is 42.6
97 percentile value is 48.13
98 percentile value is 58.13
99 percentile value is 66.13
100 percentile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

In [36]:
```
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)/1
print("100 percentile value is ",var[-1])
```

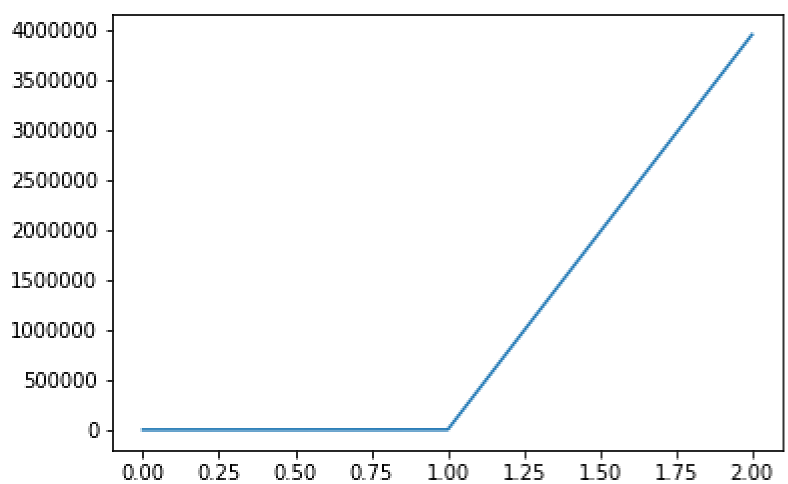
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value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

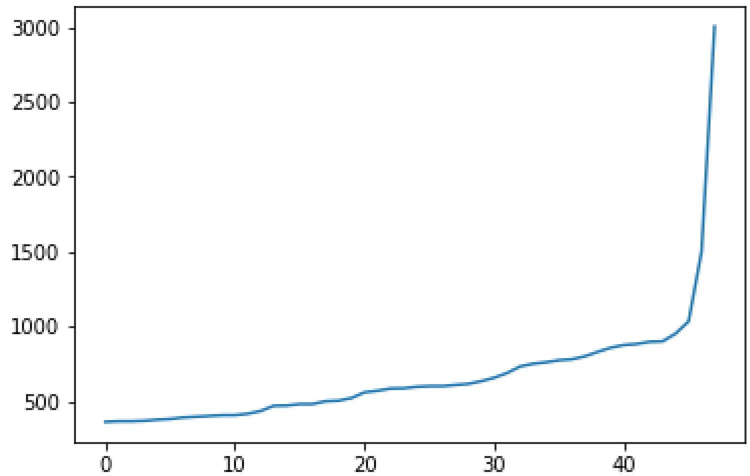
In [37]:
```
plt.plot(var[:-2])
plt.show()
```



In [38]:
```
plt.plot(var[-3:])
plt.show()
```



In [39]:
```
plt.plot(var[-50:-2])
plt.show()
```



Remove all outliers/erronous points.

In [40]:
```
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.
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropo
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.picku
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pick
    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 < 7
    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_distan
    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_amou
    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.d
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropo
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.picku
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pick

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 72
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distanc
    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_amoun

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

In [41]:
```
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(fr
```

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

Data-preperation

Clustering/Segmentation

In [42]:

```
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longit
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], cl
                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 Cluster

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

for increment in range(10, 100, 10):
    cluster_centers, cluster_len = find_clusters(increment)
    find_min_distance(cluster_centers, cluster_len)
```

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
8.0
Min inter-cluster distance =  1.0933194607372518
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
16.0
Min inter-cluster distance =  0.7123318236197774
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
```

```
22.0
Min inter-cluster distance =  0.5179286172497254
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
9.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
31.0
Min inter-cluster distance =  0.5064095487015859
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
38.0
Min inter-cluster distance =  0.36495419250817024
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
46.0
Min inter-cluster distance =  0.346654501371586
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
6.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
54.0
Min inter-cluster distance =  0.30468071844965394
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
62.0
Min inter-cluster distance =  0.29187627608454664
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2
1.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
69.0
Min inter-cluster distance =  0.18237562550345013
---
```

Inference:

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

In [43]:
```
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(coor
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_wi
```

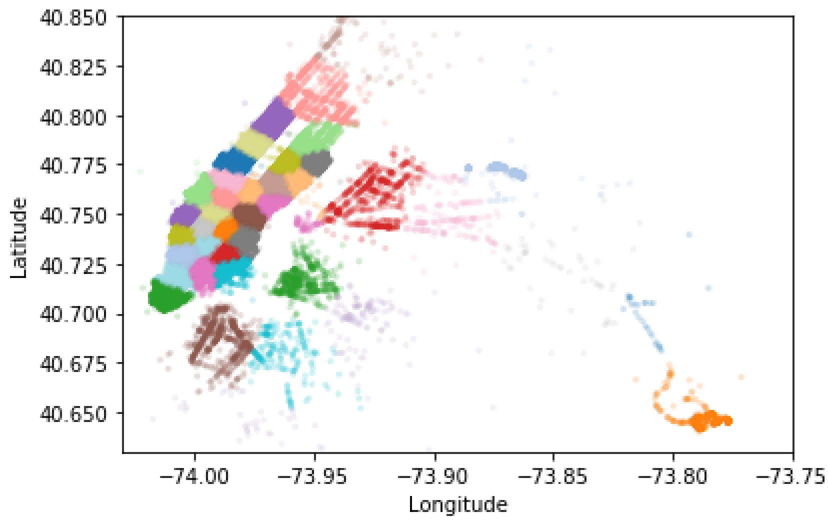
Plotting the cluster centers:

In [44]:
```
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
map_osm
```

Out[44]:

Plotting the clusters:

In [45]:
```
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.valu
               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 [46]:
```
def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,1433116
                    [1451606400,1454284800,1456790400,1459468800,1462060800,14647

    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 con
    tenminutewise_binned_unix_pickup_times=[(int((i-start_pickup_unix)/600)+33) fo
    frame['pickup_bins'] = np.array(tenminutewise_binned_unix_pickup_times)
    return frame
```

In [47]:
```
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_wi
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'
```

In [48]:
```
jan_2015_frame.head()
```

Out[48]:

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

In [49]:
```
jan_2015_groupby.head()
```

Out[49]:

		trip_distance
pickup_cluster	**pickup_bins**	
	57	104
	58	200
0	**59**	208
	60	141
	61	155

In [50]:

```
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(fram
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predic

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

    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 [51]:
```
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 [52]:
```
jan_2015_unique = return_unq_pickup_bins(jan_2015_frame)
jan_2016_unique = return_unq_pickup_bins(jan_2016_frame)
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [53]:
```
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",
    print('-'*60)
```

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

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

In [54]:

```python
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
```

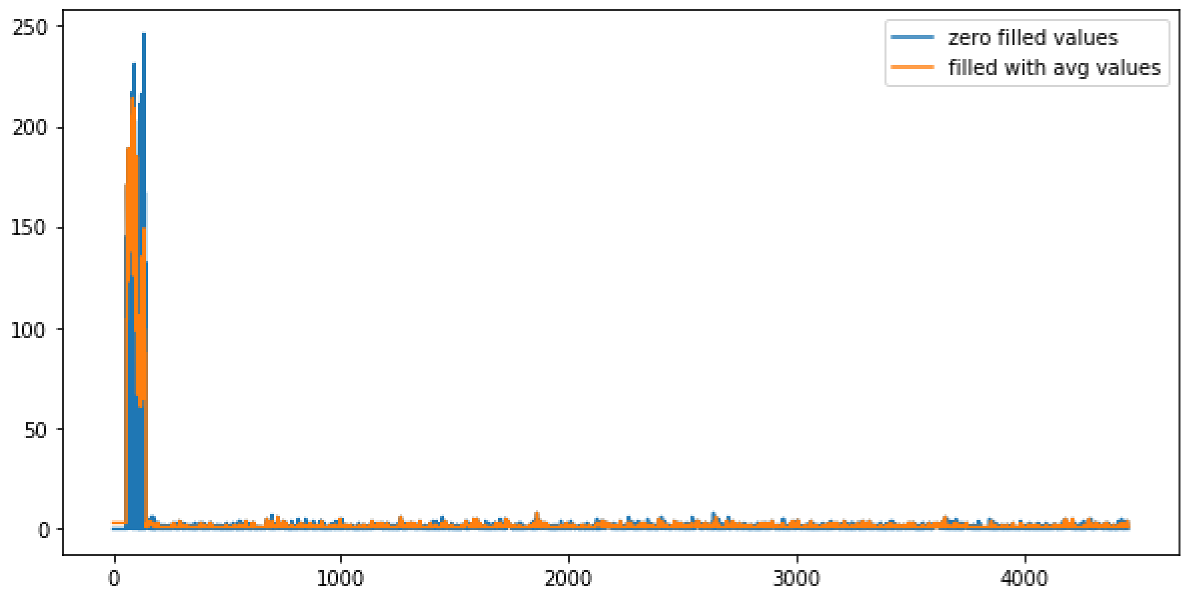
In [55]:
```
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[]
        repeat=0
        for i in range(4464):
            if repeat!=0:
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of th
            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 t
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                        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:
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.
                        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:
                    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)
                    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 [56]:
```
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015_un
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uni
```

In [57]:
```
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

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

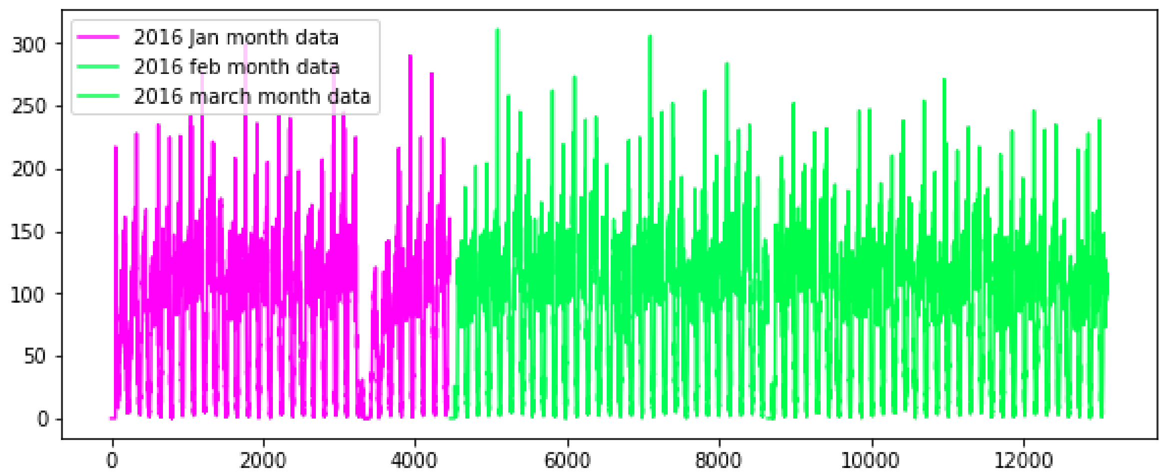
In [58]:
```
plt.figure(figsize=(10,5))
plt.plot(jan_2015_fill[4464:8920], label="zero filled values")
plt.plot(jan_2015_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```



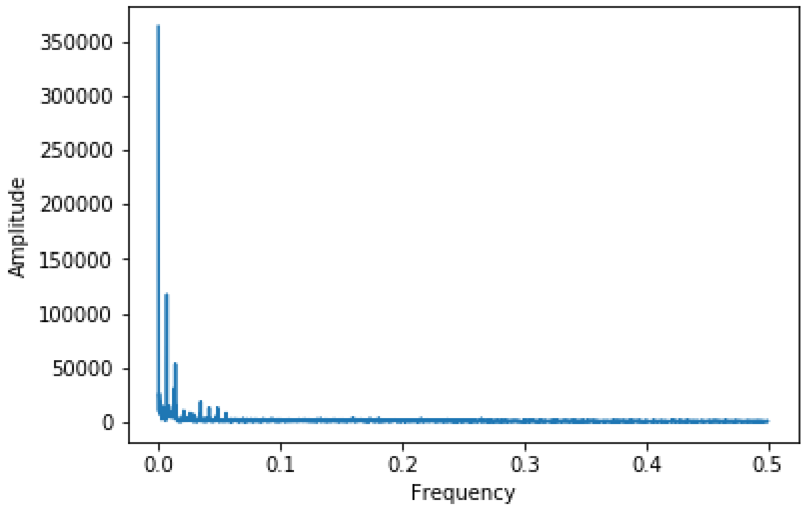
In [59]:
```
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uni
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_
mar_2016_smooth = fill_missing(mar_2016_groupby['trip_distance'].values,mar_2016_

regions_cum = []


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

Time series and Fourier Transforms

In [60]:
```
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
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), label='
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='2016 
    plt.legend()
    plt.show()
```



In [61]:
```
Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
freq = np.fft.fftfreq(4460, 1)
n = len(freq)
plt.figure()
plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



In [62]:
```
ratios_jan = pd.DataFrame()
ratios_jan['Given']=jan_2015_smooth
ratios_jan['Prediction']=jan_2016_smooth
ratios_jan['Ratios']=ratios_jan['Prediction']*1.0/ratios_jan['Given']*1.0
```

Modelling: Baseline Models

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

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

Simple Moving Averages

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

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

In [63]:

```
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_ra
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)
        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(rati
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

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

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

In [64]:
```
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)[
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_si
        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(rati
    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 [65]:

```
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_ra
        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(rati
    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 [66]:
```
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)[
        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(rati
    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

$$R'_t = \alpha * R_{t-1} + (1 - \alpha) * R'_{t-1}$$

In [67]:
```
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_ra
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].

    ratios['EA_R1_Predicted'] = predicted_values
    ratios['EA_R1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(rati
    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 [68]:
```
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)[
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Predic

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

```
In [69]: mean_err=[0]*6
median_err=[0]*6
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')
```

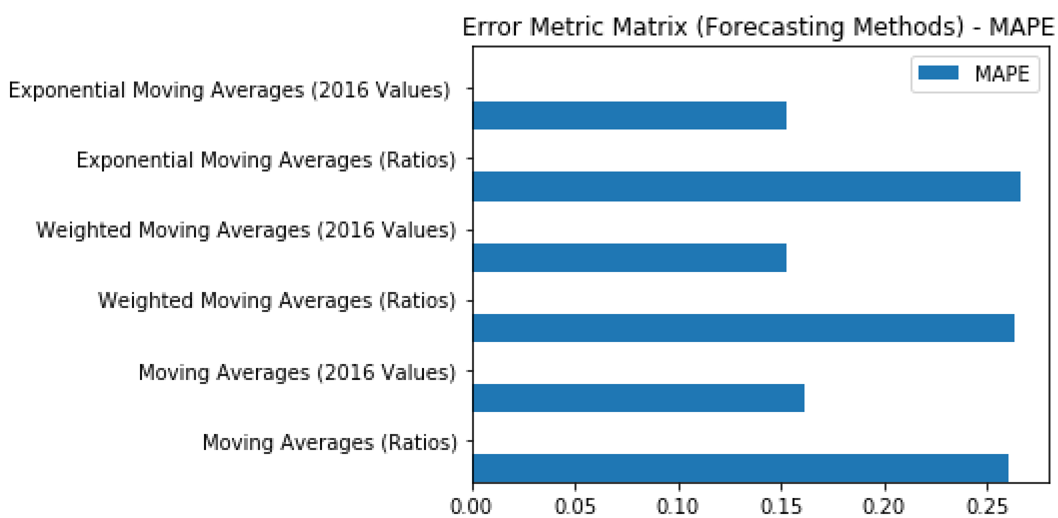
Comparison between baseline models

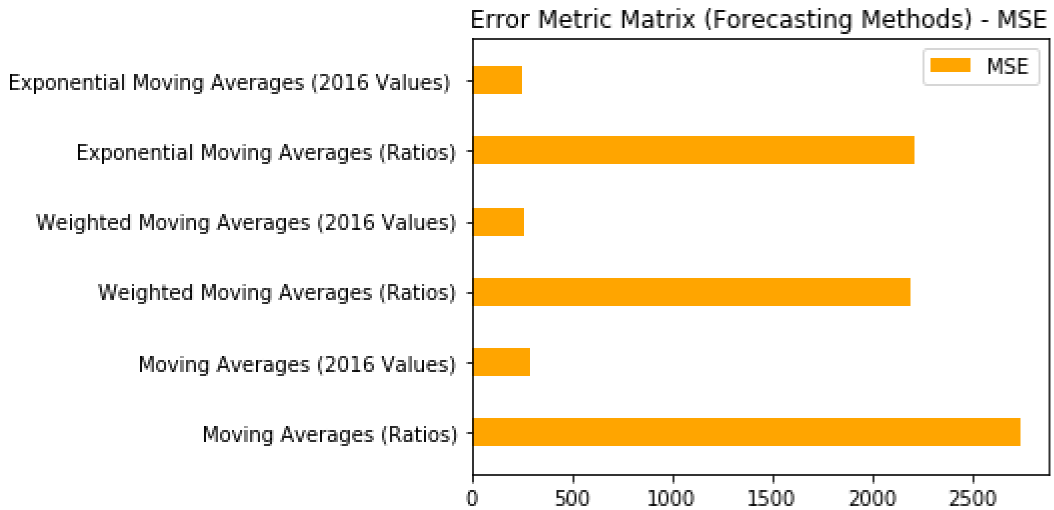
```
In [123]: print ("Error Metric Matrix (Forecasting Methods) - MAPE & MSE")
print ("--------------------------------------------------------------------------------
print ("Moving Averages (Ratios) -                             MAPE: ",mean_err[0
print ("Moving Averages (2016 Values) -                        MAPE: ",mean_err[1
print ("--------------------------------------------------------------------------------
print ("Weighted Moving Averages (Ratios) -                    MAPE: ",mean_err[2
print ("Weighted Moving Averages (2016 Values) -               MAPE: ",mean_err[3
print ("--------------------------------------------------------------------------------
print ("Exponential Moving Averages (Ratios) -              MAPE: ",mean_err[4],"
print ("Exponential Moving Averages (2016 Values) -         MAPE: ",mean_err[5],"
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------
-------------------------
Moving Averages (Ratios) -                             MAPE:  0.261123463188770
06       MSE:  2739.8888048835124
Moving Averages (2016 Values) -                        MAPE:  0.161200537860041
74        MSE:  298.25365143369174
--------------------------------------------------------------------------------
-------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.264056046453494
9       MSE:  2187.602872983871
Weighted Moving Averages (2016 Values) -               MAPE:  0.153206787922152
48       MSE:  260.315479390681
--------------------------------------------------------------------------------
-------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.266373733227229
MSE:  2213.81395609319
Exponential Moving Averages (2016 Values) -         MAPE:  0.15265037109743781
MSE:  257.1782762096774
```

In [131]:

```
df = pd.DataFrame(dict(graph=['Moving Averages (Ratios)', 'Moving Averages (2016 \
                              'Weighted Moving Averages (Ratios)','Weighted Mo
                              'Exponential Moving Averages (Ratios)','Exponent
                       n=mean_err, m=median_err))

ind = np.arange(len(df))
width = 0.4

fig, ax = plt.subplots()
ax.barh(ind, df.n, width, label='MAPE')
#ax.barh(ind + width, df.m, width, label='MSE')
fig.set_figwidth(8)
plt.gcf().subplots_adjust(left = 0.40)
plt.title("Error Metric Matrix (Forecasting Methods) - MAPE")
ax.set(yticks=ind + width, yticklabels=df.graph, ylim=[2*width - 1, len(df)])
ax.legend()

plt.show()

ind = np.arange(len(df))
width = 0.4

fig, ax = plt.subplots()
#ax.barh(ind, df.n, width, label='MAPE')
ax.barh(ind + width, df.m, width, color="orange", label='MSE')
fig.set_figwidth(8)
plt.gcf().subplots_adjust(left = 0.40)
plt.title("Error Metric Matrix (Forecasting Methods) - MSE")
ax.set(yticks=ind + width, yticklabels=df.graph, ylim=[2*width - 1, len(df)])
ax.legend()

plt.show()
```





Regression Models

In [75]:

```
number_of_time_stamps = 5
output = []
tsne_lat = []
tsne_lon = []
tsne_weekday = []
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)
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+446
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_st
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [76]:

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

Out[76]:

```
True
```

In [77]:

```
alpha=0.3
predicted_values=[]
predict_list = []
tsne_flat_exp_avg = []
fr_am_final = pd.DataFrame(columns= ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a
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)[:
    fr_am_feb_sorted = fr_am_feb.sort_values(by=["Amplitude"], ascending=False)[:
    fr_am_mar_sorted = fr_am_mar.sort_values(by=["Amplitude"], ascending=False)[:

    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
    fr_am_new_feb.columns = ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_5
    fr_am_new_mar.columns = ['f_1','a_1','f_2','a_2','f_3','a_3','f_4','a_4','f_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)


    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][
    predict_list.append(predicted_values[5:])
    predicted_values=[]
fr_am_final.drop(['f_1'],axis=1,inplace=True)

fr_am_final = fr_am_final # (fr_am_final - fr_am_final.mean()) / (fr_am_final.max
fr_am_final = fr_am_final.fillna(0)
```

In [78]:
```
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 [79]:
```
train_features =  [tsne_feature[i*13099:(13099*i+9169)] for i in range(0,40)]
test_features = [tsne_feature[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
fr_am_final_train = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3','f_4','a_
fr_am_final_test = pd.DataFrame(columns=['a_1','f_2','a_2','f_3','a_3','f_4','a_4
for i in range(0,40):
    fr_am_final_train = fr_am_final_train.append(fr_am_final[i*13099:(13099*i+916
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
fr_am_final_test.reset_index(inplace=True)
```

In [80]:
```
print("Number of data clusters",len(train_features), "Number of data points in tr
print("Number of data clusters",len(train_features), "Number of data points in te
```

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

In [81]:
```
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 [82]:
```
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 [83]:
```
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 [84]:
```
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 [85]:
```
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 [86]:
```
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 [87]:
```
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 [89]:
```
df_test.head()
```

Out[89]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg
0	84	77	89	117	111	40.776228	-73.982119	4	109
1	77	89	117	111	135	40.776228	-73.982119	4	127
2	89	117	111	135	128	40.776228	-73.982119	4	127
3	117	111	135	128	112	40.776228	-73.982119	4	116
4	111	135	128	112	130	40.776228	-73.982119	4	125

In [90]:
```
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)

df_test_lm.head()
print(df_test.shape)
print(fr_am_final_test.shape)
```

```
(157200, 9)
(157200, 10)
```

In [92]:
```
df_test_lm.head()
```

Out[92]:

	ft_5	ft_4	ft_3	ft_2	ft_1	lat	lon	weekday	exp_avg	index	a_1	f_2
0	84	77	89	117	111	40.776228	-73.982119	4	109	9169	385853.0	-0.006944
1	77	89	117	111	135	40.776228	-73.982119	4	127	9170	385853.0	-0.006944
2	89	117	111	135	128	40.776228	-73.982119	4	127	9171	385853.0	-0.006944
3	117	111	135	128	112	40.776228	-73.982119	4	116	9172	385853.0	-0.006944
4	111	135	128	112	130	40.776228	-73.982119	4	125	9173	385853.0	-0.006944

In [93]:
```
# specify parameters and distributions to sample from
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

Using Linear Regression

In [96]:
```
from sklearn.linear_model import LinearRegression
from sklearn.grid_search import GridSearchCV

lr_reg=LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[T
grid = GridSearchCV(lr_reg,parameters, cv=None)
grid.fit(df_train, tsne_train_output)

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
{'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

In [97]:
```
lr_reg=LinearRegression(copy_X=True, fit_intercept=True, normalize=False).fit(df_

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

In [98]:
```
lr_reg_lm=LinearRegression()

parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[T
grid = GridSearchCV(lr_reg,parameters, cv=None)
grid.fit(df_train, tsne_train_output)

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
{'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

In [99]:
```
lr_reg_lm=LinearRegression(copy_X=True, fit_intercept=True, normalize=False).fit(

y_pred_lm = lr_reg_lm.predict(df_test_lm)
lr_test_predictions_lm = [round(value) for value in y_pred_lm]
y_pred_lm = lr_reg_lm.predict(df_train_lm)
lr_train_predictions_lm = [round(value) for value in y_pred_lm]
```

Using Random Forest Regressor

In [103]:

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


regr1 = RandomForestRegressor()#max_features='sqrt',min_samples_leaf=4,min_sample
```

In [104]:

```
param_dist = {"max_depth": [3, None],
              "max_features": ['sqrt' , 'log2' ],
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "n_estimators":[35,40,45]
              }

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(regr1, param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time()
random_search.fit(df_train, tsne_train_output)

print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)
```

```
RandomizedSearchCV took 1297.14 seconds for 20 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.942 (std: 0.014)
Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 10,
'min_samples_split': 8, 'n_estimators': 40}

Model with rank: 2
Mean validation score: 0.942 (std: 0.014)
Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 9,
'min_samples_split': 2, 'n_estimators': 40}

Model with rank: 3
Mean validation score: 0.942 (std: 0.014)
Parameters: {'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 7,
'min_samples_split': 4, 'n_estimators': 40}
```

In [108]:

```
regr1 = RandomForestRegressor(max_features='sqrt',min_samples_leaf=10,min_samples
regr1.fit(df_train, tsne_train_output)

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

In [109]:
```
print (df_train.columns)
print (regr1.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'lat', 'lon', 'weekday',
       'exp_avg'],
      dtype='object')
[0.03619284 0.06917031 0.0901411  0.18281095 0.30168064 0.0016787
 0.00224943 0.00098928 0.31508674]
```

Using XgBoost Regressor

In [110]:
```
x_model = xgb.XGBRegressor()
param_dist = {"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]
              }

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(x_model, param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time()
random_search.fit(df_train, tsne_train_output)

print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)
```

```
RandomizedSearchCV took 613.66 seconds for 20 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.943 (std: 0.014)
Parameters: {'nthread': 5, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 0.1,
'colsample_bytree': 0.7}

Model with rank: 2
Mean validation score: 0.943 (std: 0.014)
Parameters: {'nthread': 5, 'min_child_weight': 4, 'max_depth': 4, 'gamma': 0.2,
'colsample_bytree': 0.9}

Model with rank: 3
Mean validation score: 0.943 (std: 0.014)
Parameters: {'nthread': 3, 'min_child_weight': 6, 'max_depth': 5, 'gamma': 0,
'colsample_bytree': 0.9}
```

In [111]:
```
x_model = xgb.XGBRegressor(
 learning_rate =0.1,
 n_estimators=1000,
 max_depth=5,
 min_child_weight=5,
 gamma=0.1,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=0.7,nthread=5)
x_model.fit(df_train, tsne_train_output)

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

In [87]:
```
#x_model.booster().get_score(importance_type='weight')
```

Calculating the error metric values for various models

In [132]:
```
train_mape=[]
test_mape=[]


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

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

Error Metric Matrix

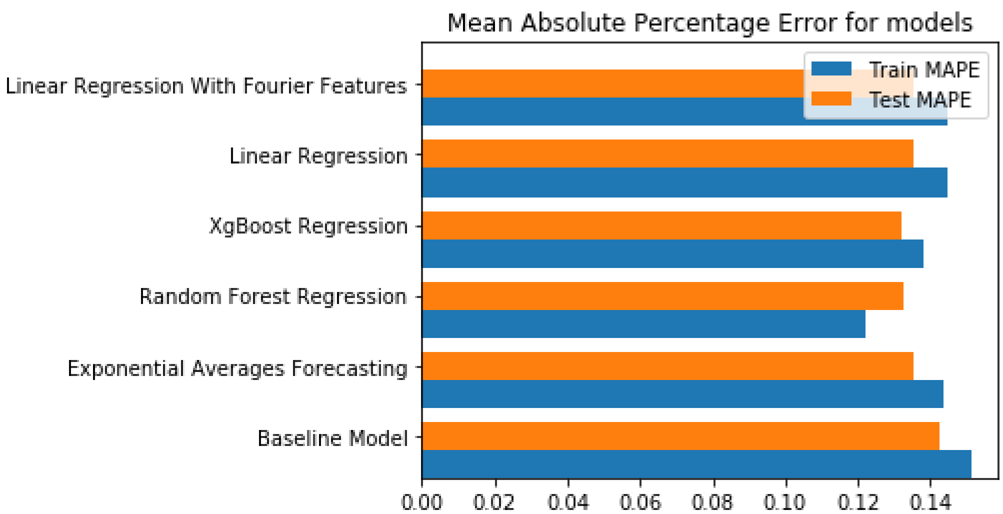
In [133]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------
print ("Baseline Model -                             Train: ",train_mape[0],"
print ("Exponential Averages Forecasting -           Train: ",train_mape[1],"
print ("Linear Regression -                          Train: ",train_mape[4],"
print ("Linear Regression With Fourier Features -    Train: ",train_mape[5],"
print ("Random Forest Regression -                   Train: ",train_mape[2],"
print ("XgBoost Regression -                         Train: ",train_mape[3],"
print ("--------------------------------------------------------------------------------
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------
-------------------------
Baseline Model -                             Train:  0.15108785776083566
Test:  0.14275551690979008
Exponential Averages Forecasting -           Train:  0.143438332208147       Te
st:  0.13521244148947784
Linear Regression -                          Train:  0.14469271604365572
Test:  0.1353982132872875
Linear Regression With Fourier Features -    Train:  0.14468596042196513
Test:  0.13527404899932888
Random Forest Regression -                   Train:  0.12238245556039458      T
est:  0.13255164256632776
XgBoost Regression -                         Train:  0.13822925247244494
Test:  0.13221009584761587
--------------------------------------------------------------------------------
-------------------------
```

Observation

In [135]:

```
df = pd.DataFrame(dict(graph=['Baseline Model', 'Exponential Averages Forecasting
                              n=train_mape, m=test_mape))

ind = np.arange(len(df))
width = 0.4

fig, ax = plt.subplots()
ax.barh(ind, df.n, width, label='Train MAPE')
ax.barh(ind + width, df.m, width, label='Test MAPE')
fig.set_figwidth(8)
plt.gcf().subplots_adjust(left = 0.40)
plt.title("Mean Absolute Percentage Error for models")
ax.set(yticks=ind + width, yticklabels=df.graph, ylim=[2*width - 1, len(df)])
ax.legend()

plt.show()
```



By comparing all the model by MAPE we can conclude that, even though all the model has MAPE between 13% - 14.5%, XgBoost has the lowest MAPE for test data is 13.22%.