Taxi Demand Prediction in New York City

Chinmay Sathe

19MT0119

—

NEURAL NETWORKS & DEEP LEARNING LAB
(MCC542)

—

Prof. Subhashis Chatterjee

Abstract

Generally, a taxi cab roams around New York city and gets its customers randomly. But, what if, an efficient system was made, such that, cabs need not go more than 0.5 miles and get a customer with a drop location, desirable to the cab driver.

Objective here is to develop a Machine Learning algorithm that predicts number of pickups as accurately as possible for each region in a 10-minute interval.

1 INTRODUCTION

A taxi driver, is our end user here. Now, based on machine learning models for Clustering and Regression with the idea of Time Series Analysis, we need to first divide the whole of New York city into some clusters and based on features taken from Jan-2015 Yellow Cab Data, we draw a conclusion on how many pickups will actually be raised upon in future. This helps both the taxi driver; they get a customer within reach every time and at the end of the day their profit is defined to increase, and the customer; they get a cab as soon as possible.

Why the city, New York?

- Because of its grid like geographical structure.
- Because in general to travel 1 mile in New York it takes nearly 10 min

Any latency constraints?

- The model should not take more than 5 minutes to give output. Although, a time between 1-2 minutes is bearable.

The workflow is as follows:

1. First step is data pre-processing on various features like latitude, longitude, trip duration, speed, distance etc.
2. Second step is to cluster regions by using KMeans algorithm.
3. In the third step, we calculated the Fourier features and merged them with the previous five features.
4. At last we applied Random Forest Regression to predict the number of pickup calls available.

Models tried are:

- K Means
- Random Forest Regressor

Performance Metrics used are:

- Mean Absolute Percentage Error
- Mean Squared Error

Suppose, if the cab driver knows that our model predicts with 10% error chances, still he can manage on his own, assuming the no. pickups according to the results obtained.

In one line the objective can be stated as:
“Given time and location, we want a taxi driver to know the number of pickups within a 10 min region.”

Since we have to predict at what time how many pickups are available, this is **Time-Series Regression type of problem.**

2 DATA AND FEATURES

2.1 Data

The data used were collected and provided to the NYC Taxi and Limousine Commission (TLC). NYC Taxi and Limousine commission regulates all the taxis in New York, we here are focusing only on Yellow cabs. NYC TLC collects and broadcast the data of each taxi they operate, on a monthly basis. Thus, on their official website data of each month since year 2009.

Since the data is very large for any general home computer's RAM to handle, we will use dask instead of pandas. Dask breaks the data into blocks and then takes the data in RAM one block at a time.

We are focusing only on datasets from Jan-2015 and Jan-2016.

The data set can be obtained from:

https://www.kaggle.com/raisulalam/nycyellowtaxi

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.

2.2 Features

In total there are 19 features, although we will later use tSNE to reduce some feature related work load, we shall define all the features here:

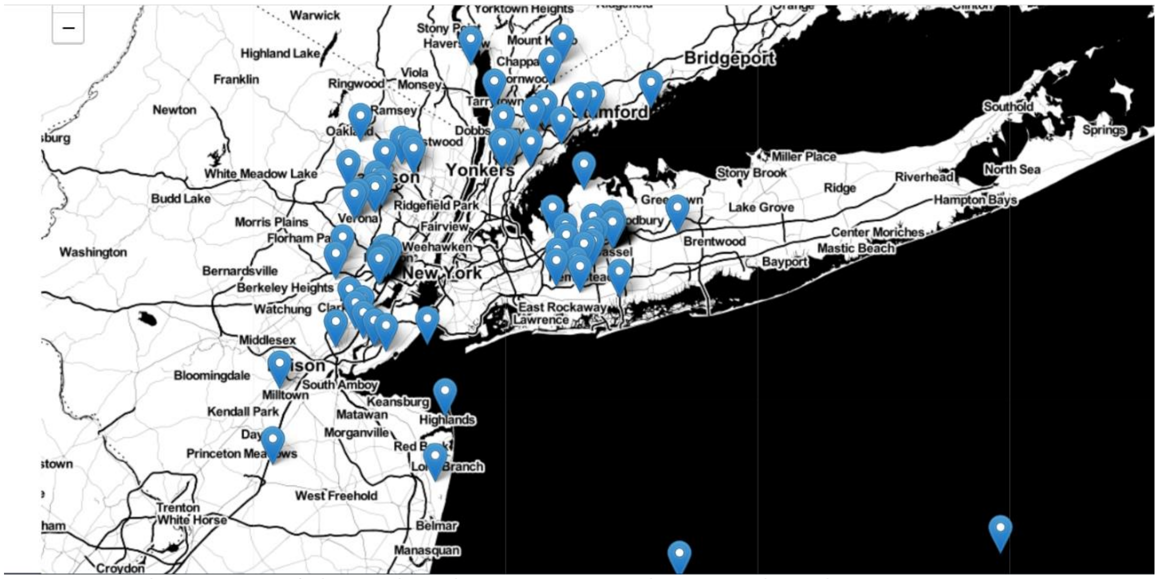
1. VendorID - A code indicating the TPEP provider that provided the record.
1.Creative Mobile Technologies & 2. VeriFone Inc.

2. tpep_pickup_datetime - The date and time when the meter was engaged.
3. tpep_dropoff_datetime - The date and time when the meter was disengaged.
4. passenger_count - The number of passengers in the vehicle. This is a driver-entered value.
5. trip_distance - The elapsed trip distance in miles reported by the taximeter.
6. pickup_longitude - Longitude where the meter was engaged.
7. pickup_latitude - Latitude where the meter was engaged.
8. 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
9. 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
10. dropoff_longitude - Longitude where the meter was disengaged.
11. dropoff_latitude - Latitude where the meter was disengaged.
12. 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
13. fare_amount - The time-and-distance fare calculated by the meter.
14. extra - Miscellaneous extras and surcharges. Currently, this only includes. the $0.50 and $1 rush hour and overnight charges.
15. mta_tax - 0.50 MTA tax that is automatically triggered based on the metered rate in use.
16. tip_amount - This field is automatically populated for credit card tips. Cash tips are not included.
17. tolls_amount - Total amount of all tolls paid in trip.
18. improvement_surcharge - 0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge began being levied in 2015.
19. total_amount - The total amount charged to passengers. Does not include cash tips.

3 PRE-PROCESSING & EDA

3.1 Pickup Latitude and Pickup Longitude

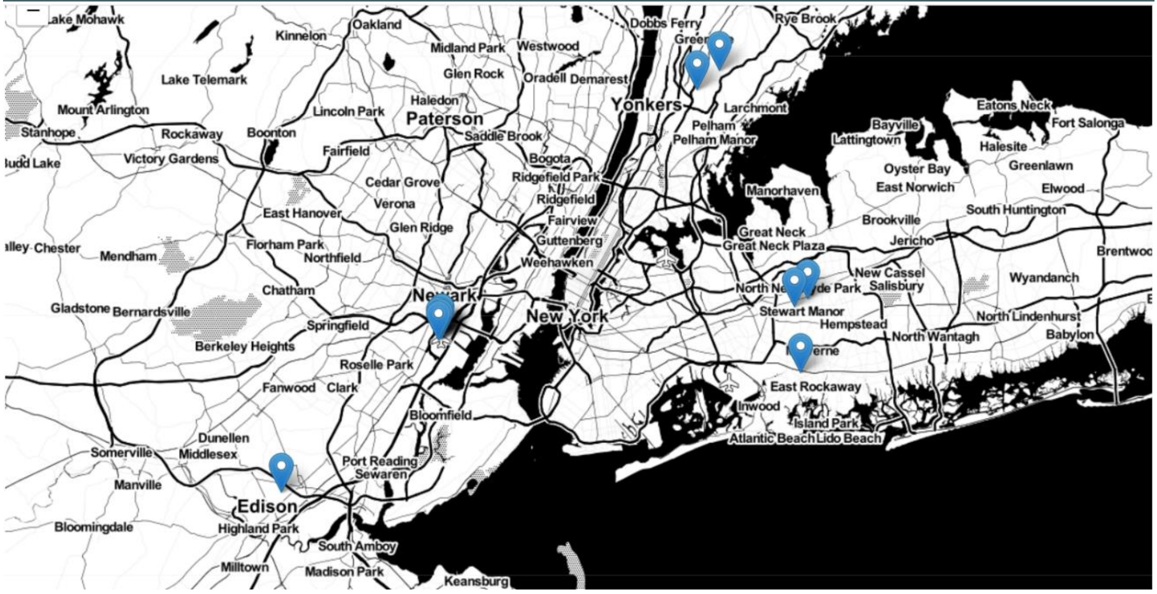
New York is bounded by the location coordinates (Lat, long) - (40.5774, -74.15) & (40.9176, -73.7004), hence, any coordinates not within these ranges are not considered by us. We are only concerned with the pickups which originate within the New York.



Map 1: It shows one of the pickup locations is in the sea. Also, there are some points just outside the boundary. Some are even in South America, Mexico & Canada.

3.2 Dropoff Latitude and Pickup Longitude

New York is bounded by the location coordinates (Lat, long) - (40.5774, -74.15) & (40.9176, -73.7004), hence, any coordinates not within these ranges are not considered by us. We are only concerned with the pickups which originate within the New York.


Map 2: The observations here are similar to those obtained while analysing pickup latitude and longitude.

3.3 Trip Durations

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

Also,
Trip Duration Time = Pickup Duration Time – Dropoff Duration Time

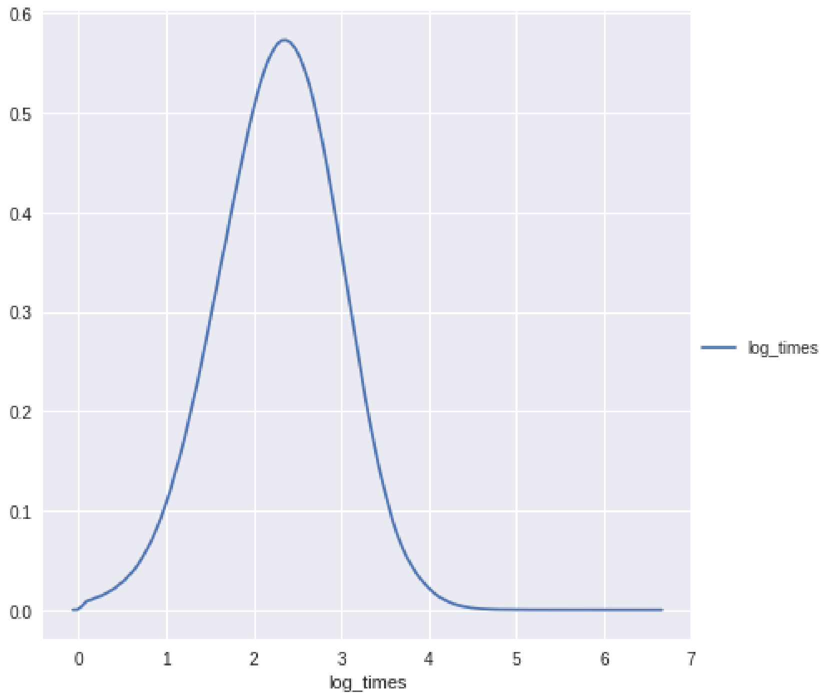


Figure 1: As observed by the graph the trip duration time follows log distribution

3.4 Speed

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

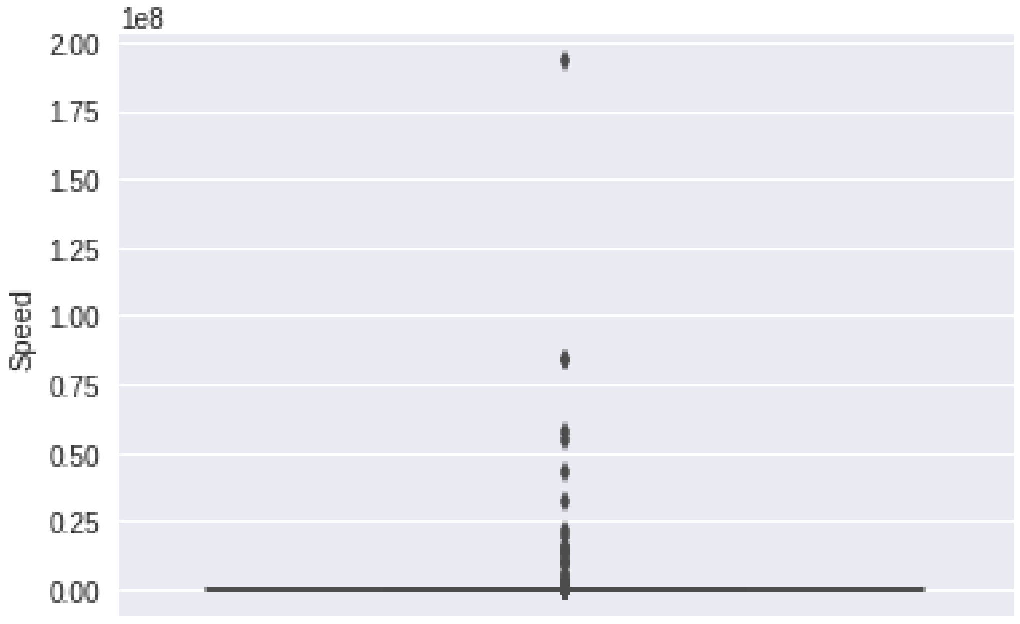


Figure 2: Box-plot of the feature Speed before outlier removal

3.5 Trip Distance

Simple Outlier removal using box plot and Quantiles

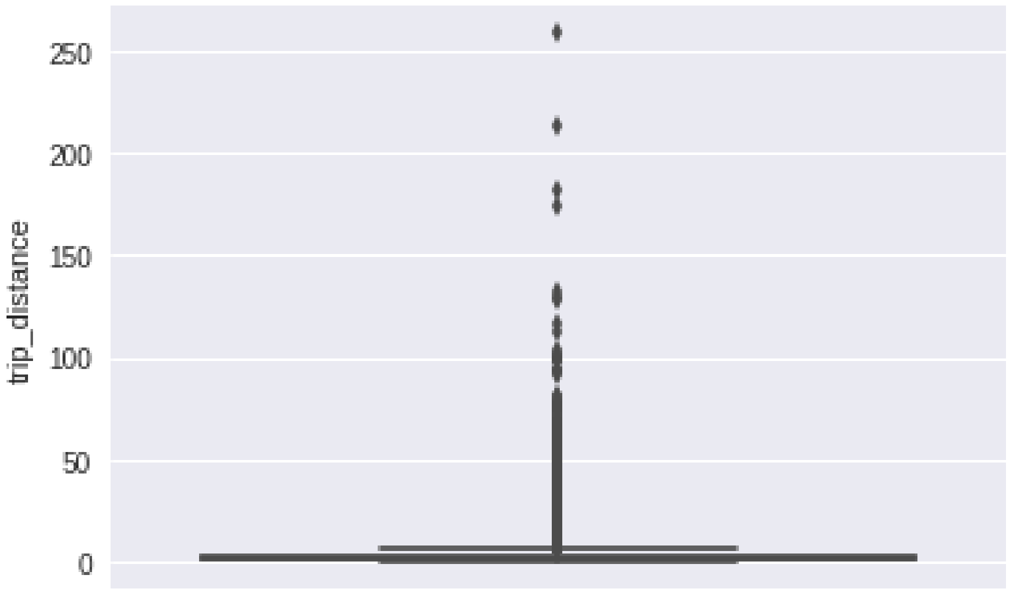


Figure 3: Box-plot of the feature Trip Distance before outlier removal

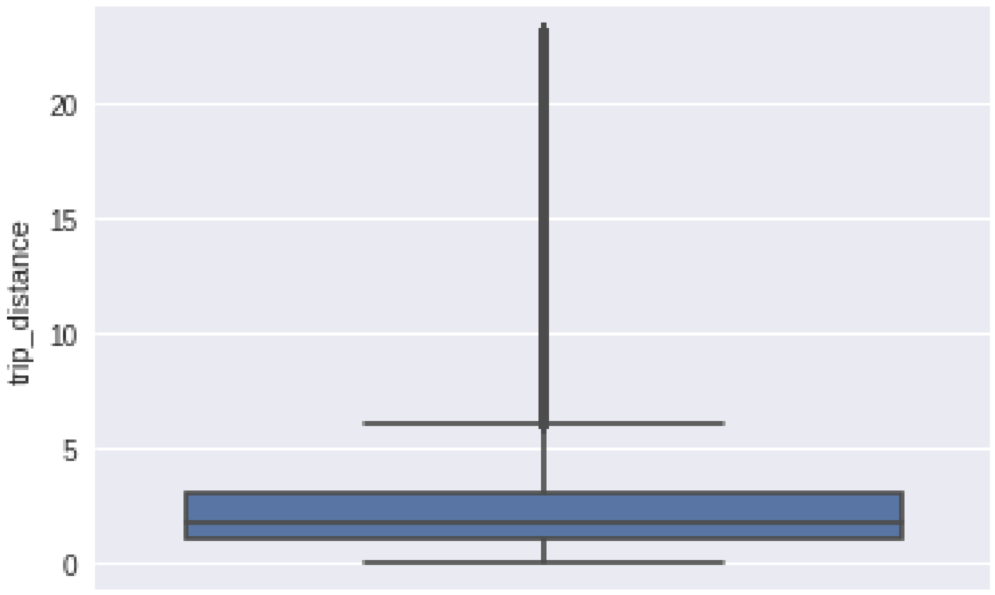


Figure 4: Box-plot of the feature Trip Distance after outlier removal

3.6 Trip Amount

We have observed that 99.9 percentile is 86.6 so we keep our fare amount limited to the value at 99.9 percentile.

3.7 Passenger Count and other outliers

In real world, passenger count in New York cannot be more than 6.
Total outliers removed from Jan-2015 dataset: 828606
Total outliers removed from Jan-2016 dataset: 664287

4 DATA PREPARATION

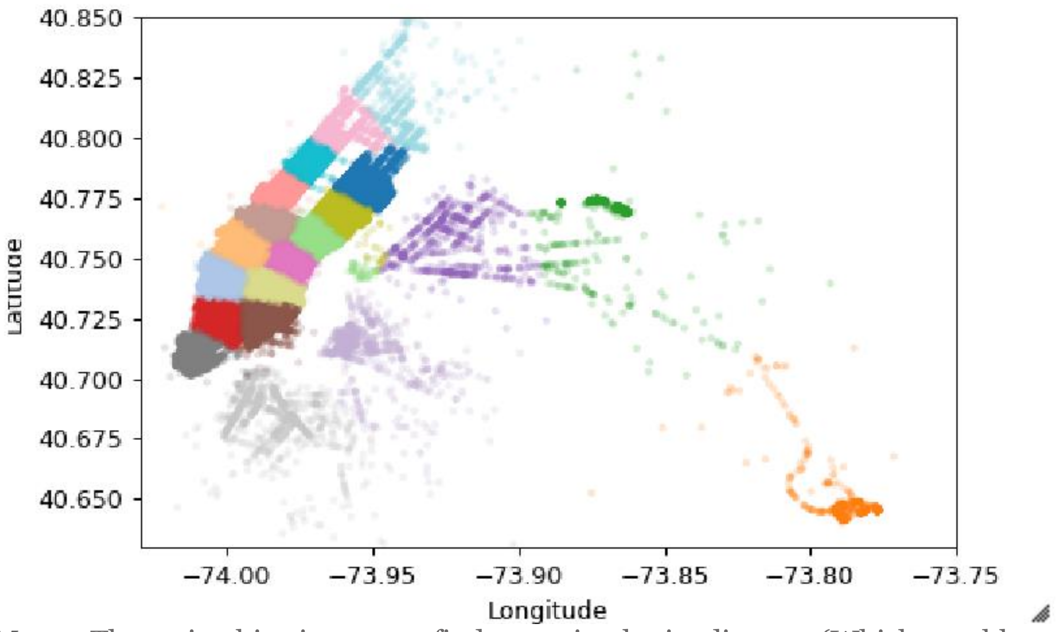
The whole process is divided into 3 process:
1. Segmentation of New York City using Clusters.
2. Time Binning Clusters into 10 min intervals.
3. Smoothing of data.

4.1 Segmentation

Using k-means algorithms for features like latitude and longitude we can divide New York city into segments. We use the fact that K means roughly creates clusters of same sizes.

On choosing a cluster size of 40,
Avg. Number of Clusters within the vicinity (i.e. inter-cluster-distance < 2): 10.0
Avg. Number of Clusters outside the vicinity (i.e. inter-cluster-distance > 2): 30.0
Min inter-cluster distance = 0.44453353183475264

Now, since, we can cover about 2 miles in 10 min, we want our inter cluster region to be less than < 2 miles, such that next cluster can be reached within 10 min. But for clusters to not be too small, we set lower bound for inter cluster distance to be 0.5 mile.



Map 3: The main objective was to find an optimal min. distance (Which roughly estimates to the radius of a cluster) between the clusters, which we got was 20.

4.2 Time Binning & Smoothing

In time binning, we simply make bins of 10 minutes each. And, in smoothing, for every 10-min interval(pickup_bin) we will check if it is there in our unique bin, if present we will add the count_values[index] to smoothed data and if not, we add smoothed data.

Number of 10 min. intervals among all the clusters: 89280 (for Jan 2015 Dataset)

Q. Why did we choose, these methods and which method is used for which data?

A. Consider we have a data of some month in 2015, let it be Jan, then assume there are:
10 pickups that are happened in 1st 10-min travel, 0 pickups in 2nd 10-min travel, 0 pickups in 3rd 10-min travel, and 20 pickups in 4th 10-min travel.

In fill_missing method we replace these values with 10, 0, 0, 20 where as in smoothing method we replace these values as 6,6,6,6,6. You can check the number of pickups that happened in the first 40-min are same in both cases. And, if you will observe that we looking at the future values when we are using smoothing, i.e. we are looking at the future number of pickups which might cause a data leakage.

So, we use smoothing for Jan 2015 data since it acts as our training data and we use simple fill_misssing method for Jan 2016 data.

5 MODEL

5.1 Fourier transform

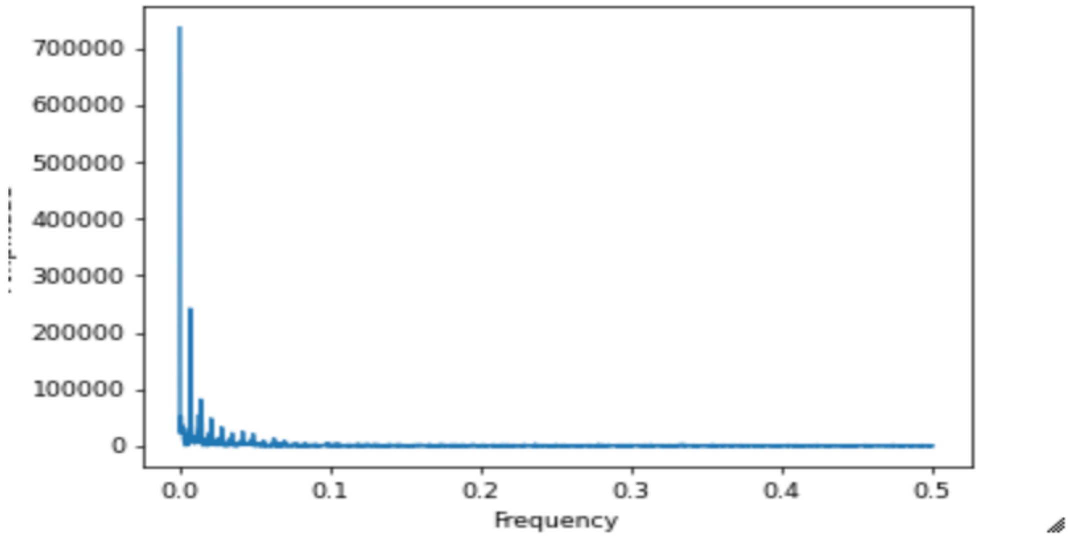


Figure 5: Amplitude/frequency plot the first peak at index 0, is the DC component. DC component just means the average of positive and negative half cycles is not zero. D is the DC component. It shifts the function up or down the y-axis. Note that it is independent of the function variable t. We will not consider its amplitude and frequency. We will start taking frequency and amplitudes from the second peak onwards.

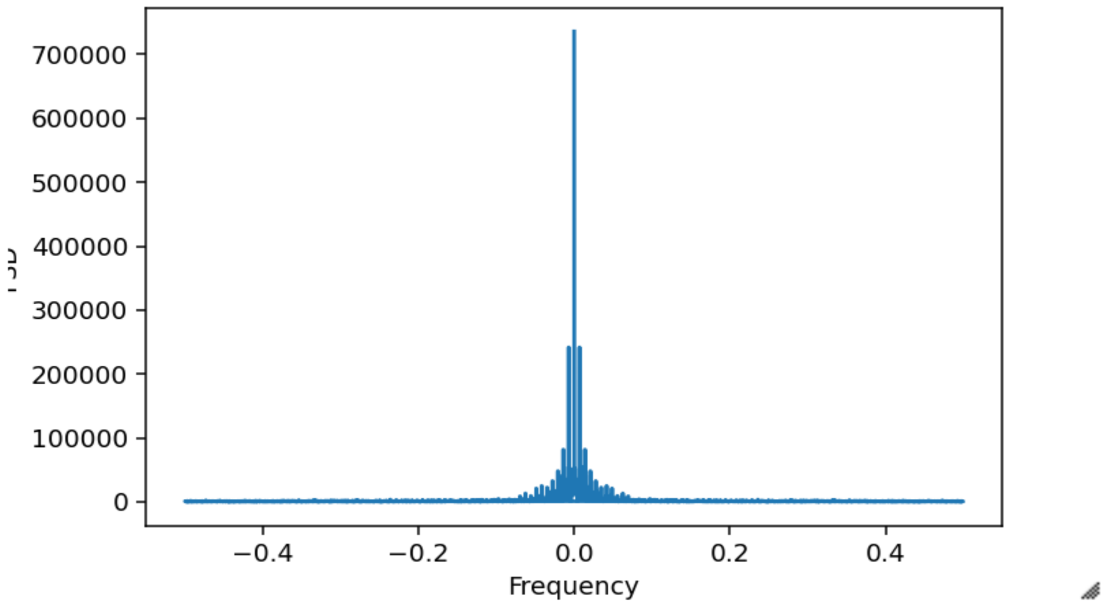


Figure 6: A [0] contains the zero-frequency term (the sum of the signal), which is always purely real for real inputs. A [1: n/2] contains the positive-frequency terms A[n/2+1:] contains the negative-frequency terms.

5.2 Random Forest Regressor

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging.

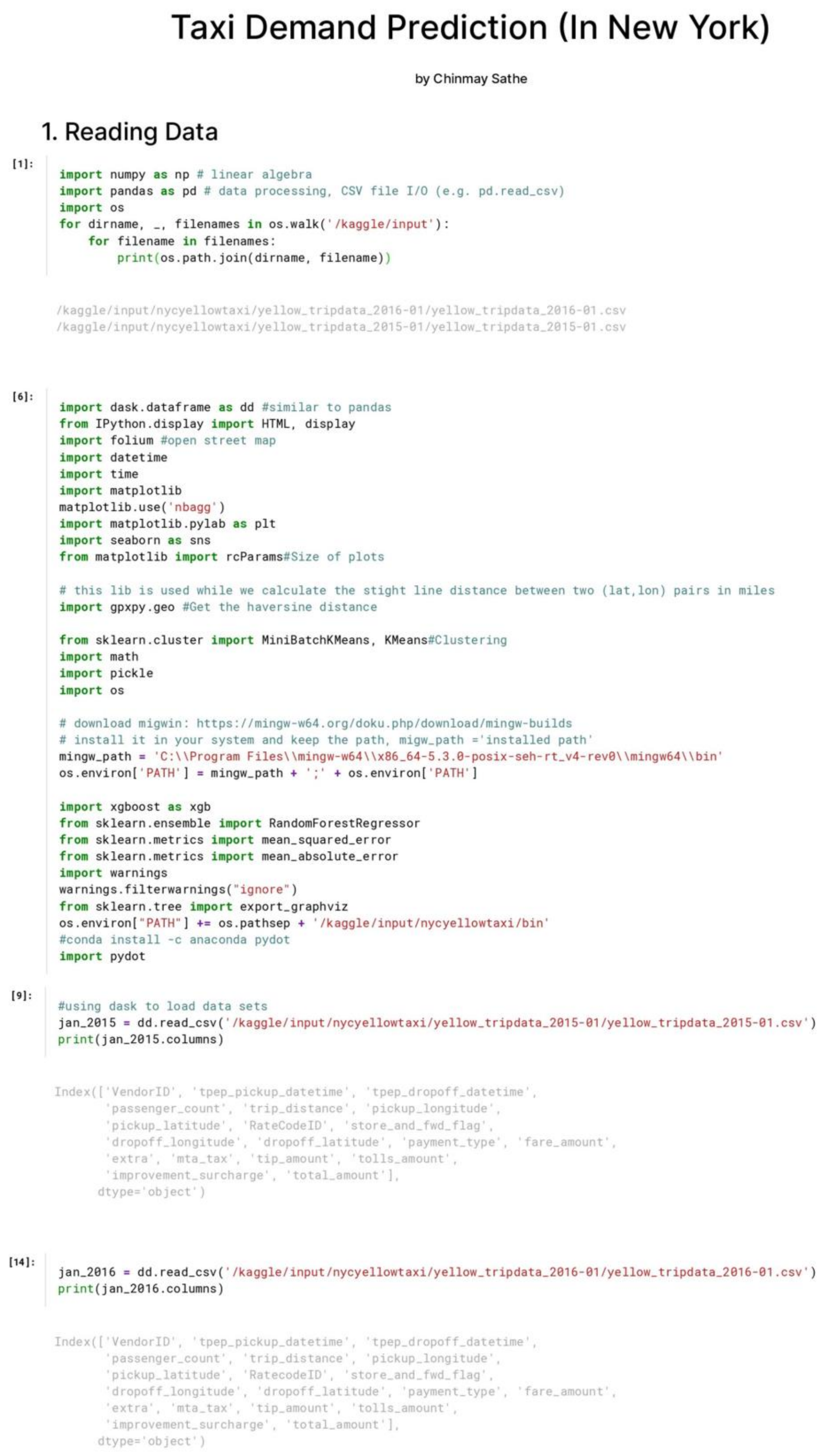
Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

6 RESULTS AND CONCLUSIONS

We conclude the following points:

- Random forest regression is performing well with train MAPE 9.31% and with test MAPE 12.22.
- The idea of dividing geographical area into cluster works, but is too time taking for a personal computer.
- Other models like, XGBoost can be tried for the same regression problems since it is also an ensemble technique-based model.
- The Time Series based transformations can be improved by having better understanding of time-series based transformations.

7 CODE



Taxi Demand Prediction (In New York)

by Chinmay Sathe

1. Reading Data

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/nycyellowtaxi/yellow_tripdata_2016-01/yellow_tripdata_2016-01.csv
/kaggle/input/nycyellowtaxi/yellow_tripdata_2015-01/yellow_tripdata_2015-01.csv
```

[6]:
```
import dask.dataframe as dd #similar to pandas
from IPython.display import HTML, display
import folium #open street map
import datetime
import time
import matplotlib
matplotlib.use('nbagg')
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams#Size of plots

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

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

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

import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
from sklearn.tree import export_graphviz
os.environ["PATH"] += os.pathsep + '/kaggle/input/nycyellowtaxi/bin'
#conda install -c anaconda pydot
import pydot
```

[9]:
```
#using dask to load data sets
jan_2015 = dd.read_csv('/kaggle/input/nycyellowtaxi/yellow_tripdata_2015-01/yellow_tripdata_2015-01.csv')
print(jan_2015.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')
```

[14]:
```
jan_2016 = dd.read_csv('/kaggle/input/nycyellowtaxi/yellow_tripdata_2016-01/yellow_tripdata_2016-01.csv')
print(jan_2016.columns)
```

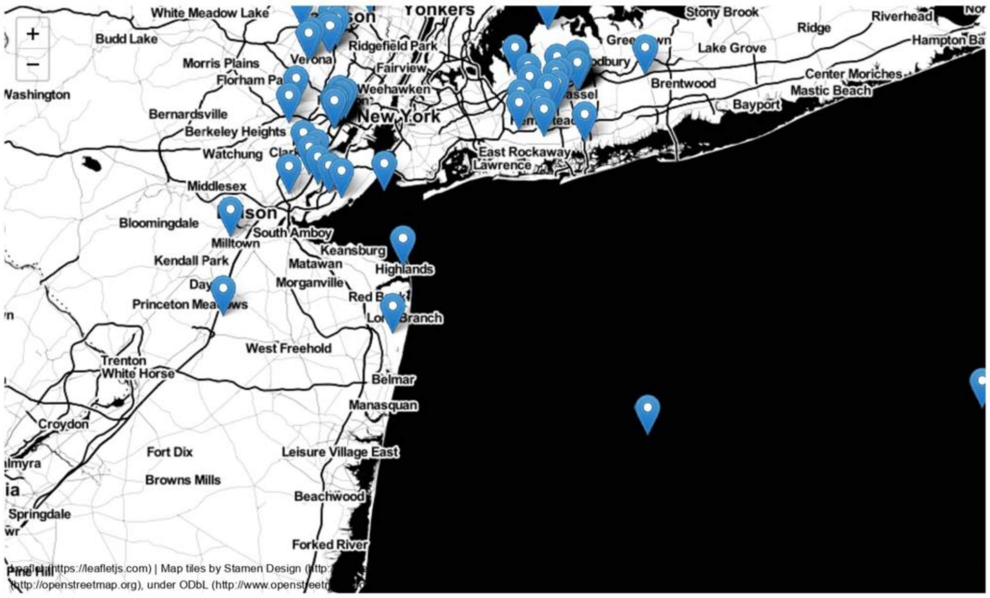
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax',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

2. Exploratory Data Analaysis

2.1. Pickup Latitude & Longitude

```
[15]:
# all the points outside newyork city to pickup_outliers
pickup_outliers = jan_2015[((jan_2015.pickup_longitude <= -74.15) | (jan_2015.pickup_latitude <= 40.5774)| \
                   (jan_2015.pickup_longitude >= -73.7004) | (jan_2015.pickup_latitude >= 40.9176))]

# creating a map with the a base location
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')

# plot first 10000 outliers on map
sample_1 = pickup_outliers.head(10000)

for i,j in sample_1.iterrows():
    if int(j['pickup_latitude']) != 0:
        folium.Marker(list((j['pickup_latitude'],j['pickup_longitude']))).add_to(map_osm)

map_osm
```

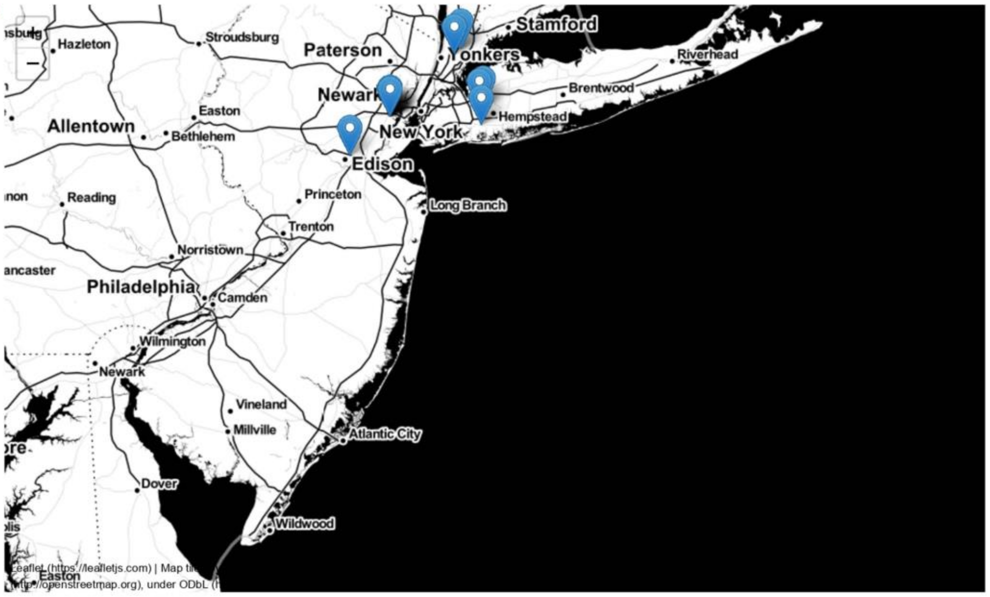
Out[15]:



2.2. Dropoff Latitude & Longitude

[16]:
```python
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_locations
outlier_locations = jan_2015[((jan_2015.dropoff_longitude <= -74.15) | (jan_2015.dropoff_latitude <= 40.5774)| \
                   (jan_2015.dropoff_longitude >= -73.7004) | (jan_2015.dropoff_latitude >= 40.9176))]

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

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

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

# we will spot only first 100 outliers on the map, plotting all the outliers will take more time
sample_locations = outlier_locations.head(100)
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[16]:



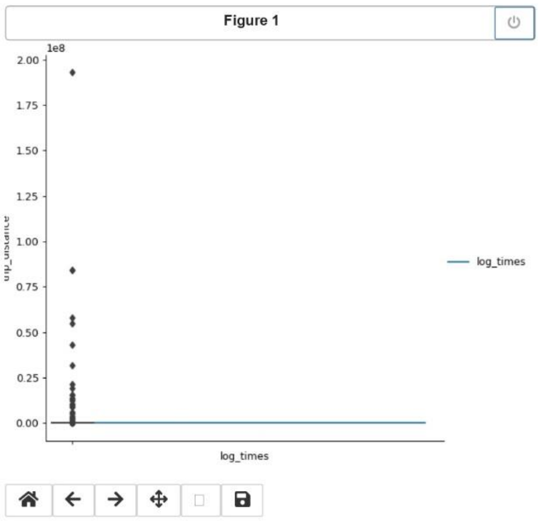
2.3. Trip Duration

[17]:
```python
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetuple())
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'].values]
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'].values]
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)
    new_frame = month[['passenger_count','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_
    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(jan_2015)
```

[18]:

```
#looking further from the 99th percecntile
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))

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

#converting the values to log-values to chec for log-normal
import math
updated_duration_of_trip['log_times']=[math.log(i) for i in updated_duration_of_trip['trip_times'].values]
#pdf of log-values
sns.FacetGrid(updated_duration_of_trip,size=6).map(sns.kdeplot,"log_times").add_legend();
plt.show();
```

```
99.0 percentile value is 46.75
99.1 percentile value is 48.06666666666667
99.2 percentile value is 49.56666666666667
99.3 percentile value is 51.28333333333333
99.4 percentile value is 53.31666666666667
99.5 percentile value is 55.833333333333336
99.6 percentile value is 59.13333333333333
99.7 percentile value is 63.9
99.8 percentile value is 71.86666666666666
99.9 percentile value is 101.6
```



2.4. Speed

[31]:

```
updated_duration_of_trip['Speed'] = 60*(updated_duration_of_trip['trip_distance']/updated_duration_of_trip['trip_times'])
sns.boxplot(y="Speed", data = updated_duration_of_trip)
plt.show()
```

[20]:
```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["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
```

[21]:
```
#removing further outliers based on the 99.9th percentile value
updated_duration_of_trip=updated_duration_of_trip[(updated_duration_of_trip.Speed>0) & (updated_duration_of_trip.Speed<45.3
```

2.5. Trip Distance

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

[33]:
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["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 18.0
99.1 percentile value is 18.2
99.2 percentile value is 18.4
99.3 percentile value is 18.6
99.4 percentile value is 18.9
99.5 percentile value is 19.2
99.6 percentile value is 19.56
99.7 percentile value is 20.01
99.8 percentile value is 20.6
99.9 percentile value is 21.34
100 percentile value is  22.99
```

[25]:
```
#removing further outliers based on the 99.9th percentile value
updated_duration_of_trip = updated_duration_of_trip[(updated_duration_of_trip.trip_distance>0) & (updated_duration_of_trip.
```

[26]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = updated_duration_of_trip)
plt.show()
```

2.6. Total Fare

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

```
[28]:
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["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])
```

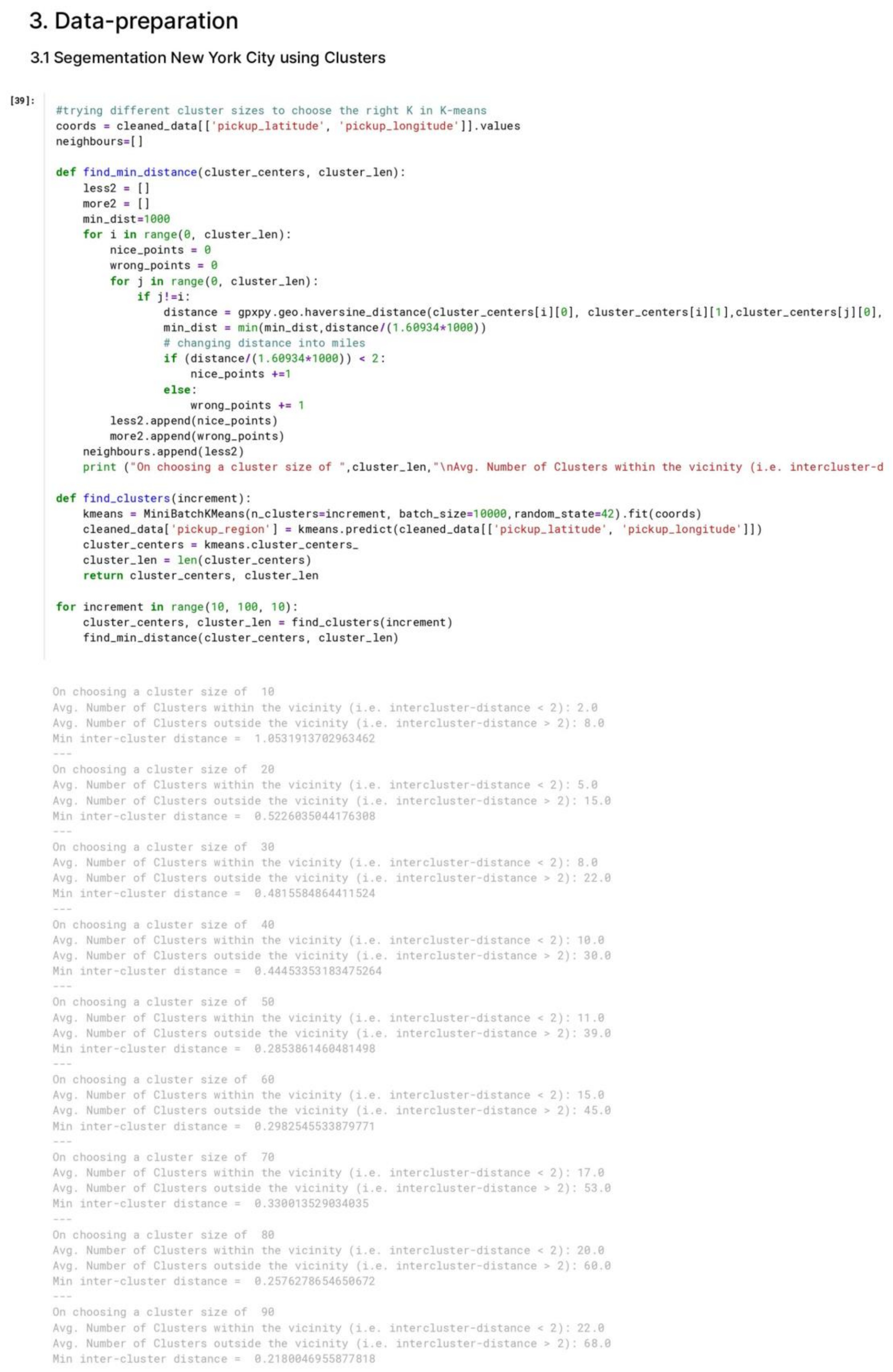
```
99.0 percentile value is 65.8
99.1 percentile value is 67.55
99.2 percentile value is 68.8
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.73
99.6 percentile value is 69.76
99.7 percentile value is 72.46
99.8 percentile value is 75.16
99.9 percentile value is 86.6
100 percentile value is  3950611.6
```

2.7. Finally Removing all other outliers

```
[34]:
#removing all outliers based on our univariate analysis above
def remove_outliers(new_frame):


    a = new_frame.shape[0]
    temp_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_longitude <= -73.7004) &\
                   (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitude <= 40.9176)) & \
                   ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitude >= 40.5774)& \
                   (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitude <= 40.9176))]
    b = temp_frame.shape[0]

    temp_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    c = temp_frame.shape[0]

    temp_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 23)]
    d = temp_frame.shape[0]

    temp_frame = new_frame[(new_frame.Speed <= 45.31) & (new_frame.Speed >= 0)]
    e = temp_frame.shape[0]

    temp_frame = new_frame[(new_frame.total_amount <86.6) & (new_frame.total_amount >0)]
    f = temp_frame.shape[0]

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

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 23)]
    new_frame = new_frame[(new_frame.Speed < 45.31) & (new_frame.Speed > 0)]
    new_frame = new_frame[(new_frame.total_amount <86.6) & (new_frame.total_amount >0)]
    new_frame = new_frame[new_frame.passenger_count < 6]
    print ("Total outliers removed",a - new_frame.shape[0])
    return new_frame
```

```
[35]:
cleaned_data = remove_outliers(frame_with_durations)
print("fraction of data points that remain after removing outliers", float(len(cleaned_data))/len(frame_with_durations))
```

```
Total outliers removed 828606
fraction of data points that remain after removing outliers 0.9350061251930154
```



3. Data-preparation

3.1 Segementation New York City using Clusters

[39]:

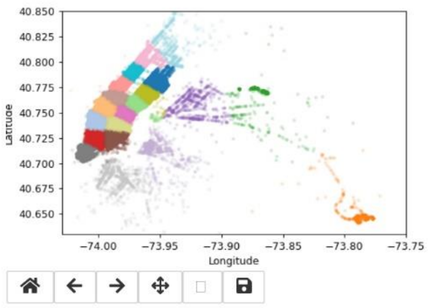
```
#trying different cluster sizes to choose the right K in K-means
coords = cleaned_data[['pickup_latitude', 'pickup_longitude']].values
neighbours=[]

def find_min_distance(cluster_centers, cluster_len):
    less2 = []
    more2 = []
    min_dist=1000
    for i in range(0, cluster_len):
        nice_points = 0
        wrong_points = 0
        for j in range(0, cluster_len):
            if j!=i:
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0], cluster_centers[i][1],cluster_centers[j][0],
                min_dist = min(min_dist,distance/(1.60934*1000))
                # changing distance into miles
                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 the vicinity (i.e. intercluster-d

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fit(coords)
    cleaned_data['pickup_region'] = kmeans.predict(cleaned_data[['pickup_latitude', 'pickup_longitude']])
    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 =  1.0531913702963462
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 15.0
Min inter-cluster distance =  0.5226035044176308
---
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 =  0.4815584864411524
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 10.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 30.0
Min inter-cluster distance =  0.44453353183475264
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 11.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 39.0
Min inter-cluster distance =  0.2853861460481498
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 15.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 45.0
Min inter-cluster distance =  0.2982545533879771
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 17.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 53.0
Min inter-cluster distance =  0.330013529034035
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 20.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 60.0
Min inter-cluster distance =  0.2576278654650672
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 22.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 68.0
Min inter-cluster distance =  0.2180046955877818
```

```
[40]:
kmeans = MiniBatchKMeans(n_clusters=20, batch_size=10000,random_state=0).fit(coords)
cleaned_data['pickup_region'] = kmeans.predict(cleaned_data[['pickup_latitude', 'pickup_longitude']])
```

```
[43]:
#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], s=10, lw=0,
               c=frame.pickup_region.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(cleaned_data)
```

Figure 2



3.2 Time Binning Clusters into 10 min intervals

```
[44]:
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 to est
    tenminutewise_binned_unix_pickup_times=[(int((i-start_pickup_unix)/600)+33) for i in unix_pickup_times]
    frame['10min_bins'] = np.array(tenminutewise_binned_unix_pickup_times)
    return frame
```

[46]:
```
import math
cleaned_data['log_fare'] = np.log(cleaned_data['total_amount'])
cleaned_data['log_dist'] = np.log(cleaned_data['trip_distance'])
jan_2015_frame = add_pickup_bins(cleaned_data,1,2015)
jan_2015_groupby = jan_2015_frame[['pickup_region','10min_bins','trip_distance']].groupby(['pickup_region','10min_bins']).c
jan_2015_frame.head()
```

Out[46]:

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

[47]:
```
jan_2015_groupby.head()
```

Out[47]:

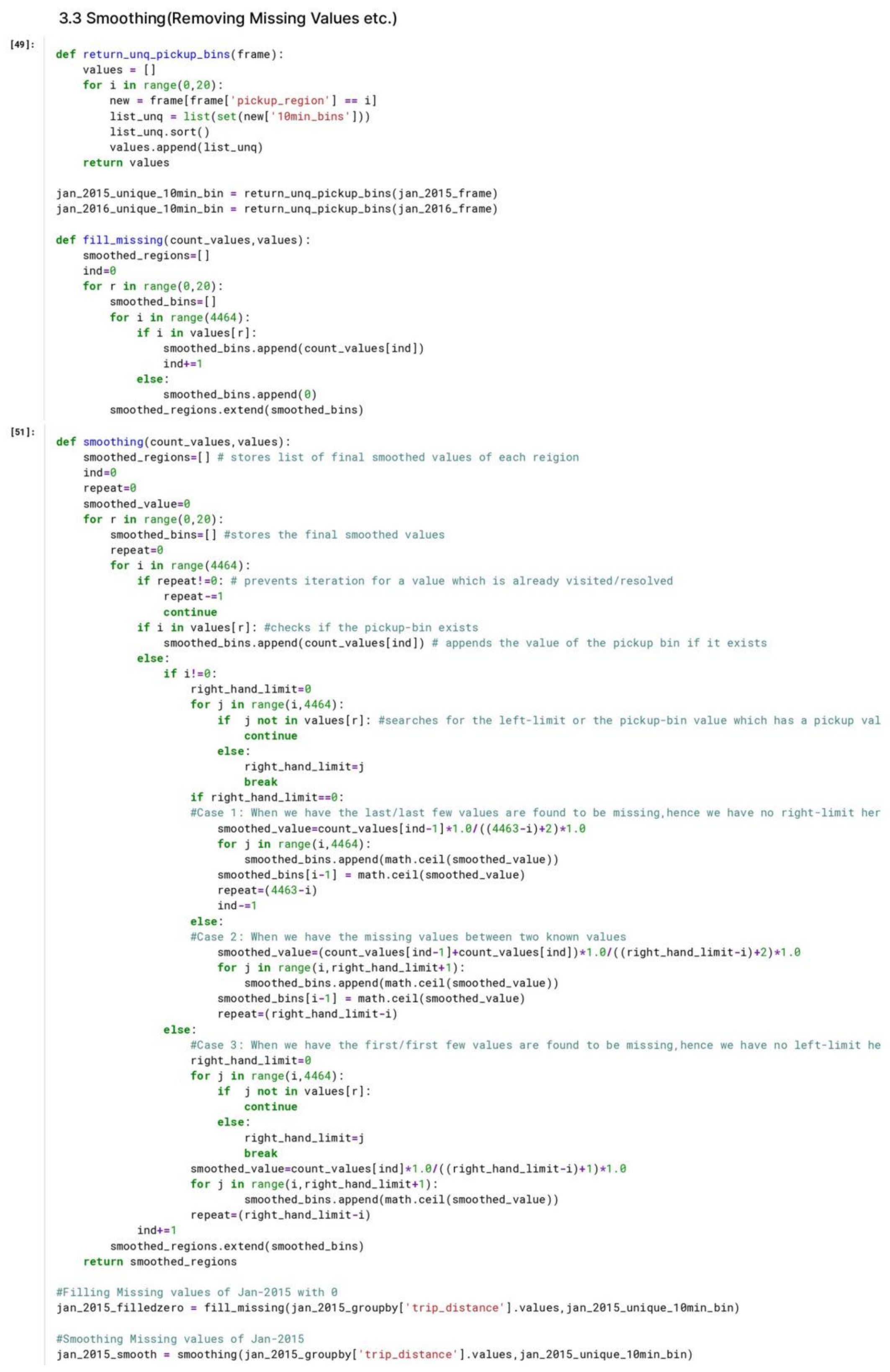
		trip_distance
pickup_region	**10min_bins**	
0	**33**	143
	34	273
	35	326
	36	338
	37	369

Doing same stuff for Jan-2016 Data

[48]:
```
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_region'] = kmeans.predict(frame_with_durations_outliers_removed[['pickup_
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(frame_with_durations_outliers_removed_20

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_no,year_no)
    final_groupby_frame = final_updated_frame[['pickup_region','10min_bins','trip_distance']].groupby(['pickup_region','10m

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('/kaggle/input/nycyellowtaxi/yellow_tripdata_2016-01/yellow_tripdata_2016-01.csv')
jan_2016_frame,jan_2016_pickups = datapreparation(month_jan_2016,kmeans,1,2016)
```

```
Return with trip times..
Remove outliers..
Total outliers removed 664287
Estimating clusters..
Final groupbying..
```

3.3 Smoothing(Removing Missing Values etc.)



3.3 Smoothing(Removing Missing Values etc.)

[49]:
```
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,20):
        new = frame[frame['pickup_region'] == i]
        list_unq = list(set(new['10min_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values

jan_2015_unique_10min_bin = return_unq_pickup_bins(jan_2015_frame)
jan_2016_unique_10min_bin = return_unq_pickup_bins(jan_2016_frame)

def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,20):
        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)
```

[51]:
```
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(0,20):
        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/resolved
                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 bin if it exists
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin value which has a pickup val
                            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,hence we have no right-limit her
                        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_hand_limit-i)+2)*1.0
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missing,hence we have no left-limit he
                    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

#Filling Missing values of Jan-2015 with 0
jan_2015_filledzero = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015_unique_10min_bin)

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_unique_10min_bin)
```

[56]:

```
print("number of 10 min. intervals among all the clusters ",len(jan_2015_filledzero))
```

```
number of 10 min. intervals among all the clusters  89280
```

[53]:

```
# Jan-2015 data is smoothed, Jan 2016 data missing values are filled with zero
jan_2016_smooth = fill_missing(jan_2016_pickups['trip_distance'].values,jan_2016_unique_10min_bin)
three_month_pickups_2016 = []
for i in range(0,20):
    three_month_pickups_2016.append(jan_2016_smooth[4464*i:4464*(i+1)])
```

[54]:

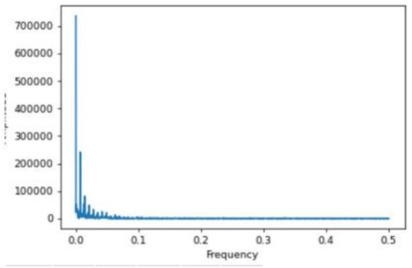
```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) values as jan-2016
ratios_jan = pd.DataFrame()
ratios_jan['Given']=jan_2015_smooth
ratios_jan['Prediction']=jan_2016_smooth
ratios_jan['Ratios']=ratios_jan['Prediction']*1.0/ratios_jan['Given']*1.0
```

4.Fourier Transform

[57]:

```
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()
```

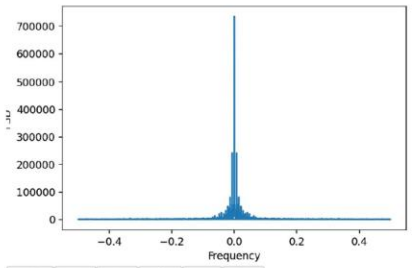
Figure 3

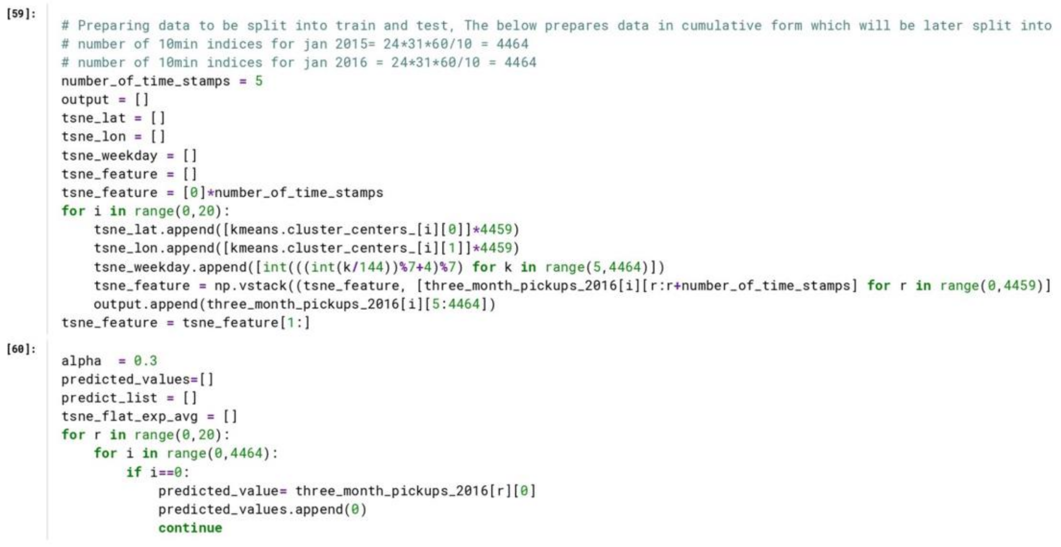


[58]:

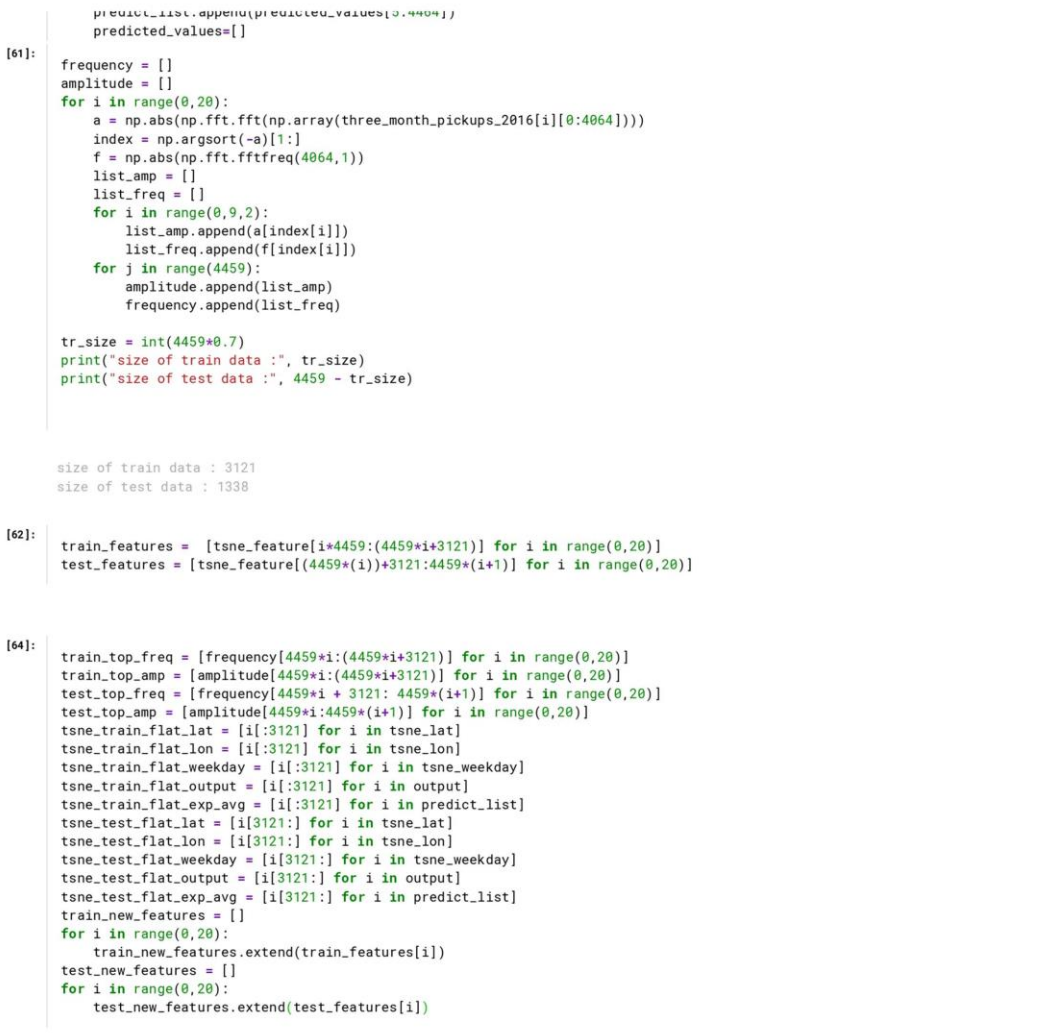
```
# ploting by taking PSD = absolute( complex valued amplitude)
plt.figure()
plt.plot( freq, np.abs(Y) )
plt.xlabel("Frequency")
plt.ylabel("PSD")
plt.show()
```

Figure 4





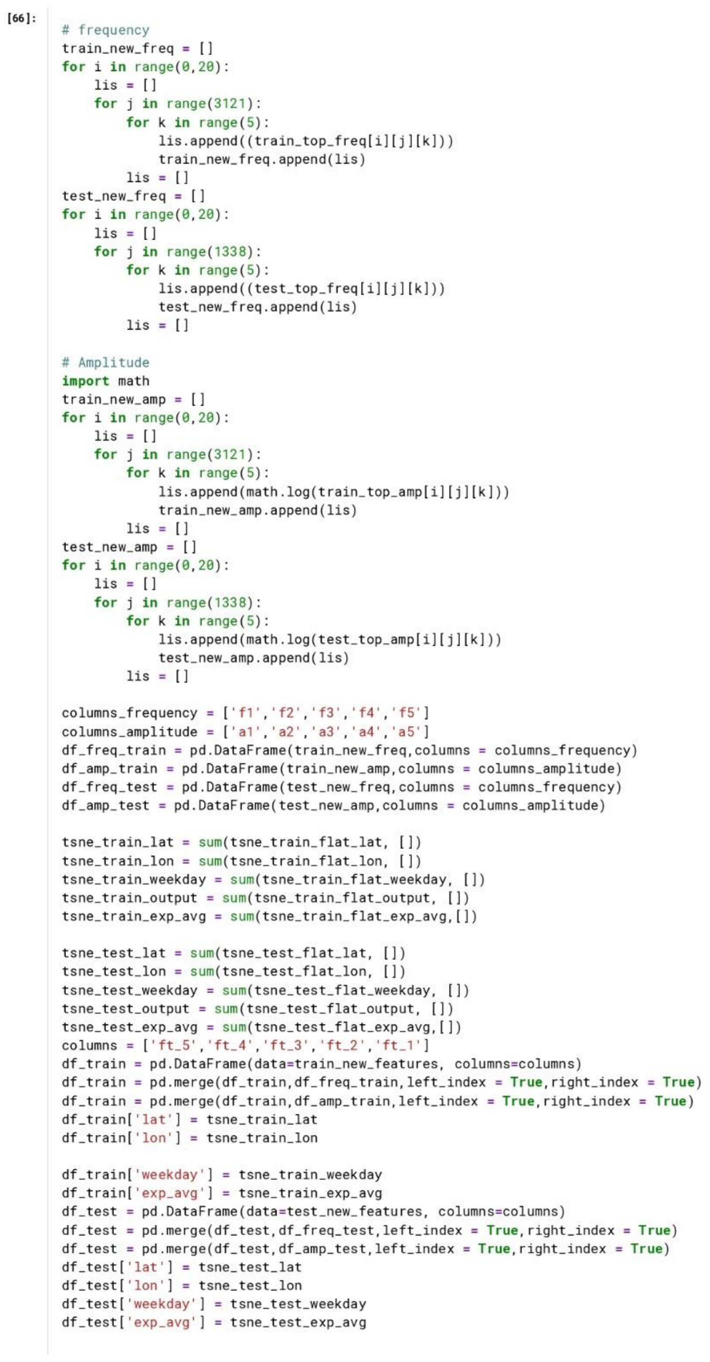
```
[59]:
    # Preparing data to be split into train and test, The below prepares data in cumulative form which will be later split into
    # number of 10min indices for jan 2015= 24*31*60/10 = 4464
    # number of 10min indices for jan 2016 = 24*31*60/10 = 4464
    number_of_time_stamps = 5
    output = []
    tsne_lat = []
    tsne_lon = []
    tsne_weekday = []
    tsne_feature = []
    tsne_feature = [0]*number_of_time_stamps
    for i in range(0,20):
        tsne_lat.append([kmeans.cluster_centers_[i][0]]*4459)
        tsne_lon.append([kmeans.cluster_centers_[i][1]]*4459)
        tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464)])
        tsne_feature = np.vstack((tsne_feature, [three_month_pickups_2016[i][r:r+number_of_time_stamps] for r in range(0,4459)]
        output.append(three_month_pickups_2016[i][5:4464])
    tsne_feature = tsne_feature[1:]

[60]:
    alpha  = 0.3
    predicted_values=[]
    predict_list = []
    tsne_flat_exp_avg = []
    for r in range(0,20):
        for i in range(0,4464):
            if i==0:
                predicted_value= three_month_pickups_2016[r][0]
                predicted_values.append(0)
                continue
```



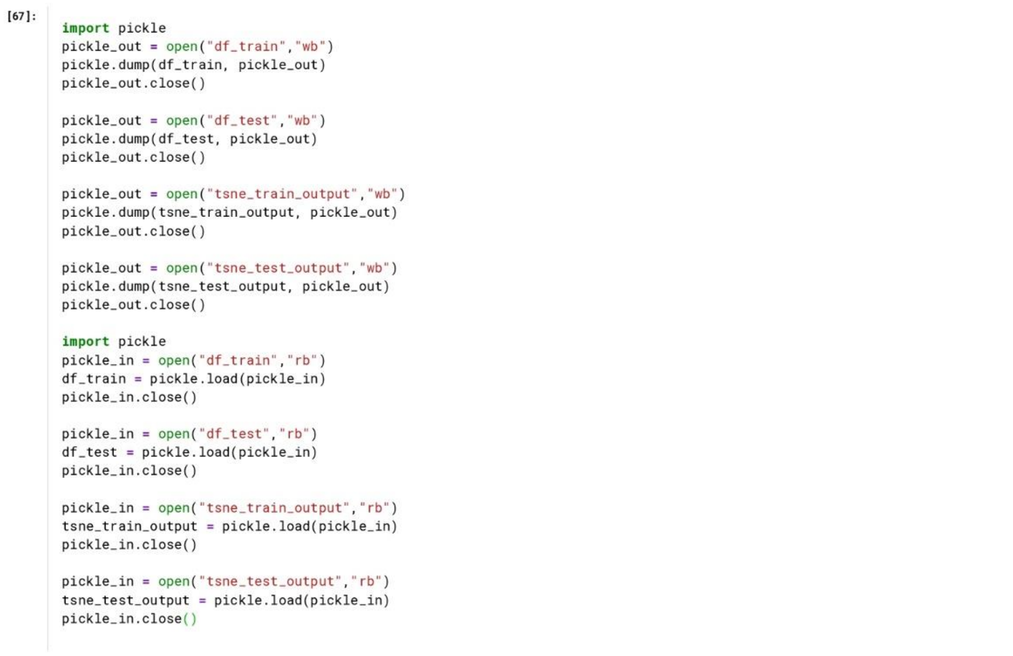
```
        predict_list.append(predicted_values[5:4464])
        predicted_values=[]

[61]:
    frequency = []
    amplitude = []
    for i in range(0,20):
        a = np.abs(np.fft.fft(np.array(three_month_pickups_2016[i][0:4064])))
        index = np.argsort(-a)[1:]
        f = np.abs(np.fft.fftfreq(4064,1))
        list_amp = []
        list_freq = []
        for i in range(0,9,2):
            list_amp.append(a[index[i]])
            list_freq.append(f[index[i]])
        for j in range(4459):
            amplitude.append(list_amp)
            frequency.append(list_freq)

    tr_size = int(4459*0.7)
    print("size of train data :", tr_size)
    print("size of test data :", 4459 - tr_size)
```

```
size of train data : 3121
size of test data : 1338
```

```
[62]:
    train_features =  [tsne_feature[i*4459:(4459*i+3121)] for i in range(0,20)]
    test_features = [tsne_feature[(4459*(i))+3121:4459*(i+1)] for i in range(0,20)]

[64]:
    train_top_freq = [frequency[4459*i:(4459*i+3121)] for i in range(0,20)]
    train_top_amp = [amplitude[4459*i:(4459*i+3121)] for i in range(0,20)]
    test_top_freq = [frequency[4459*i + 3121: 4459*(i+1)] for i in range(0,20)]
    test_top_amp = [amplitude[4459*i:4459*(i+1)] for i in range(0,20)]
    tsne_train_flat_lat = [i[:3121] for i in tsne_lat]
    tsne_train_flat_lon = [i[:3121] for i in tsne_lon]
    tsne_train_flat_weekday = [i[:3121] for i in tsne_weekday]
    tsne_train_flat_output = [i[:3121] for i in output]
    tsne_train_flat_exp_avg = [i[:3121] for i in predict_list]
    tsne_test_flat_lat = [i[3121:] for i in tsne_lat]
    tsne_test_flat_lon = [i[3121:] for i in tsne_lon]
    tsne_test_flat_weekday = [i[3121:] for i in tsne_weekday]
    tsne_test_flat_output = [i[3121:] for i in output]
    tsne_test_flat_exp_avg = [i[3121:] for i in predict_list]
    train_new_features = []
    for i in range(0,20):
        train_new_features.extend(train_features[i])
    test_new_features = []
    for i in range(0,20):
        test_new_features.extend(test_features[i])
```

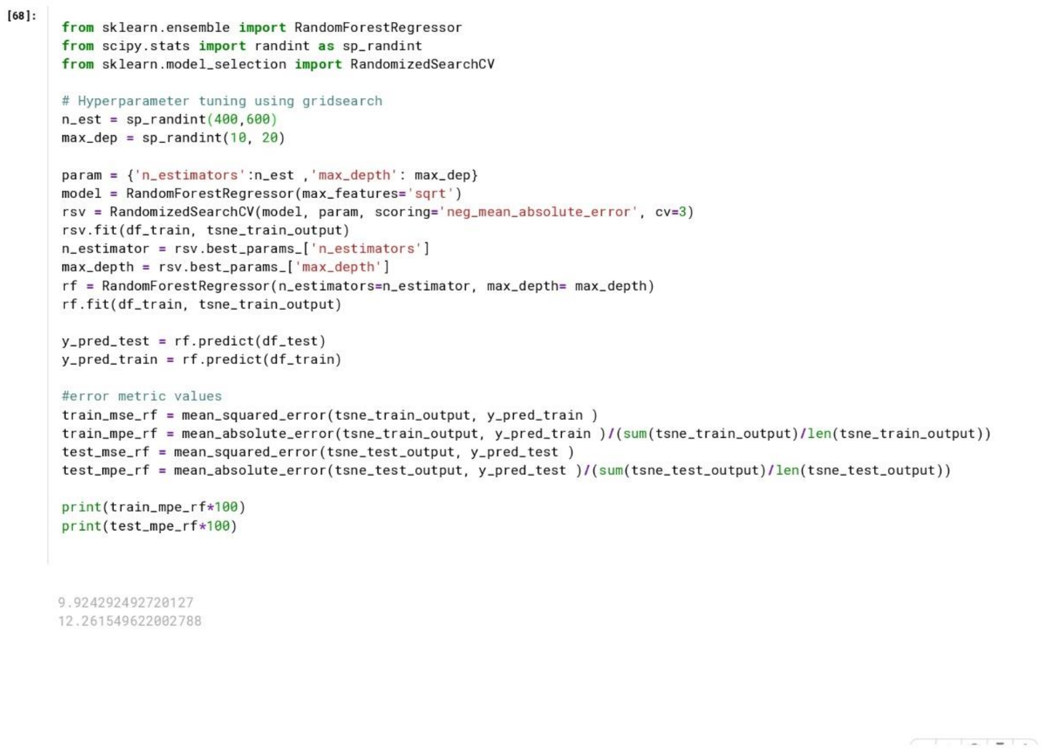


```
[66]:
# frequency
train_new_freq = []
for i in range(0,20):
    lis = []
    for j in range(3121):
        for k in range(5):
            lis.append((train_top_freq[i][j][k]))
            train_new_freq.append(lis)
        lis = []
test_new_freq = []
for i in range(0,20):
    lis = []
    for j in range(1338):
        for k in range(5):
            lis.append((test_top_freq[i][j][k]))
            test_new_freq.append(lis)
        lis = []

# Amplitude
import math
train_new_amp = []
for i in range(0,20):
    lis = []
    for j in range(3121):
        for k in range(5):
            lis.append(math.log(train_top_amp[i][j][k]))
            train_new_amp.append(lis)
        lis = []
test_new_amp = []
for i in range(0,20):
    lis = []
    for j in range(1338):
        for k in range(5):
            lis.append(math.log(test_top_amp[i][j][k]))
            test_new_amp.append(lis)
        lis = []

columns_frequency = ['f1','f2','f3','f4','f5']
columns_amplitude = ['a1','a2','a3','a4','a5']
df_freq_train = pd.DataFrame(train_new_freq,columns = columns_frequency)
df_amp_train = pd.DataFrame(train_new_amp,columns = columns_amplitude)
df_freq_test = pd.DataFrame(test_new_freq,columns = columns_frequency)
df_amp_test = pd.DataFrame(test_new_amp,columns = columns_amplitude)

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

tsne_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,[])
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1']
df_train = pd.DataFrame(data=train_new_features, columns=columns)
df_train = pd.merge(df_train,df_freq_train,left_index = True,right_index = True)
df_train = pd.merge(df_train,df_amp_train,left_index = True,right_index = True)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon

df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg
df_test = pd.DataFrame(data=test_new_features, columns=columns)
df_test = pd.merge(df_test,df_freq_test,left_index = True,right_index = True)
df_test = pd.merge(df_test,df_amp_test,left_index = True,right_index = True)
df_test['lat'] = tsne_test_lat
df_test['lon'] = tsne_test_lon
df_test['weekday'] = tsne_test_weekday
df_test['exp_avg'] = tsne_test_exp_avg
```



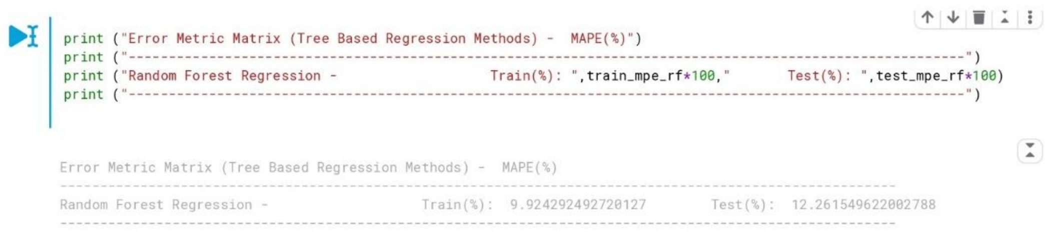
```
[67]:
import pickle
pickle_out = open("df_train","wb")
pickle.dump(df_train, pickle_out)
pickle_out.close()

pickle_out = open("df_test","wb")
pickle.dump(df_test, pickle_out)
pickle_out.close()

pickle_out = open("tsne_train_output","wb")
pickle.dump(tsne_train_output, pickle_out)
pickle_out.close()

pickle_out = open("tsne_test_output","wb")
pickle.dump(tsne_test_output, pickle_out)
pickle_out.close()

import pickle
pickle_in = open("df_train","rb")
df_train = pickle.load(pickle_in)
pickle_in.close()

pickle_in = open("df_test","rb")
df_test = pickle.load(pickle_in)
pickle_in.close()

pickle_in = open("tsne_train_output","rb")
tsne_train_output = pickle.load(pickle_in)
pickle_in.close()

pickle_in = open("tsne_test_output","rb")
tsne_test_output = pickle.load(pickle_in)
pickle_in.close()
```

5. Applying Random Forest Regressor



```
[68]:
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

# Hyperparameter tuning using gridsearch
n_est = sp_randint(400,600)
max_dep = sp_randint(10, 20)

param = {'n_estimators':n_est ,'max_depth': max_dep}
model = RandomForestRegressor(max_features='sqrt')
rsv = RandomizedSearchCV(model, param, scoring='neg_mean_absolute_error', cv=3)
rsv.fit(df_train, tsne_train_output)
n_estimator = rsv.best_params_['n_estimators']
max_depth = rsv.best_params_['max_depth']
rf = RandomForestRegressor(n_estimators=n_estimator, max_depth= max_depth)
rf.fit(df_train, tsne_train_output)

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

#error metric values
train_mse_rf = mean_squared_error(tsne_train_output, y_pred_train )
train_mpe_rf = mean_absolute_error(tsne_train_output, y_pred_train )/(sum(tsne_train_output)/len(tsne_train_output))
test_mse_rf = mean_squared_error(tsne_test_output, y_pred_test )
test_mpe_rf = mean_absolute_error(tsne_test_output, y_pred_test )/(sum(tsne_test_output)/len(tsne_test_output))

print(train_mpe_rf*100)
print(test_mpe_rf*100)
```

```
9.924292492720127
12.261549622002788
```



```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)")
print ("--------------------------------------------------------------------------------------------------------")
print ("Random Forest Regression -                     Train(%): ",train_mpe_rf*100,"       Test(%): ",test_mpe_rf*100)
print ("--------------------------------------------------------------------------------------------------------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)
----------------------------------------------------------------------------------------------------
Random Forest Regression -                    Train(%):  9.924292492720127        Test(%):  12.261549622002788
----------------------------------------------------------------------------------------------------
```

8 REFERENCES

https://stackoverflow.com/questions/27546476/what-fft-descriptors-should-be-used-as-feature-to-implement-classification-or-cl

https://dsp.stackexchange.com/questions/10062/when-should-i-calculate-psd-instead-of-plain-fft-magnitude-spectrum