Taxi demand prediction in New York City

In [1]:

```
#Importing Libraries
# pip3 install graphviz
#pip3 install dask
#pip3 install toolz
#pip3 install cloudpickle
# https://www.youtube.com/watch?v=ieW3G7ZzRZ0
# https://github.com/dask/dask-tutorial
# please do go through this python notebook: https://github.com/dask/dask-
tutorial/blob/master/07_dataframe.ipynb
import dask.dataframe as dd#similar to pandas

import pandas as pd#pandas to create small dataframes

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

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

import time #Convert to unix time

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

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

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

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

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

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

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

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

In [364]:

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

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

In [3]:

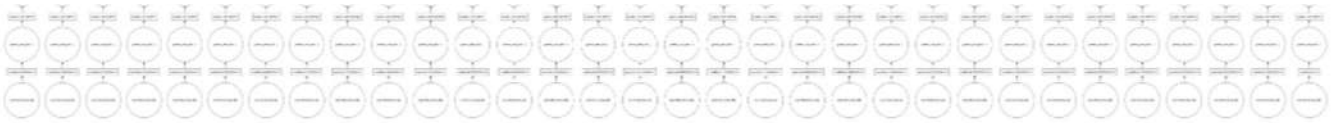
```
# However unlike Pandas, operations on dask.dataframes don't trigger immediate computation,
# instead they add key-value pairs to an underlying Dask graph. Recall that in the diagram below,
# circles are operations and rectangles are results.

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

Out[3]:

In [4]:

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

Out[4]:



Features in the dataset:

Y= store and forward trip N= not a store and forward trip

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.
Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_ latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. 1. Credit card 2. Cash 3. No charge 4. Dispute 5. Unknown 6. Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes. the $0.50 and 1$ rush hour and overnight charges.
MTA_tax	0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips.Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

ML Problem Formulation

Time-series forecasting and Regression

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

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

Performance metrics

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

Data Cleaning

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

In [5]:

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

Out[5]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latit
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	-73.993896	40.750111
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30	-74.001648	40.724243
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80	-73.963341	40.802788
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50	-74.009087	40.713818
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00	-73.971176	40.762428

1. Pickup Latitude and Pickup Longitude

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

In [14]:

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

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

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

map_osm = folium.Map(location=[40.734695, -73.990372])

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

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

2. Dropoff Latitude & Dropoff Longitude

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

In [15]:

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

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

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

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

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

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

3. Trip Durations:

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

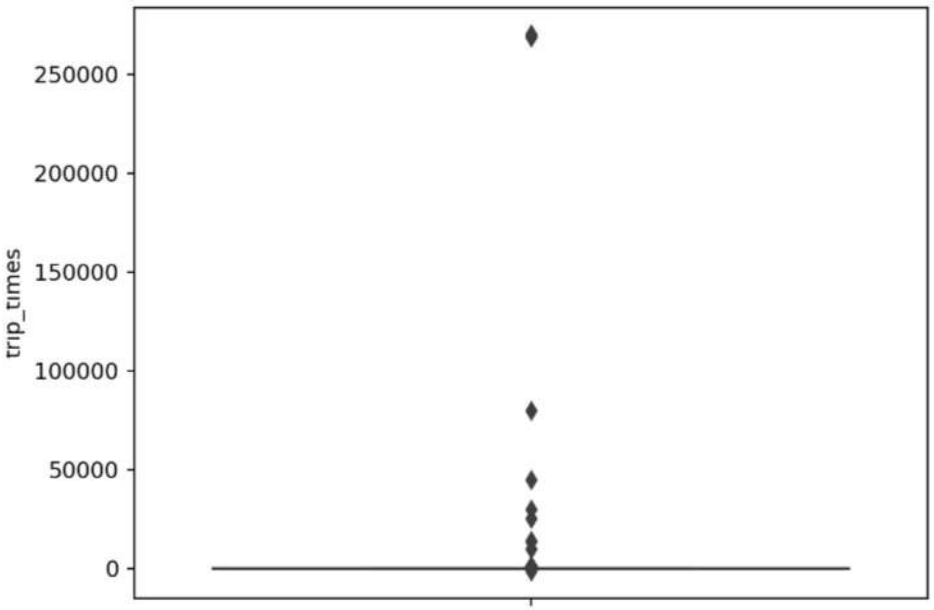
In [16]:

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

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



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

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

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

```
# print(frame_with_durations.head())
#   passenger_count trip_distance pickup_longitude pickup_latitude dropoff_longitude
dropoff_latitude total_amount trip_times pickup_times Speed
#   1                 1.59       -73.993896         40.750111      -73.974785       40.750618
17.05     18.050000 1.421329e+09 5.285319
#   1                 3.30      -74.001648        40.724243     -73.994415       40.759109
.80     19.833333 1.420902e+09 9.983193
#   1                 1.80       -73.963341        40.802788      -73.951820       40.824413
10.80     10.050000 1.420902e+09 10.746269
#   1                 0.50      -74.009087        40.713818      -74.004326       40.719986
4.80      1.866667 1.420902e+09 16.071429
#   1                 3.00      -73.971176        40.762428      -74.004181       40.742653
6.30     19.316667 1.420902e+09 9.318378
frame_with_durations = return_with_trip_times(month)
```

In [17]:

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



In [18]:

```
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
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 -1200.0
10 percentile value is 3.7333333333333334
20 percentile value is 5.233333333333333
30 percentile value is 6.633333333333334
40 percentile value is 8.083333333333334
50 percentile value is 9.683333333333334
60 percentile value is 11.55
70 percentile value is 13.9
80 percentile value is 17.166666666666668
90 percentile value is 22.866666666666667
100 percentile value is  270300.55
```

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

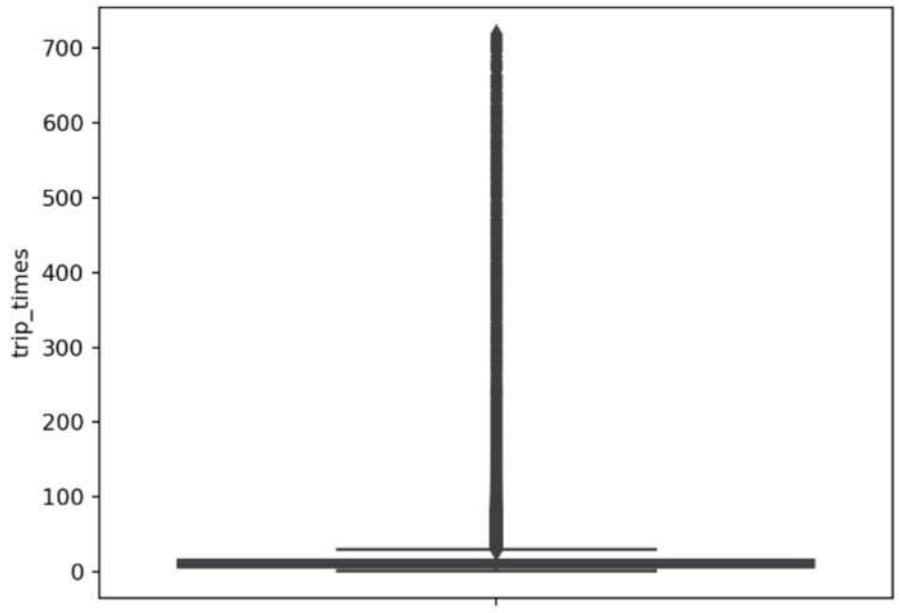
```
90 percentile value is 22.866666666666667
91 percentile value is 23.75
92 percentile value is 24.75
93 percentile value is 25.9
94 percentile value is 27.233333333333334
95 percentile value is 28.833333333333332
96 percentile value is 30.85
97 percentile value is 33.56666666666667
98 percentile value is 37.6
99 percentile value is 45.15
100 percentile value is  270300.55
```

In [20]:

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

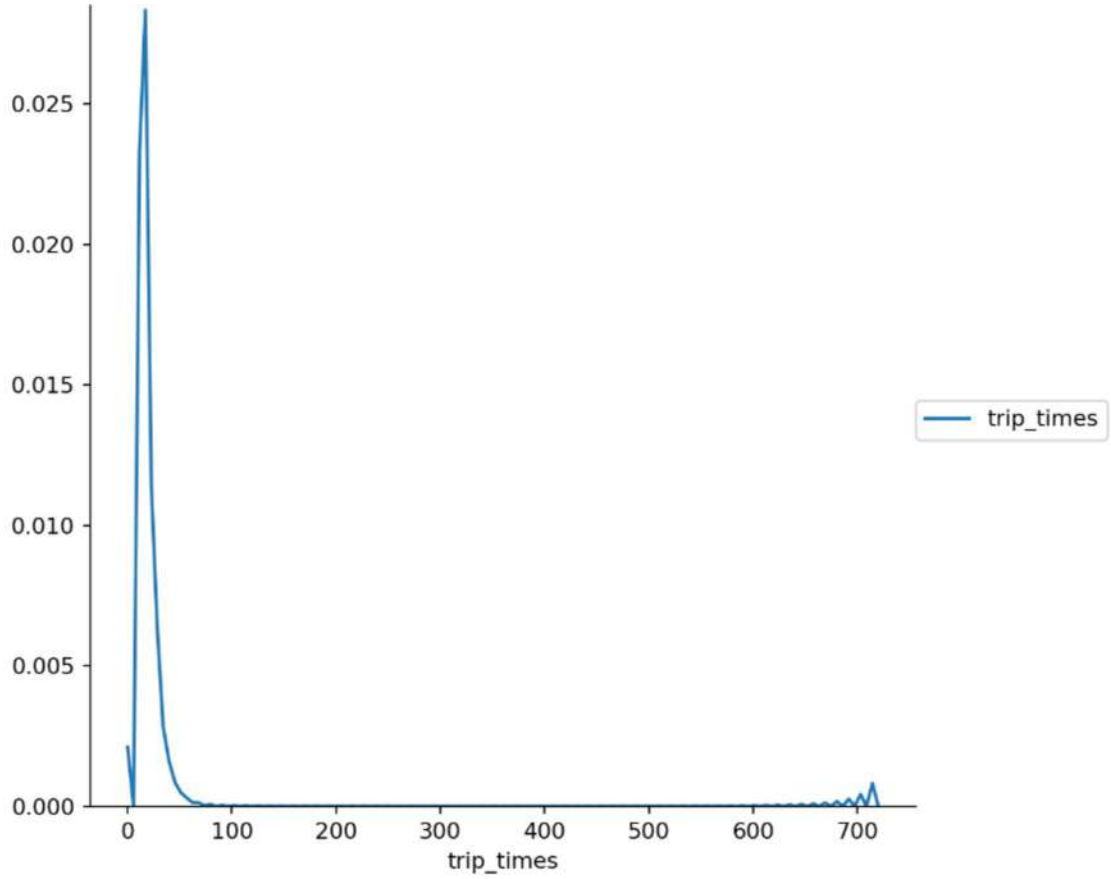
In [21]:

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



In [22]:

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

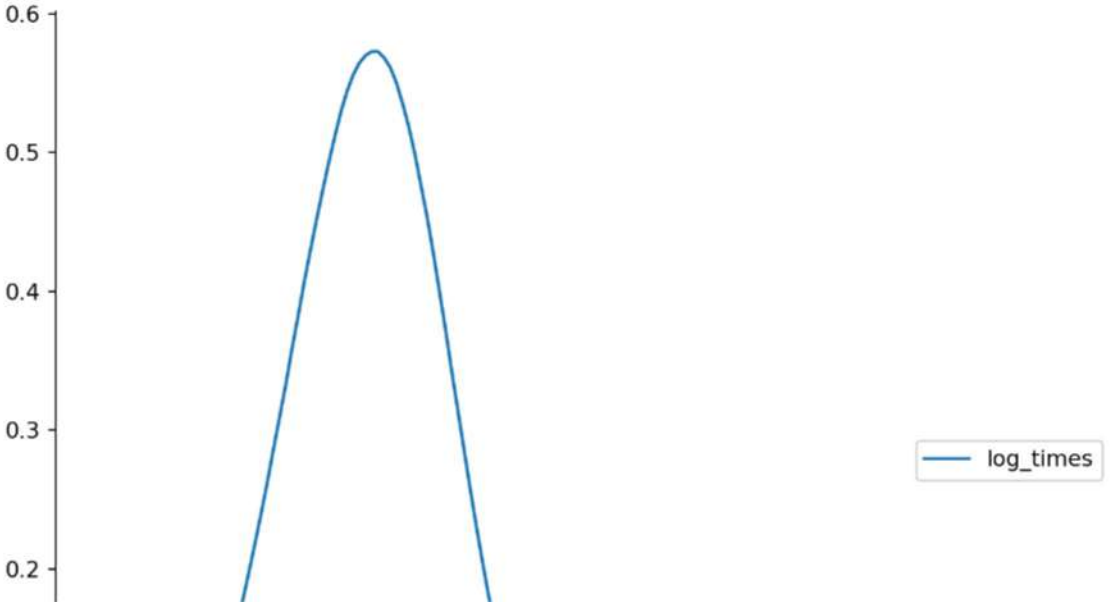


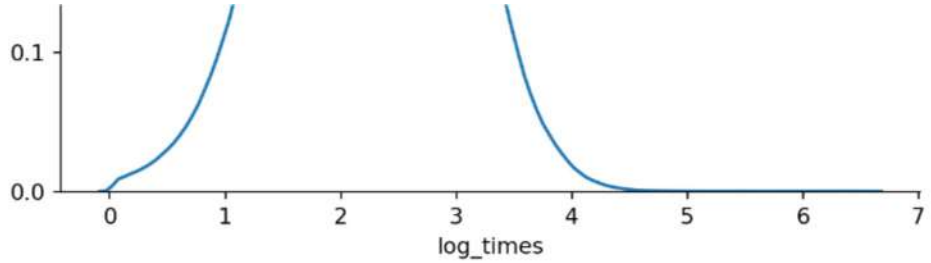
In [23]:

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

In [24]:

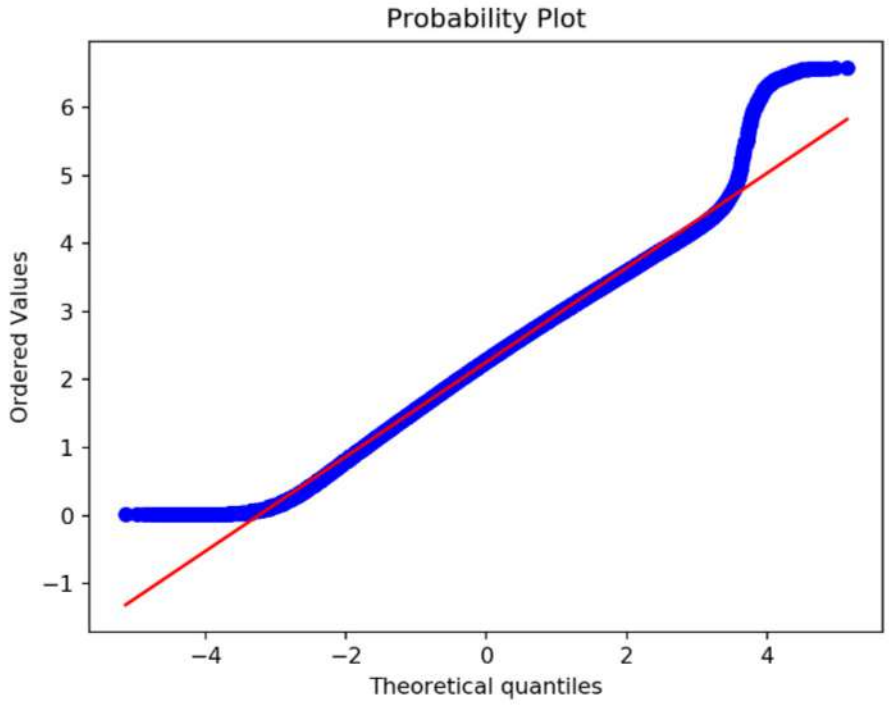
```
#pdf of log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```





In [27]:

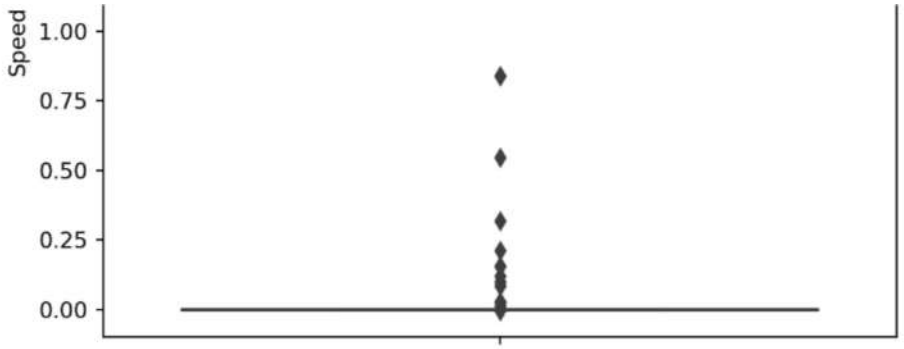
```
#Q-Q plot for checking if trip-times is log-normal
import scipy
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [28]:

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



In [29]:

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

```
0 percentile value is 0.0
10 percentile value is 6.683168316831683
20 percentile value is 8.126410835214447
30 percentile value is 9.27400468384075
40 percentile value is 10.353982300884956
50 percentile value is 11.474103585657371
60 percentile value is 12.73846153846154
70 percentile value is 14.311926605504588
80 percentile value is 16.584569732937688
90 percentile value is 21.174234424498415
100 percentile value is  192857142.85714284
```

In [30]:

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

```
90 percentile value is 21.174234424498415
91 percentile value is 21.95121951219512
92 percentile value is 22.857142857142854
93 percentile value is 23.89354838709677
94 percentile value is 25.094117647058827
95 percentile value is 26.499454743729554
96 percentile value is 28.178913738019173
97 percentile value is 30.278145695364238
98 percentile value is 33.079518072289154
99 percentile value is 37.18826405867971
100 percentile value is  192857142.85714284
```

In [31]:

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

```
99.0 percentile value is 37.18826405867971
99.1 percentile value is 37.738197424892704
99.2 percentile value is 38.32985386221294
```

```
99.5 percentile value is 40.51305861868834
99.6 percentile value is 41.43222506393862
99.7 percentile value is 42.55841584158416
99.8 percentile value is 44.02896451846488
99.9 percentile value is 46.36758321273516
100 percentile value is  192857142.85714284
```

In [32]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.Speed>0) &
(frame_with_durations.Speed<45.31)]
```

In [33]:

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

Out[33]:

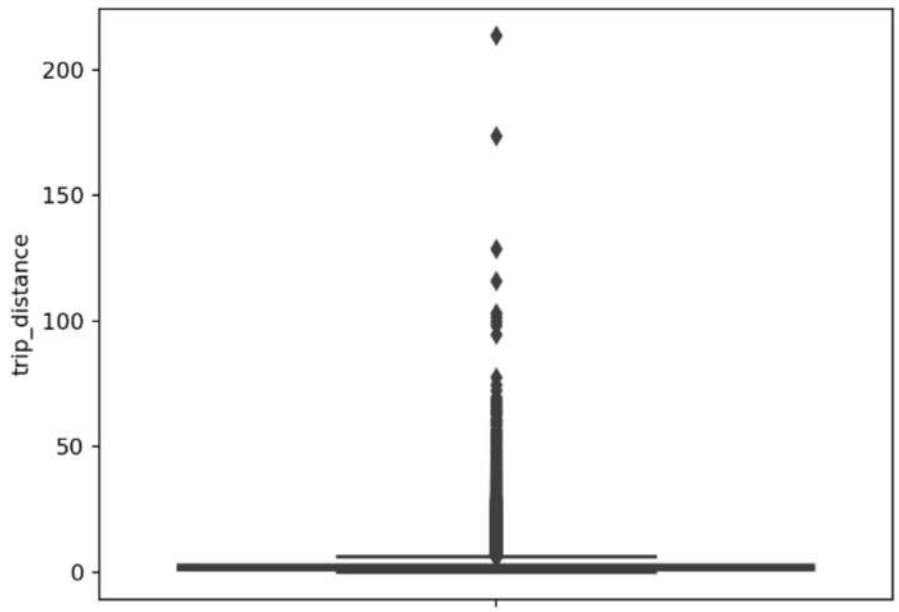
```
12.944727634036159
```

The avg speed in Newyork speed is 12.94miles/hr, so a cab driver can travel2 miles per 10min on avg.

4. Trip Distance

In [34]:

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



In [35]:

```
#calculating trip distance values at each percntile 0,10,20,30,40,50,60,70,80,90,100
for i in range(0,100,10):
```

```
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 1.4
50 percentile value is 1.7
60 percentile value is 2.1
70 percentile value is 2.67
80 percentile value is 3.65
90 percentile value is 6.17
100 percentile value is  213.6
```

In [36]:

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

```
90 percentile value is 6.17
91 percentile value is 6.7
92 percentile value is 7.36
93 percentile value is 8.2
94 percentile value is 9.02
95 percentile value is 9.9
96 percentile value is 10.97
97 percentile value is 12.71
98 percentile value is 16.67
99 percentile value is 18.4
100 percentile value is  213.6
```

In [37]:

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

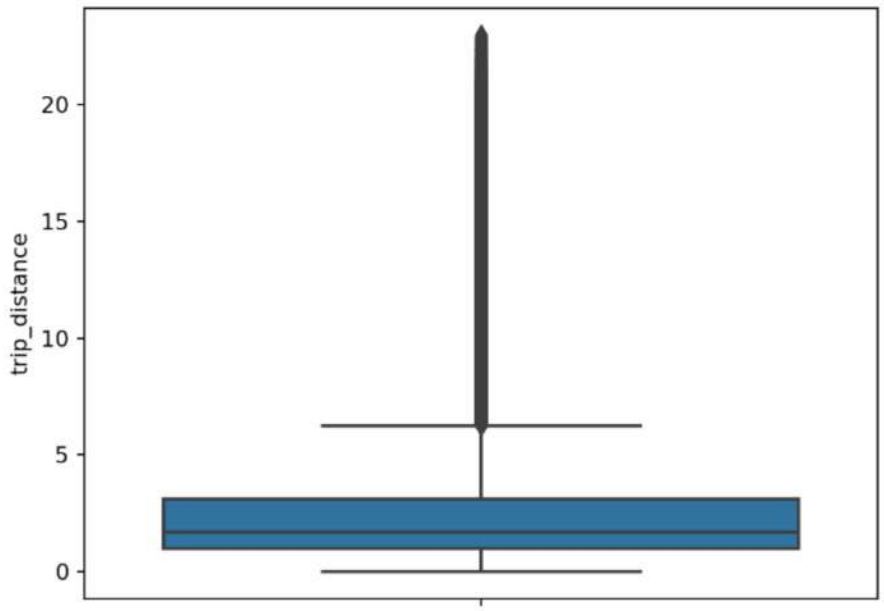
```
99.0 percentile value is 18.4
99.1 percentile value is 18.59
99.2 percentile value is 18.8
99.3 percentile value is 19.06
99.4 percentile value is 19.36
99.5 percentile value is 19.7
99.6 percentile value is 20.18
99.7 percentile value is 20.7
99.8 percentile value is 21.38
99.9 percentile value is 22.7
100 percentile value is  213.6
```

In [38]:

```
#removing further outliers based on the 99.9th percentile value
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_distance>0) &
(frame_with_durations.trip_distance<23)]
```

In [39]:

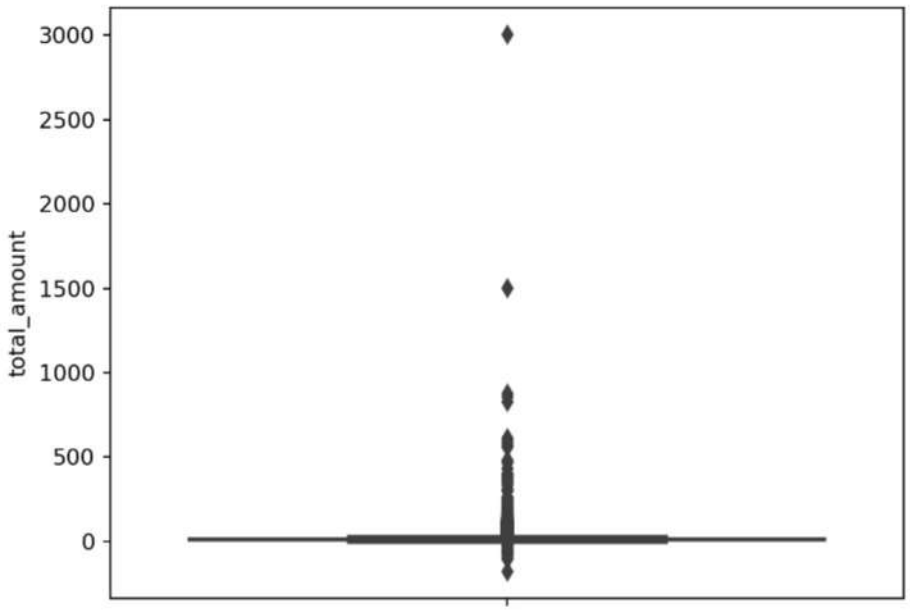
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [40]:

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



In [41]:

```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80,90,100
```

```
    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 -176.3
10 percentile value is 6.3
20 percentile value is 7.5
30 percentile value is 8.6
40 percentile value is 9.8
50 percentile value is 11.0
60 percentile value is 12.5
70 percentile value is 14.75
80 percentile value is 18.12
90 percentile value is 26.0
100 percentile value is  3006.35
```

In [42]:

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

```
90 percentile value is 26.0
91 percentile value is 27.8
92 percentile value is 29.75
93 percentile value is 32.3
94 percentile value is 35.63
95 percentile value is 39.13
96 percentile value is 43.13
97 percentile value is 49.2
98 percentile value is 58.13
99 percentile value is 66.65
100 percentile value is  3006.35
```

In [43]:

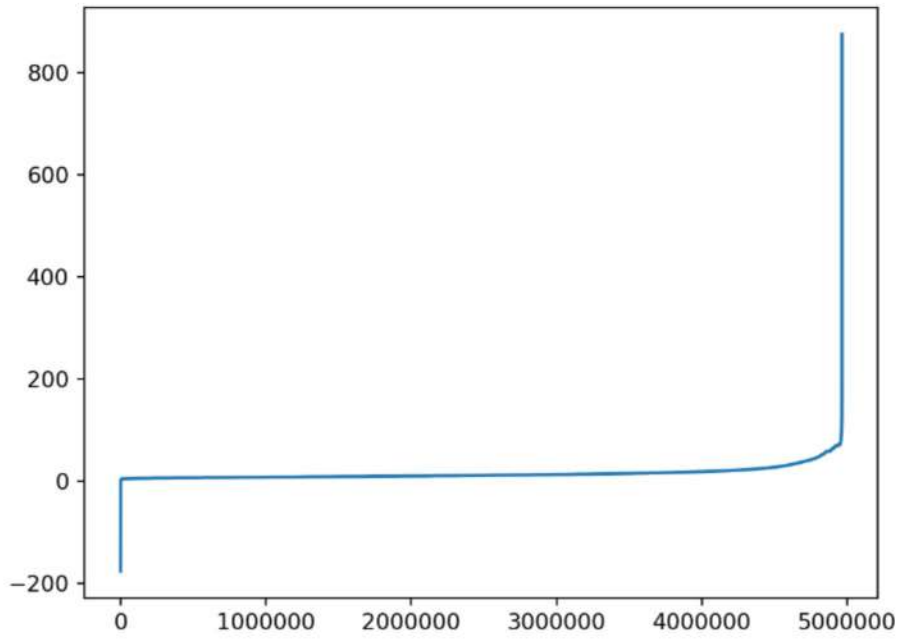
```
#calculating total fare amount values at each percntile
99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

```
99.0 percentile value is 66.65
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 69.6
99.5 percentile value is 69.73
99.6 percentile value is 69.75
99.7 percentile value is 72.46
99.8 percentile value is 75.33
99.9 percentile value is 87.3
100 percentile value is  3006.35
```

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

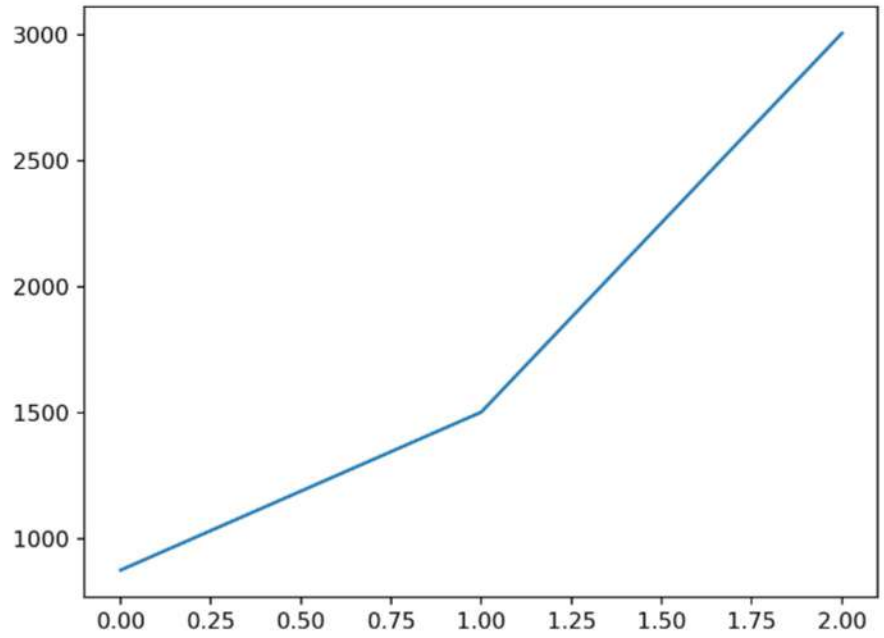
In [44]:

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



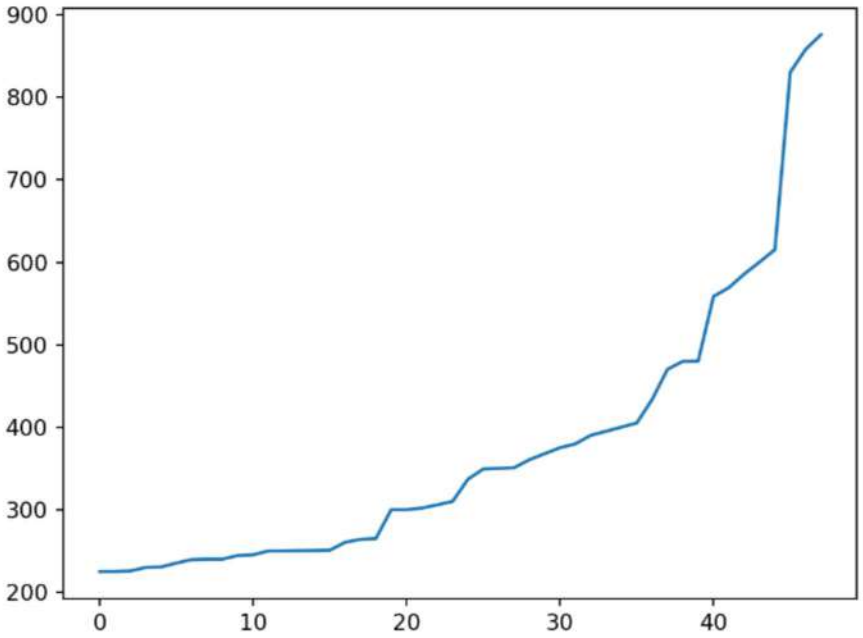
In [45]:

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



In [46]:

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



Remove all outliers/erronous points.

In [47]:

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


    a = new_frame.shape[0]
    print ("Number of pickup records = ",a)
    temp_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_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]
    print ("Number of outlier coordinates lying outside NY boundaries:",(a-b))


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


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

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

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


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_longitude <
= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_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 <1000) & (new_frame.total_amount >0)]

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

In [0]:

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

```
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 [49]:

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

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

def find_clusters(increment):
```

```
kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

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

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 7.0
Min inter-cluster distance =  0.9021992479685559
---
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 =  0.6377479356868238
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 7.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 23.0
Min inter-cluster distance =  0.5056237259045817
---
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 =  0.422097622420765
---
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 =  0.3864255614282014
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 44.0
Min inter-cluster distance =  0.2188940094900535
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 19.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 51.0
Min inter-cluster distance =  0.23995198567819048
---
On choosing a cluster size of  80
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): 58.0
Min inter-cluster distance =  0.10054452037896516
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 25.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 65.0
Min inter-cluster distance =  0.15988500241440332
---
```

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 [19]:

```
# if check for the 50 clusters you can observe that there are two clusters with only 0.3 miles apa
rt from each other
# so we choose 40 clusters for solve the further problem

# Getting 40 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(coords)
```

Plotting the cluster centers:

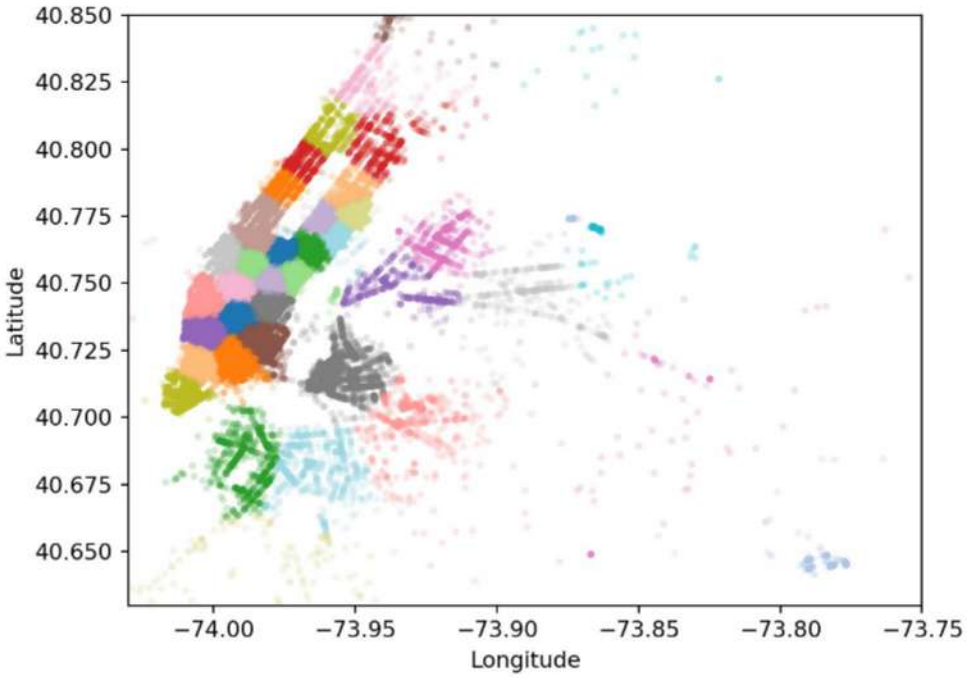
In [52]:

```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372])
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])), popup=(str(cluster_centers[i
][0])+str(cluster_centers[i][1]))).add_to(map_osm)
map_osm.save('cluster_center2')
```

Plotting the clusters:

In [18]:

```
#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_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 [54]:

```
# 1422748800 : 2015-02-01 00:00:00
# 1425168000 : 2015-03-01 00:00:00
# 1427846400 : 2015-04-01 00:00:00
# 1430438400 : 2015-05-01 00:00:00
# 1433116800 : 2015-06-01 00:00:00

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

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

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

In [55]:

```
frame_with_durations_outliers_removed.head()
```

Out[55]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	tri
0	5	4.00	-73.971436	40.760201	-73.921181	40.768269	14.5	11
3	1	0.80	-73.860847	40.757294	-73.868111	40.752285	6.3	3.
4	2	2.57	-73.969017	40.754269	-73.994133	40.761600	15.8	21
5	1	1.58	-73.987579	40.765270	-73.976921	40.776970	12.2	10
6	3	2.50	-73.957008	40.774502	-73.966019	40.800617	11.8	10

In [97]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins

jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2015)
jan_2015_01_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance']].groupby(['p
ickup_cluster','pickup_bins']).count()
```

In [99]:

```
# we add two more columns 'pickup_cluster'(to which cluster it belogns to)
# and 'pickup_bins' (to which 10min intravel the trip belongs to)
jan_2015_frame.tail()
```

Out[99]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amou
122617	2	7.97	-73.874451	40.774040	-73.960220	40.806362	35.13
122618	1	2.09	-73.981598	40.780685	-73.969543	40.797710	10.80
122619	5	1.92	-73.992844	40.758205	-73.993500	40.737671	14.30
122620	1	1.28	-73.991829	40.750160	-73.982338	40.756161	13.30
122621	1	0.50	-73.954529	40.764095	-73.960724	40.760784	8.16

In [108]:

```
# hear the trip_distance represents the number of pickups that are happend in that particular 10mi
n intravel
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels 24*31*60/10 =4464
bins)
jan_2015_01_groupby.head()
```

Out[108]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**1**	58
	2	130
	3	155
	4	215
	5	215

In [2]:

```
# it contains the preprocessed data of feb 2015 which was preprocessed by me earlier
n=pd.read_csv('yellow_tripdata_2015-02.csv')
```

In [4]:

```
n.tail()
```

Out[4]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amo
4856491	1	3.18	-73.980759	40.738060	-73.982979	40.771656	25.56
4856492	1	2.00	-73.993835	40.756634	-73.986855	40.736626	13.30
4856493	1	0.80	-73.985283	40.763573	-73.982140	40.775116	7.80
4856494	1	2.16	-73.983391	40.730316	-74.004990	40.746513	18.96
4856495	1	1.00	-73.973137	40.782570	-73.980789	40.770084	7.80

In [5]:

```
jan_2015_02_groupby =
n[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cluster','pickup_bins']).coun
t()
jan_2015_01_groupby =
month[['pickup_cluster','pickup_bins','trip_distance']].groupby(['pickup_cluster','pickup_bins']).
count()
```

In [6]:

```
jan_2015_01_groupby.head() #for every cluster i there is 1698 time bins
```

Out[6]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**1**	58
	2	130

		trip_distance
	4	215
pickup_cluster	pickup_bins	
	5	215

Smoothing

In [7]:

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

# for each cluster region we will collect all the indices of 10min intravels in which the pickups
are happened
# we got an observation that there are some pickpbins that doesnt have any pickups
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,40):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [8]:

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

#jan
jan_2015_01_unique = return_unq_pickup_bins(month)
jan_2015_02_unique = return_unq_pickup_bins(n)
```

In [75]:

```
# for each cluster number of 10min intravels with 0 pickups
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",1698 -
len(set(jan_2015_02_unique[i])))
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  226
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  59
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  0
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  3
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  0
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  6
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  0
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  5
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  2
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  1
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  339
```

```
---------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  0
---------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  6
---------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  218
---------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  0
---------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  248
---------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  6
---------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  9
---------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  14
---------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  0
---------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  60
---------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  349
---------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  0
---------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  198
---------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  1061
---------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  44
---------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  1
---------------------------------------------------------
```

there are two ways to fill up these values

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

In [9]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
```

```
def fill_missing(count_values,values,size):
    smoothed_regions=[]
    ind=0
    for r in range(0,40):
        smoothed_bins=[]
        for i in range(size+1):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    print(len(smoothed_regions))
    print(ind)
    return smoothed_regions
```

In [10]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the
above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values,size):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(size+1):
            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,size+1):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin
value which has a pickup value
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing,hence w
have no right-limit here
                        smoothed_value=count_values[ind-1]*1.0/((size-i)+2)*1.0

                        for j in range(i,size+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(size-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_lim
t-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 here
```

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

In [11]:

```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pickups that are h
appened
jan_2015_02_fill =
fill_missing(jan_2015_02_groupby['trip_distance'].values,jan_2015_02_unique,1697)

#Smoothing Missing values of Jan-2015
jan_2015_02_smooth = smoothing(jan_2015_02_groupby['trip_distance'].values,jan_2015_02_unique,1697
)
```

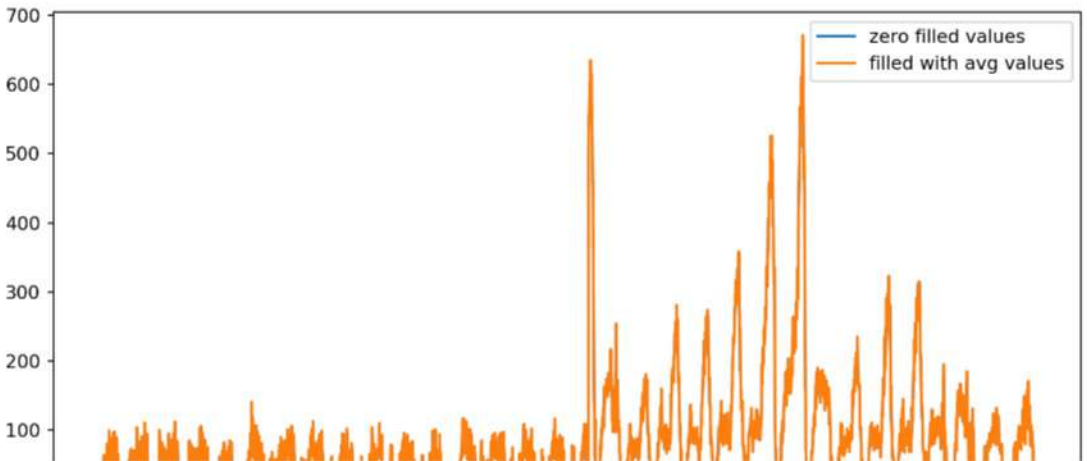
```
67920
65050
```

In [31]:

```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of pickups that are h
appened
jan_2015_01_fill =
fill_missing(jan_2015_01_groupby['trip_distance'].values,jan_2015_01_unique,1792)

#Smoothing Missing values of Jan-2015
jan_2015_01_smooth = smoothing(jan_2015_01_groupby['trip_distance'].values,jan_2015_01_unique,1792
)
```

In [85]:

```
# Smoothing vs Filling
# sample plot that shows two variations of filling missing values
# we have taken the number of pickups for cluster region 2
plt.figure(figsize=(10,5))
plt.plot(jan_2015_02_fill[4464:8920], label="zero filled values")
plt.plot(jan_2015_02_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```



In [0]:

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

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

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

In [12]:

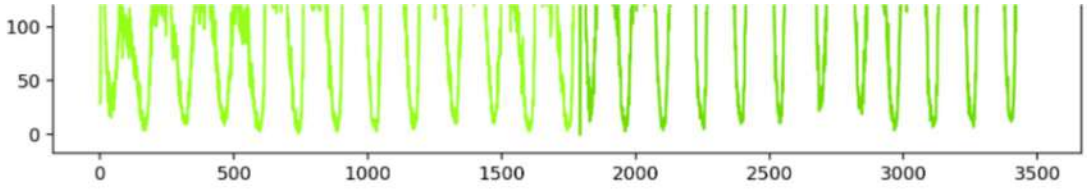
```
# jan -2015 data is smoothed while feb -2015 data is simply filled to avoid data leakage

jan_2015_01_smooth = smoothing(jan_2015_01_groupby['trip_distance'].values,jan_2015_01_unique,1792
)

jan_2015_02_fill =
fill_missing(jan_2015_02_groupby['trip_distance'].values,jan_2015_02_unique,1697)

regions_cum = []

for i in range(0,40):
    regions_cum.append(jan_2015_01_smooth[1793*i:1793*(i+1)]+jan_2015_02_fill[1698*i:1698*(i+1)])
```

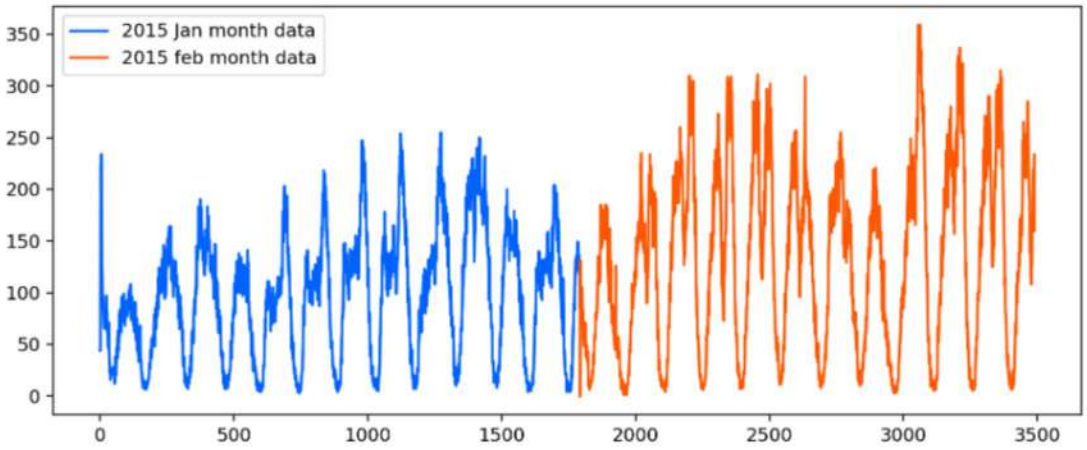
```
67920
65050
```

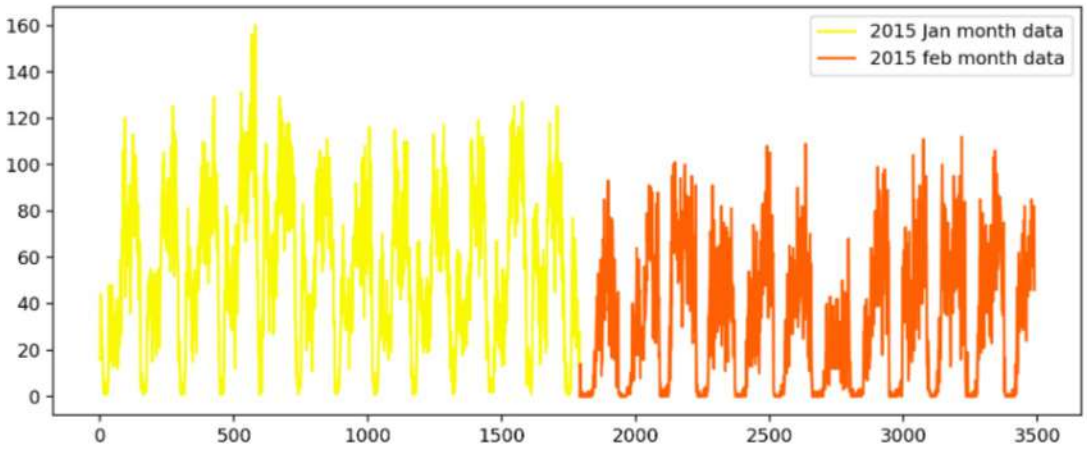
Time series and Fourier Transforms

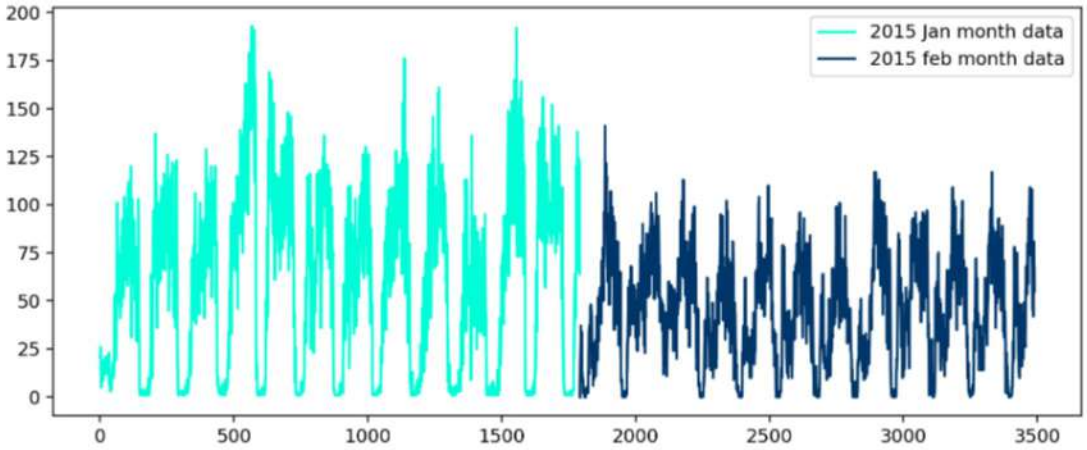
In [13]:

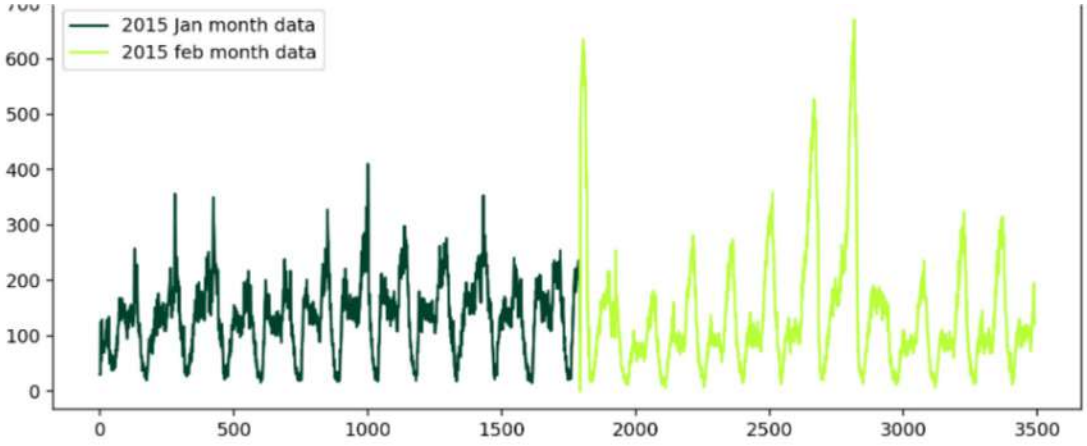
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,1793))
second_x = list(range(1793,3491))
for i in range(40):
    plt.figure(figsize=(10,4))
    plt.plot(first_x,regions_cum[i][:1793], color=uniqueish_color(), label='2015 Jan month data')
    plt.plot(second_x,regions_cum[i][1793:], color=uniqueish_color(), label='2015 feb month data')

    plt.legend()
    plt.show()
```

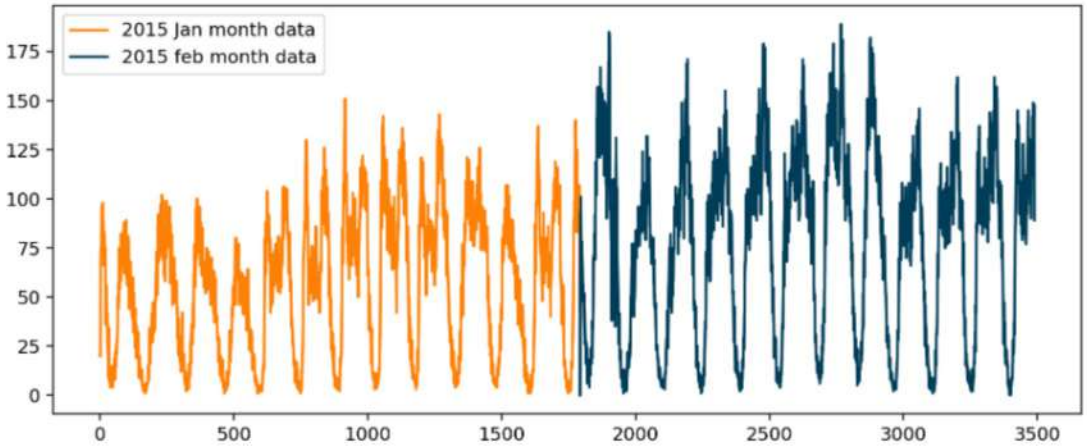

100
50
0
0
500
1000
1500
2000
2500
3000
3500

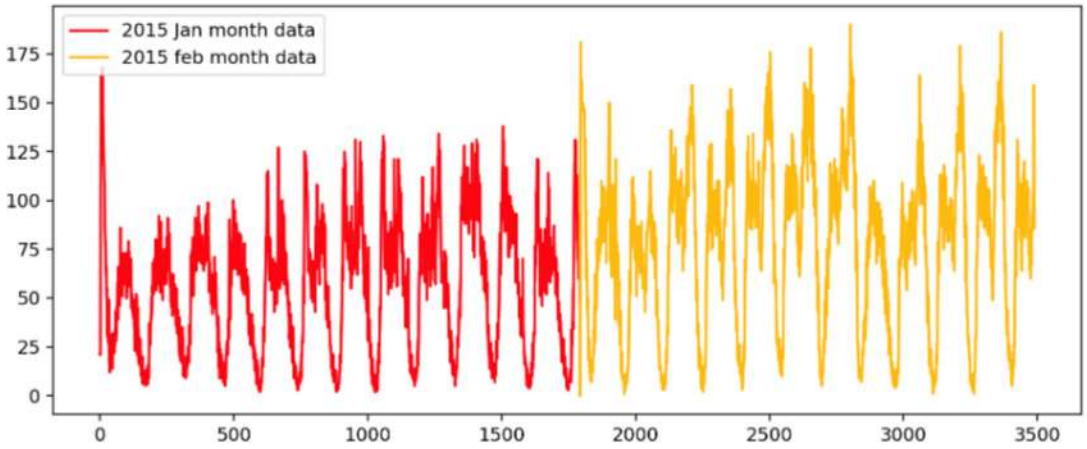

2015 Jan month data
2015 feb month data
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

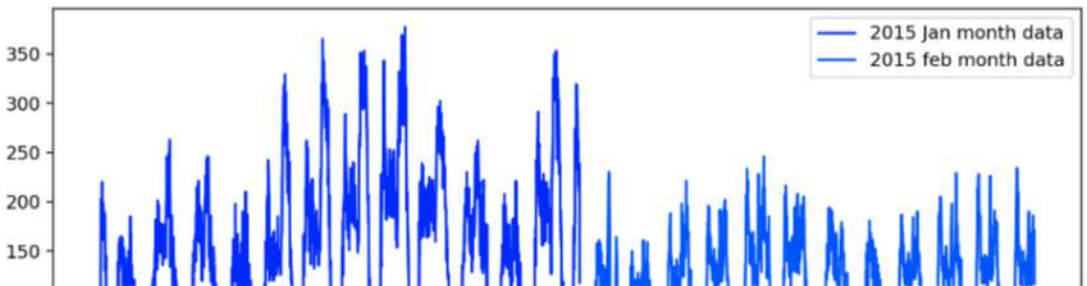

2015 Jan month data
2015 feb month data
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

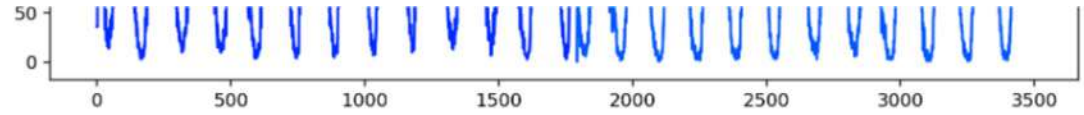

2015 Jan month data
2015 feb month data
200
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500

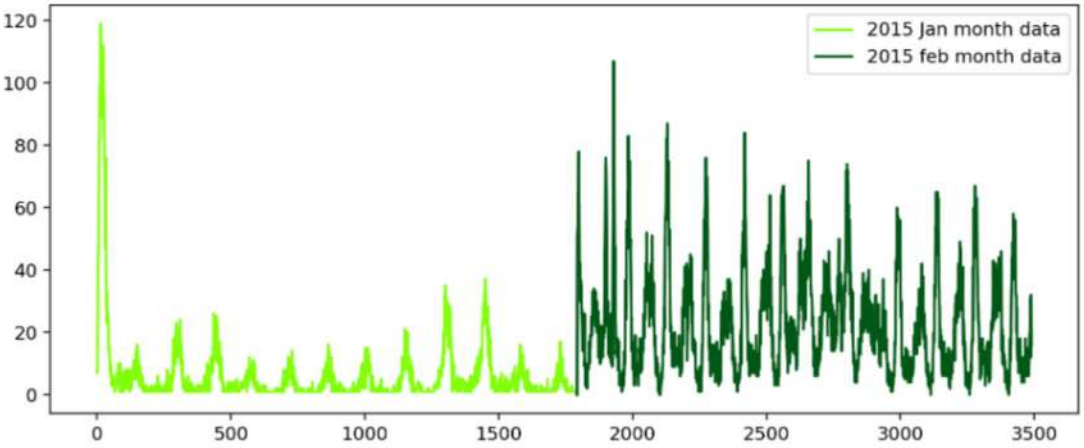

2015 Jan month data
2015 feb month data
0
100
200
300
400
500
600
0
500
1000
1500
2000
2500
3000
3500

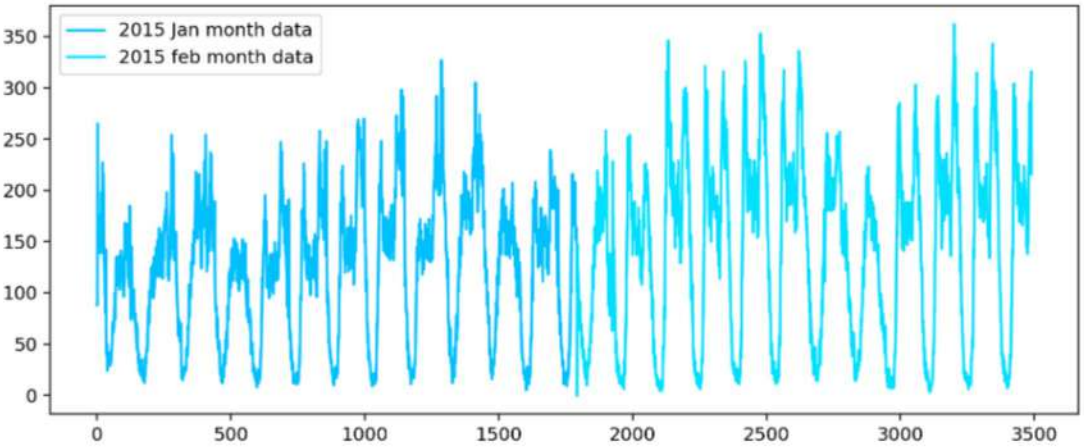

2015 Jan month data
2015 feb month data
0
25
50
75
100
125
150
175
0
500
1000
1500
2000
2500
3000
3500

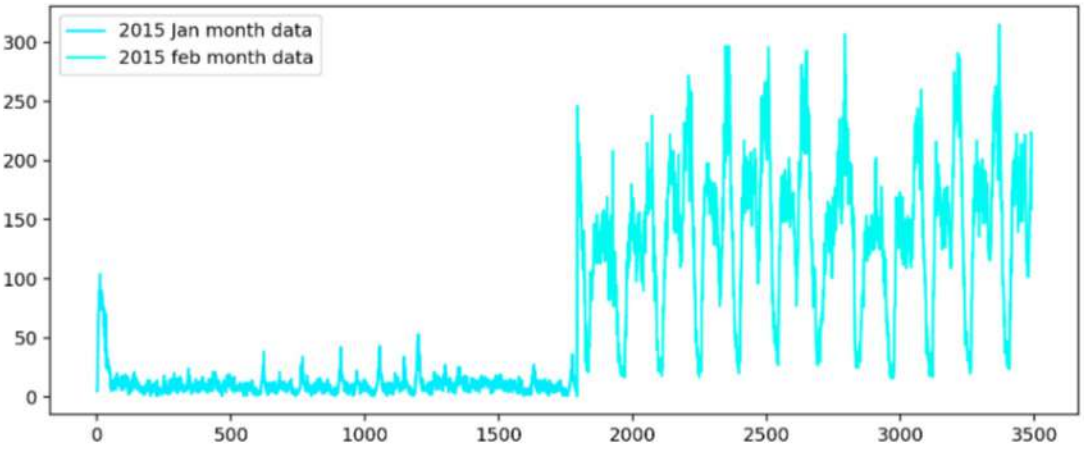

2015 Jan month data
2015 feb month data
0
25
50
75
100
125
150
175
0
500
1000
1500
2000
2500
3000
3500

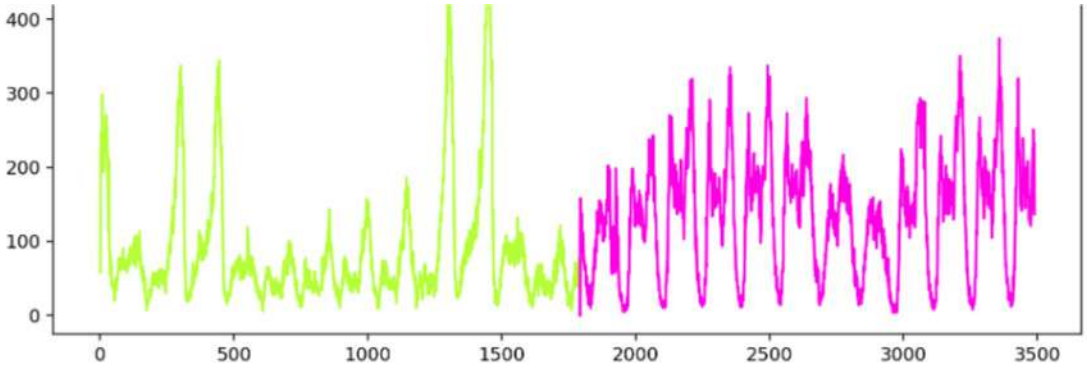

2015 Jan month data
2015 feb month data
150
200
250
300
350

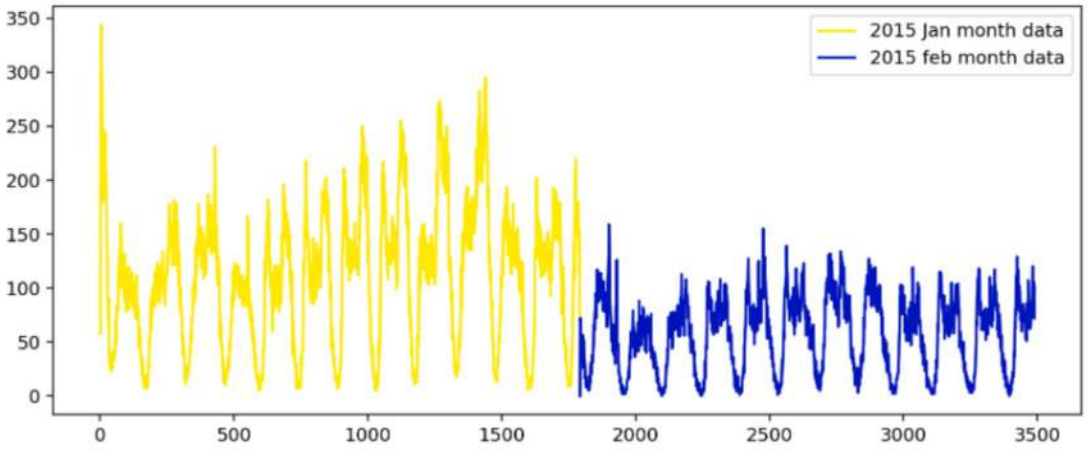

50
0
0
500
1000
1500
2000
2500
3000
3500

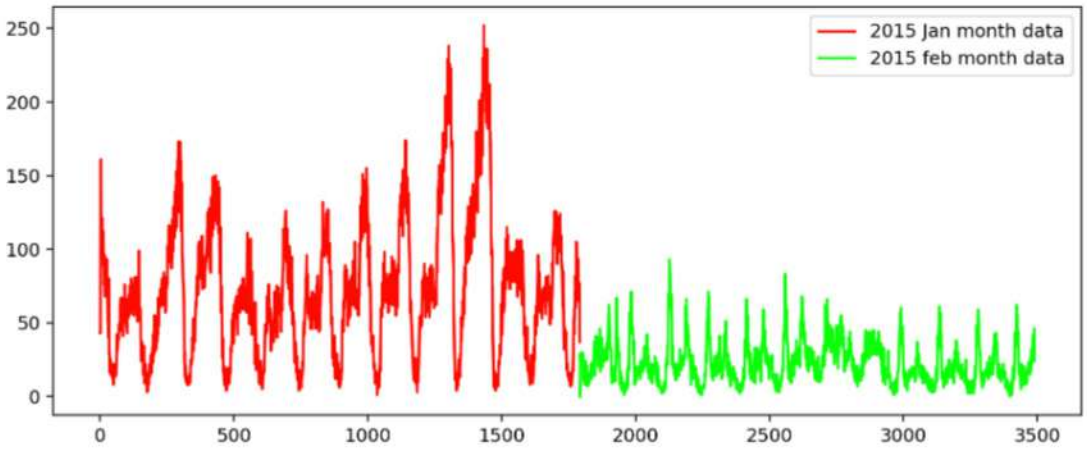

2015 Jan month data
2015 feb month data
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

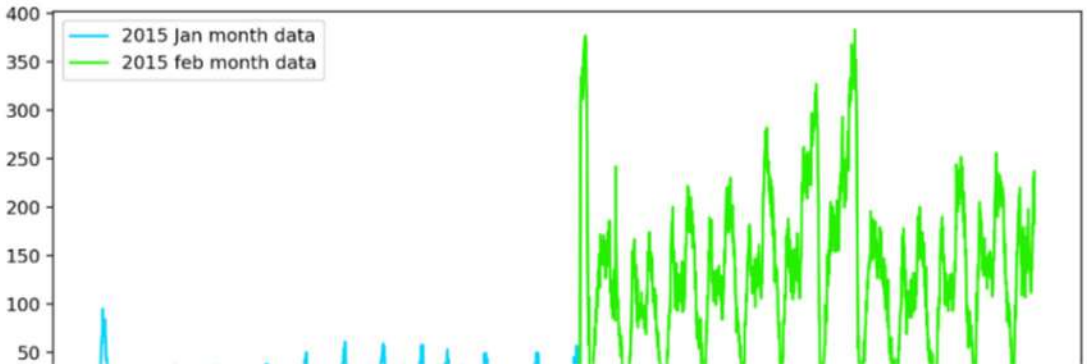

2015 Jan month data
2015 feb month data
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500


2015 Jan month data
2015 feb month data
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

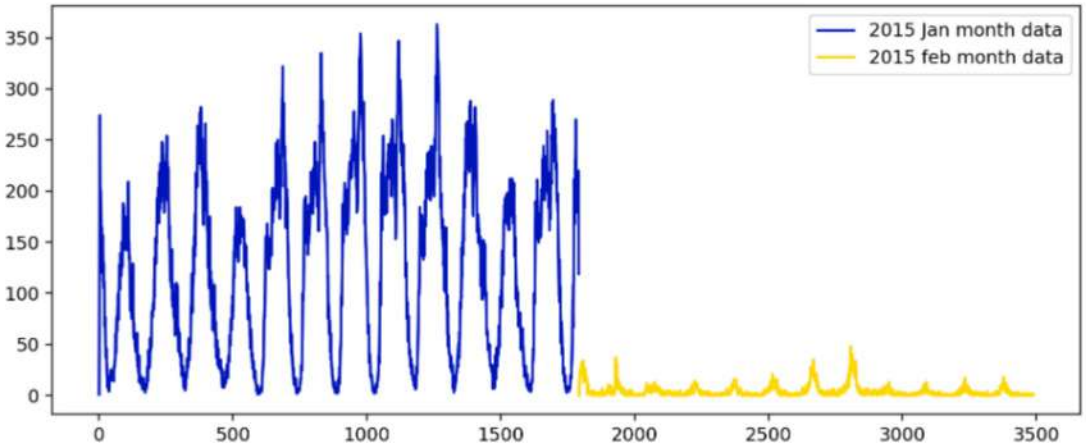

400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500

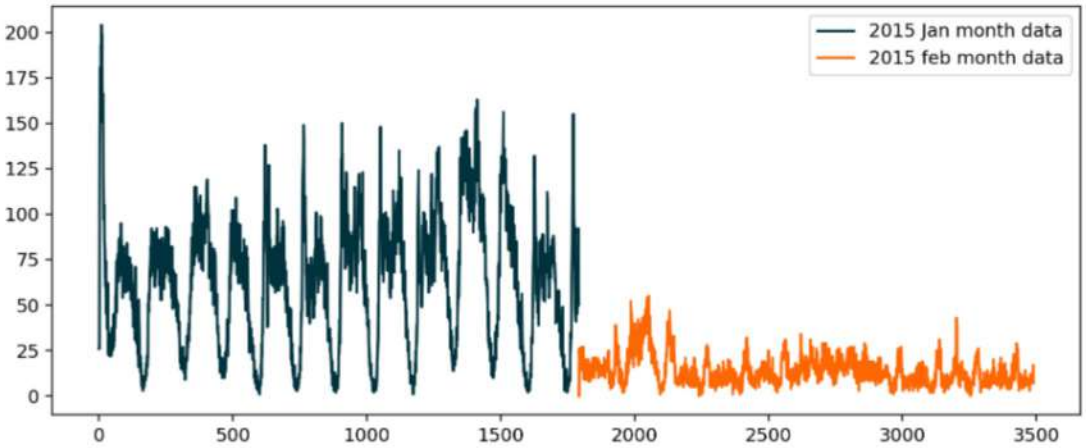

2015 Jan month data
2015 feb month data
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

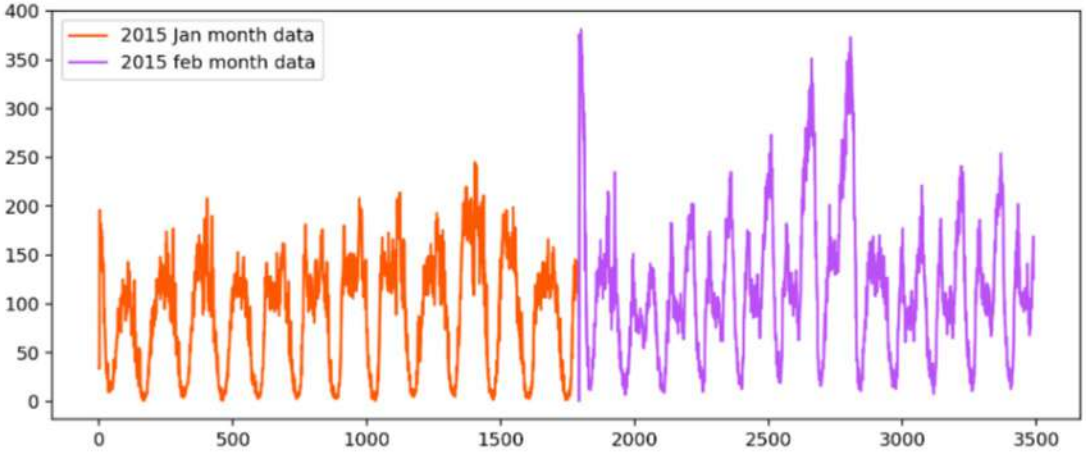

2015 Jan month data
2015 feb month data
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500


2015 Jan month data
2015 feb month data
400
350
300
250
200
150
100
50

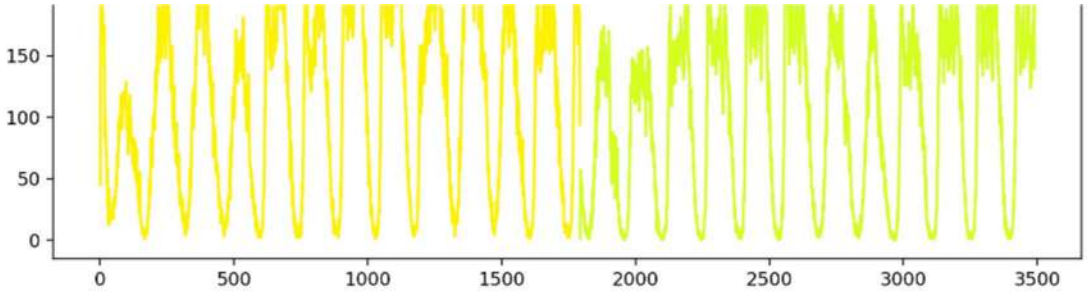

0
500
1000
1500
2000
2500
3000
3500

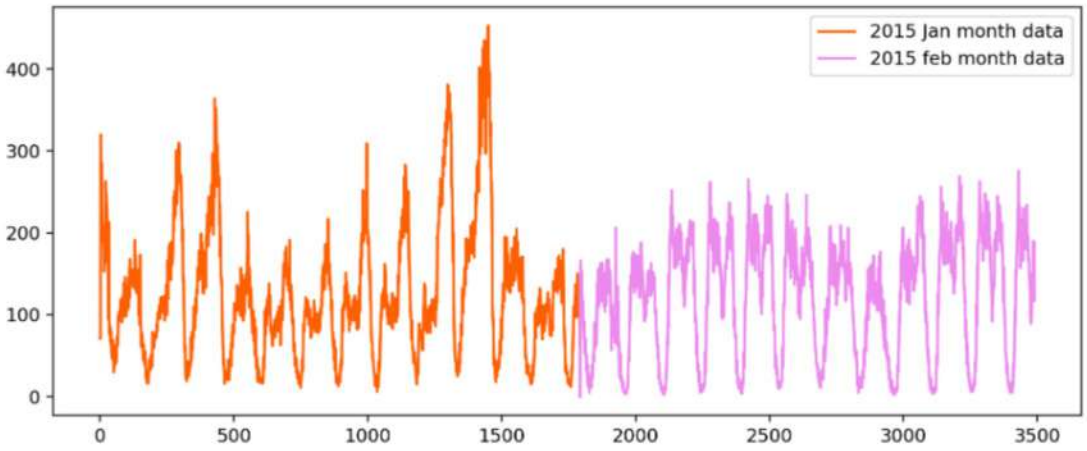

2015 Jan month data
2015 feb month data
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

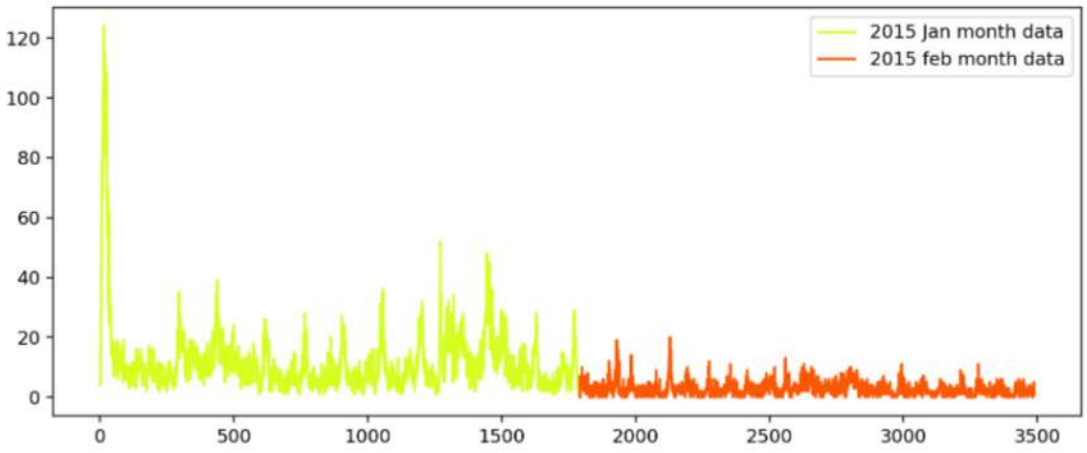

2015 Jan month data
2015 feb month data
200
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500

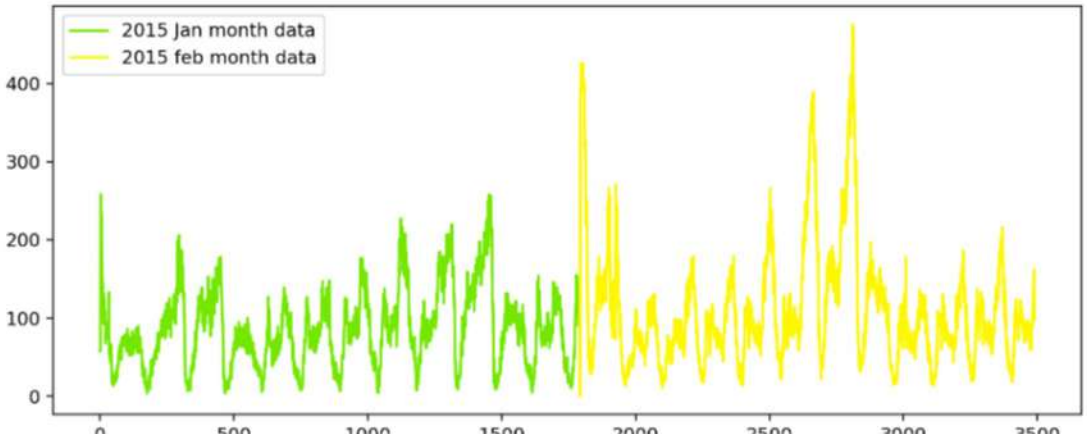

2015 Jan month data
2015 feb month data
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

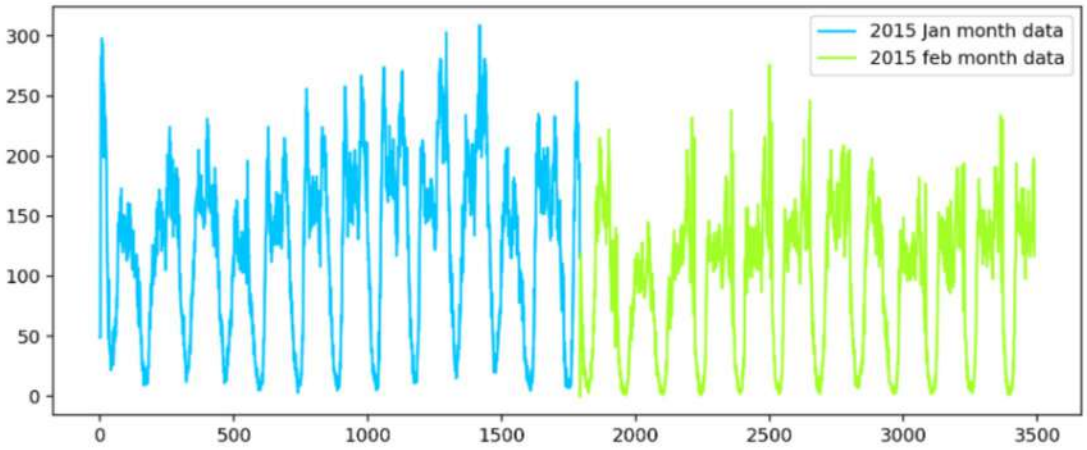

2015 Jan month data
2015 feb month data
300
250

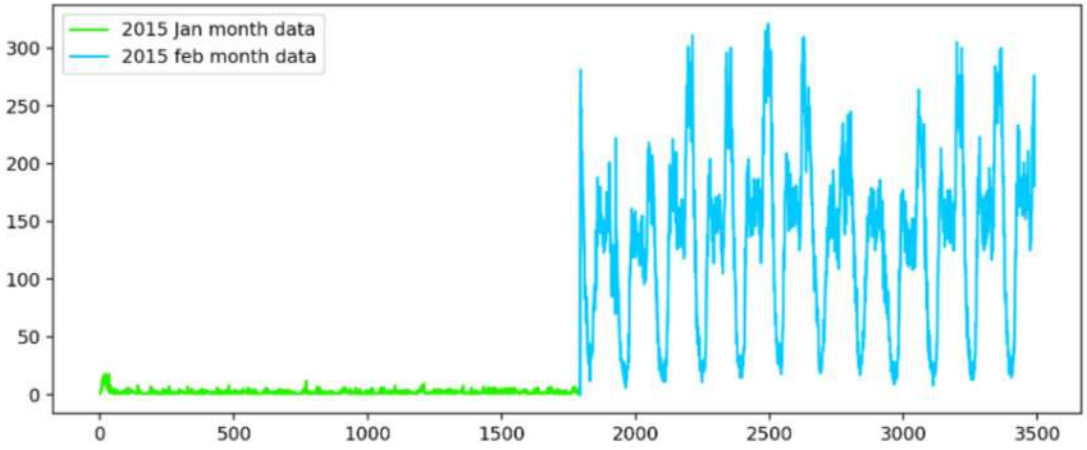

150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

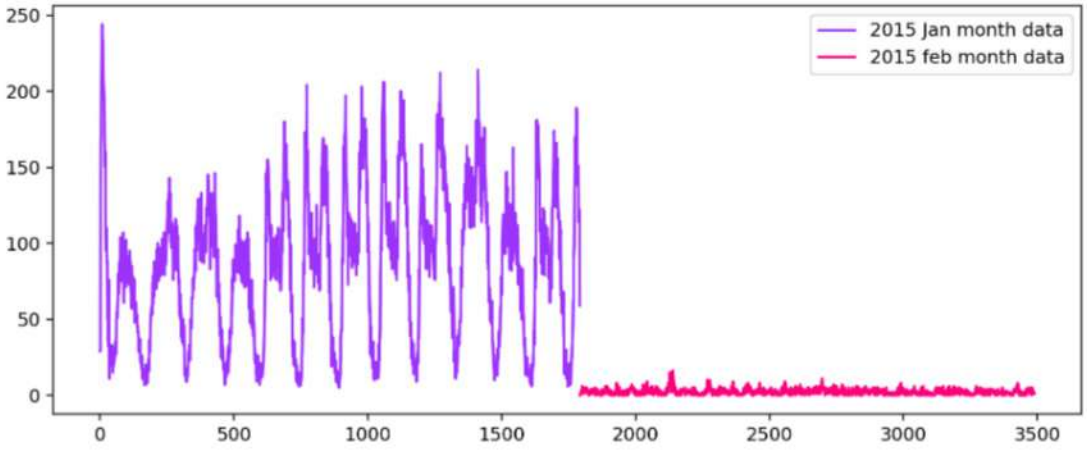

2015 Jan month data
2015 feb month data
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500

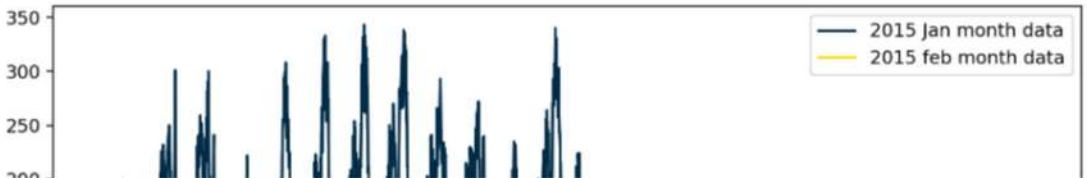

2015 Jan month data
2015 feb month data
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

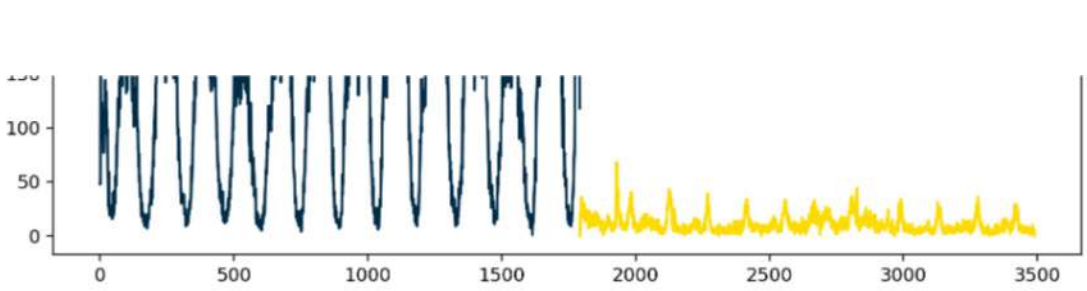

2015 Jan month data
2015 feb month data
400
300
200
100
0

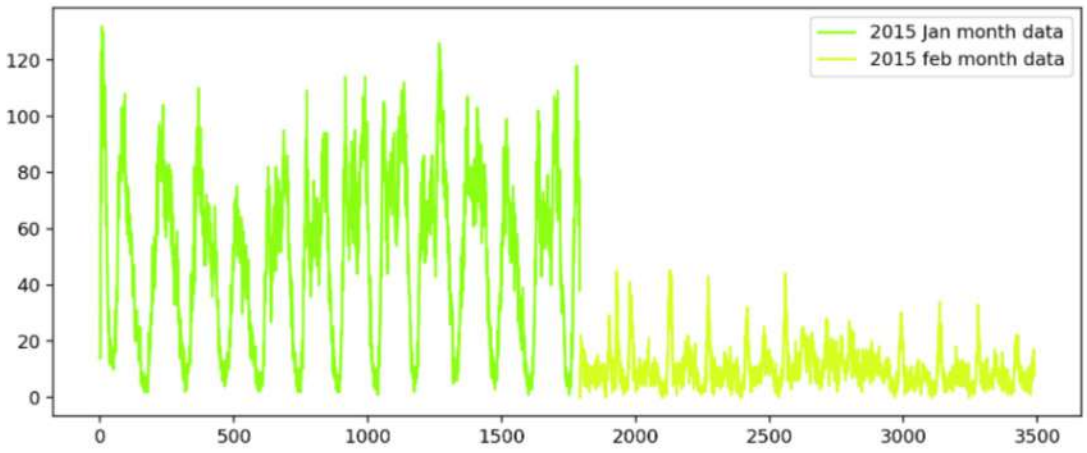

2015 Jan month data
2015 feb month data
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

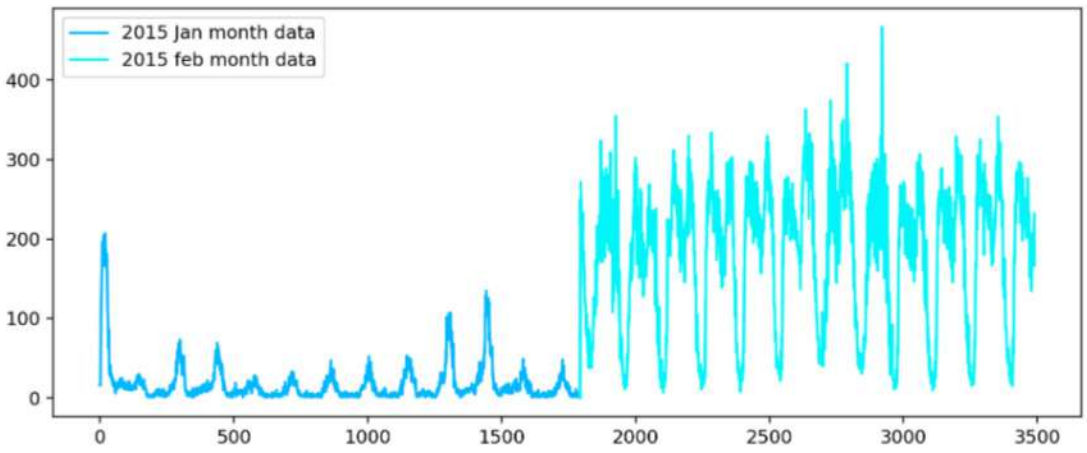

2015 Jan month data
2015 feb month data
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

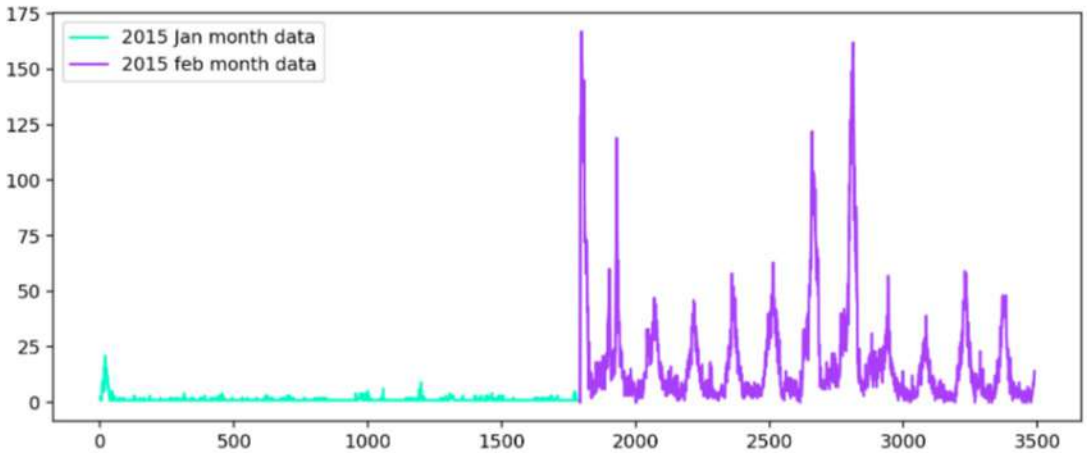

2015 Jan month data
2015 feb month data
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

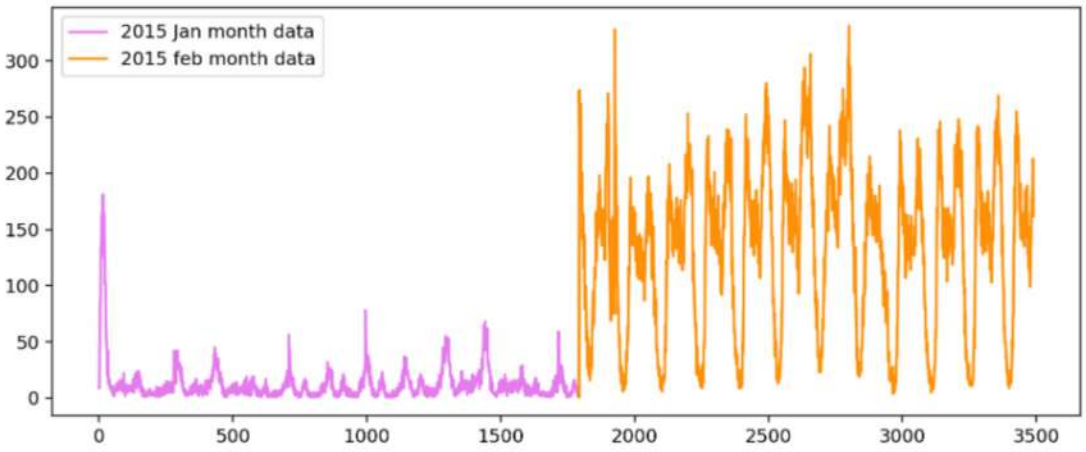

2015 Jan month data
2015 feb month data
350
300
250

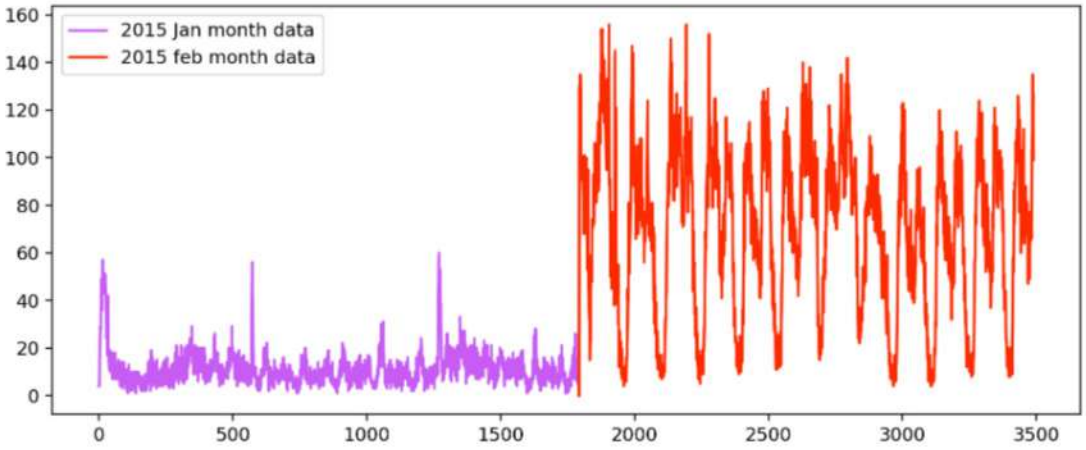

100
50
0
0
500
1000
1500
2000
2500
3000
3500

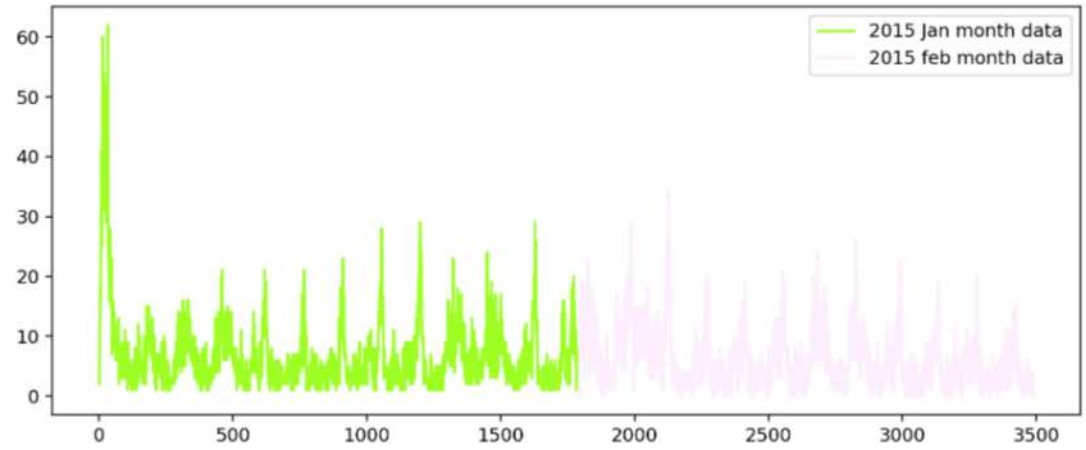

2015 Jan month data
2015 feb month data
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

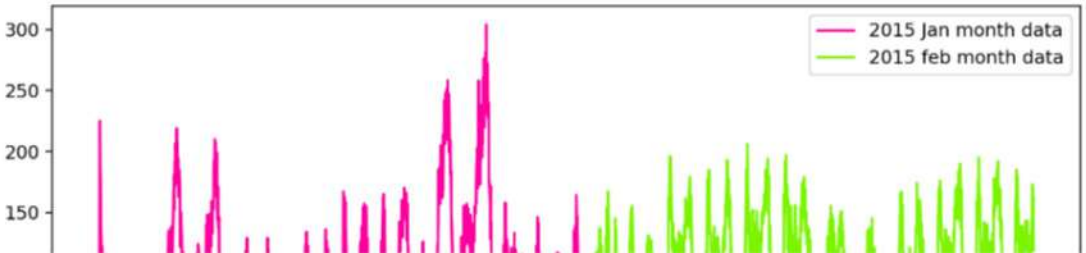

2015 Jan month data
2015 feb month data
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500

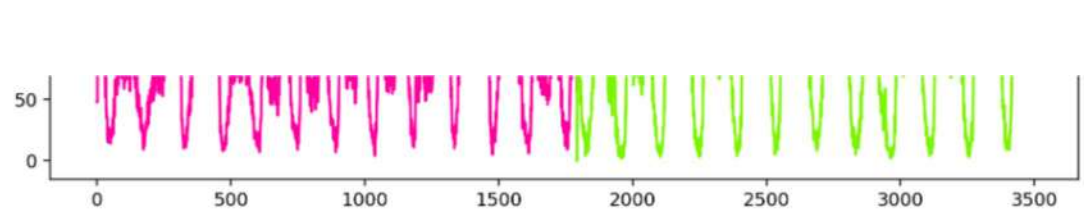

2015 Jan month data
2015 feb month data
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500

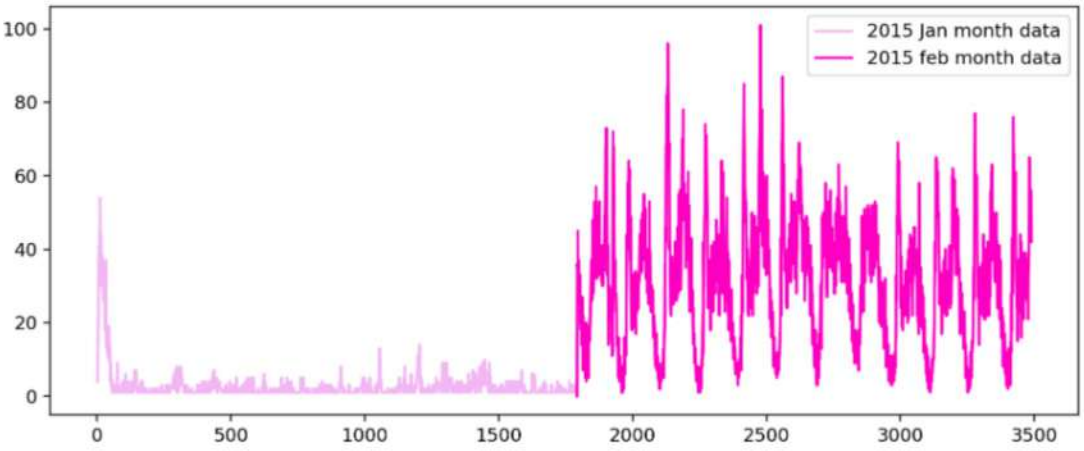

2015 Jan month data
2015 feb month data
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

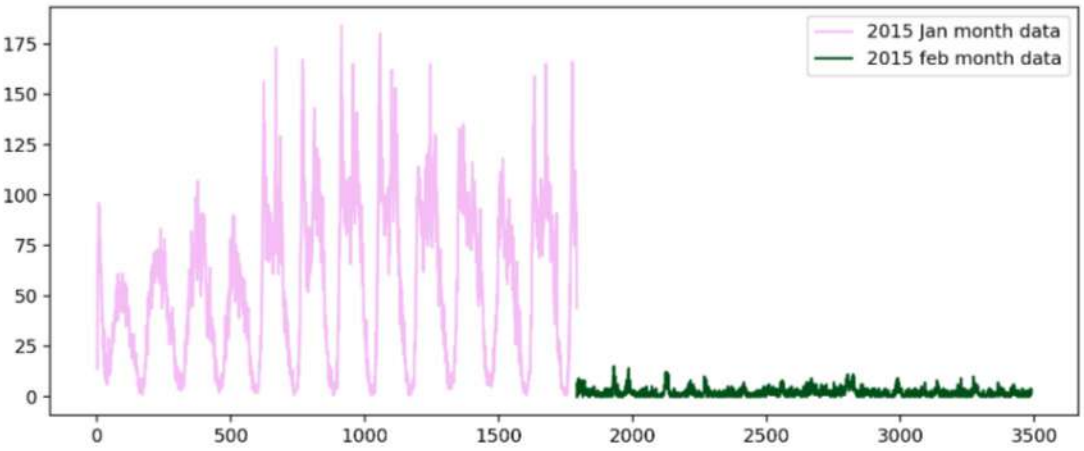

2015 Jan month data
2015 feb month data
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

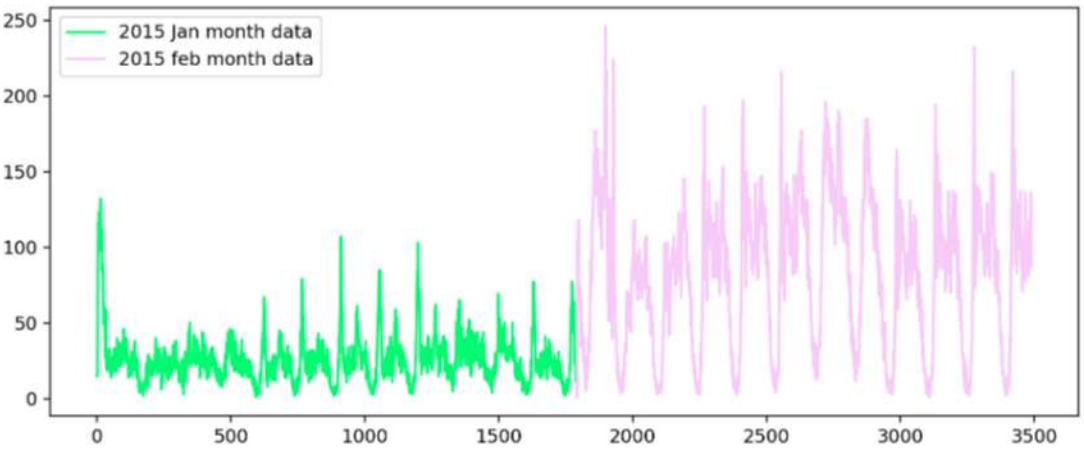

2015 Jan month data
2015 feb month data
60
50
40
30
20
10
0
0
500
1000
1500
2000
2500
3000
3500

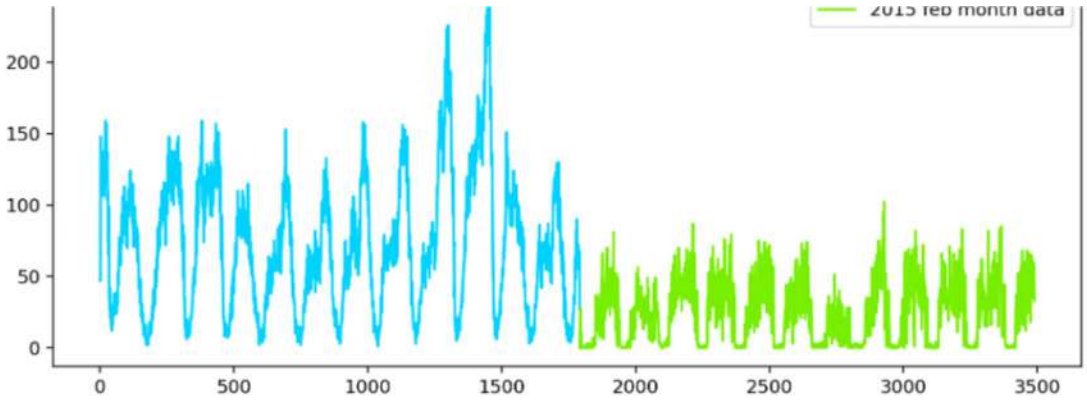

2015 Jan month data
2015 feb month data
300
250
200
150

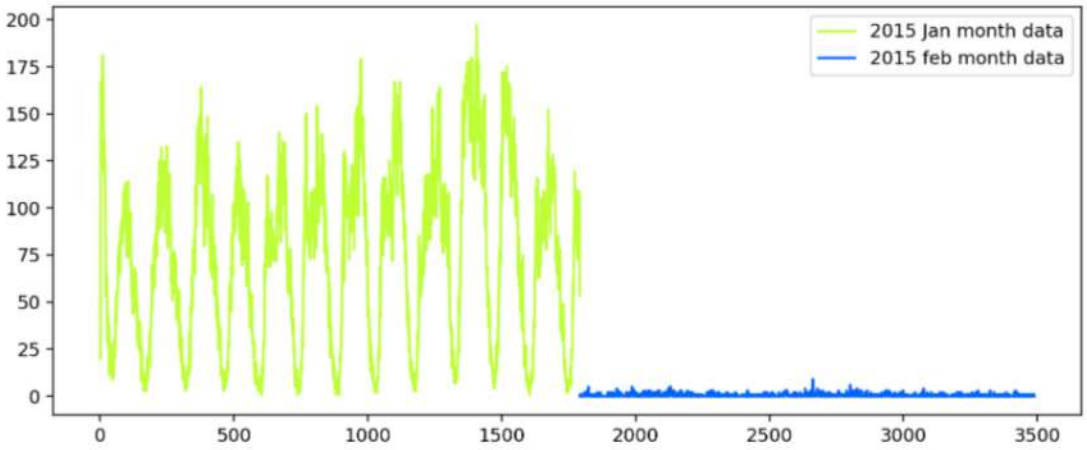

50
0
0
500
1000
1500
2000
2500
3000
3500

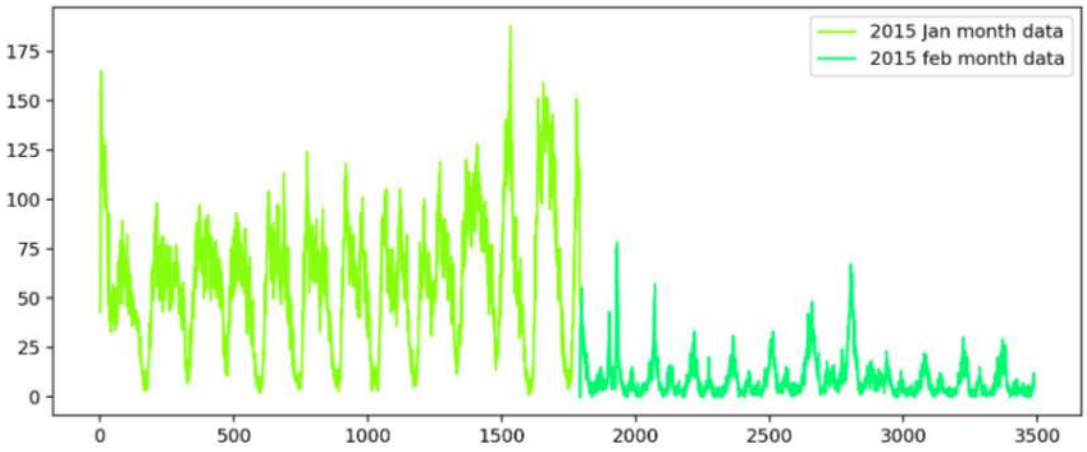

2015 Jan month data
2015 feb month data
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500

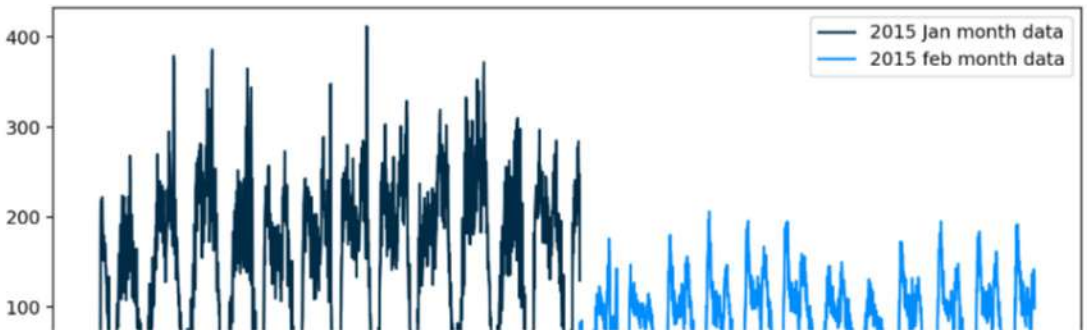

2015 Jan month data
2015 feb month data
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500


2015 Jan month data
2015 feb month data
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500


2015 feb month data
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500

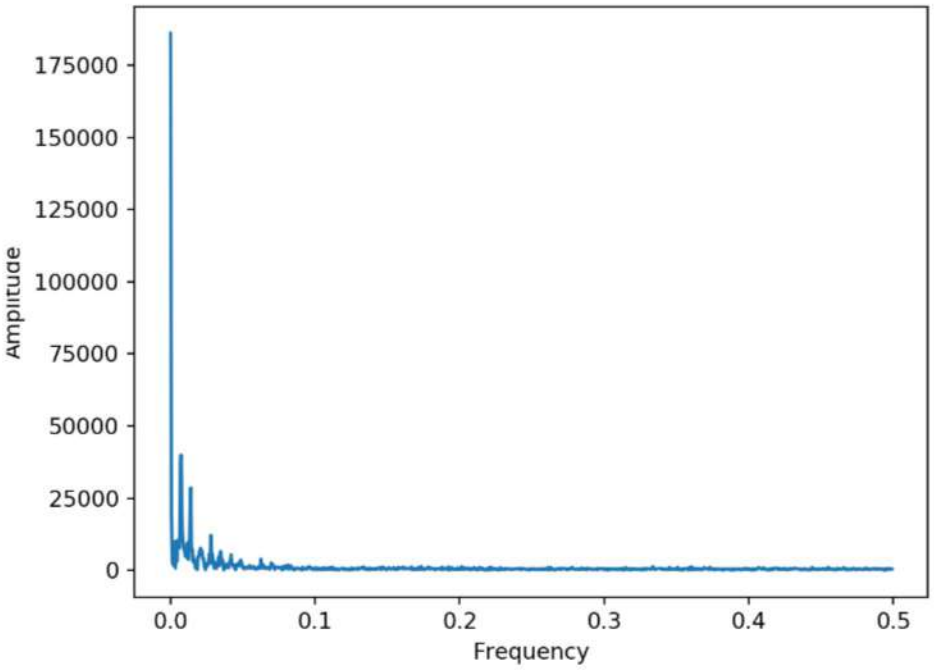

2015 Jan month data
2015 feb month data
200
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500


2015 Jan month data
2015 feb month data
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500


2015 Jan month data
2015 feb month data
400
300
200
100

In [114]:

```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function :
https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
Y    = np.fft.fft(np.array(jan_2015_01_smooth)[0:1793])
# read more about the fftfreq:
https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html
freq = np.fft.fftfreq(1793, 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 [297]:

```
#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_01_smooth[:67920]
ratios_jan['Prediction']=jan_2015_02_fill
ratios_jan['Ratios']=ratios_jan['Prediction']*1.0/ratios_jan['Given']*1.0
```

Modelling: Baseline Models

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

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

Simple Moving Averages

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

In [16]:

```
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(0,67920):
        if i%67920==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Pred
iction'].values)[i],1))))
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)])/window_size
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


    ratios['MA_R_Predicted'] = predicted_values
    ratios['MA_R_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].v
alues))
    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 [17]:

```
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(0,67920):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_size:
(i+1)])/window_size)
        else:
            predicted_value=int(sum((ratios['Prediction'].values)[0:(i+1)])/(i+1))

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

to the subsequent older ones

Weighted Moving Averages using Ratio Values - $R_t = (N * R_{t-1} + (N-1) * R_{t-2} + (N-2) * R_{t-3} .... 1 * R_{t-n})/(N * (N+1)/2)$

In [18]:

```
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(0,67920):
        if i%67920==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Pred
iction'].values)[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

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

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

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

In [19]:

```
def WA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,67920):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Prediction'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

        else:
            sum_values=0
```

```
                sum_values += j*(ratios['Prediction'].values)[j-1]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

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

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

Exponential Weighted Moving Averages

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

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

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

In [21]:

```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.1
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,67920):
        if i%67920==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Pred
iction'].values)[i],1))))
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].values)[i])

    ratios['EA_R1_Predicted'] = predicted_values
    ratios['EA_R1_Error'] = error
    mape_err = (sum(error)/len(error))/(sum(ratios['Prediction'].values)/len(ratios['Prediction'].v
alues))
    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 [24]:

```
def EA_P1_Predictions(ratios,month):
    predicted_value= (ratios['Prediction'].values)[0]
    alpha=0.31
    error=[]
```

```
        if i%40920==0:
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Prediction'].values)[i])
)

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

In [25]:

```
mean_err=[0]*10
median_err=[0]*10

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

Comparison between baseline models

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

In [26]:

```
print ("Error Metric Matrix (Forecasting Methods) - MAPE & MSE")
print ("----------------------------------------------------------------------------------------
----------")
print ("Moving Averages (Ratios) -                              MAPE: ",mean_err[0],"      MSE: ",me
ian_err[0])
print ("Moving Averages (2015 Values) -                         MAPE: ",mean_err[1],"       MSE: ",m
dian_err[1])
print ("----------------------------------------------------------------------------------------
----------")
print ("Weighted Moving Averages (Ratios) -                     MAPE: ",mean_err[2],"      MSE: ",me
dian_err[2])
print ("Weighted Moving Averages (2015 Values) -                MAPE: ",mean_err[3],"      MSE: ",me
dian_err[3])
print ("----------------------------------------------------------------------------------------
----------")
print ("Exponential Moving Averages (Ratios) -              MAPE: ",mean_err[4],"      MSE: ",media
n_err[4])
print ("Exponential Moving Averages (2015 Values) -         MAPE: ",mean_err[5],"      MSE: ",media
n_err[5])
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
----------------------------------------------------------------------------------------------
-
Moving Averages (Ratios) -                              MAPE:  0.3258660153328656       MSE:  12039.
55108951708
Moving Averages (2015 Values) -                         MAPE:  0.1312565685218314        MSE:  230.2
25883392226
----------------------------------------------------------------------------------------------
-
Weighted Moving Averages (Ratios) -                     MAPE:  0.3353123321835332       MSE:
15031.759467020023
Weighted Moving Averages (2015 Values) -                MAPE:  0.12661247945020443       MSE:
222.04268256772673
```

```
6671.924734982332
Exponential Moving Averages (2015 Values) -        MAPE:  0.12666931054818123       MSE:  220.358€
4252061248
```

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

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

Regression Models

Train-Test Split

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

In [31]:

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

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

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

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


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


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

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

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

```
tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*(1793-5))
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*(1793-5))
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of pickups that ar
e happened in last 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,1793)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104],
[x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for r in ran
ge(0,len(regions_cum[i])-number_of_time_stamps)]))
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [32]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_weekday[0]) =
= 40*(1788) == len(output)*len(output[0])
```

Out[32]:

```
True
```

In [33]:

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

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

# from the baseline models we said the exponential weighted moving avarage gives us the best error
# we will try to add the same exponential weighted moving avarage at t as a feature to our data
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.35

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

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5
,x6,x7..x13104], [x5,x6,x7..x13104], .. 40 lsits]
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    for i in range(0,1793):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

In [36]:

```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 months of 2016 pick
up data
# and split it such that for every region we have 70% data in train and 30% in test,
```

```
print("size of test data :", int((1793)*0.3))
```

```
size of train data : 1255
size of test data : 537
```

In [37]:

```
train_features =  [tsne_feature[i*1788:(1788*i+1255)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(1788*i)+1255:(1788*(i+1))] for i in range(0,40)]
```

In [38]:

```
print("Number of data clusters",len(train_features), "Number of data points in trian data",
len(train_features[0]), "Each data point contains", len(train_features[0][0]),"features")
print("Number of data clusters",len(train_features), "Number of data points in test data",
len(test_features[0]), "Each data point contains", len(test_features[0][0]),"features")
```

```
Number of data clusters 40 Number of data points in trian data 1255 Each data point contains 5 fea
tures
Number of data clusters 40 Number of data points in test data 533 Each data point contains 5 featu
res
```

In [39]:

```
tsne_train_flat_lat = [i[:1255] for i in tsne_lat]
tsne_train_flat_lon = [i[:1255] for i in tsne_lon]
tsne_train_flat_weekday = [i[:1255] for i in tsne_weekday]
tsne_train_flat_output = [i[:1255] for i in output]
tsne_train_flat_exp_avg = [i[:1255] for i in predict_list]
```

In [40]:

```
tsne_test_flat_lat = [i[1255:] for i in tsne_lat]
tsne_test_flat_lon = [i[1255:] for i in tsne_lon]
tsne_test_flat_weekday = [i[1255:] for i in tsne_weekday]
tsne_test_flat_output = [i[1255:] for i in output]
tsne_test_flat_exp_avg = [i[1255:] for i in predict_list]
```

In [41]:

```
# the above contains values in the form of list of lists (i.e. list of values of each region), her
e we make all of them in one list
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 [354]:

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

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

In [355]:

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

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

In [356]:

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

In [357]:

```
df_train['cluster'] = kmeans.predict(df_train[['lat', 'lon']])
df_train.drop(['lat','lon'], axis=1, inplace=True)
```

In [358]:

```
df_train.head()
```

Out[358]:

	ft_5	ft_4	ft_3	ft_2	ft_1	weekday	exp_avg	cluster
0	29	29	130	155	215	4	191	0
1	29	130	155	215	215	4	207	0
2	130	155	215	215	197	4	199	0
3	155	215	215	197	200	4	199	0
4	215	215	197	200	235	4	224	0

In [359]:

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

In [360]:

```
df_test['cluster'] = kmeans.predict(df_test[['lat', 'lon']])
df_test.drop(['lat','lon'], axis=1, inplace=True)
```

In [361]:

```
df_test.head()
```

Out[361]:

	ft_5	ft_4	ft_3	ft_2	ft_1	weekday	exp_avg	cluster
0	131	190	164	188	203	5	196	0
1	190	164	188	203	203	5	200	0
2	164	188	203	203	214	5	209	0
3	188	203	203	214	207	5	207	0

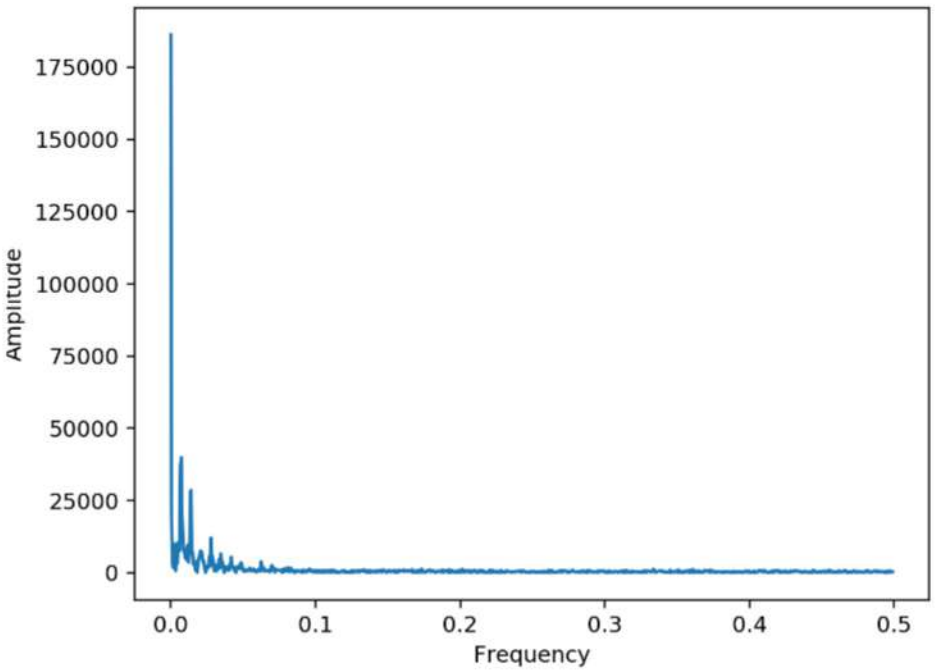
Incorporating Fourier features

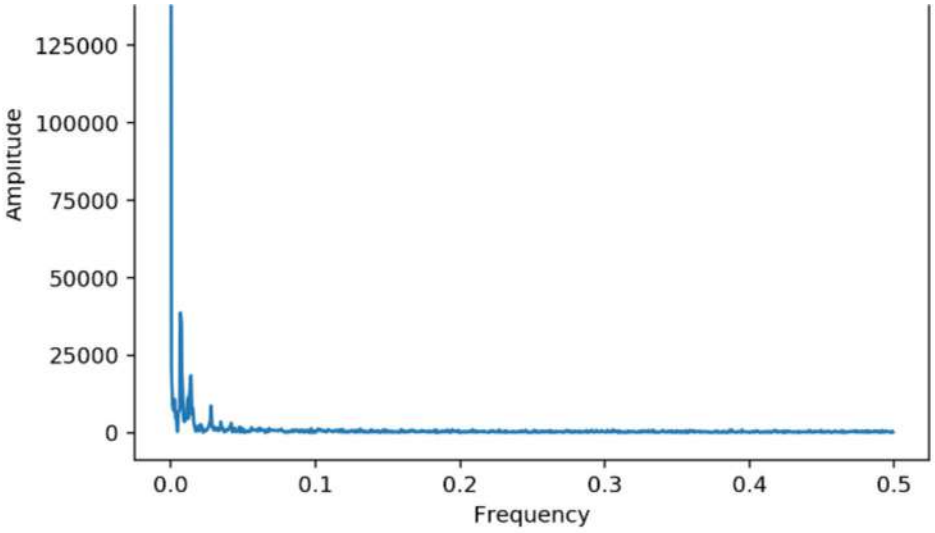
In [110]:

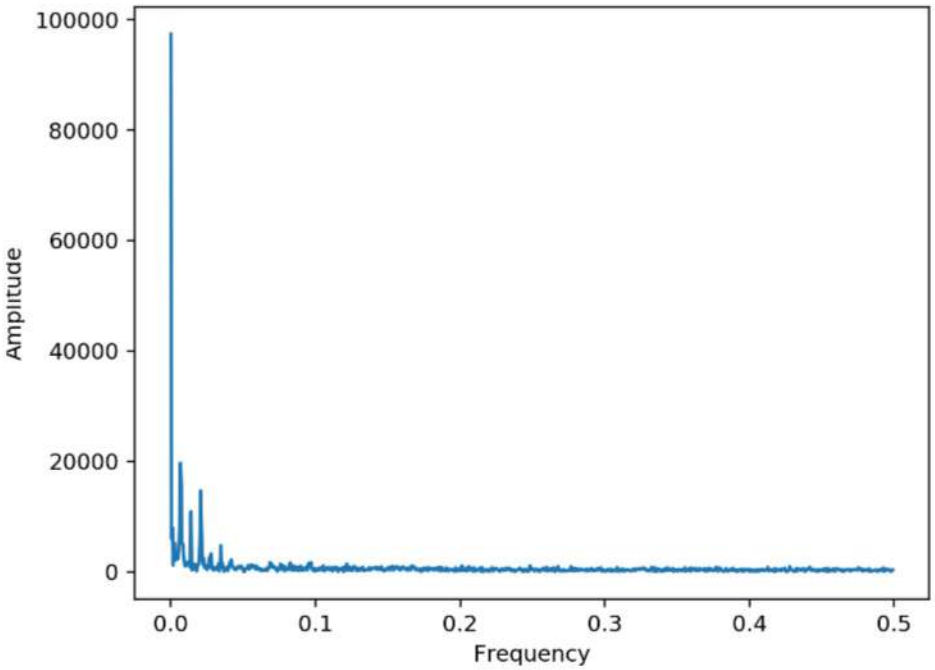
```
top_amp=[]
top_freq=[]
for i in range(40):
    Y    = np.fft.fft(np.array(jan_2015_01_smooth[1793*i:1793*(i+1)]))
    freq = np.fft.fftfreq(1793, 1)
    n = len(freq)
    pos_freq=freq[:int(n/2)+1]
    pos_amp=(np.abs(Y)[:int(n/2)+1])
    pos_amp = [math.log(pos_amp[i]) for i in range(0,897)]
    df = pd.DataFrame()
    df['freq']=pos_freq
    df['amp']=pos_amp
    df.sort_values(by='amp', ascending=False, inplace=True)
    top_amp.append(np.mean(df['amp'][1:6]))
    top_freq.append(np.mean(df['freq'][1:6]))
```

In [304]:

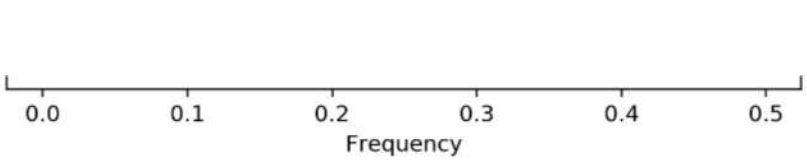
```
for i in range(40):
    Y    = np.fft.fft(np.array(jan_2015_01_smooth[1793*i:1793*(i+1)]))
    freq = np.fft.fftfreq(1793, 1)
    n = len(freq)
    plt.figure()
    plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    plt.show()
```

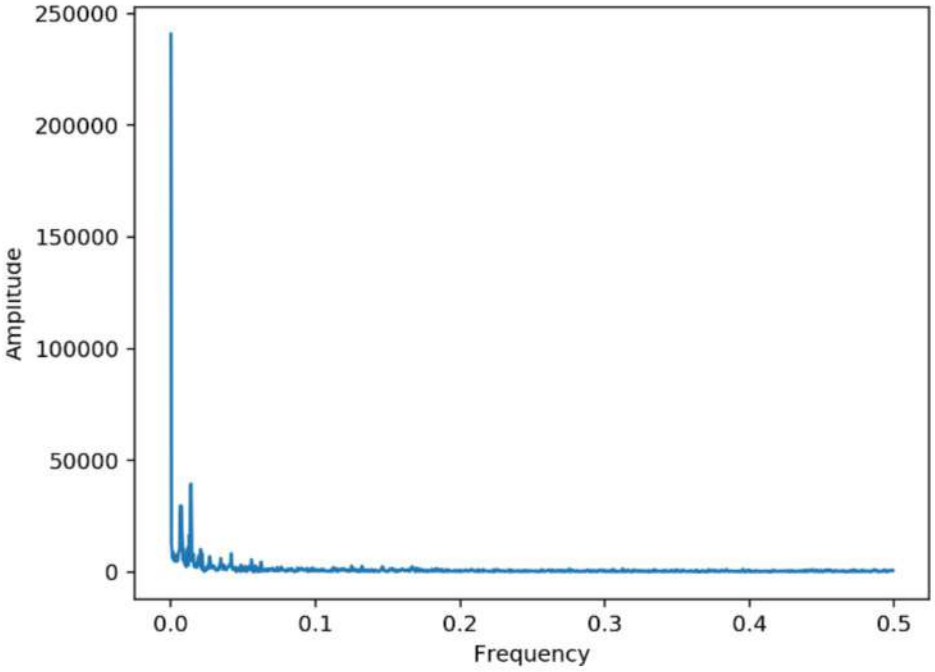


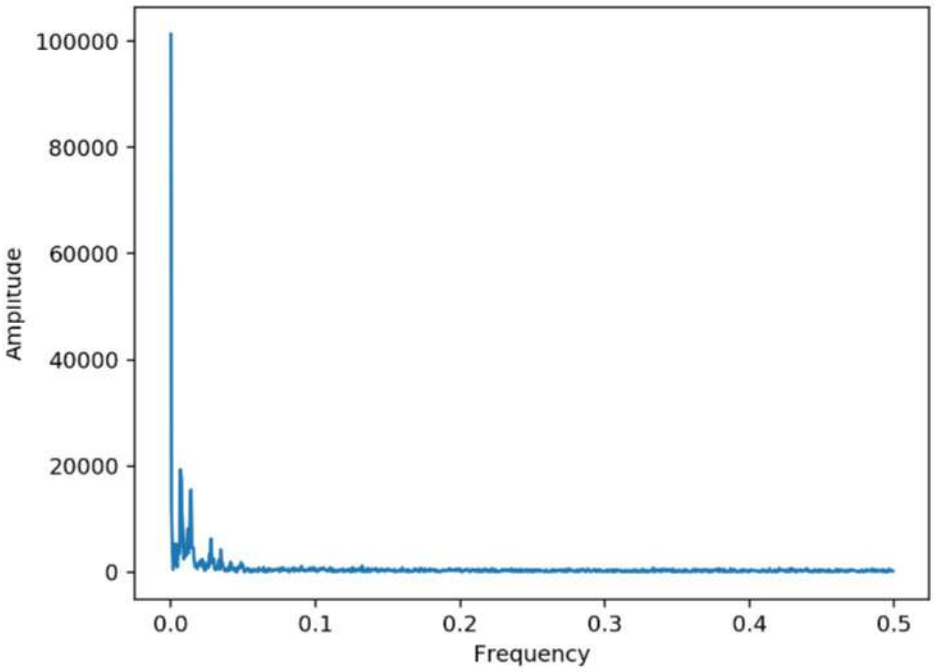

125000
100000
75000
50000
25000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

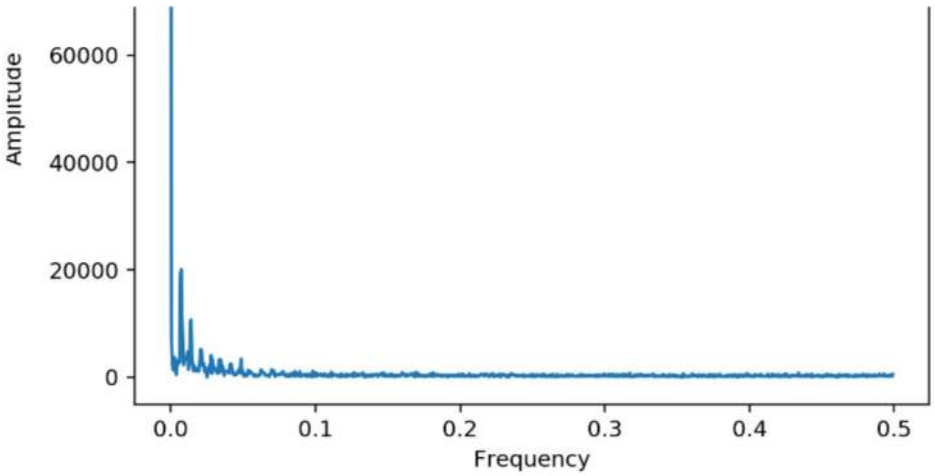

100000
80000
60000
40000
20000
Amplitude

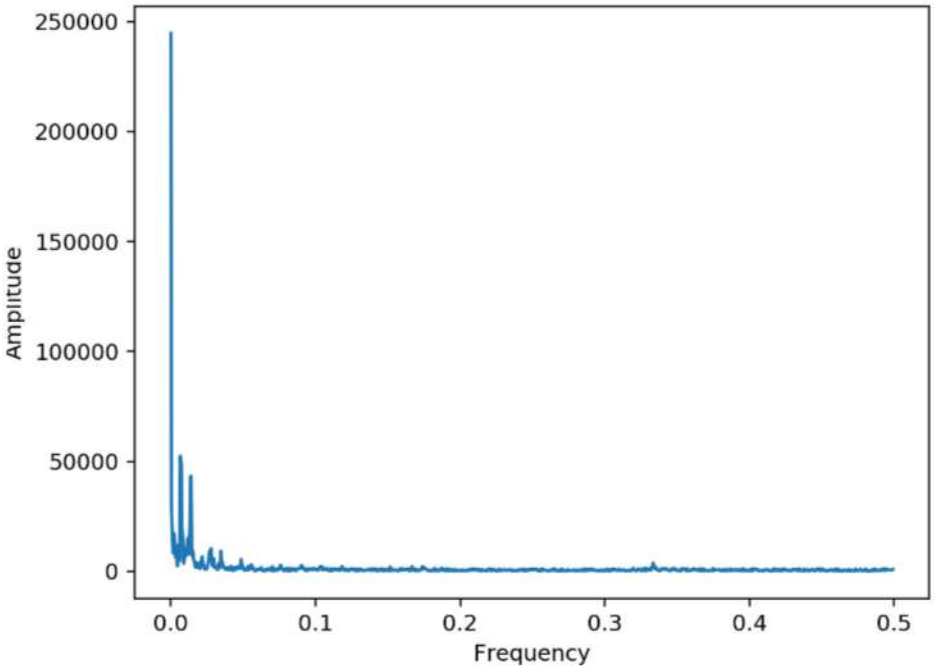

0.0
0.1
0.2
0.3
0.4
0.5
Frequency

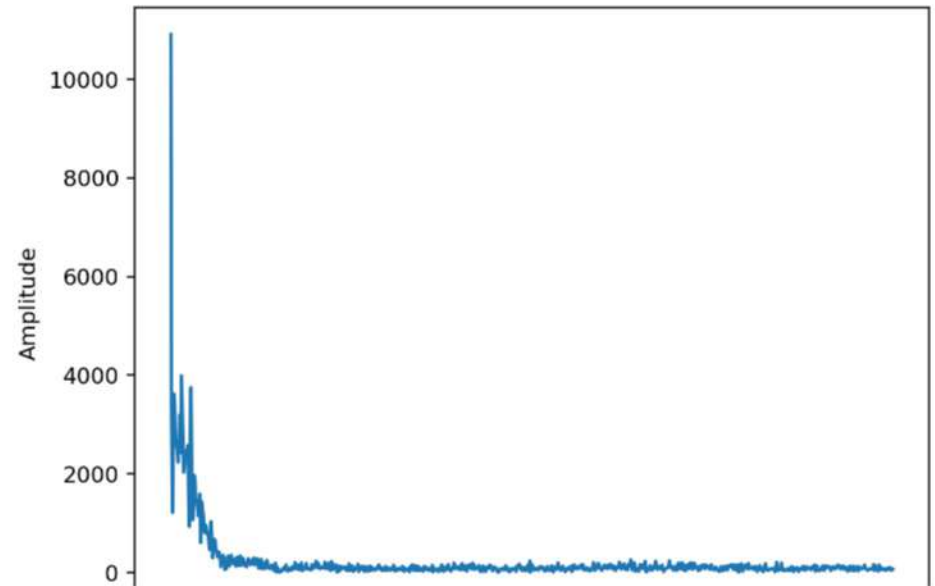

250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

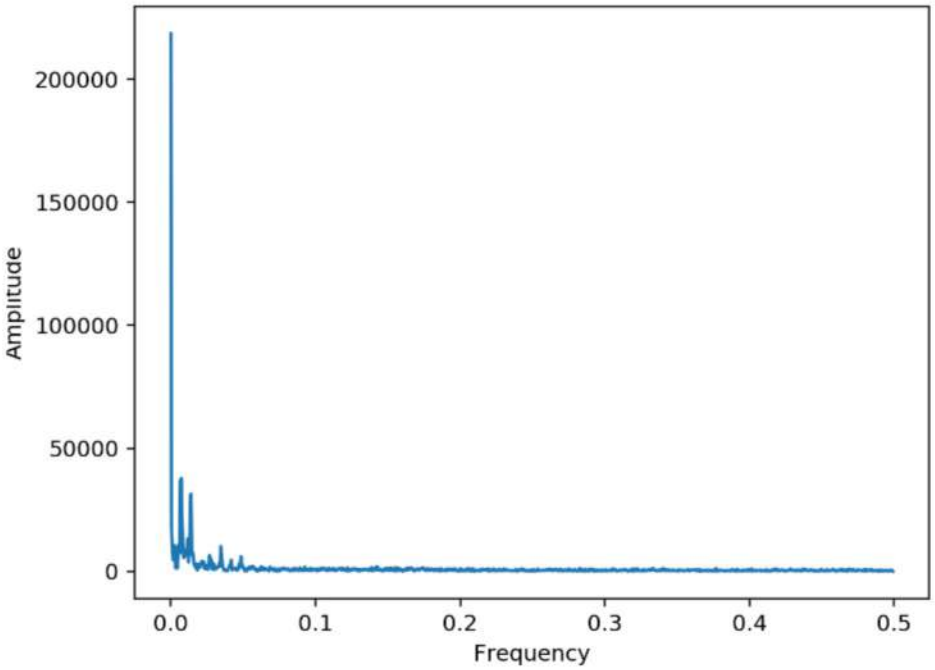

100000

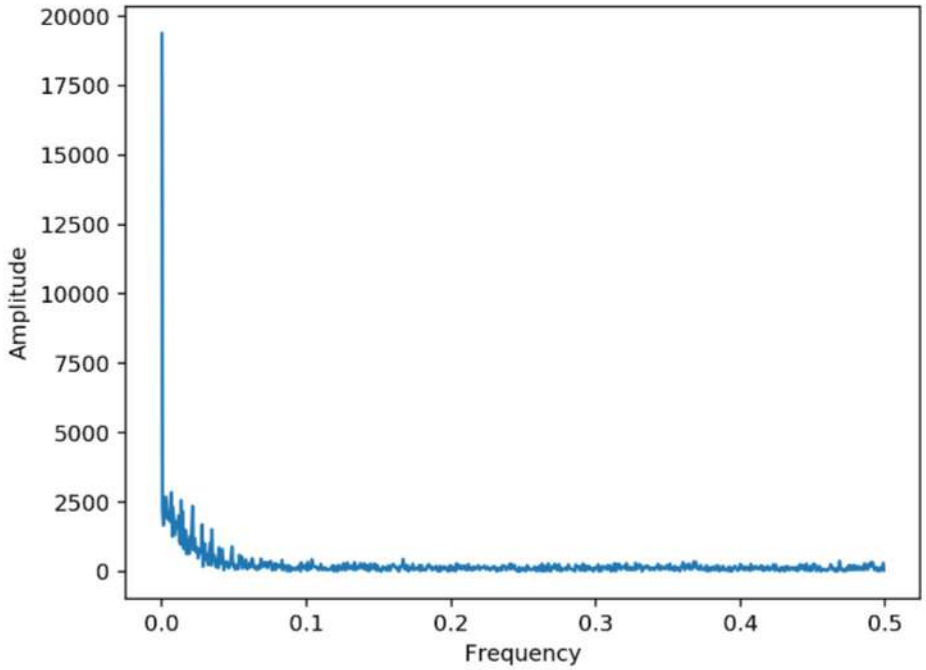

Amplitude
60000
40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Amplitude
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

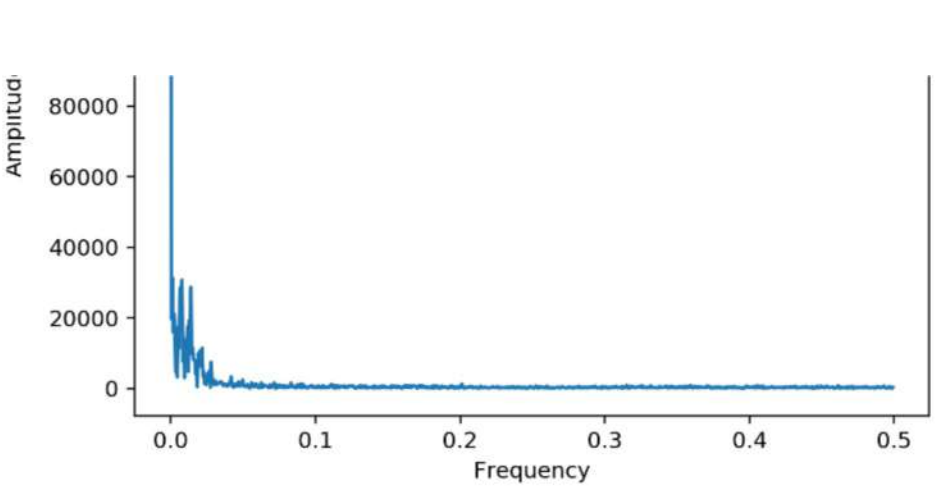

Amplitude
10000
8000
6000
4000
2000
0

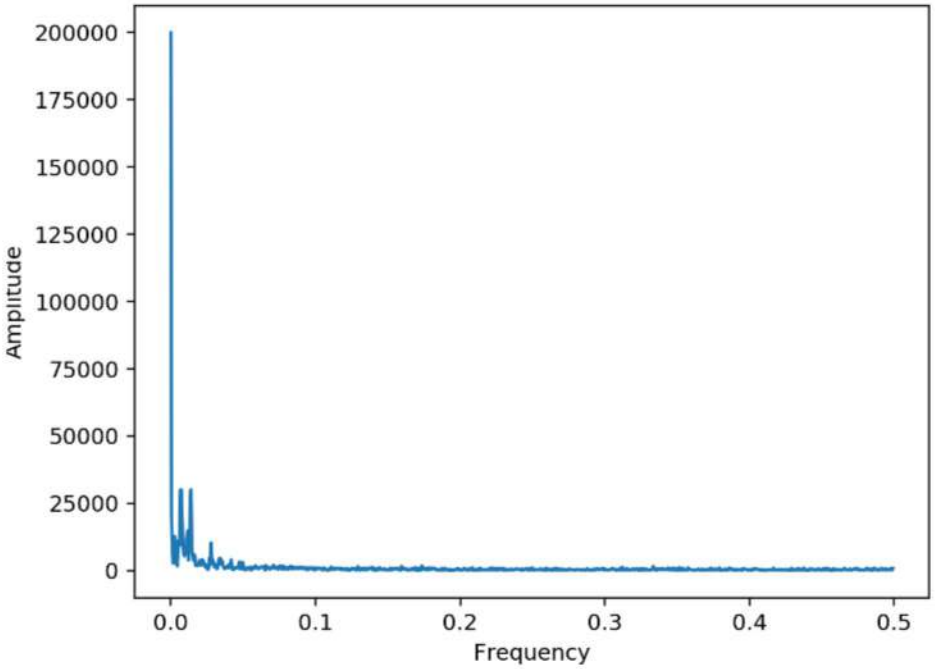
Frequency

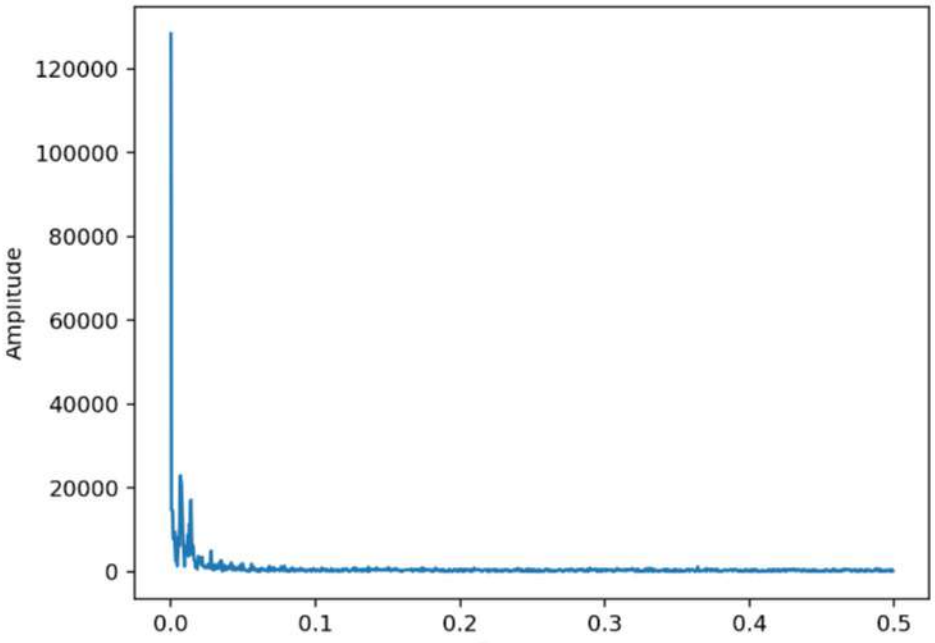

200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

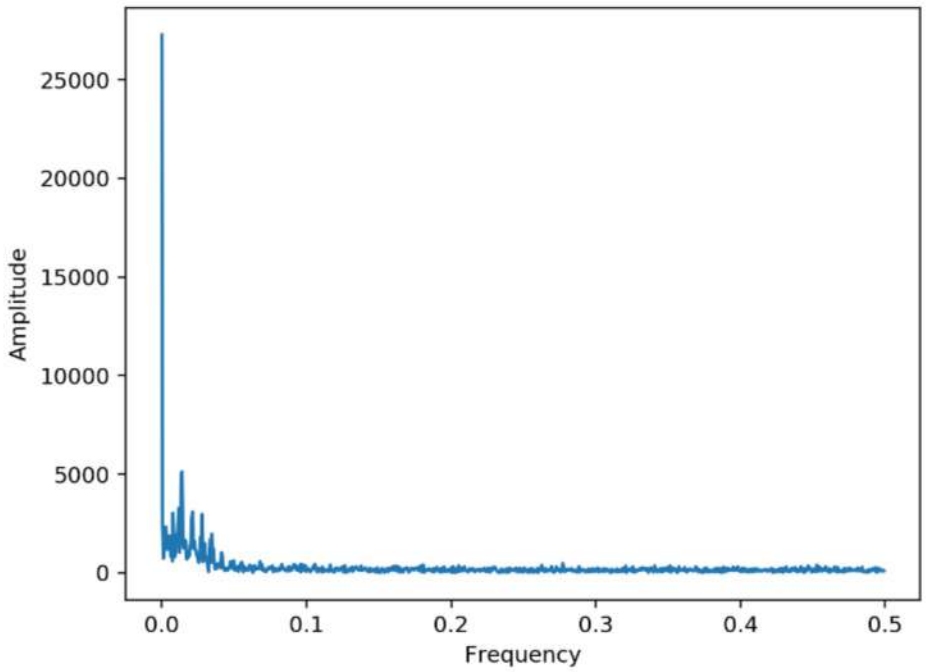

20000
17500
15000
12500
10000
7500
5000
2500
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

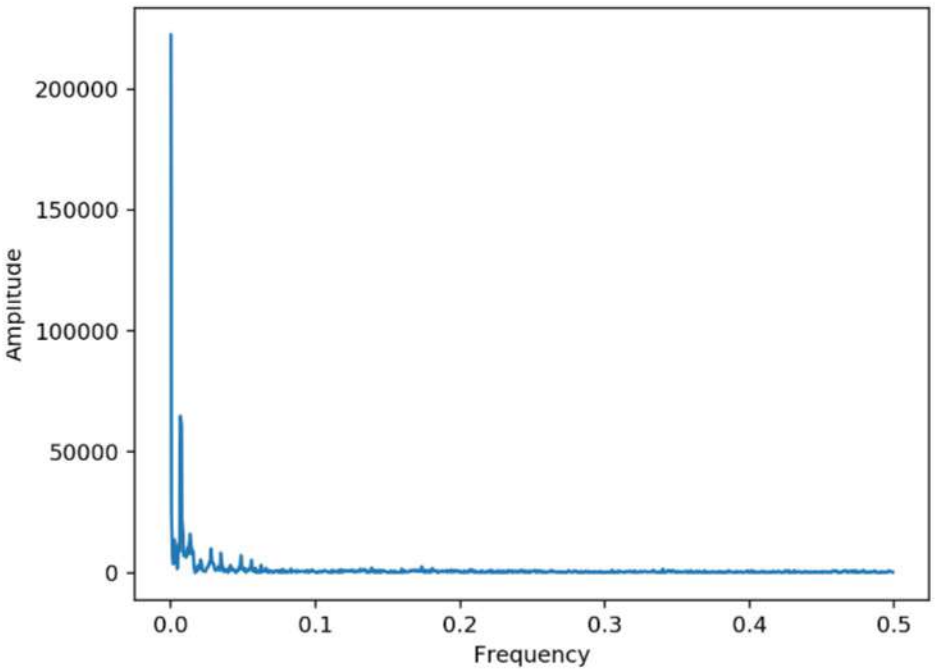

140000
120000


Amplitude
80000
60000
40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

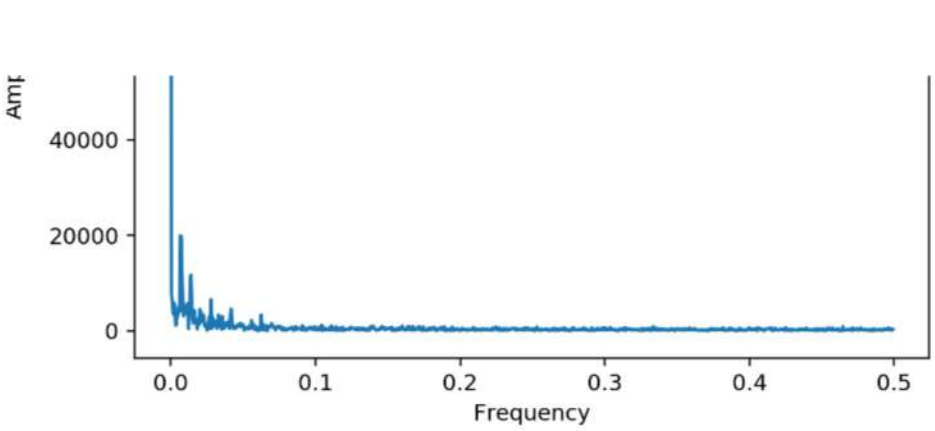

Amplitude
200000
175000
150000
125000
100000
75000
50000
25000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

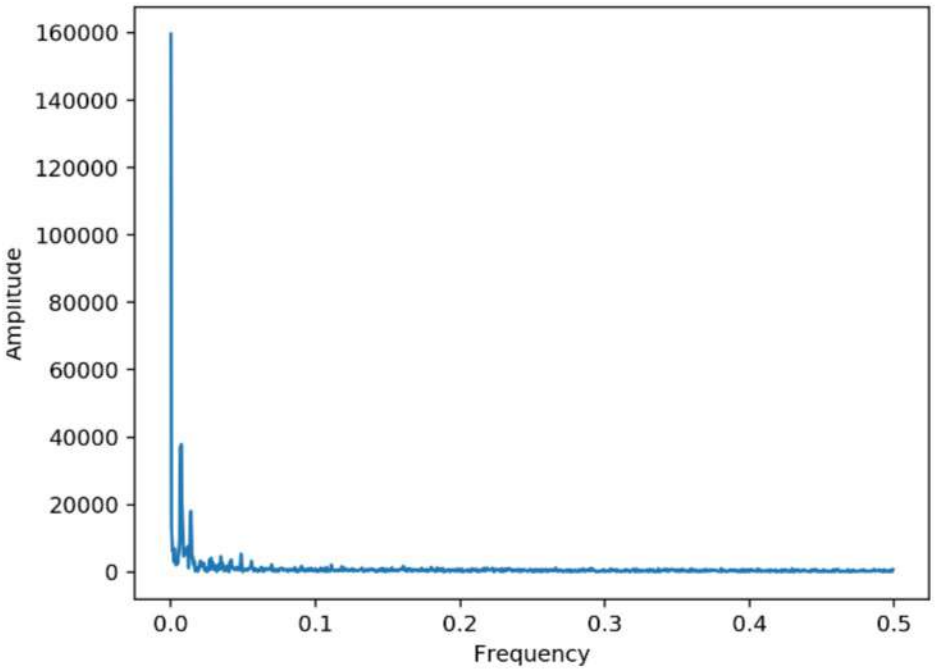

Amplitude
120000
100000
80000
60000
40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5

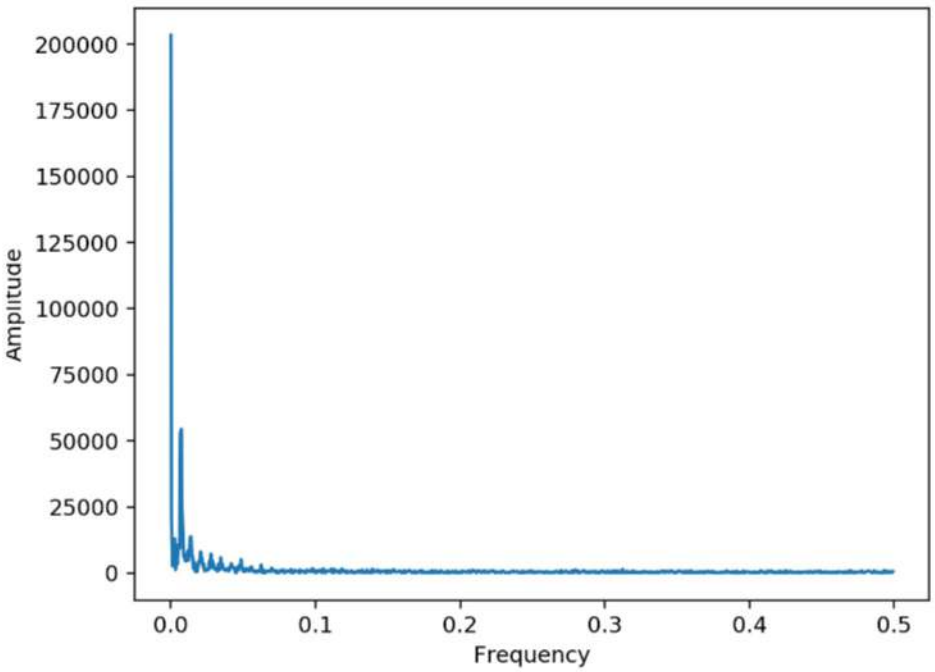

25000
20000
15000
10000
5000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

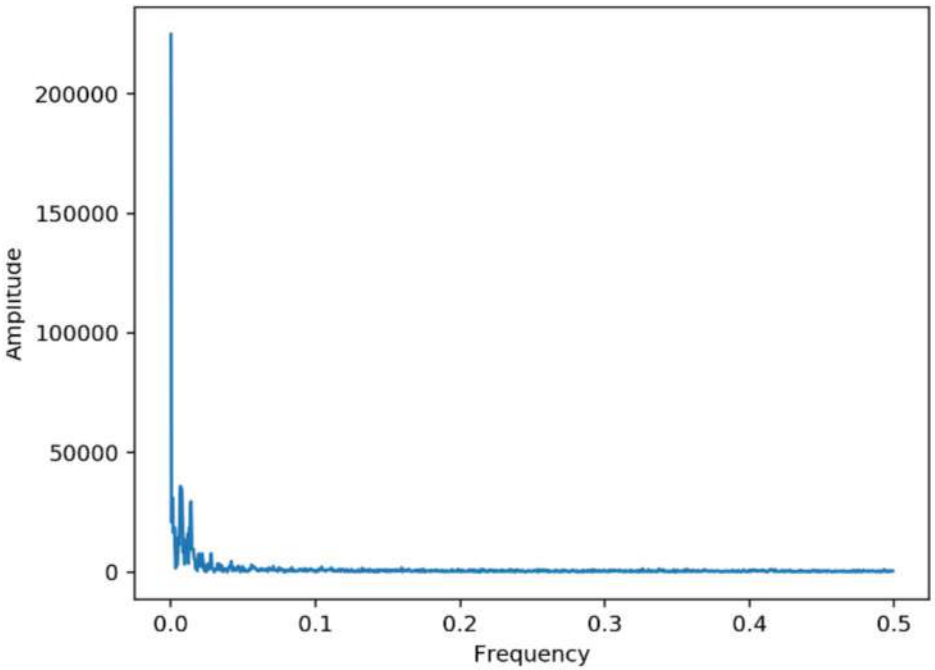

200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

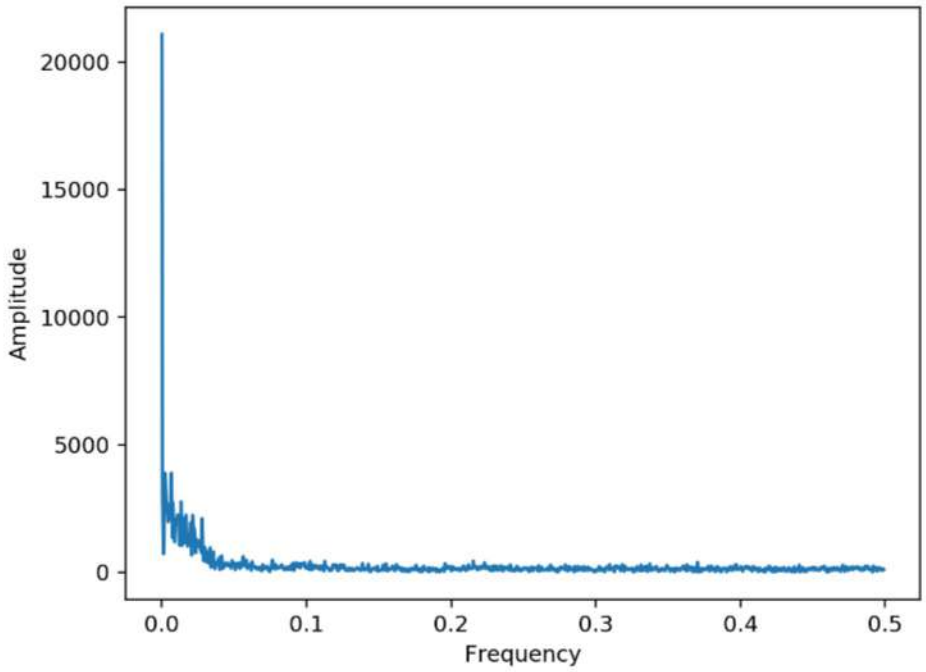

100000
80000


40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

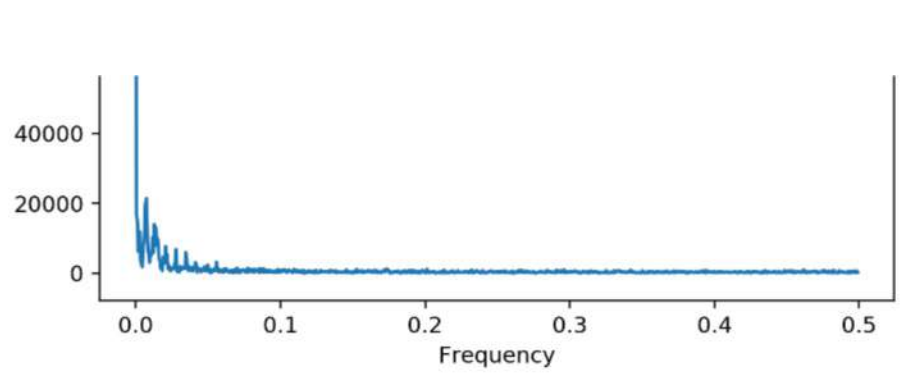

160000
140000
120000
100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

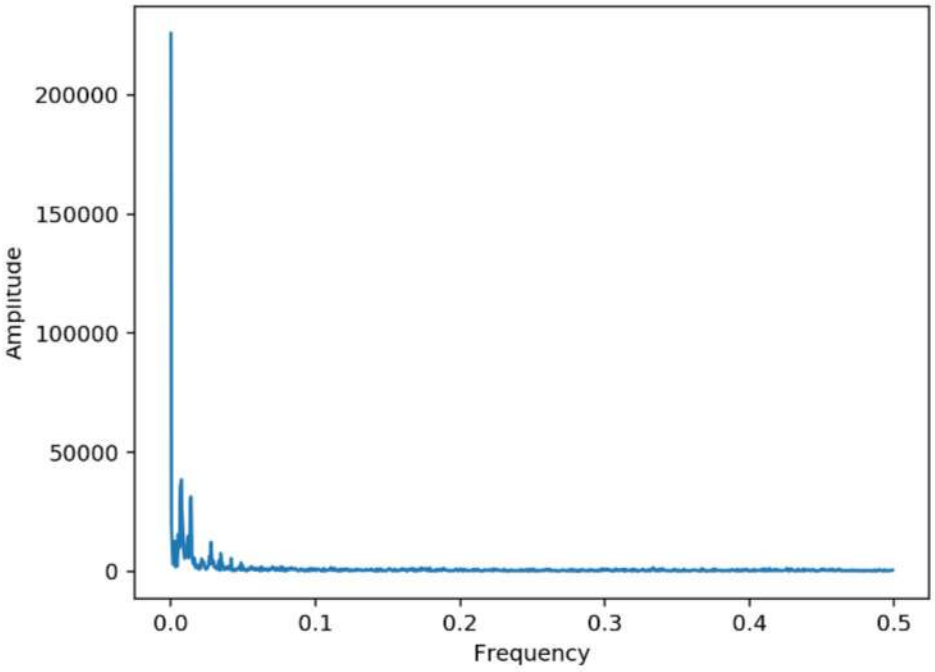

200000
175000
150000
125000
100000
75000
50000
25000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

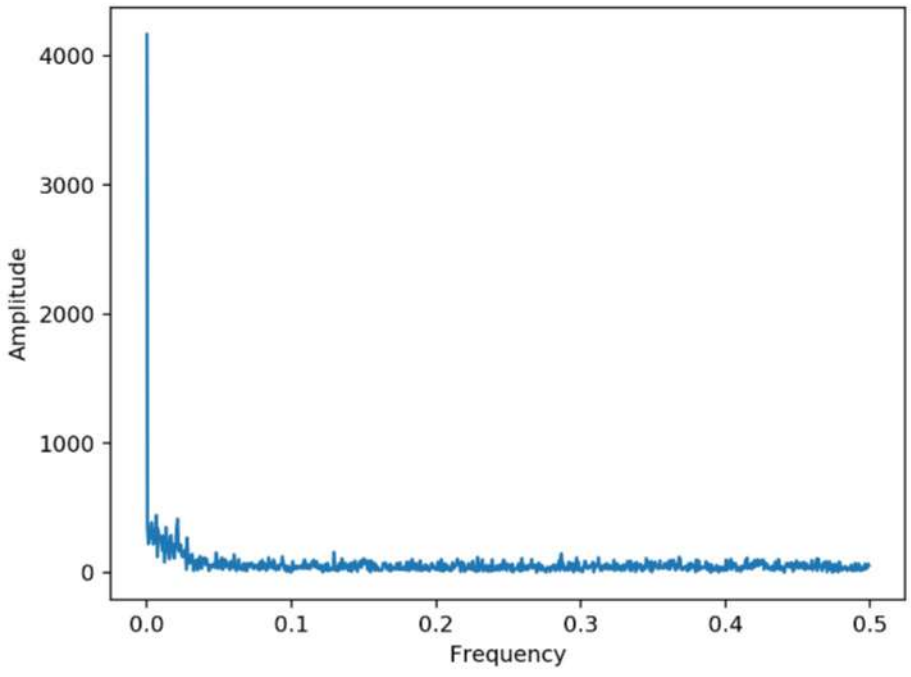

200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

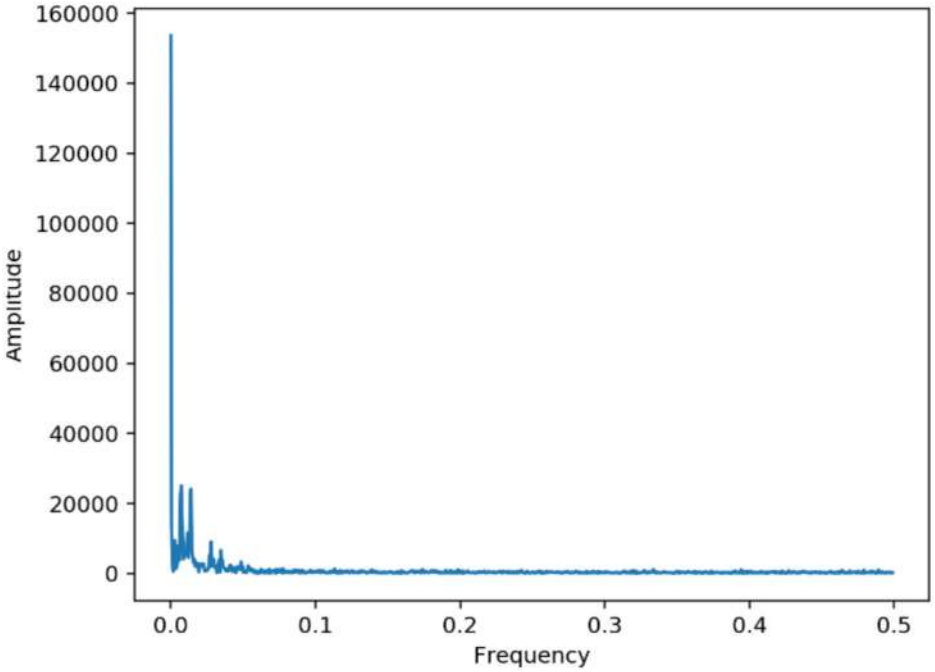

20000
15000
10000
5000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

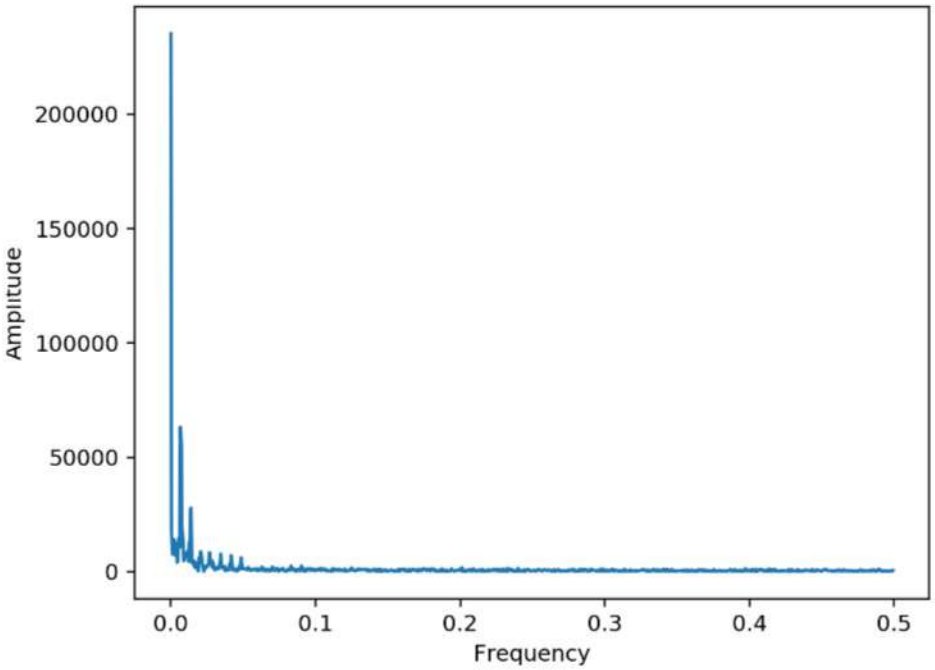

160000
140000
120000
100000
80000


40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

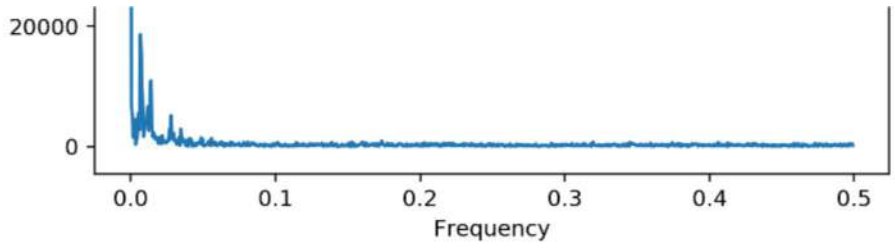

200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

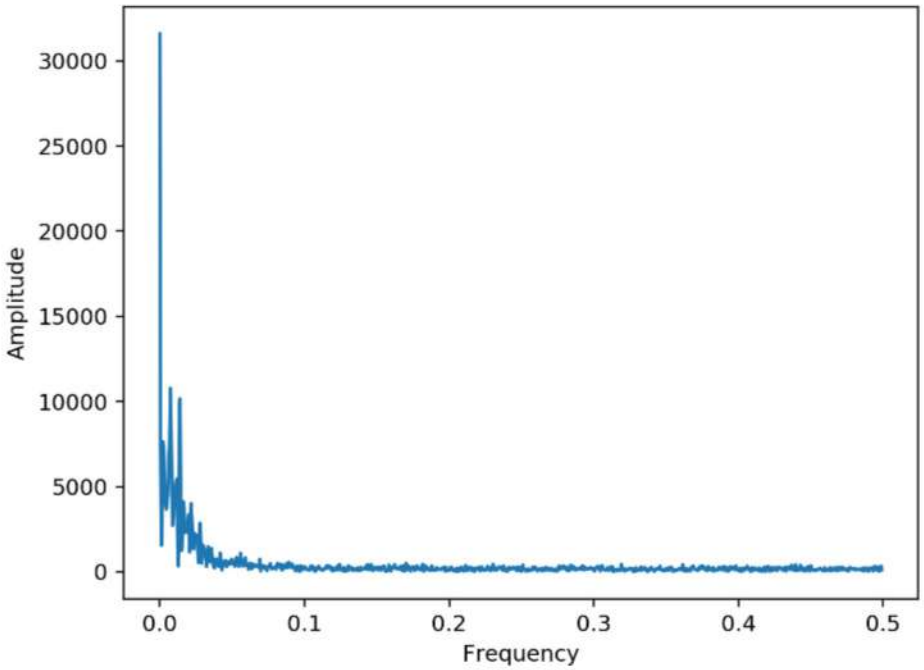

4000
3000
2000
1000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

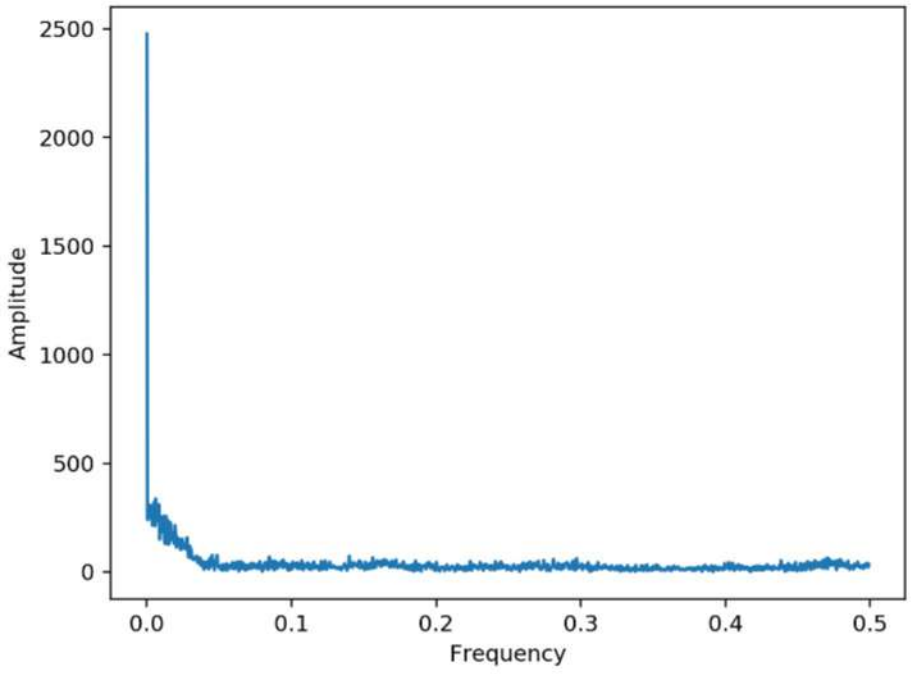

160000
140000
120000
100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

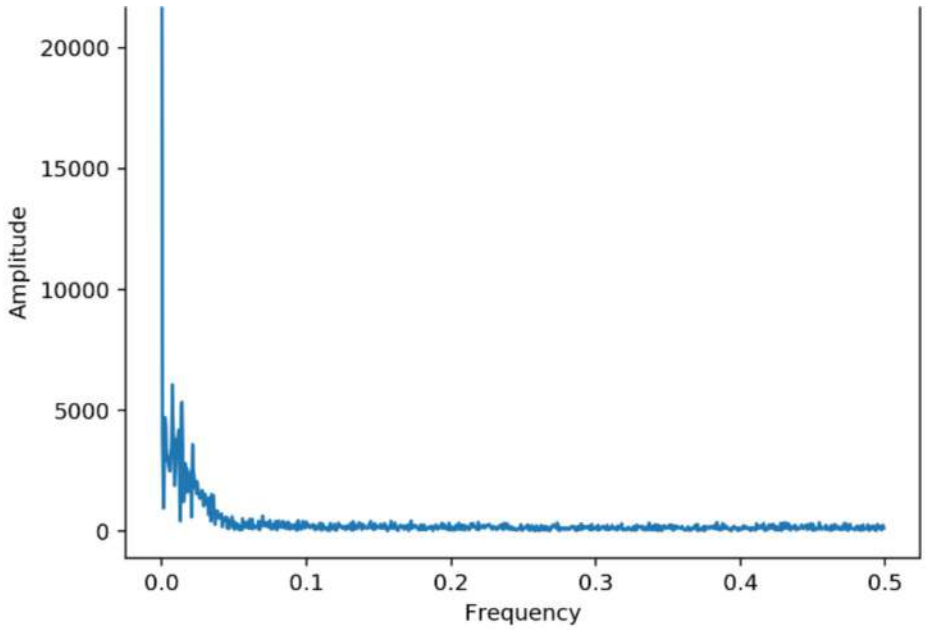

200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

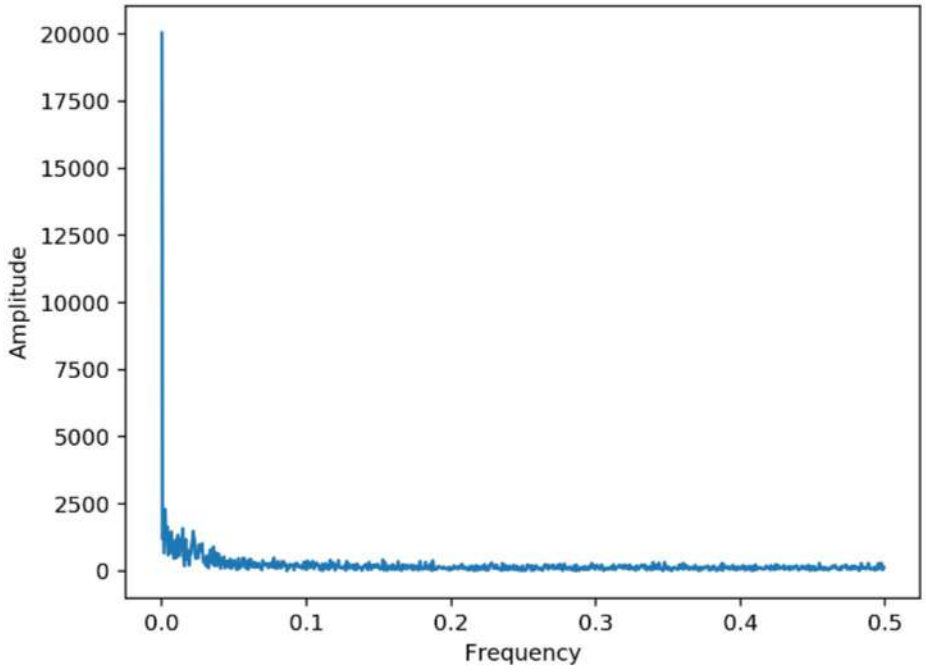

80000
60000
40000
Amplitude


20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

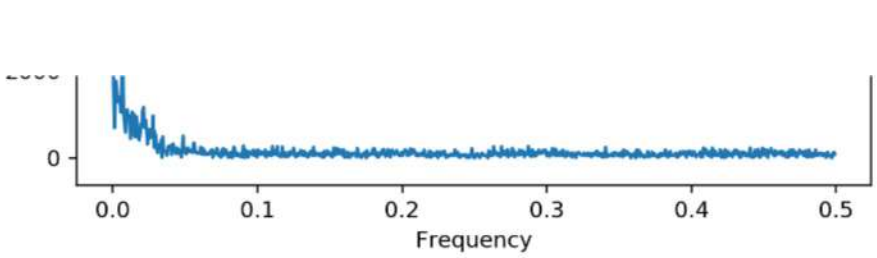

30000
25000
20000
15000
10000
5000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

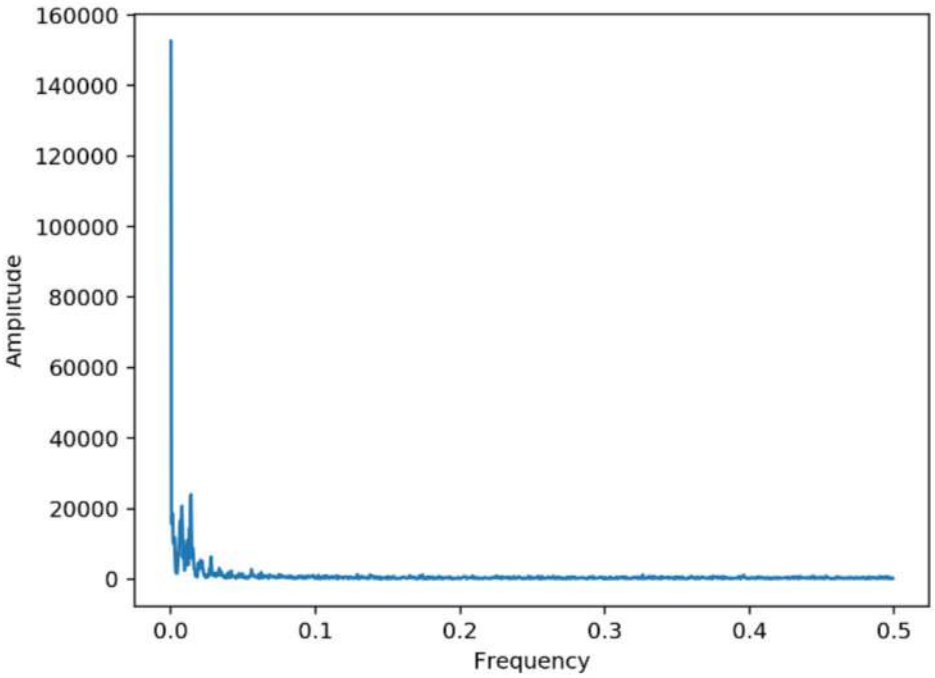

2500
2000
1500
1000
500
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

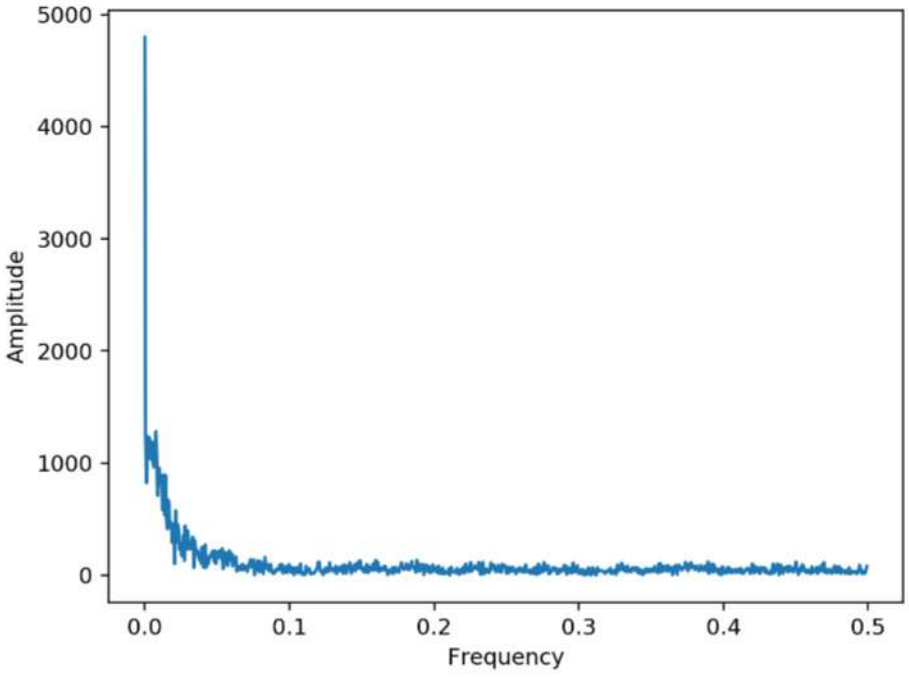

20000
15000
10000
5000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


20000
17500
15000
12500
10000
7500
5000
2500
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

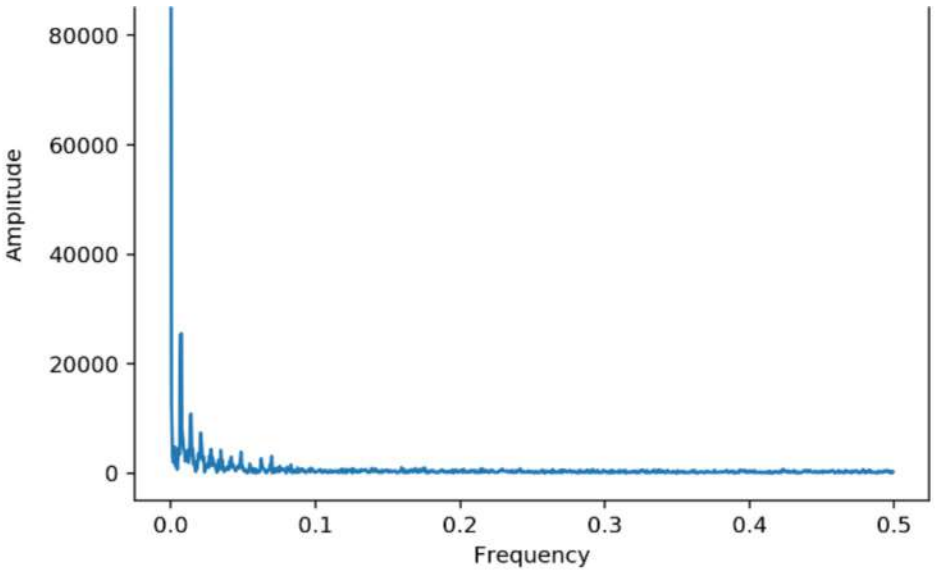

12000
10000
8000
6000
4000
Amplitude

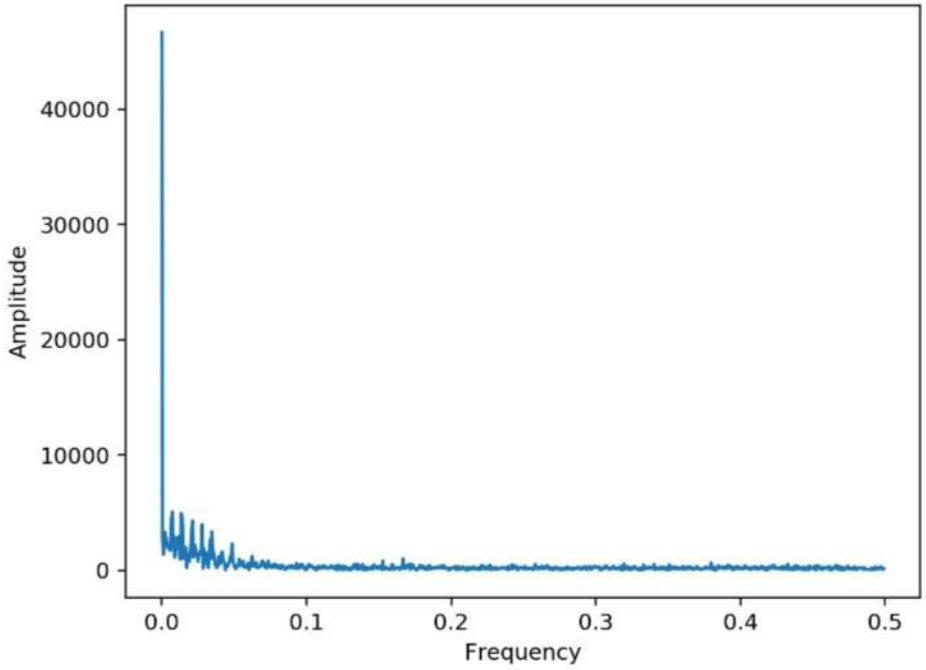

0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


160000
140000
120000
100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

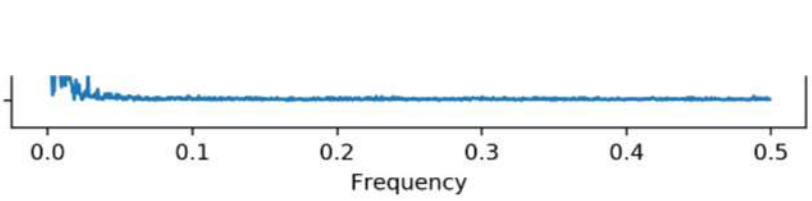

5000
4000
3000
2000
1000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

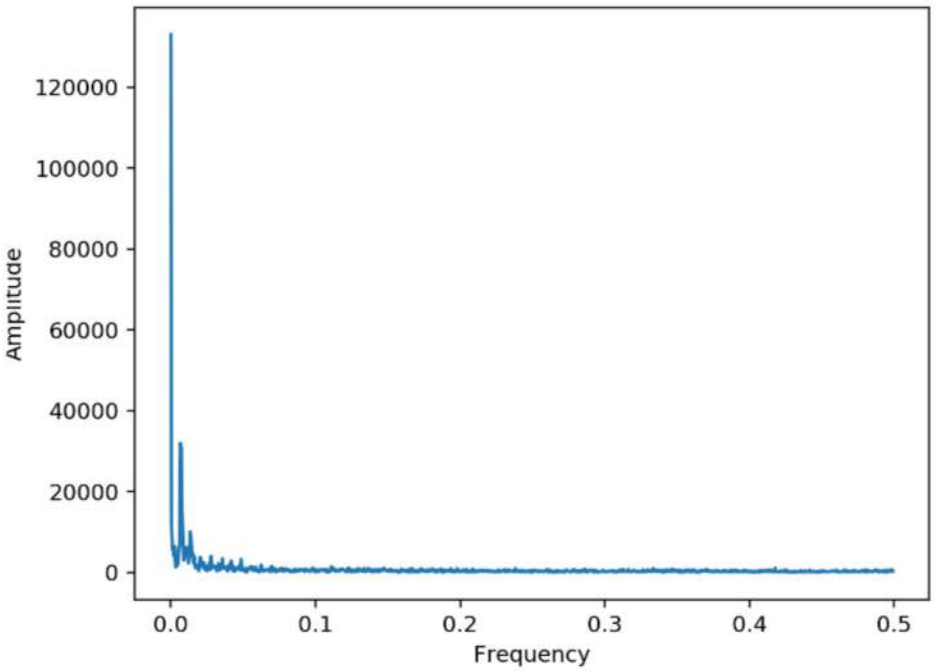

100000

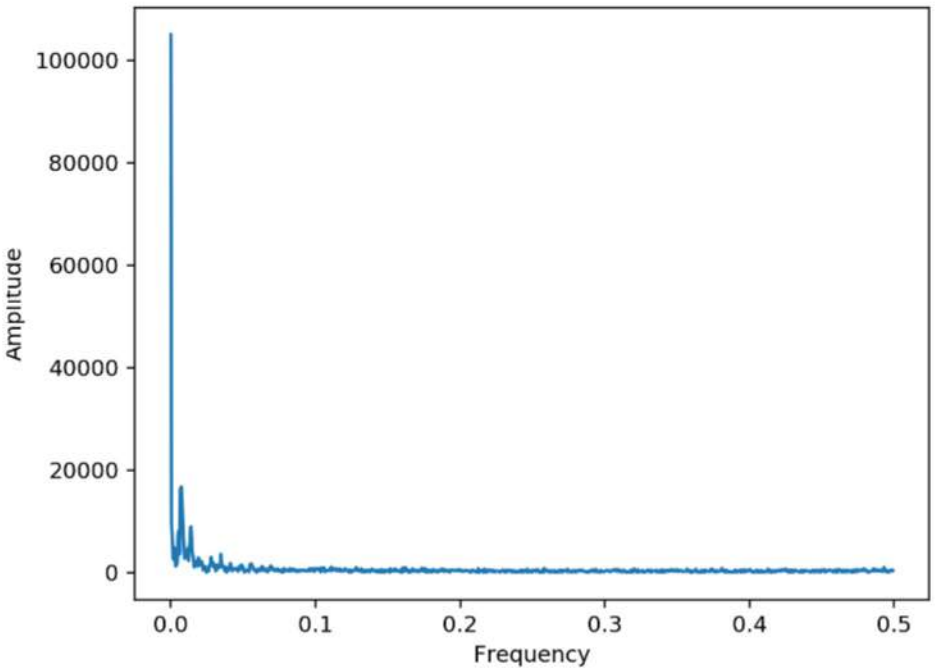

80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


40000
30000
20000
10000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

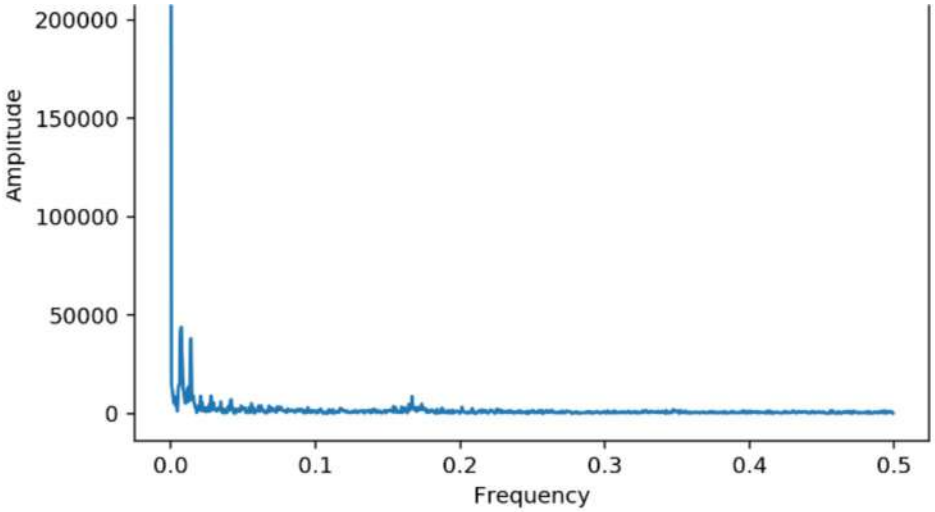

120000
100000
80000
60000
40000
Amplitude


0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


120000
100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


100000
80000
60000
40000
20000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


250000



In [118]:

```
for i in range(50200):
    k=df_train.loc[i,'cluster']
    df_train.loc[i,'Mean_Amplitude']=top_amp[k]
    df_train.loc[i,'Mean_Frequence']=top_freq[k]
```

In [119]:

```
df_train.head()
```

Out[119]:

	ft_5	ft_4	ft_3	ft_2	ft_1	weekday	exp_avg	cluster	Mean_Amplitude	Mean_Frequence
0	29	29	130	155	215	4	191	0	10.24532	0.00725
1	29	130	155	215	215	4	207	0	10.24532	0.00725
2	130	155	215	215	197	4	199	0	10.24532	0.00725
3	155	215	215	197	200	4	199	0	10.24532	0.00725
4	215	215	197	200	235	4	224	0	10.24532	0.00725

In [120]:

```
for i in range(21320):
    k=df_test.loc[i,'cluster']
    df_test.loc[i,'Mean_Amplitude']=top_amp[k]
    df_test.loc[i,'Mean_Frequence']=top_freq[k]
```

In [121]:

```
df_test.head()
```

Out[121]:

	ft_5	ft_4	ft_3	ft_2	ft_1	weekday	exp_avg	cluster	Mean_Amplitude	Mean_Frequence
0	131	190	164	188	203	5	196	0	10.24532	0.00725
1	190	164	188	203	203	5	200	0	10.24532	0.00725
2	164	188	203	203	214	5	209	0	10.24532	0.00725
3	188	203	203	214	207	5	207	0	10.24532	0.00725
4	203	203	214	207	195	5	198	0	10.24532	0.00725

Using Linear Regression

In [92]:

```
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot
```

In [362]:

```
LR = linear_model.SGDRegressor(loss='squared_loss', max_iter=1000, tol=1e-3,learning_rate='optimal'
)

parameters = {'alpha':[10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3,10**4]}
clf = GridSearchCV(LR, parameters, cv=2, scoring='r2')
clf.fit(df_train, tsne_train_output)
```

Out[362]:

```
GridSearchCV(cv=2, error_score='raise-deprecating',
       estimator=SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
       eta0=0.01, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='squared_loss', max_iter=1000,
       n_iter=None, n_iter_no_change=5, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, tol=0.001, validation_fraction=0.1,
       verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='r2', verbose=0)
```

In [363]:

```
print('best score is',clf.best_score_,'best alpha is', clf.best_params_)
```

```
best score is 0.8704485028441211 best alpha is {'alpha': 10000}
```

In [337]:

```
LR = linear_model.SGDRegressor(alpha=10000, loss='squared_loss', max_iter=1000, tol=1e-3,learning_r
ate='optimal')

LR.fit(df_train, tsne_train_output)

y_pred = LR.predict(df_test)
lr_test_predictions = [round(value) for value in y_pred]

y_pred = LR.predict(df_train)
lr_train_predictions = [round(value) for value in y_pred]
```

Using Random Forest Regressor

In [126]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [252]:

```
RF = RandomForestRegressor(random_state=0)
parameters = {'max_depth':[1, 5, 10, 50],'n_estimators':[50, 100, 200]}
clf = RandomizedSearchCV(RF, parameters, cv=2, scoring='r2',random_state=0)
clf.fit(df_train, tsne_train_output)
```

Out[252]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
```

```
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False),
           fit_params=None, iid='warn', n_iter=10, n_jobs=None,
           param_distributions={'max_depth': [1, 5, 10, 50], 'n_estimators': [50, 100, 200]},
           pre_dispatch='2*n_jobs', random_state=0, refit=True,
           return_train_score='warn', scoring='r2', verbose=0)
```

In [253]:

```
print('best score is',clf.best_score_,'best alpha is', clf.best_params_)
```

```
best score is 0.9545969609715881 best alpha is {'n_estimators': 100, 'max_depth': 5}
```

In [254]:

```
RF = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=0)

RF.fit(df_train, tsne_train_output)

y_pred = RF.predict(df_test)
rnd_test_predictions = [round(value) for value in y_pred]

y_pred = RF.predict(df_train)
rnd_train_predictions = [round(value) for value in y_pred]
```

Using XgBoost Regressor

In [138]:

```
import xgboost as xgb
```

In [255]:

```
XG = xgb.XGBRegressor(subsample=0.7, colsample_bytree=0.7, random_state=0, reg_alpha=1,reg_lambda=0
)

parameters = {'max_depth':[1, 5, 10, 50],'n_estimators':[50, 100, 200]}
clf = RandomizedSearchCV(XG, parameters, cv=2, scoring='r2')
clf.fit(df_train, tsne_train_output)
```

Out[255]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.7, gamma=0, importance_type='gain',
       learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='reg:linear', random_state=0, reg_alpha=1,
       reg_lambda=0, scale_pos_weight=1, seed=None, silent=True,
       subsample=0.7),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'max_depth': [1, 5, 10, 50], 'n_estimators': [50, 100, 200]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='r2', verbose=0)
```

In [256]:

```
print('best score is',clf.best_score_,'best alpha is', clf.best_params_)
```

```
best score is 0.9535670381021343 best alpha is {'n_estimators': 100, 'max_depth': 1}
```

In [257]:

```
XG = xgb.XGBRegressor(n_estimators=100, max_depth=1,subsample=0.7, colsample_bytree=0.7, random_sta
te=0, reg_alpha=1,reg_lambda=0)
```

```
y_pred = XG.predict(df_test)
xgb_test_predictions = [round(value) for value in y_pred]

y_pred = XG.predict(df_train)
xgb_train_predictions = [round(value) for value in y_pred]
```

Calculating the error metric values for various models

In [261]:

```
train_mape=[]
test_mape=[]


train_mape.append((mean_absolute_error(tsne_train_output,rnd_train_predictions))/(sum(tsne_train_ou
tput)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,
xgb_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,
lr_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))


test_mape.append((mean_absolute_error(tsne_test_output,
rnd_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output,
xgb_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output,
lr_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
```

Error Metric Matrix

In [368]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("----------------------------------------------------------------------------------------------------
----------")
print ("Random Forest Regression -                          Train: ",train_mape[0],"      Test: ",te
t_mape[0])
print ("XgBoost Regression -                  Train: ",train_mape[1],"     Test: ",test_mape[1])
print ("Linear Regression -                       Train: ",train_mape[2],"      Test: ",test_mape
2])
print ("----------------------------------------------------------------------------------------------------
----------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
----------------------------------------------------------------------------------------------------
-
Random Forest Regression -                          Train:  0.12394861578621968       Test:  0.12726
85697900028
XgBoost Regression -                  Train:  0.11394861578621968      Test:  0.1178821643653683
Linear Regression -                       Train:  0.16233000236053188       Test:
0.1694168068900222
----------------------------------------------------------------------------------------------------
-
```

Result

In [369]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Mape (Train)", "Mape (test)"]
x.add_row(["Simple Moving average",  'n/a', 0.131])
x.add_row(["Weighted Moving average",  'n/a', 0.129])
x.add_row(["Exponential Weighted Moving average",  'n/a', 0.126])
x.add_row(["Random Forest",  0.123, 0.127])
```

```
print(x)
```

```
+-----------------------------------+-------------+-------------+
|            Vectorizer             | Mape (Train) | Mape (test) |
+-----------------------------------+-------------+-------------+
|       Simple Moving average       |     n/a     |    0.131    |
|      Weighted Moving average      |     n/a     |    0.129    |
| Exponential Weighted Moving average |     n/a     |    0.126    |
|           Random Forest           |    0.123    |    0.127    |
|              XG Boost             |    0.113    |    0.117    |
|         Linear regression         |    0.162    |    0.169    |
+-----------------------------------+-------------+-------------+
```

Step By Step Procedure

1. Downloaded the dataset from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml for Jan 2015 and Feb 2015
2. Our objective -To find number of pickups, given location cordinates(latitude and longitude) and time, in the query reigion and surrounding regions.
3. Set the performace metrics as Mean Absolute percentage error and Mean Squared error
4. Loaded the dataset into Dask Dataframe and sorted the dataset based on Pick Up Time in ascending order
5. Found out the outlier Pickup location(based on latitude and longitude values) found out many location which were outside New York
6. Found out the outlier Drop Off location, surprisingly some drop off location were in the middle of the ocean (Faulty GPS may be)
7. Found out the outlier trip durations as the maximum allowed trip duration in a 24 hour interval is 12 hours. some trip durations were in negative too and few were very large.
8. Found out the outlier Speed of vehicle. Found one speed value upto 192857142 miles per hour. That is totally ridiculous
9. Using the speed data we found out that the avg speed in Newyork is 12.94 miles/hr, so a cab driver can travel 2 miles per 10min on avg. Thats why I decided to use 10 min time bin. As cab driver will not bother to travel for 10 min or 2 miles to find customer.
10. Found out the outlier Trip distance. One trip distance was found to be 213 miles, so that is also ridiculous.
11. Found out the outlier Total Fare. One outlier fare amount was 3006 dollars
12. Removed all the outlier points from the dataset. Fraction of data points that remain after removing outliers was 97 %
13. Found out the optimal number of clusters which will divide the new york city into optimal number of region bins
14. On applying K means clustering we found out that 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 = 0.422097622420765
15. We choose 40 as optimal no of cluster as we dont want our Min inter cluster distance to be too low and also want more no of cluster to have a intercluster-distance less than 2 Miles
16. I plotted the cluster
17. First I converted the Pick Up times into Unix Timestamp and then I performed the time binning and use 10 min time interval for binning.
18. Then I found out the total no of Pickups for each time bin in each cluster individually.
19. I found out that for many clusters there were many time bins, in which there were no Pick Ups. So I have to perform the smoothening, and fill some values for those time bins in each cluster which have 0 Pick Ups
20. For smoothening, I will use two approach Fill by 0 or Fill by average.
21. For 2015 Jan data i will use Fill by average and for 2015 Feb data i will use Fill by 0 to avoid data leakage.
22. I created a new DataFrame ratios_ran and in the ratios_ran['Given'] I filled it with 2015 Jan data, and in the ratios_jan['Predictions'] I filled it with 2015 Feb data, and in the ratios_jan['Ratios'] I filled it with the ratios of the two.
23. I created some simple Baseline models named as Simple Moving Average, Weighted Moving Average, and Exponential Weighted Moving Average.
24. For each of these Baseline models, I used two approach to predict the no of Customer for Pick Ups. (a) One is to use the Ratio of 2015 Jan and 2015 Feb pickup values at Time bin 'T' for Cluster 'C' and predict the Pick Up values at Time bin 'T+1' for Cluster 'C' for 2015 Feb. (b) Other is to use the Pick Up values at previous Time bins for 2015 Feb (i.e T-n .....T-1) and predict the Pick Up value at Time bin 'T+1'
25. Best Baseline model was found out to be Exponential Weighted Moving Average which gave the least MAPE value.
26. Now I will use Regression Models
27. I cant use simple Train test split as I have to maintain the Time series of the data and split it according to the Time intervals (i.e for every region we have 70% data in train and 30% in test, ordered date-wise for every region)
28. I added pickup latitude and pickup longtitude values for each data point
29. I added each weekday values for each data point
30. I added Exponential Weighted Moving Average for each data points.
31. I added previous 5 pickup values for each time bin for each cluster for each data points.

Each data point contains 5 features

33. I found out the fourier features for each cluster
34. I got the top 5 Amplitude and the frequency at which they occur for each cluster and append it to the train dataset and test dataset
35. As individual values of top 5 amplitude and the frequency at which they occur were not giving good MAPE values so I calculated the Mean of top 5 amplitude and the frequency and added it to the dataset.
36. I used three Regression Models named as Linear Regression, Random Forest Regression and XGBoost Regression
37. The best Regression Model was found out to be XGBoost Regression which gave the least MAPE value

The best vectorizer was found to be XG Boost which gave MAPE value less than 0.12