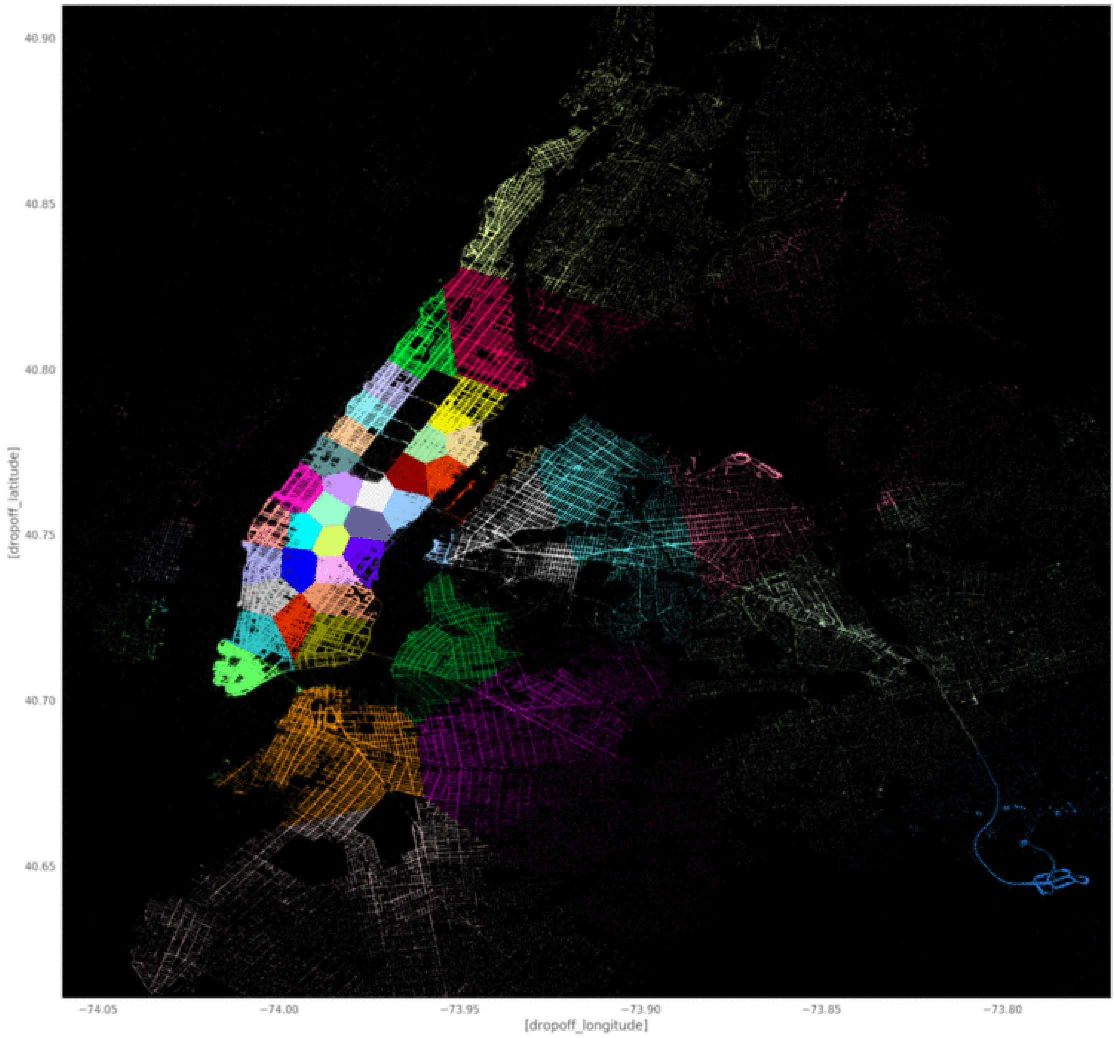
Taxi demand prediction in New York City



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/blo
b/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 straight line distance between two (lat,lon)
 pairs in miles
import gpxpy.geo #Get the haversine distance

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

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

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

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

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

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

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	1. 62G	11135470	19
yellow_tripdata_2016-07	884Mb	10294080	17
yellow_tripdata_2016-08	854Mb	9942263	17
yellow_tripdata_2016-09	870Mb	10116018	17
yellow_tripdata_2016-10	933Mb	10854626	17
yellow_tripdata_2016-11	868Mb	10102128	17
yellow_tripdata_2016-12	897Mb	10449408	17
yellow_tripdata_2015-01	1.84Gb	12748986	19
yellow_tripdata_2015-02	1.81Gb	12450521	19
yellow_tripdata_2015-03	1.94Gb	13351609	19
yellow_tripdata_2015-04	1.90Gb	13071789	19
yellow_tripdata_2015-05	1.91Gb	13158262	19
yellow_tripdata_2015-06	1.79Gb	12324935	19
yellow_tripdata_2015-07	1.68Gb	11562783	19
yellow_tripdata_2015-08	1.62Gb	11130304	19
yellow_tripdata_2015-09	1.63Gb	11225063	19
yellow_tripdata_2015-10	1.79Gb	12315488	19
yellow_tripdata_2015-11	1.65Gb	11312676	19
yellow_tripdata_2015-12	1.67Gb	11460573	19

In [2]:

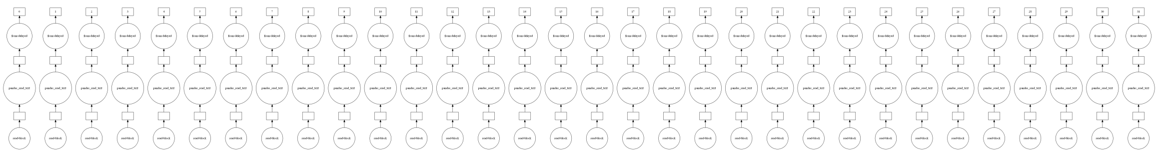
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ip
ynb
month = dd.read_csv('yellow_tripdata_2015-01.csv')
print(month.columns)
```

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

In [3]:

```
month.visualize()
```

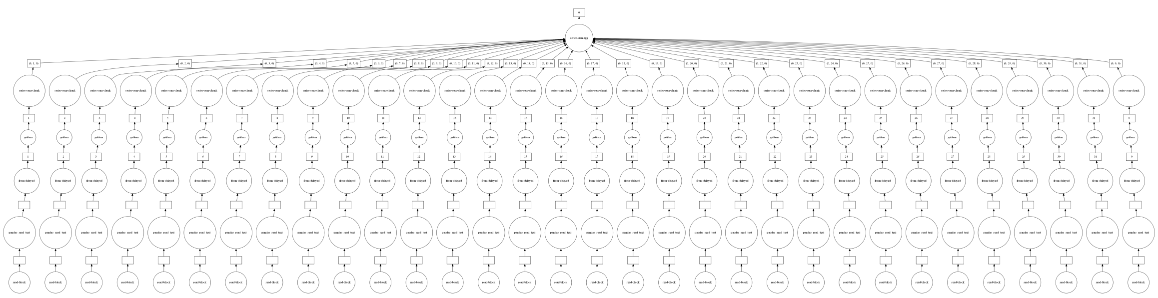
Out[3]:



In [4]:

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

Out[4]:



Features in the dataset:

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. 1. Standard rate 2. JFK 3. Newark 4. Nassau or Westchester 5. Negotiated fare 6. Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip
Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_ latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. 1. Credit card 2. Cash 3. No charge 4. Dispute 5. Unknown 6. Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes. the $0.50and1$ 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_di
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.80
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.50
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3.00

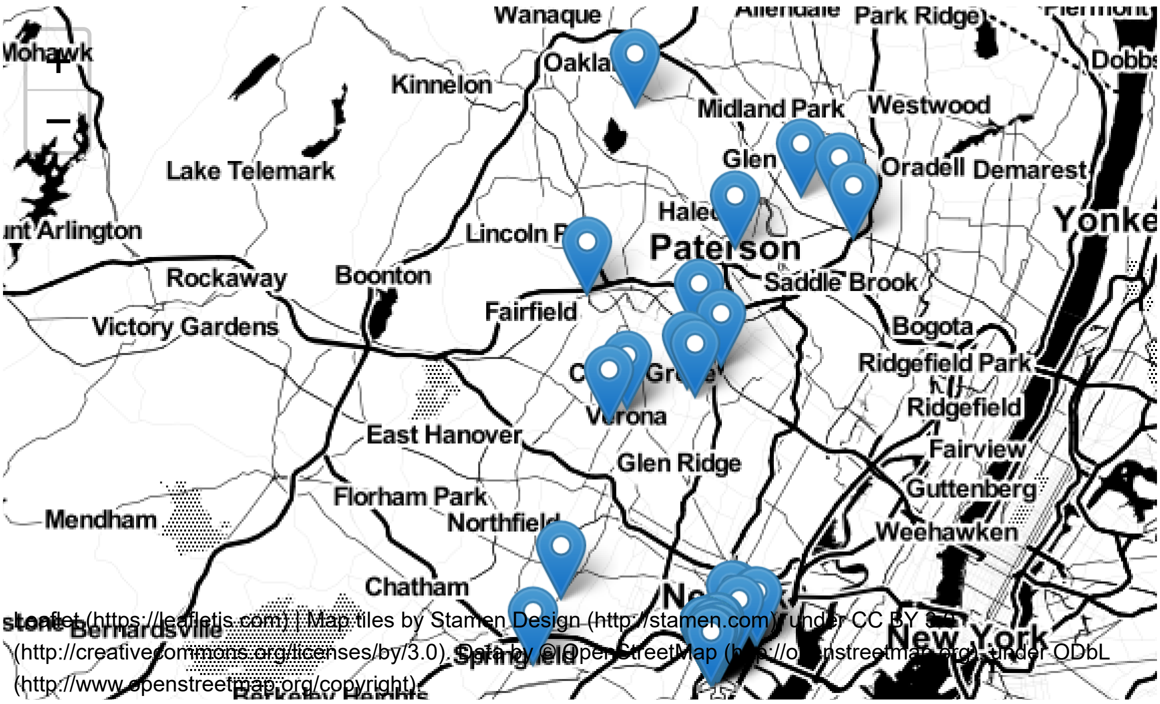
1. Pickup Latitude and Pickup Longitude

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

In [6]:

```
# Plotting pickup cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_lo
cations
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.ht
ml

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

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

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

Out[6]:



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

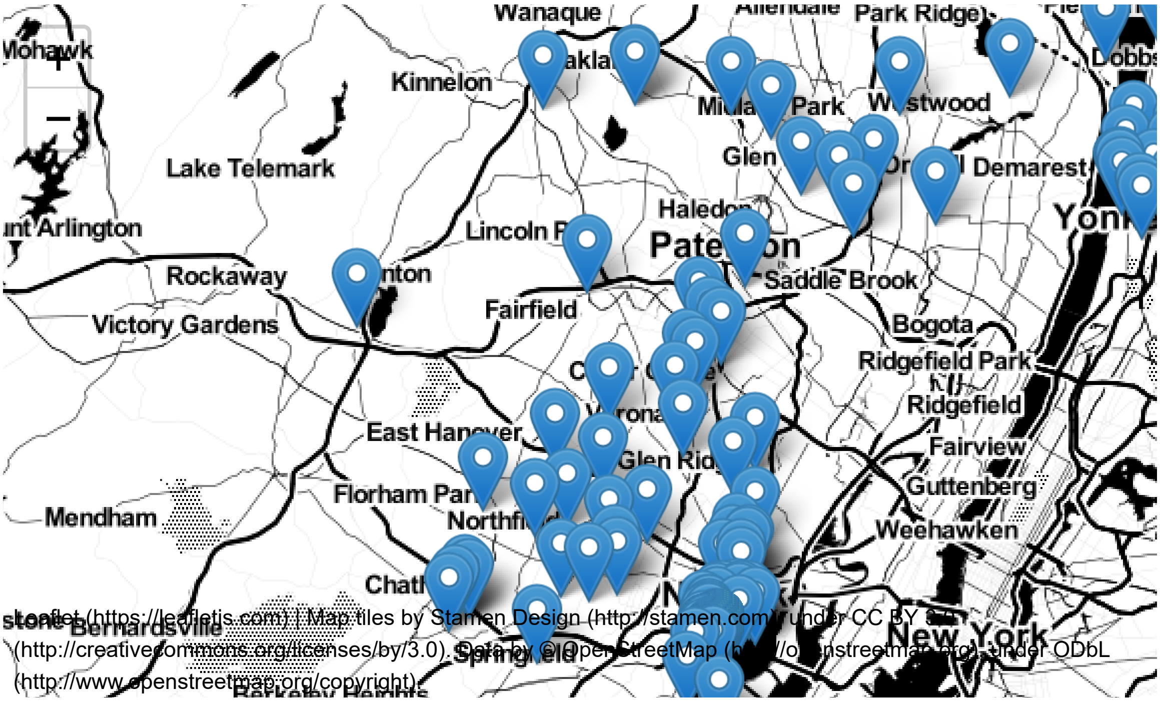
2. Dropoff Latitude & Dropoff Longitude

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

In [7]:

```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outlier_lo
cations
outlier_locations = month[((month.dropoff_longitude <= -74.15) | (month.dropoff_latitud
e <= 40.5774)| \
                   (month.dropoff_longitude >= -73.7004) | (month.dropoff_latitude >= 4
0.9176))]

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

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

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

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

Out[7]:



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

3. Trip Durations:

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

In [9]:

```
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetuple())

def return_with_trip_times(month):
    duration = month[['tpep_pickup_datetime','tpep_dropoff_datetime']].compute()
    #pickups and dropoffs to unix time
    duration_pickup = [convert_to_unix(x) for x in duration['tpep_pickup_datetime'].val
ues]
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'].valu
es]
    #calculate duration of trips
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)

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

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


    return new_frame

frame_with_durations = return_with_trip_times(month)
```

In [10]:

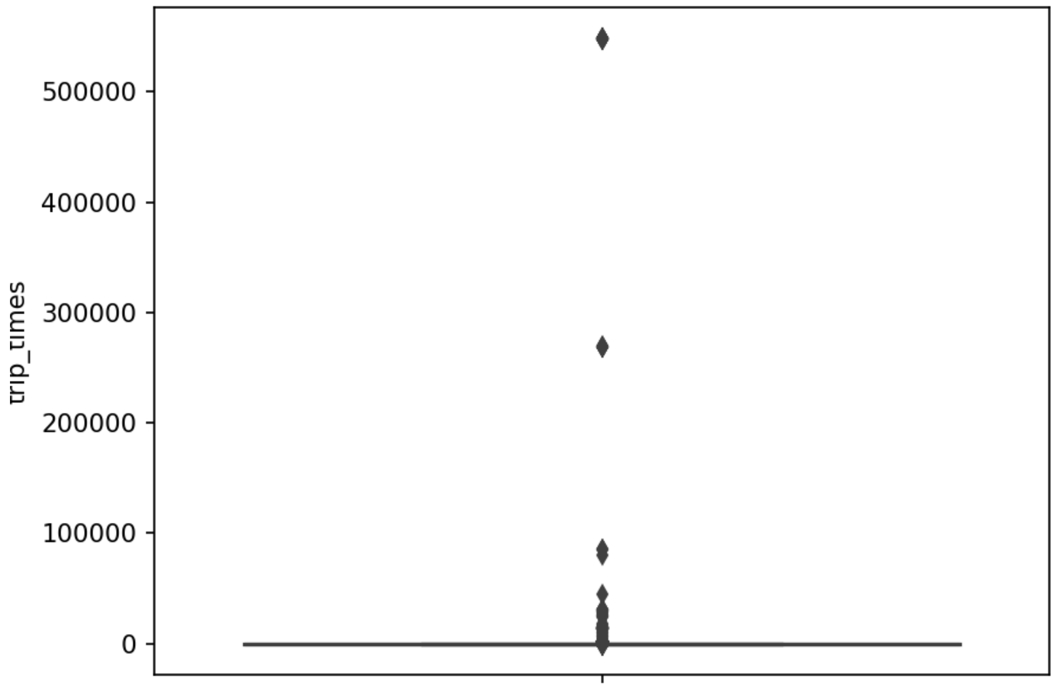
```
frame_with_durations.head()
```

Out[10]:

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

In [11]:

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



In [12]:

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

```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333
80 percentile value is 17.633333333333333
90 percentile value is 23.45
===========================================================================
=========================
12748986
===========================================================================
=========================
100 percentile value is  548555.6333333333
```

In [13]:

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

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

In [14]:

```
#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 [15]:

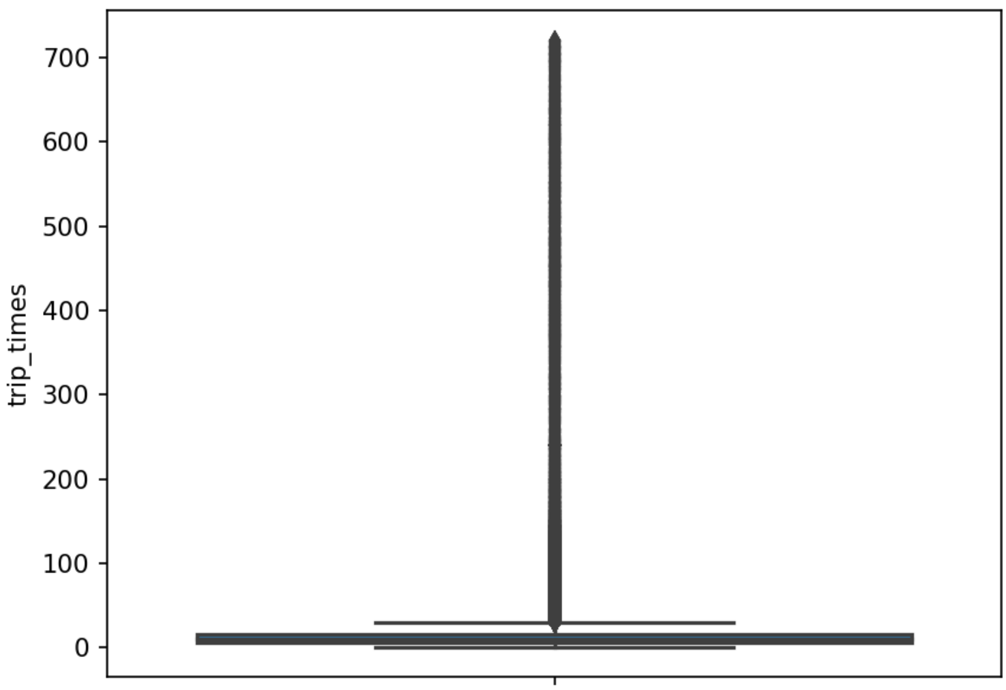
```
frame_with_durations_modified.head()
```

Out[15]:

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

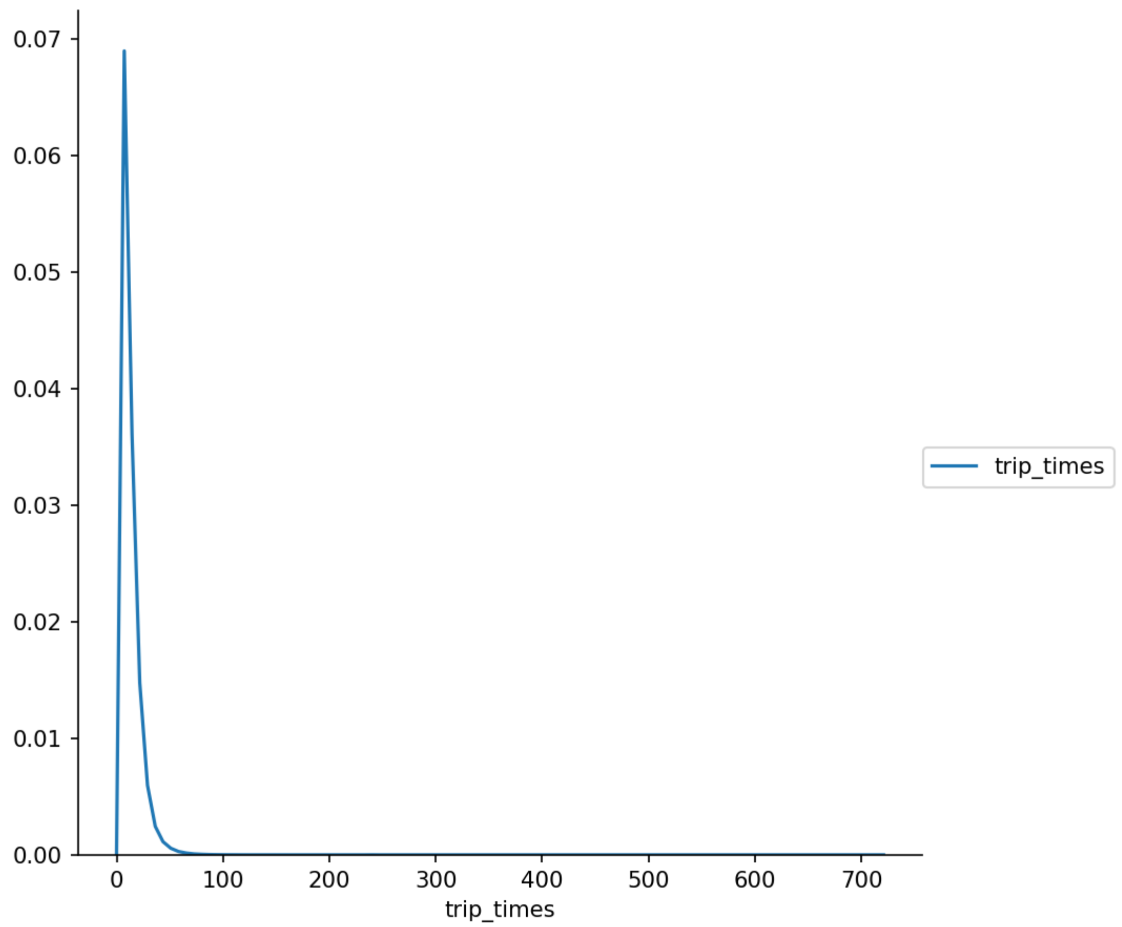
In [16]:

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



In [17]:

```
#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 [18]:

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

In [19]:

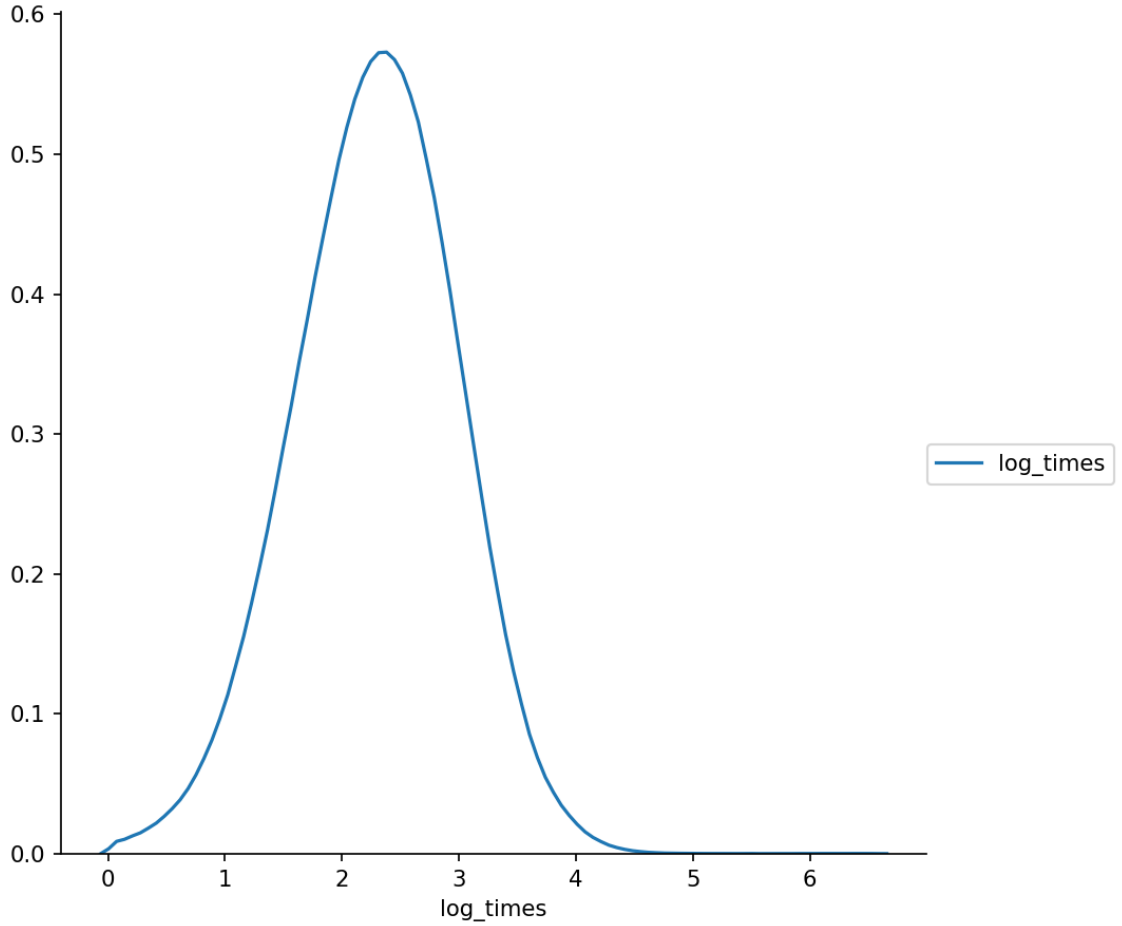
```
frame_with_durations_modified.head()
```

Out[19]:

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

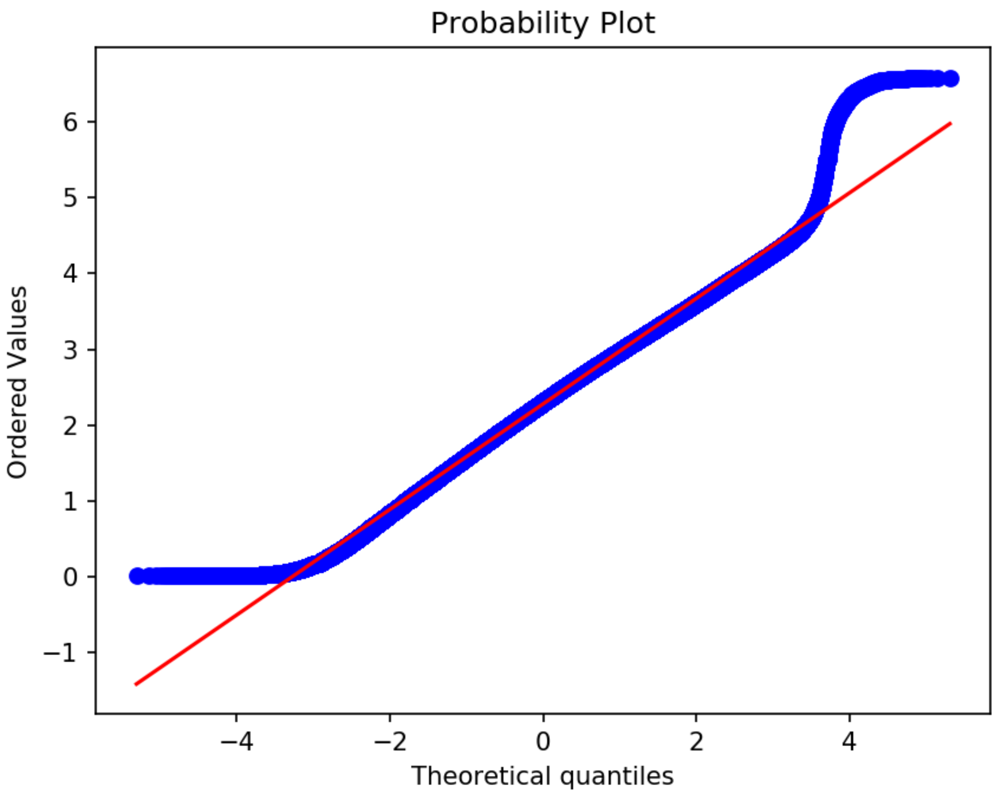
In [20]:

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



In [21]:

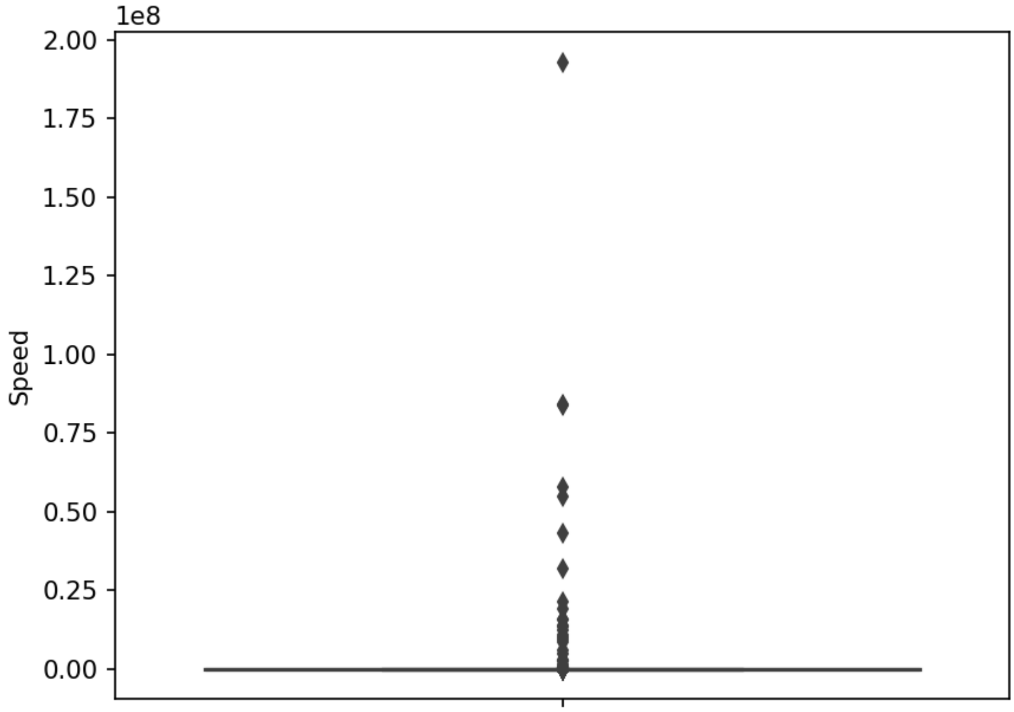
```
import scipy
#Q-Q plot for checking if trip-times is log-normal
scipy.stats.probplot(frame_with_durations_modified['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [22]:

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



In [23]:

```
#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("length of variable",len(var))
print("100 percentile value is ",var[-1])
```

```
0 percentile value is 0.0
10 percentile value is 6.409495548961425
20 percentile value is 7.80952380952381
30 percentile value is 8.929133858267717
40 percentile value is 9.98019801980198
50 percentile value is 11.06865671641791
60 percentile value is 12.286689419795222
70 percentile value is 13.796407185628745
80 percentile value is 15.963224893917962
90 percentile value is 20.186915887850468
length of variable 12635246
100 percentile value is  192857142.85714284
```

In [24]:

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

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

In [25]:

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

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

In [26]:

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

In [27]:

```
frame_with_durations_modified.shape
```

Out[27]:

```
(12647156, 10)
```

In [28]:

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

Out[28]:

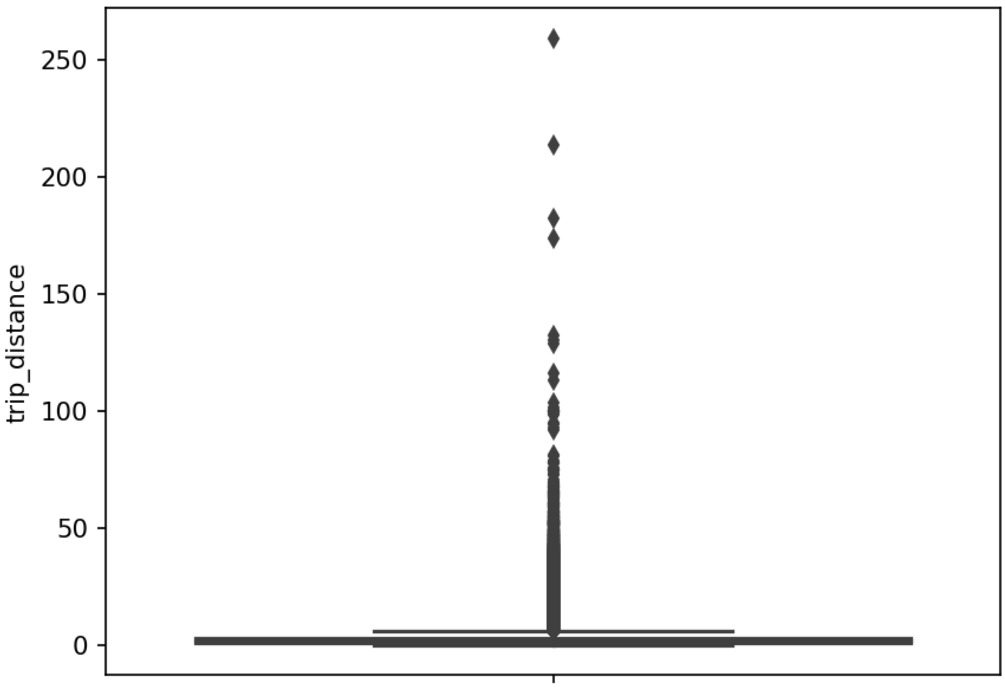
```
12.450173996027528
```

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

4. Trip Distance

In [29]:

```
# 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 [30]:

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

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

In [31]:

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

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

In [32]:

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

In [33]:

```
#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 [34]:

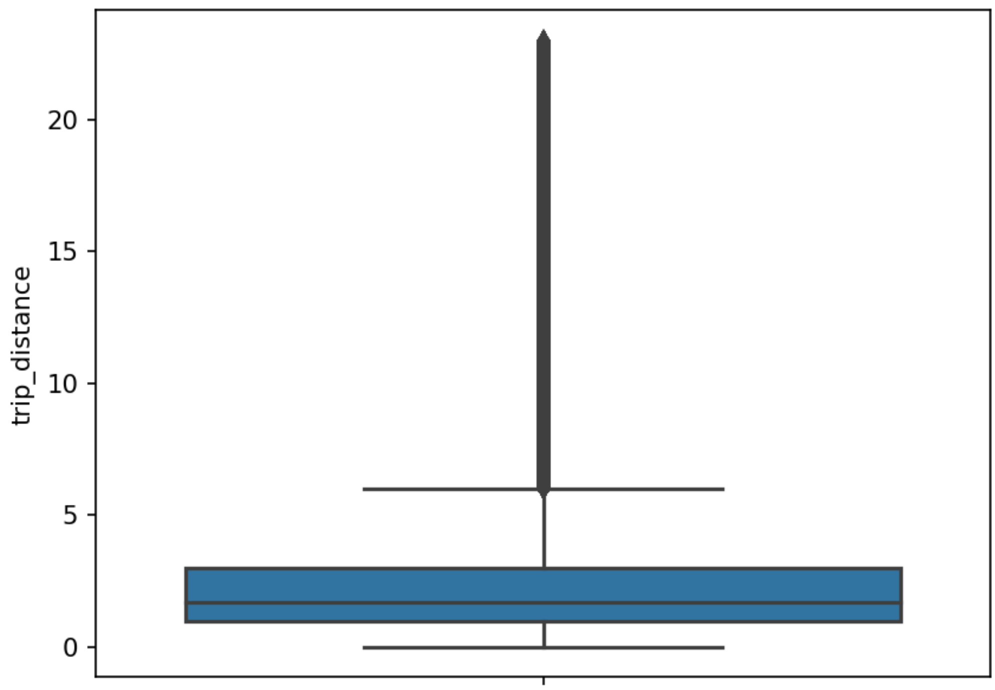
```
frame_with_durations_modified.shape
```

Out[34]:

```
(12656389, 10)
```

In [35]:

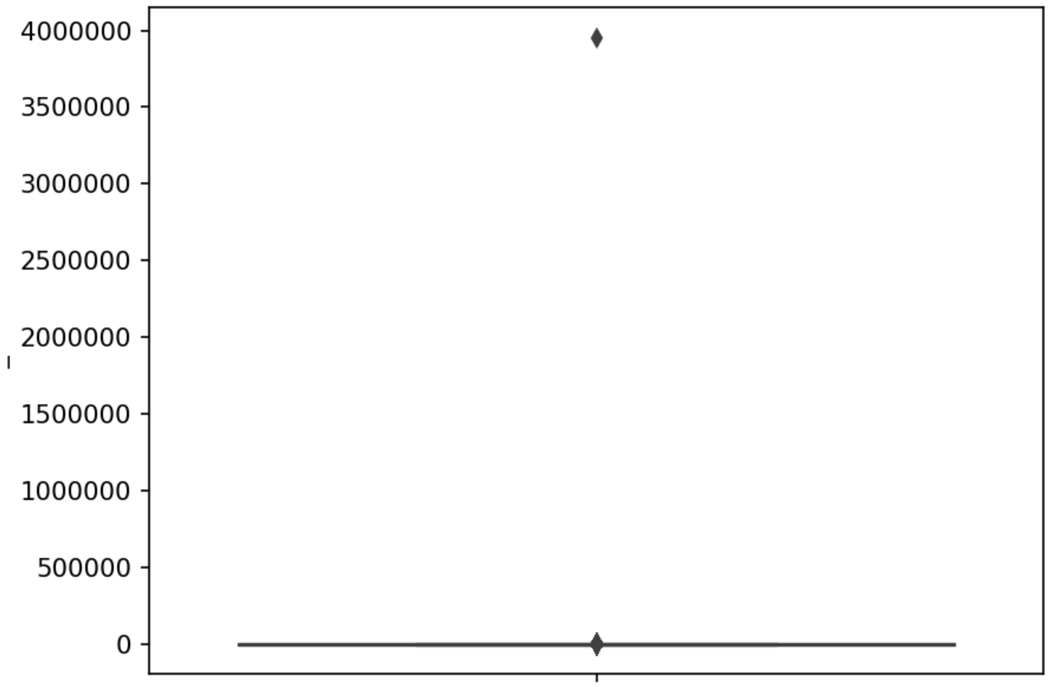
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [36]:

```
# 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 [37]:

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

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

In [38]:

```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,98,99,1
00
for i in range(90,100):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print("100 percentile value is ",var[-1])
```

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

In [39]:

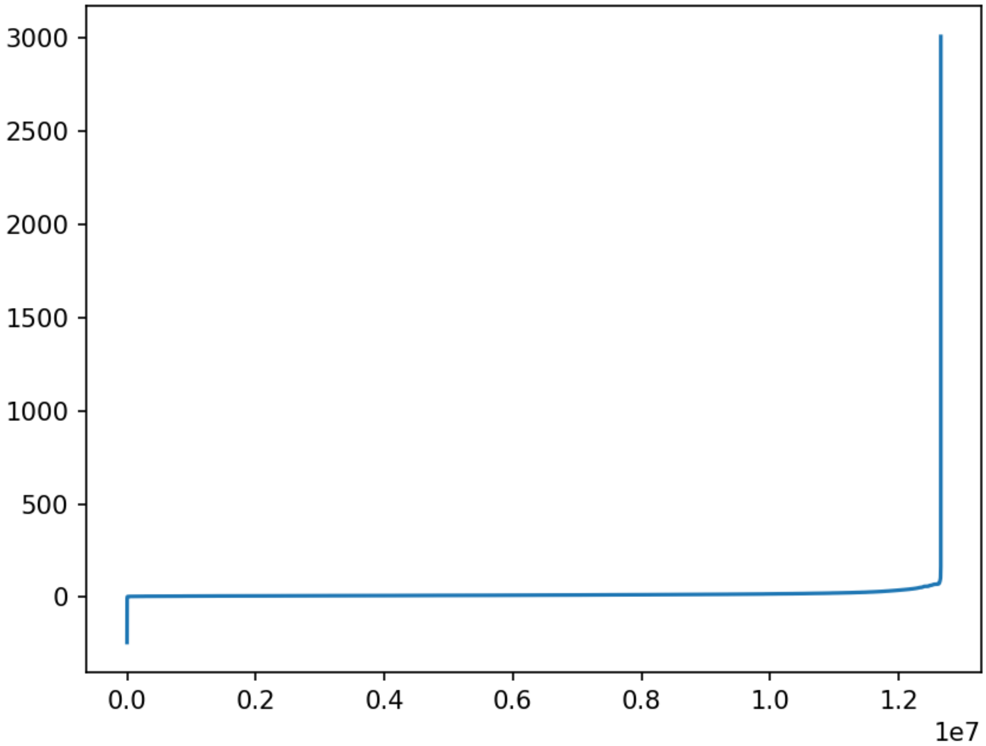
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,9
9.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100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

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

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

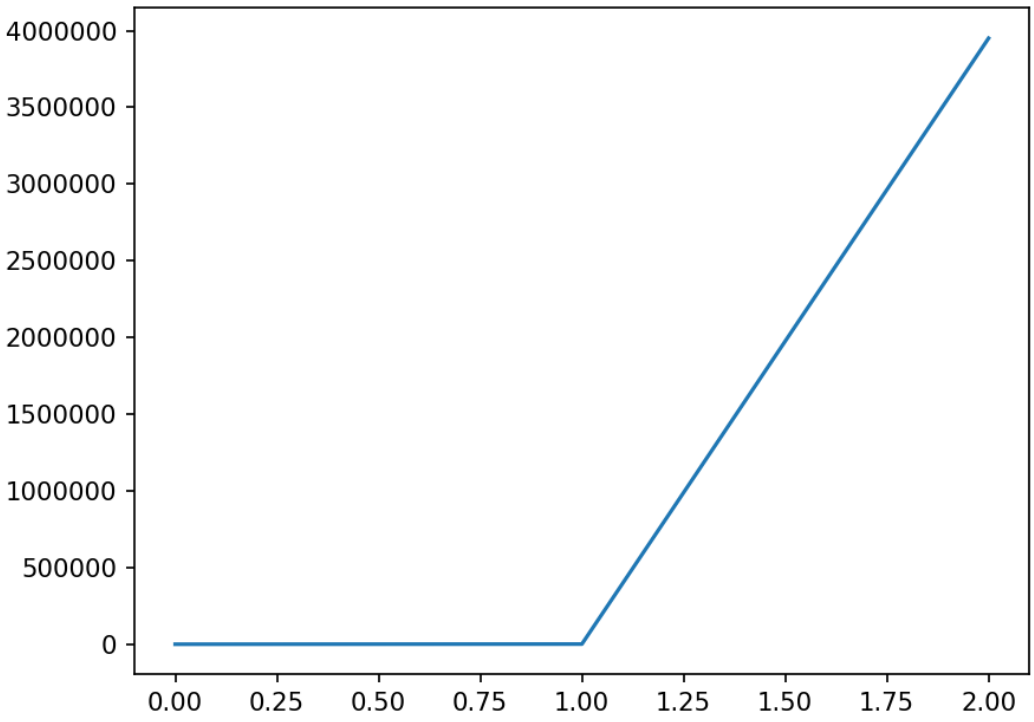
In [40]:

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



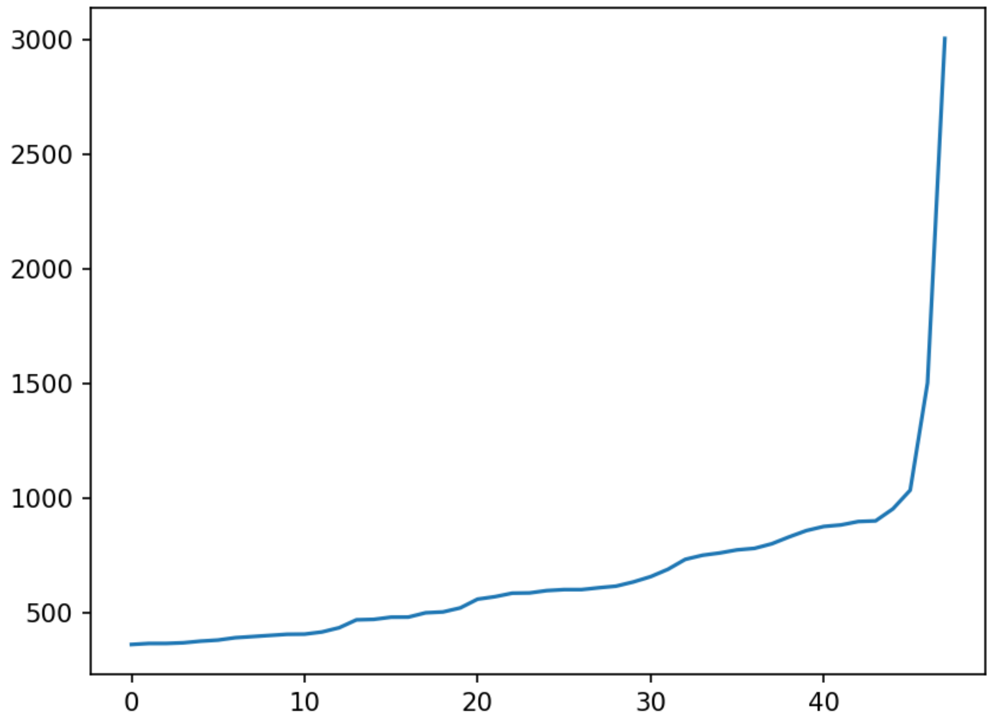
In [41]:

```
# 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 [42]:

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



Remove all outliers/erronous points.

In [43]:

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


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


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


    temp_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 2
3)]
    d = temp_frame.shape[0]
    print ("Number of outliers from trip distance analysis:",(a-d))

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

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


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

    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)]
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distance < 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 [44]:

```
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_wi
th_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 0.970357642560
7495
```

Data-preparation

Clustering/Segmentation

In [45]:

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

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

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

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

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 8.0
Min inter-cluster distance =  1.0945442325142662
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 16.0
Min inter-cluster distance =  0.7131298007388065
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 22.0
Min inter-cluster distance =  0.5185088176172186
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 32.0
Min inter-cluster distance =  0.5069768450365043
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 38.0
Min inter-cluster distance =  0.365363025983583 83
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 46.0
Min inter-cluster distance =  0.34704283494173577
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 54.0
Min inter-cluster distance =  0.30502203163245994
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 62.0
Min inter-cluster distance =  0.292203245317388
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance <
2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance >
2): 69.0
```

```
Min inter-cluster distance =  0.182579928570332733
---
```

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 [46]:

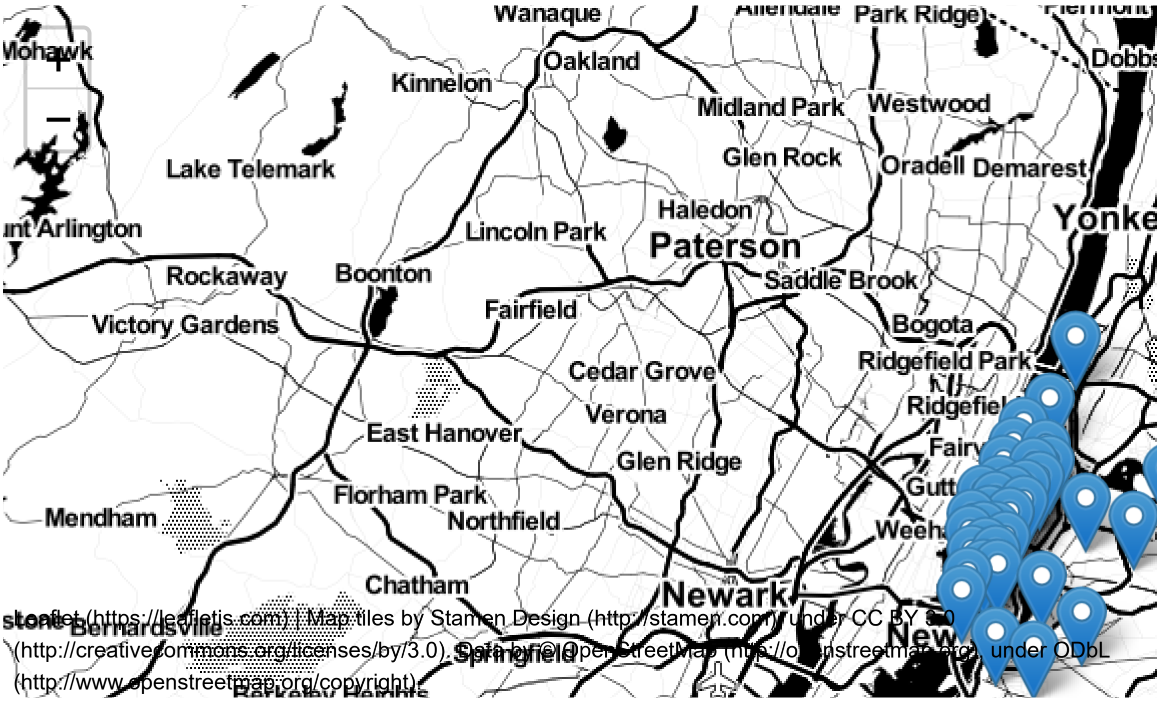
```
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(coords)
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_dur
ations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
```

Plotting the cluster centers:

In [47]:

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

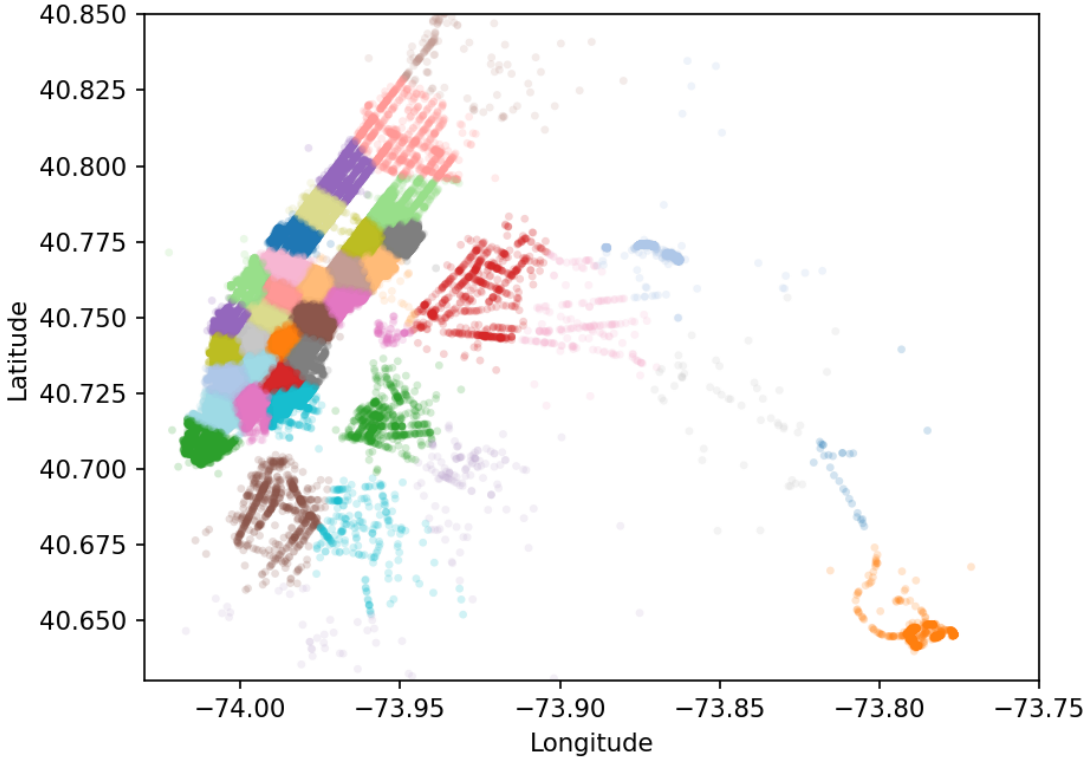
Out[47]:



Plotting the clusters:

In [48]:

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

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [49]:

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

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

In [50]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with_dur
ations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
jan_2015_frame = add_pickup_bins(frame_with_durations_outliers_removed,1,2015)
jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance']].gro
upby(['pickup_cluster','pickup_bins']).count()
```

In [51]:

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

Out[51]:

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

In [52]:

```
jan_2015_groupby.head()
```

Out[52]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**1**	105
	2	199
	3	208
	4	141
	5	155

In [53]:

```
def datapreparation(month,kmeans,month_no,year_no):

    print ("Return with trip times..")

    frame_with_durations = return_with_trip_times(month)

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

    print ("Estimating clusters..")
    frame_with_durations_outliers_removed['pickup_cluster'] = kmeans.predict(frame_with
_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predict(fram
e_with_durations_outliers_removed_2016[['pickup_latitude', 'pickup_longitude']])

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_n
o,year_no)
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip_dis
tance']].groupby(['pickup_cluster','pickup_bins']).count()

    return final_updated_frame,final_groupby_frame

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

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

```
Return with trip times..
Remove outliers..
Number of pickup records =  10906858
Number of outlier coordinates lying outside NY boundaries: 214677
Number of outliers from trip times analysis: 27190
Number of outliers from trip distance analysis: 79742
Number of outliers from speed analysis: 21047
Number of outliers from fare analysis: 4991
Total outliers removed 297784
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  11382049
Number of outlier coordinates lying outside NY boundaries: 223161
Number of outliers from trip times analysis: 27670
Number of outliers from trip distance analysis: 81902
Number of outliers from speed analysis: 22437
Number of outliers from fare analysis: 5476
Total outliers removed 308177
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  12210952
Number of outlier coordinates lying outside NY boundaries: 232444
Number of outliers from trip times analysis: 30868
Number of outliers from trip distance analysis: 87318
Number of outliers from speed analysis: 23889
Number of outliers from fare analysis: 5859
Total outliers removed 324635
---
Estimating clusters..
Final groupbying..
```

In [54]:

```
month_jan_2016.head()
```

Out[54]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_di
0	2	2016-01-01 00:00:00	2016-01-01 00:00:00	2	1.10
1	2	2016-01-01 00:00:00	2016-01-01 00:00:00	5	4.90
2	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	10.54
3	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	4.75
4	2	2016-01-01 00:00:00	2016-01-01 00:00:00	3	1.76

In [55]:

```
month_feb_2016.head()
```

Out[55]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_di
0	2	2016-02-25 17:24:20	2016-02-25 17:27:20	2	0.70
1	2	2016-02-25 23:10:50	2016-02-25 23:31:50	2	5.52
2	2	2016-02-01 00:00:01	2016-02-01 00:10:52	6	1.99
3	1	2016-02-01 00:00:04	2016-02-01 00:05:16	1	1.50
4	2	2016-02-01 00:00:05	2016-02-01 00:20:59	1	5.60

In [56]:

```
month_mar_2016.head()
```

Out[56]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_di
0	1	2016-03-01 00:00:00	2016-03-01 00:07:55	1	2.50
1	1	2016-03-01 00:00:00	2016-03-01 00:11:06	1	2.90
2	2	2016-03-01 00:00:00	2016-03-01 00:31:06	2	19.98
3	2	2016-03-01 00:00:00	2016-03-01 00:00:00	3	10.78
4	2	2016-03-01 00:00:00	2016-03-01 00:00:00	5	30.43

Smoothing

In [57]:

```
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 [58]:

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

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

#feb
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [59]:

```
# for each cluster number of 10min intravels with 0 pickups
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",4464 -
len(set(jan_2015_unique[i])))
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1986
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  30
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  355
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  154
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  35
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  118
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  45
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  43
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  27
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  32
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  59
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  1191
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1358
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  54
------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  30
------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  30
------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  164
------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  32
------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  215
------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1181
```

```
----------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  43
----------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  45
----------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  44
----------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  40
----------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  43
----------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  37
----------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  322
----------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  37
----------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  44
----------------------------------------------------------
```

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 [60]:

```
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,40):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [61]:

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

```
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [62]:

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

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

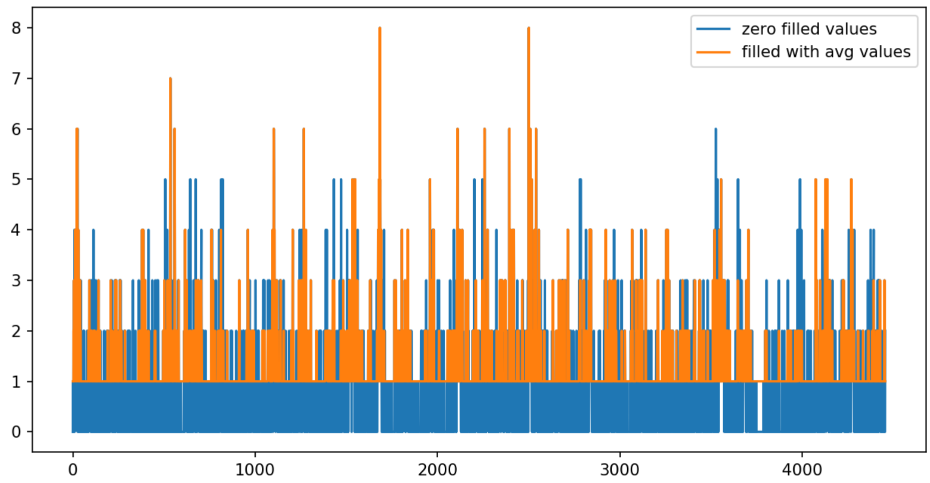
In [63]:

```
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

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

In [64]:

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



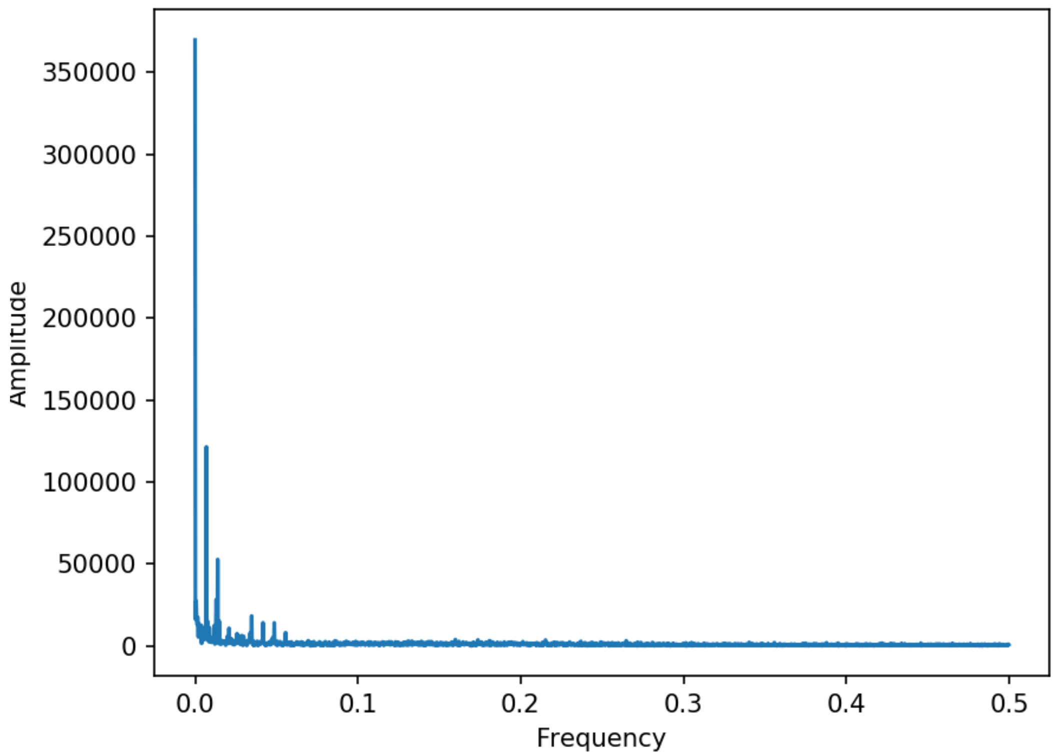
In [66]:

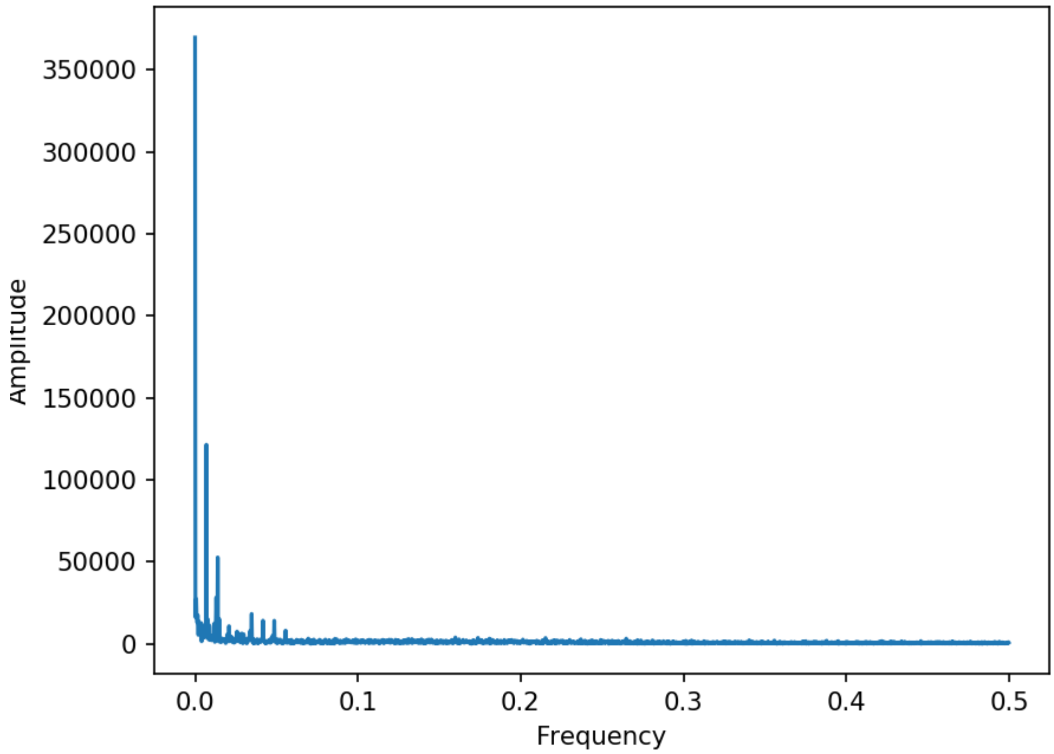
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled with z
ero
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_unique)
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique
)
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_unique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

# Making list of all the values of pickup data in every bin for a period of 3 months an
d storing them region-wise
regions_cum = []
for i in range(0,40):
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_smooth[4176*i:4176*(
i+1)]+mar_2016_smooth[4464*i:4464*(i+1)])
```

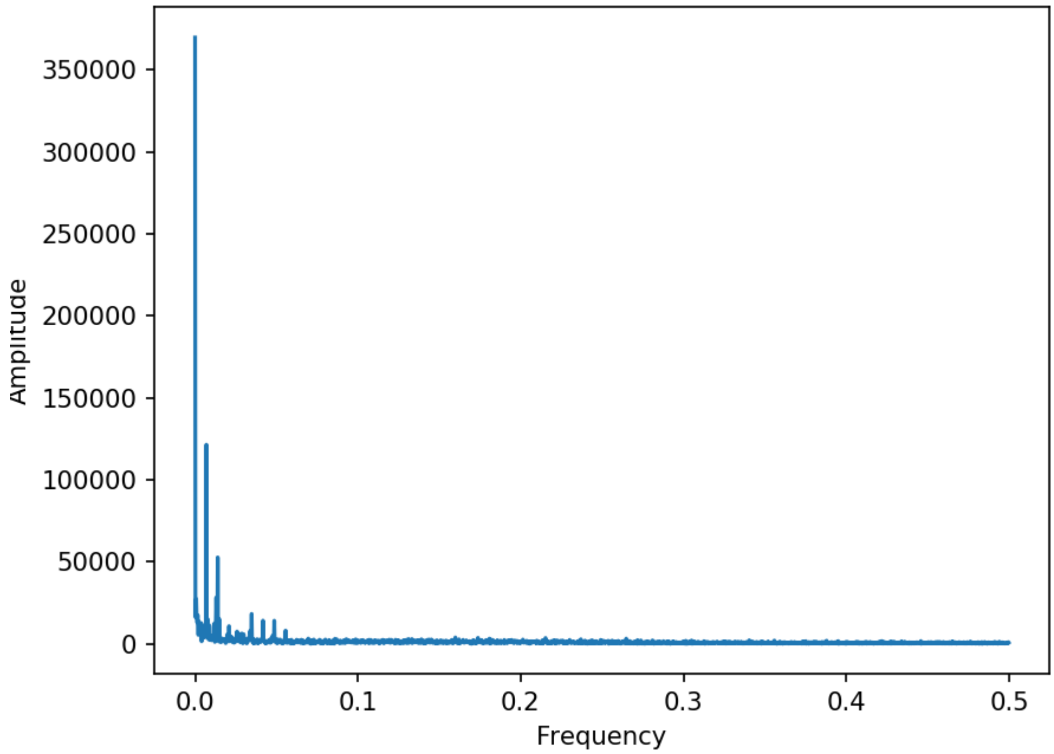
Fourier Transforms

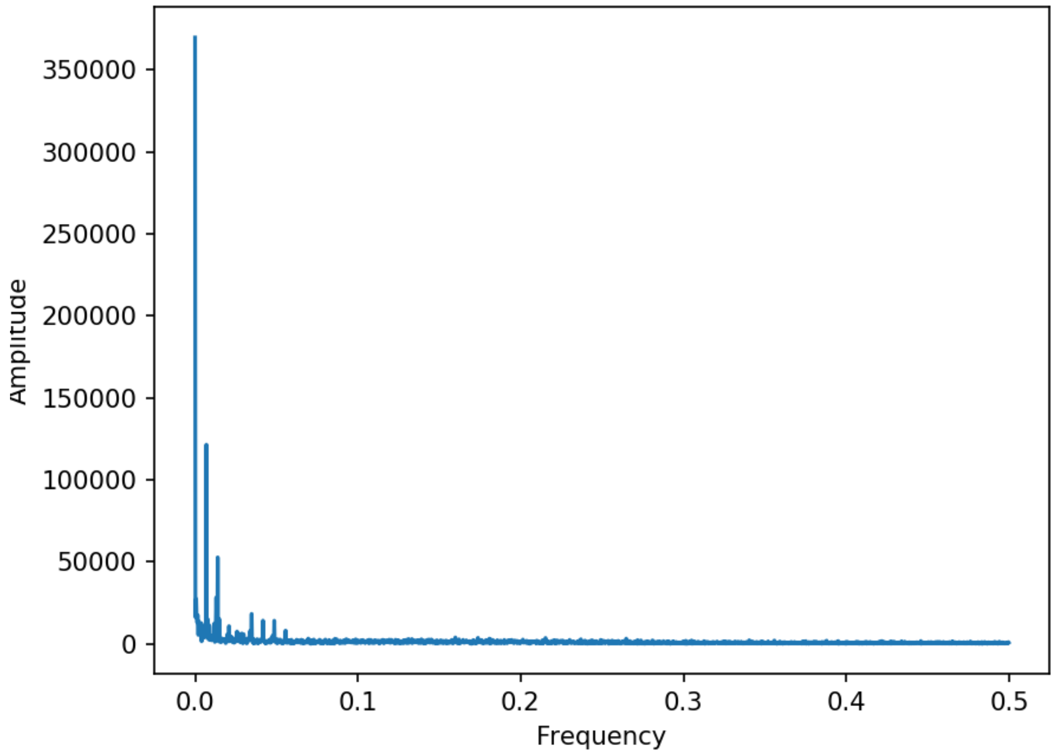
In [67]:

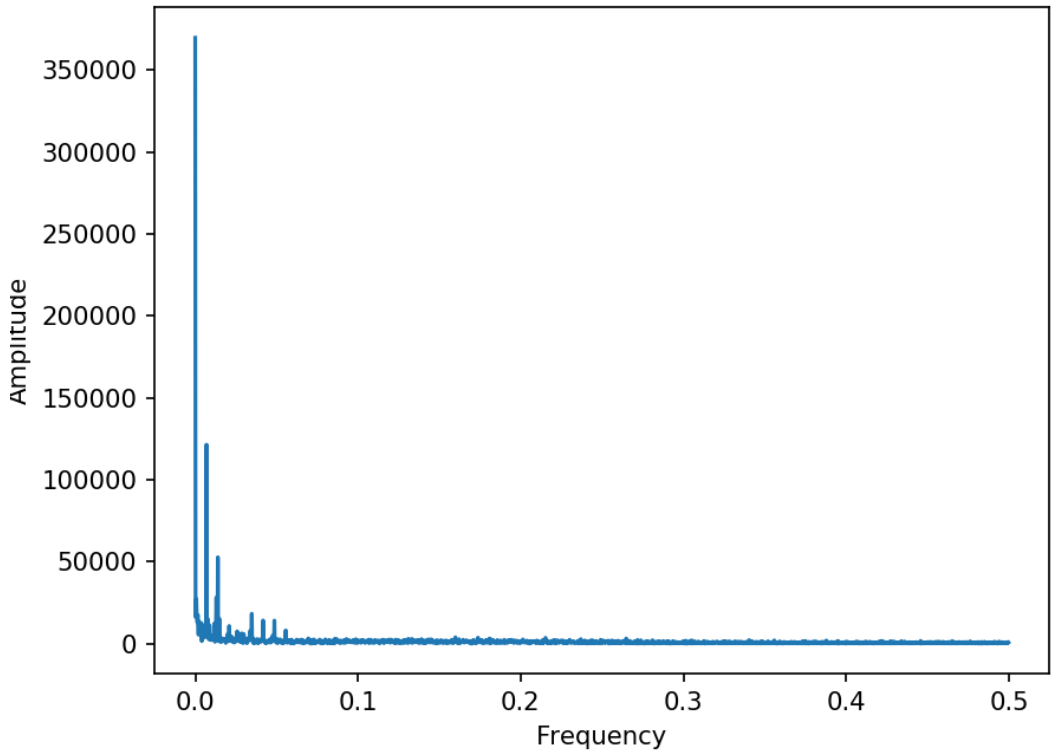
```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/generated/n
umpy.fft.fft.html
for i in range(40):
    Y    = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generated/num
py.fft.fftfreq.html
    freq = np.fft.fftfreq(4460, 1)
    n = len(freq)
    plt.figure()
    plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    plt.show()
```


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

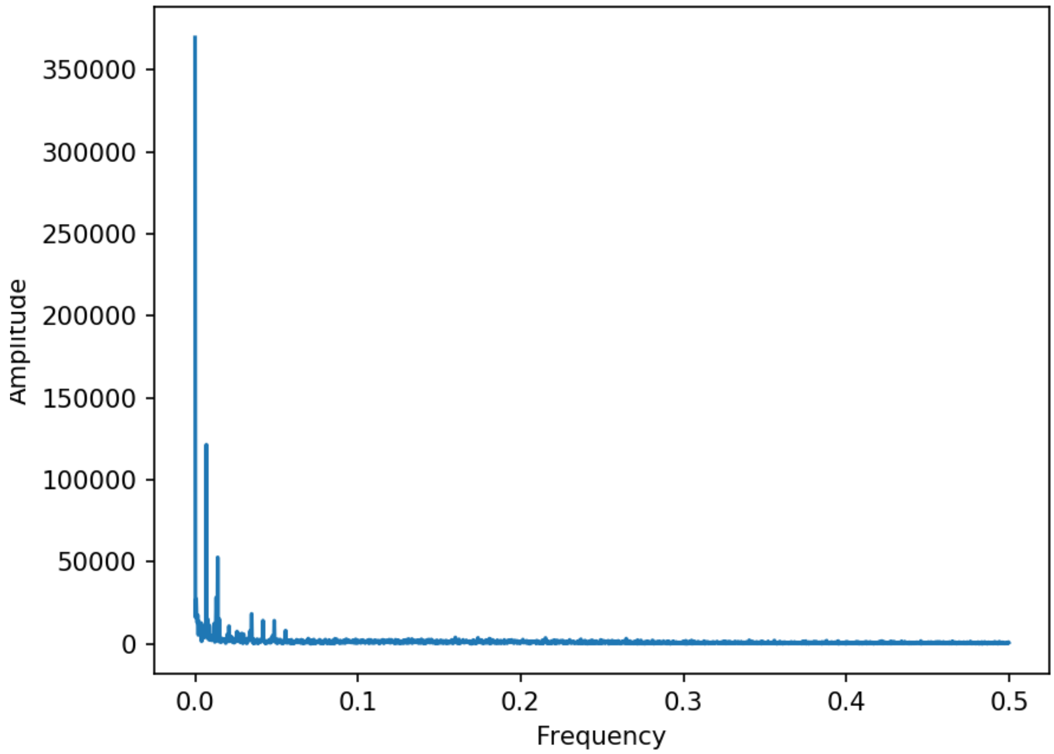

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

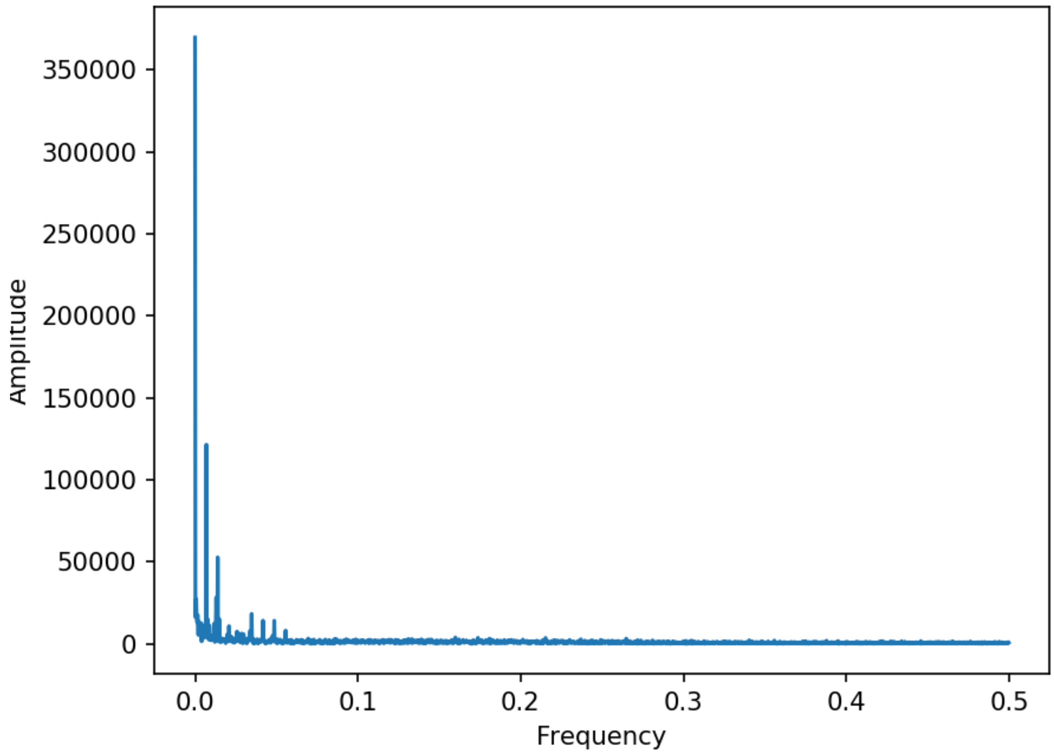

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

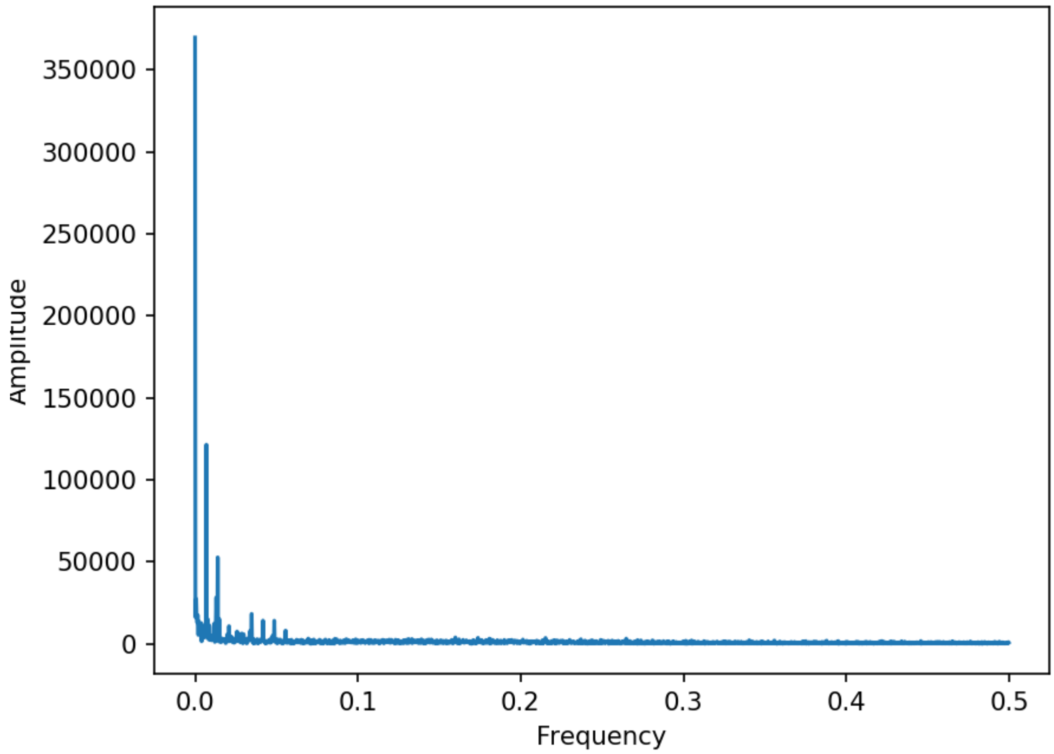

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

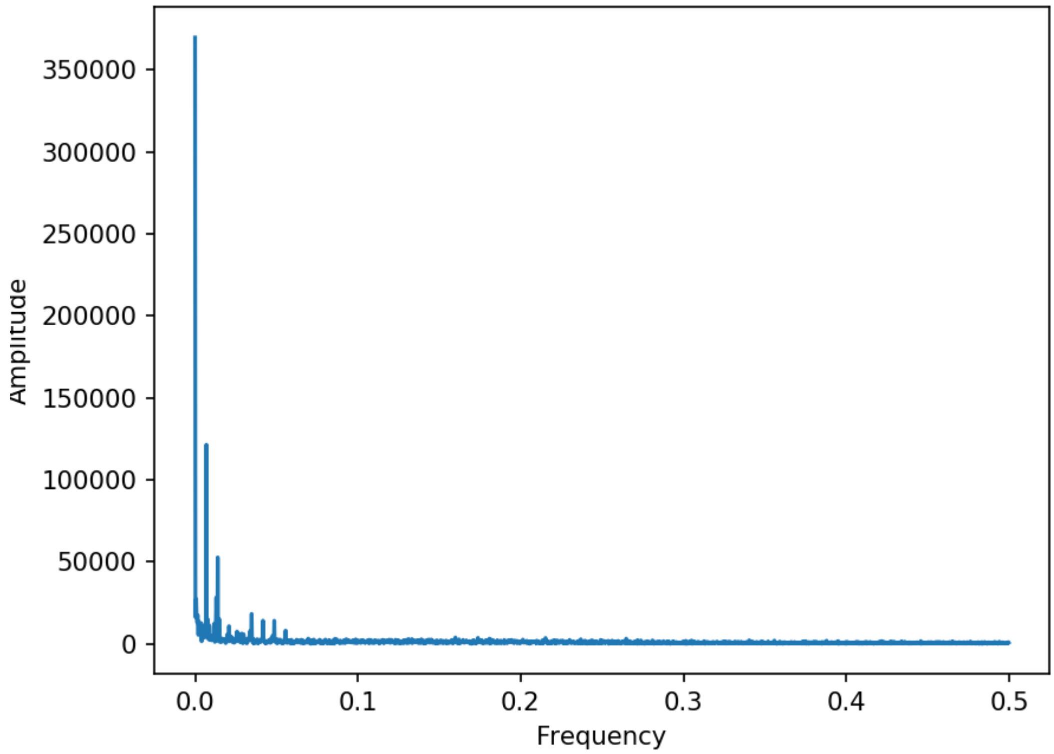


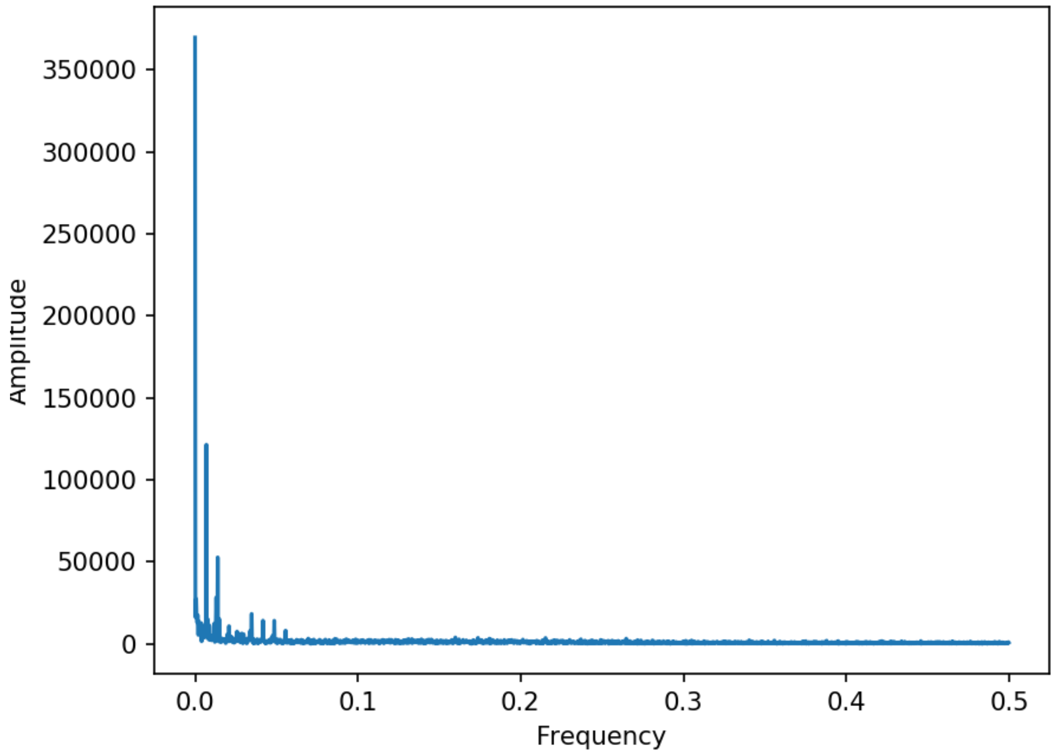
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

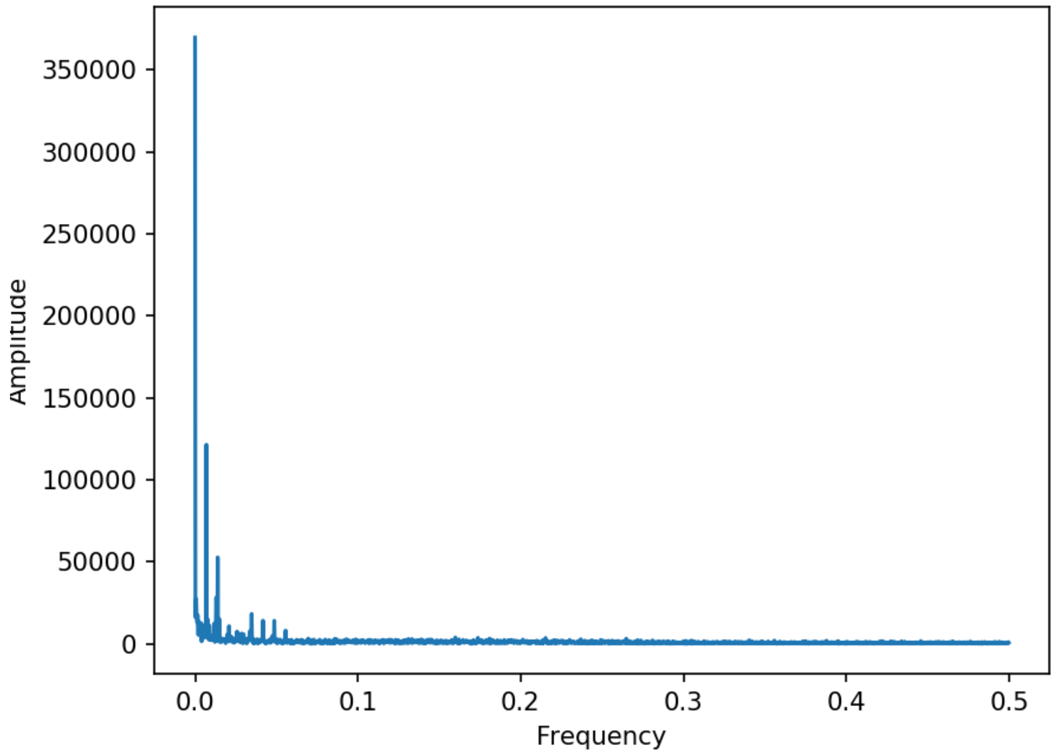

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

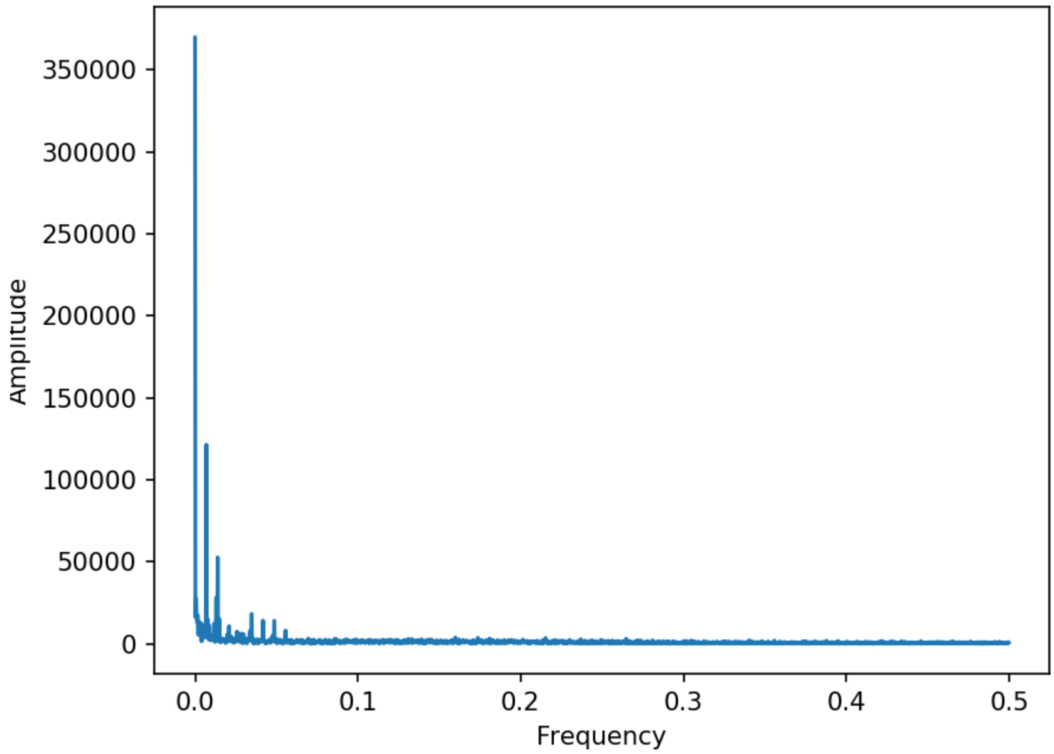

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

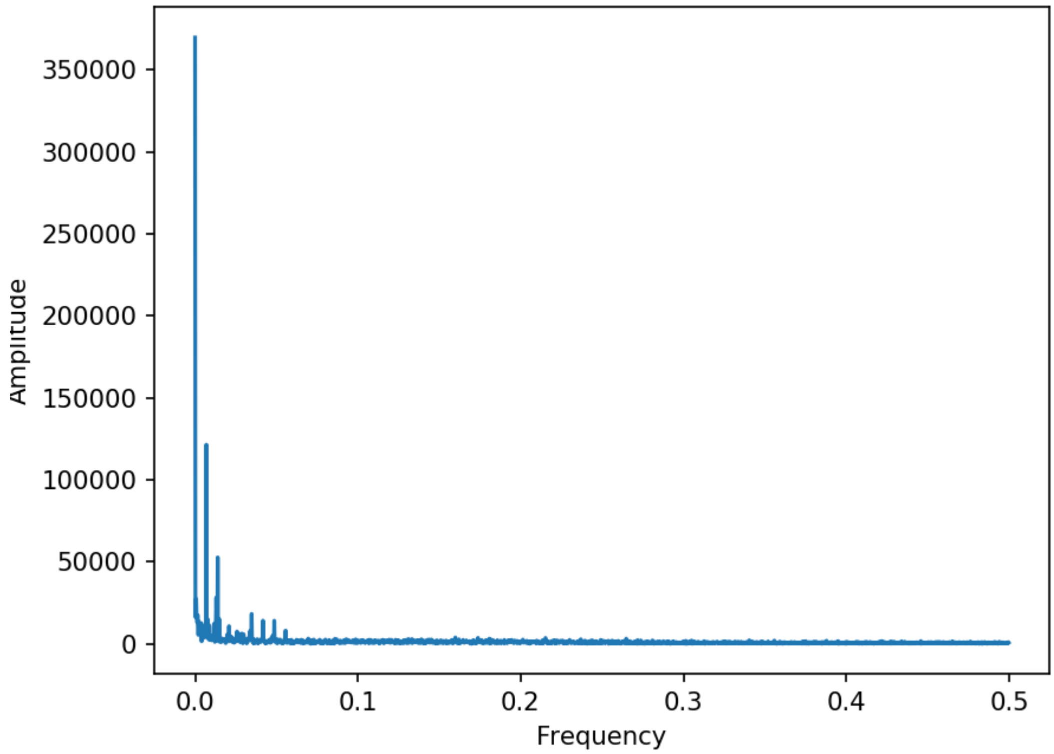

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

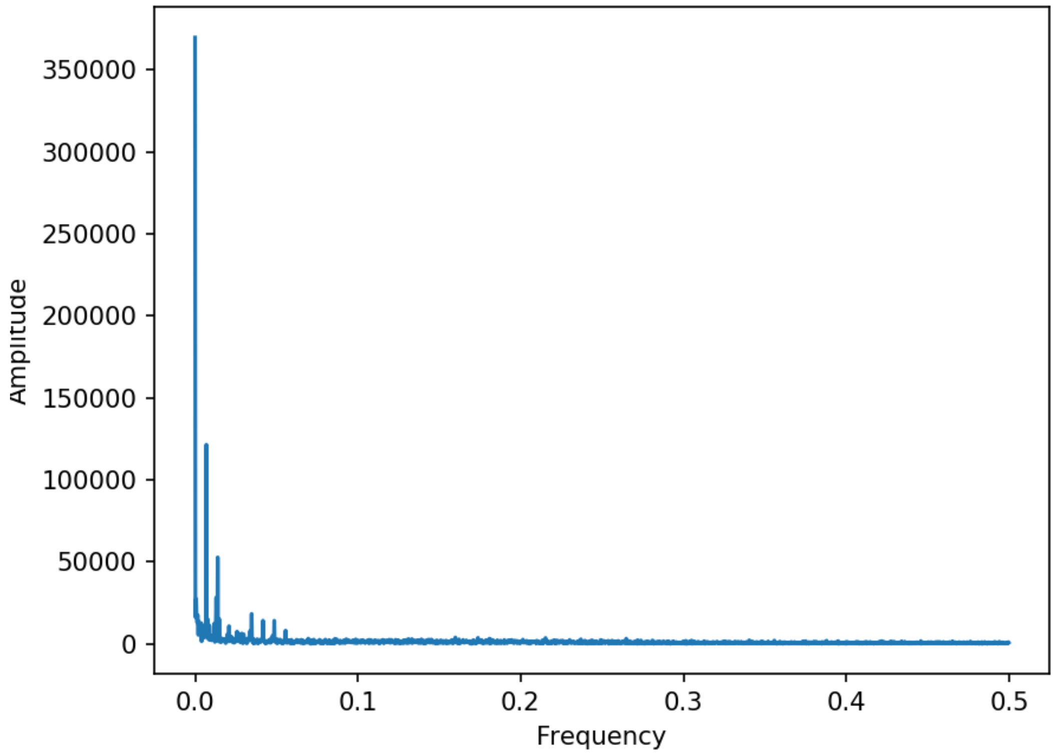

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

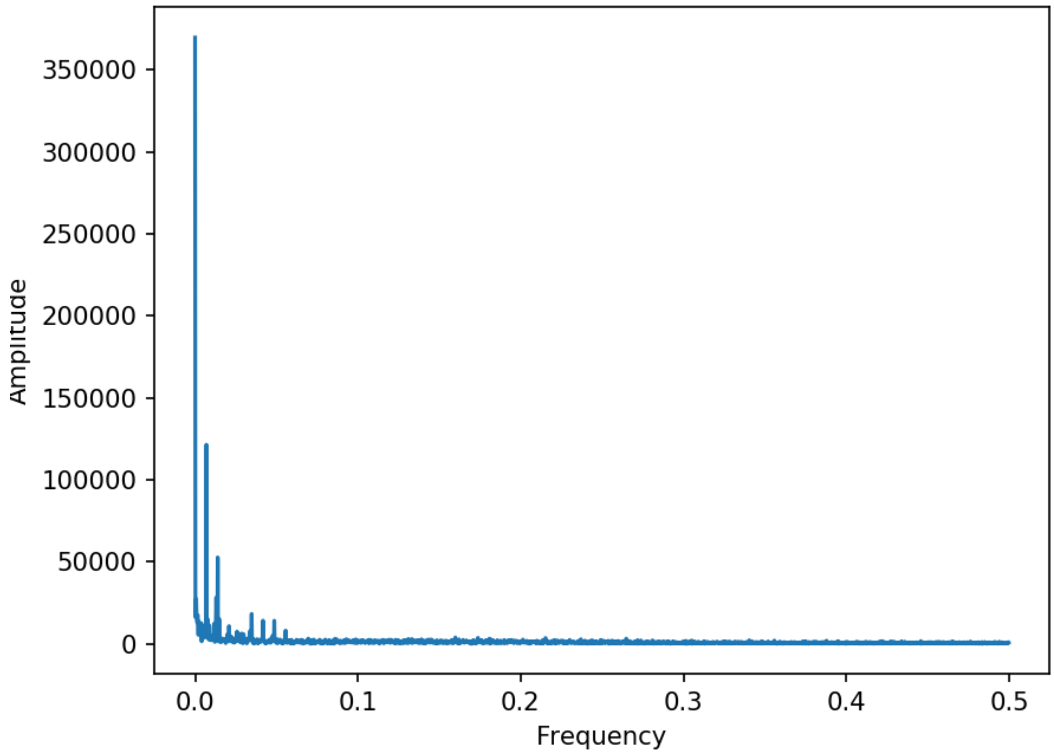

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

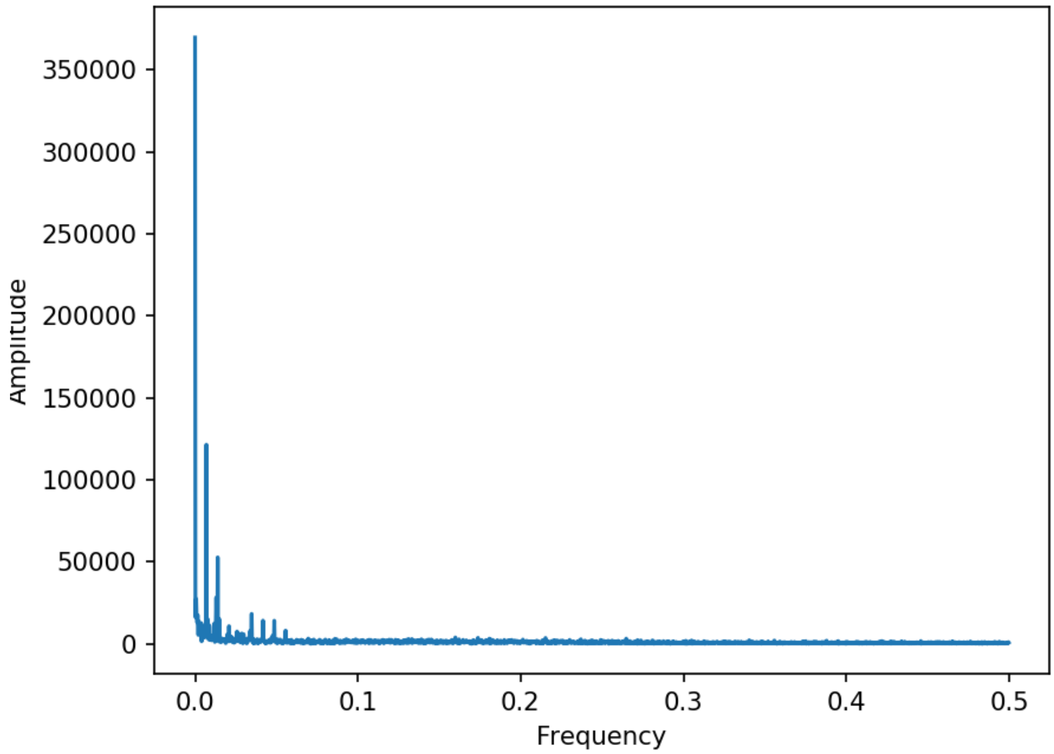

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

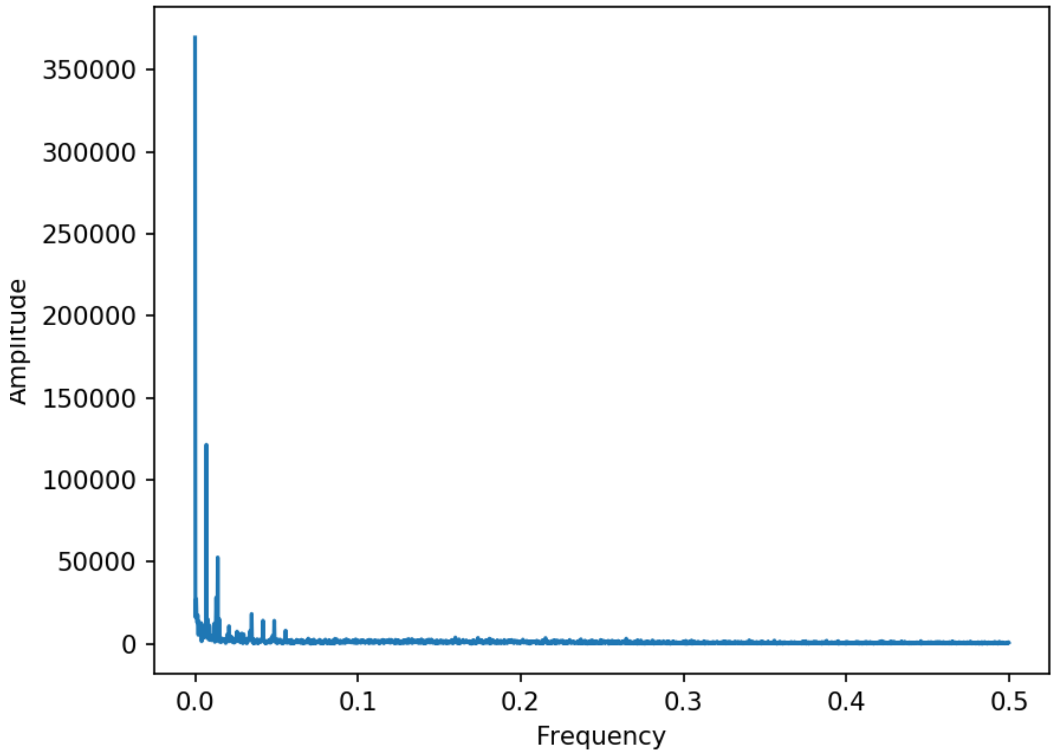

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

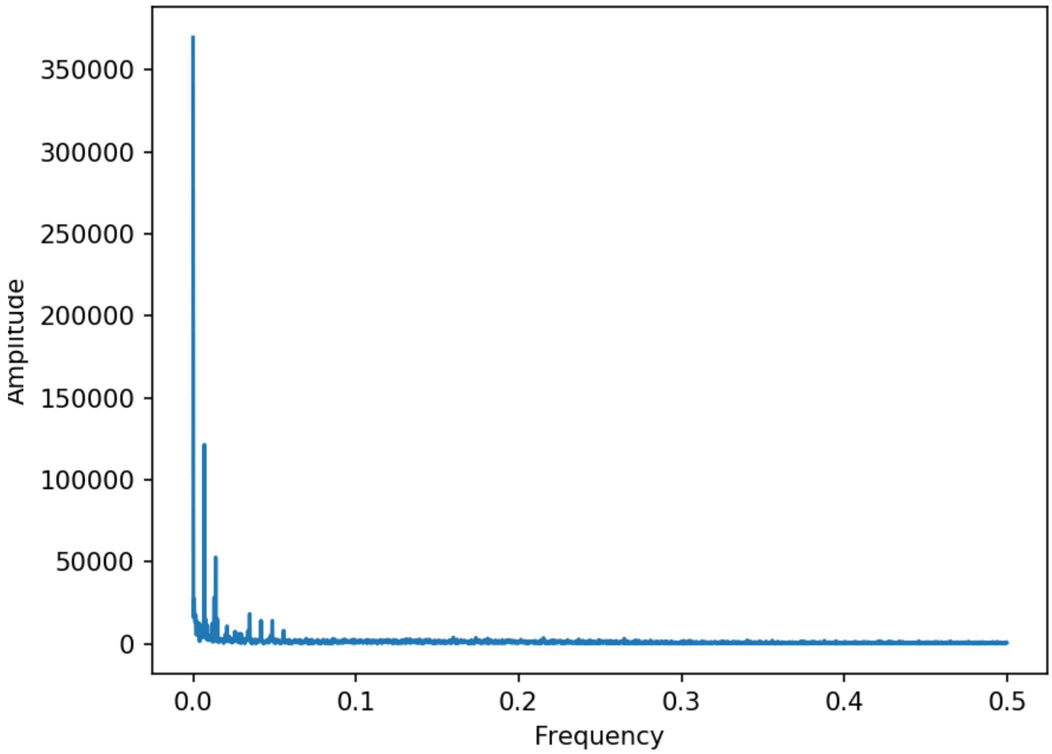

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

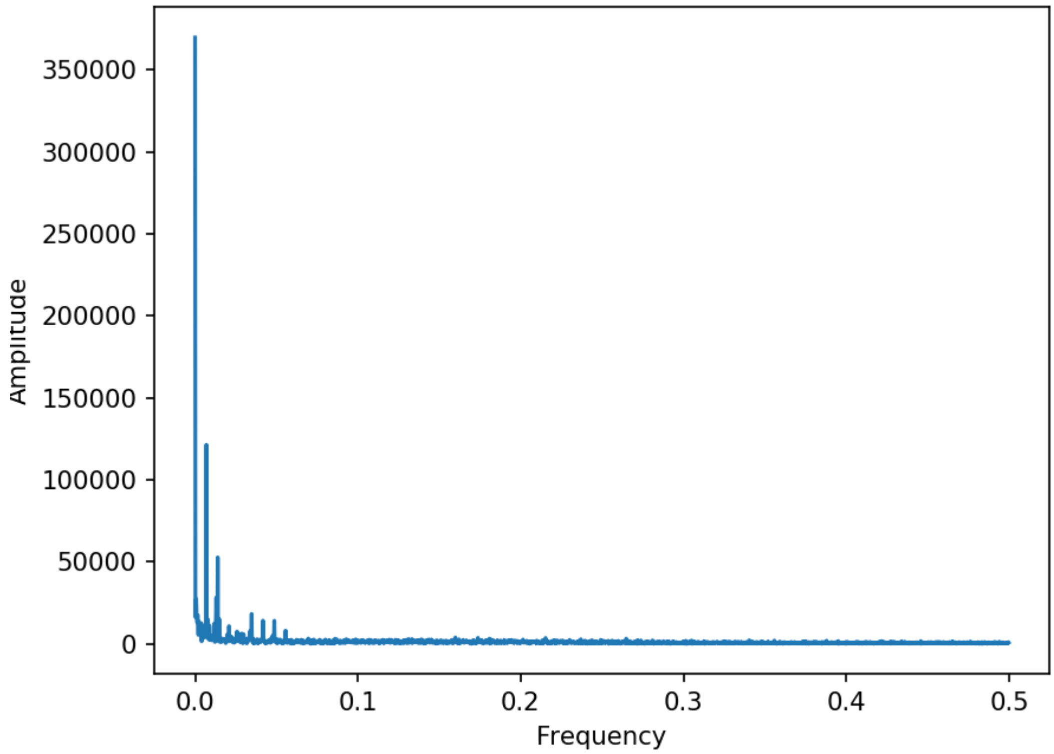


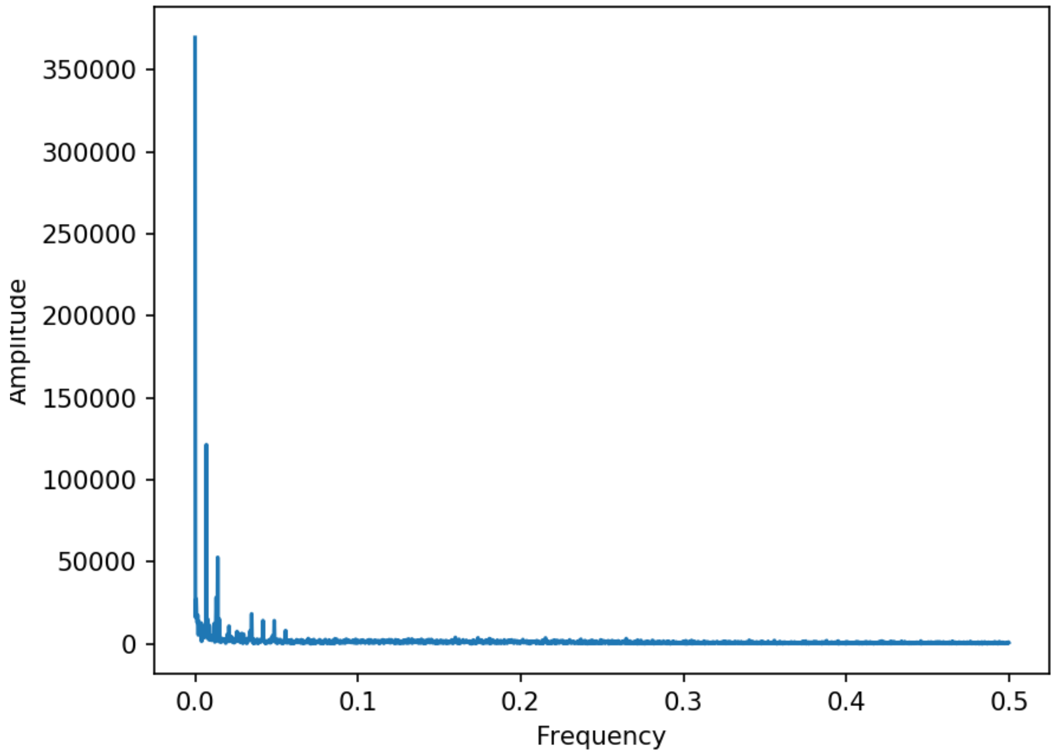
350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

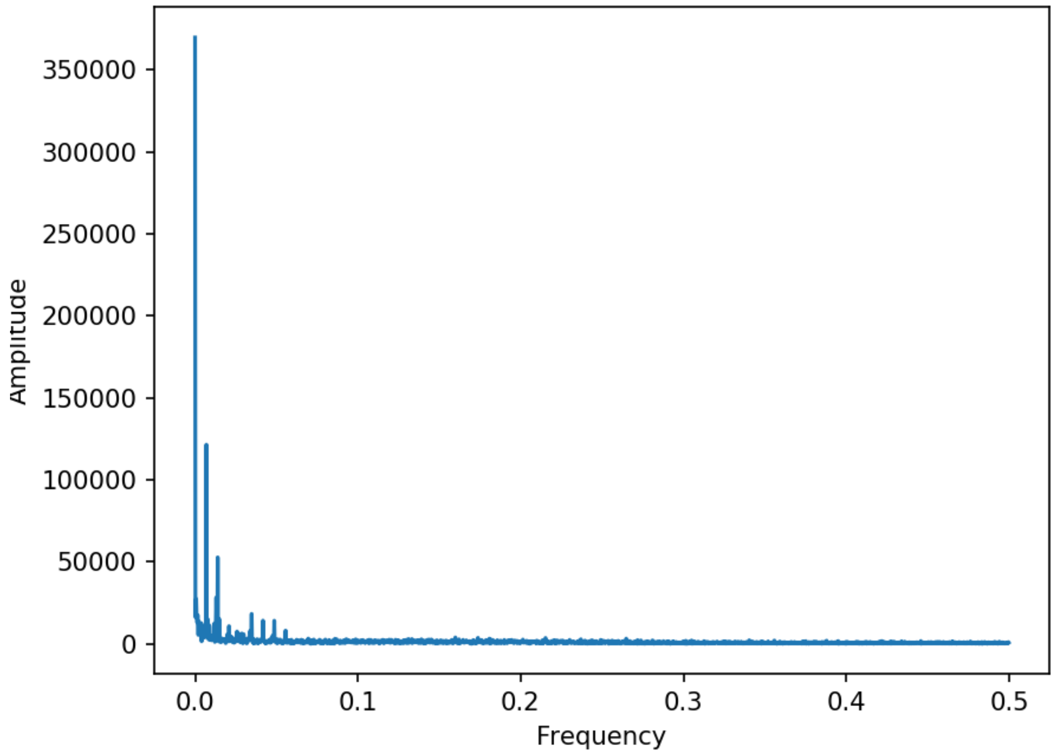

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

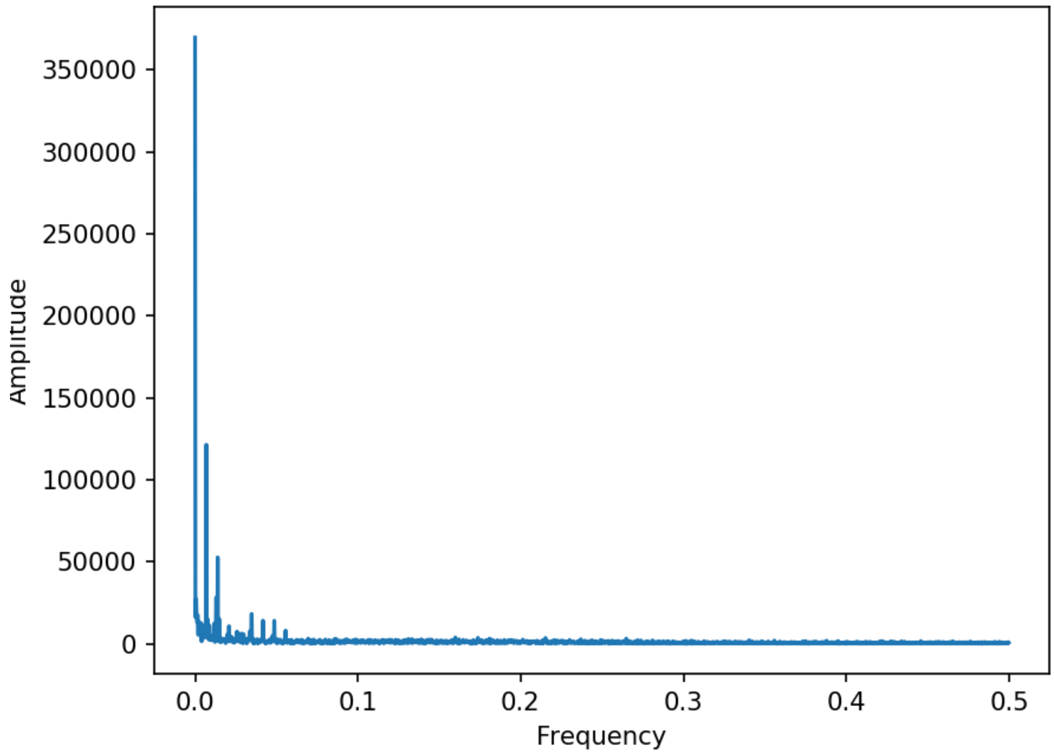

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

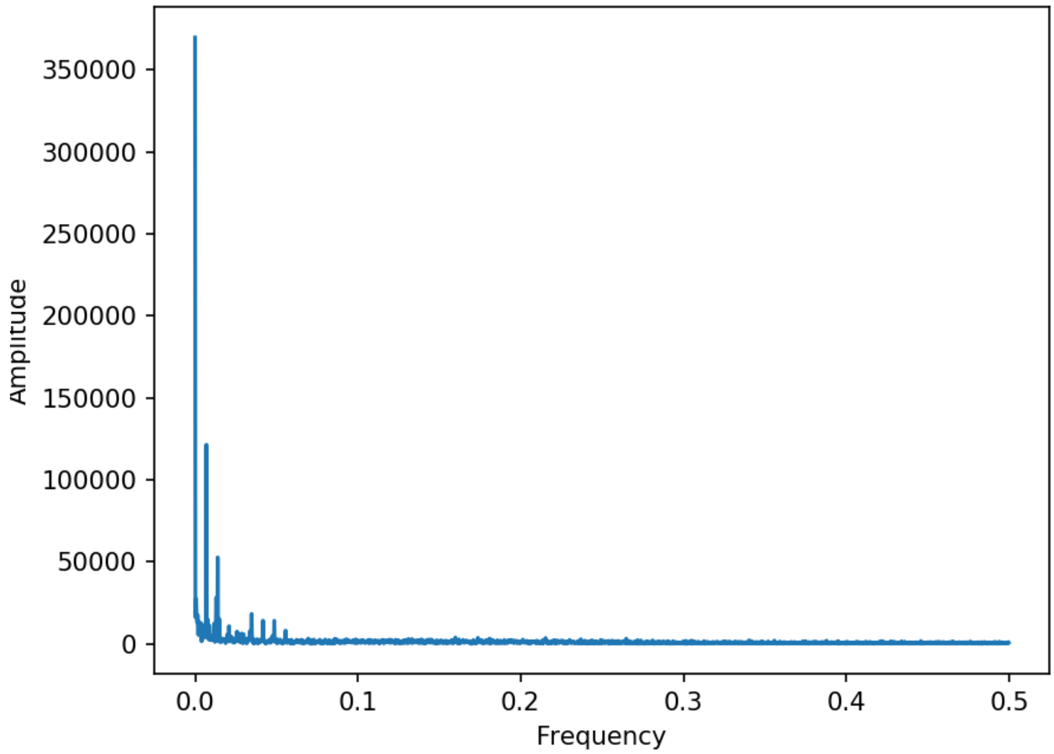

Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

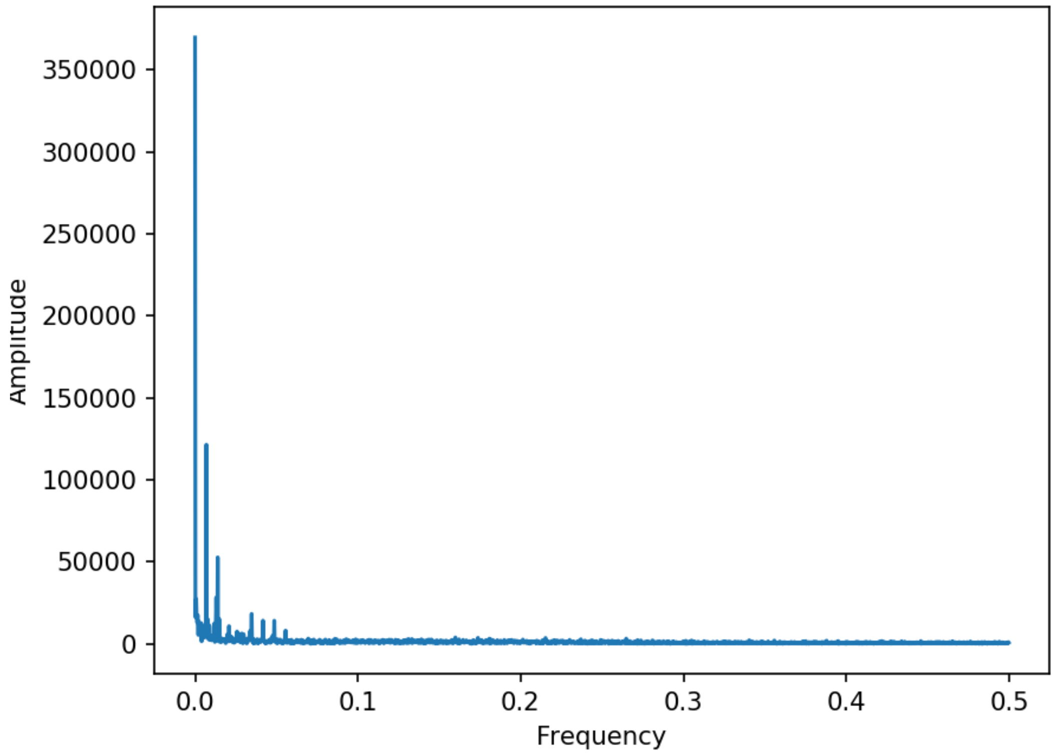

Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

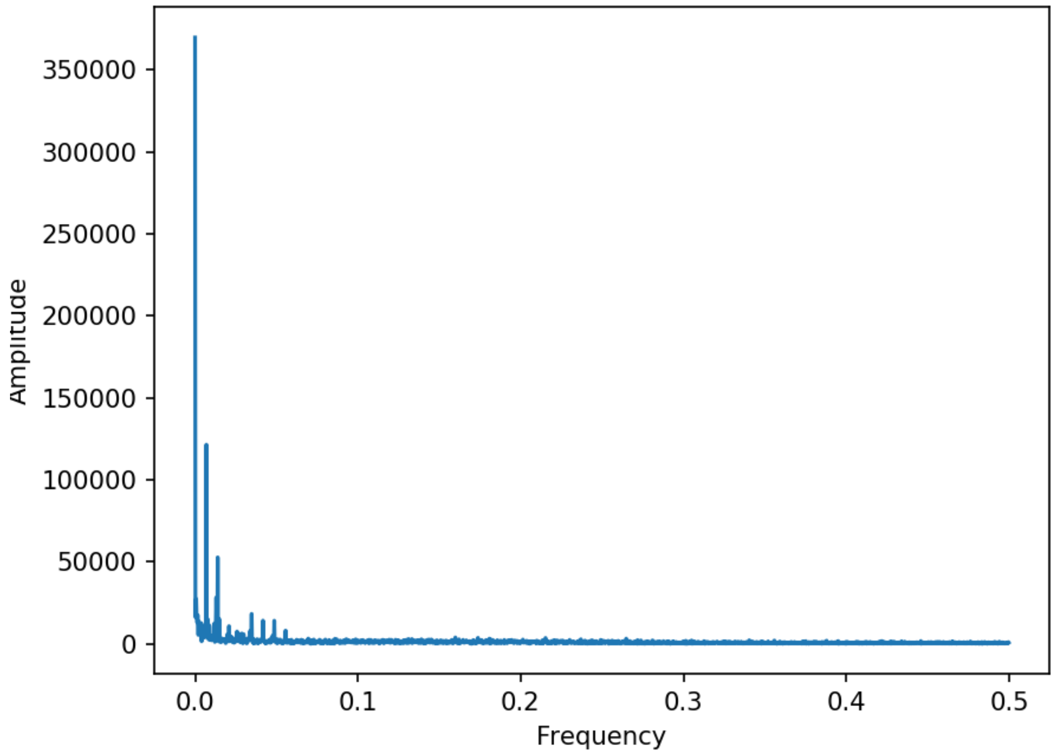

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

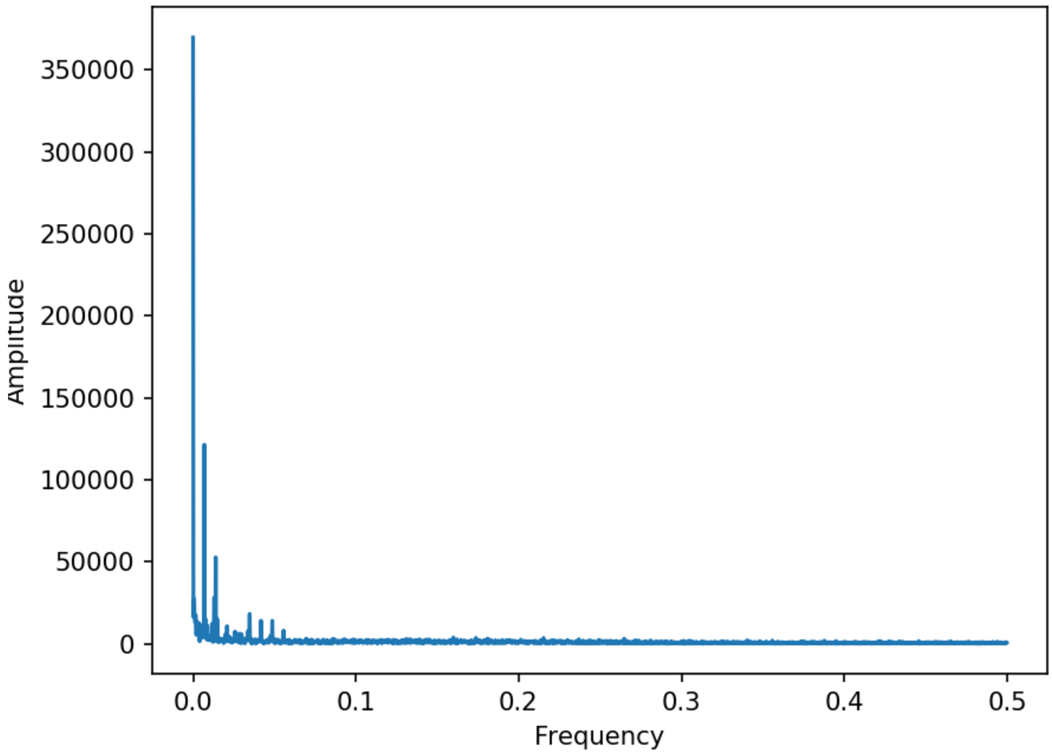

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

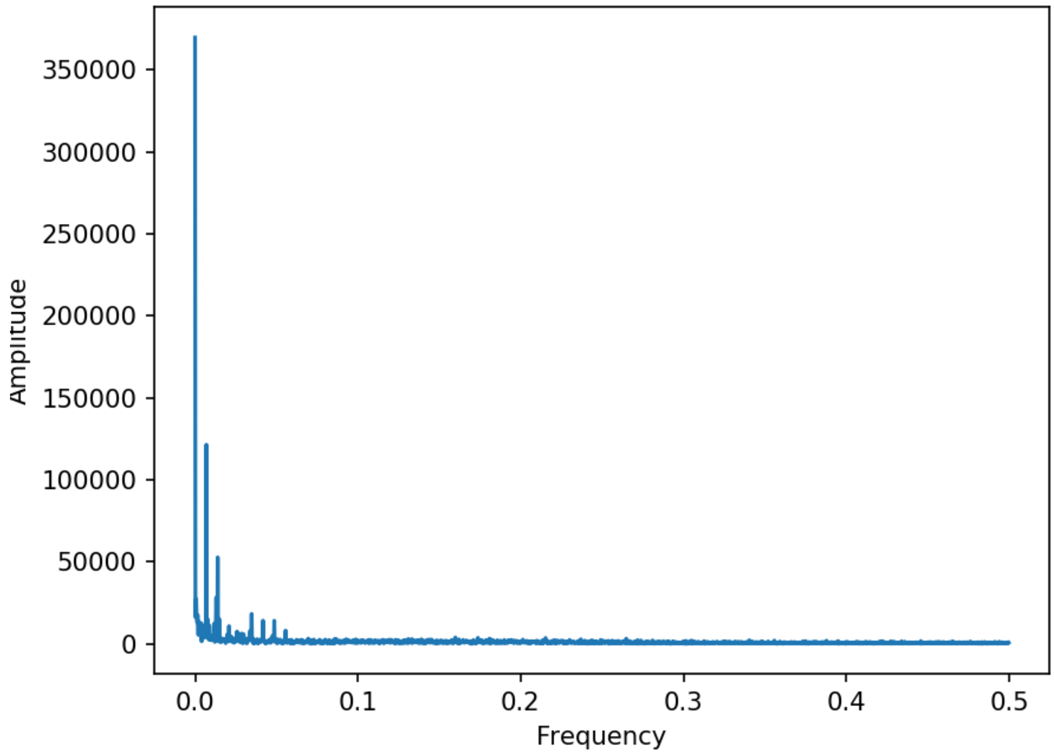

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

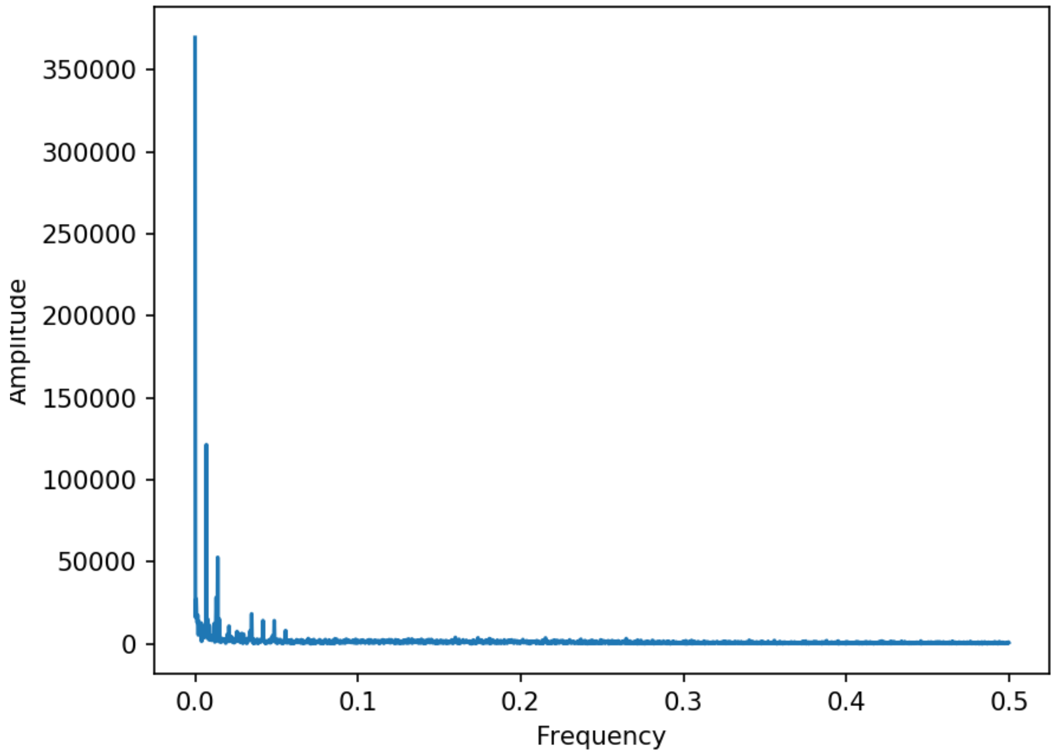

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

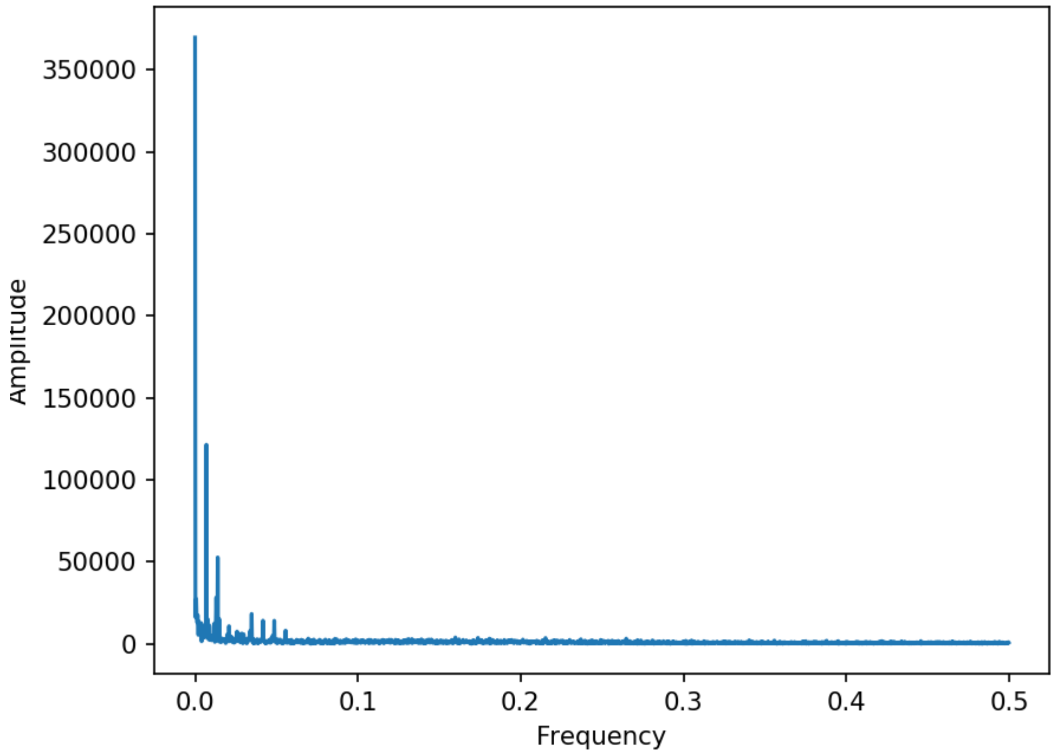

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

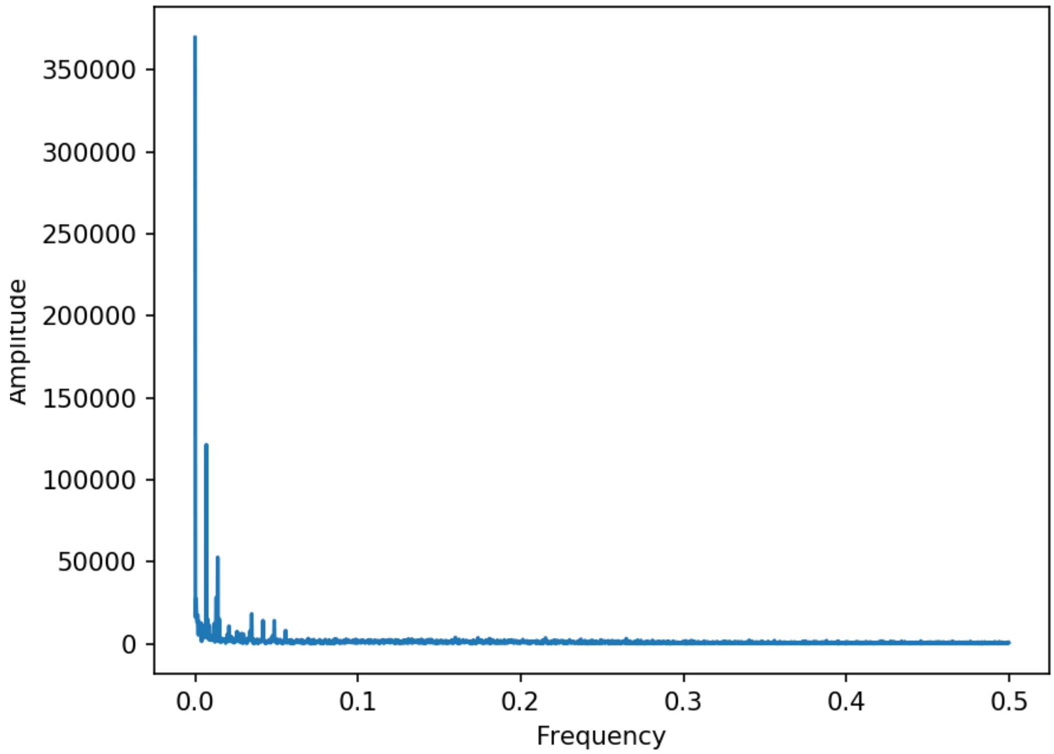


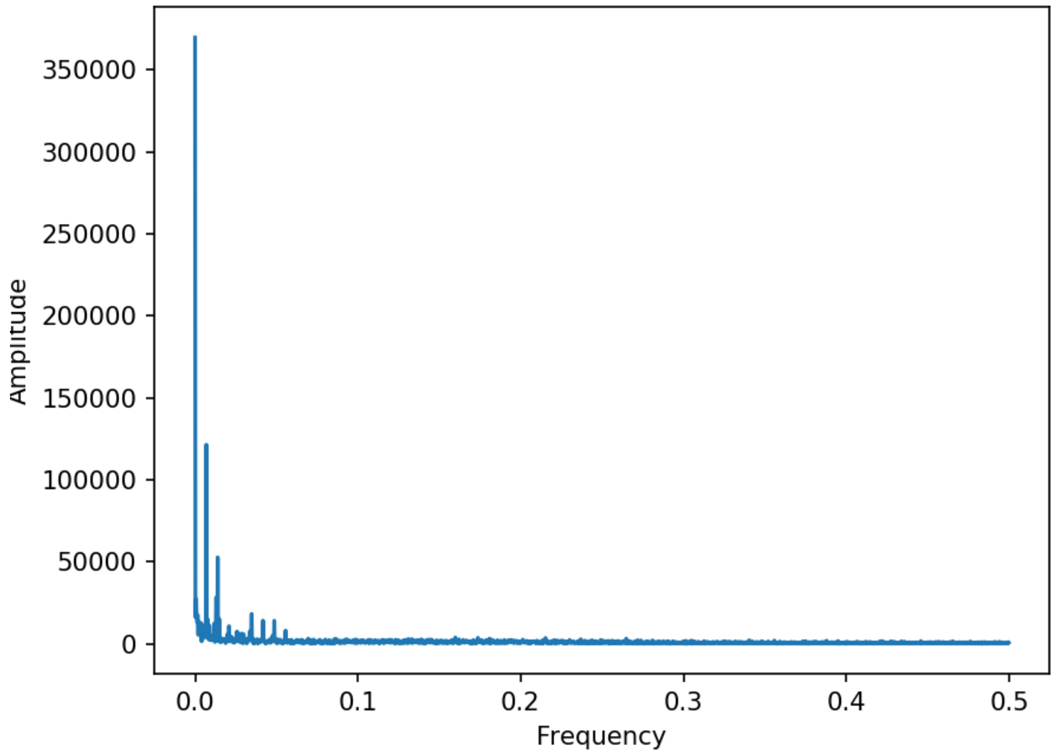
350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

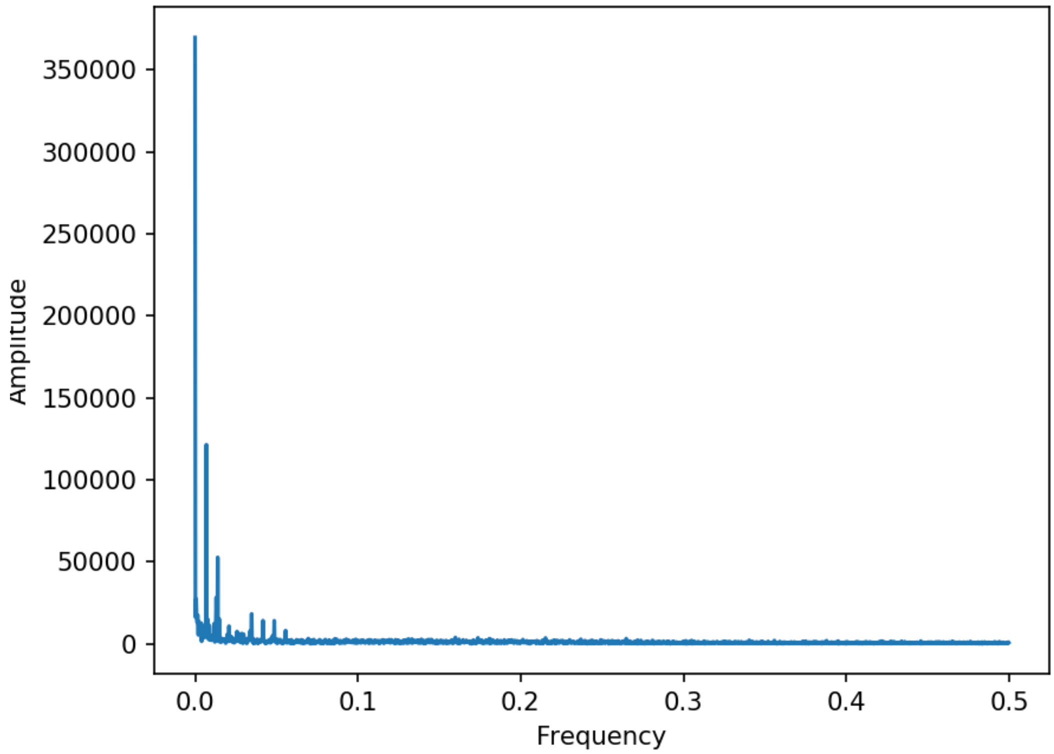

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

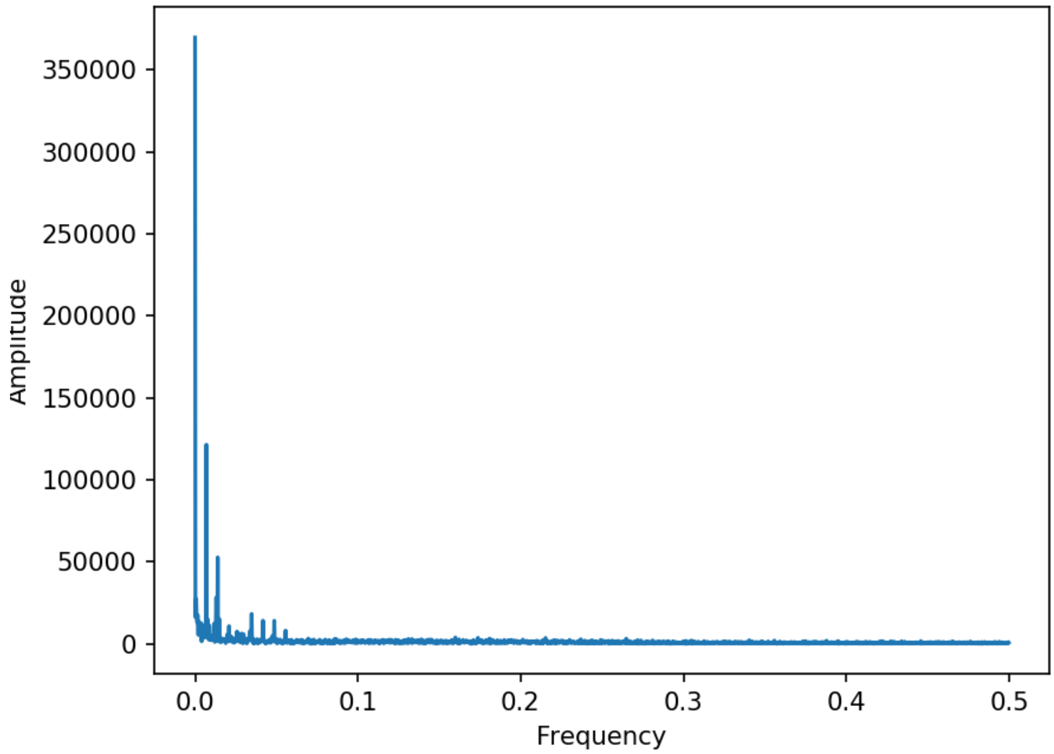

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

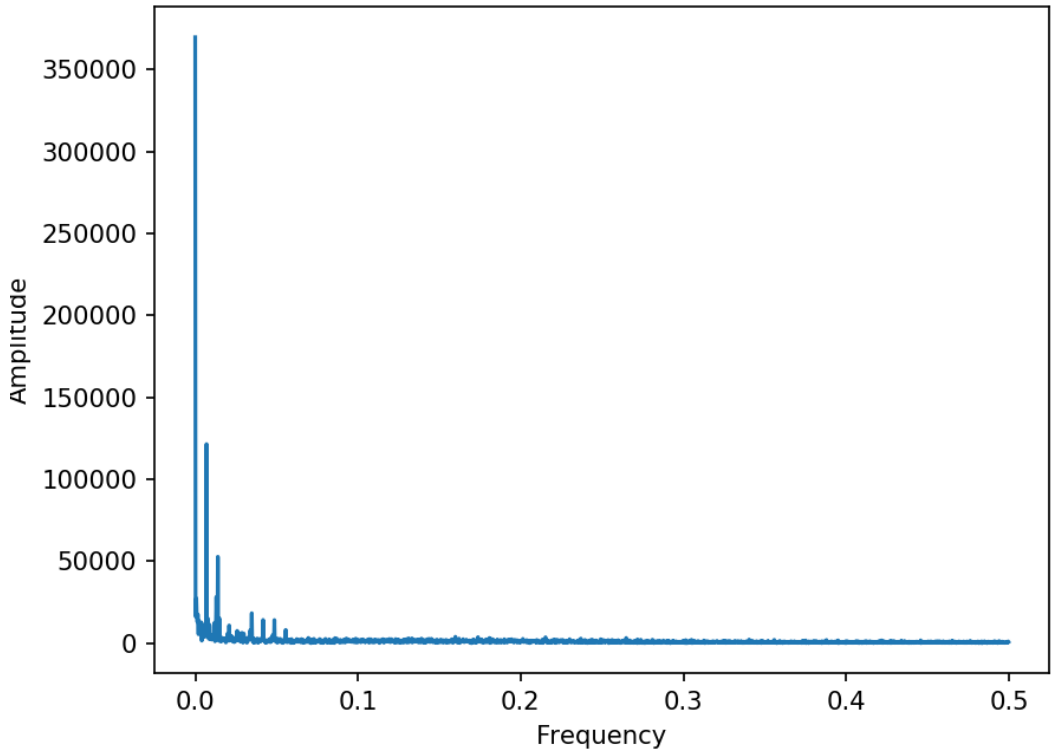

Amplitude
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5

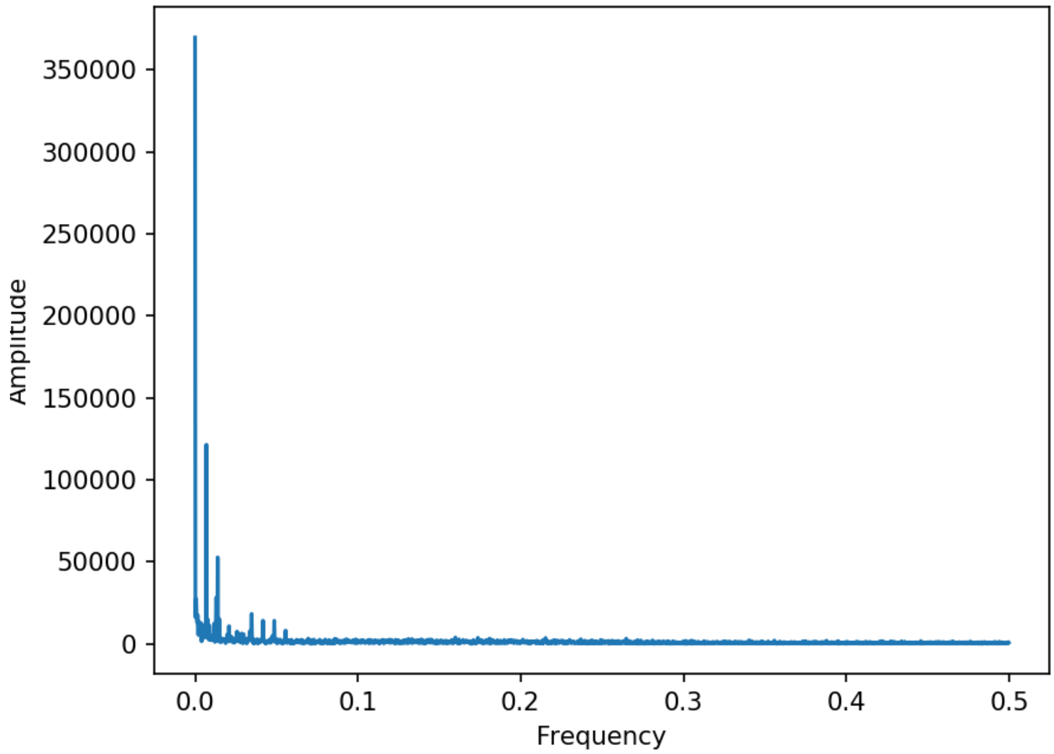

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

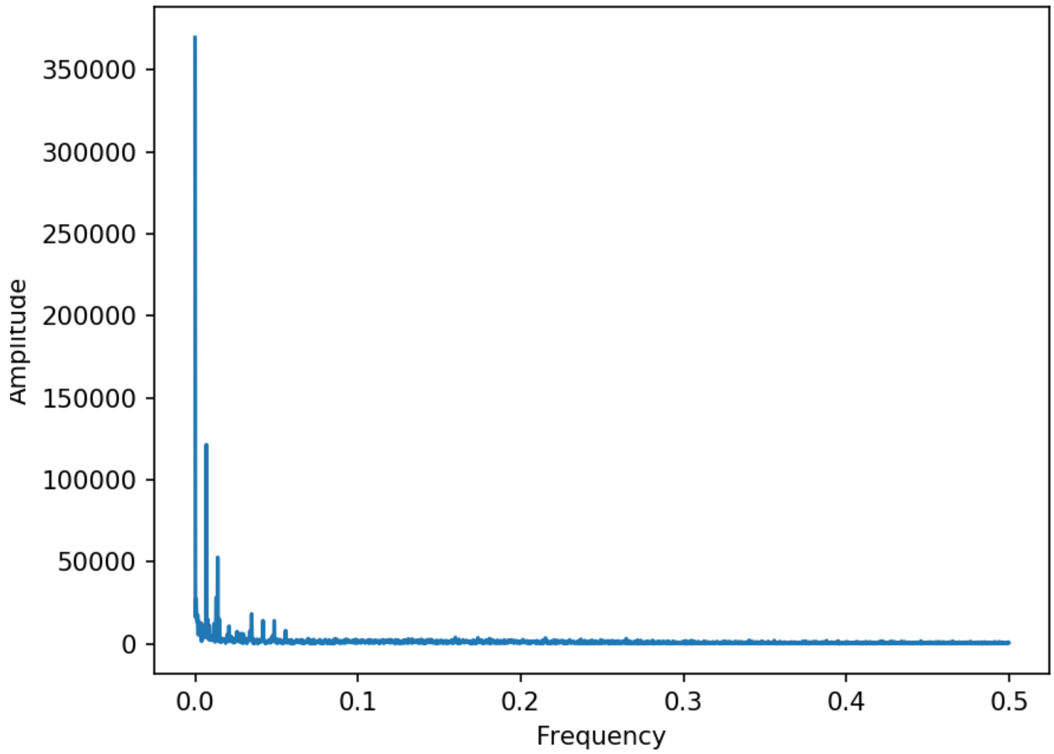

Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

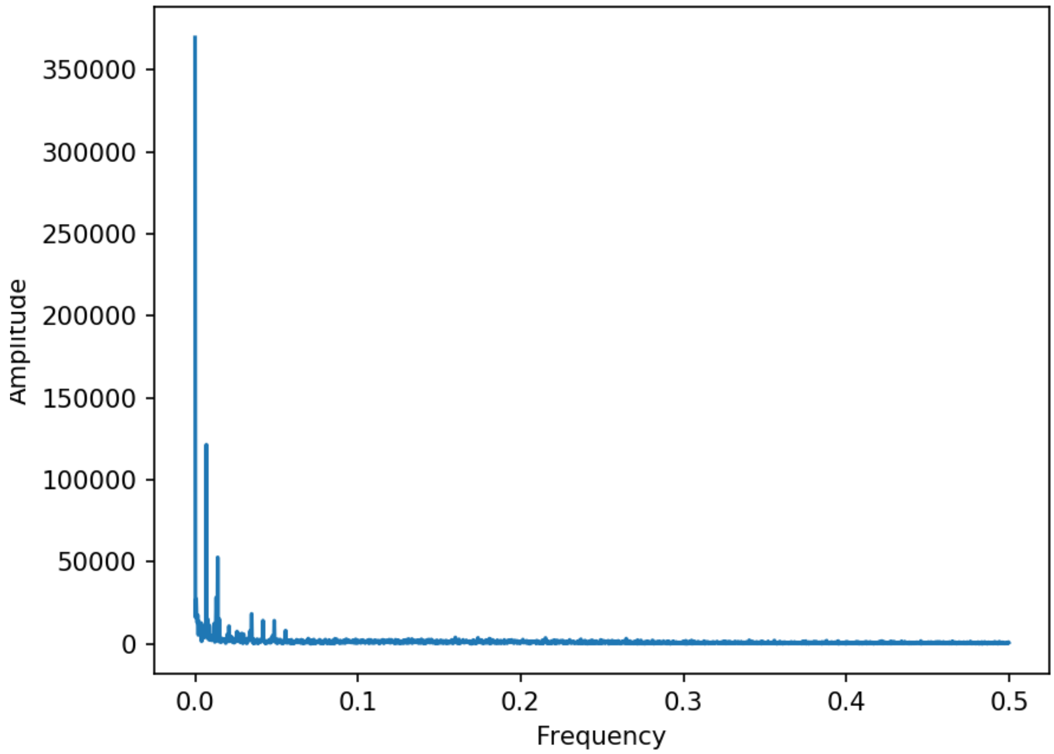

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

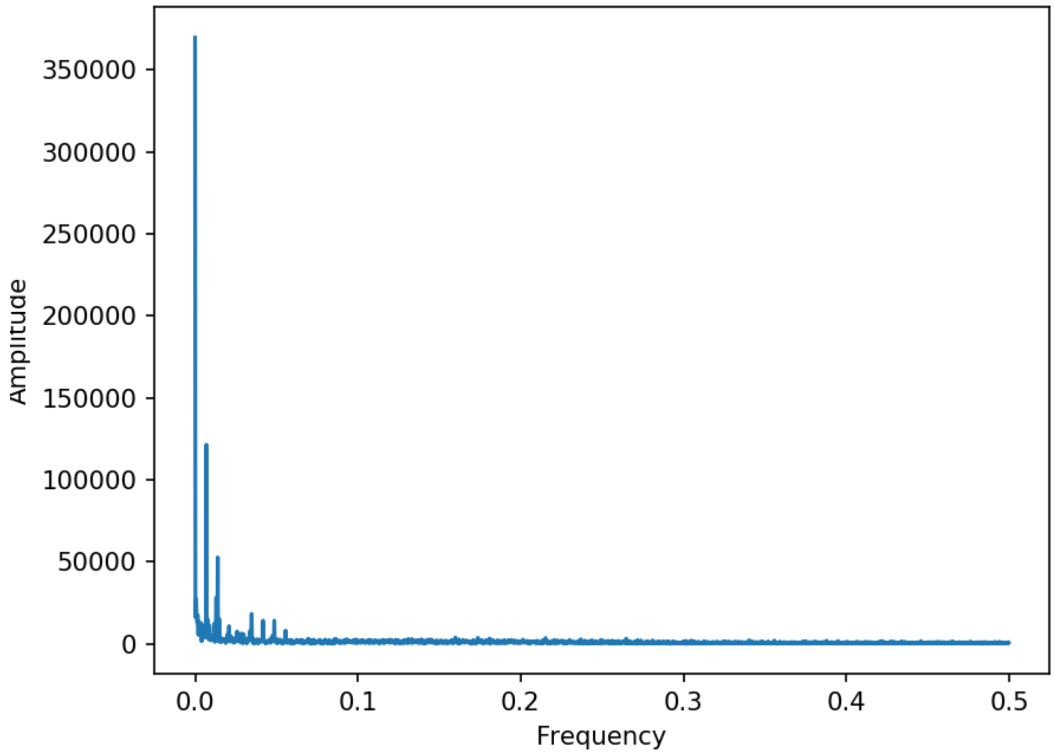

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

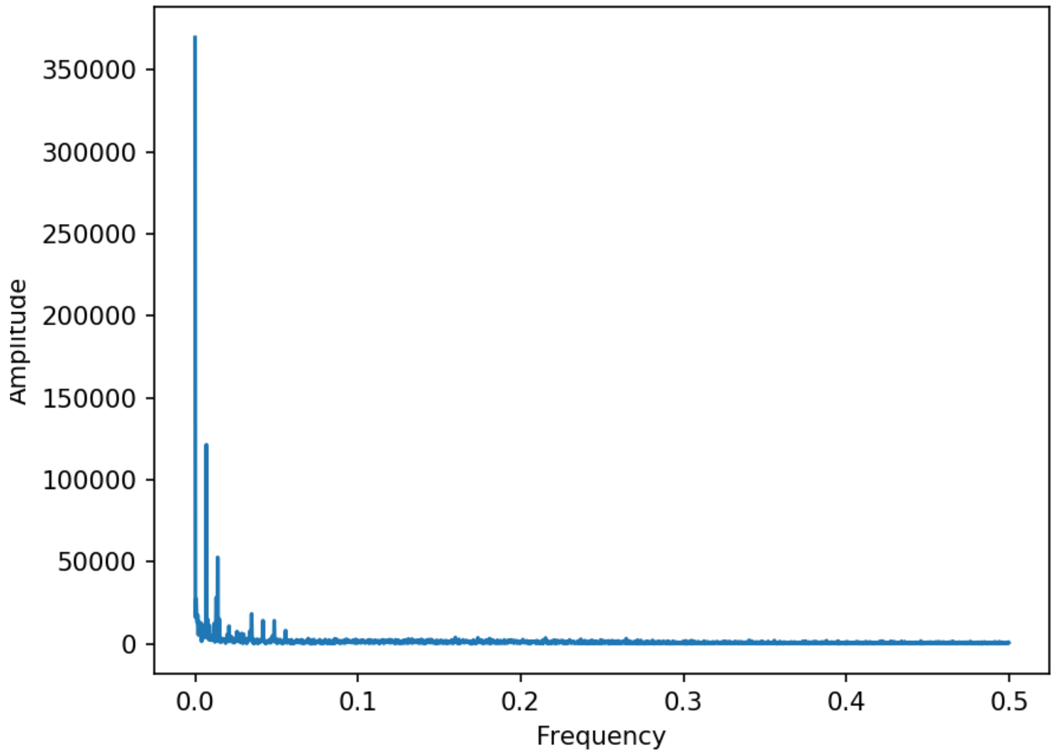

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

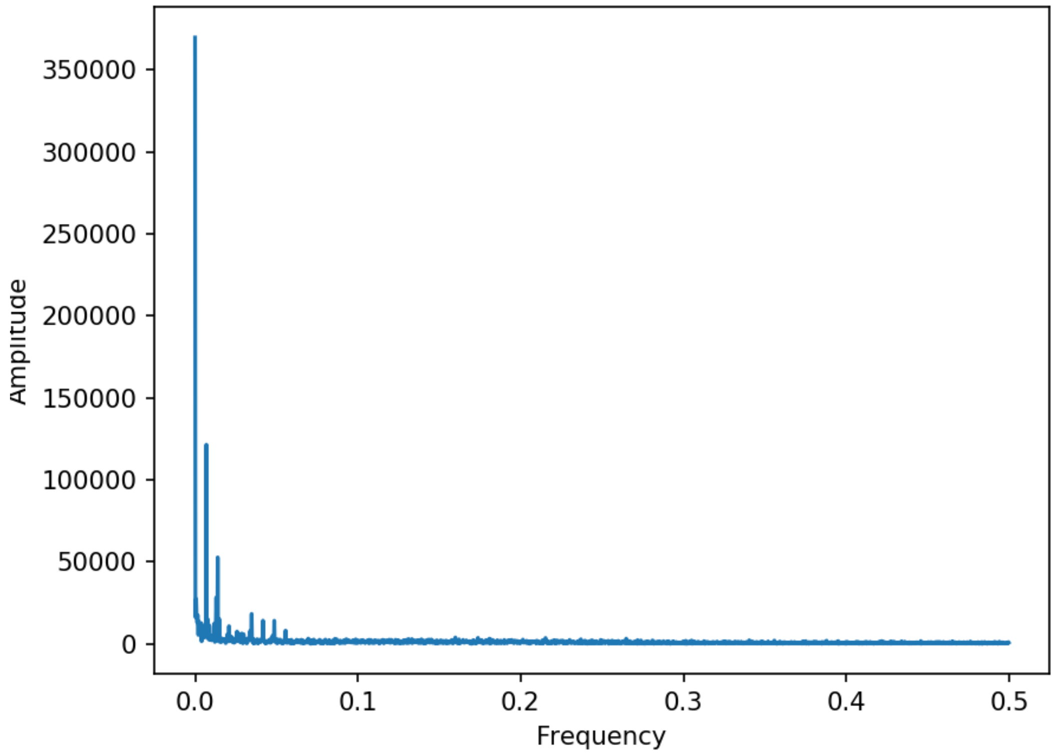

350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


350000
300000
250000
200000
150000
100000
50000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency



In [68]:

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

Modelling: Baseline Models

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

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

Simple Moving Averages

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

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

In [69]:

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


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

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

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

In [70]:

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

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

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

Weighted Moving Averages

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

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

In [71]:

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

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

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

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

In [72]:

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

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

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

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

Exponential Weighted Moving Averages

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

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

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

In [73]:

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

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

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

In [74]:

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

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

In [75]:

```
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 [76]:

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
---------------------------------------------------------------------------
------------------------------
Moving Averages (Ratios) -                             MAPE:  0.1821155173
392136       MSE:  400.0625504032258
Moving Averages (2016 Values) -                        MAPE:  0.1429284968
6975506        MSE:  174.84901993727598
---------------------------------------------------------------------------
------------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.1784869254
376018       MSE:  384.01578741039424
Weighted Moving Averages (2016 Values) -               MAPE:  0.1355108843
6182082       MSE:  162.46707549283155
---------------------------------------------------------------------------
------------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.1778355019486
1494       MSE:  378.34610215053766
Exponential Moving Averages (2016 Values) -         MAPE:  0.1350915263669
572       MSE:  159.73614471326164
```

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 [77]:

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

```
13104
```

For fourier transform we have to find amplitude and frequency

In [78]:

```
number_of_time_stamps = 10

output = []
tsne_lat = []
tsne_lon = []
tsne_weekday = []
tsne_feature = []


tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13094)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13094)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of pick
ups that are happened in last 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(10,4464+4176+4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x
3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps]
for r in range(0,len(regions_cum[i])-number_of_time_stamps)]))
    output.append(regions_cum[i][10:])
tsne_feature = tsne_feature[1:]
```

In [79]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_w
eekday[0]) == 40*13094 == len(output)*len(output[0])
```

Out[79]:

```
True
```

In [80]:

```
alpha = 0.3

predicted_values=[]

predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    for i in range(0,13104):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[10:])
    predicted_values=[]
```

In [81]:

```
amplitude = []
frequency = []
for i in range(40):
    amps  = np.abs(np.fft.fft(regions_cum[i][0:13104]))
    freqs = np.abs(np.fft.fftfreq(13104, 1))
    amp_indices = np.argsort(-amps)[1:]
    amp_values = []
    freq_values = []
    for j in range(0, 10):   #taking top ten amplitudes and frequencies
        amp_values.append(amps[amp_indices[j]])
        freq_values.append(freqs[amp_indices[j]])
    for k in range(13104):    #those top 10 frequencies and amplitudes are same for all
the points in one cluster
        amplitude.append(amp_values)
        frequency.append(freq_values)
```

In [82]:

```
len(amplitude)
```

Out[82]:

```
524160
```

In [83]:

```
len(frequency)
```

Out[83]:

```
524160
```

In [84]:

```
len(amp_values)
```

Out[84]:

```
10
```

In [ ]:

```
amplitude[0]
```

In [ ]:

```
frequency[0]
```

In [87]:

```
# extracting first 9165 timestamp values i.e 70% of 13094 (total timestamps) for our tr
aining data
train_features =  [tsne_feature[i*13094:(13094*i+9165)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(13094*(i))+9165:13094*(i+1)] for i in range(0,40)]
```

In [88]:

```
 # extracting first 9165 amplitude values i.e 70% of 13094 (total timestamps) for our t
raining data
train_amp = [amplitude[i*13094:(13094*i+9165)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_amp = [amplitude[(13094*(i))+9165:13094*(i+1)] for i in range(0,40)]
```

In [89]:

```
 # extracting first 9165 freq values i.e 70% of 13094 (total freqs) for our training da
ta
train_freq = [frequency[i*13094:(13094*i+9165)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_freq = [frequency[(13094*(i))+9165:13094*(i+1)] for i in range(0,40)]
```

In [95]:

```
# extracting first 9165 timestamp values i.e 70% of 13094 (total timestamps) for our tr
aining data
tsne_train_flat_lat = [i[:9165] for i in tsne_lat]
tsne_train_flat_lon = [i[:9165] for i in tsne_lon]
tsne_train_flat_weekday = [i[:9165] for i in tsne_weekday]
tsne_train_flat_output = [i[:9165] for i in output]
tsne_train_flat_exp_avg = [i[:9165] for i in predict_list]
```

In [97]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) for o
ur test data
tsne_test_flat_lat = [i[9165:] for i in tsne_lat]
tsne_test_flat_lon = [i[9165:] for i in tsne_lon]
tsne_test_flat_weekday = [i[9165:] for i in tsne_weekday]
tsne_test_flat_output = [i[9165:] for i in output]
tsne_test_flat_exp_avg = [i[9165:] for i in predict_list]
```

In [98]:

```
# the above contains values in the form of list of lists (i.e. list of values of each r
egion), here we make all of them in one list
train_new_features = []
for i in range(0,40):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,40):
    test_new_features.extend(test_features[i])
train_freqs=[]
test_freqs=[]
train_amps=[]
test_amps=[]
for i in range(0,40):
    train_freqs.extend(train_freq[i])
    test_freqs.extend(test_freq[i])
    train_amps.extend(train_amp[i])
    test_amps.extend(test_amp[i])
```

In [101]:

```
len(train_new_features)
```

Out[101]:

```
366600
```

In [102]:

```
train_new_features[0]
```

Out[102]:

```
array([  0,  63, 217, 189, 137, 135, 129, 150, 164, 152], dtype=int64)
```

In [103]:

```
train_features = np.hstack((train_new_features,train_freqs,train_amps))
test_features = np.hstack((test_new_features,test_freqs,test_amps))
```

In [104]:

```
len(train_features)
```

Out[104]:

```
366600
```

In [ ]:

```
train_features[0]
```

In [106]:

```
# converting lists of lists into single 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 [107]:

```
tsne_train_lat[0]
```

Out[107]:

40.776227610093365

In [108]:

```
len(tsne_train_lat)
```

Out[108]:

366600

In [110]:

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

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

In [111]:

```
len(tsne_test_exp_avg)
```

Out[111]:

157160

In [112]:

```
len(test_features)
```

Out[112]:

157160

In [113]:

```
# Preparing the data frame for our train data
columns = ['ft_-10','ft_9','ft_8','ft_7','ft_6','ft_5','ft_4','ft_3','ft_2','ft_1',
           'freq1','freq2','freq3','freq4','freq5','freq6','freq7','freq8','freq9','fre
q10',
           'amp1','amp2','amp3','amp4','amp5','amp6','amp7','amp8','amp9','amp10']
df_train = pd.DataFrame(data=train_features, columns=columns)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon
df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg

print(df_train.shape)
```

```
(366600, 34)
```

In [115]:

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

```
(157160, 34)
```

Linear Regression

In [145]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

alphas = [0.0001,0.001,0.01,0.1,1,10,100,1000]
lr_reg=Ridge()
grid = GridSearchCV(estimator=lr_reg,param_grid={'alpha':alphas}, cv=3,scoring = 'neg_m
ean_absolute_error')
grid.fit(df_train, tsne_train_output)
```

Out[145]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=
None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1
000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [146]:

```
print(grid.best_estimator_.alpha)
```

```
1
```

In [149]:

```
lr_reg = Ridge(alpha = 1)
lr_reg.fit(df_train, tsne_train_output)

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

In [160]:

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

clf = RandomForestRegressor(n_jobs=-1)
param_dist = {"n_estimators":[50,100,150,200,250],
              "max_depth": [50,100,150,200,250],
              "min_samples_split": sp_randint(10,50),
              "min_samples_leaf": sp_randint(10,15)}

regr = RandomizedSearchCV(clf, param_distributions=param_dist,verbose=10,random_state=4
2,
                                   n_iter=3, scoring='neg_mean_absolute_error',n_jobs=-
1)
regr.fit(df_train, tsne_train_output)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed: 32.9min
[Parallel(n_jobs=-1)]: Done  10 out of  15 | elapsed: 46.7min remaining: 2
3.4min
[Parallel(n_jobs=-1)]: Done  12 out of  15 | elapsed: 69.7min remaining: 1
7.4min
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 81.5min finished
```

Out[160]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    criterion='mse',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=
0.0,
                                                    min_impurity_split=Non
e,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_lea
f=0.0,
                                                    n_estimators=100, n_job
s=-1,
                                                    oob_score=Fal...
                   param_distributions={'max_depth': [50, 100, 150, 200, 2
50],
                                        'min_samples_leaf': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x0000021735169FD0>,
                                        'min_samples_split': <scipy.stats.
_distn_infrastructure.rv_frozen object at 0x0000021732E639B0>,
                                        'n_estimators': [50, 100, 150, 20
0,
                                                          250]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_er
ror',
                   verbose=10)
```

In [161]:

```
regr.best_estimator_
```

Out[161]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=200, max_features='auto', max_leaf_nodes=N
one,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=14,
                      min_samples_split=24, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [163]:

```
regr1 = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=200, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=14,
                      min_samples_split=24, min_weight_fraction_leaf=0.0,
                      n_estimators=150, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)

regr1.fit(df_train, tsne_train_output)

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

In [165]:

```
clf = xgb.XGBRegressor(
 learning_rate =0.1,
 min_child_weight=3,
 gamma=0,n_jobs=-1,
 subsample=0.8,
 reg_alpha=200, reg_lambda=200,
 colsample_bytree=0.8,nthread=4)

param_dist = {"n_estimators":[50,100,150,200,250],
              "max_depth": [50,100,150,200,250],
              "min_samples_split": sp_randint(10,50),
              "min_samples_leaf": sp_randint(10,15)}

xgb = RandomizedSearchCV(clf, param_distributions=param_dist,random_state=42,n_jobs=-1,
verbose = 10,
                                   n_iter=5, scoring='neg_mean_absolute_error')
xgb.fit(df_train, tsne_train_output)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed: 49.7min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 69.0min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 185.4min
[Parallel(n_jobs=-1)]: Done  21 out of  25 | elapsed: 217.2min remaining:
41.4min
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed: 226.0min finished
```

Out[165]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=0.8, gamma=0,
                                          gpu_id=None, importance_type='ga
in',
                                          interaction_constraints=None,
                                          learning_rate=0.1,
                                          max_delta_step=None, max_depth=N
one,
                                          min_child_weight=3, missing=nan,
                                          monotone_constraints=None,
                                          n_estim...
                   param_distributions={'max_depth': [50, 100, 150, 200, 2
50],
                                        'min_samples_leaf': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x0000021732E42908>,
                                        'min_samples_split': <scipy.stats.
_distn_infrastructure.rv_frozen object at 0x0000021732E42550>,
                                        'n_estimators': [50, 100, 150, 20
0,
                                                         250]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_er
ror',
                   verbose=10)
```

In [167]:

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

In [168]:

```
#feature importances
x_model.get_booster().get_score(importance_type='weight')
```

Out[168]:

```
{'ft_1': 753,
 'exp_avg': 490,
 'ft_2': 673,
 'ft_3': 553,
 'ft_4': 447,
 'ft_9': 418,
 'ft_-10': 475,
 'amp5': 104,
 'ft_5': 412,
 'amp9': 106,
 'ft_7': 365,
 'ft_6': 454,
 'amp1': 161,
 'ft_8': 406,
 'amp2': 31,
 'amp10': 18,
 'amp7': 134,
 'lon': 141,
 'amp3': 125,
 'lat': 156,
 'freq3': 49,
 'freq9': 102,
 'freq7': 65,
 'amp4': 32,
 'weekday': 97,
 'amp6': 25,
 'freq5': 45,
 'freq6': 13,
 'amp8': 26,
 'freq10': 28,
 'freq8': 12,
 'freq1': 12,
 'freq4': 10,
 'freq2': 4}
```

In [169]:

```
train_mape=[]
test_mape=[]

train_mape.append((mean_absolute_error(tsne_train_output,df_train['ft_1'].values))/(sum
(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].values))/(
sum(tsne_train_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_predictions))/(sum(
tsne_train_output)/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(t
sne_train_output)/len(tsne_train_output)))

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/(sum(t
sne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values))/(su
m(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, rndf_test_predictions))/(sum(ts
ne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output, xgb_test_predictions))/(sum(tsn
e_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)))
```

In [170]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------
-------------------------")
print ("Linear Regression -                         Train: ",train_mape[3],"      Test:
",test_mape[3])
print ("Random Forest Regression -                   Train: ",train_mape[2],"     Test:
",test_mape[2])
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------
------------------------------
Linear Regression -                         Train:  0.11073887184864212
Test:  0.12078173265104312
Random Forest Regression -                   Train:  0.104336480850723
Test:  0.12235336585709074
```

In [171]:

```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print ("--------------------------------------------------------------------------------------------------------------")
print ("Linear Regression -                         Train: ",train_mape[4],"      Test: ",test_mape[4])
print ("Random Forest Regression -                   Train: ",train_mape[2],"     Test: ",test_mape[2])
print ("XgBoost Regression -                         Train: ",train_mape[3],"      Test: ",test_mape[3])
print ("--------------------------------------------------------------------------------------------------------------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------------
Linear Regression -                         Train:  0.131740643004255
Test:  0.12706100001759005
Random Forest Regression -                   Train:  0.104336480850723
Test:  0.12235336585709074
XgBoost Regression -                         Train:  0.11073887184864212
Test:  0.12078173265104312
--------------------------------------------------------------------------------------------------------------
```