Taxi demand prediction in New York City

Assignments

In [1]:

```
'''
Task 1: Incorporate Fourier features as features into Regression models and measure MAPE. <br>

Task 2: Perform hyper-parameter tuning for Regression models.
        2a. Linear Regression: Grid Search
        2b. Random Forest: Random Search
        2c. Xgboost: Random Search
Task 3: Explore more time-series features using Google search/Quora/Stackoverflow
to reduce the MAPE to < 12%
'''
```

Out[1]:

```
'\nTask 1: Incorporate Fourier features as features into Regression models and measure MAPE. <br>\
n\nTask 2: Perform hyper-parameter tuning for Regression models.\n        2a. Linear Regression: G
rid Search\n        2b. Random Forest: Random Search \n        2c. Xgboost: Random Search\nTask 3:
Explore more time-series features using Google search/Quora/Stackoverflow\nto reduce the MAPE to <
12%\n'
```

In [2]:

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

import pandas as pd#pandas to create small dataframes

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

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

import time #Convert to unix time

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

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

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

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

```
import 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")
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV
from datetime import datetime
import math
import scipy
from prettytable import PrettyTable
from sklearn.preprocessing import StandardScaler
```

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 [3]:

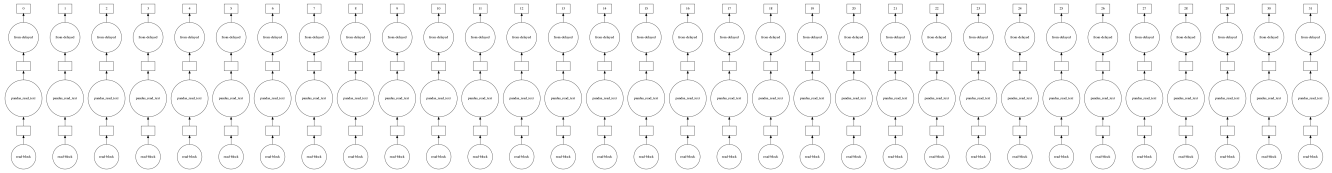
```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('yellow_tripdata_2015-01.csv')
print(month.columns)
```

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

In [4]:

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

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

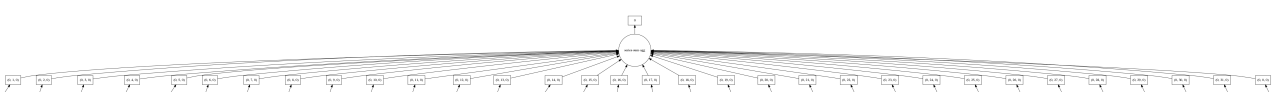
Out[4]:

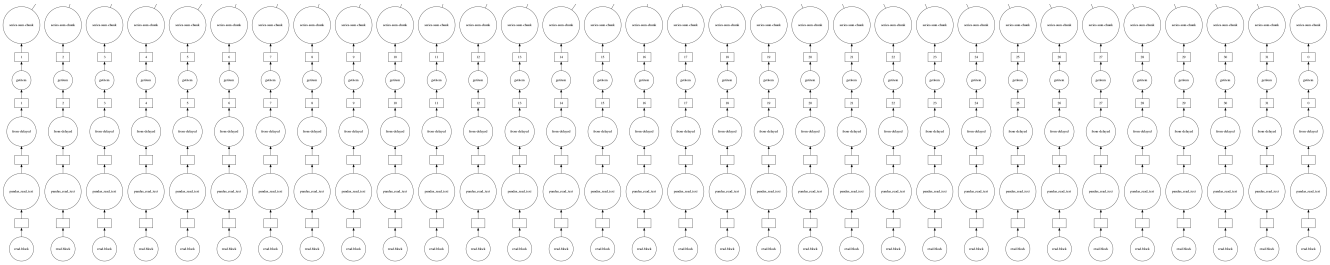


In [5]:

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

Out[5]:





Features in the dataset:

Field Name		Description
VendorID	1. 2.	A code indicating the TPEP provider that provided the record. Creative Mobile Technologies 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	1. 2. 3. 4. 5. 6.	The final rate code in effect at the end of the trip. Standard rate JFK Newark Nassau or Westchester Negotiated fare 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	1. 2. 3. 4. 5. 6.	A numeric code signifying how the passenger paid for the trip. Credit card Cash No charge Dispute Unknown Voided trip
Fare_amount		The time-and-distance fare calculated by the meter.
Extra		Miscellaneous extras and surcharges. Currently, this only includes. the $0.50 and 1$ rush hour and overnight charges.
MTA_tax		0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge		0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge began being levied in 2015.
Tip_amount		Tip amount – This field is automatically populated for credit card tips.Cash tips are not included.
Tolls_amount		Total amount of all tolls paid in trip.
Total_amount		The total amount charged to passengers. Does not include cash tips.

ML Problem Formulation

Time-series forecasting and Regression

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

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

Performance metrics

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

Data Cleaning

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

In [6]:

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

Out[6]:

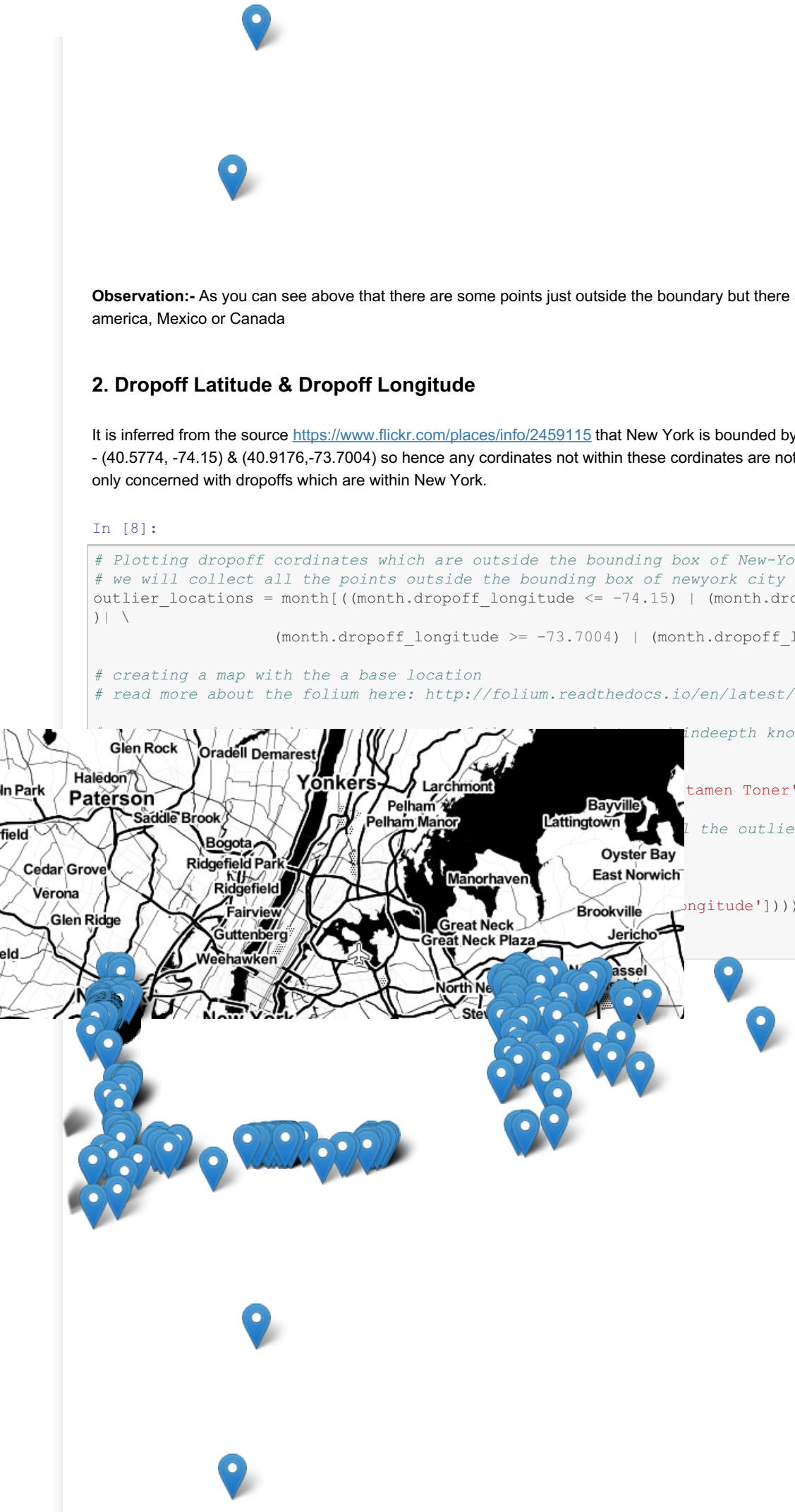
	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RateCode
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	-73.993896	40.750111	
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.30	-74.001648	40.724243	
2	1	2015-01-10 20:33:38	2015-01-10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

1. Pickup Latitude and Pickup Longitude

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

In [7]:

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

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

                                                                    indeepth knowledge on these maps and


                                                                    tamen Toner')

                                                                    l the outliers will take more time


                                                                    itude'])).add_to(map_osm)
```



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

2. Dropoff Latitude & Dropoff Longitude

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

In [8]:

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

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

                                                                    indeepth knowledge on these maps and


                                                                    tamen Toner')

                                                                    l the outliers will take more time


                                                                    ngitude']])).add_to(map_osm)
```

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]:

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

# in out data we have time in the formate "YYYY-MM-DD HH:MM:SS" we convert thiss sting to python t
ime formate and then into unix time stamp
# https://stackoverflow.com/a/27914405
def convert_to_unix(t):
    #we have a time in the format "YYYY-MM-DD HH:MM:SS", which is a string
    change = datetime.strptime(t, "%Y-%m-%d %H:%M:%S") #this will convert the String time into date
time format
    t_tuple = change.timetuple() #this will convert the datetime formatted time into structured ti
me
    return time.mktime(t_tuple) + 3600
```

In [10]:

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

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

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

    return new_frame

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

In [11]:

```
frame_with_durations = return_with_trip_times(month)
```

In [12]:

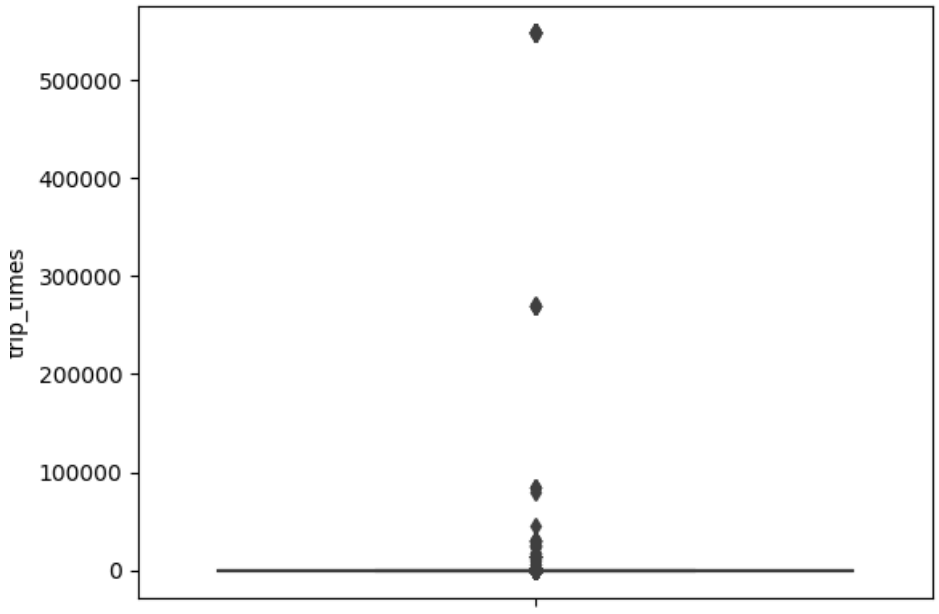
```
frame_with_durations.head()
```

Out[12]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pick
0	1	1.59	-73.993896	40.750111	-73.974785	40.750618	17.05	18.050000	1.42
1	1	3.30	-74.001648	40.724243	-73.994415	40.759109	17.80	19.833333	1.42
2	1	1.80	-73.963341	40.802788	-73.951820	40.824413	10.80	10.050000	1.42
3	1	0.50	-74.009087	40.713818	-74.004326	40.719986	4.80	1.866667	1.42
4	1	3.00	-73.971176	40.762428	-74.004181	40.742653	16.30	19.316667	1.42

In [13]:

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



In [14]:

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

```
0 percentile value is -1211.0166666666667
10 percentile value is 3.8333333333333335
20 percentile value is 5.383333333333334
30 percentile value is 6.816666666666666
40 percentile value is 8.3
50 percentile value is 9.95
```

```
60 percentile value is 11.866666666666667
70 percentile value is 14.283333333333333
80 percentile value is 17.633333333333333
90 percentile value is 23.45
100 percentile value is  548555.6333333333
```

In [15]:

```
#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])
```

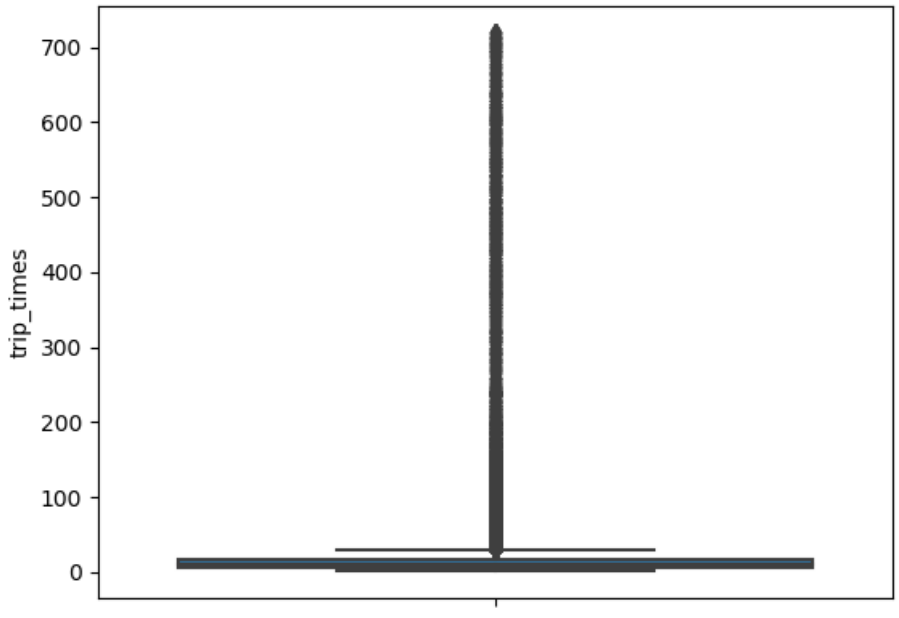
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98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

In [16]:

```
#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 [17]:

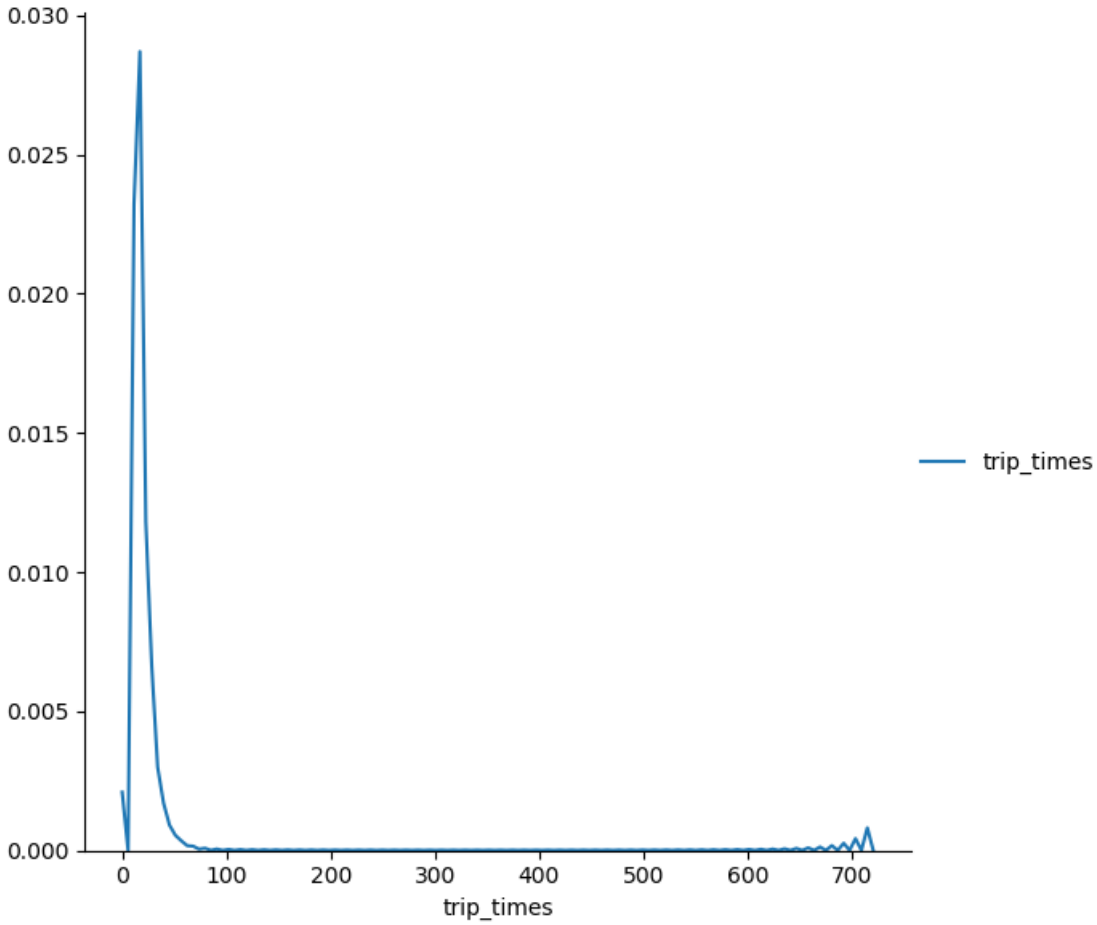
```
#box-plot after removal of outliers
sns.boxplot(y="trip_times", data =frame_with_durations_modified)
plt.show()
```



In [18]:

```
#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();
```

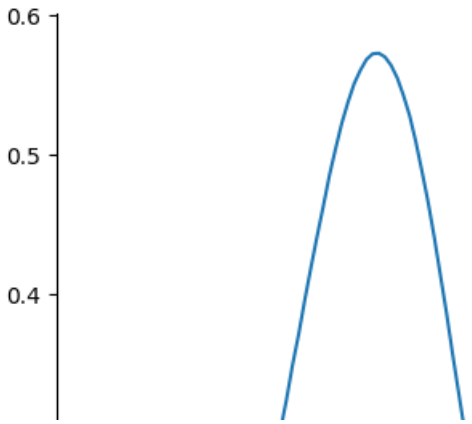


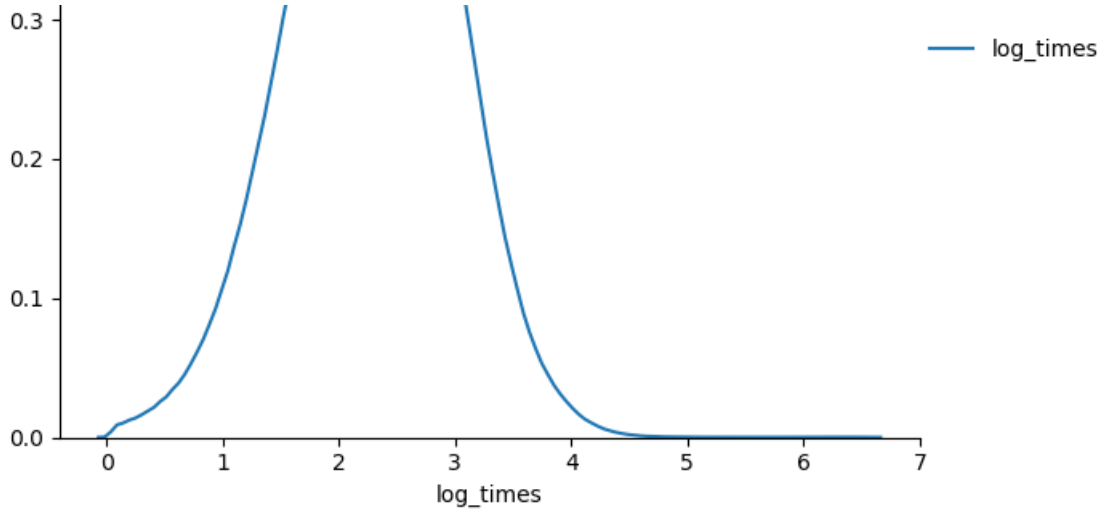
In [19]:

```
#converting the values to log-values to chec for log-normal

frame_with_durations_modified['log_times']=[math.log(i) for i in frame_with_durations_modified['trip_times'].values]
```

In [20]:

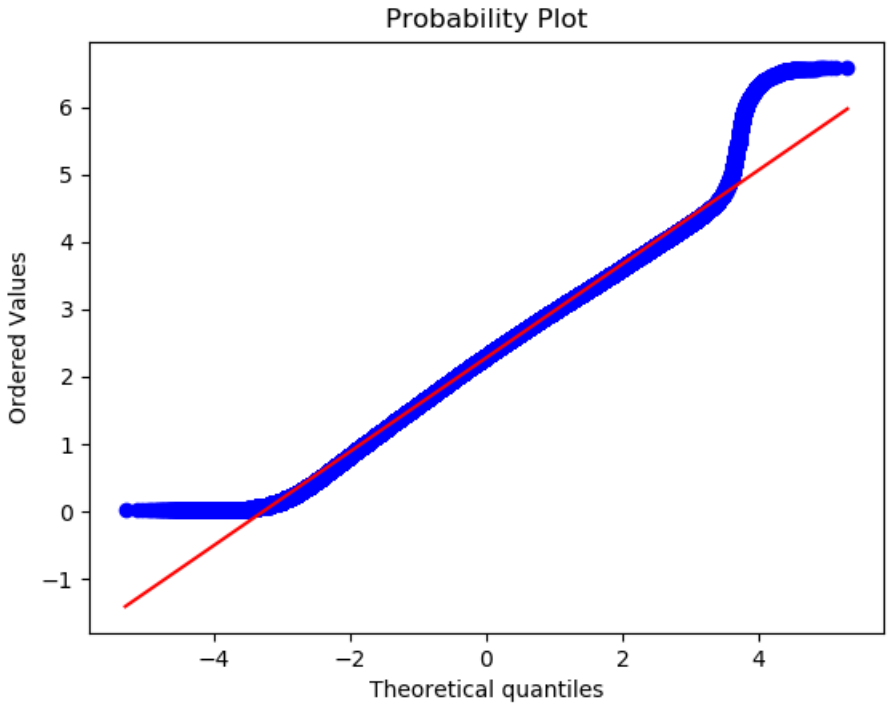
```
#pdf of log-values
sns.FacetGrid(frame_with_durations_modified,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```





In [21]:

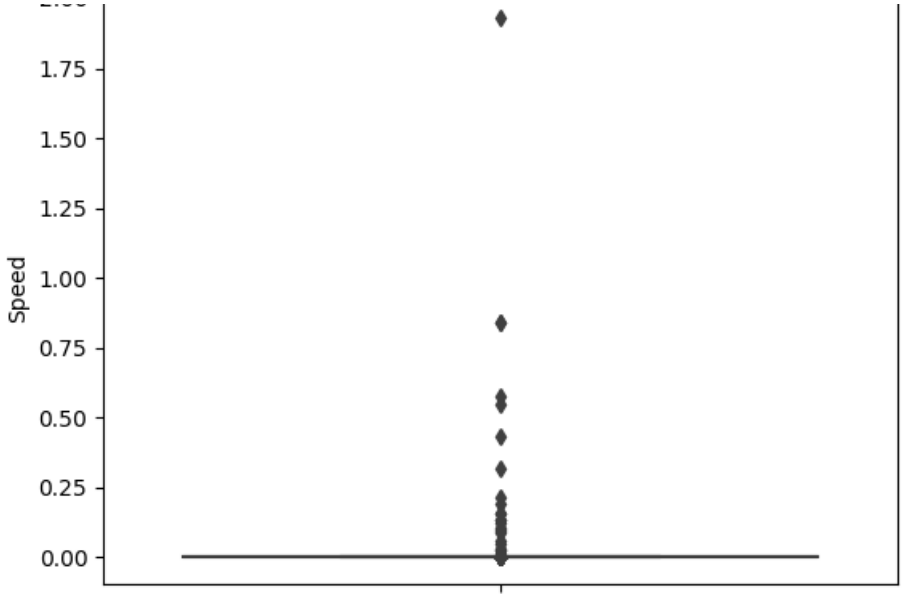
```
#Q-Q plot for checking if trip-times is log-normal

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_distance']/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("100 percentile value is ",var[-1])
```

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

In [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) &
(frame_with_durations.Speed<45.31)]
```

In [27]:

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

Out[27]:

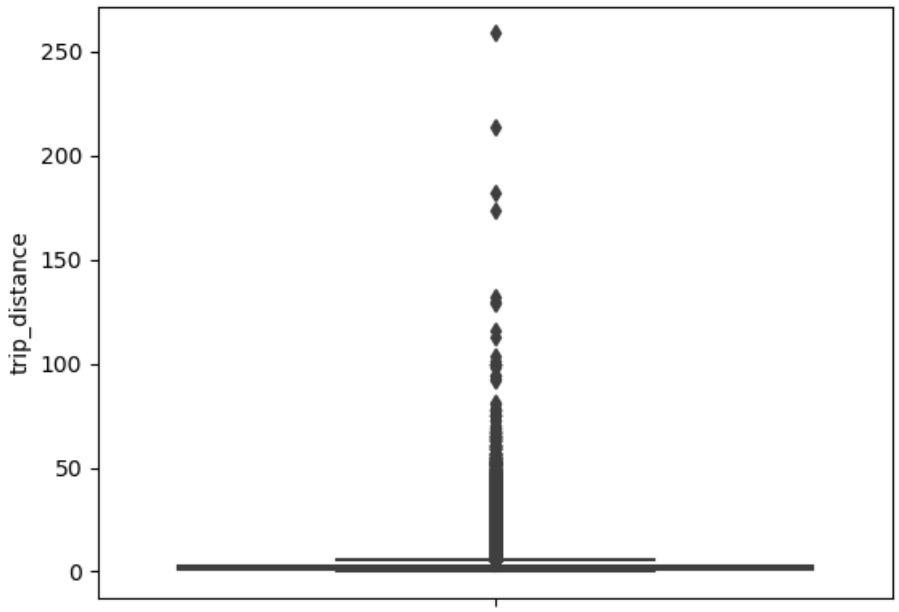
```
12.450173996027528
```

The avg speed in Newyork speed is 12.45miles/hr, so a cab driver can travel2 miles per 10min on avg.

4. Trip Distance

In [28]:

```
# 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 [29]:

```
#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 [30]:

```
#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 [31]:

```
#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])
```

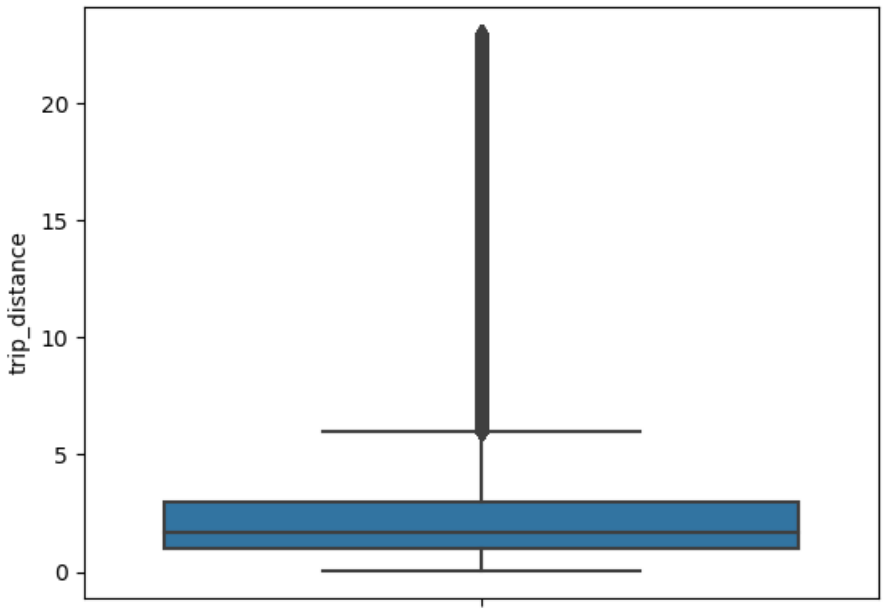
```
99.0 percentile value is 18.17
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.83
99.4 percentile value is 19.13
99.5 percentile value is 19.5
99.6 percentile value is 19.96
99.7 percentile value is 20.5
99.8 percentile value is 21.22
99.9 percentile value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

In [32]:

```
#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 [33]:

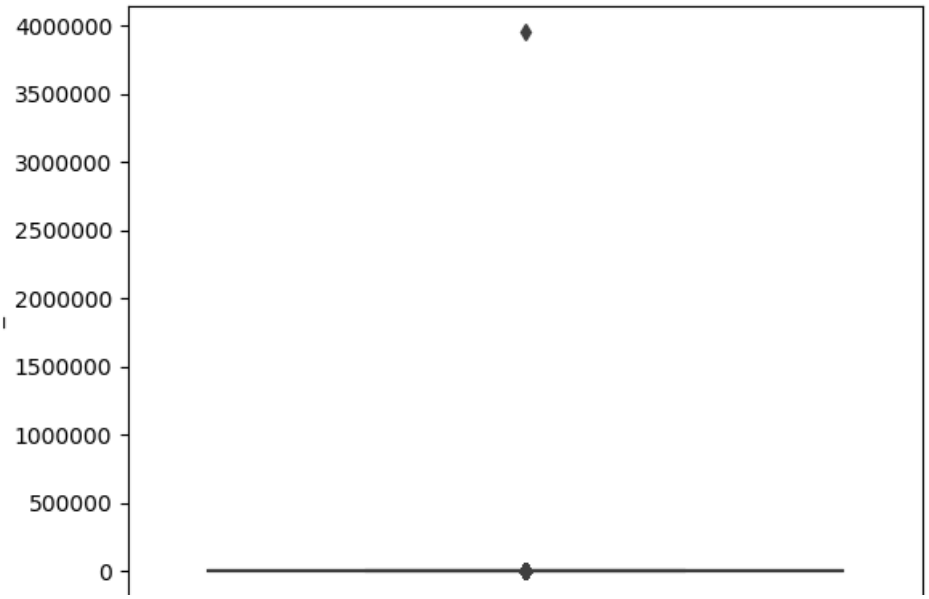
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



5. Total Fare

In [34]:

```
# 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 [35]:

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

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

In [36]:

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

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

In [37]:

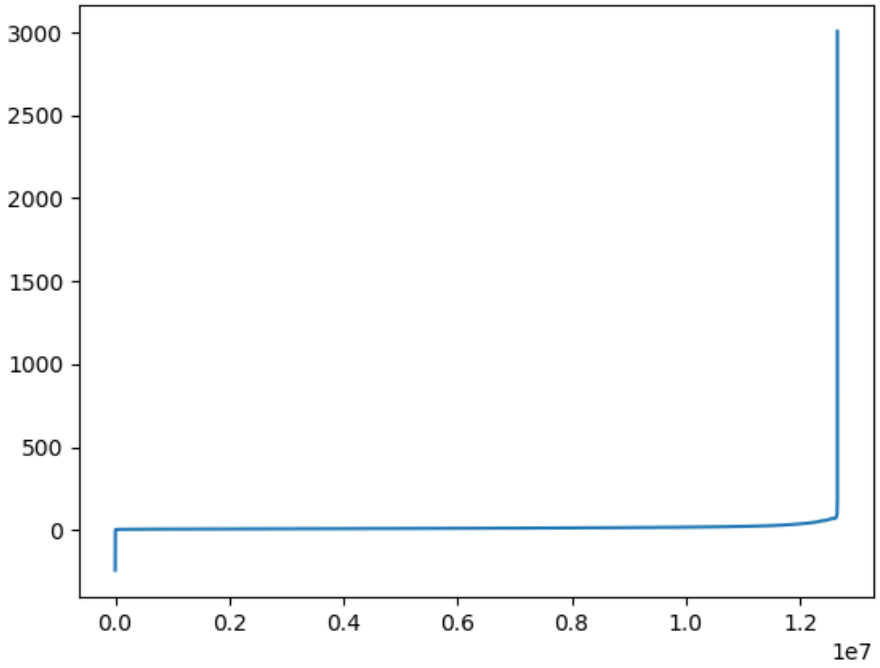
```
#calculating total fare amount values at each percntile
99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

```
99.0 percentile value is 66.13
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

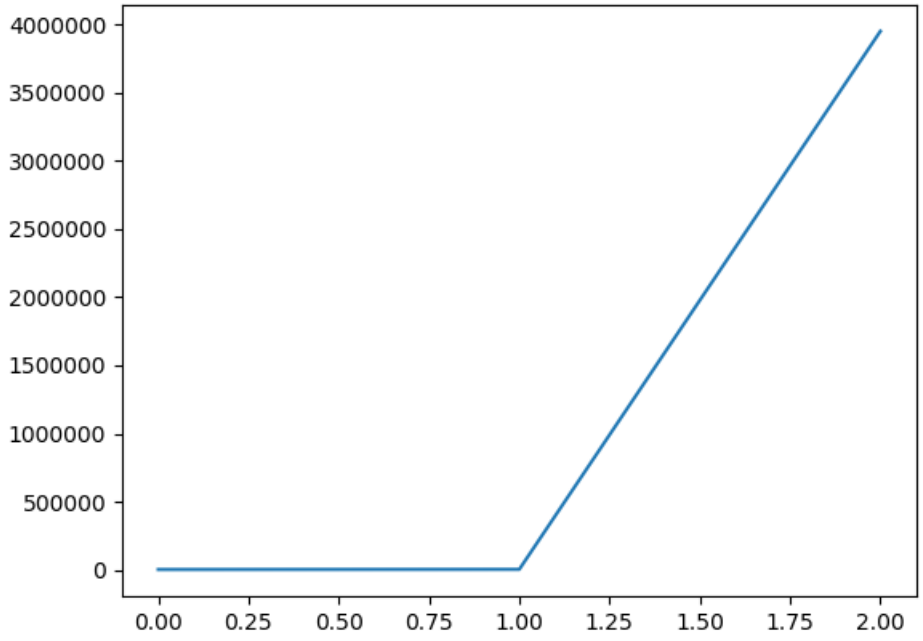
In [38]:

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



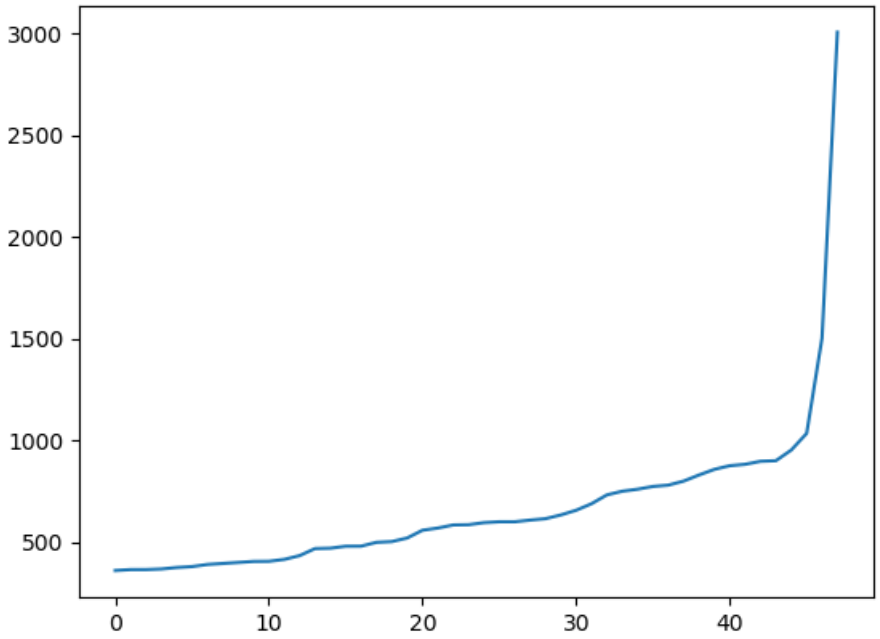
In [39]:

```
# 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 [40]:

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



Remove all outliers/erronous points.

In [41]:

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


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


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


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

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

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


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

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

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

In [42]:

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

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

Data-preperation

Clustering/Segmentation

In [43]:

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

def find_min_distance(cluster_centers, cluster_len):
    nice_points = 0
    wrong_points = 0
    less2 = []
    more2 = []
    min_dist=1000
    for i in range(0, cluster_len):
        nice_points = 0
        wrong_points = 0
        for j in range(0, cluster_len):
            if j!=i:
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0], cluster_centers[i][1
,cluster_centers[j][0], cluster_centers[j][1])
                min_dist = min(min_dist,distance/(1.60934*1000))
                if (distance/(1.60934*1000)) <= 2:
                    nice_points +=1
                else:
                    wrong_points += 1
```

```
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters within the vici
nity (i.e. intercluster-distance < 2):", np.ceil(sum(less2)/len(less2)), "\nAvg. Number of
Clusters outside the vicinity (i.e. intercluster-distance > 2):", np.ceil(sum(more2)/len(more2)),"
\nMin inter-cluster distance = ",min_dist,"\n---")

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fit(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 regions
#that are close to any cluster center
# and make sure that the minimum inter cluster should not be very less
for increment in range(10, 100, 10):
    cluster_centers, cluster_len = find_clusters(increment)
    find_min_distance(cluster_centers, cluster_len)
```

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 8.0
Min inter-cluster distance =  1.0945442325142543
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 4.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 16.0
Min inter-cluster distance =  0.7131298007387813
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 22.0
Min inter-cluster distance =  0.5185088176172206
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 32.0
Min inter-cluster distance =  0.5069768450363973
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 12.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 38.0
Min inter-cluster distance =  0.365363025983595
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 14.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 46.0
Min inter-cluster distance =  0.34704283494187155
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 16.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 54.0
Min inter-cluster distance =  0.30502203163244707
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 18.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 62.0
Min inter-cluster distance =  0.29220324531738534
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 69.0
Min inter-cluster distance =  0.18257992857034985
---
```

Inference:

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

In [44]:

```
# Getting 30 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=30, batch_size=10000,random_state=0).fit(coords)
frame_with_durations_outliers_removed['pickup_cluster'] =
kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
```

Plotting the cluster centers:

In [45]:

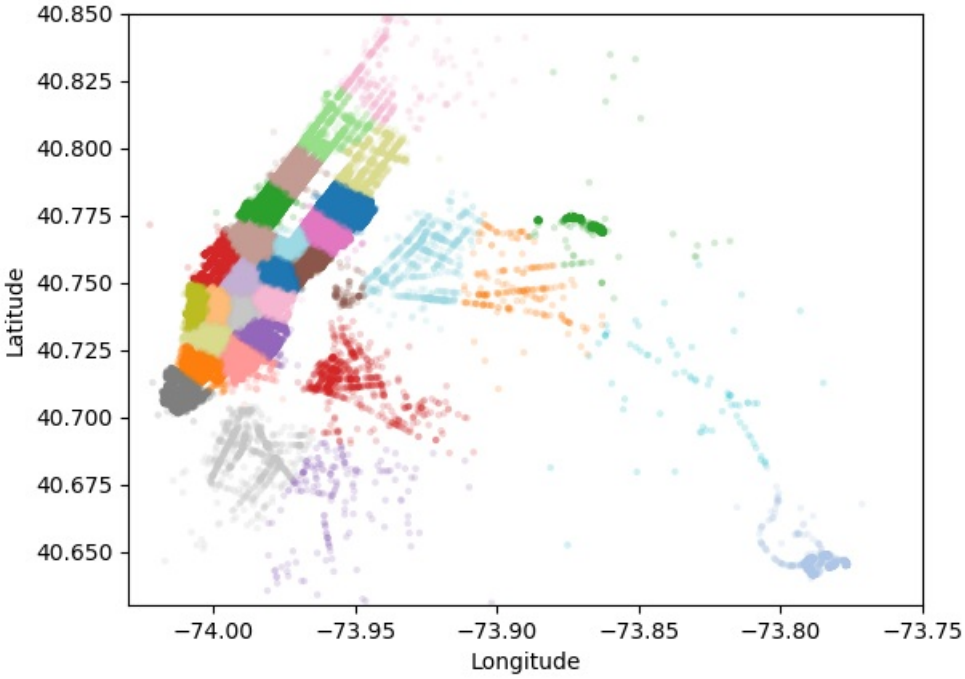
```
# Plotting the cluster centers on OSM
cluster_centers = kmeans.cluster_centers_
cluster_len = len(cluster_centers)
map_osm = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(cluster_len):
    folium.Marker(list((cluster_centers[i][0],cluster_centers[i][1])), popup=(str(cluster_centers[i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

Out[45]:

Plotting the clusters:

In [46]:

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

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [47]:

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

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

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

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

In [48]:

```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
frame_with_durations_outliers_removed['pickup_cluster'] =
kmeans.predict(frame_with_durations_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']].groupby(['pickup_cluster','pickup_
bins']).count()
```

In [49]:

```
# 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[49]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pick
0	1	1.59	-73.993896	40.750111	-73.974785	40.750618	17.05	18.050000	1.42
1	1	3.30	-74.001648	40.724243	-73.994415	40.759109	17.80	19.833333	1.42
2	1	1.80	-73.963341	40.802788	-73.951820	40.824413	10.80	10.050000	1.42
3	1	0.50	-74.009087	40.713818	-74.004326	40.719986	4.80	1.866667	1.42
4	1	3.00	-73.971176	40.762428	-74.004181	40.742653	16.30	19.316667	1.42

In [50]:

```
# hear the trip_distance represents the number of pickups that are happend in that particular 10mi
n intravel
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels 24*31*60/10 =4464
bins)
jan_2015_groupby.head()
```

Out[50]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**7**	138
	8	262
	9	311
	10	326
	11	381

Data Preparation for January 2016 data

In [51]:

```
# Up till now we cleaned data and prepared data for the month of Jan 2015.

# now doing the same operations for the month of Jan 2016.

# 1. Get the dataframe which includes only required colums.
# 2. Add trip_duration, speed, unix time stamp of pickup_time.
# 4. Remove the outliers based on trip_duration, speed, trip_distance, total_amount.
# 5. Remove all the points where pickup and dropoff are outside of New York City area.
# 6. Add pickup_cluster to each data point.
# 7. Add time_bin (index of 10min intravel to which that trip belongs to).
# 8. Group by data, based on 'pickup_cluster' and 'time_bin'
startTime = datetime.now()
frame_2016 = dd.read_csv("yellow_tripdata_2016-01.csv")

print("PREPARATION OF JANUARY 2016 DATA.")
print("-"*35)

print("Number of columns = "+str(len(frame_2016.columns)))
print("-"*35)

new_frame2 = return_with_trip_times(frame_2016) #return_with_trip_times
print("New Frame for Jan 2016 creation done")
print("-"*35)

new_frame_cleaned2 = new_frame2[(new_frame2.trip_times>1) & (new_frame2.trip_times<720)]
print("Trip Duration Outliers removed")
```

```
print("-"*35)

new_frame_cleaned2 = new_frame_cleaned2[(new_frame_cleaned2.Speed>0) & (new_frame_cleaned2.Speed<45
.31)]
print("Speed Outliers removed")
print("-"*35)

new_frame_cleaned2 = new_frame_cleaned2[(new_frame_cleaned2.trip_distance>0) & (new_frame_cleaned2.
trip_distance<23)]
print("Trip Distance Outliers removed")
print("-"*35)

new_frame_cleaned2 = new_frame_cleaned2[(new_frame_cleaned2.total_amount>0) & (new_frame_cleaned2.t
otal_amount<86.6)]
print("Total Amount Outliers removed")
print("-"*35)

new_frame_cleaned2 = new_frame_cleaned2[(((new_frame_cleaned2.pickup_latitude >= 40.5774) &
(new_frame_cleaned2.pickup_latitude <= 40.9176)) & ((new_frame_cleaned2.pickup_longitude >= -74.15
) & (new_frame_cleaned2.pickup_longitude <= -73.7004)))]
print("Pickups outside of NYC are removed")
print("-"*35)

new_frame_cleaned2 = new_frame_cleaned2[(((new_frame_cleaned2.dropoff_latitude >= 40.5774) &
(new_frame_cleaned2.dropoff_latitude <= 40.9176)) & ((new_frame_cleaned2.dropoff_longitude >=
-74.15) & (new_frame_cleaned2.dropoff_longitude <= -73.7004)))]
print("Dropoffs outside of NYC are removed")
print("-"*35)

new_frame_cleaned2["pickup_cluster"] = kmeans.predict(new_frame_cleaned2[["pickup_latitude",
"pickup_longitude"]])
print("Pickup Clusters are assigned")
print("-"*35)

jan_2016_data = add_pickup_bins(new_frame_cleaned2, 1, 2016)
print("Pickup time bins are assigned")
print("-"*35)

jan_2016_timeBin_groupBy = jan_2016_data[["pickup_cluster", "pickup_bins",
"trip_distance"]].groupby(by = ["pickup_cluster", "pickup_bins"]).count()
print("Pickup cluster and time bins are grouped.")
print("-"*35)

print("Done...")
print("-"*35)

print("Fraction of Total data left = "+str(new_frame_cleaned2.shape[0]/new_frame2.shape[0]))
print("Total Number of outliers removed = "+str(new_frame2.shape[0] - new_frame_cleaned2.shape[0]))
print("-"*35)

print("Total Time taken for execution of Jan 2016 data = "+str(datetime.now() - startTime))
print("-"*35)
```

```
PREPARATION OF JANUARY 2016 DATA.
-----------------------------------
Number of columns = 19
-----------------------------------
New Frame for Jan 2016 creation done
-----------------------------------
Trip Duration Outliers removed
-----------------------------------
Speed Outliers removed
-----------------------------------
Trip Distance Outliers removed
-----------------------------------
Total Amount Outliers removed
-----------------------------------
Pickups outside of NYC are removed
-----------------------------------
Dropoffs outside of NYC are removed
-----------------------------------
Pickup Clusters are assigned
-----------------------------------
Pickup time bins are assigned
-----------------------------------
Pickup cluster and time bins are grouped.
```

```
-------------------------------------
Done...
-------------------------------------
Fraction of Total data left = 0.9702401919966318
Total Number of outliers removed = 324586
-------------------------------------
Total Time taken for execution of Jan 2016 data = 0:04:52.859601
-------------------------------------
```

In [52]:

```
jan_2016_data.head()
```

Out[52]:

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	total_amount	trip_times	pick
5	2	5.52	-73.980118	40.743050	-73.913490	40.763142	20.3	18.50	1.45
6	2	7.45	-73.994057	40.719990	-73.966362	40.789871	27.3	26.75	1.45
7	1	1.20	-73.979424	40.744614	-73.992035	40.753944	10.3	11.90	1.45
8	1	6.00	-73.947151	40.791046	-73.920769	40.865578	19.3	11.20	1.45
9	1	3.21	-73.998344	40.723896	-73.995850	40.688400	12.8	11.10	1.45

In [53]:

```
jan_2016_timeBin_groupBy.head()
```

Out[53]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**7**	104
	8	242
	9	299
	10	327
	11	340

Smoothing

In [54]:

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

# for each cluster region we will collect all the indices of 10min intravels in which the pickups
are happened
# we got an observation that there are some pickpbins that doesnt have any pickups
def return_unq_pickup_bins(frame):
    values = []
    for i in range(0,30): #we have total 30 clusters
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [55]:

```
#now for Jan-2015, we have to find out, how many time_bins are there where there is no pickup in a
ny of the cluster region
unique_binswithPickup_Jan_2015 = return_unq_pickup_bins(jan_2015_frame)
for i in range(30):             #we have total 30 clusters
    print("For cluster ID {}, total number of time bins with no pickup in this clutser region is {
}".format(i, (4464 - len(unique_binswithPickup_Jan_2015[i]))))
    print("-"*90)
```

```
#there are total 4464 time bins in Jan - 2015.
#"unique_binswithPickup_Jan_2015" contains all the unique time bins, where pickup happened. It con
tains 30 sub-arrays as there are 30 clusters
#and each sub-array contains the unique ID of all the time bins where pickup happened in the clust
ers which is the index of that sub-array.
```

```
For cluster ID 0, total number of time bins with no pickup in this clutser region is 26
--------------------------------------------------------------------------------------------
For cluster ID 1, total number of time bins with no pickup in this clutser region is 30
--------------------------------------------------------------------------------------------
For cluster ID 2, total number of time bins with no pickup in this clutser region is 150
--------------------------------------------------------------------------------------------
For cluster ID 3, total number of time bins with no pickup in this clutser region is 35
--------------------------------------------------------------------------------------------
For cluster ID 4, total number of time bins with no pickup in this clutser region is 170
--------------------------------------------------------------------------------------------
For cluster ID 5, total number of time bins with no pickup in this clutser region is 40
--------------------------------------------------------------------------------------------
For cluster ID 6, total number of time bins with no pickup in this clutser region is 320
--------------------------------------------------------------------------------------------
For cluster ID 7, total number of time bins with no pickup in this clutser region is 35
--------------------------------------------------------------------------------------------
For cluster ID 8, total number of time bins with no pickup in this clutser region is 39
--------------------------------------------------------------------------------------------
For cluster ID 9, total number of time bins with no pickup in this clutser region is 46
--------------------------------------------------------------------------------------------
For cluster ID 10, total number of time bins with no pickup in this clutser region is 98
--------------------------------------------------------------------------------------------
For cluster ID 11, total number of time bins with no pickup in this clutser region is 32
--------------------------------------------------------------------------------------------
For cluster ID 12, total number of time bins with no pickup in this clutser region is 37
--------------------------------------------------------------------------------------------
For cluster ID 13, total number of time bins with no pickup in this clutser region is 326
--------------------------------------------------------------------------------------------
For cluster ID 14, total number of time bins with no pickup in this clutser region is 35
--------------------------------------------------------------------------------------------
For cluster ID 15, total number of time bins with no pickup in this clutser region is 29
--------------------------------------------------------------------------------------------
For cluster ID 16, total number of time bins with no pickup in this clutser region is 25
--------------------------------------------------------------------------------------------
For cluster ID 17, total number of time bins with no pickup in this clutser region is 40
--------------------------------------------------------------------------------------------
For cluster ID 18, total number of time bins with no pickup in this clutser region is 30
--------------------------------------------------------------------------------------------
For cluster ID 19, total number of time bins with no pickup in this clutser region is 35
--------------------------------------------------------------------------------------------
For cluster ID 20, total number of time bins with no pickup in this clutser region is 40
--------------------------------------------------------------------------------------------
For cluster ID 21, total number of time bins with no pickup in this clutser region is 38
--------------------------------------------------------------------------------------------
For cluster ID 22, total number of time bins with no pickup in this clutser region is 34
--------------------------------------------------------------------------------------------
For cluster ID 23, total number of time bins with no pickup in this clutser region is 49
--------------------------------------------------------------------------------------------
For cluster ID 24, total number of time bins with no pickup in this clutser region is 49
--------------------------------------------------------------------------------------------
For cluster ID 25, total number of time bins with no pickup in this clutser region is 27
--------------------------------------------------------------------------------------------
For cluster ID 26, total number of time bins with no pickup in this clutser region is 26
--------------------------------------------------------------------------------------------
For cluster ID 27, total number of time bins with no pickup in this clutser region is 720
--------------------------------------------------------------------------------------------
For cluster ID 28, total number of time bins with no pickup in this clutser region is 34
--------------------------------------------------------------------------------------------
For cluster ID 29, total number of time bins with no pickup in this clutser region is 29
--------------------------------------------------------------------------------------------
```

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 [56]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 0 to the smoothed data
# we finally return smoothed data
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,30):
        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 [57]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the
above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,30):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already visited/resolved
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup bin if it
exists
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin
value which has a pickup value
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing,hence 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/((right_hand_lim
t-i)+2)*1.0
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missing,hence
we have no left-limit here
                    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 [58]:

```
jan_2015_fillZero = fill_missing(jan_2015_groupby["trip_distance"].values,
unique_binswithPickup_Jan_2015)
# here in jan_2015_timeBin_groupBy dataframe the "trip_distance" represents the number of pickups
that are happened.
jan_2015_fillSmooth = smoothing(jan_2015_groupby["trip_distance"].values,
unique_binswithPickup_Jan_2015)

#"unique_binswithPickup_Jan_2015" contains all the unique time bins, where pickup happened. It con
tains 30 sub-arrays as there are 30 clusters
#and each sub-array contains the unique ID of all the time bins where pickup happened in the clust
ers which is the index of that sub-array.
```

In [59]:

```
def countZeros(num):
    count = 0
    for i in num:
        if i == 0:
            count += 1
    return count
```

In [60]:

```
print("Number of values filled with zero in zero fill data= "+str(countZeros(jan_2015_fillZero)))
```

```
Number of values filled with zero in zero fill data= 2803
```

In [61]:

```
print("Sanity check for number of zeros in smoothed data = "+str(countZeros(jan_2015_fillSmooth)))
```

```
Sanity check for number of zeros in smoothed data = 0
```

In [62]:

```
print("Total number of pickup values = "+str(len(jan_2015_fillZero)))
print("Total number of pickup values = "+str(len(jan_2015_fillSmooth)))
```

```
Total number of pickup values = 133920
Total number of pickup values = 133920
```

In [63]:

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

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

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

In [64]:

```
unique_binswithPickup_Jan_2016 =  return_unq_pickup_bins(jan_2016_data)
```

In [65]:

```
# Jan-2015 data is smoothed, Jan-2016 data missing values are filled with zero
jan_2016_fillZero = fill_missing(jan_2016_timeBin_groupBy["trip_distance"].values,
unique_binswithPickup_Jan_2016)
```

In [66]:

```
regionWisePickup_Jan_2016 = []
for i in range(30):
    regionWisePickup_Jan_2016.append(jan_2016_fillZero[4464*i:((4464*i)+4464)])
#"regionWisePickup_Jan_2016" is a list of lists which contains 30 sub lists, where the index of ea
ch sub-list is the
#corresponding cluster number and the element of each sub-list is the pickup value. So, we know th
at there are 4464 time bins
#in Jan 2016, hence, each sub-list is of size 4464.
```

In [67]:

```
print(len(regionWisePickup_Jan_2016))
print(len(regionWisePickup_Jan_2016[0]))
```

```
30
4464
```

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

In [68]:

```
Ratios_DF = pd.DataFrame()
Ratios_DF["Given"] = jan_2015_fillSmooth
Ratios_DF["Prediction"] = jan_2016_fillZero
Ratios_DF["Ratio"] = Ratios_DF["Prediction"]*1.0/Ratios_DF["Given"]*1.0
```

In [69]:

```
Ratios_DF.head(15)
```

Out[69]:

	Given	Prediction	Ratio
0	18	0	0.000000
1	18	0	0.000000
2	18	0	0.000000
3	18	0	0.000000
4	18	0	0.000000
5	18	0	0.000000
6	18	0	0.000000
7	18	104	5.777778
8	262	242	0.923664
9	311	299	0.961415
10	326	327	1.003067
11	381	340	0.892388
12	363	316	0.870523
13	373	325	0.871314
14	344	323	0.938953

In [70]:

```
Ratios_DF.shape
```

Out[70]:

```
(133920, 3)
```

In [71]:

```
print("Total Number of zeros in Ratio column = "+str(Ratios_DF["Ratio"].value_counts()[0]))
```

```
Total Number of zeros in Ratio column = 4315
```

In [72]:

```
print("Total Number of zeros in Prediction column = "+str(Ratios_DF["Prediction"].value_counts()[0
]))
```

```
Total Number of zeros in Prediction column = 4315
```

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

In [73]:

```
#simple_moving_average_ratios
def MA_R_Predictions(ratios):
    predicted_ratio=(ratios['Ratio'].values)[0]
    error=[]
    predicted_values=[]
    window_size=3
```

```
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        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['Pred
iction'].values)[i],1))))
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratio'].values)[(i+1)-window_size:(i+1)])/window_size
        else:
            predicted_ratio=sum((ratios['Ratio'].values)[0:(i+1)])/(i+1)


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

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

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

In [74]:

```
#simple_moving_average_predictions
def MA_P_Predictions(ratios):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=1
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        predicted_values.append(predicted_value)
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        if i+1>=window_size:
            predicted_value=int(sum((ratios['Prediction'].values)[(i+1)-window_size:
(i+1)])/window_size)
        else:
            predicted_value=int(sum((ratios['Prediction'].values)[0:(i+1)])/(i+1))

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

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

Weighted Moving Averages

The Moving Avergaes Model used gave equal importance to all the values in the window used, but we know intuitively that the future 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 [75]:

```
#weighted_moving_average_ratios
def WA_R_Predictions(ratios):
```

```
    predicted_ratio=(ratios['Ratio'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        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['Pred
iction'].values)[i],1))))
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratio'].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['Ratio'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

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

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

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

In [77]:

```
#weighted_moving_average_predictions
def WA_P_Predictions(ratios):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,4464*30):
        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['Prediction'].v
```

```
alues))
    mse_err = sum([e**2 for e in error])/len(error)
    return ratios,mape_err,mse_err
```

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

Exponential Weighted Moving Averages

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

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

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

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

In [78]:

```
#exponential_weighted_moving_average_ratios
def EA_R1_Predictions(ratios):
    predicted_ratio=(ratios['Ratio'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*30):
        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['Pred
iction'].values)[i],1))))
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratio'].values)[i])

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

In [80]:

```
#exponential_weighted_moving_average_predictions
def EA_P1_Predictions(ratios):
    predicted_value= (ratios['Prediction'].values)[0]
    alpha=0.3
    error=[]
    predicted_values=[]
    for i in range(0,4464*30):
        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))))
```

```
        error.append(abs((math.pow(predicted_value-(ratios['Prediction'].values)[i],1))))
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Prediction'].values)[i]))
)

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

In [81]:

```
r1, mape1, mse1 = MA_R_Predictions(Ratios_DF)
r2, mape2, mse2 = MA_P_Predictions(Ratios_DF)
r3, mape3, mse3 = WA_R_Predictions(Ratios_DF)
r4, mape4, mse4 = WA_P_Predictions(Ratios_DF)
r5, mape5, mse5 = EA_R1_Predictions(Ratios_DF)
r6, mape6, mse6 = EA_P1_Predictions(Ratios_DF)
```

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 [82]:

```
error_table_baseline = pd.DataFrame(columns = ["Model", "MAPE(%)", "MSE"])

error_table_baseline = error_table_baseline.append(pd.DataFrame([["Simple Moving Average Ratios",
mape1*100, mse1]], columns = ["Model", "MAPE(%)", "MSE"]))
error_table_baseline = error_table_baseline.append(pd.DataFrame([["Simple Moving Average
Predictions", mape2*100, mse2]], columns = ["Model", "MAPE(%)", "MSE"]))
error_table_baseline = error_table_baseline.append(pd.DataFrame([["Weighted Moving Average Ratios"
, mape3*100, mse3]], columns = ["Model", "MAPE(%)", "MSE"]))
error_table_baseline = error_table_baseline.append(pd.DataFrame([["Weighted Moving Average
Predictions", mape4*100, mse4]], columns = ["Model", "MAPE(%)", "MSE"]))
error_table_baseline = error_table_baseline.append(pd.DataFrame([["Exponential Weighted Moving Ave
rage Ratios", mape5*100, mse5]], columns = ["Model", "MAPE(%)", "MSE"]))
error_table_baseline = error_table_baseline.append(pd.DataFrame([["Exponential Weighted Moving Ave
rage Predictions", mape6*100, mse6]], columns = ["Model", "MAPE(%)", "MSE"]))

error_table_baseline.reset_index(drop = True, inplace = True)
```

In [83]:

```
error_table_baseline.style.highlight_min(axis=0)
```

Out[83]:

	Model	MAPE(%)	MSE
0	Simple Moving Average Ratios	17.2765	687.828
1	Simple Moving Average Predictions	13.3644	317.68
2	Weighted Moving Average Ratios	17.0286	661.128
3	Weighted Moving Average Predictions	12.8186	290.815
4	Exponential Weighted Moving Average Ratios	16.9733	658.212
5	Exponential Weighted Moving Average Predictions	12.8048	288.04

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 [84]:

```
#"regionWisePickup_Jan_2016" is a list of lists which contains 30 sub lists, where the index of ea
ch sub-list is the
#corresponding cluster number and the element of each sub-list is the pickup value. So, we know th
at there are 4464 time bins
#in Jan 2016, hence, each sub-list is of size 4464.
#"regionWisePickup_Jan_2016" is a list of lists [[x1,x2,x3..4464], [x1,x2,x3..4464],
[x1,x2,x3..4464], [x1,x2,x3..4464], [x1,x2,x3..4464], .. 30 lists]
#Here, x1,x2,x3... are pickup values at time stamp 1,2,3... respectively

# print(len(regionWisePickup_Jan_2016))
# 30
# print(len(regionWisePickup_Jan_2016[0]))
# 4464

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

# TruePickups varaible
# it is list of lists
# It will be used as true labels/ground truth. Now since we are taking previous 5 pickups as a tra
ining data for predicting
# next pickup(here next pickup will be a true/ground truth pickup), so "TruePickups" will not cont
ain first five pickups of each
# cluster. It will contain number of pickups 4459 for each cluster.
TruePickups = []


# lat will contain 4464-5=4459 times latitude of cluster center for every cluster.
# Ex: [[cent_lat 4459 times],[cent_lat 4459 times], [cent_lat 4459 times].... 30 lists]
# it is list of lists
lat = []


# lon will contain 4464-5=4459 times longitude of cluster center for every cluster.
# Ex: [[cent_lat 4459 times],[cent_lat 4459 times], [cent_lat 4459 times].... 30 lists]
# it is list of lists
lon = []

# we will code each day
# sunday = 0, monday=1, tue = 2, wed=3, thur=4, fri=5,sat=6
day_of_week = []


# for every cluster we will be adding 4459 values, each value represent to which day of the week t
hat pickup bin belongs to
# it is list of lists

# feat is a numbpy array, of shape (133770, 5). {4459*30 = 133770.}
# each row corresponds to an entry in our data
# for the first row we will have [f0,f1,f2,f3,f4] fi=number of pickups happened in i+1st 10min int
ravel(bin)
# the second row will have [f1,f2,f3,f4,f5]
# the third row will have [f2,f3,f4,f5,f6]
# and so on...
feat = []


centerOfRegions = kmeans.cluster_centers_
feat = [0]*number_of_time_stamps
for i in range(30):
    lat.append([centerOfRegions[i][0]]*4459)
    lon.append([centerOfRegions[i][1]]*4459)
    #1 January 2016 is a Friday so we start our day from 5: "(int(j/144))%7+5"
    # Our prediction starts from 5th 10min interval since we need to have number of pickups that a
```

```
re happened in last 5 pickup bins.
    day_of_week.append([int(((int(j/144)%7)+5)%7) for j in range(5, 4464)])
    #"regionWisePickup_Jan_2016" is a list of lists which contains 30 sub lists, where the index o
f each sub-list is the
    #corresponding cluster number and the element of each sub-list is the pickup value. So, we kno
w that there are 4464 time bins
    #in Jan 2016, hence, each sub-list is of size 4464.
    #"regionWisePickup_Jan_2016" is a list of lists [[x1,x2,x3..4464], [x1,x2,x3..4464],
[x1,x2,x3..4464], [x1,x2,x3..4464], [x1,x2,x3..4464], .. 30 lists]
    #Here, x1,x2,x3... are pickup values at time stamp 1,2,3... respectively
    feat = np.vstack((feat, [regionWisePickup_Jan_2016[i][k:k+number_of_time_stamps] for k in range
(0, len(regionWisePickup_Jan_2016[i]) - (number_of_time_stamps))]))
    TruePickups.append(regionWisePickup_Jan_2016[i][5:])
    #output contains pickup values of all the regions and of each time stamp, except first 5 time
stamp pickups of each region.
feat = feat[1:]
```

In [85]:

```
len(lat[0])*len(lat) == len(lon[0])*len(lon) == len(day_of_week[0])*len(day_of_week) == 4459*30 ==
len(feat) == len(TruePickups[0])*len(TruePickups)
```

Out[85]:

```
True
```

Adding Predictions of Weighted Moving Average Predictions as a feature in our data

Getting the predictions of weighted moving averages to be used as a feature in cumulative form.

Upto now we computed 8 features for every data point that starts from 50th min of the day.

1. cluster center latitude
2. cluster center longitude
3. day of the week
4. f_t_1: number of pickups that are happened previous t-1st 10min interval
5. f_t_2: number of pickups that are happened previous t-2nd 10min interval
6. f_t_3: number of pickups that are happened previous t-3rd 10min interval
7. f_t_4: number of pickups that are happened previous t-4th 10min interval
8. f_t_5: number of pickups that are happened previous t-5th 10min interval

From the baseline models we said that the weighted moving avarage predictions gives us the best error. We will try to add the same weighted moving avarage predictions at time t as a feature to our data.
Weighted Moving Average -> $P_t = (N * P_{t-1} + (N-1) * P_{t-2} + (N-2) * P_{t-3} \ldots 1 * P_{t-n})/(N * (N+1)/2)$

In [86]:

```
# "predicted_pickup_values": it is a temporary array that store weighted moving avarag prediction
values for each 10min intervl,
# for each cluster it will get reset.
# for every cluster it contains 4464 values
predicted_pickup_values = []

# "predicted_pickup_values_list"
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x4463], [x5,x6,x7..x4463], [x5,x6,x7..x4463], ... 40
lists]
predicted_pickup_values_list = []

predicted_value = -1  #it will contain cuurent predicted_value. Default is given -1 which will be
replaced later

window_size = 2
for i in range(30):
    for j in range(4464):
        if j == 0:
            predicted_value = regionWisePickup_Jan_2016[i][j]
            predicted_pickup_values.append(0)
        else:
            if j>=window_size:
                sumPickups = 0
```

```
                sumOfWeights = 0
                for k in range(window_size, 0, -1):
                    sumPickups += k*(regionWisePickup_Jan_2016[i][j -window_size + (k - 1)])
                    sumOfWeights += k
                predicted_value = int(sumPickups/sumOfWeights)
                predicted_pickup_values.append(predicted_value)
            else:
                sumPickups = 0
                sumOfWeights = 0
                for k in range(j, 0, -1):
                    sumPickups += k*regionWisePickup_Jan_2016[i][k-1]
                    sumOfWeights += k
                predicted_value = int(sumPickups/sumOfWeights)
                predicted_pickup_values.append(predicted_value)

    predicted_pickup_values_list.append(predicted_pickup_values[5:])
    predicted_pickup_values = []
```

In [87]:

```
len(predicted_pickup_values_list[0])*len(predicted_pickup_values_list) == 4459*30
```

Out[87]:

```
True
```

Adding Fourier Features

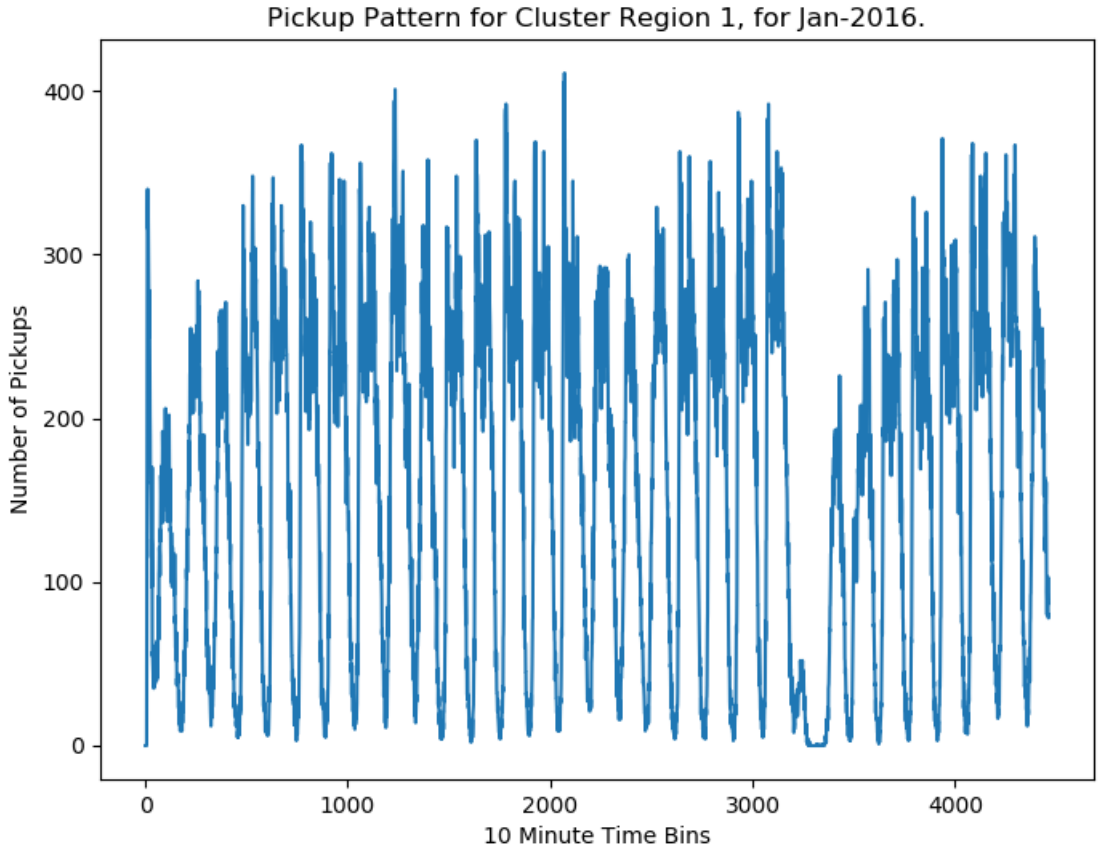
In [88]:

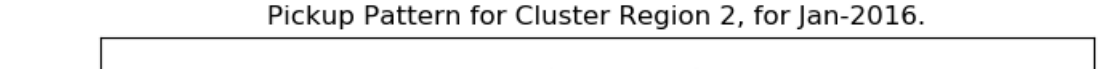
```
%matplotlib notebook
for i in range(30):
    fig = plt.figure(figsize = (8, 6))
    plt.plot(regionWisePickup_Jan_2016[i][:4464])
    plt.title("Pickup Pattern for Cluster Region "+str(i+1)+", for Jan-2016.")
    plt.xlabel("10 Minute Time Bins")
    plt.ylabel("Number of Pickups")
```

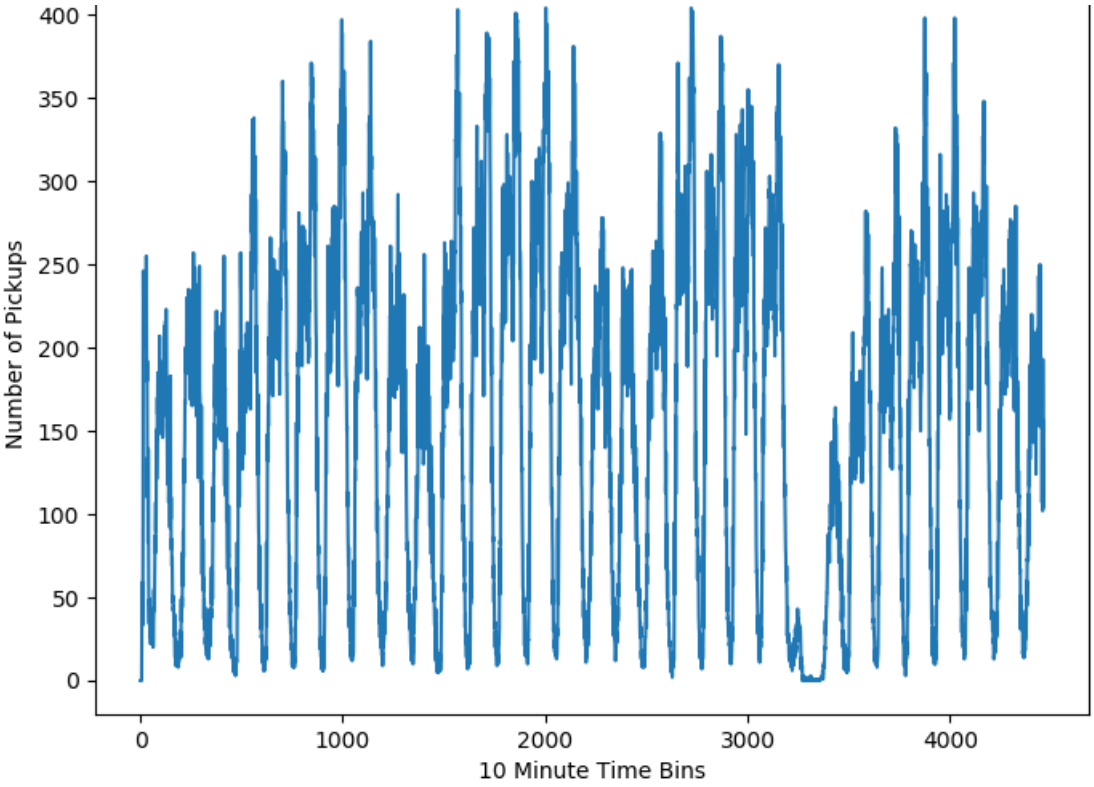
```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line 
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py", 
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54, 
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line 
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py", 
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54, 
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line 
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py", 
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54, 
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
```

```
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
```

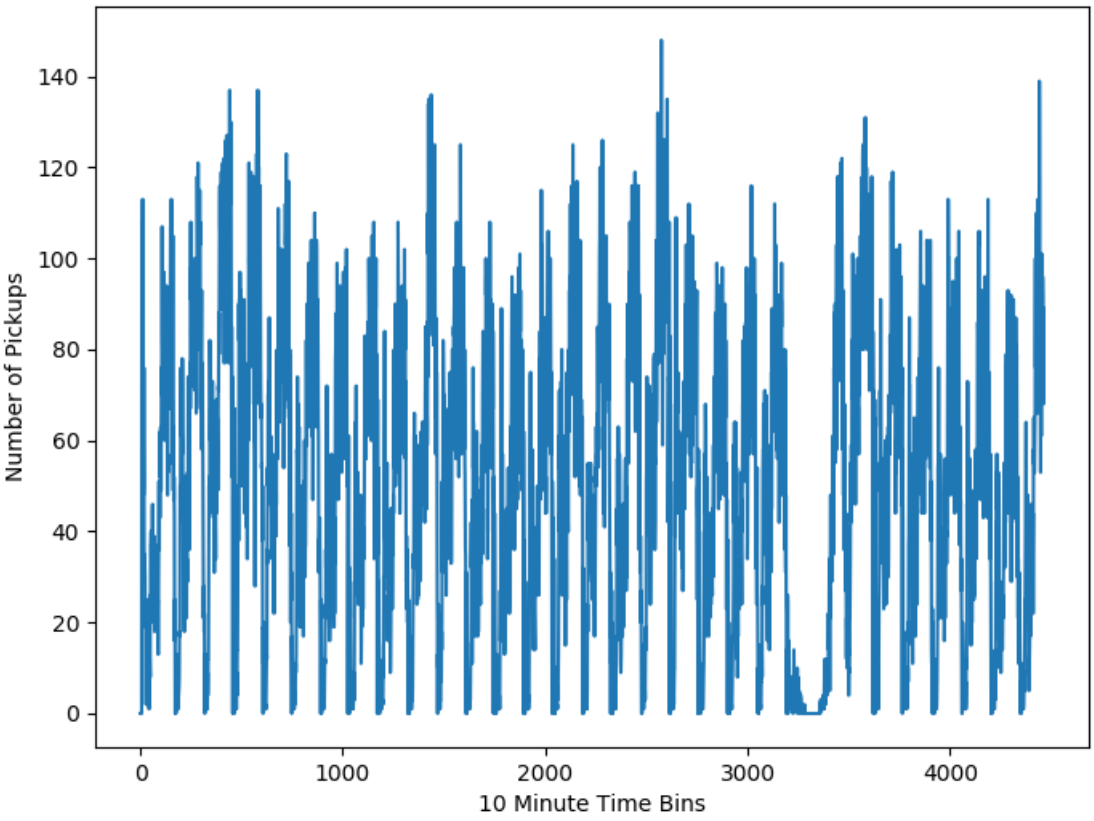
```
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/__init__.py", line
215, in process
    func(*args, **kwargs)
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/backends/backend_nbagg.py",
line 236, in <lambda>
    canvas.mpl_connect('close_event', lambda event: Gcf.destroy(num))
  File "/home/ubuntu/anaconda3/lib/python3.6/site-packages/matplotlib/_pylab_helpers.py", line 54,
in destroy
    cls._activeQue.remove(manager)
ValueError: list.remove(x): x not in list
```



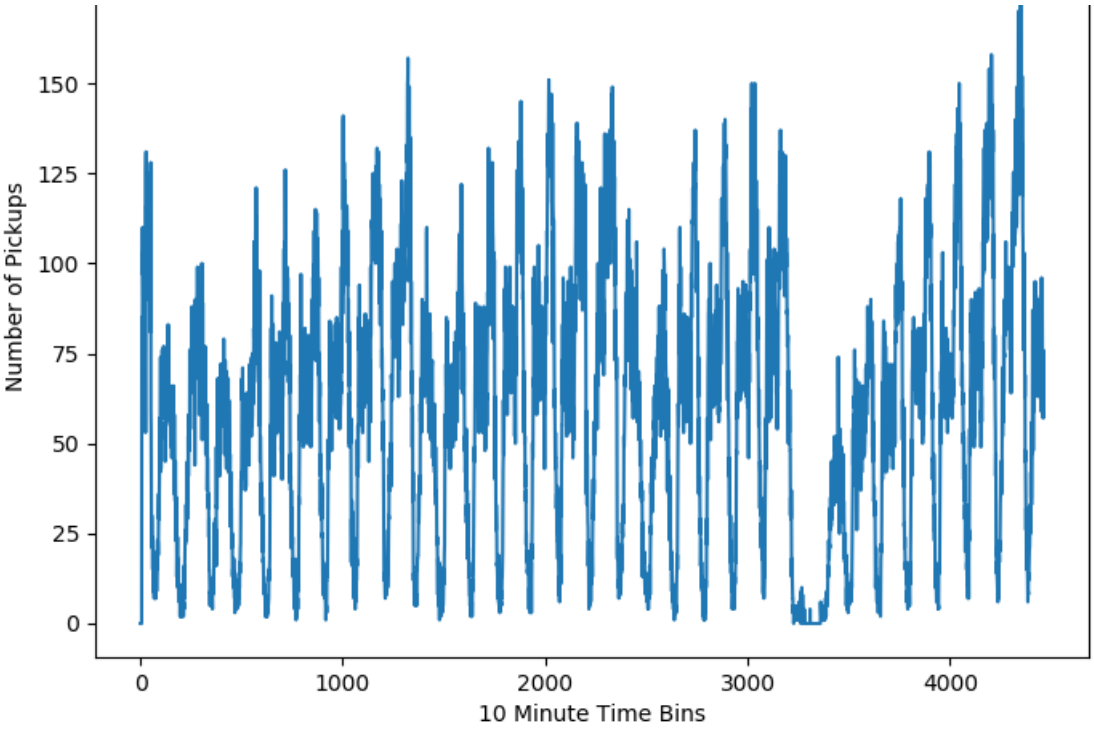


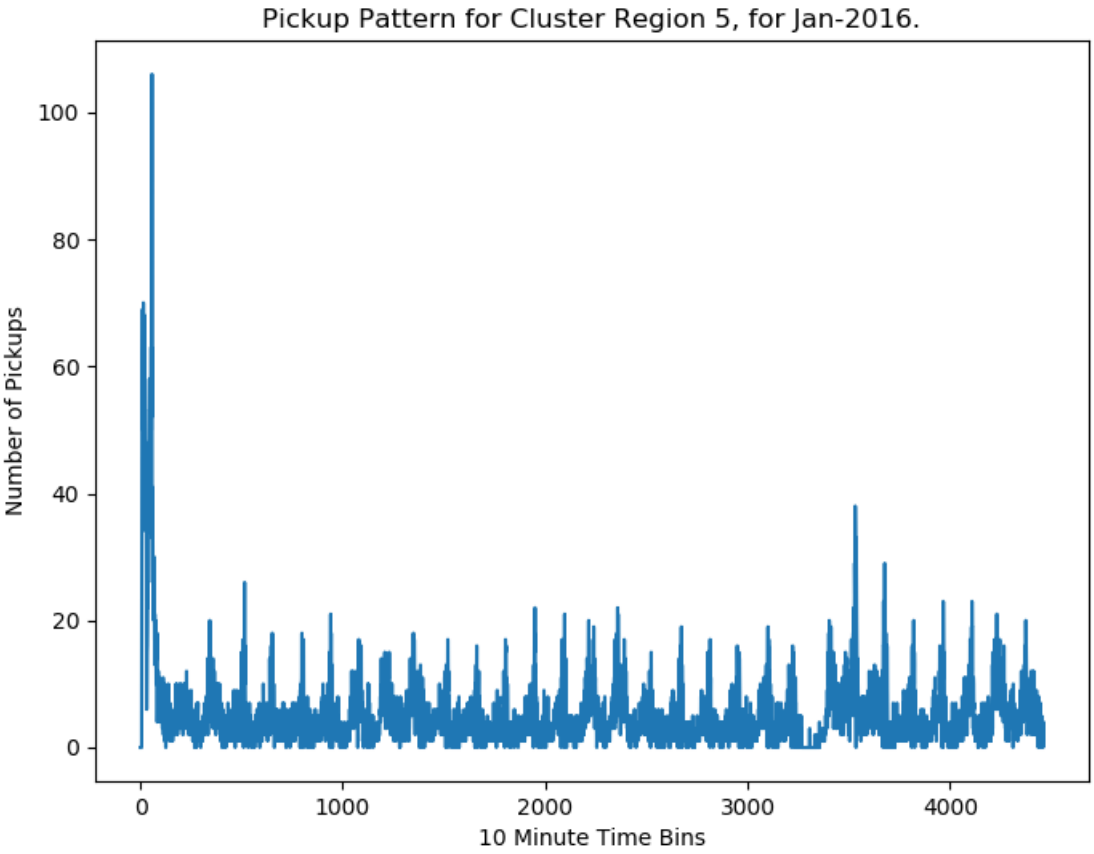


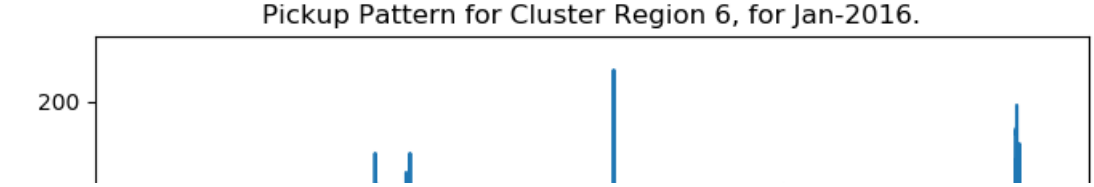
Pickup Pattern for Cluster Region 3, for Jan-2016.

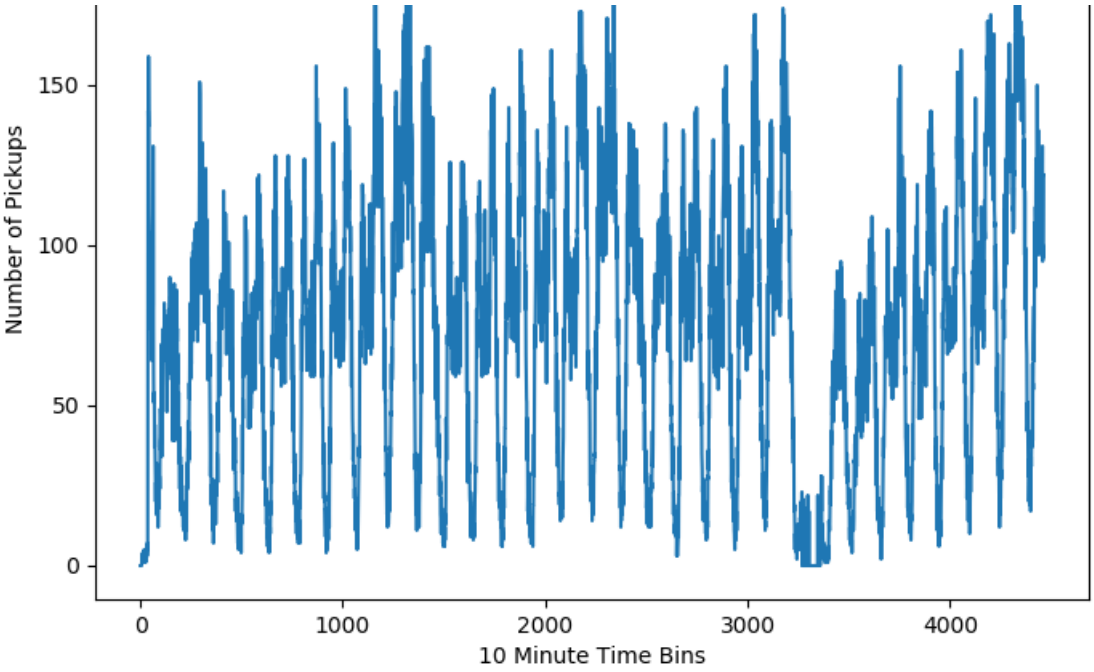


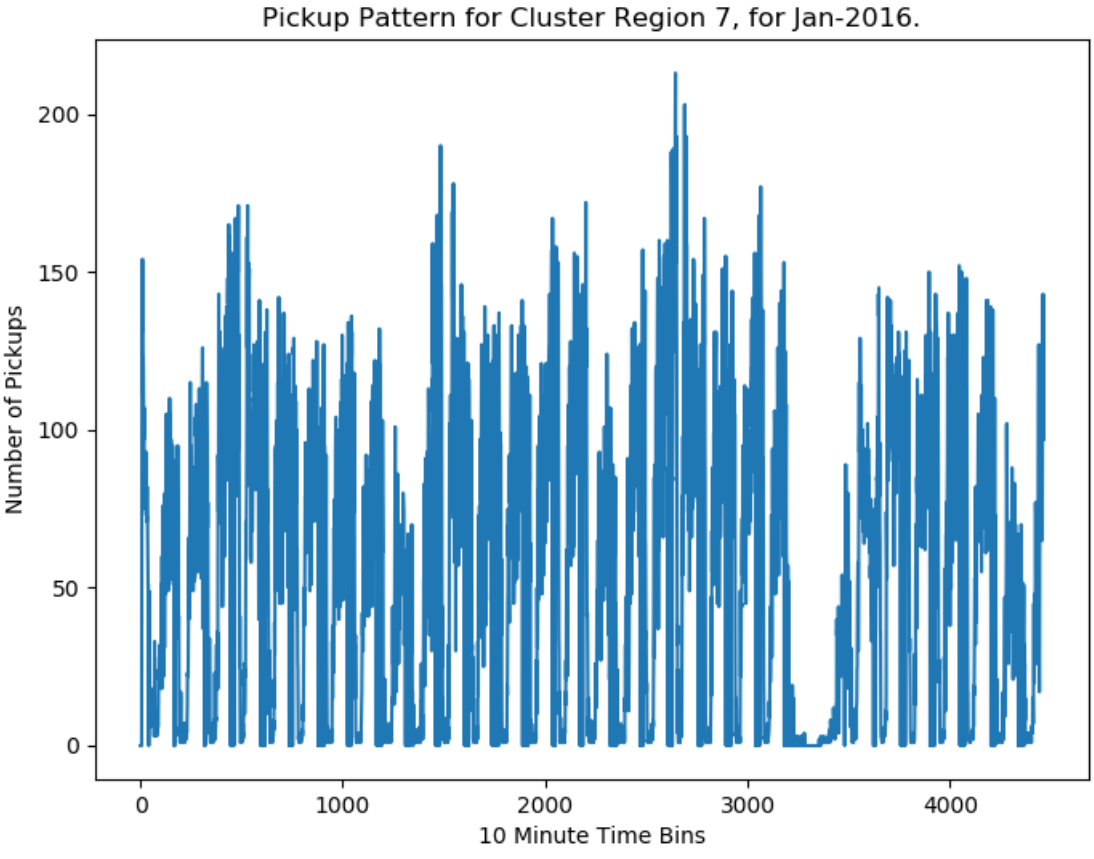
Pickup Pattern for Cluster Region 4, for Jan-2016.

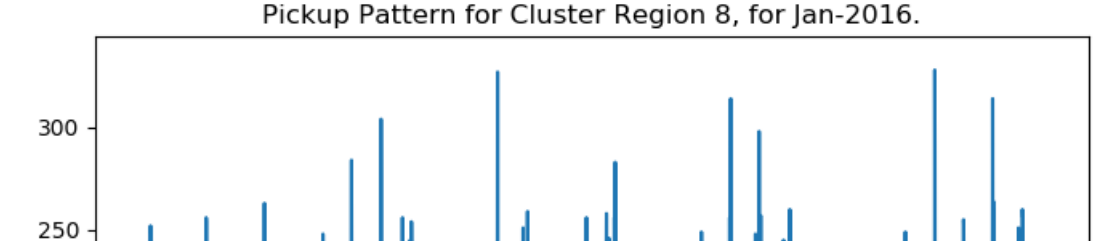

Number of Pickups
150
125
100
75
50
25
0
0
1000
2000
3000
4000
10 Minute Time Bins

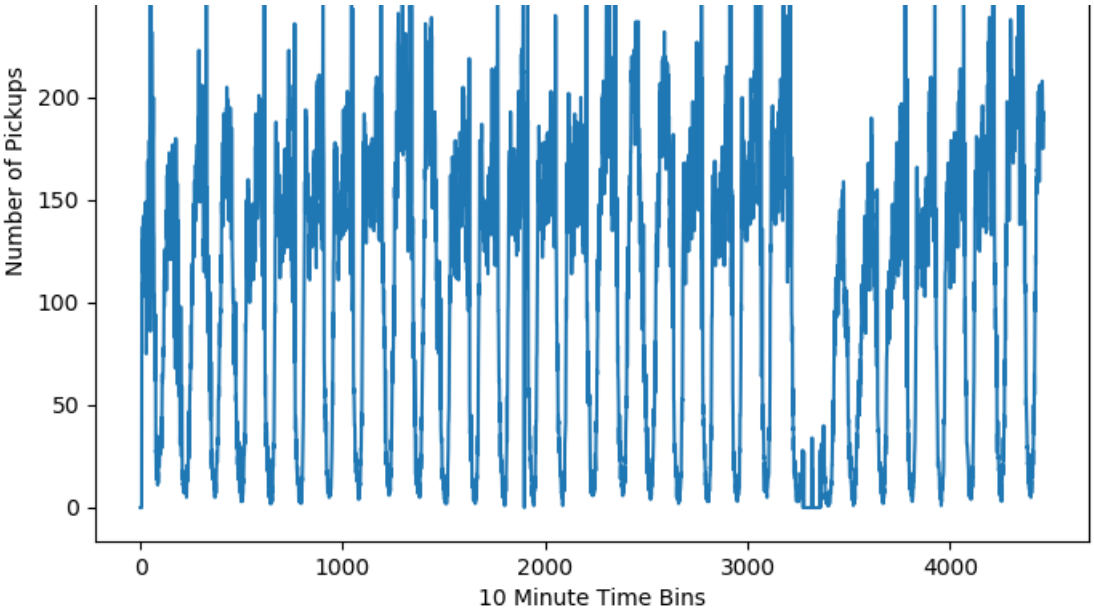

Pickup Pattern for Cluster Region 5, for Jan-2016.
Number of Pickups
100
80
60
40
20
0
0
1000
2000
3000
4000
10 Minute Time Bins


Pickup Pattern for Cluster Region 6, for Jan-2016.
200

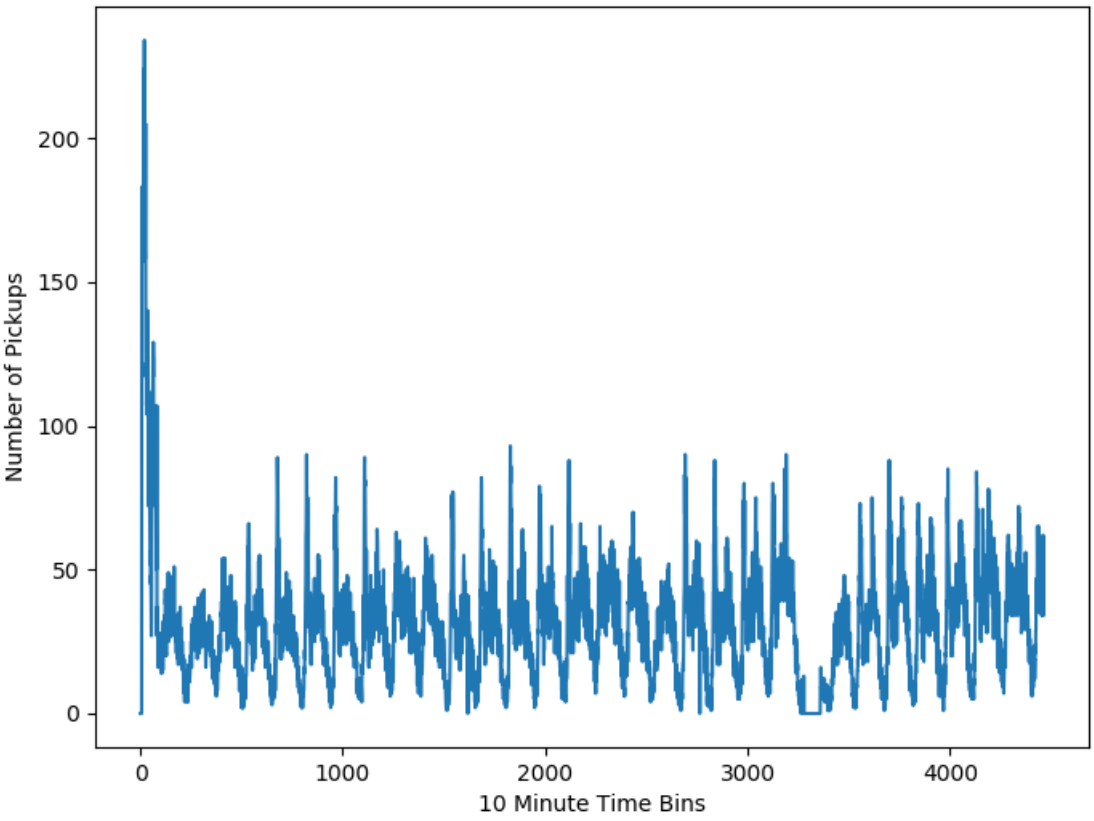

Number of Pickups
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins


Pickup Pattern for Cluster Region 7, for Jan-2016.
Number of Pickups
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

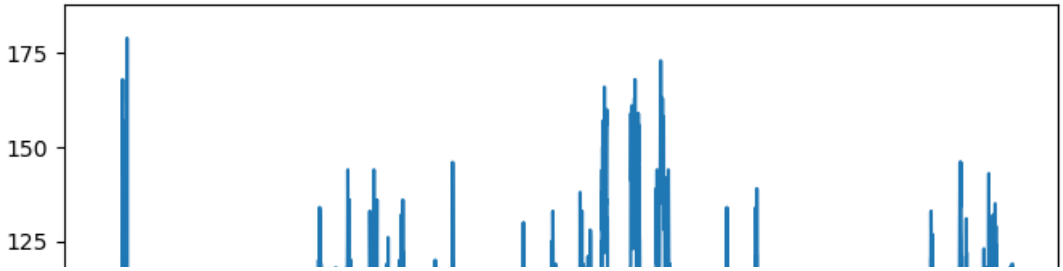

Pickup Pattern for Cluster Region 8, for Jan-2016.
300
250

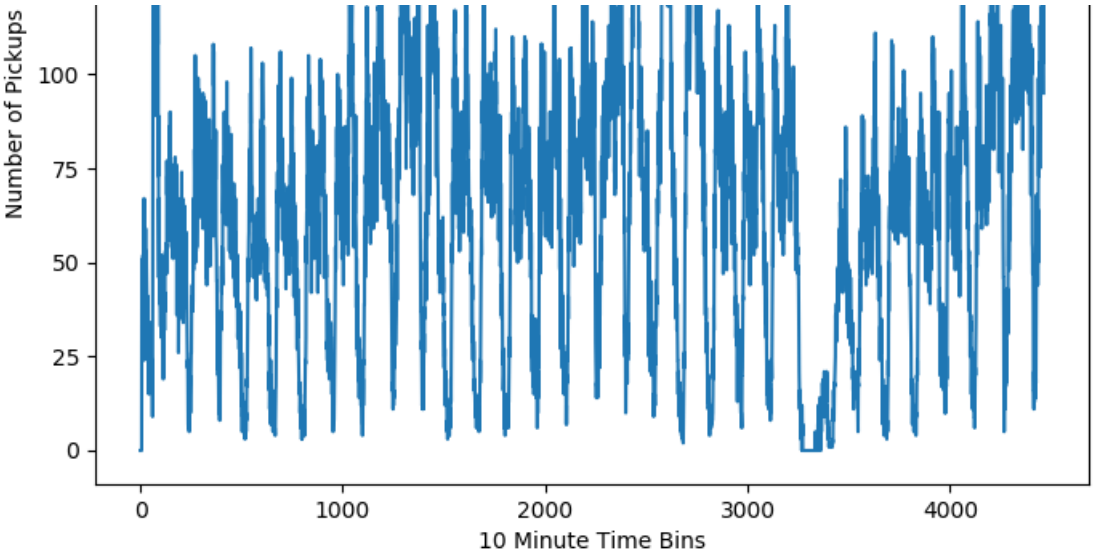


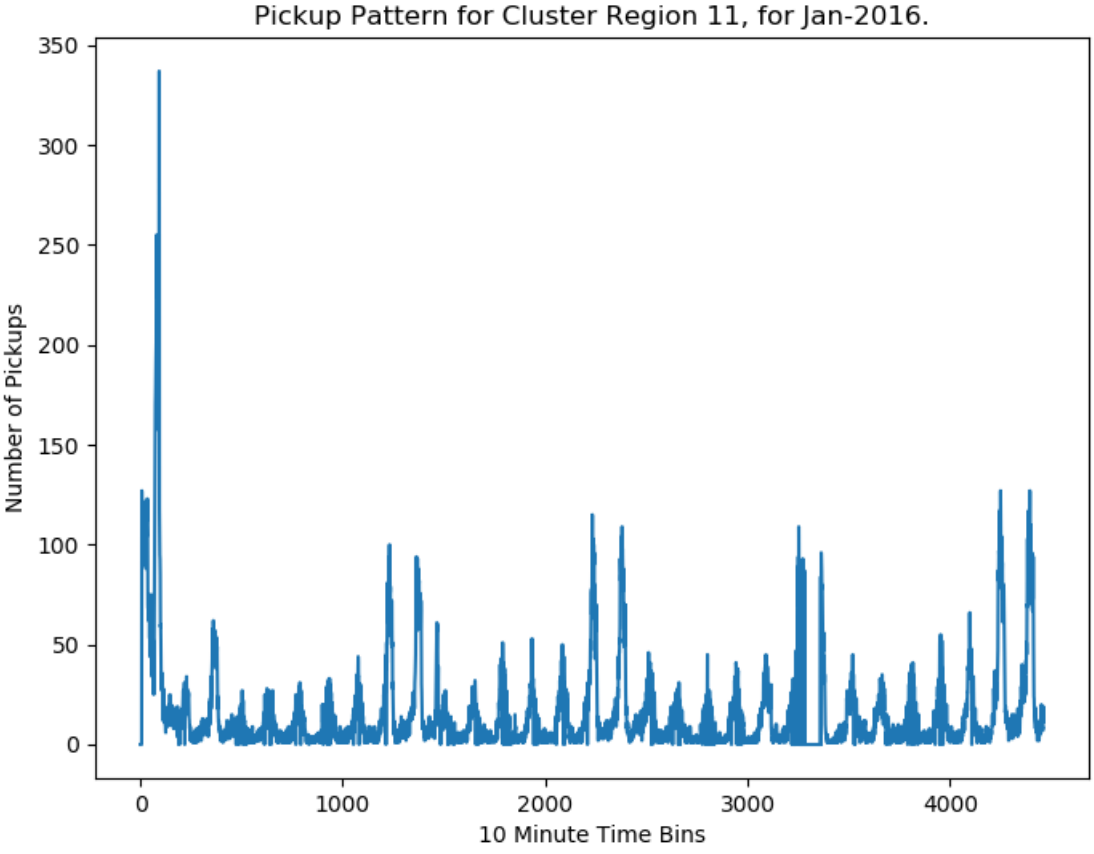
Pickup Pattern for Cluster Region 9, for Jan-2016.

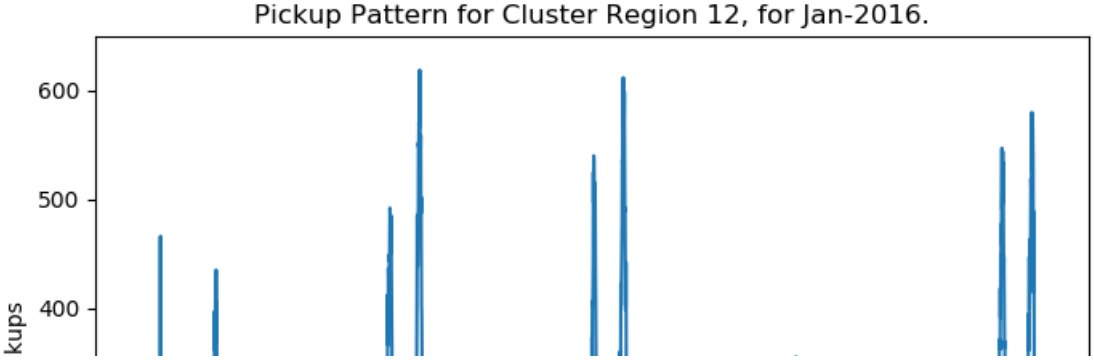


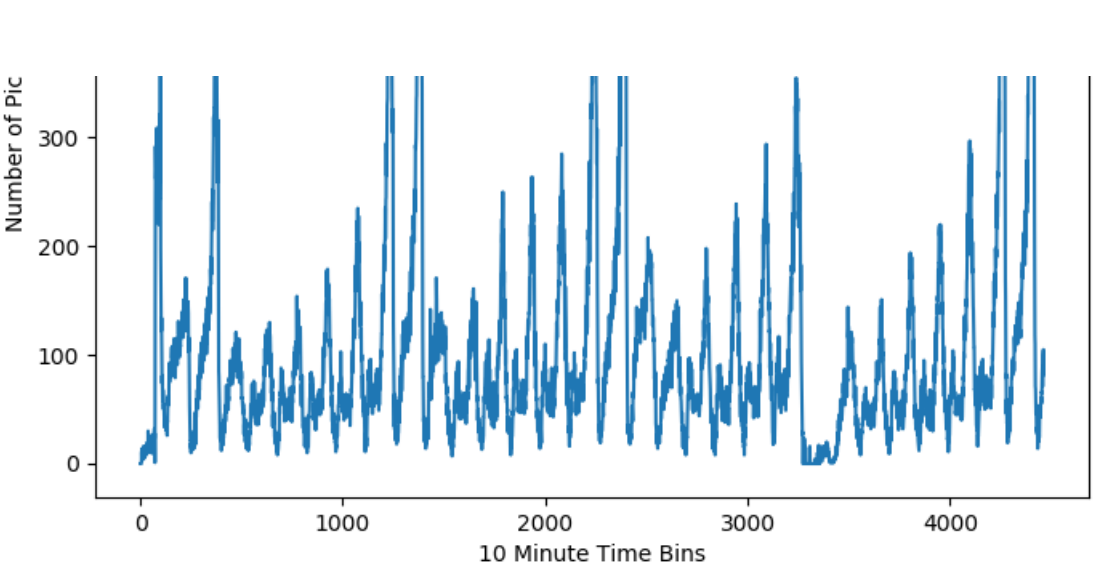
Pickup Pattern for Cluster Region 10, for Jan-2016.

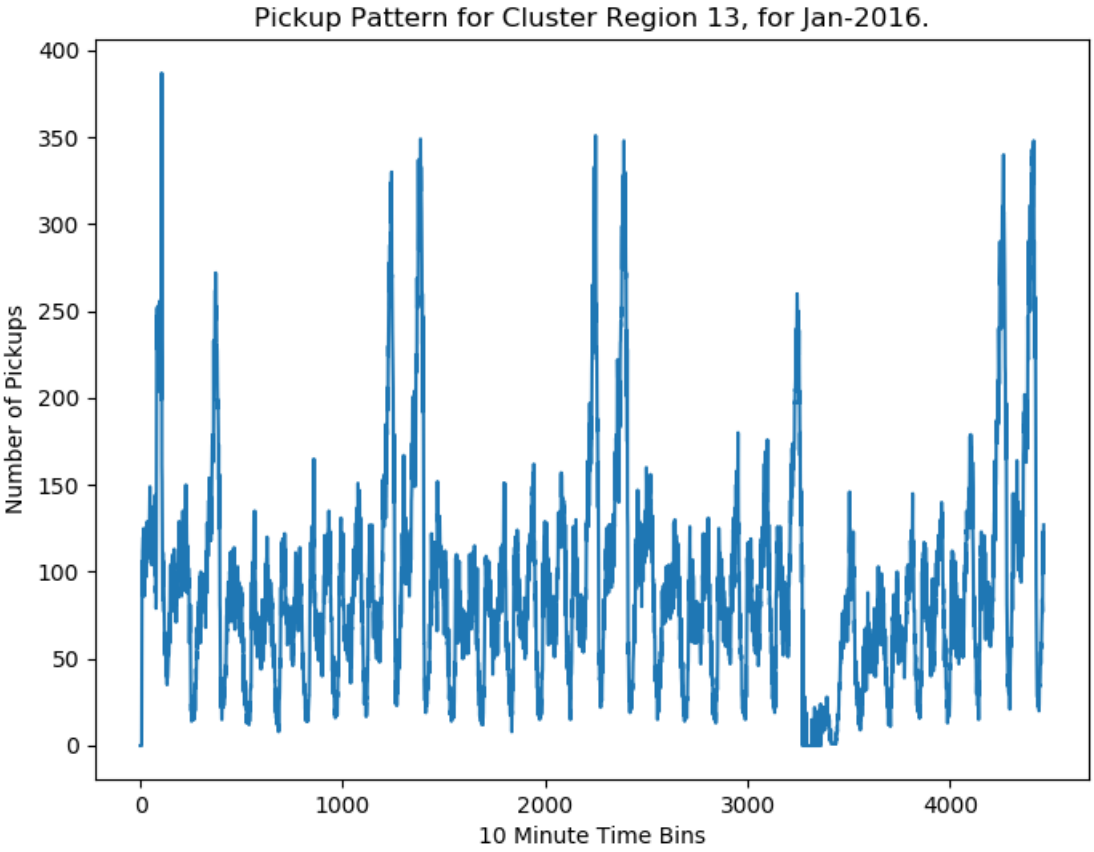


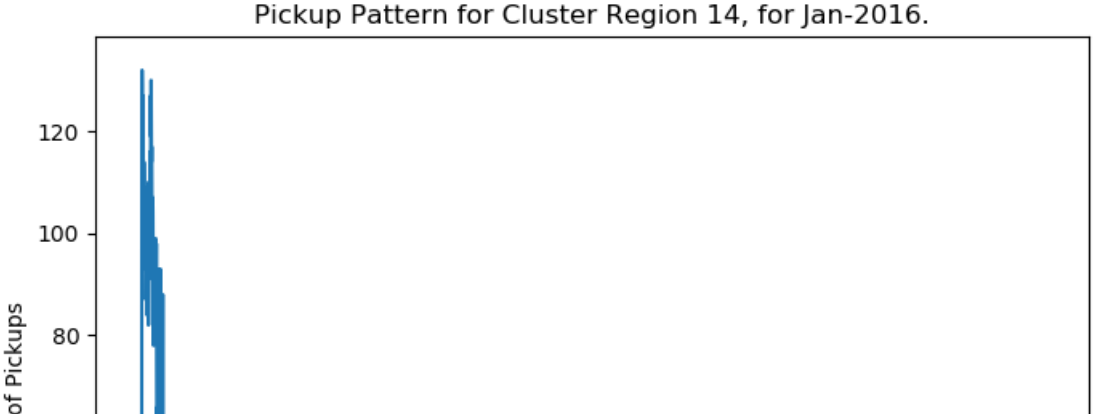

Number of Pickups
100
75
50
25
0
0
1000
2000
3000
4000
10 Minute Time Bins

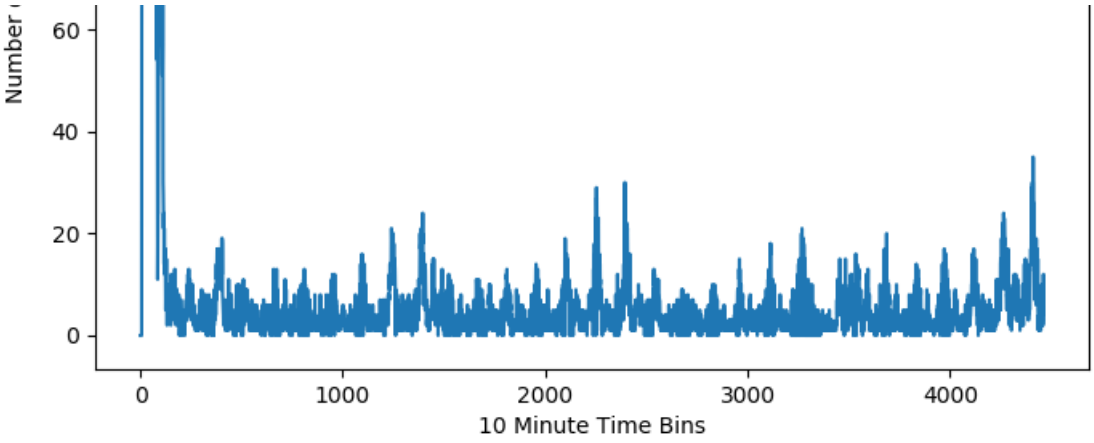

Pickup Pattern for Cluster Region 11, for Jan-2016.
Number of Pickups
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

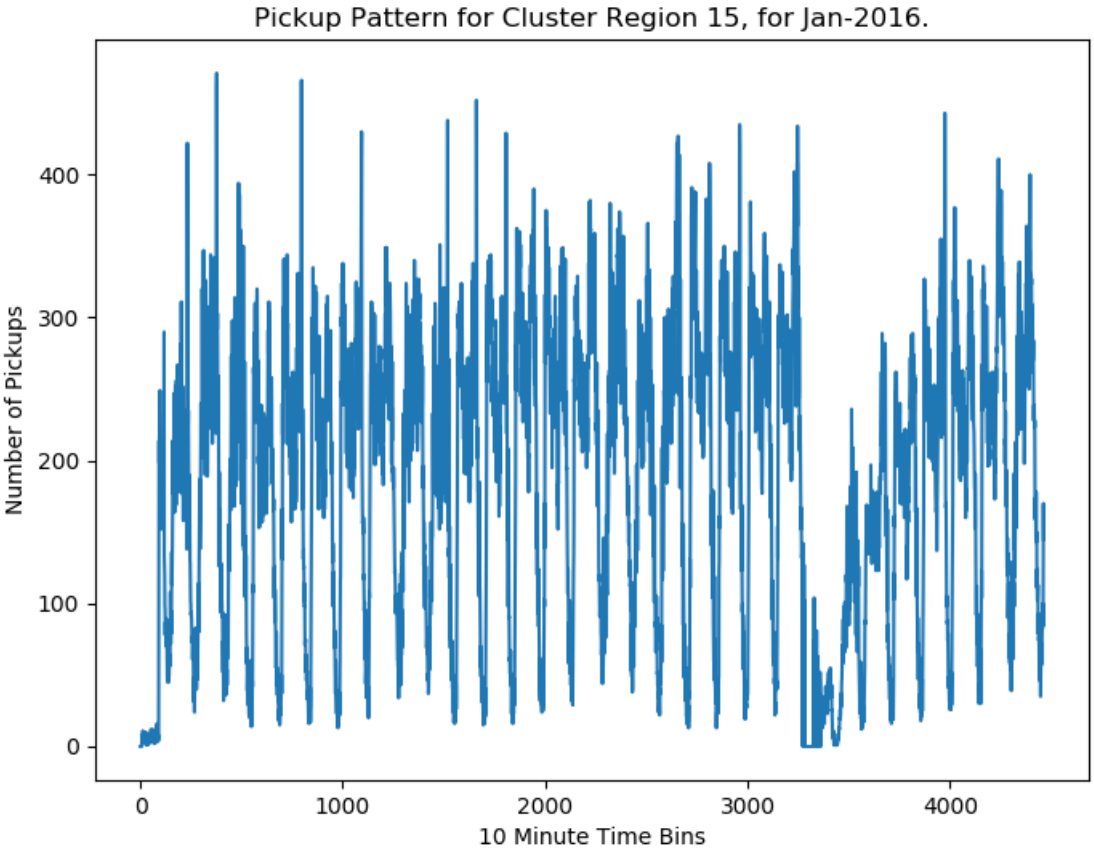

Pickup Pattern for Cluster Region 12, for Jan-2016.
600
500
400

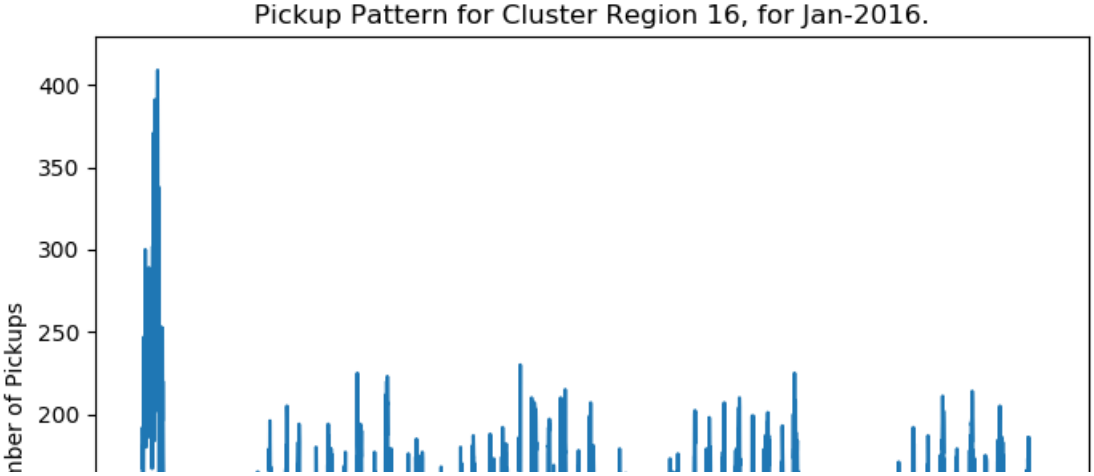

Number of Pic
300
200
100
0
0
1000
2000
3000
4000
10 Minute Time Bins

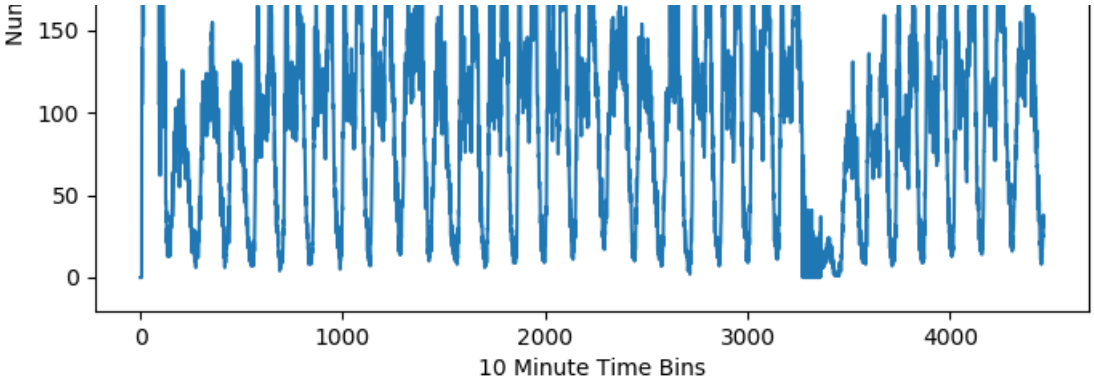

Pickup Pattern for Cluster Region 13, for Jan-2016.
Number of Pickups
400
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

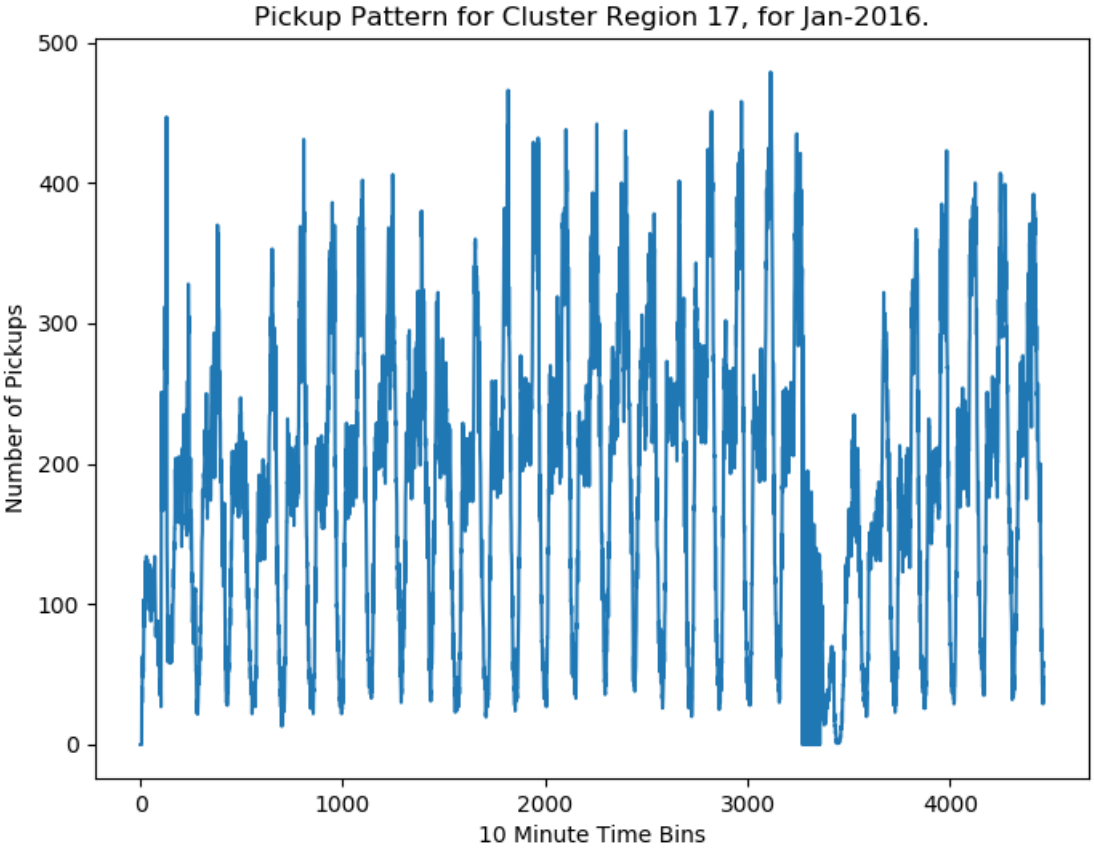

Pickup Pattern for Cluster Region 14, for Jan-2016.
of Pickups
120
100
80

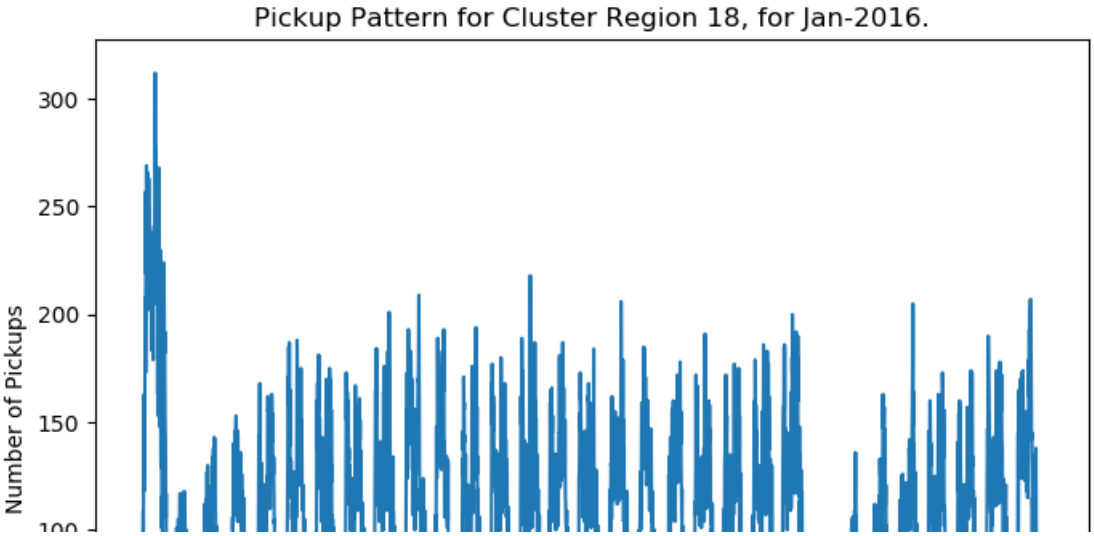

60
40
20
0
0
1000
2000
3000
4000
10 Minute Time Bins

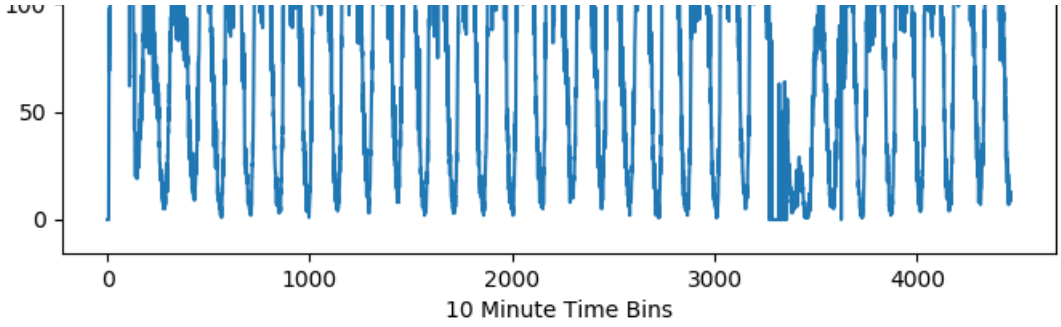

Pickup Pattern for Cluster Region 15, for Jan-2016.
Number of Pickups
400
300
200
100
0
0
1000
2000
3000
4000
10 Minute Time Bins

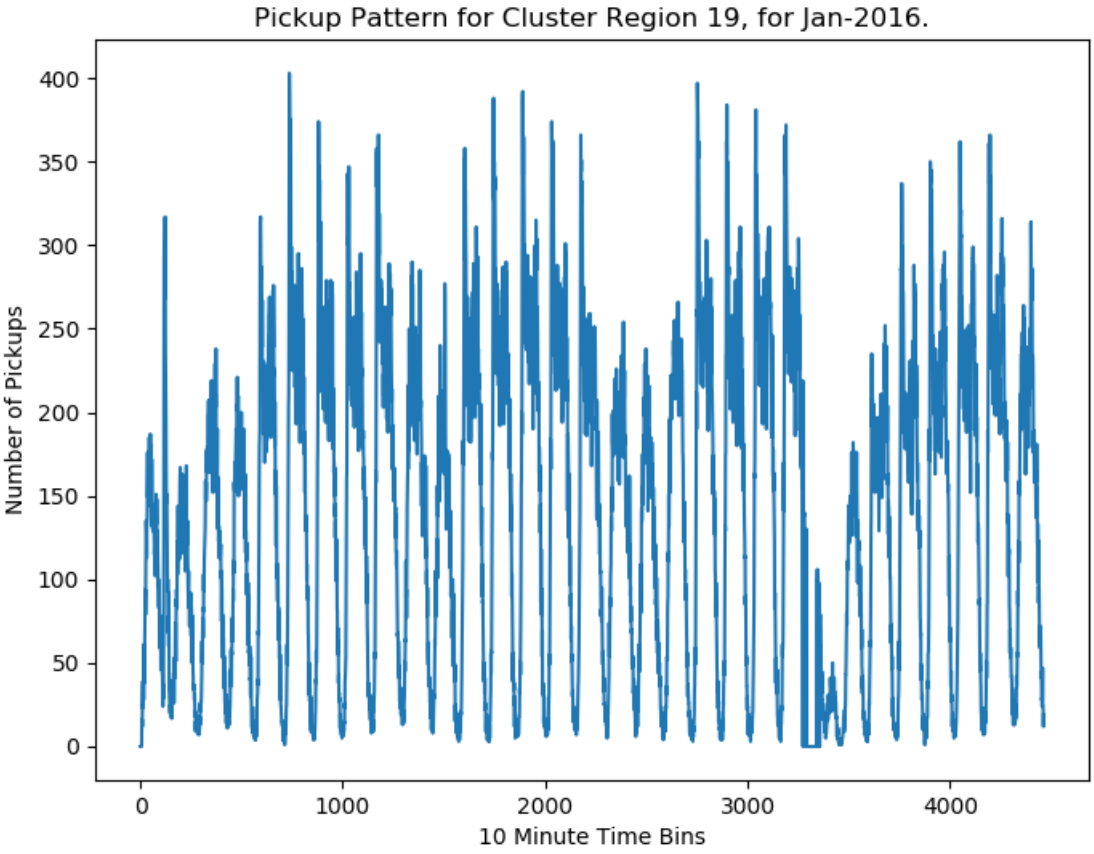

Pickup Pattern for Cluster Region 16, for Jan-2016.
400
350
300
250
200

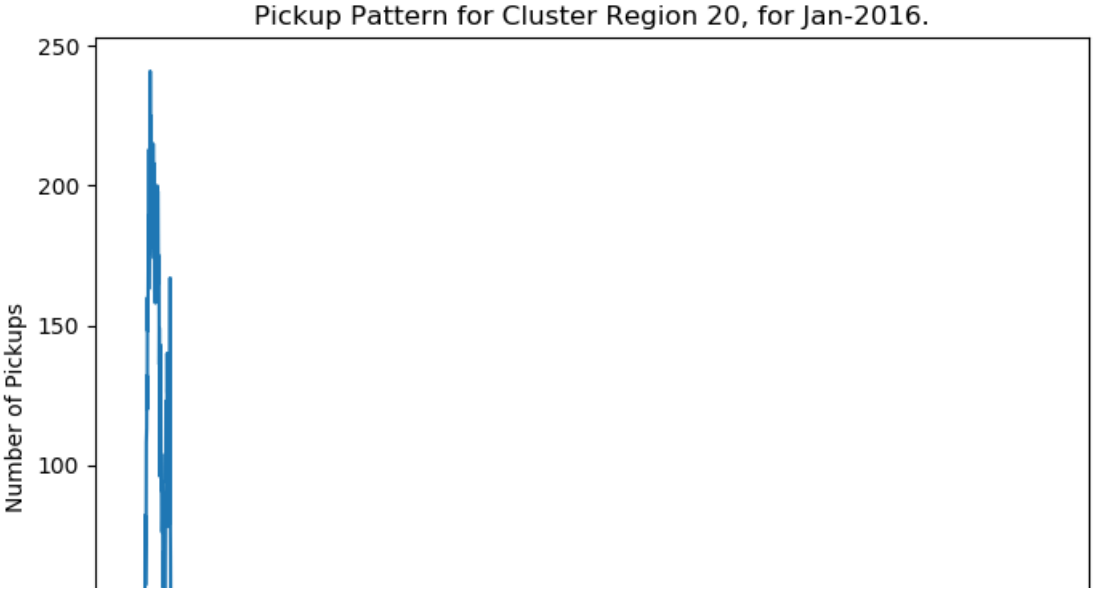

Num
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

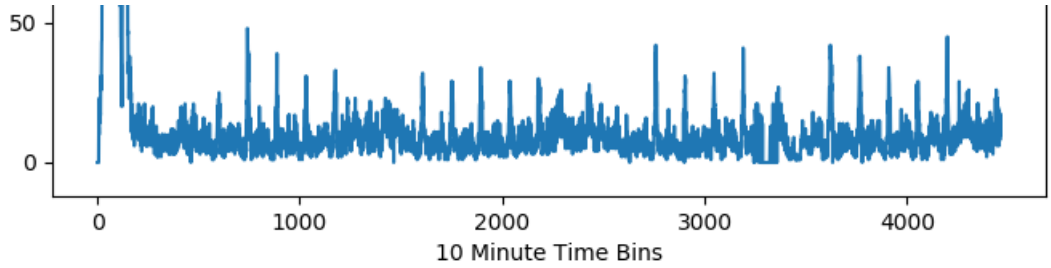

Pickup Pattern for Cluster Region 17, for Jan-2016.
Number of Pickups
500
400
300
200
100
0
0
1000
2000
3000
4000
10 Minute Time Bins

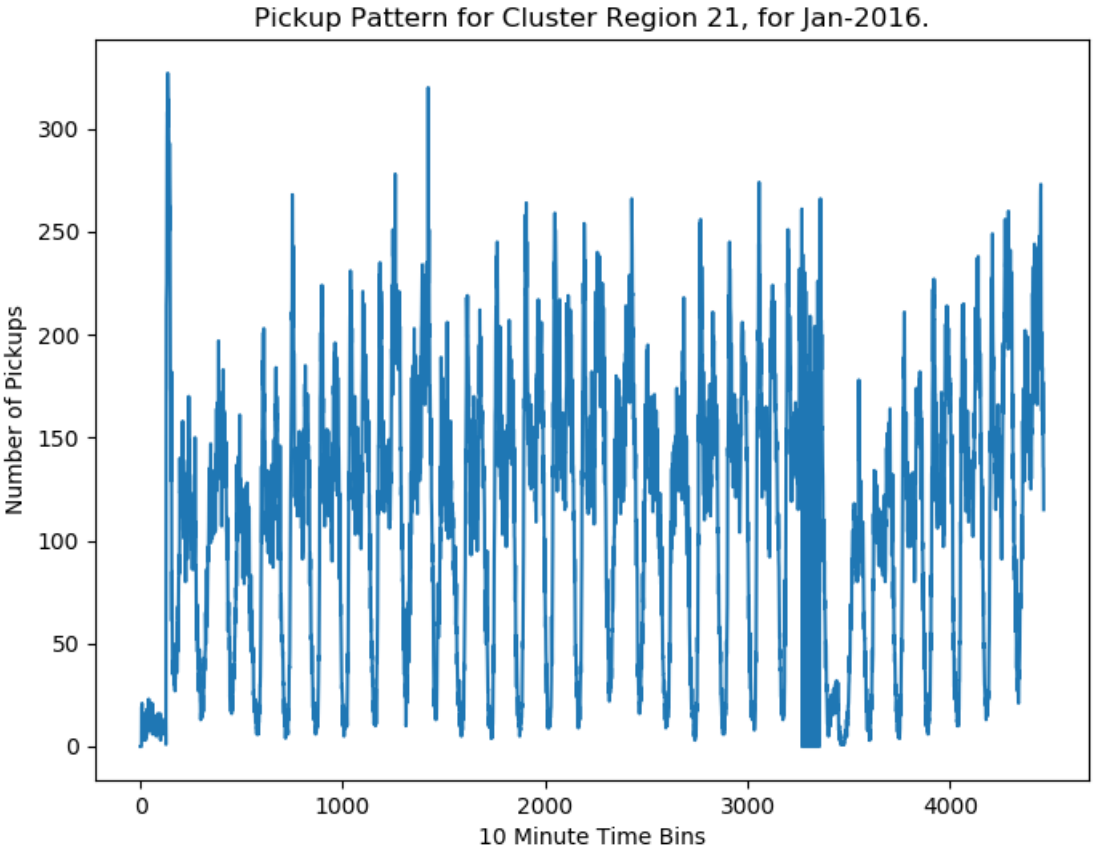

Pickup Pattern for Cluster Region 18, for Jan-2016.
Number of Pickups
300
250
200
150

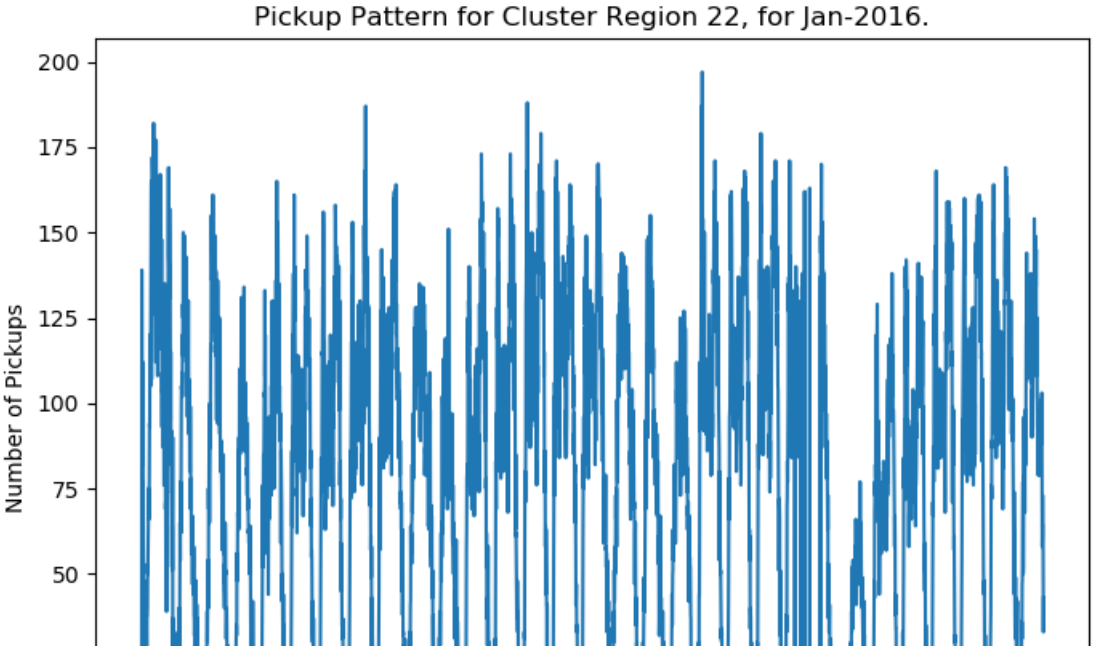

100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

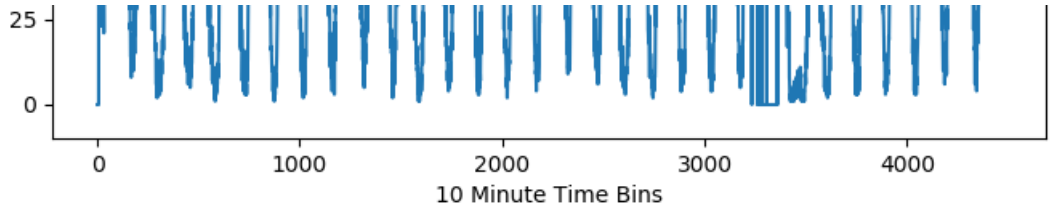

Pickup Pattern for Cluster Region 19, for Jan-2016.
Number of Pickups
400
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

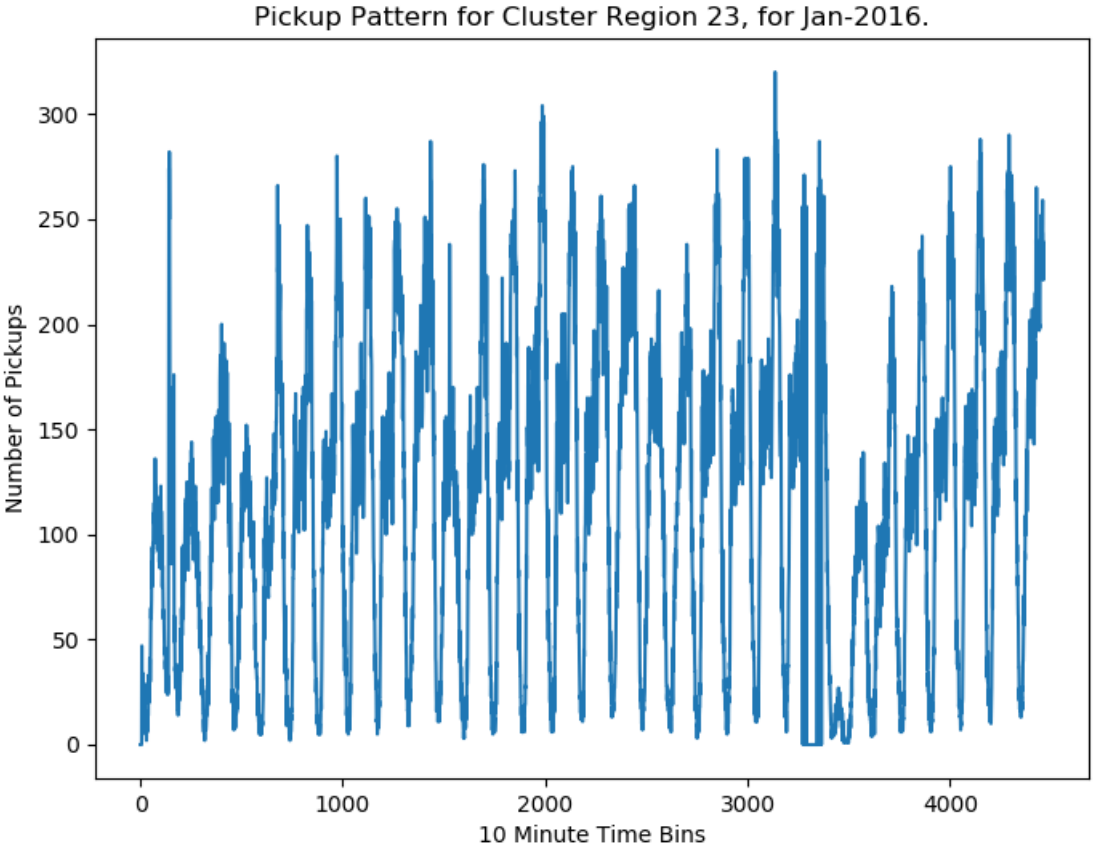

Pickup Pattern for Cluster Region 20, for Jan-2016.
Number of Pickups
250
200
150
100

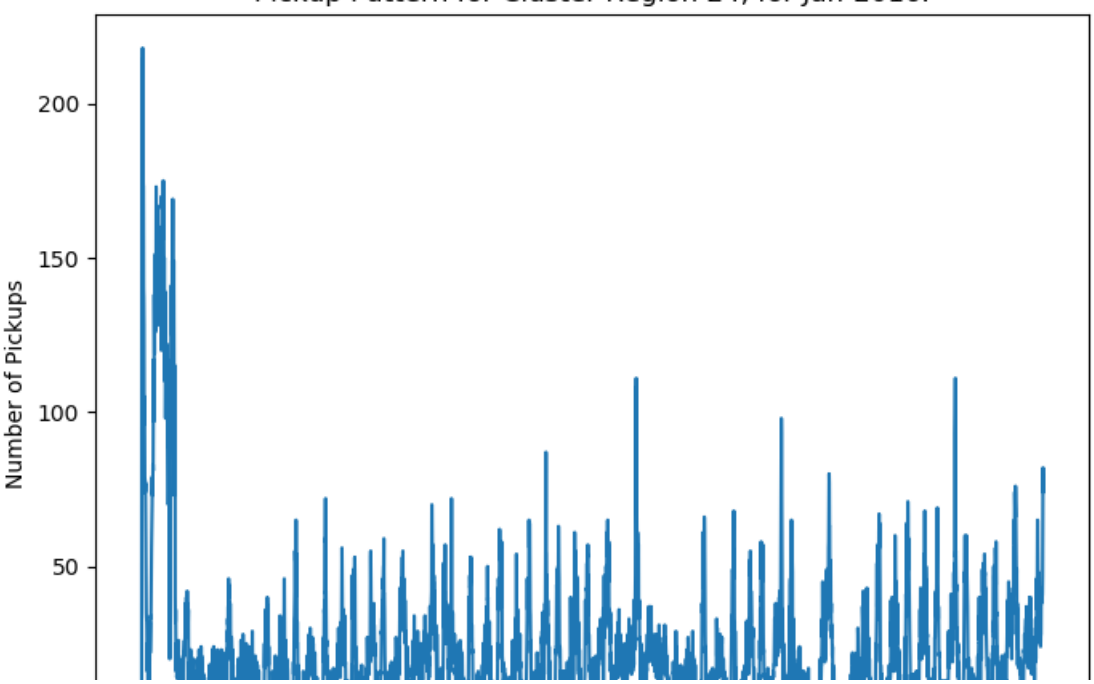

50
0
0
1000
2000
3000
4000
10 Minute Time Bins

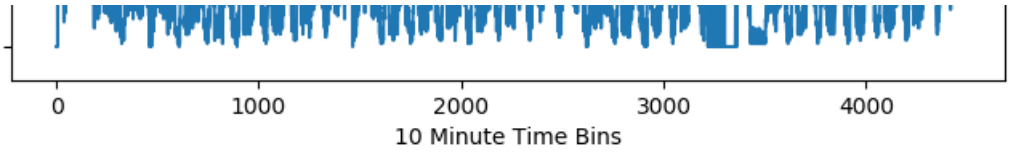

Pickup Pattern for Cluster Region 21, for Jan-2016.
Number of Pickups
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

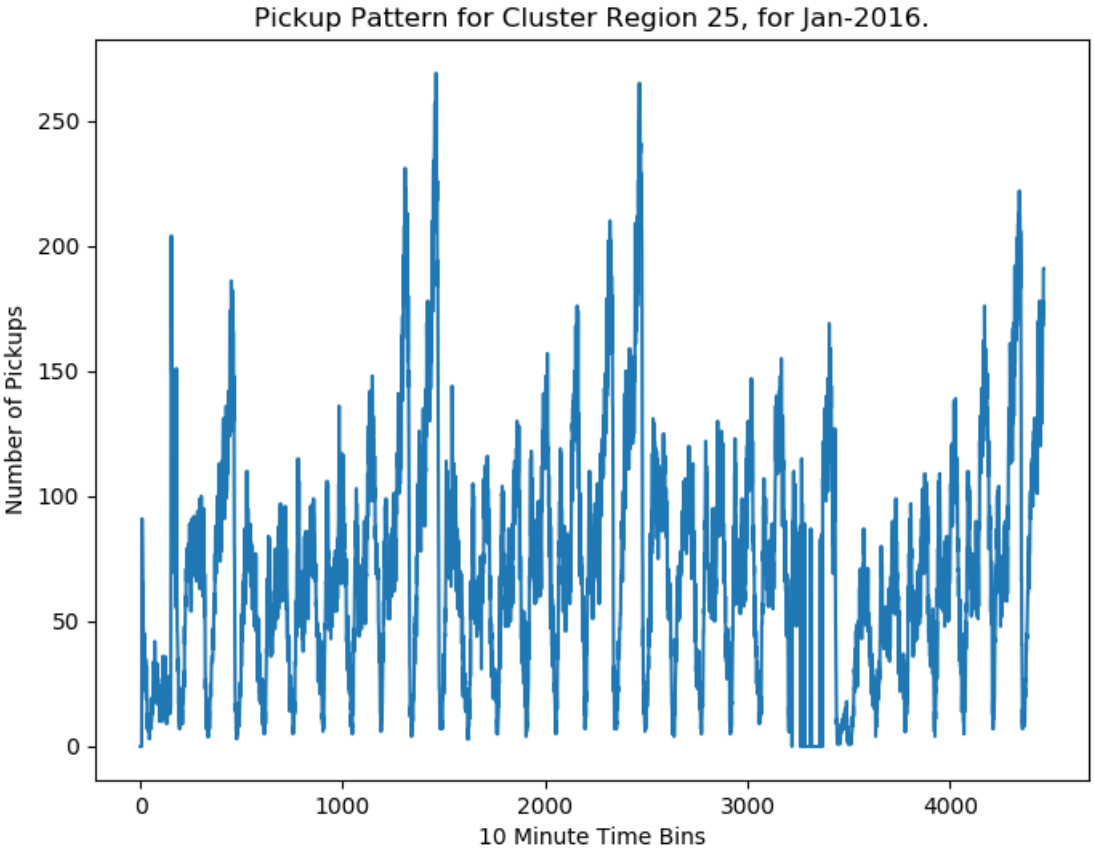

Pickup Pattern for Cluster Region 22, for Jan-2016.
Number of Pickups
200
175
150
125
100
75
50

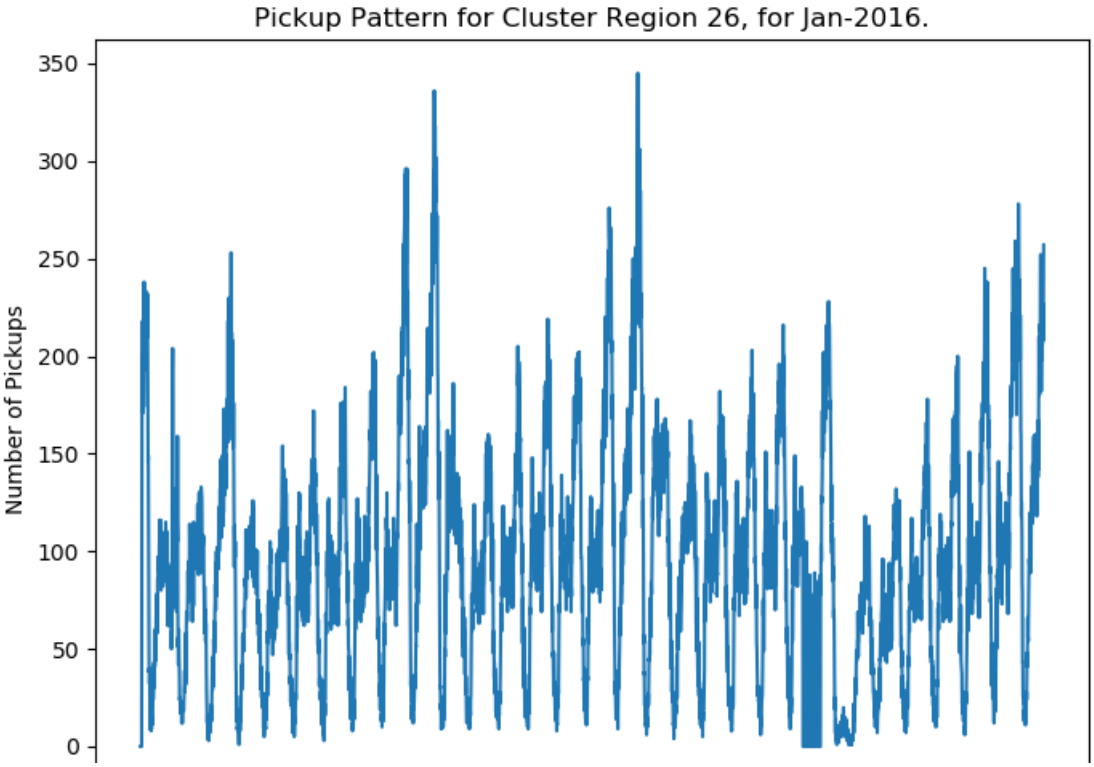

25
0
0
1000
2000
3000
4000
10 Minute Time Bins

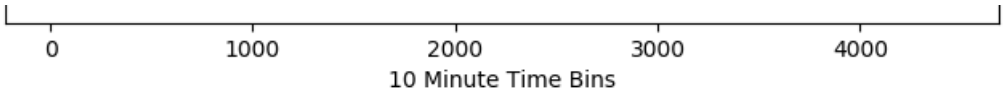

Pickup Pattern for Cluster Region 23, for Jan-2016.
Number of Pickups
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

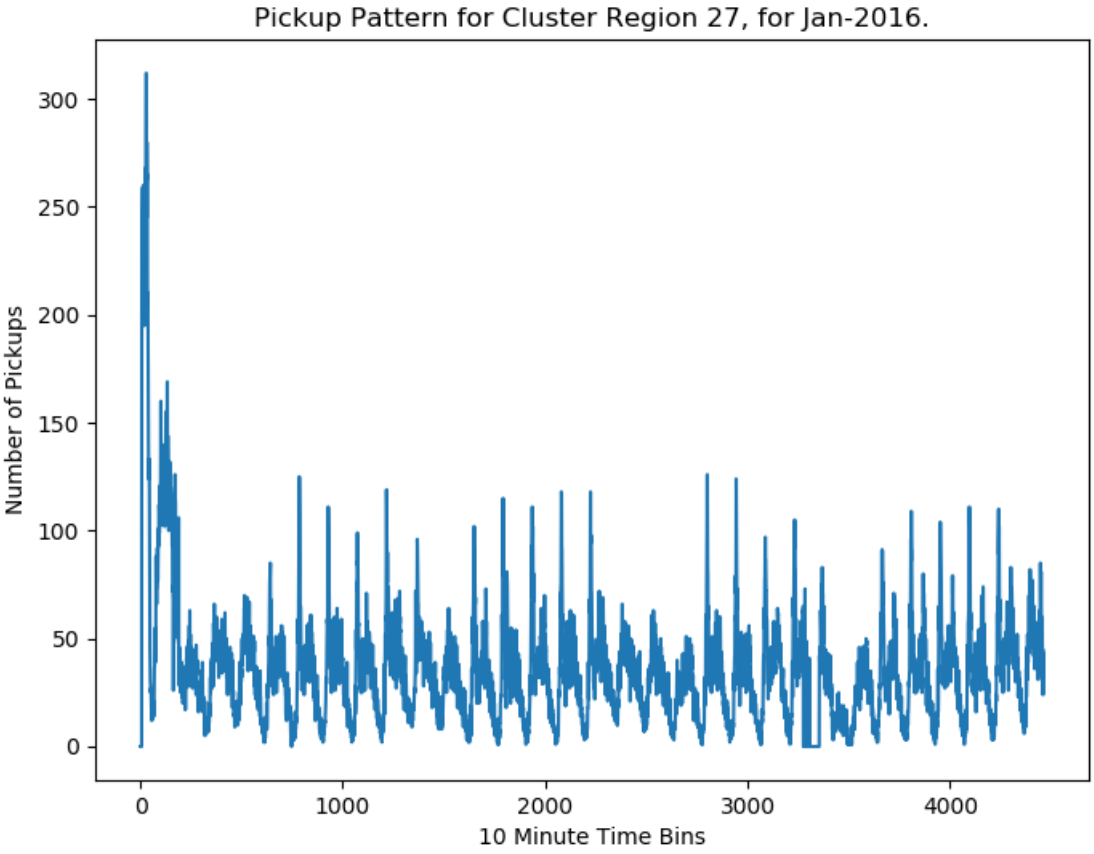

Pickup Pattern for Cluster Region 24, for Jan-2016.
Number of Pickups
200
150
100
50

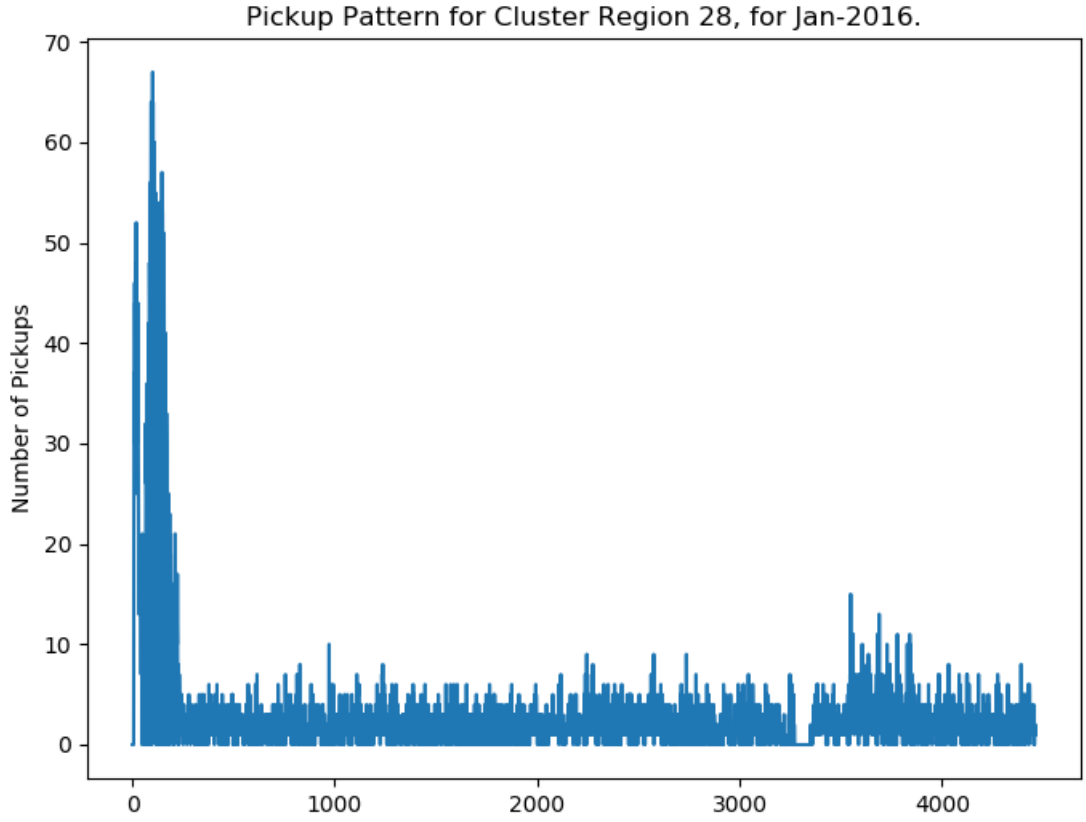

0
0
1000
2000
3000
4000
10 Minute Time Bins


Pickup Pattern for Cluster Region 25, for Jan-2016.
Number of Pickups
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins

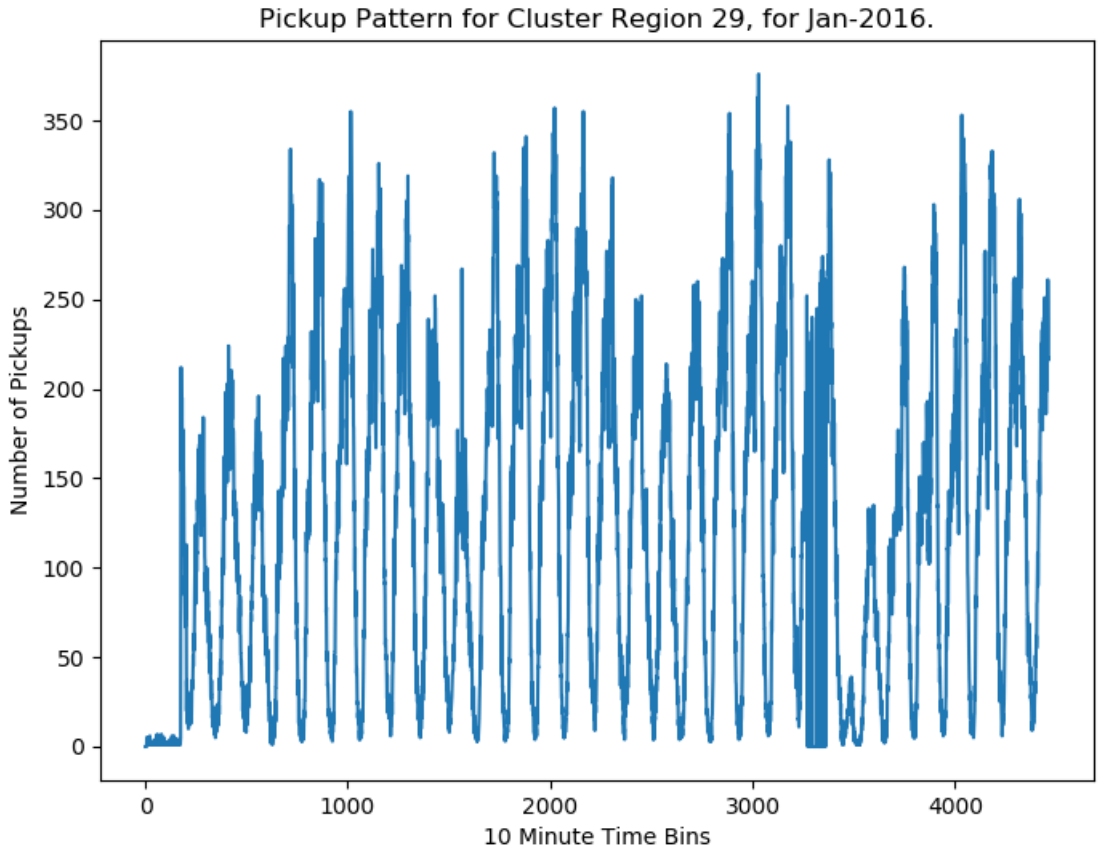

Pickup Pattern for Cluster Region 26, for Jan-2016.
Number of Pickups
350
300
250
200
150
100
50
0

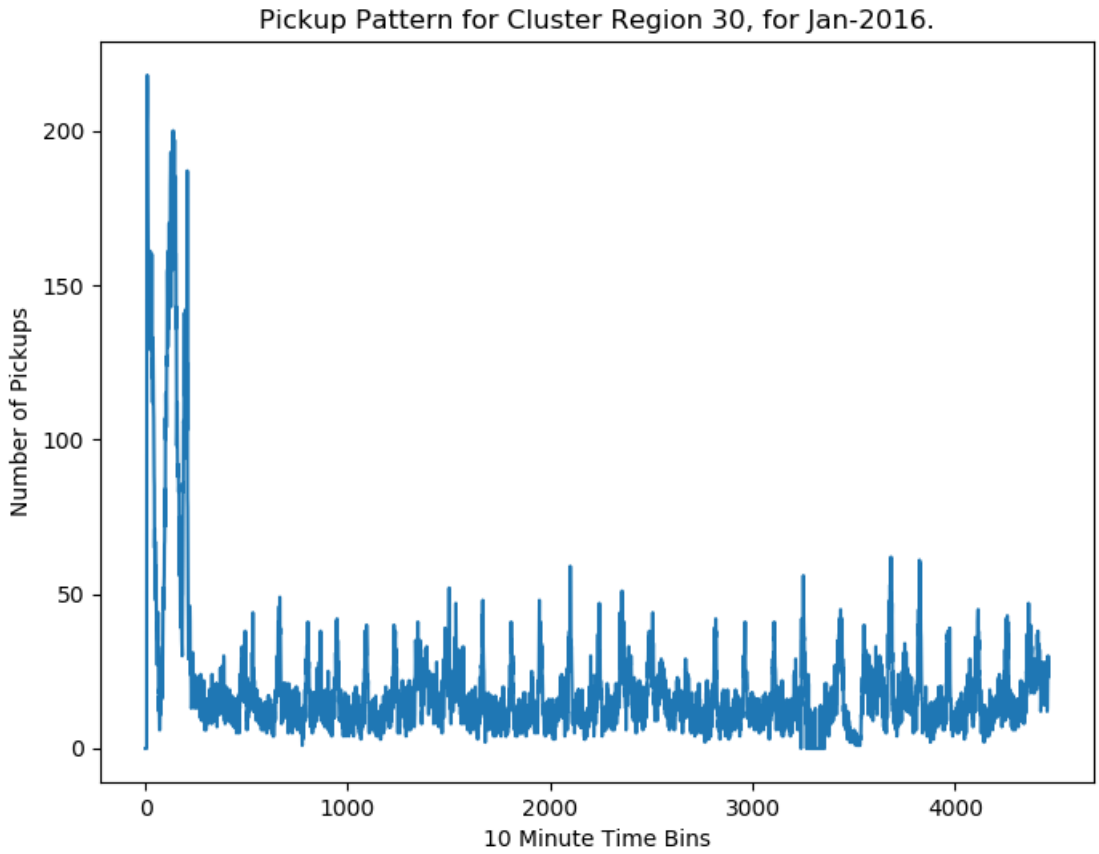

0
1000
2000
3000
4000
10 Minute Time Bins


Pickup Pattern for Cluster Region 27, for Jan-2016.
Number of Pickups
300
250
200
150
100
50
0
0
1000
2000
3000
4000
10 Minute Time Bins


Pickup Pattern for Cluster Region 28, for Jan-2016.
Number of Pickups
70
60
50
40
30
20
10
0
0
1000
2000
3000
4000

10 Minute Time Bins

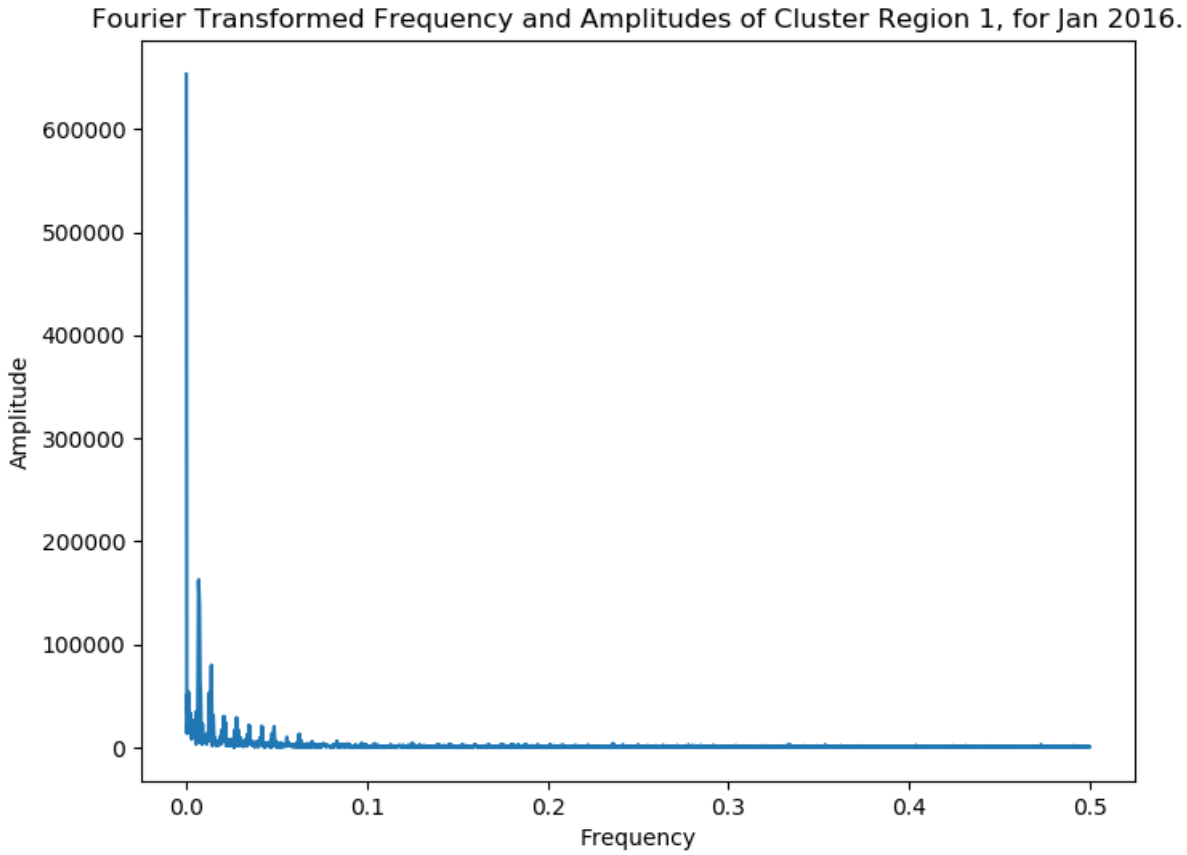


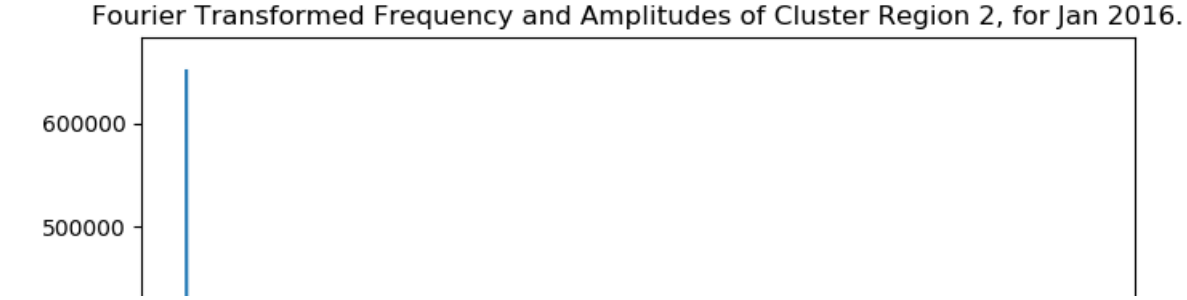


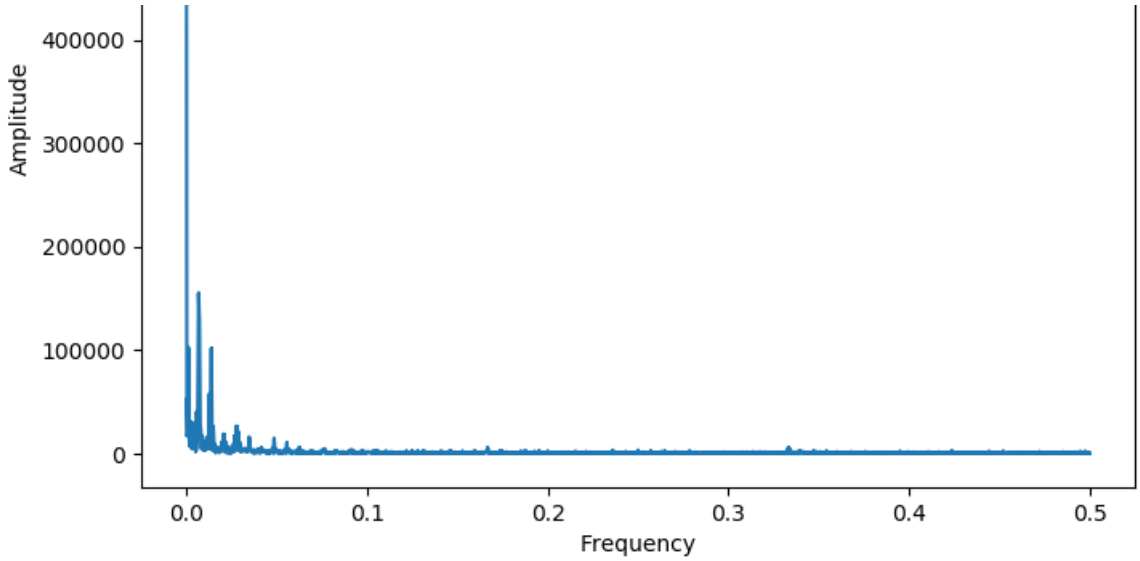
- From the above graphs of pickups of each cluster we can observe that there is a repeating pattern in pickups in 24hrs time period

In [89]:

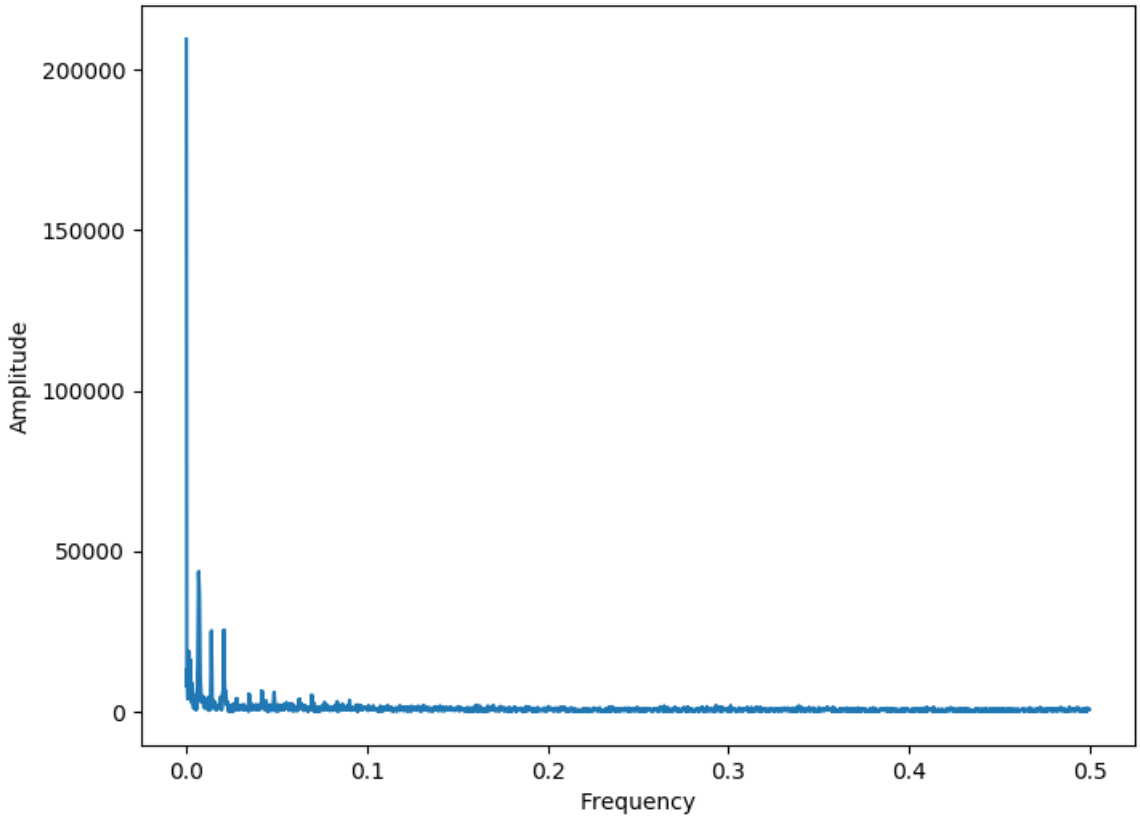
```
# Let's decompose these repeating waves by using fourier transform and
#use their frequencies and their corresponding amplitudes as features in our data
for i in range(30):
    Y  = np.abs(np.fft.fft(regionWisePickup_Jan_2016[i][0:4096]))
    # read more about the fftfreq:
https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html
    freq = np.abs(np.fft.fftfreq(4096, 1))
    n = len(freq)
    plt.figure(figsize = (8, 6))
    plt.plot(freq[:], Y[:])
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    plt.title("Fourier Transformed Frequency and Amplitudes of Cluster Region "+str(i+1)+", for
Jan 2016.")
    plt.show()
```



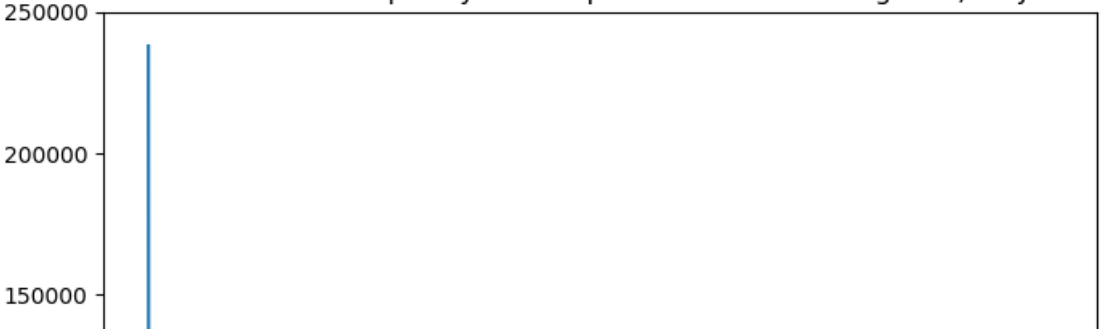


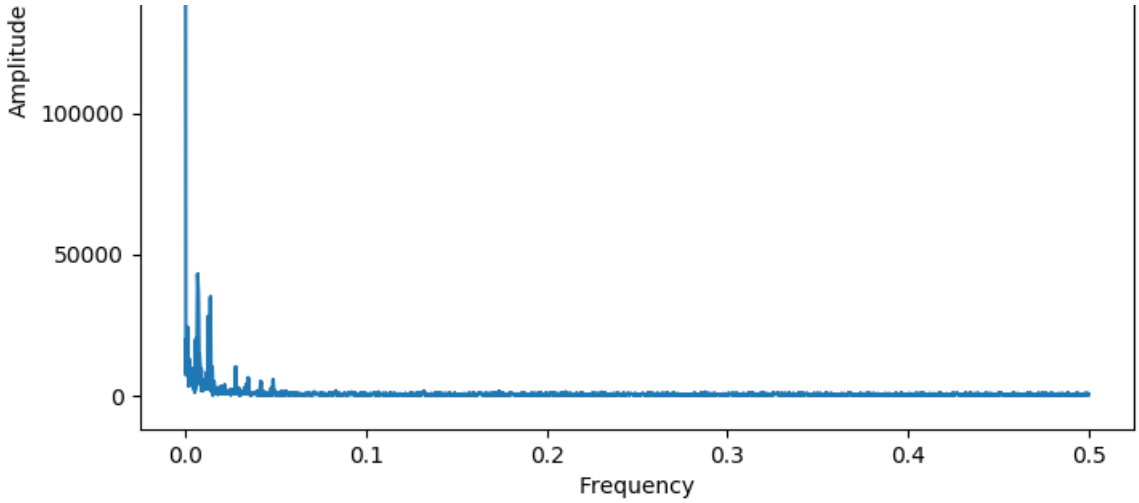


Fourier Transformed Frequency and Amplitudes of Cluster Region 3, for Jan 2016.

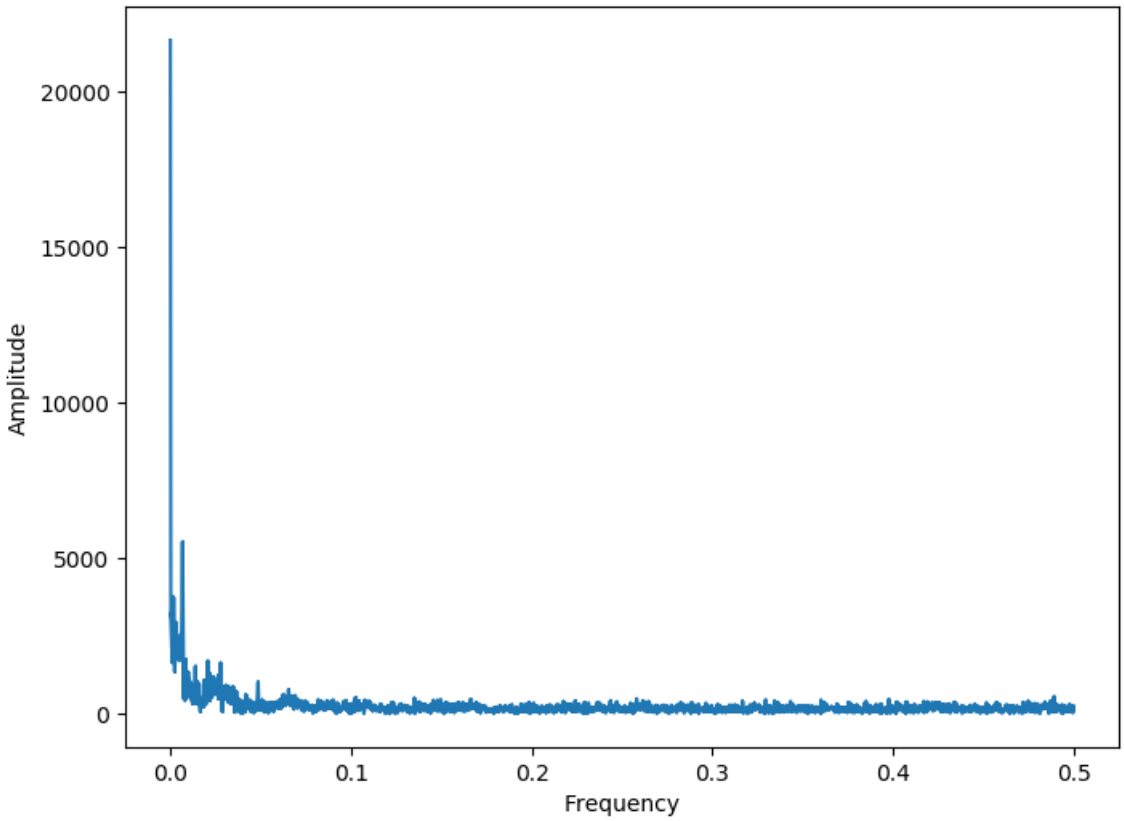


Fourier Transformed Frequency and Amplitudes of Cluster Region 4, for Jan 2016.

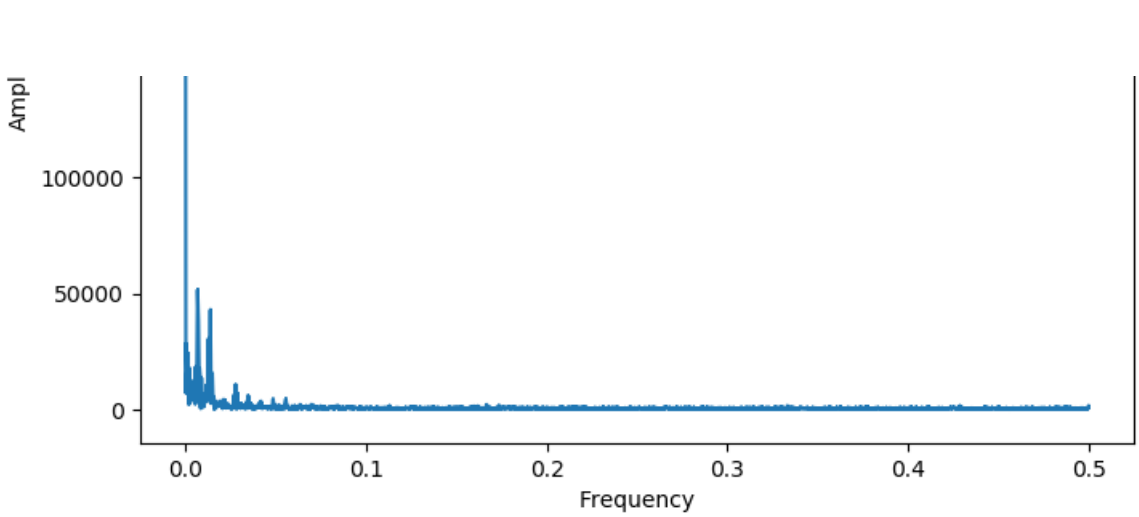




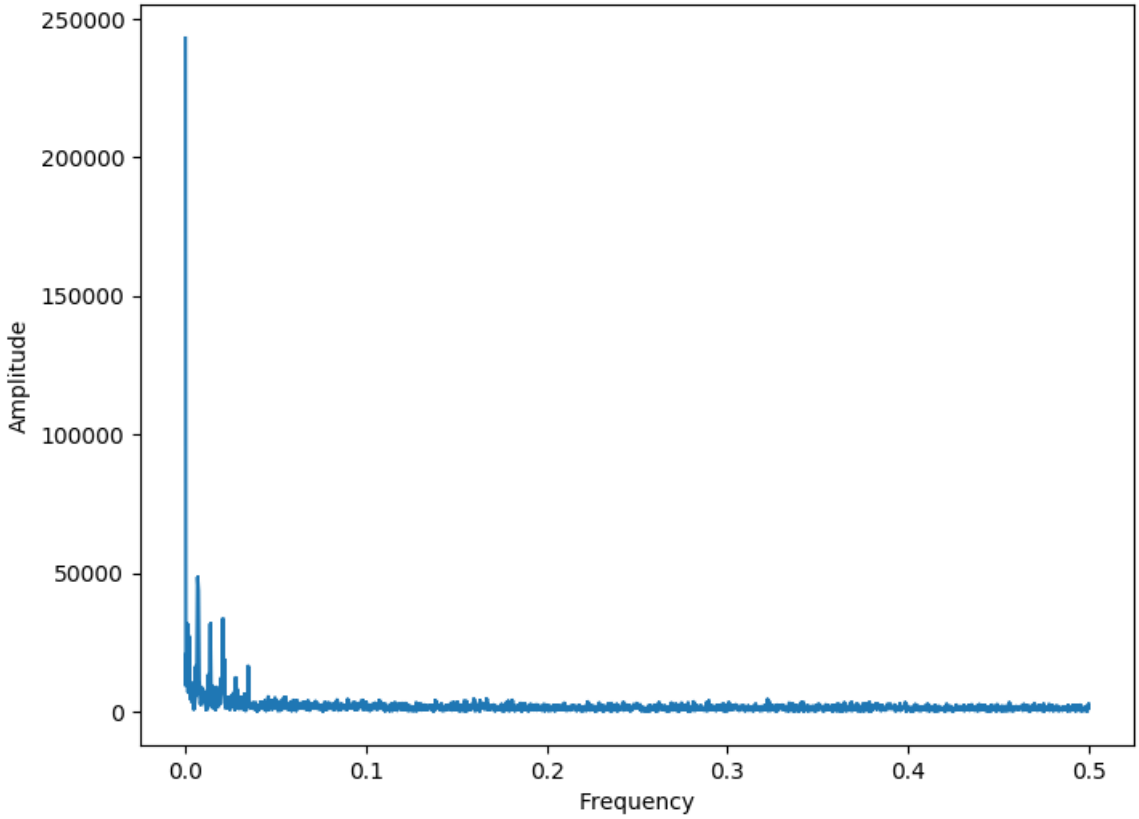
Fourier Transformed Frequency and Amplitudes of Cluster Region 5, for Jan 2016.



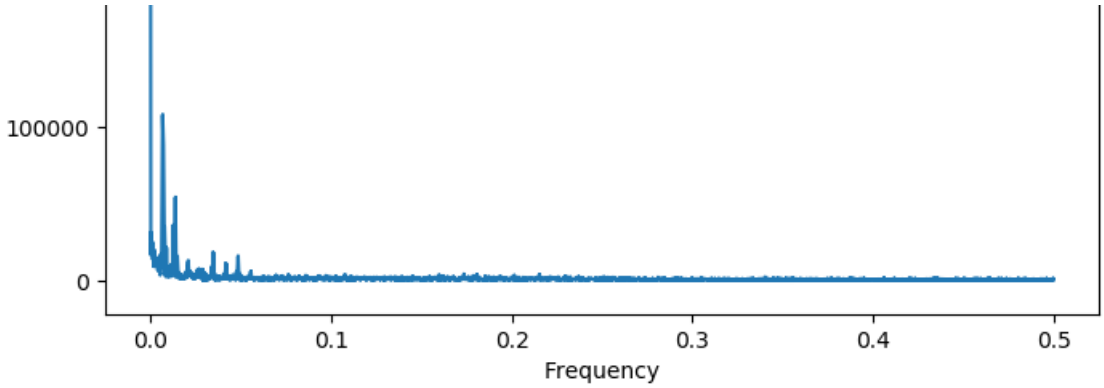
Fourier Transformed Frequency and Amplitudes of Cluster Region 6, for Jan 2016.



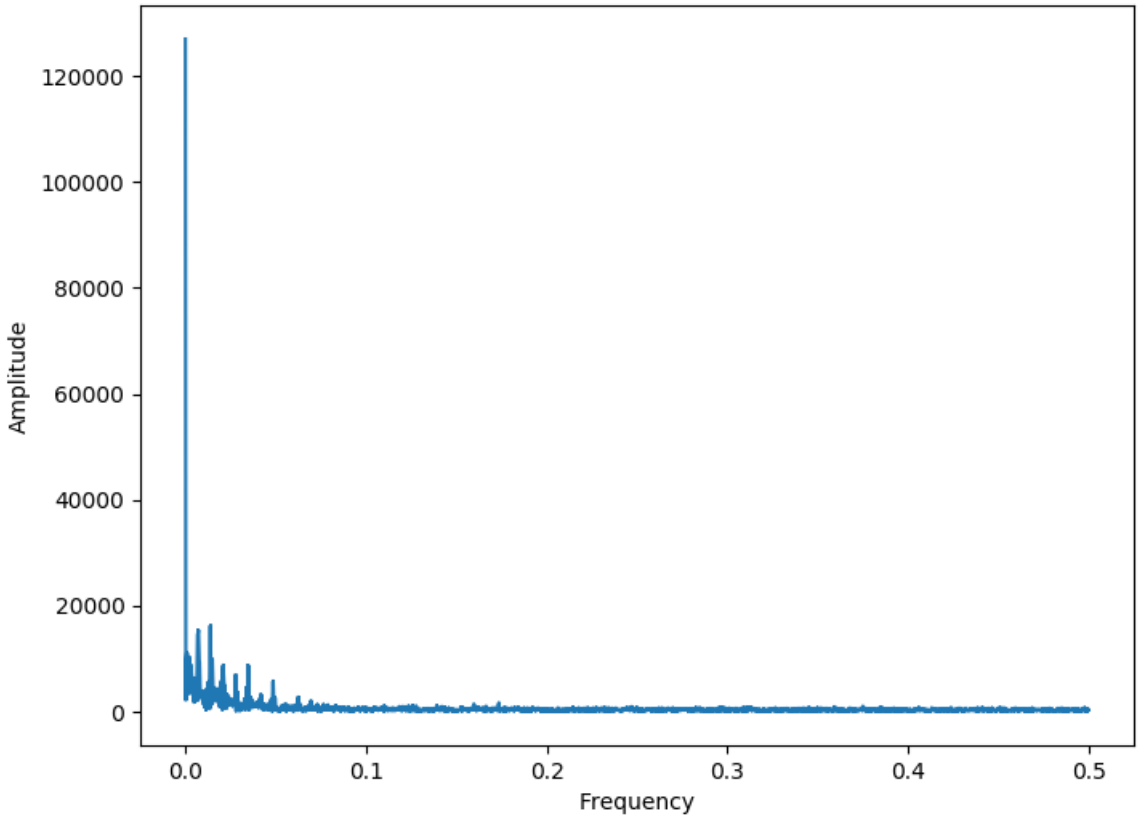
Fourier Transformed Frequency and Amplitudes of Cluster Region 7, for Jan 2016.



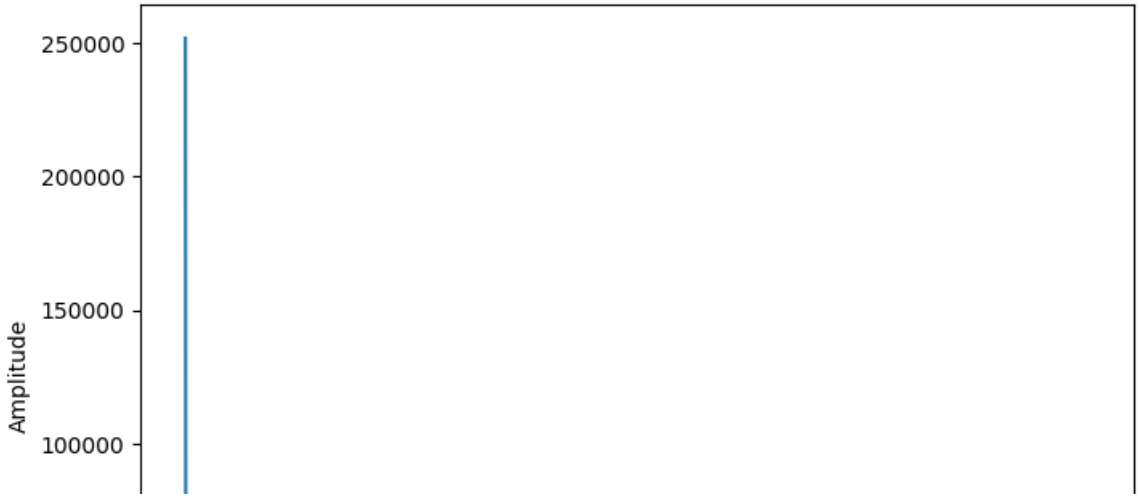
Fourier Transformed Frequency and Amplitudes of Cluster Region 8, for Jan 2016.

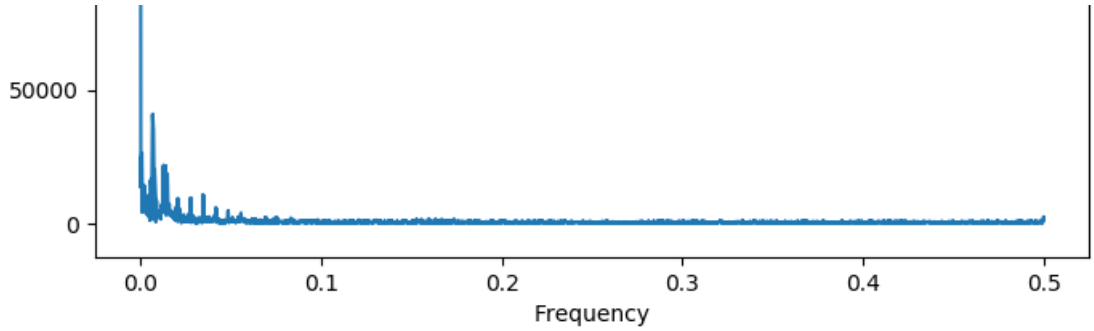


Fourier Transformed Frequency and Amplitudes of Cluster Region 9, for Jan 2016.

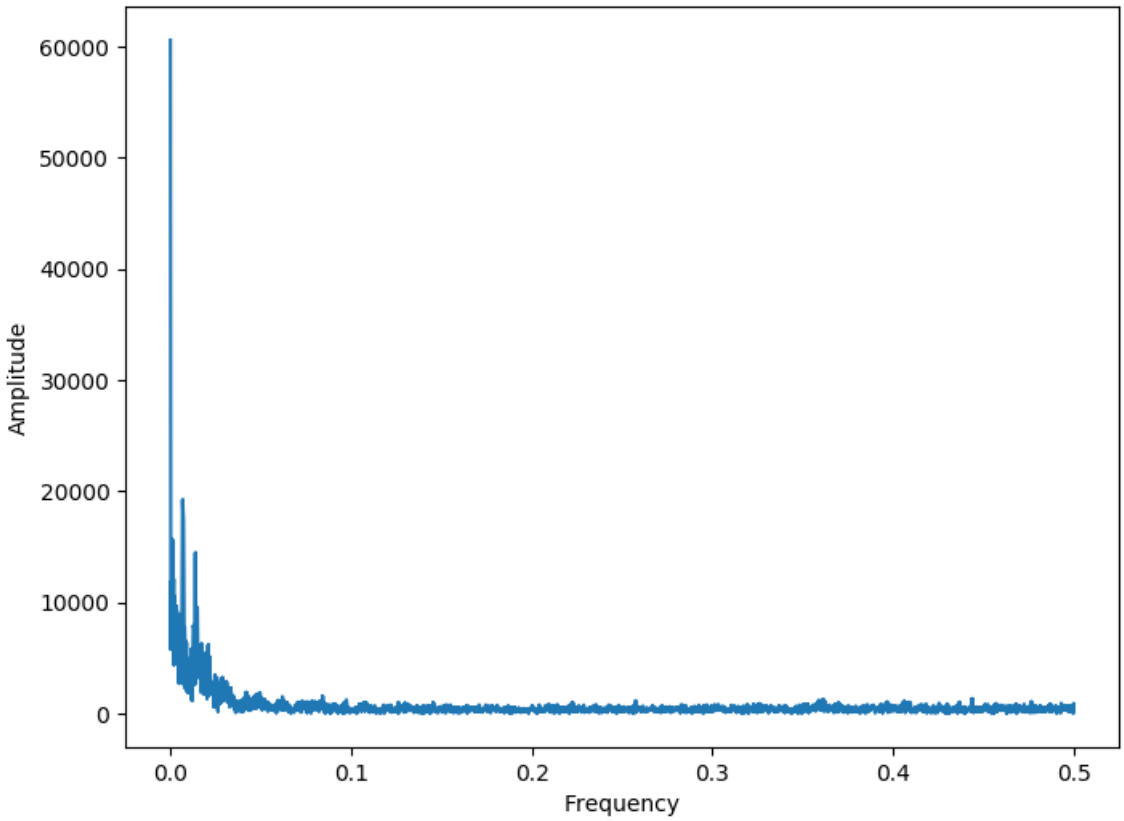


Fourier Transformed Frequency and Amplitudes of Cluster Region 10, for Jan 2016.

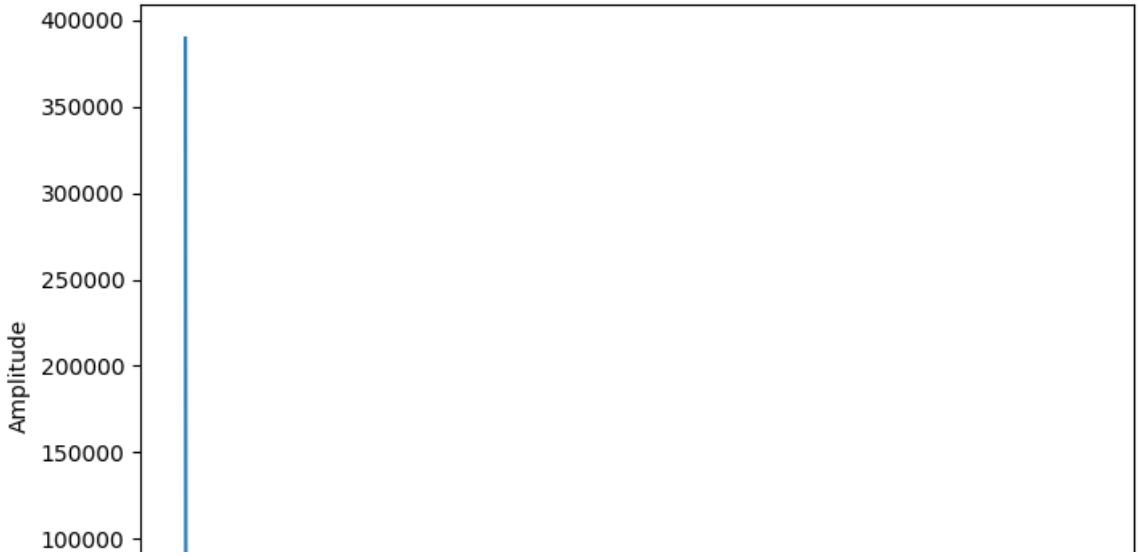


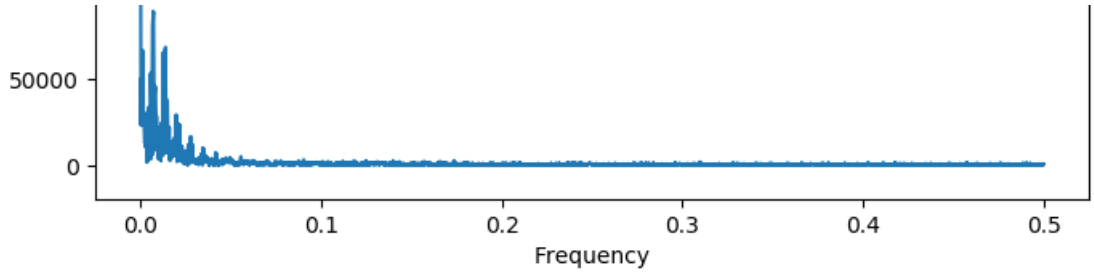


Fourier Transformed Frequency and Amplitudes of Cluster Region 11, for Jan 2016.

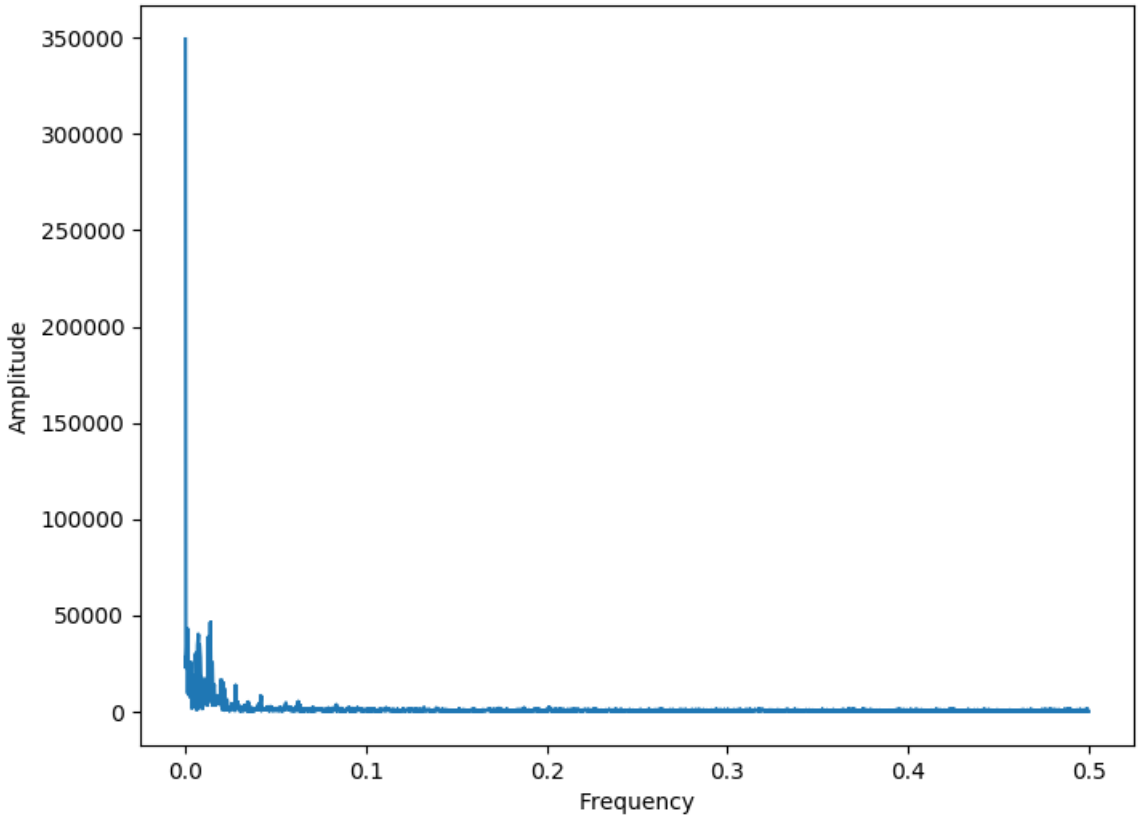


Fourier Transformed Frequency and Amplitudes of Cluster Region 12, for Jan 2016.

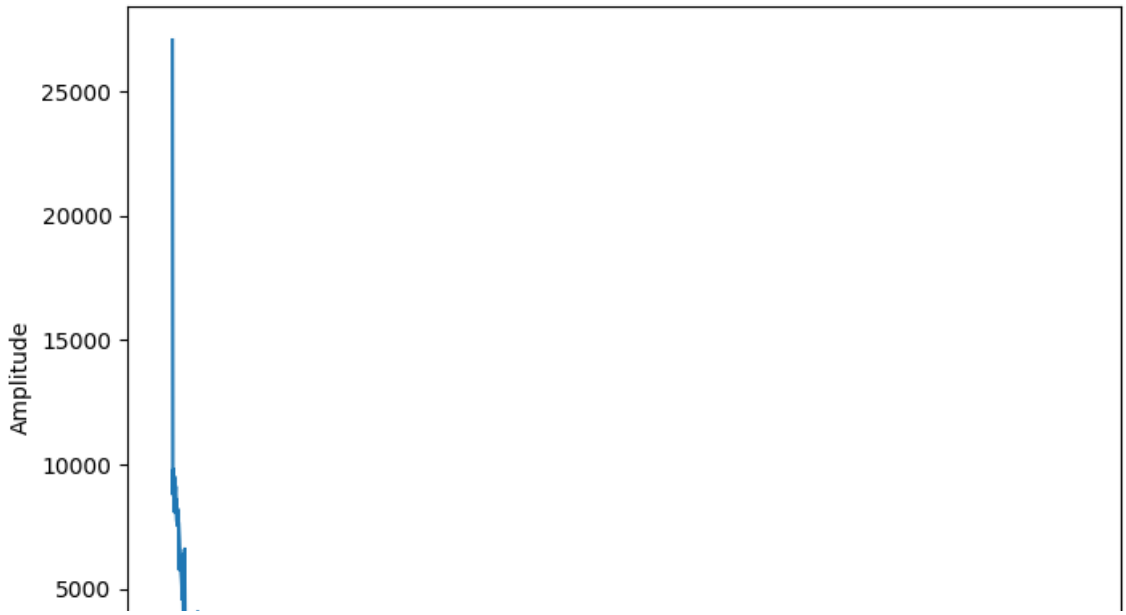


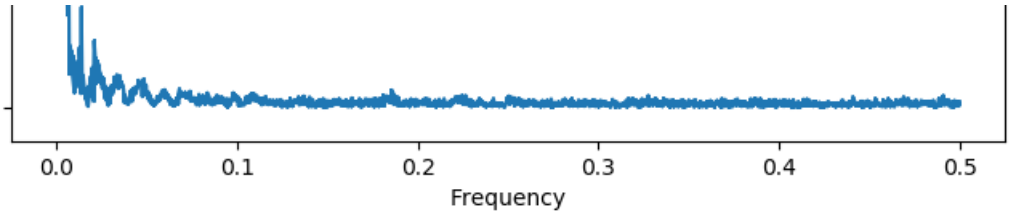


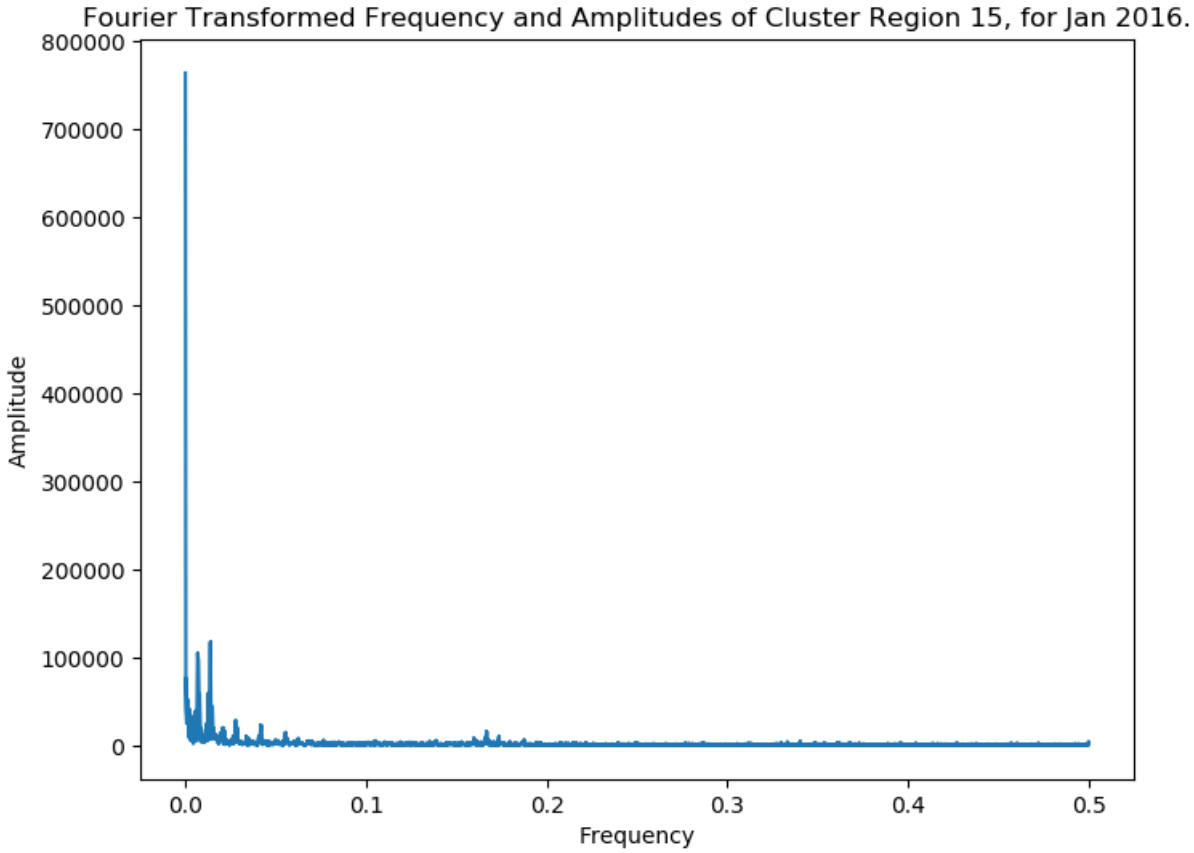
Fourier Transformed Frequency and Amplitudes of Cluster Region 13, for Jan 2016.

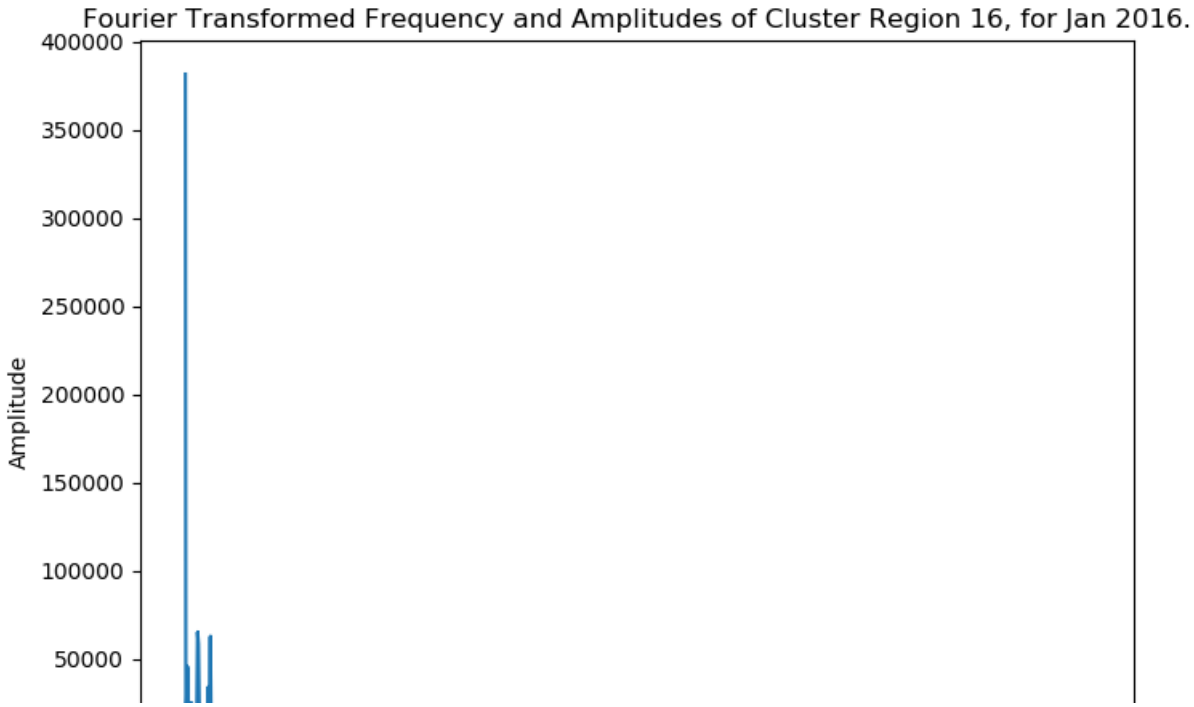


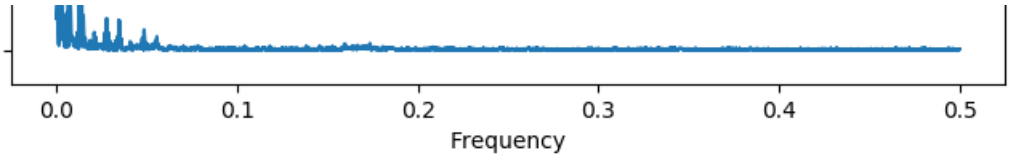
Fourier Transformed Frequency and Amplitudes of Cluster Region 14, for Jan 2016.

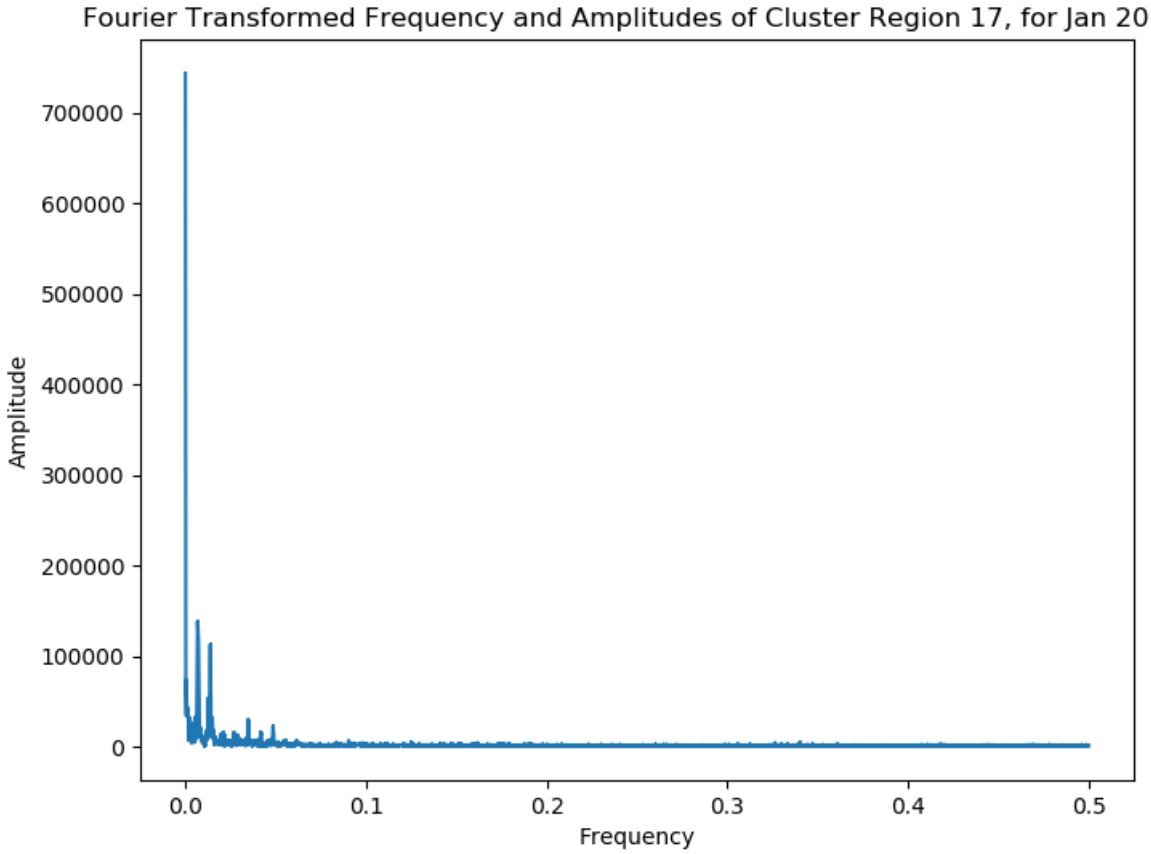


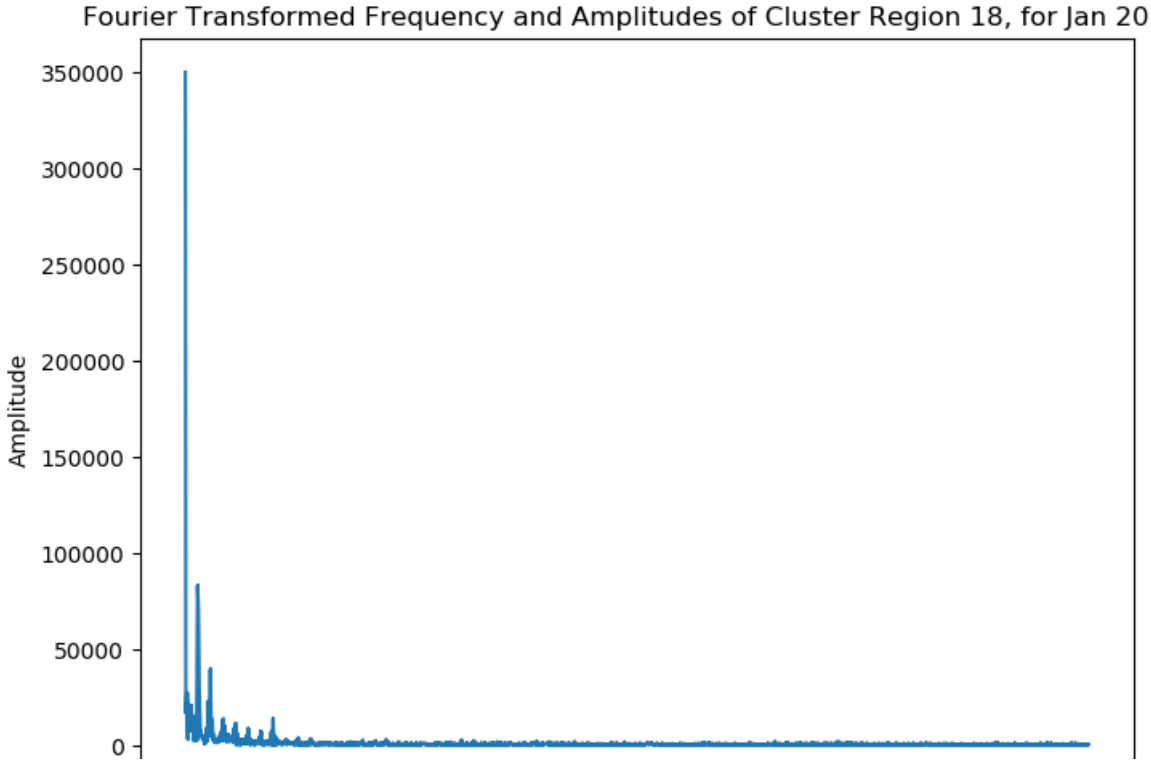

0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

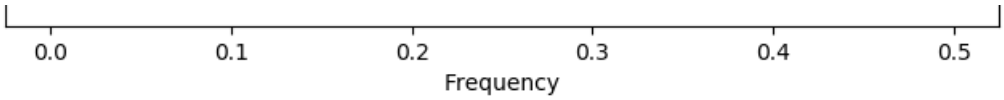

Fourier Transformed Frequency and Amplitudes of Cluster Region 15, for Jan 2016.
800000
700000
600000
500000
400000
300000
200000
100000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

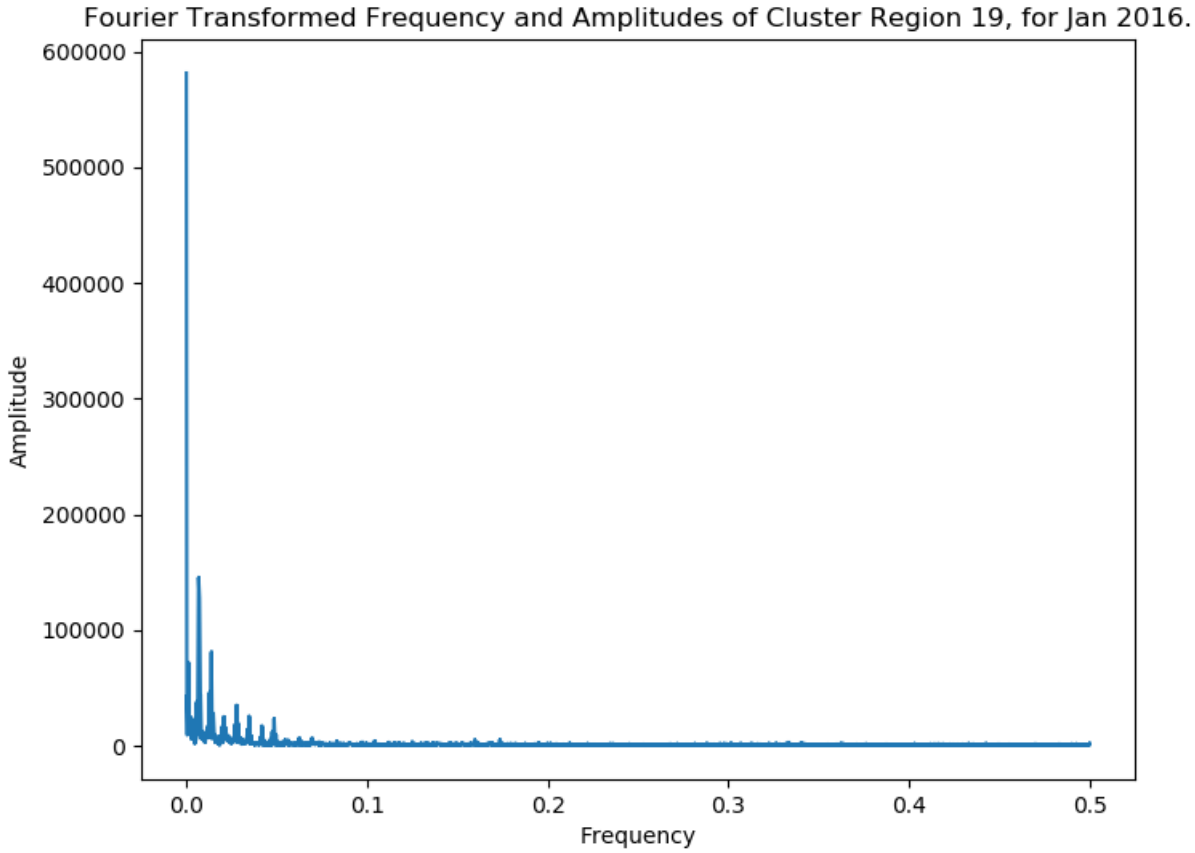

Fourier Transformed Frequency and Amplitudes of Cluster Region 16, for Jan 2016.
400000
350000
300000
250000
200000
150000
100000
50000
Amplitude

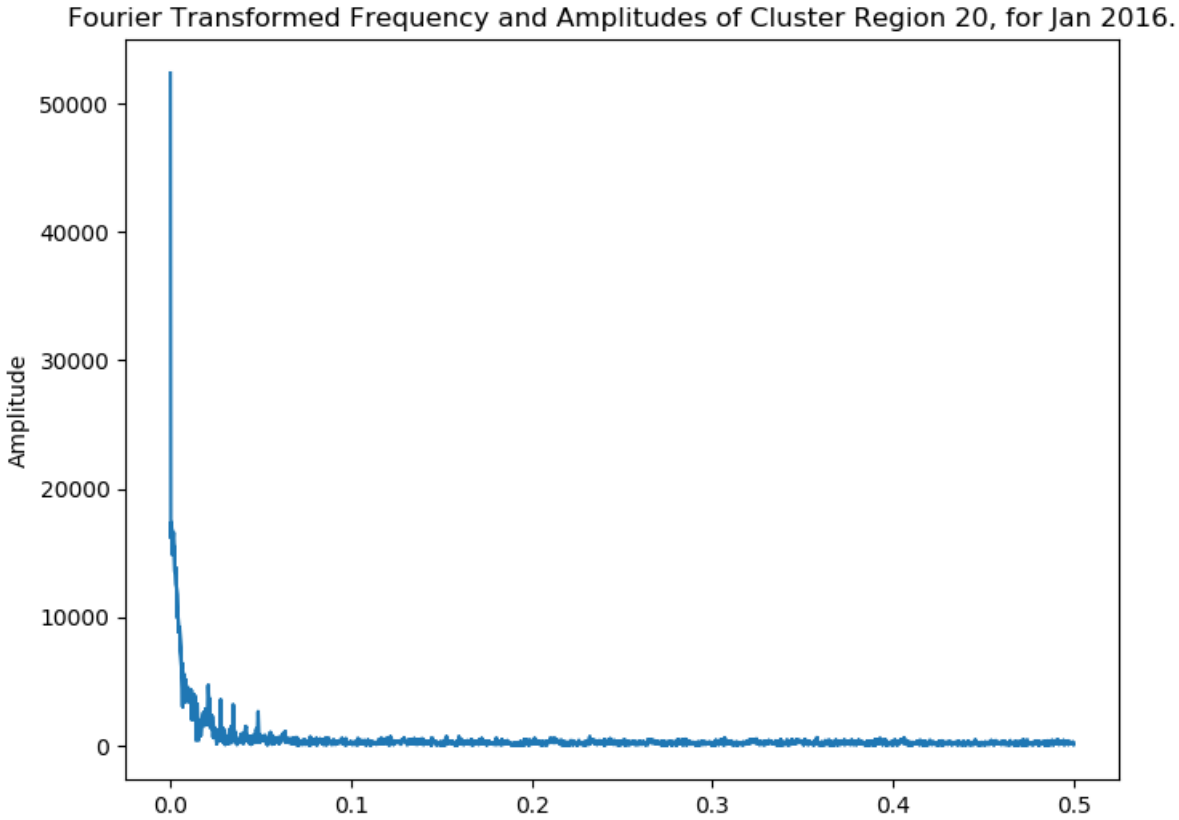

0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Fourier Transformed Frequency and Amplitudes of Cluster Region 17, for Jan 2016.
700000
600000
500000
400000
300000
200000
100000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

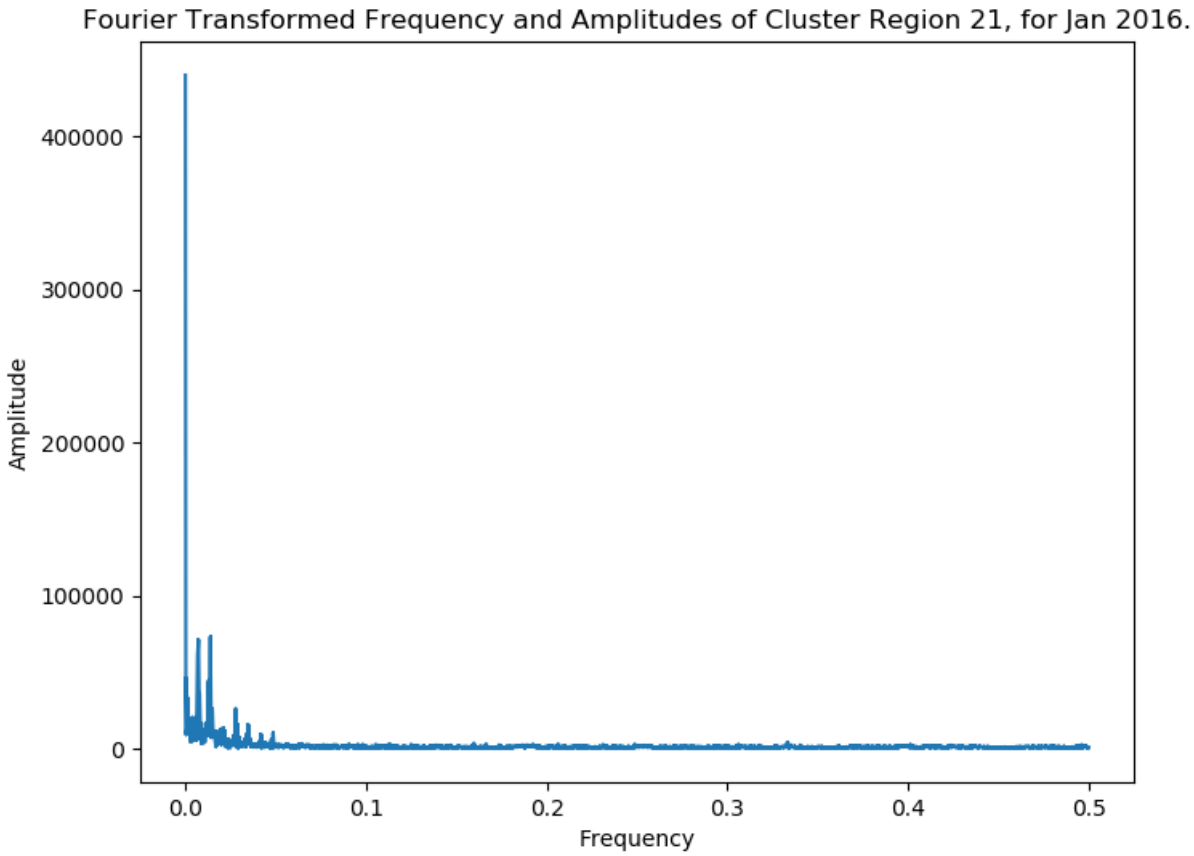

Fourier Transformed Frequency and Amplitudes of Cluster Region 18, for Jan 2016.
350000
300000
250000
200000
150000
100000
50000
0
Amplitude

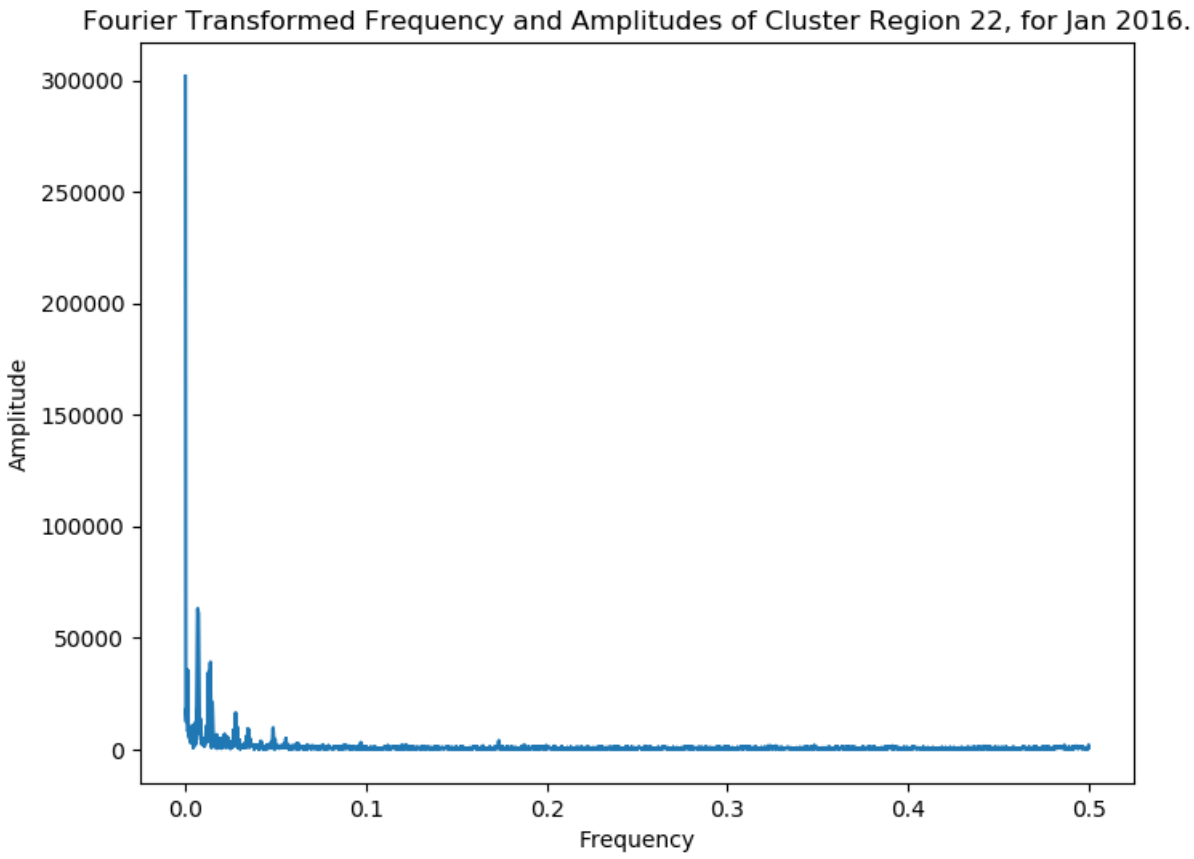

0.0
0.1
0.2
0.3
0.4
0.5
Frequency

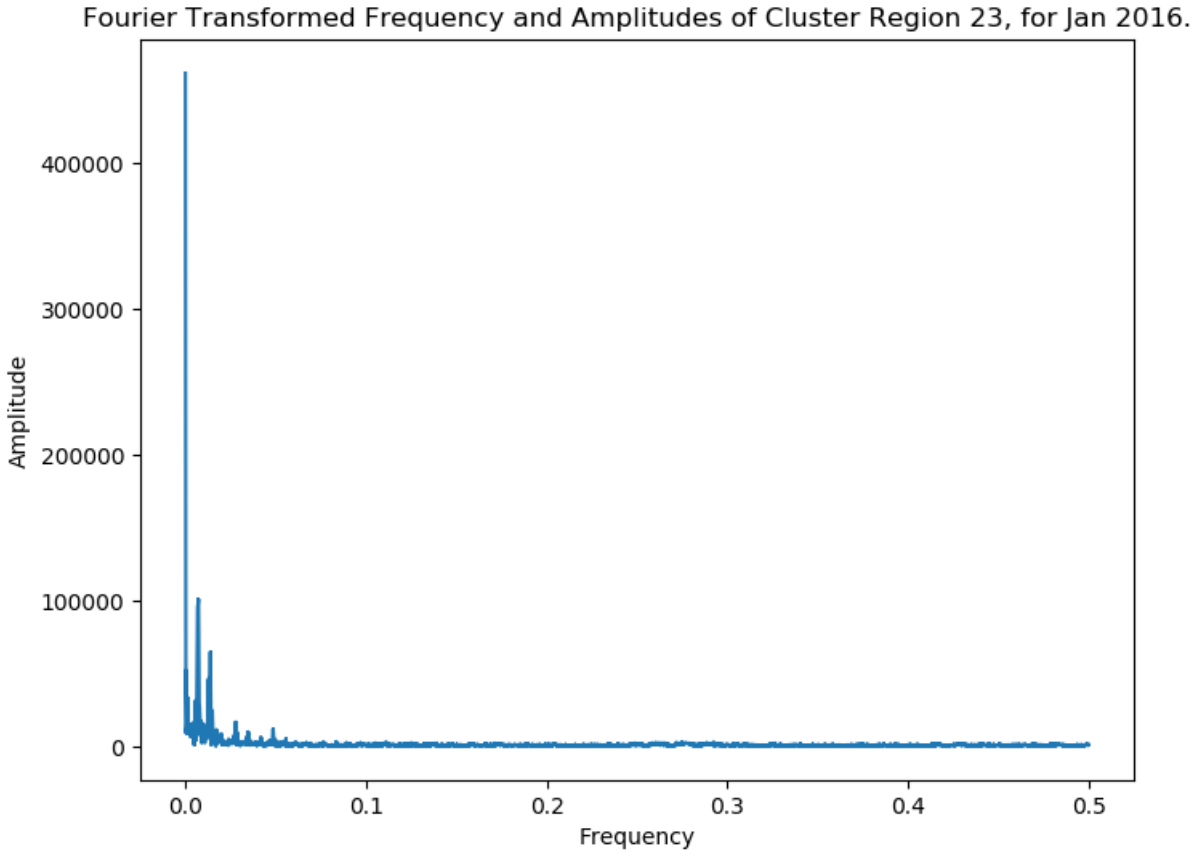

Fourier Transformed Frequency and Amplitudes of Cluster Region 19, for Jan 2016.
600000
500000
400000
300000
200000
100000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

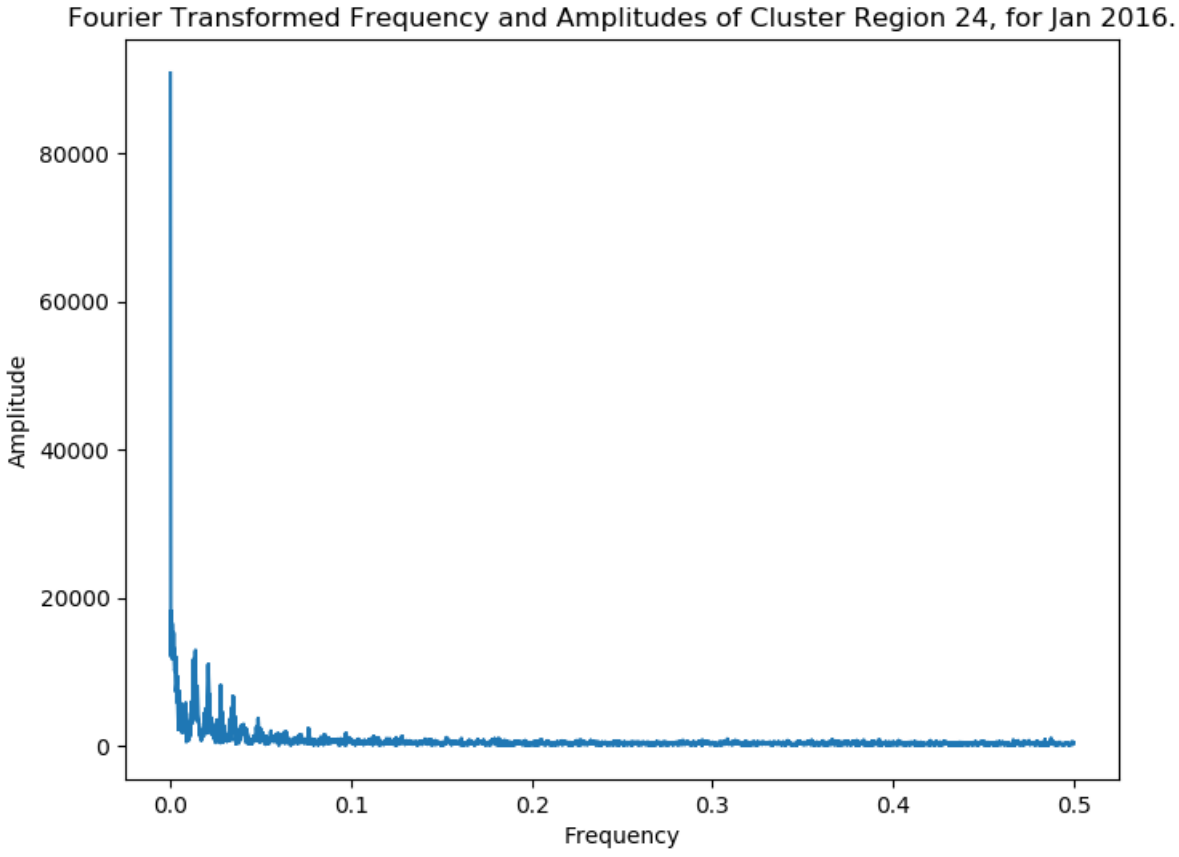

Fourier Transformed Frequency and Amplitudes of Cluster Region 20, for Jan 2016.
50000
40000
30000
20000
10000
0
Amplitude
0.0
0.1
0.2
0.3
0.4
0.5

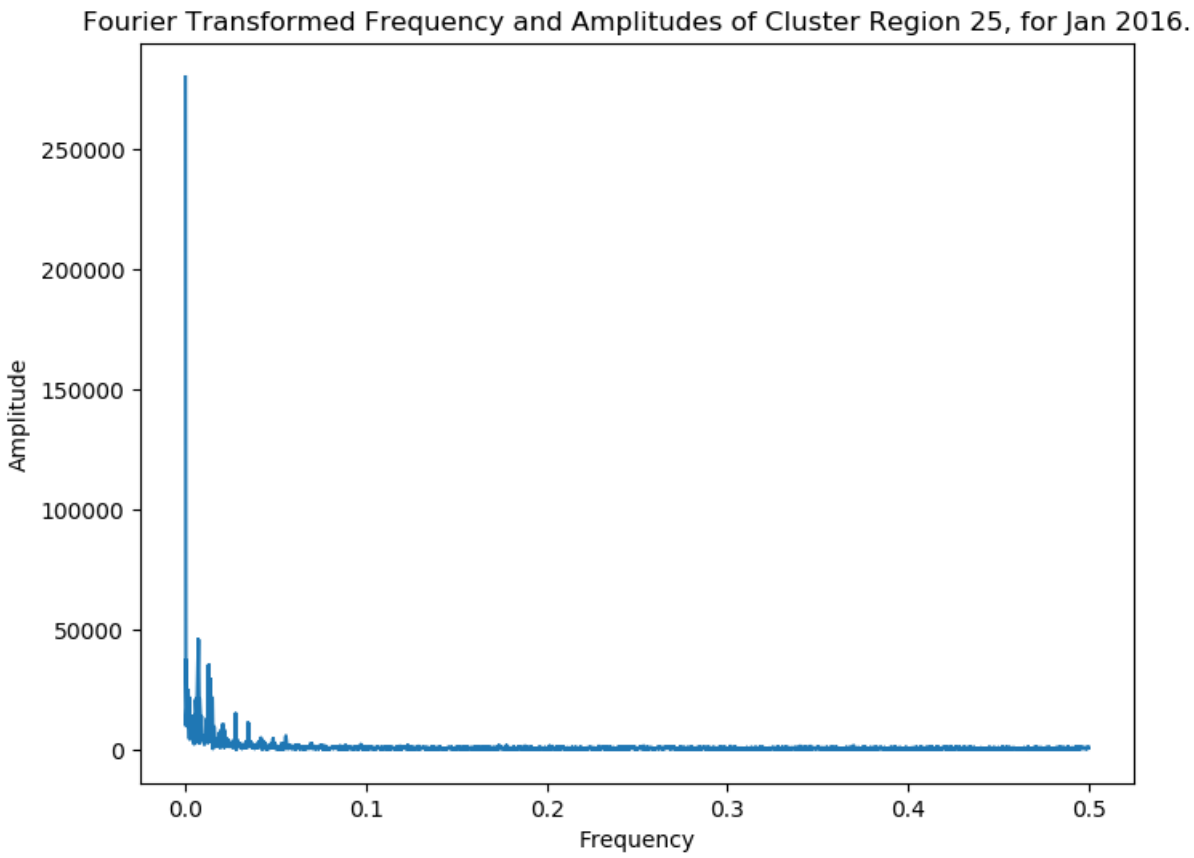
Frequency

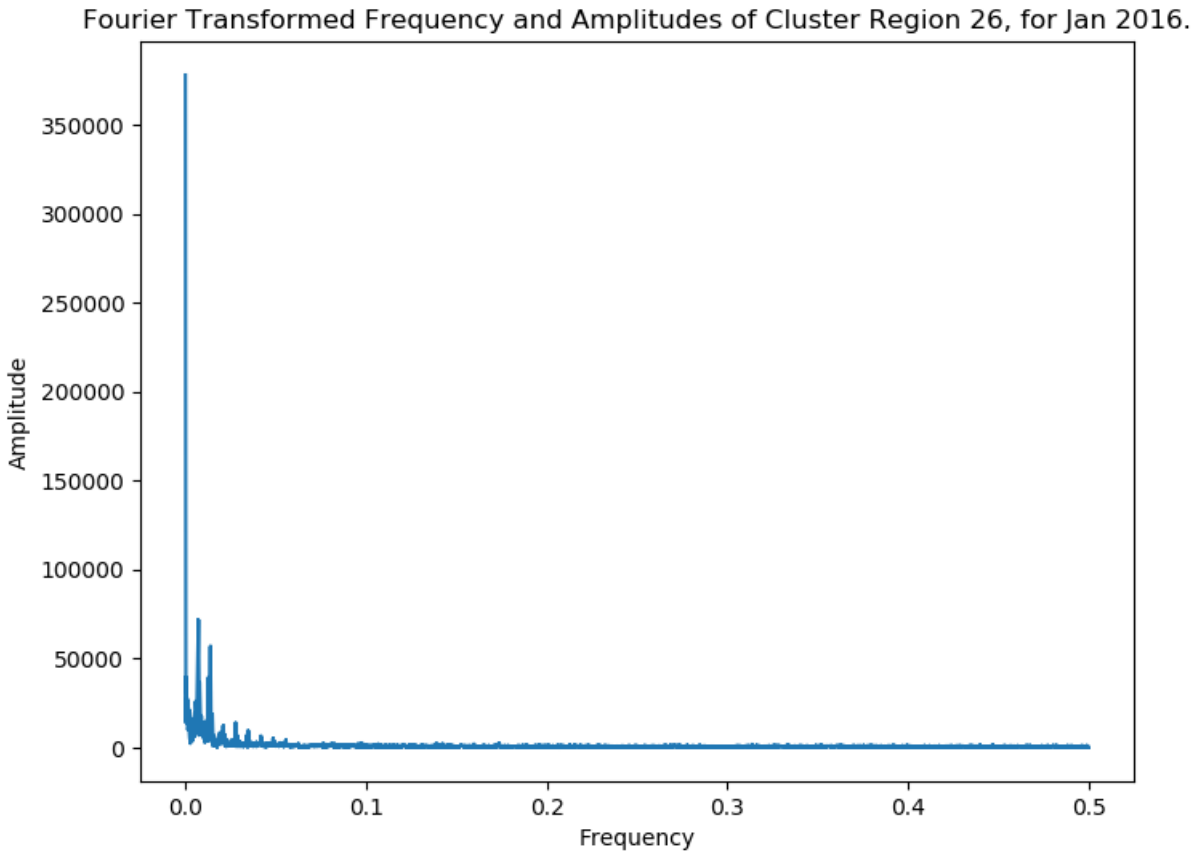

Fourier Transformed Frequency and Amplitudes of Cluster Region 21, for Jan 2016.
Amplitude
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Fourier Transformed Frequency and Amplitudes of Cluster Region 22, for Jan 2016.
Amplitude
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

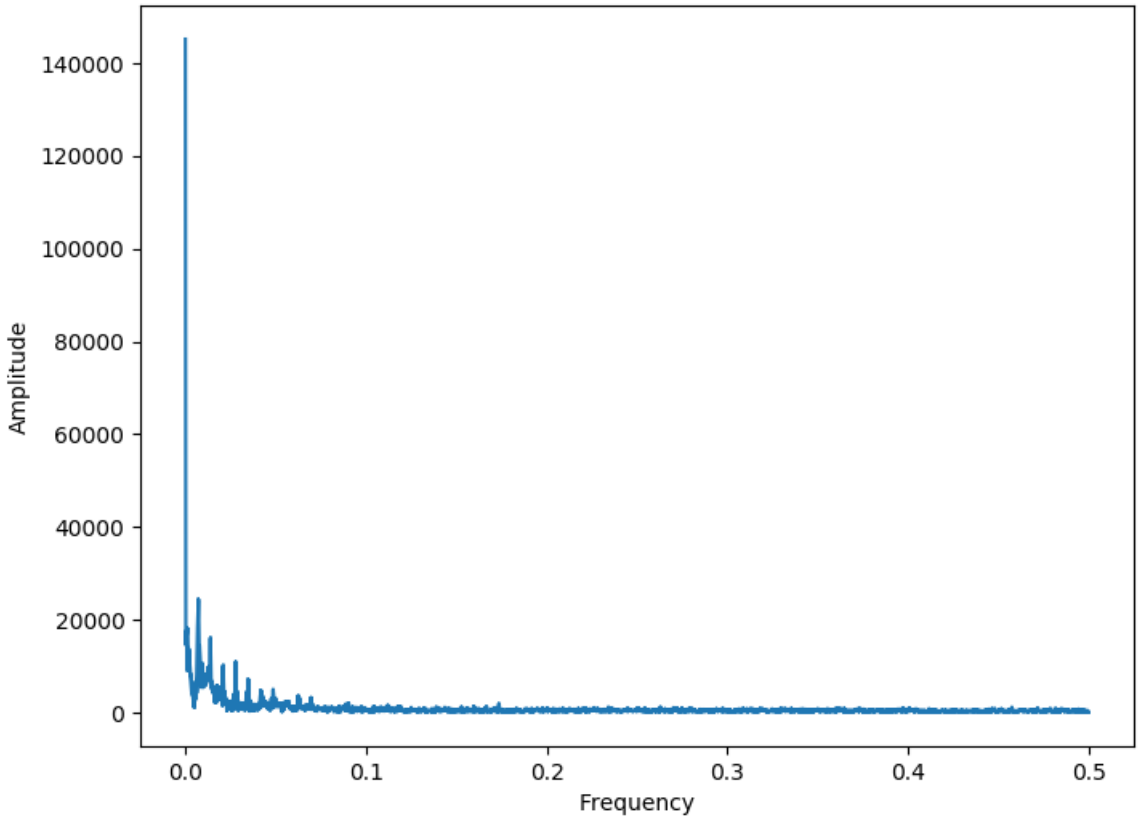

Fourier Transformed Frequency and Amplitudes of Cluster Region 23, for Jan 2016.
Amplitude
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Fourier Transformed Frequency and Amplitudes of Cluster Region 24, for Jan 2016.
Amplitude
80000
60000
40000
20000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

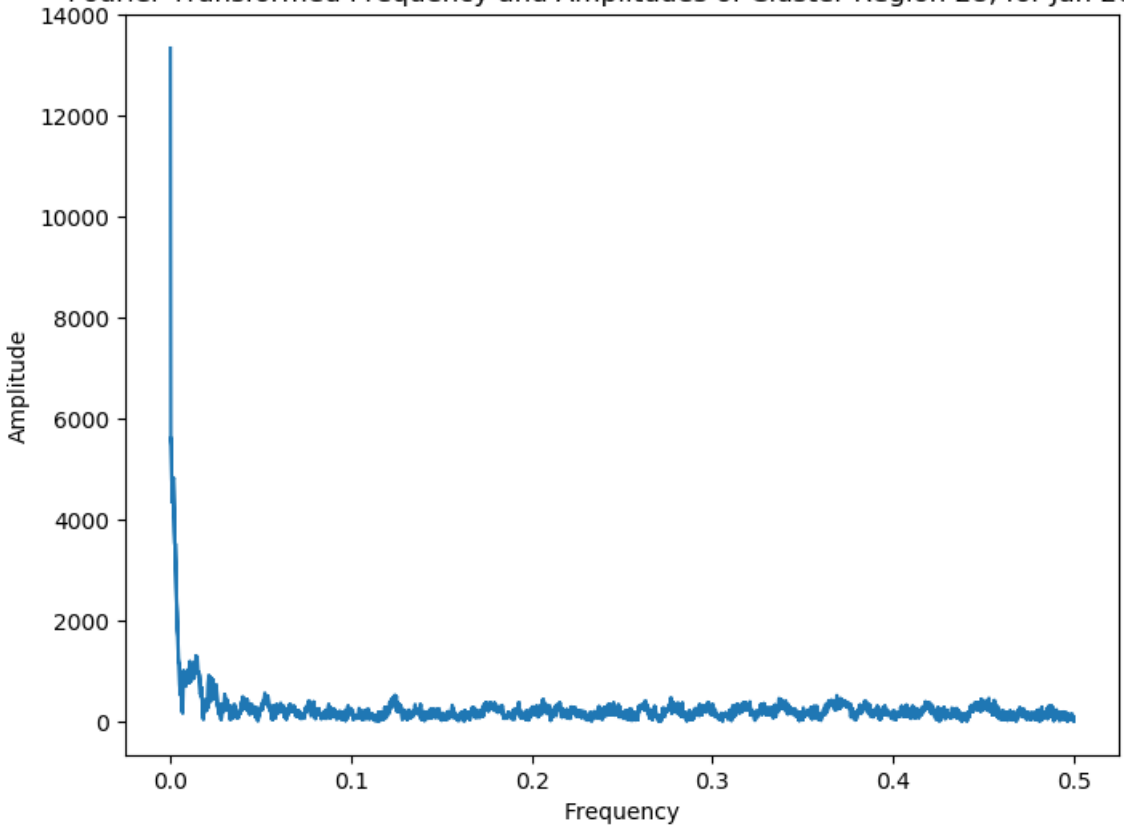

Fourier Transformed Frequency and Amplitudes of Cluster Region 25, for Jan 2016.
Amplitude
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency


Fourier Transformed Frequency and Amplitudes of Cluster Region 26, for Jan 2016.
Amplitude
350000
300000
250000
200000
150000
100000
50000
0
0.0
0.1
0.2
0.3
0.4
0.5
Frequency

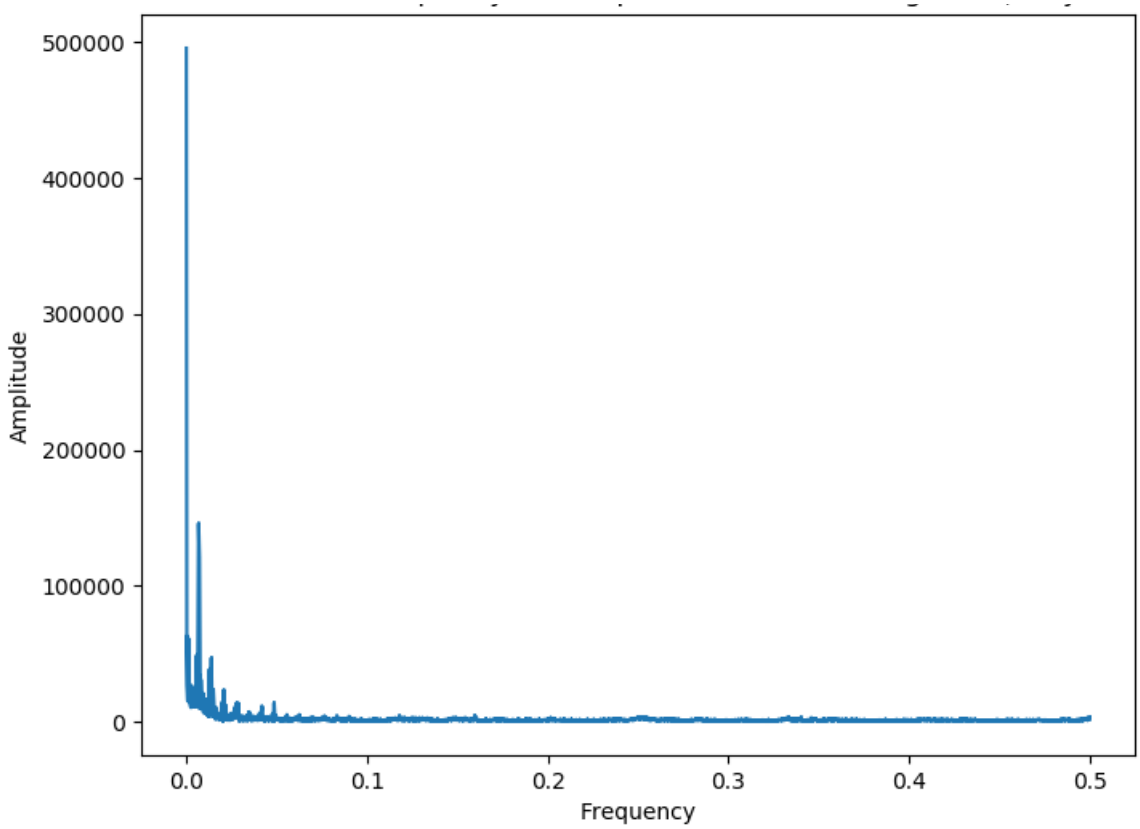
Fourier Transformed Frequency and Amplitudes of Cluster Region 27, for Jan 2016.

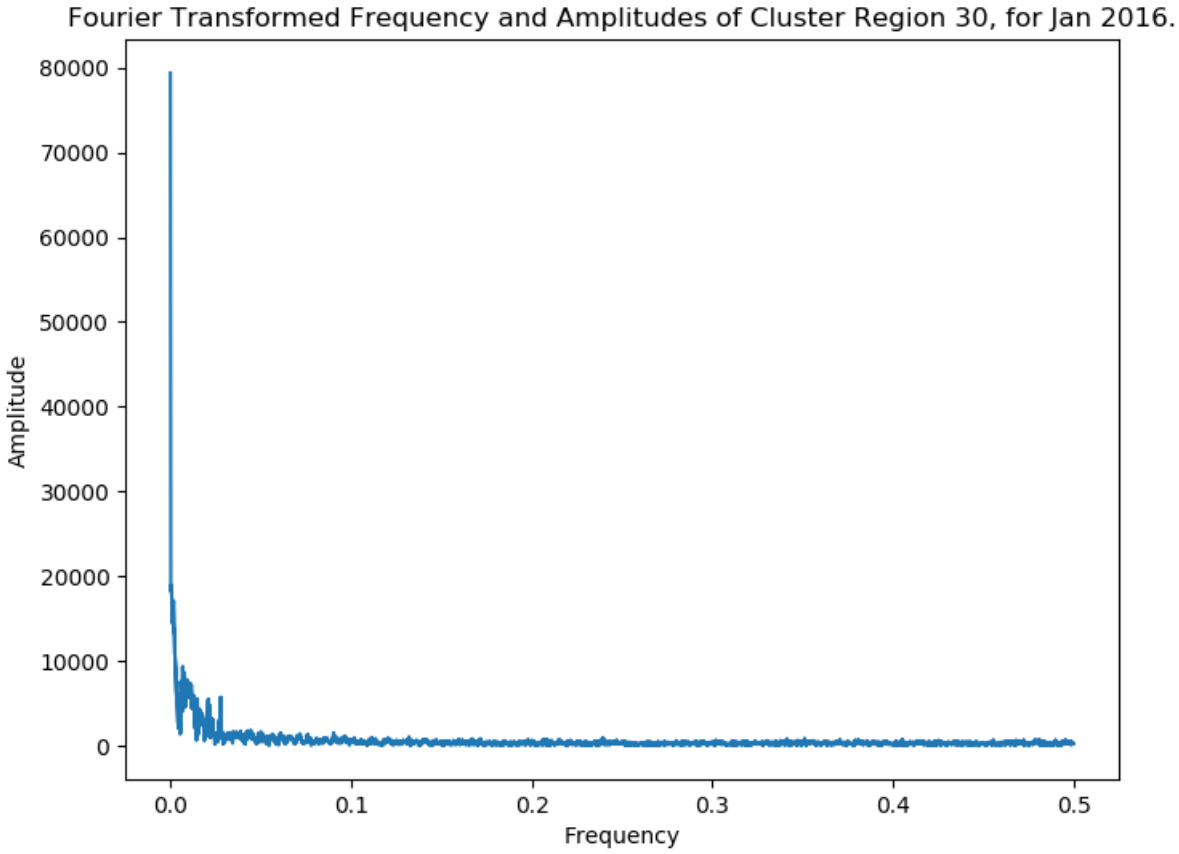


Fourier Transformed Frequency and Amplitudes of Cluster Region 28, for Jan 2016.



Fourier Transformed Frequency and Amplitudes of Cluster Region 29, for Jan 2016.





In [90]:

```
amplitude_lists = []
frequency_lists = []
for i in range(30):
    ampli   = np.abs(np.fft.fft(regionWisePickup_Jan_2016[i][0:4096]))
```

```
    freq = np.abs(np.fft.fftfreq(4096, 1))
    ampli_indices = np.argsort(-ampli)[1:]        #it will return an array of indices for which cor
responding amplitude values are sorted in reverse order.
    amplitude_values = []
    frequency_values = []
    for j in range(0, 9, 2):   #taking top five amplitudes and frequencies
        amplitude_values.append(ampli[ampli_indices[j]])
        frequency_values.append(freq[ampli_indices[j]])
    for k in range(4459):    #those top 5 frequencies and amplitudes are same for all the points i
n one cluster
        amplitude_lists.append(amplitude_values)
        frequency_lists.append(frequency_values)
```

Now we have built our all the features. We have finally now following 19 features in our data:

1. **f_t_1:** Number of pickups that are happened previous t-1st 10min interval
2. **f_t_2:** Number of pickups that are happened previous t-2nd 10min interval
3. **f_t_3:** Number of pickups that are happened previous t-3rd 10min interval
4. **f_t_4:** Number of pickups that are happened previous t-4th 10min interval
5. **f_t_5:** Number of pickups that are happened previous t-5th 10min interval
6. **Freq1:** Fourier Frequency corresponding to 1st highest amplitude
7. **Freq2:** Fourier Frequency corresponding to 2nd highest amplitude
8. **Freq3:** Fourier Frequency corresponding to 3rd highest amplitude
9. **Freq4:** Fourier Frequency corresponding to 4th highest amplitude
10. **Freq5:** Fourier Frequency corresponding to 5th highest amplitude
11. **Amp1:** Amplitude corresponding to 1st highest fourier transformed wave.
12. **Amp2:** Amplitude corresponding to 2nd highest fourier transformed wave.
13. **Amp3:** Amplitude corresponding to 3rd highest fourier transformed wave.
14. **Amp4:** Amplitude corresponding to 4th highest fourier transformed wave.
15. **Amp5:** Amplitude corresponding to 5th highest fourier transformed wave.
16. **Latitude:** Latitude of Cluster center.
17. **Longitude:** Longitude of Cluster Center.
18. **WeekDay:** Day of week of pickup.
19. **WeightedAvg:**: Weighted Moving Average Prediction values.

Data Preparation for regression models

Before we start predictions using the tree based regression models we take Jan 2016 pickup data and split it such that for every region we have 80% data in train and 20% in test, ordered date-wise for every region.

In [91]:

```
print("size of total train data :" +str(int(133770*0.8)))
print("size of total test data :" +str(int(133770*0.2)))
```

```
size of total train data :107016
size of total test data :26754
```

In [92]:

```
print("size of train data for one cluster:" +str(int(4459*0.8)))
print("size of total test data for one cluster:" +str(int(4459*0.2)))
```

```
size of train data for one cluster:3567
size of total test data for one cluster:891
```

In [93]:

```
train_previousFive_pickups  = [feat[i*4459:(4459*i+3567)] for i in range(30)]
test_previousFive_pickups  = [feat[(i*4459)+3567:(4459*(i+1))] for i in range(30)]
```

In [94]:

```
train_fourier_frequencies = [frequency_lists[i*4459:(4459*i+3567)] for i in range(30)]
test_fourier_frequencies = [frequency_lists[(i*4459)+3567:(4459*(i+1))] for i in range(30)]
```

In [96]:

```
train_fourier_amplitudes = [amplitude_lists[i*4459:(4459*i+3567)] for i in range(30)]
test_fourier_amplitudes = [amplitude_lists[(i*4459)+3567:(4459*(i+1))] for i in range(30)]
```

In [97]:

```
print("Train Data: Total number of clusters = {}. Number of points in each cluster = {}. Total num
ber of training points = {}".format(len(train_previousFive_pickups),
len(train_previousFive_pickups[0]), len(train_previousFive_pickups)*len(train_previousFive_pickups[
0])))
print("Test Data: Total number of clusters = {}. Number of points in each cluster = {}. Total numb
er of test points = {}".format(len(test_previousFive_pickups), len(test_previousFive_pickups[0]),
len(test_previousFive_pickups)*len(test_previousFive_pickups[0])))
```

```
Train Data: Total number of clusters = 30. Number of points in each cluster = 3567. Total number o
f training points = 107010
Test Data: Total number of clusters = 30. Number of points in each cluster = 892. Total number of
test points = 26760
```

In [98]:

```
#80% train data
train_lat = [i[:3567] for i in lat]
train_lon = [i[:3567] for i in lon]
train_weekDay = [i[:3567] for i in day_of_week]
train_weighted_avg = [i[:3567] for i in predicted_pickup_values_list]
train_TruePickups = [i[:3567] for i in TruePickups]
```

In [99]:

```
# 20% data as test data
test_lat = [i[3567:] for i in lat]
test_lon = [i[3567:] for i in lon]
test_weekDay = [i[3567:] for i in day_of_week]
test_weighted_avg = [i[3567:] for i in predicted_pickup_values_list]
test_TruePickups = [i[3567:] for i in TruePickups]
```

In [100]:

```
# convert from lists of lists of list to lists of list
train_pickups = []
test_pickups = []
train_freq = []
test_freq = []
train_amp = []
test_amp = []
for i in range(30):
    train_pickups.extend(train_previousFive_pickups[i])
    test_pickups.extend(test_previousFive_pickups[i])
    train_freq.extend(train_fourier_frequencies[i])
    test_freq.extend(test_fourier_frequencies[i])
    train_amp.extend(train_fourier_amplitudes[i])
    test_amp.extend(test_fourier_amplitudes[i])
```

In [101]:

```
train_prevPickups_freq_amp = np.hstack((train_pickups, train_freq, train_amp))
test_prevPickups_freq_amp = np.hstack((test_pickups, test_freq, test_amp))
```

In [102]:

```
print("Number of data points in train data = {}. Number of columns till now =
{}".format(len(train_prevPickups_freq_amp), len(train_prevPickups_freq_amp[0])))
print("Number of data points in test data = {}. Number of columns till now =
{}".format(len(test_prevPickups_freq_amp), len(test_prevPickups_freq_amp[0])))
```

```
Number of data points in train data = 107010. Number of columns till now = 15
```

```
Number of data points in test data = 26760. Number of columns till now = 15
```

In [103]:

```
# 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]

train_flat_lat = sum(train_lat, [])
train_flat_lon = sum(train_lon, [])
train_flat_weekDay = sum(train_weekDay, [])
train_weighted_avg_flat = sum(train_weighted_avg, [])
train_TruePickups_flat = sum(train_TruePickups, [])

test_flat_lat = sum(test_lat, [])
test_flat_lon = sum(test_lon, [])
test_flat_weekDay = sum(test_weekDay, [])
test_weighted_avg_flat = sum(test_weighted_avg, [])
test_TruePickups_flat = sum(test_TruePickups, [])
```

In [104]:

```
#train dataframe
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1', 'freq1', 'freq2','freq3','freq4','freq5', 'Amp1', 'A
mp2', 'Amp3', 'Amp4', 'Amp5']
Train_DF = pd.DataFrame(data = train_prevPickups_freq_amp, columns = columns)
Train_DF["Latitude"] = train_flat_lat
Train_DF["Longitude"] = train_flat_lon
Train_DF["WeekDay"] = train_flat_weekDay
Train_DF["WeightedAvg"] = train_weighted_avg_flat
```

In [105]:

```
#test dataframe
Test_DF = pd.DataFrame(data = test_prevPickups_freq_amp, columns = columns)
Test_DF["Latitude"] = test_flat_lat
Test_DF["Longitude"] = test_flat_lon
Test_DF["WeekDay"] = test_flat_weekDay
Test_DF["WeightedAvg"] = test_weighted_avg_flat
```

In [106]:

```
print("Shape of train data = "+str(Train_DF.shape))
print("Shape of test data = "+str(Test_DF.shape))
```

```
Shape of train data = (107010, 19)
Shape of test data = (26760, 19)
```

In [107]:

```
Train_DF.head(15)
```

Out[107]:

	ft_5	ft_4	ft_3	ft_2	ft_1	freq1	freq2	freq3	freq4	freq5	Amp1	Amp2	Amp3	
0	0.0	0.0	0.0	0.0	0.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
1	0.0	0.0	0.0	0.0	0.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
2	0.0	0.0	0.0	0.0	0.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
3	0.0	0.0	0.0	0.0	104.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
4	0.0	0.0	0.0	104.0	242.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
5	0.0	0.0	104.0	242.0	299.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58

	ft_5	ft_4	ft_3	ft_2	ft_1	freq1	freq2	freq3	freq4	freq5	Amp1	Amp2	Amp3	
6	0.0	104.0	242.0	299.0	327.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
7	104.0	242.0	299.0	327.0	340.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
8	242.0	299.0	327.0	340.0	316.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
9	299.0	327.0	340.0	316.0	325.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
10	327.0	340.0	316.0	325.0	323.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
11	340.0	316.0	325.0	323.0	311.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
12	316.0	325.0	323.0	311.0	290.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
13	325.0	323.0	311.0	290.0	266.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
14	323.0	311.0	290.0	266.0	260.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58

In [108]:

```
Test_DF.head(15)
```

Out[108]:

	ft_5	ft_4	ft_3	ft_2	ft_1	freq1	freq2	freq3	freq4	freq5	Amp1	Amp2	Amp3	
0	235.0	242.0	250.0	291.0	281.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
1	242.0	250.0	291.0	281.0	274.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
2	250.0	291.0	281.0	274.0	240.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
3	291.0	281.0	274.0	240.0	233.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
4	281.0	274.0	240.0	233.0	246.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
5	274.0	240.0	233.0	246.0	215.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
6	240.0	233.0	246.0	215.0	254.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
7	233.0	246.0	215.0	254.0	212.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
8	246.0	215.0	254.0	212.0	177.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
9	215.0	254.0	212.0	177.0	181.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
10	254.0	212.0	177.0	181.0	139.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
11	212.0	177.0	181.0	139.0	187.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
12	177.0	181.0	139.0	187.0	190.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
13	181.0	139.0	187.0	190.0	189.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58
14	139.0	187.0	190.0	189.0	126.0	0.006836	0.00708	0.013916	0.007812	0.000977	162504.411625	139486.014915	79801.885774	58

Standardizing train and test data

In [109]:

```
#standardizing the data

train_std = StandardScaler().fit_transform(Train_DF)
```

```
test_std = StandardScaler().fit_transform(Test_DF)
```

In [110]:

```
print(train_std.shape)
print(test_std.shape)
```

```
(107010, 19)
(26760, 19)
```

For some odd reason,my kernel kept crashing while running random forest regressor. So I decided to save my train and test files and run the models separately in another notebook.

In [118]:

```
print(type(Train_DF))
print(type(train_TruePickups_flat))
print(type(Test_DF))
print(type(test_TruePickups_flat))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
<class 'list'>
```

In [122]:

```
#https://datatofish.com/export-dataframe-to-csv/
Train_DF_csv = Train_DF.to_csv (r'Train_DF.csv', index = None, header=True)
#Don't forget to add '.csv' at the end of the path
```

In [125]:

```
#https://stackoverflow.com/questions/899103/writing-a-list-to-a-file-with-python
with open('train_TruePickups_flat.txt', 'w') as f:
    for item in train_TruePickups_flat:
        f.write("%s\n" % item)
```

In [126]:

```
#https://datatofish.com/export-dataframe-to-csv/
Test_DF_csv = Test_DF.to_csv (r'Test_DF.csv', index = None, header=True)
#Don't forget to add '.csv' at the end of the path
```

In [127]:

```
#https://stackoverflow.com/questions/899103/writing-a-list-to-a-file-with-python
with open('test_TruePickups_flat.txt', 'w') as f:
    for item in test_TruePickups_flat:
        f.write("%s\n" % item)
```

Please open nyc_final_models.ipynb to see how the models are performing.