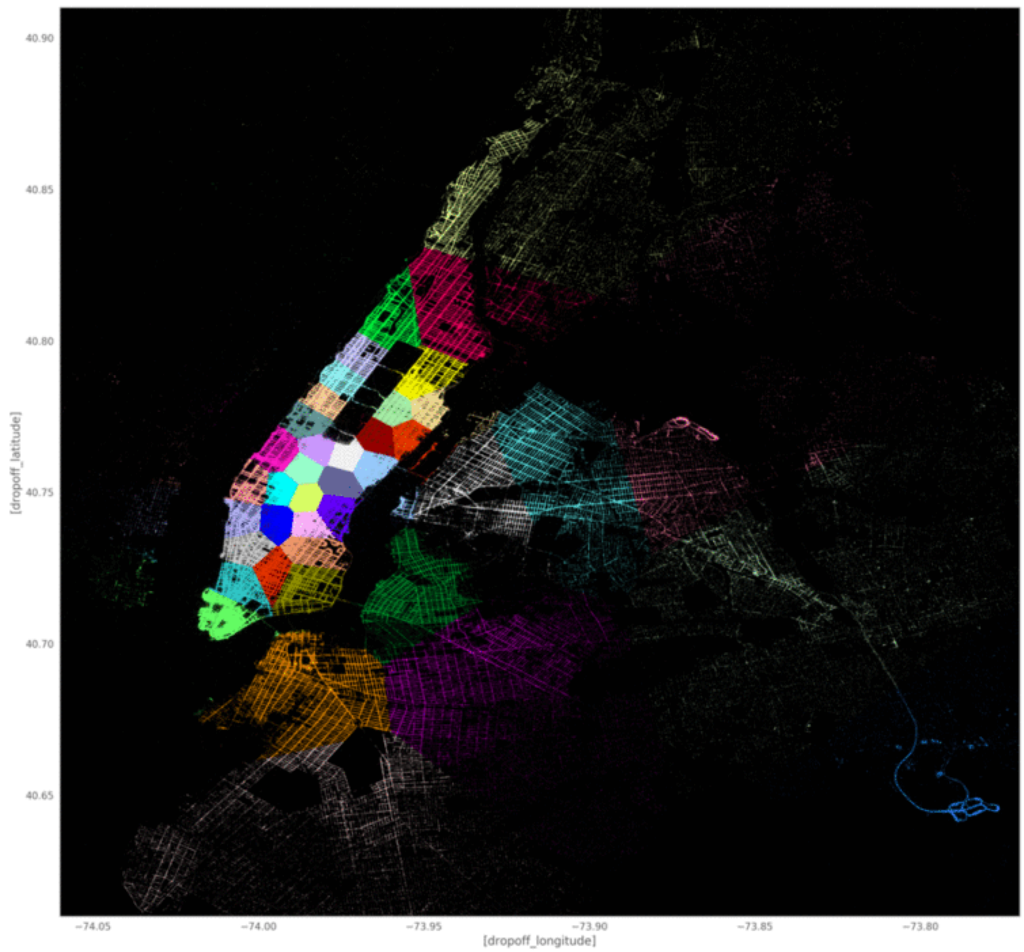
Taxi demand prediction in New York City



In [2]:
```
!pip install gpxpy
```

```
Requirement already satisfied: gpxpy in /usr/local/lib/python3.6/dist-packages
(1.3.5)
```

In [0]:
```
#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-tutoria
import dask.dataframe as dd#similar to pandas

import pandas as pd#pandas to create small dataframes

from IPython.display import HTML, display
# 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
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,lo
import gpxpy.geo #Get the haversine distance

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

# download migwin: https://mingw-w64.org/doku.php/download/mingw-builds
# install it in your system and keep the path, migw_path ='installed path'
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mi
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.tree import export_graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
import pydotplus
```

In [17]:

```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages
(1.5.3)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.
6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packa
ges (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-package
s (from kaggle) (2019.3.9)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist
-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packag
es (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
(from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-
packages (from kaggle) (3.0.1)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->kaggle) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.
6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: text-unidecode==1.2 in /usr/local/lib/python3.6/
dist-packages (from python-slugify->kaggle) (1.2)
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[17]:

{'kaggle.json': b'{"username":"pankajkarki","key":"563115ce1ea9892ab835dfbe5b8a
cba1"}'}

In [18]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d pankajkarki/taxidemand

!ls
```

```
Downloading taxidemand.zip to /content
 99% 1.75G/1.76G [00:18<00:00, 88.3MB/s]
100% 1.76G/1.76G [00:18<00:00, 102MB/s]
df_test   kaggle.json  taxidemand.zip    tsne_train_output
df_train  sample_data  tsne_test_output
```

In [19]:
```
!unzip taxidemand.zip
```

```
Archive:  taxidemand.zip
  inflating: yellow_tripdata_2015-01.csv
  inflating: yellow_tripdata_2016-01.csv
  inflating: yellow_tripdata_2016-02.csv
  inflating: yellow_tripdata_2016-03.csv
```

Data Information

Ge the data from : [http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml (http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml)](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)

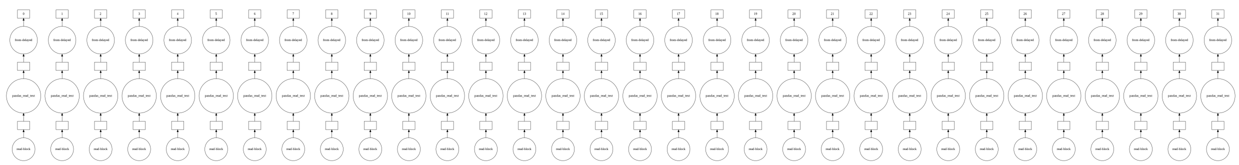
file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	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 [4]:
```python
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_datafr
jan_2015 = dd.read_csv('yellow_tripdata_2015-01.csv')
print(jan_2015.columns)
```

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

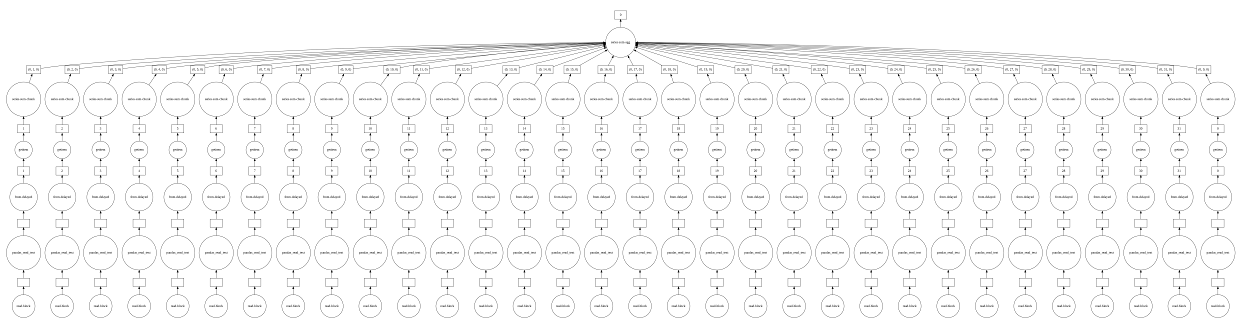
In [0]:
```python
# However unlike Pandas, operations on dask.dataframes don't trigger immediate co
# instead they add key-value pairs to an underlying Dask graph. Recall that in th
# circles are operations and rectangles are results.

# to see the visulaization you need to install graphviz
# pip3 install graphviz if this doesnt work please check the install_graphviz.jpg
jan_2015.visualize()
```

Out[202]:



In [0]:
```python
jan_2015.fare_amount.sum().visualize()
```

Out[203]:



Features in the dataset:

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.

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

ML Problem Formulation

Time-series forecasting and Regression

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

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

Performance metrics

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

Data Cleaning

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

In [5]:
```
#table below shows few datapoints along with all our features
jan_2015.head(5)
```

Out[5]:

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

1. Pickup Latitude and Pickup Longitude

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

In [6]:
```
# all the points outside newyork city to pickup_outliers
pickup_outliers = jan_2015[((jan_2015.pickup_longitude <= -74.15) | (jan_2015.pic
                    (jan_2015.pickup_longitude >= -73.7004) | (jan_2015.pickup_lat

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

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

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

map_osm
```

...

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

2. Dropoff Latitude & Dropoff Longitude

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

In [7]:
```
# Plotting dropoff cordinates which are outside the bounding box of New-York
# we will collect all the points outside the bounding box of newyork city to outl
outlier_locations = jan_2015[((jan_2015.dropoff_longitude <= -74.15) | (jan_2015.
                   (jan_2015.dropoff_longitude >= -73.7004) | (jan_2015.dropoff_l

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

# note: you dont need to remember any of these, you dont need indeepth knowledge 

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

# we will spot only first 100 outliers on the map, plotting all the outliers will
sample_locations = outlier_locations.head(100)
for i,j in sample_locations.iterrows():
    if int(j['pickup_latitude']) != 0:
        folium.Marker(list((j['dropoff_latitude'],j['dropoff_longitude']))).add_t
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 [0]:
```
#The timestamps are converted to unix so as to get duration(trip-time) & speed al

# in out data we have time in the formate "YYYY-MM-DD HH:MM:SS" we convert thiss 
# https://stackoverflow.com/a/27914405
def convert_to_unix(s):
    return time.mktime(datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S").timetup

# 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):
    #Compute several dask collections at once.
    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
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'
    #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','pick

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

# print(frame_with_durations.head())
#  passenger_count  trip_distance   pickup_longitude    pickup_latitude dropoff_l
#   1                  1.59       -73.993896           40.750111        -73.97478
#   1                   3.30        -74.001648          40.724243        -73.99441
#   1                   1.80        -73.963341          40.802788        -73.95182
#   1                   0.50        -74.009087          40.713818        -74.00432
#   1                   3.00        -73.971176          40.762428        -74.00418
frame_with_durations = return_with_trip_times(jan_2015)
```

In [9]:
```
print(frame_with_durations.head())
```

```
   passenger_count  trip_distance  pickup_longitude  pickup_latitude  \
0                1           1.59        -73.993896        40.750111
1                1           3.30        -74.001648        40.724243
2                1           1.80        -73.963341        40.802788
3                1           0.50        -74.009087        40.713818
4                1           3.00        -73.971176        40.762428

   dropoff_longitude  dropoff_latitude  total_amount  trip_times  \
0         -73.974785         40.750618         17.05   18.050000
1         -73.994415         40.759109         17.80   19.833333
2         -73.951820         40.824413         10.80   10.050000
3         -74.004326         40.719986          4.80    1.866667
4         -74.004181         40.742653         16.30   19.316667

   pickup_times      Speed
0  1.421349e+09   5.285319
1  1.420922e+09   9.983193
2  1.420922e+09  10.746269
3  1.420922e+09  16.071429
4  1.420922e+09   9.318378
```

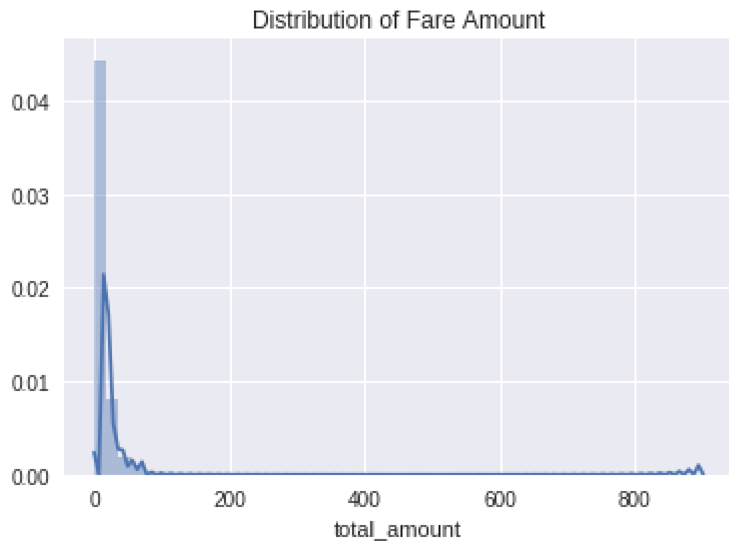
Exploratory Data Analysis

In [166]:
```
sns.distplot(frame_with_durations['total_amount'])

plt.title('Distribution of Fare Amount')
```

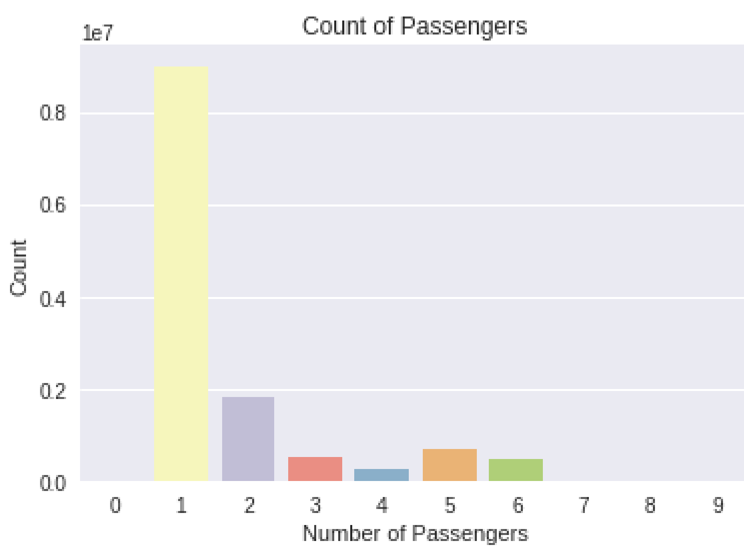
Out[166]: Text(0.5, 1.0, 'Distribution of Fare Amount')

In [186]:
```
sns.distplot(clean_df['total_amount'])

plt.title('Distribution of Fare Amount')
```

Out[186]: Text(0.5, 1.0, 'Distribution of Fare Amount')



In [168]:
```
passenger_count = frame_with_durations.groupby(['passenger_count']).count()

sns.barplot(passenger_count.index, passenger_count['pickup_times'], palette = "Se

plt.xlabel('Number of Passengers')
plt.ylabel('Count')
plt.title('Count of Passengers')
plt.show()
```

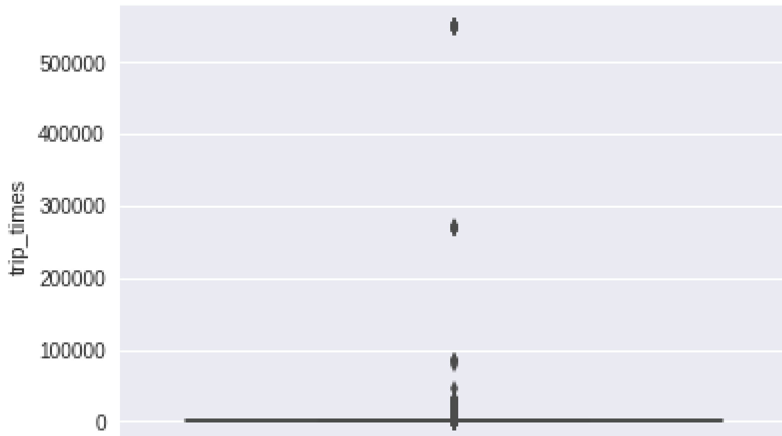


In [170]:
```
passenger_fare = frame_with_durations.groupby(['passenger_count']).mean()

sns.barplot(passenger_fare.index, passenger_fare['total_amount'], palette = "Set3

plt.xlabel('Number of Passengers')
plt.ylabel('Average Fare Price')
plt.title('Average Fare Price for Number of Passengers')
plt.show()
```

Observation:

1. From first and second plot we can see that the fare starts from 0 and reches upto 1000. After removing the outliers.
2. From third plot it is clear that mostly single number of passenger tarvels in the cab and it can exceed upto 7.
3. From the fourth plot we can say that if the number of passengers is too many the fare is also high.

In [10]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt

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



In [11]:
```
#calculating 0-100th percentile to find a the correct percentile value for remova
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 [12]:
```
#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
```

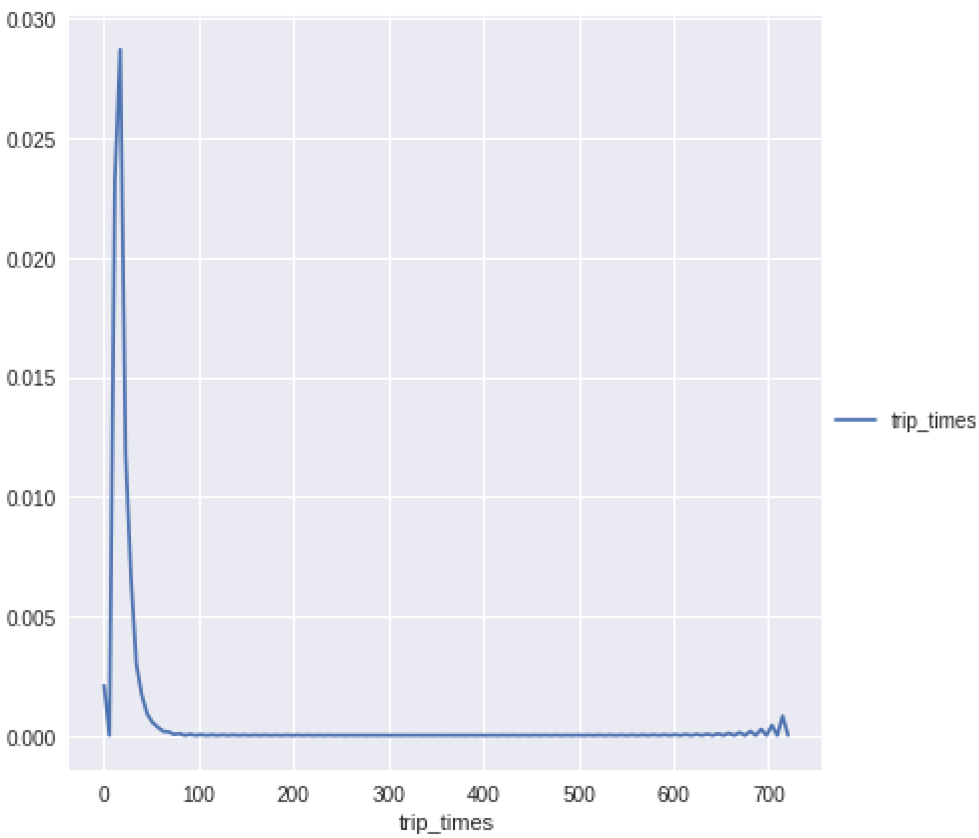
In [13]:
```
for i in np.arange(0.0, 1.0, 0.1):
    var =frame_with_durations["trip_times"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/1
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 46.75
99.1 percentile value is 48.06666666666667
99.2 percentile value is 49.56666666666667
99.3 percentile value is 51.28333333333333
99.4 percentile value is 53.31666666666667
99.5 percentile value is 55.833333333333336
99.6 percentile value is 59.13333333333333
99.7 percentile value is 63.9
99.8 percentile value is 71.86666666666666
99.9 percentile value is 101.6
100 percentile value is  548555.6333333333
```

In [0]:
```
#removing data based on our analysis and TLC regulations
updated_duration_of_trip =frame_with_durations[(frame_with_durations.trip_times>1
```

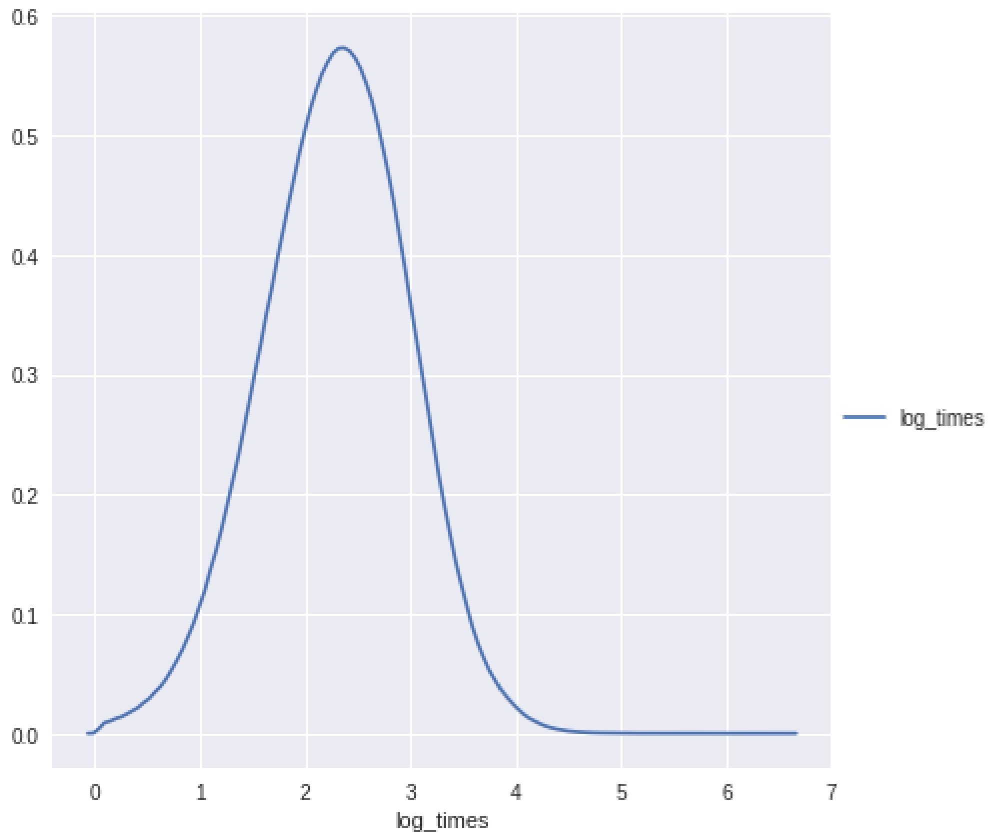
In [15]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_times", data = updated_duration_of_trip)
plt.show()
```

In [16]:
```
#pdf of trip-times after removing the outliers
sns.FacetGrid(updated_duration_of_trip,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```

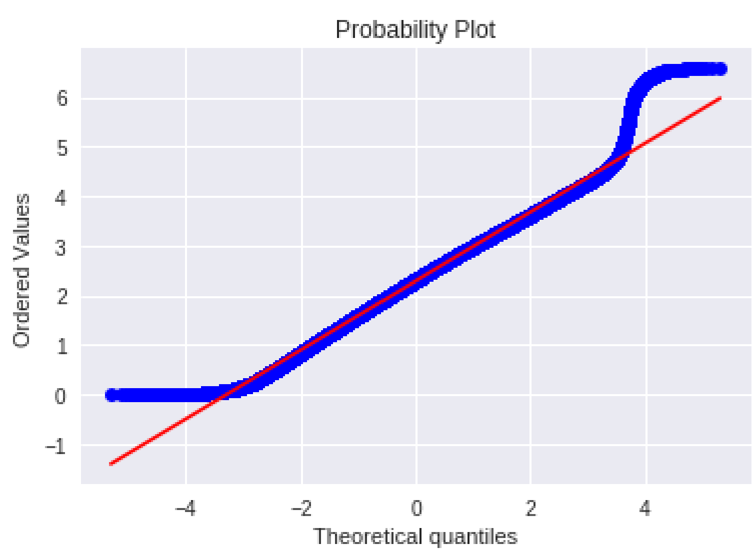


In [0]:
```
#converting the values to log-values to chec for log-normal
import math
updated_duration_of_trip['log_times']=[math.log(i) for i in updated_duration_of_t
```

In [18]:
```
#pdf of log-values
sns.FacetGrid(updated_duration_of_trip,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```

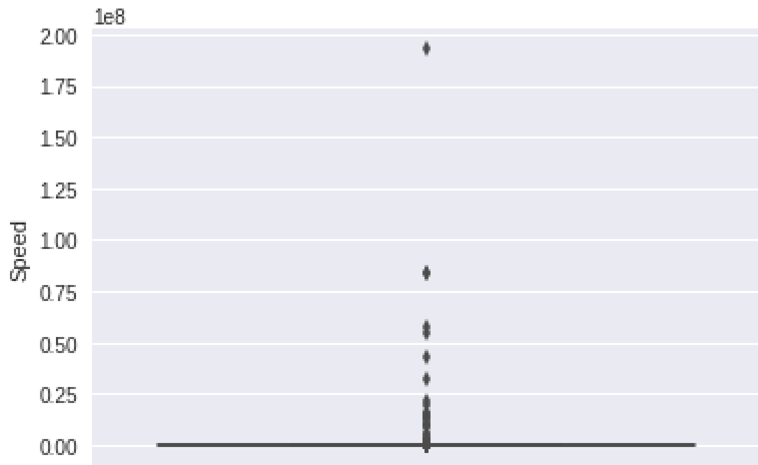


In [19]:
```
#Q-Q plot for checking if trip-times is log-normal
import scipy.stats
import matplotlib.pyplot as plt
scipy.stats.probplot(updated_duration_of_trip['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [20]:
```
# check for any outliers in the data after trip duration outliers removed
# box-plot for speeds with outliers
updated_duration_of_trip['Speed'] = 60*(updated_duration_of_trip['trip_distance'].
sns.boxplot(y="Speed", data = updated_duration_of_trip)
plt.show()
```



In [21]:
```
#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 = updated_duration_of_trip["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 [22]:
```
#calculating speed values at each percntile 90,91,92,93,94,95,96,97,98,99,100
for i in range(90,100):
    var = updated_duration_of_trip["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 [23]:
```
#calculating speed values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5,99.6,99
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["Speed"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/1
print("100 percentile value is ",var[-1])
```

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

In [0]:
```
#removing further outliers based on the 99.9th percentile value
updated_duration_of_trip=updated_duration_of_trip[(updated_duration_of_trip.Speed
```

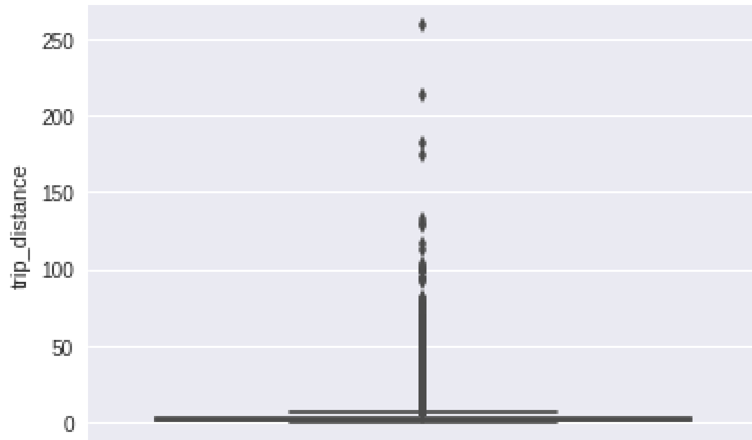
In [25]:
```
#avg.speed of cabs in New-York
sum(updated_duration_of_trip["Speed"]) / float(len(updated_duration_of_trip["Spee
```

Out[25]: 12.452320837813998

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

4. Trip Distance

In [26]:
```
# 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 = updated_duration_of_trip)
plt.show()
```



In [27]:
```
#calculating trip distance values at each percntile 0,10,20,30,40,50,60,70,80,90,.
for i in range(0,100,10):
    var = updated_duration_of_trip["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 0.67
20 percentile value is 0.9
30 percentile value is 1.1
40 percentile value is 1.39
50 percentile value is 1.7
60 percentile value is 2.08
70 percentile value is 2.61
80 percentile value is 3.6
90 percentile value is 5.98
100 percentile value is  258.9
```

In [28]:
```
#calculating trip distance values at each percntile 90,91,92,93,94,95,96,97,98,99
for i in range(90,100):
    var = updated_duration_of_trip["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])
```

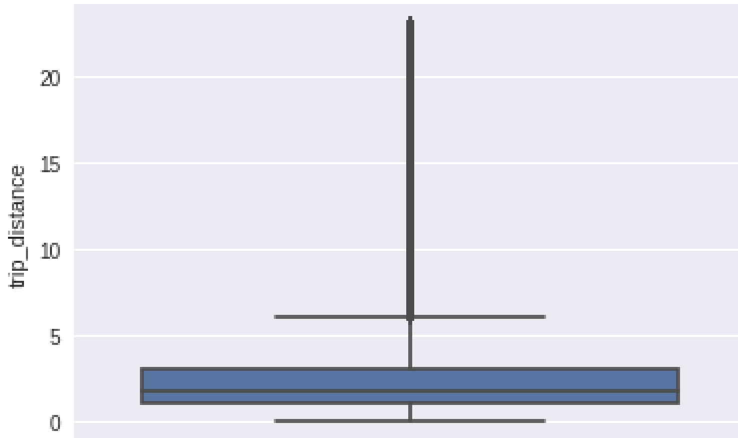
```
90 percentile value is 5.98
91 percentile value is 6.47
92 percentile value is 7.09
93 percentile value is 7.87
94 percentile value is 8.74
95 percentile value is 9.6
96 percentile value is 10.6
97 percentile value is 12.1
98 percentile value is 16.06
99 percentile value is 18.18
100 percentile value is  258.9
```

In [29]:
```
#calculating trip distance values at each percntile 99.0,99.1,99.2,99.3,99.4,99.5
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["trip_distance"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/1
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 18.18
99.1 percentile value is 18.37
99.2 percentile value is 18.6
99.3 percentile value is 18.84
99.4 percentile value is 19.14
99.5 percentile value is 19.5
99.6 percentile value is 19.97
99.7 percentile value is 20.51
99.8 percentile value is 21.23
99.9 percentile value is 22.58
100 percentile value is  258.9
```

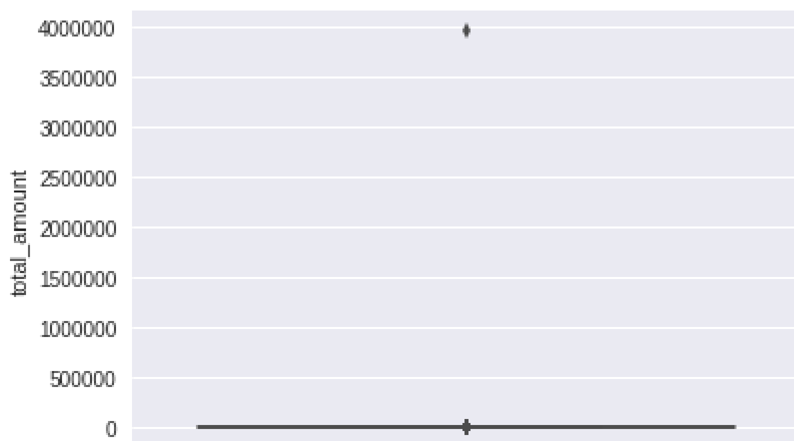
In [0]:
```
#removing further outliers based on the 99.9th percentile value
updated_duration_of_trip = updated_duration_of_trip[(updated_duration_of_trip.tri
```

In [31]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = updated_duration_of_trip)
plt.show()
```



5. Total Fare

In [32]:
```
# up to now we have removed the outliers based on trip durations, cab speeds, and
# 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 =updated_duration_of_trip)
plt.show()
```



In [33]:
```
#calculating total fare amount values at each percntile 0,10,20,30,40,50,60,70,80
for i in range(0,100,10):
    var = updated_duration_of_trip["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 6.35
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 [34]:
```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,9
for i in range(90,100):
    var = updated_duration_of_trip["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 29.13
93 percentile value is 31.55
94 percentile value is 34.63
95 percentile value is 38.13
96 percentile value is 42.13
97 percentile value is 47.53
98 percentile value is 57.68
99 percentile value is 65.8
100 percentile value is  3950611.6
```

In [35]:
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,
for i in np.arange(0.0, 1.0, 0.1):
    var = updated_duration_of_trip["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/1
print("100 percentile value is ",var[-1])
```

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

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

Remove all outliers/erronous points.

In [0]:
```
#removing all outliers based on our univariate analysis above
def remove_outliers(new_df):

    a = new_df.shape[0]
    print ("Number of pickup records = ",a)

####################################################################################

    new_frame = new_df[((new_df.dropoff_longitude >= -74.15) & (new_df.dropoff_lo
                    (new_df.dropoff_latitude >= 40.5774) & (new_df.dropoff_lati
                    ((new_df.pickup_longitude >= -74.15) & (new_df.pickup_latit
                    (new_df.pickup_longitude <= -73.7004) &  (new_df.pickup_lat


    new_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 72
    new_frame = new_frame[(new_frame.trip_distance > 0) & (new_frame.trip_distanc
    new_frame = new_frame[(new_frame.Speed < 45.31) & (new_frame.Speed > 0)]
    new_frame = new_frame[(new_frame.total_amount <1000) & (new_frame.total_amoun

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

In [38]:
```
print ("Removing outliers in the month of Jan-2015")
print ("----")
clean_df = remove_outliers(frame_with_durations)
print("fraction of data points remaining after removing outliers",(len(clean_df)/
```

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12748986
Total outliers removed 377910
---

fraction of data points remaining after removing outliers 0.9703576425607495
```

Data-preperation

Clustering/Segmentation

In [0]:
```
# function for clustering

def find_regions(k):
    ''' number of clusters = k'''
    ''' returns cluster centers'''
    ''' each cluster represents a region'''

    kmeans = MiniBatchKMeans(n_clusters= k, batch_size=10000,random_state=42).fit

    cluster_centers = kmeans.cluster_centers_
    NumOfCluster = len(cluster_centers)
    return cluster_centers, NumOfCluster
```

In [0]:
```
# function to find distance between cluster

def min_distance(cluster_centers, n_clusters):
    '''number of cluster = n_clusters'''
    '''distances between regions are calculated as
       the distance between corresponding cluster centers'''

    # for any given region(cluster)
    # nice_points temp variable stores num of regions within radius 2 miles
    # bad_points temp variable stores num of regions not within 2 miles radius
    nice_points = 0
    bad_points = 0
    less2 = [] # store nice_points for each cluster
    more2 = [] # store bad points for each cluster
    min_dist=1000
    for i in range(0, n_clusters):

        nice_points = 0
        bad_points = 0

        for j in range(0, n_clusters):

            if j!=i:

                # gpxpy.geo gives distance between two latitudes and longitudes i
                # syntax: gpxpy.geo.haversine_distance(lat_1, long_1, lat_2, long_
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0], cl
                                                        cluster_centers[j][0],clu

                # 1 Mile = 1609.34 meter
                min_dist = min(min_dist,distance/(1609.34))
                if (distance/(1609.34)) <= 2:
                    nice_points +=1
                else:
                    bad_points += 1

        less2.append(nice_points)
        more2.append(bad_points)

    neighbours.append(less2)
    print("\n If Number of clusters: {}".format(n_clusters))
    print("Avg. Number of Clusters within 2 Miles radius: ", np.ceil(sum(less2)/l
    print("Avg. Number of Clusters NOT within 2 Miles radius: ",np.ceil(sum(more2
    print("Min inter-cluster distance = ",min_dist,"\n","---"*10)
```

In [43]:
```
#trying different cluster sizes to choose the right K in K-means
coords = clean_df[['pickup_latitude', 'pickup_longitude']].values
neighbours=[]

# choose number of clusters such that, more num of clusters are close to any clus
# at the same time make sure that the minimum inter cluster dist should not be ve

for increment in range(10, 100, 10):
    cluster_centers, NumOfClusters = find_regions(increment)
    min_distance(cluster_centers, NumOfClusters)
```

```
 If Number of clusters: 10
Avg. Number of Clusters within 2 Miles radius:  2.0
Avg. Number of Clusters NOT within 2 Miles radius:  8.0
Min inter-cluster distance =  1.0945442325142543
 ------------------------------

 If Number of clusters: 20
Avg. Number of Clusters within 2 Miles radius:  4.0
Avg. Number of Clusters NOT within 2 Miles radius:  16.0
Min inter-cluster distance =  0.7131298007387813
 ------------------------------

 If Number of clusters: 30
Avg. Number of Clusters within 2 Miles radius:  8.0
Avg. Number of Clusters NOT within 2 Miles radius:  22.0
Min inter-cluster distance =  0.5185088176172206
 ------------------------------

 If Number of clusters: 40
Avg. Number of Clusters within 2 Miles radius:  8.0
Avg. Number of Clusters NOT within 2 Miles radius:  32.0
Min inter-cluster distance =  0.5069768450363973
 ------------------------------

 If Number of clusters: 50
Avg. Number of Clusters within 2 Miles radius:  12.0
Avg. Number of Clusters NOT within 2 Miles radius:  38.0
Min inter-cluster distance =  0.365363025983595
 ------------------------------

 If Number of clusters: 60
Avg. Number of Clusters within 2 Miles radius:  14.0
Avg. Number of Clusters NOT within 2 Miles radius:  46.0
Min inter-cluster distance =  0.34704283494187155
 ------------------------------

 If Number of clusters: 70
Avg. Number of Clusters within 2 Miles radius:  16.0
Avg. Number of Clusters NOT within 2 Miles radius:  54.0
Min inter-cluster distance =  0.30502203163244707
 ------------------------------

 If Number of clusters: 80
Avg. Number of Clusters within 2 Miles radius:  18.0
```

```
Avg. Number of Clusters NOT within 2 Miles radius:  62.0
Min inter-cluster distance =  0.29220324531738534
 ------------------------------

 If Number of clusters: 90
Avg. Number of Clusters within 2 Miles radius:  21.0
Avg. Number of Clusters NOT within 2 Miles radius:  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 30

In [0]:
```
# for k= 50 clusters the Min inter-cluster distance only 0.3 miles apart from eac
# for k= 30 and 40 there Min inter-cluster distance is about 0.5 miles
# Avg. Number of Clusters within 2 Miles radius = 8 is also same for 30 and 40
# but Avg. Number of Clusters NOT within 2 Miles radius is less for k=30 than k =
# So we choose 30 clusters for solve the further problem
# Getting 30 clusters using the kmeans

kmeans = MiniBatchKMeans(n_clusters=30, batch_size=10000,random_state=0).fit(coor

# columns 'pickup_cluster' added
clean_df['pickup_cluster'] = kmeans.predict(clean_df[['pickup_latitude', 'pickup_
```

In [0]:
```
cluster_centers = kmeans.cluster_centers_
NumOfClusters = len(cluster_centers)
```

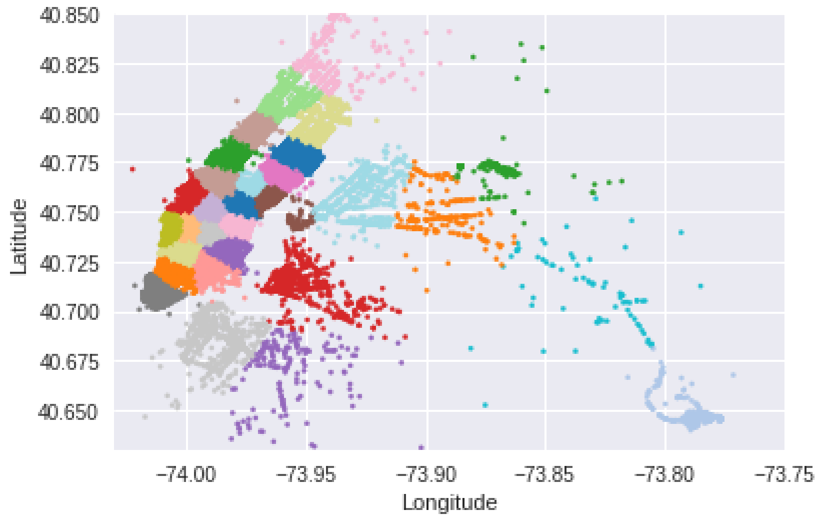
Plotting the cluster centers:

In [46]:
```
# Plotting the cluster centers on OSM
map_3 = folium.Map(location=[40.734695, -73.990372], tiles='Stamen Toner')
for i in range(NumOfClusters):
    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_3)
map_3
```

Out[46]:

Plotting the clusters:

In [47]:
```
#Visualising the clusters on a map
def plot_regions(frame):
    city_long_border = (-74.03, -73.75)
    city_lat_border = (40.63, 40.85)
    fig = fig, ax = plt.subplots(ncols=1, nrows=1)
    ax.scatter(frame.pickup_longitude.values[:100000],
               frame.pickup_latitude.values[:100000], s=5,
               c=frame.pickup_cluster.values[:100000], cmap='tab20')

    ax.set_xlim(city_long_border)
    ax.set_ylim(city_lat_border)
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    plt.show()

plot_regions(clean_df)
```



Time-binning

In [0]:
```
#Refer:https://www.unixtimestamp.com/
# 1420070400 : 2015-01-01 00:00:00
# 1451606400 : 2016-01-01 00:00:00
# 1454284800 : 2016-02-01 00:00:00
# 1456790400 : 2016-03-01 00:00:00


def add_pickup_bins(frame,month,year):


    '''subtract pickup time from the unix time of 12:00AM for start of the month'
    '''then divide that by 600 in order to make a 10minute bin'''

    unix_pick_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400],[1451606400,1454284800,1456790400]]

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

In [0]:
```
# column 'pickup_bins' added

jan_2015_frame = add_pickup_bins(clean_df,1,2015)

jan_2015_groupby = jan_2015_frame[['pickup_cluster','pickup_bins','trip_distance'
                   .groupby(['pickup_cluster','pickup_bins']).count()
```

In [50]:
```
# 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[50]:

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

In [51]:
```
# hear the trip_distance represents the number of pickups that are happend in tha
# this data frame has two indices
# primary index: pickup_cluster (cluster number)
# secondary index : pickup_bins (we devid whole months time into 10min intravels .
jan_2015_groupby.head()
```

Out[51]:

pickup_cluster	pickup_bins	trip_distance
0	33	138
	34	262
	35	311
	36	325
	37	381

In [0]:
```
# upto now we cleaned data and prepared data for the month 2015,

# now do the same operations for months Jan, Feb, March of 2016
# 1. get the dataframe which inlcudes only required colums
# 2. adding trip times, speed, unix time stamp of pickup_time
# 4. remove the outliers based on trip_times, speed, trip_duration, total_amount
# 5. add pickup_cluster to each data point
# 6. add pickup_bin (index of 10min intravel to which that trip belongs to)
# 7. group by data, based on 'pickup_cluster' and 'pickuo_bin'

# Data Preparation for the months of Jan,Feb and March 2016
def data_prep(month,kmeans,month_no,year_no):

    print ("Return df with required columns only")

    new_df = return_with_trip_times(month)

    print ("Remove outliers..")
    clean_df = remove_outliers(new_df)

    print ("Estimating clusters..")
    clean_df['pickup_cluster'] = kmeans.predict(clean_df[['pickup_latitude', 'pic

    print ("Final groupby..")
    final_frame = add_pickup_bins(clean_df, month_no, year_no)
    final_groupby_frame = final_frame[['pickup_cluster','pickup_bins','trip_dista
                          .groupby(['pickup_cluster','pickup_bins']).count()

    return final_frame, final_groupby_frame
```

In [55]:
```
month_jan_2016 = dd.read_csv('yellow_tripdata_2016-01.csv')
month_feb_2016 = dd.read_csv('yellow_tripdata_2016-02.csv')
month_mar_2016 = dd.read_csv('yellow_tripdata_2016-03.csv')

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

```
Return df with required columns only
Remove outliers..
Number of pickup records =  10906858
Total outliers removed 297784
---

Estimating clusters..
Final groupby..
Return df with required columns only
Remove outliers..
Number of pickup records =  11382049
Total outliers removed 308177
---

Estimating clusters..
Final groupby..
Return df with required columns only
Remove outliers..
Number of pickup records =  12210952
Total outliers removed 324635
---

Estimating clusters..
Final groupby..
```

Smoothing

In [0]:
```
# 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 wh
# we got an observation that there are some pickpbins that doesnt have any pickup
def unq_pickup_bins(frame):
    '''the indices of all the unique time_bins where'''
    ''' there is a pickup for all the 30 clusters'''
    values = []
    for i in range(0,30):
        new = frame[frame['pickup_cluster'] == i]
        list_unq = list(set(new['pickup_bins']))
        list_unq.sort()
        values.append(list_unq)
    return values
```

In [0]:
```
# for every month we get all indices of 10min intravels in which atleast one picku
#jan
jan_2015_unique = unq_pickup_bins(jan_2015_frame)
jan_2016_unique = unq_pickup_bins(jan_2016_frame)

#feb
feb_2016_unique = unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = unq_pickup_bins(mar_2016_frame)
```

In [58]:
```
# for each cluster number of 10min intravels with 0 pickups
for i in range(30):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",
          4464 - len(set(jan_2015_unique[i])))
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  149
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  169
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  39
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  319
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  45
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  97
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  325
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  24
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  39
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  39
------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  33
------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  48
------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  48
------------------------------------------------------------
```

```
for the  25 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  719
------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
```

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 [0]:

```
# 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
# 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 bi
# 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):
    '''Fills zero for every bin where no pickup data is present'''
    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 [0]:
```
# 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
# 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 bi
# 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
# 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 visit
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of th
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for left-limit or picku
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: last few values are missing,hence no right-limit pre
                        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: missing values are between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.
                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: first few values are missing,hence no left-limit pre
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]:
                            continue
                        else:
                            right_hand_limit=j
```

```
                        break
                    smoothed_value=count_values[ind]*1.0/((right_hand_limit-i)+1)
                    for j in range(i,right_hand_limit+1):
                        smoothed_bins.append(math.ceil(smoothed_value))
                    repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [0]:

```
#Filling Missing values of Jan-2015 with 0
jan_2015_fill = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2015_un

#Smoothing Missing values of Jan-2015
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uni
```

In [62]:

```
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*30*60/10 = 4320
# for each cluster we will have 4464 values, therefore 20*4464 = 89280 (length of
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

```
number of 10min intravels among all the clusters  133920
```

In [0]:

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

# Ans: consider we have data of some month in 2015 jan 1st, 10 _ _ _ 20, i.e ther
# 10min intravel, 0 pickups happened in 2nd 10mins intravel, 0 pickups happened i
# 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 c
# that are happened in the first 40min are same in both cases, but if you can obs
# wheen you are using smoothing we are looking at the future number of pickups wh

# 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 [0]:
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled 
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_uni
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_
mar_2016_smooth = fill_missing(mar_2016_groupby['trip_distance'].values,mar_2016_

# Making list of all the values of pickup data in every bin for a period of 3 mon
smooth16 = []

# a =[1,2,3]
# b = [2,3,4]
# a+b = [1, 2, 3, 2, 3, 4]

# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*31*60/10 = 4464
# regions_cum: it will contain 20 lists, each list will contain 4464+4176+4464 va
# that are happened for three months in 2016 data

for i in range(0,30):
    smooth16.append(jan_2016_smooth[4464*i:4464*(i+1)]  \
                       +feb_2016_smooth[4176*i:4176*(i+1)] \
                       +mar_2016_smooth[4464*i:4464*(i+1)])
```

In [65]:
```
print(len(smooth16))
len(smooth16[0])
```

```
30
```

Out[65]:
```
13104
```

In [0]:
```
#Preparing the Dataframe only with x(i) values as jan-2015 data and y(i) values a
ratios_jan = pd.DataFrame()
ratios_jan['Given']=jan_2015_smooth
ratios_jan['Prediction']=jan_2016_smooth
ratios_jan['Ratios']=ratios_jan['Prediction']*1.0/ratios_jan['Given']*1.0
```

Modelling: Baseline Models

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

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

Simple Moving Averages

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

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

In [0]:
```
def MA_R_Predictions(ratios,month):

    '''simple_moving_average_ratios'''

    predicted_ratio=(ratios['Ratios'].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_ra
        if i+1>=window_size:
            predicted_ratio=sum((ratios['Ratios'].values)[(i+1)-window_size:(i+1)
        else:
            predicted_ratio=sum((ratios['Ratios'].values)[0:(i+1)])/(i+1)


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

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

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

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

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

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

Weighted Moving Averages

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

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

In [0]:
```
def WA_R_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.5
    error=[]
    predicted_values=[]
    window_size=5
    predicted_ratio_values=[]
    for i in range(0,4464*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_ra
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Ratios'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff
        else:
            sum_values=0
            sum_of_coeff=0
            for j in range(i+1,0,-1):
                sum_values += j*(ratios['Ratios'].values)[j-1]
                sum_of_coeff+=j
            predicted_ratio=sum_values/sum_of_coeff

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

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

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

In [0]:
```
def WA_P_Predictions(ratios,month):
    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)[
        if i+1>=window_size:
            sum_values=0
            sum_of_coeff=0
            for j in range(window_size,0,-1):
                sum_values += j*(ratios['Prediction'].values)[i-window_size+j]
                sum_of_coeff+=j
            predicted_value=int(sum_values/sum_of_coeff)

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

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

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

Exponential Weighted Moving Averages

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

In exponential moving averages we use a single hyperparameter alpha (α) which is a value between 0 & 1 and based on the value of the hyperparameter alpha the weights and the window sizes are configured.
For eg. If $\alpha = 0.9$ then the number of days on which the value of the current iteration is based is~ $1/(1 - \alpha) = 10$ i.e. we consider values 10 days prior before we predict the value for the current

iteration. Also the weights are assigned using $2/(N+1) = 0.18$,where N = number of prior values being considered, hence from this it is implied that the first or latest value is assigned a weight of 0.18 which keeps exponentially decreasing for the subsequent values.

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

In [0]:
```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*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_ra
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].

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

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

In [0]:
```

def EA_P1_Predictions(ratios,month):
    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)[
        predicted_value =int((alpha*predicted_value) + (1-alpha)*((ratios['Predic

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

In [0]:
```
mean_err=[0]*6
median_err=[0]*6
ratios_jan,mean_err[0],median_err[0]=MA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[1],median_err[1]=MA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[2],median_err[2]=WA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[3],median_err[3]=WA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[4],median_err[4]=EA_R1_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[5],median_err[5]=EA_P1_Predictions(ratios_jan,'jan')
```

Comparison between baseline models

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

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------
-------------------------
Moving Averages (Ratios) -                             MAPE:  0.211616696487420
2       MSE:  7399.9824298088415
Moving Averages (2016 Values) -                        MAPE:  0.134854479726749
97        MSE:  326.3647028076464
--------------------------------------------------------------------------------
-------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.212698212180444
24       MSE:  6559.883602150538
Weighted Moving Averages (2016 Values) -               MAPE:  0.129432550289535
6       MSE:  296.25813918757467
--------------------------------------------------------------------------------
-------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.2122523879026215
MSE:  5155.116980286738
Exponential Moving Averages (2016 Values) -         MAPE:  0.12922266732265716
MSE:  293.96470280764635
```

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

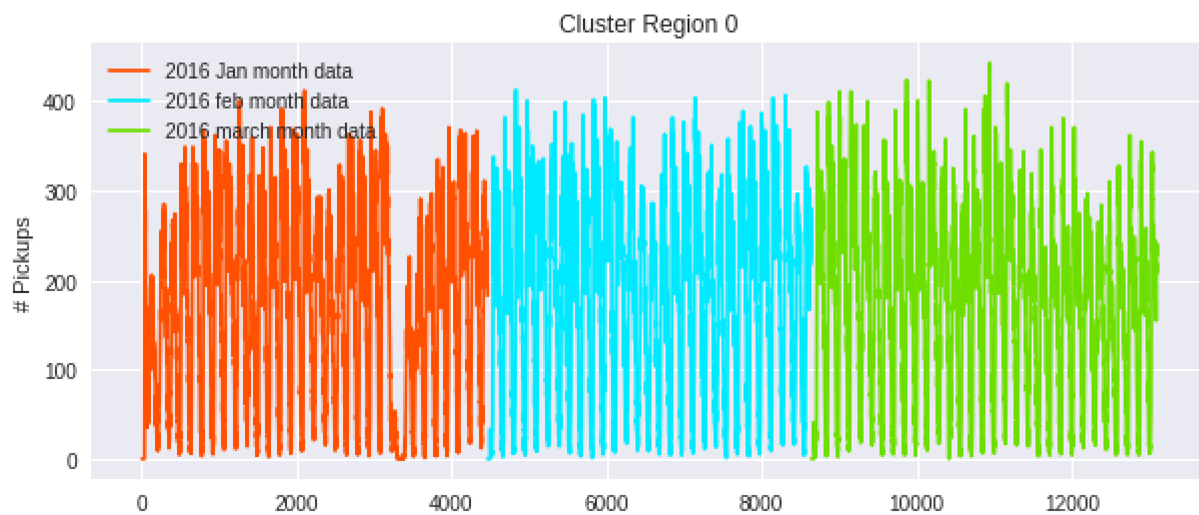
Time series and Fourier Transforms

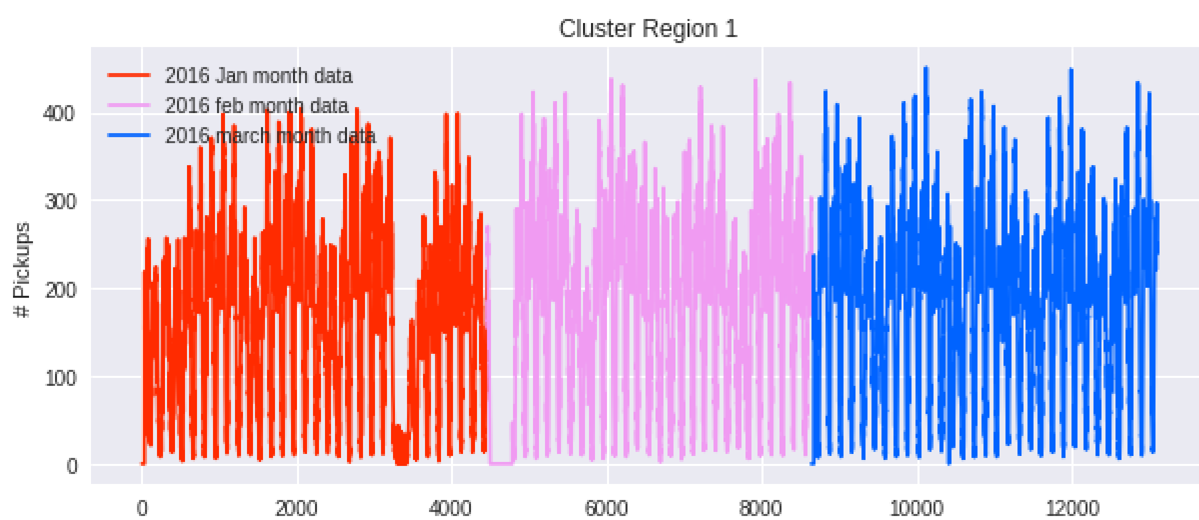
Ploting time series data

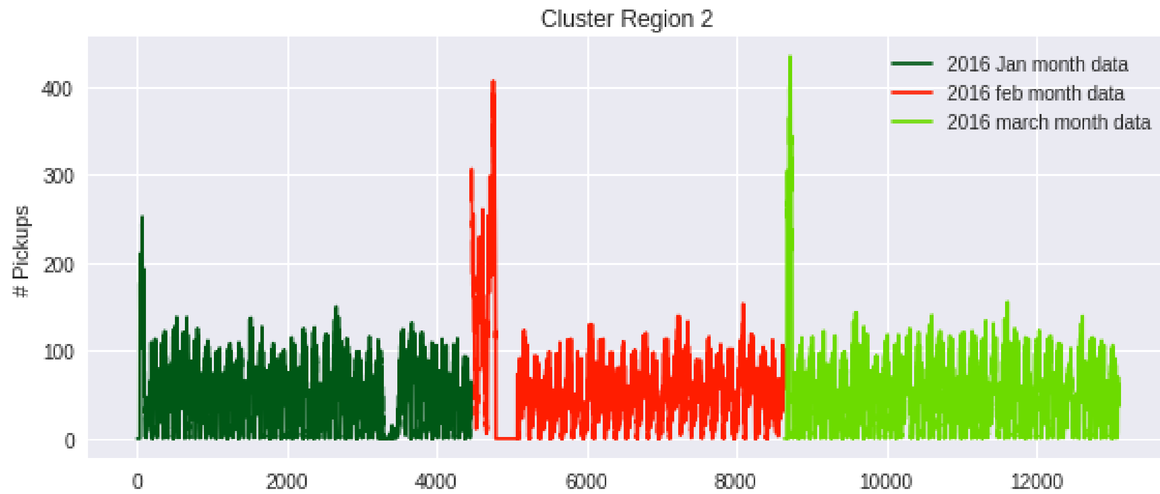
Plot and observe patterns,for each region and month to decide if Fourier Transform is useful

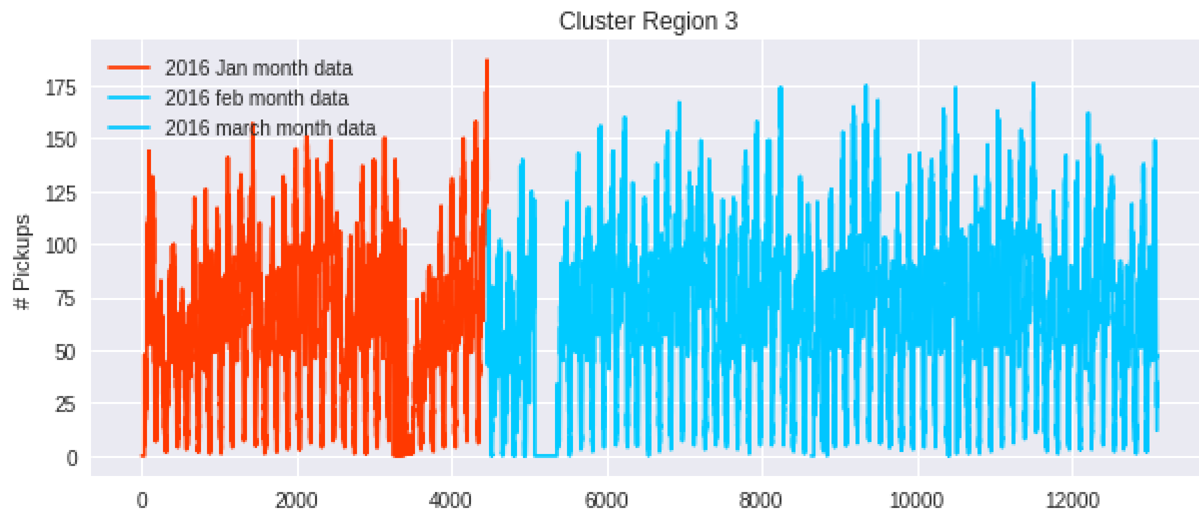
In [83]:

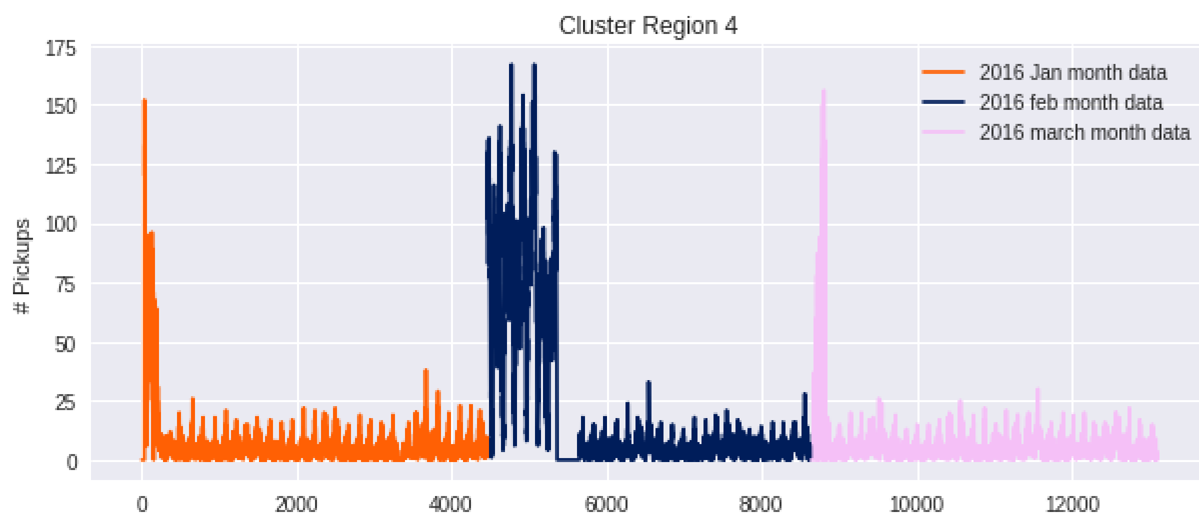
```
def uni_color():
    """There are better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())

first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))

for i in range(30):
    plt.figure(figsize=(10,4))
    plt.title("Cluster Region "+str(i))
    plt.ylabel("# Pickups")
    plt.plot(first_x, smooth16[i][:4464], color=uni_color(), label='2016 Jan mont
    plt.plot(second_x, smooth16[i][4464:8640], color=uni_color(), label='2016 feb
    plt.plot(third_x, smooth16[i][8640:], color=uni_color(), label='2016 march mo
    plt.legend()
    plt.show()
```

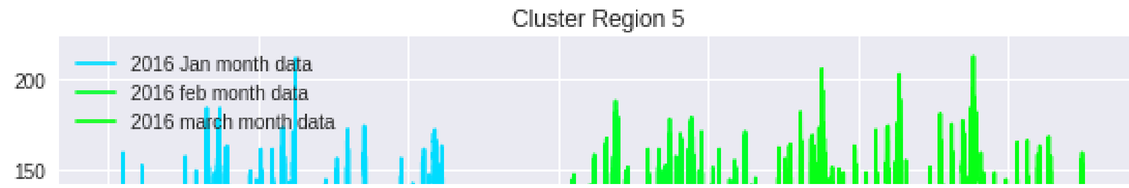


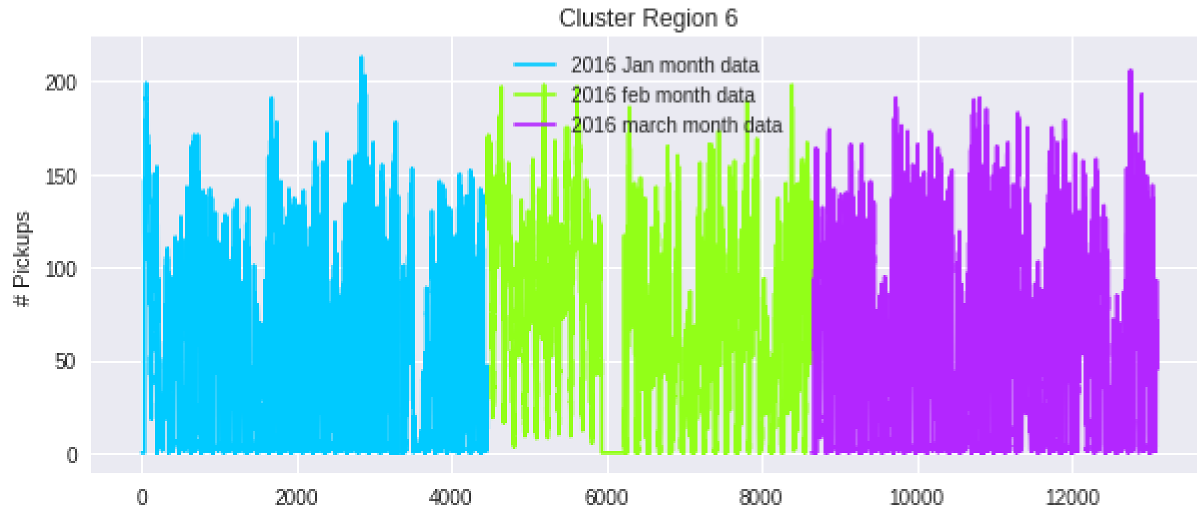


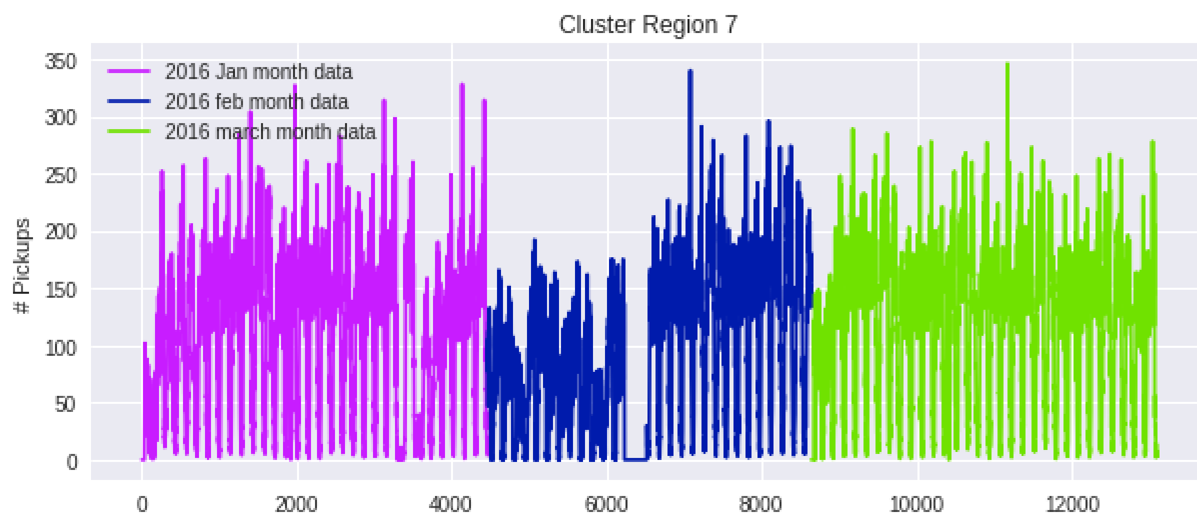

Cluster Region 2
Pickups
2016 Jan month data
2016 feb month data
2016 march month data

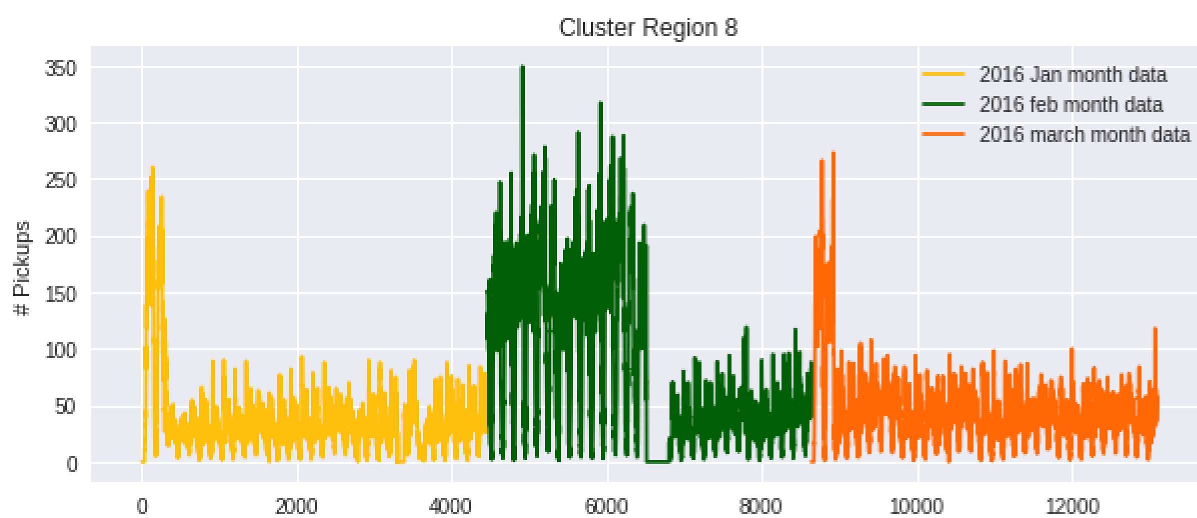

Cluster Region 3
Pickups
2016 Jan month data
2016 feb month data
2016 march month data

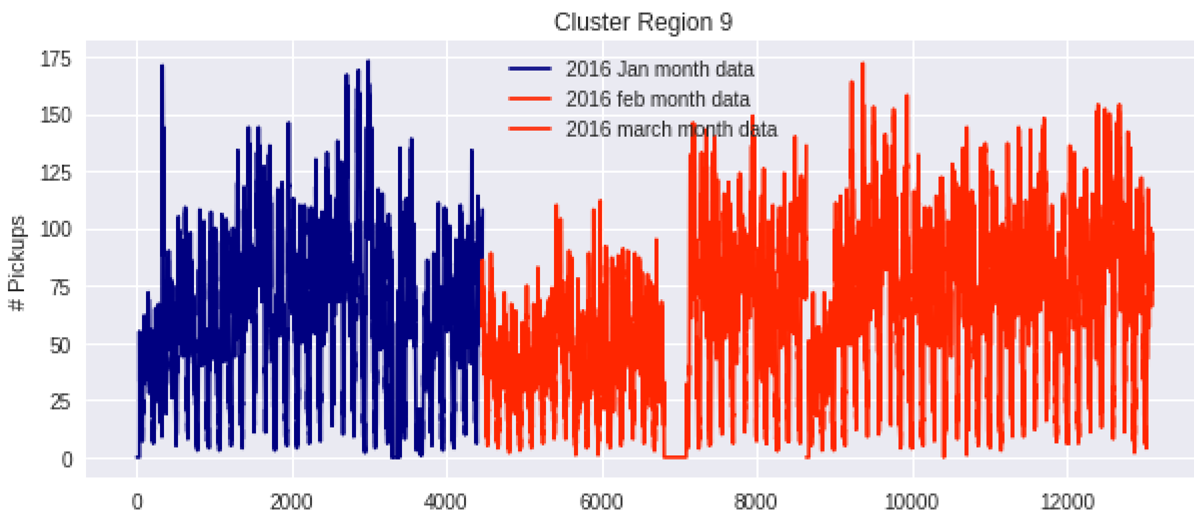

Cluster Region 4
Pickups
2016 Jan month data
2016 feb month data
2016 march month data

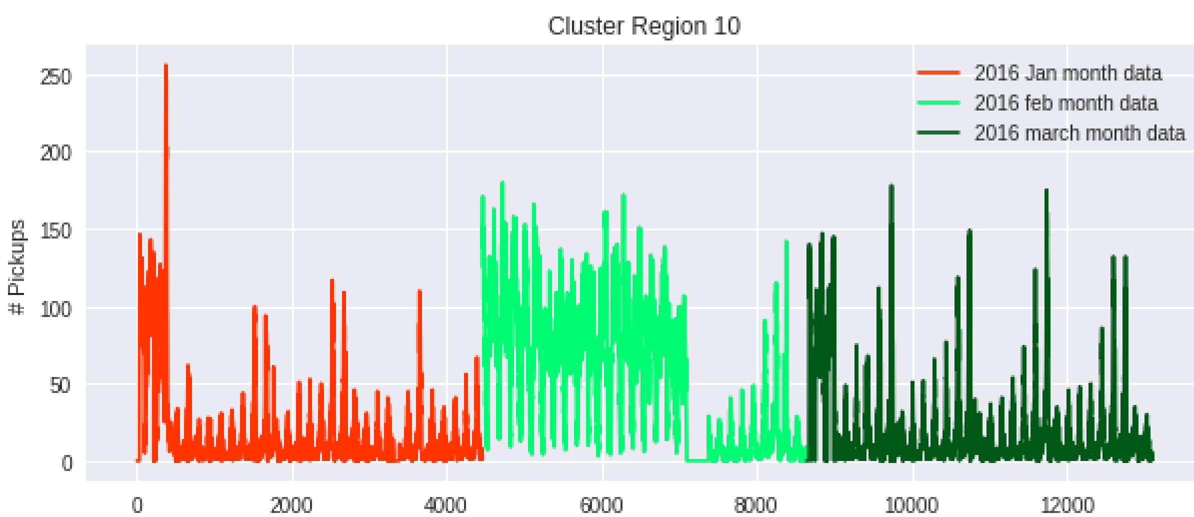

Cluster Region 5
2016 Jan month data
2016 feb month data
2016 march month data
200
150


Cluster Region 6
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

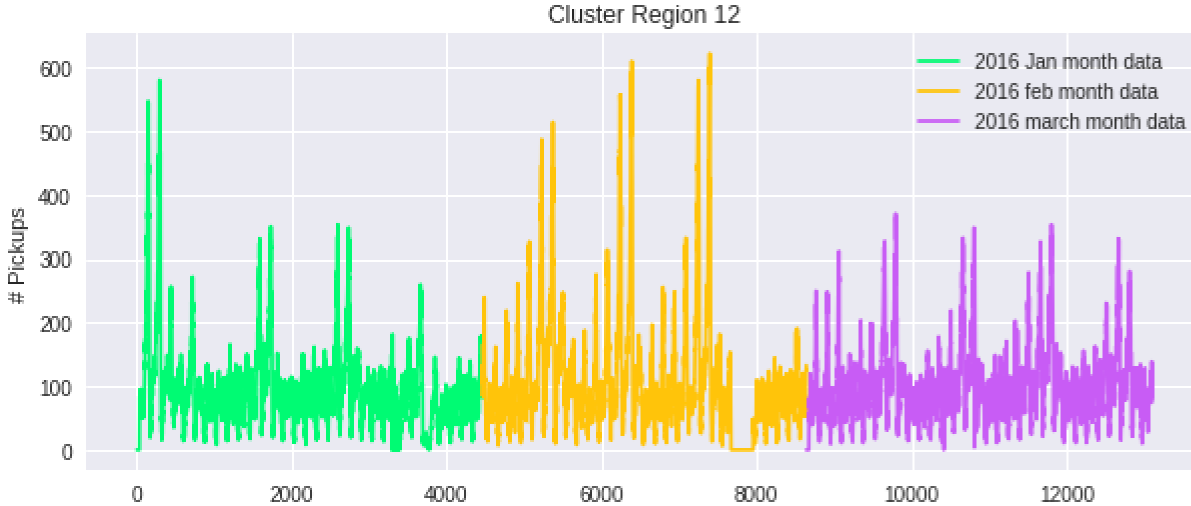

Cluster Region 7
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

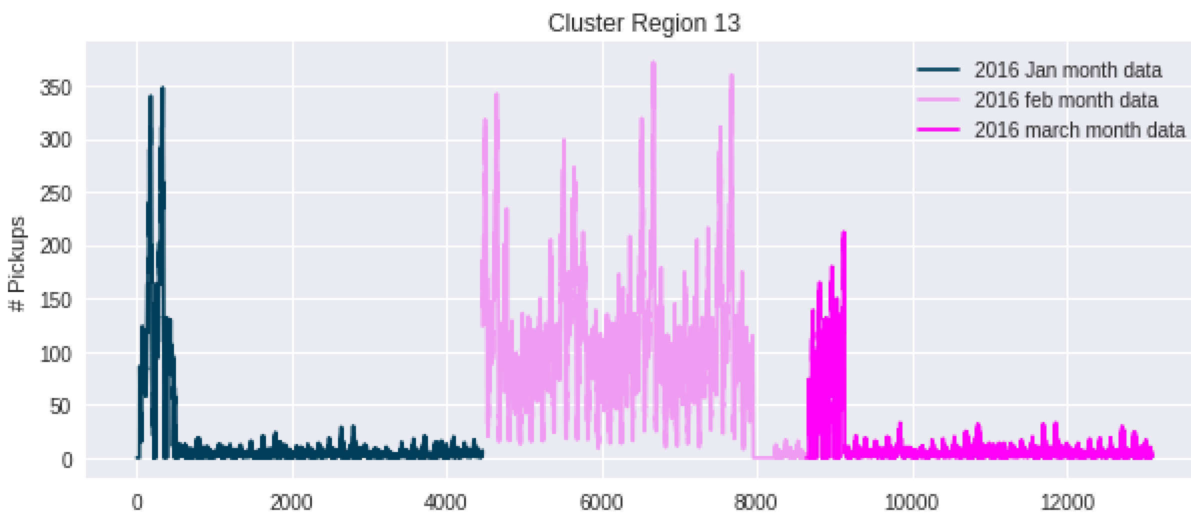

Cluster Region 8
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000


Cluster Region 9
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
175
150
125
100
75
50
25
0
0
2000
4000
6000
8000
10000
12000


Cluster Region 10
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

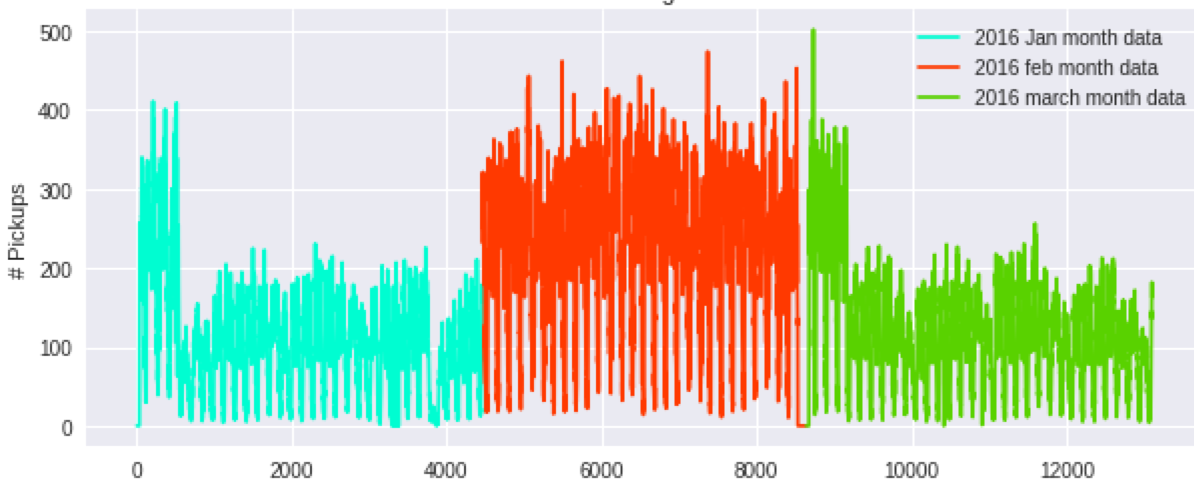

Cluster Region 11
700


Cluster Region 12
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
500
600
0
2000
4000
6000
8000
10000
12000

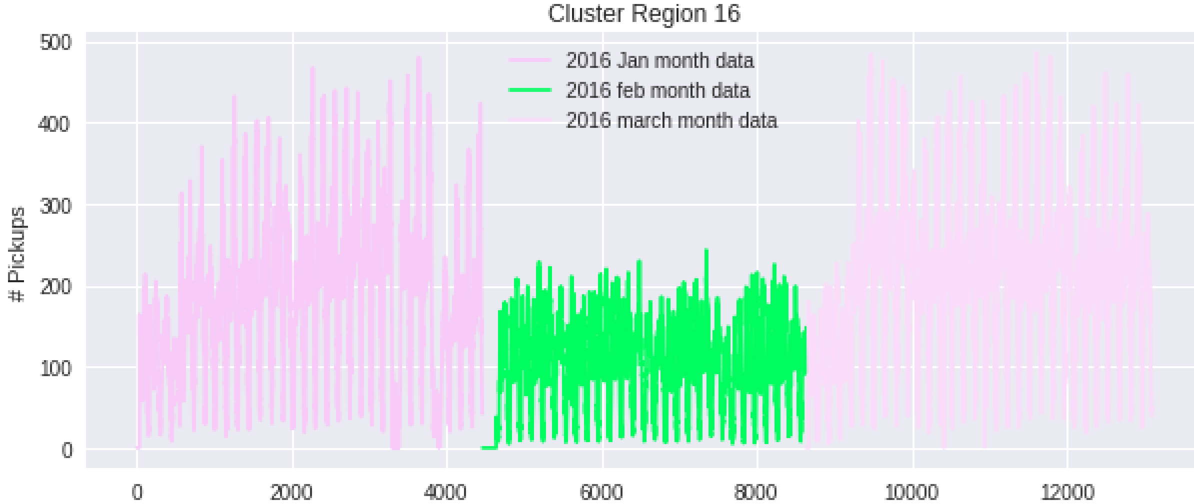

Cluster Region 13
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

Cluster Region 14

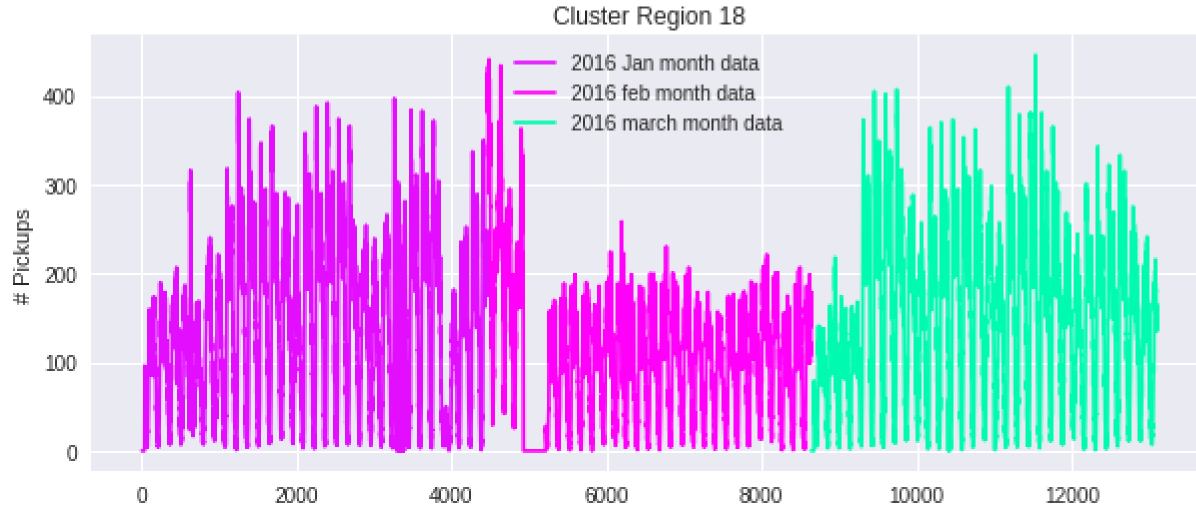
Cluster Region 15

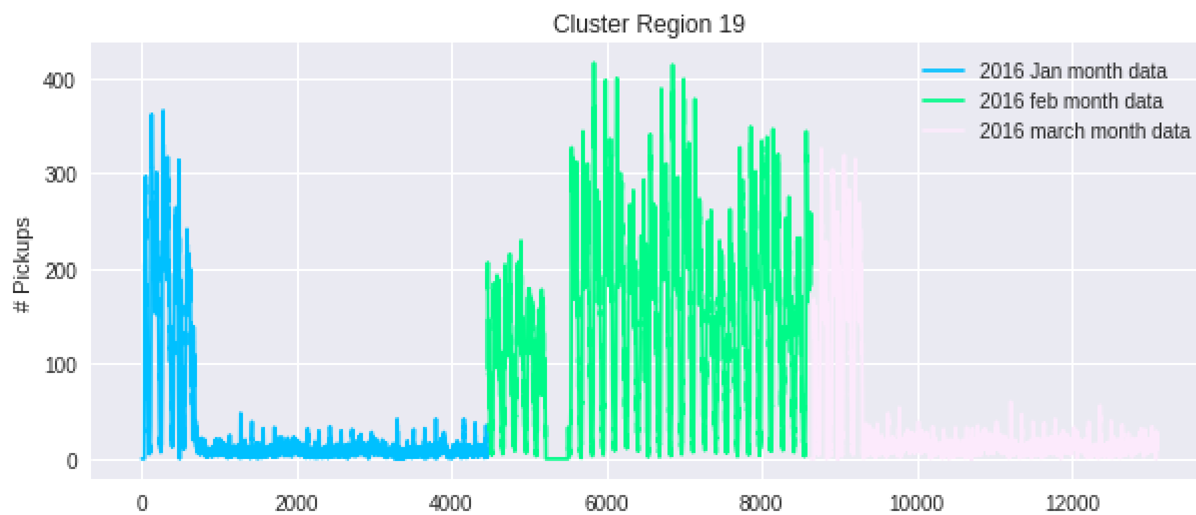


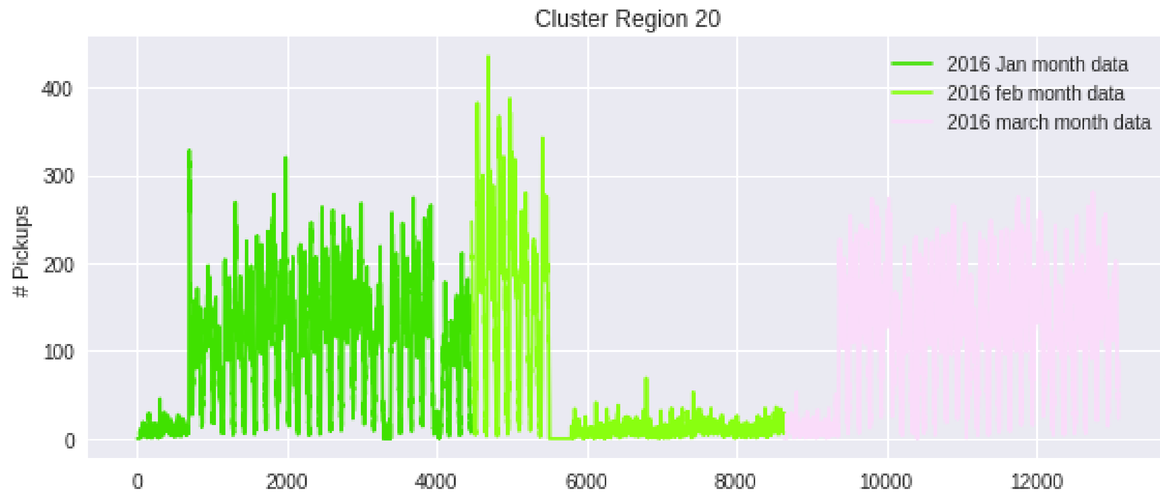
Cluster Region 16

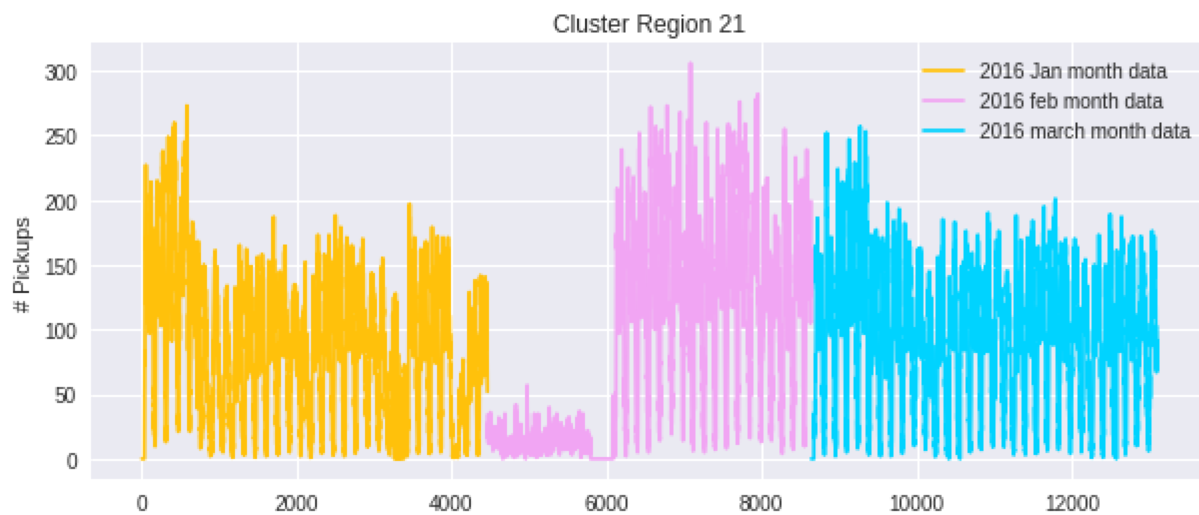


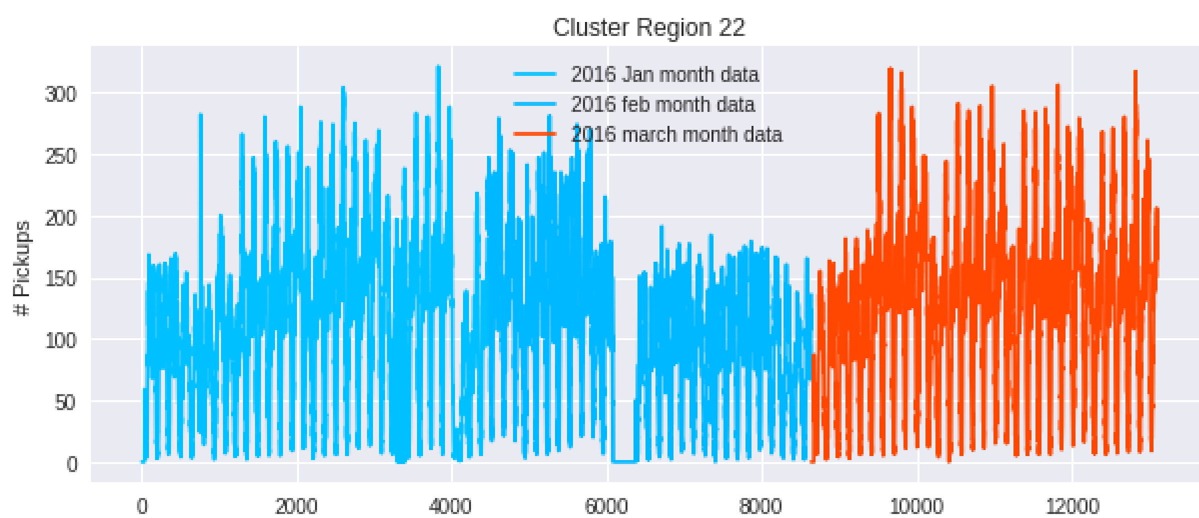

Cluster Region 17
500
2016 Jan month data

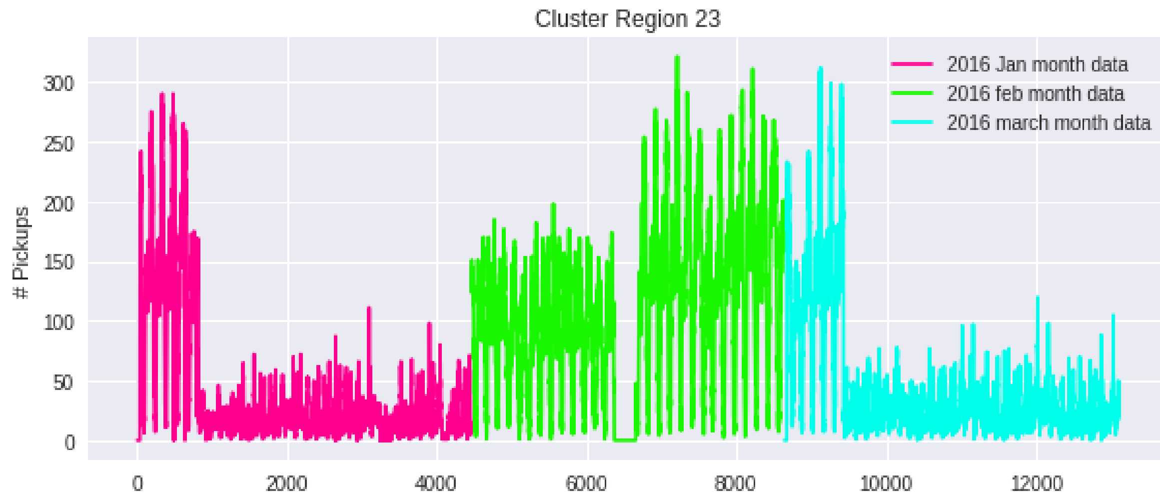

Cluster Region 18
Pickups
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data

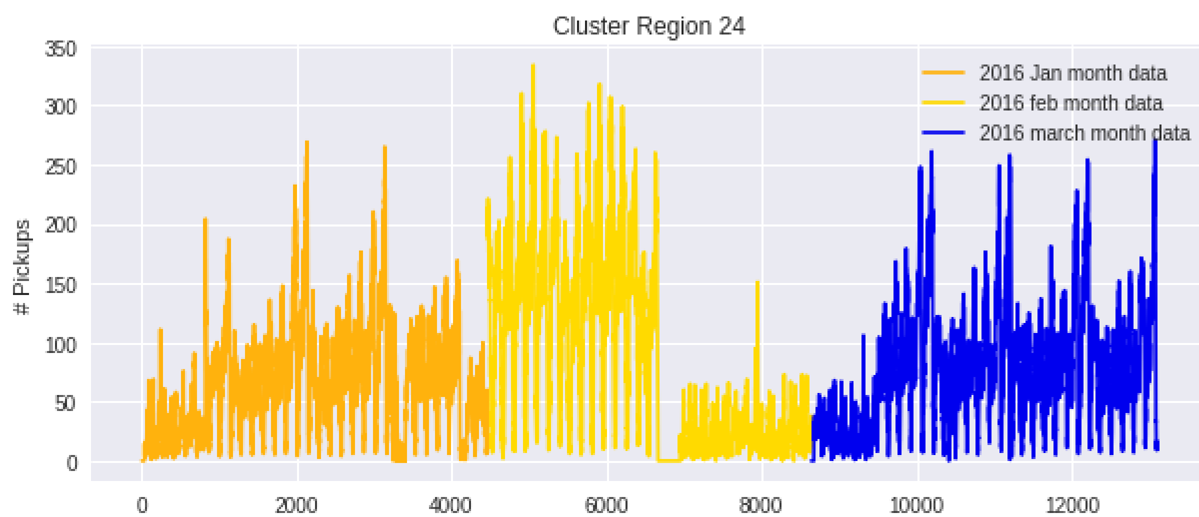

Cluster Region 19
Pickups
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data


Cluster Region 20
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000


Cluster Region 21
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

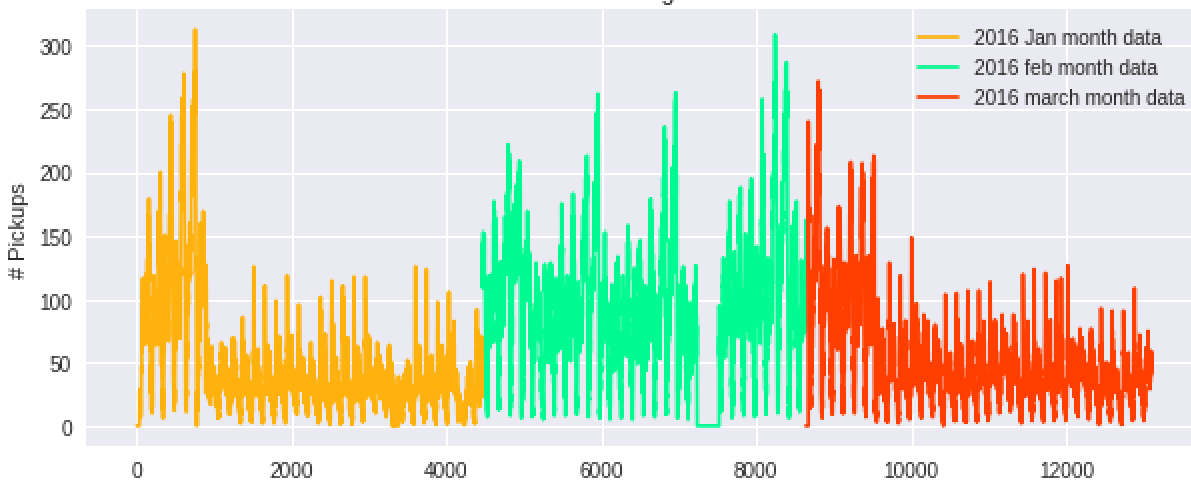

Cluster Region 22
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000


Cluster Region 23
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000

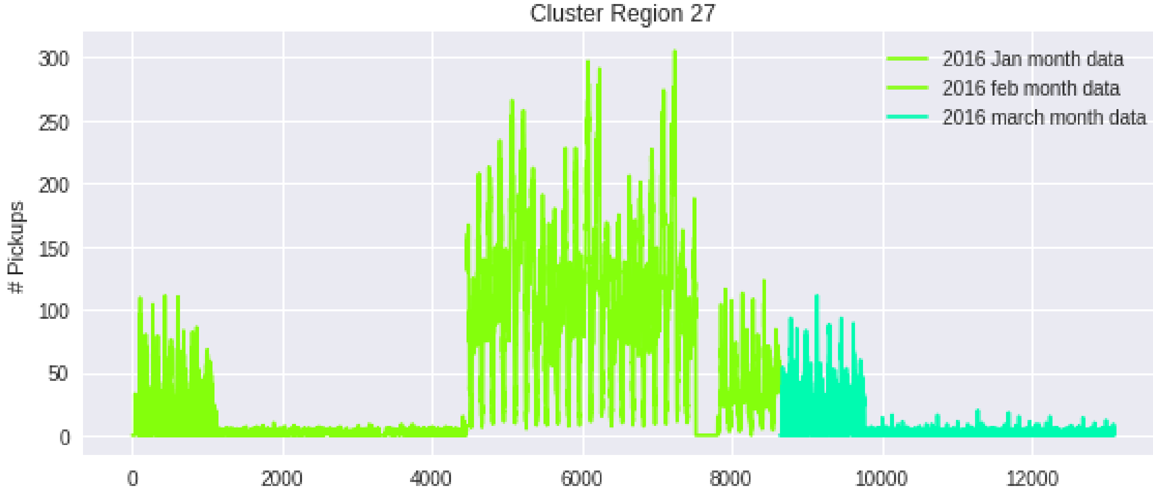

Cluster Region 24
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
0
50
100
150
200
250
300
350
0
2000
4000
6000
8000
10000
12000

Cluster Region 25

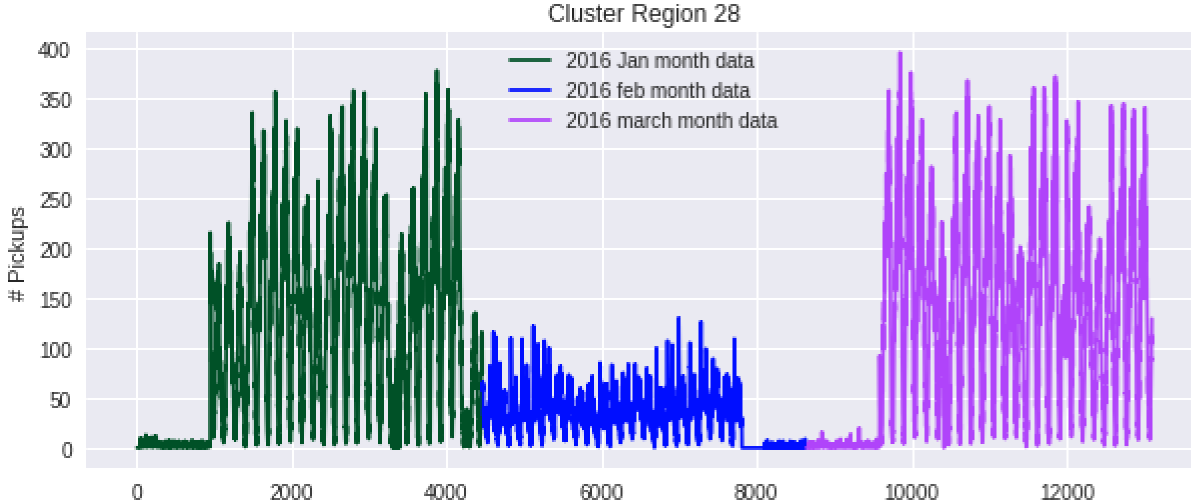
Cluster Region 26

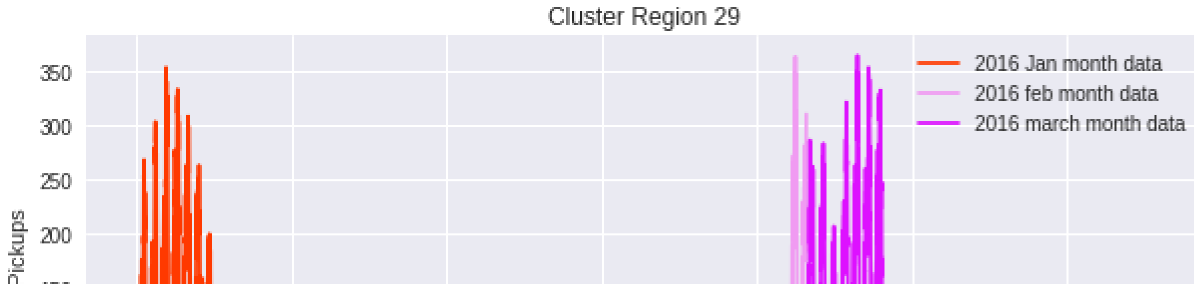


Cluster Region 27

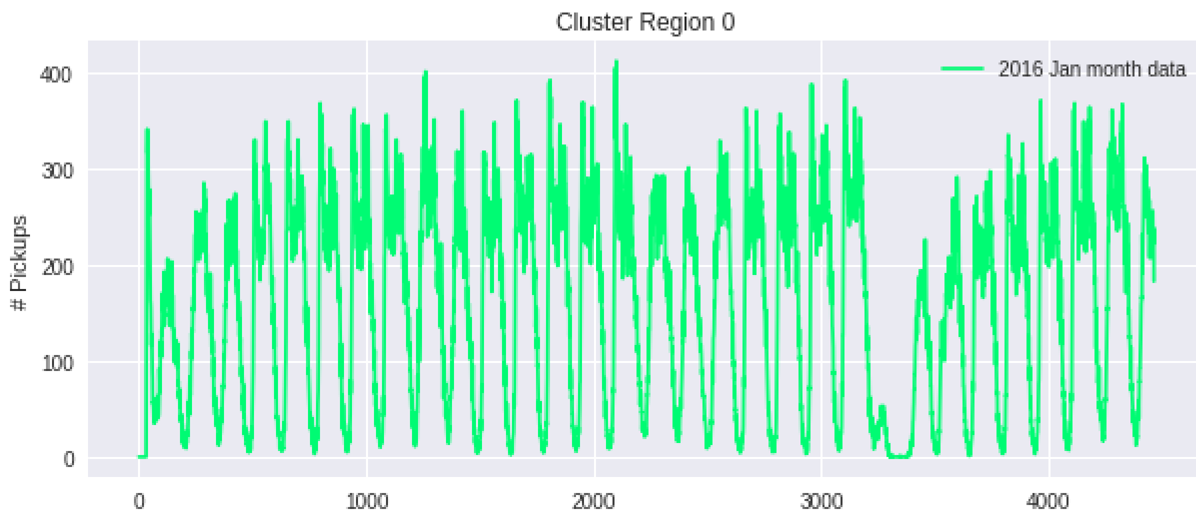


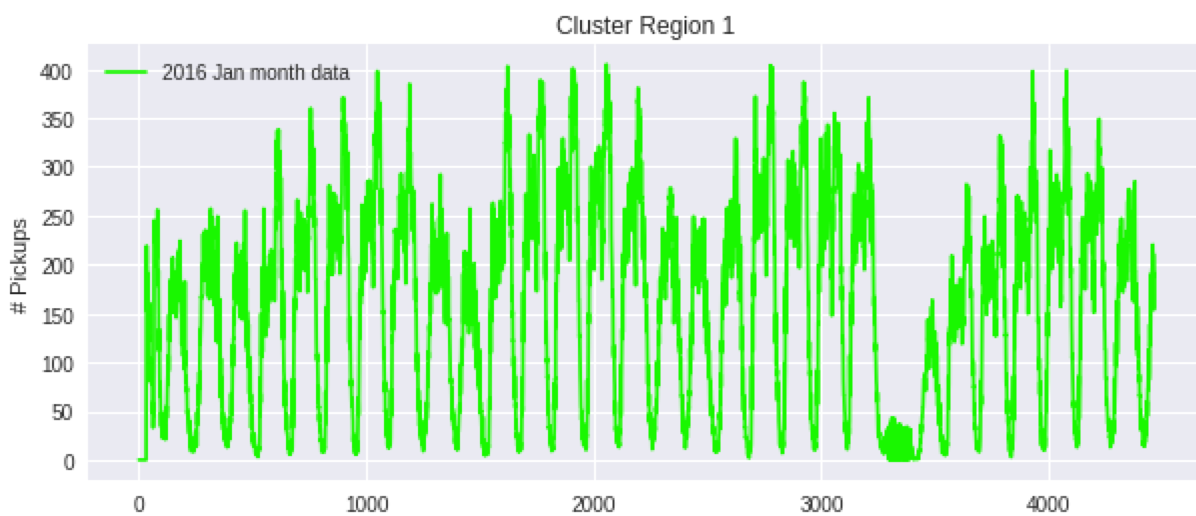
Cluster Region 28

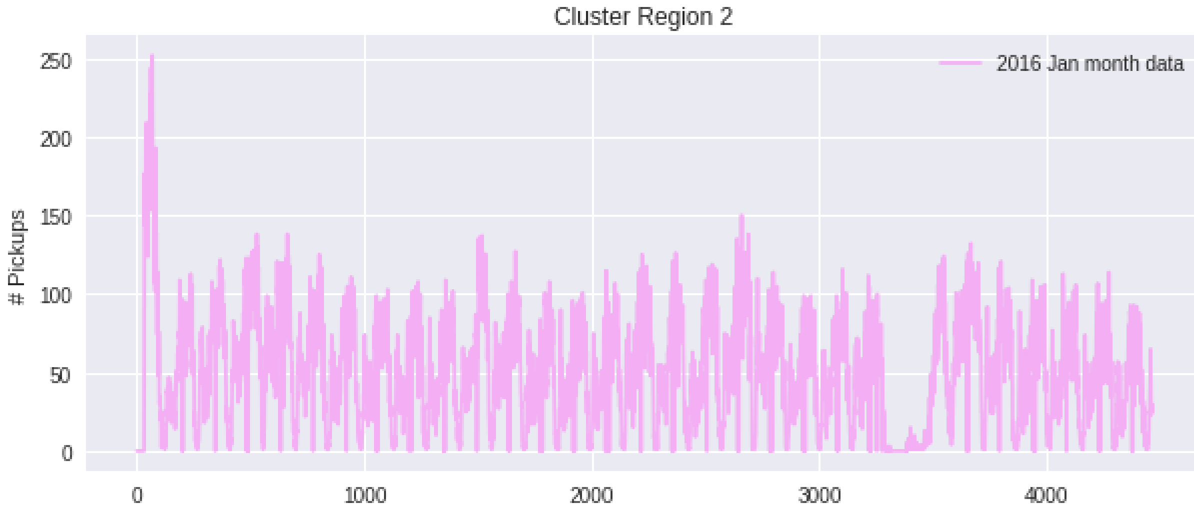


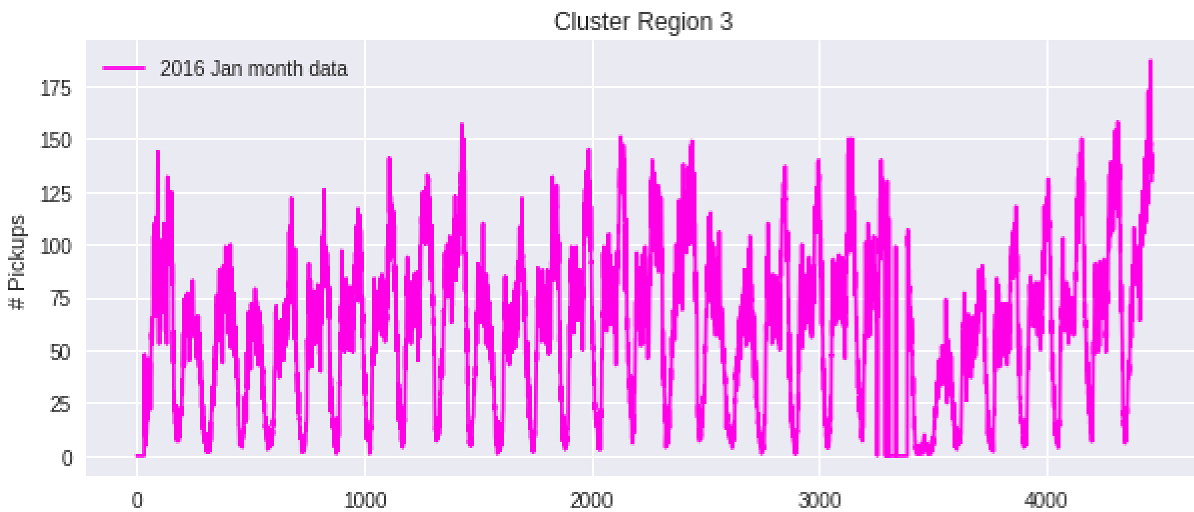

Cluster Region 29
350
300
250
200
Pickups
2016 Jan month data
2016 feb month data
2016 march month data

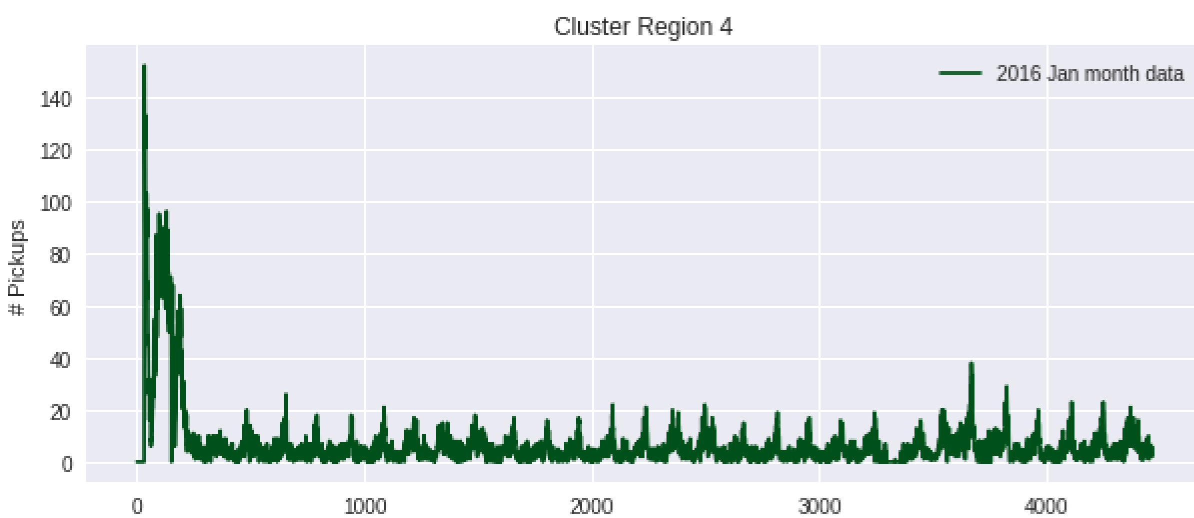
In [85]:
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(20):
    plt.figure(figsize=(10,4))
    plt.title("Cluster Region "+str(i))
    plt.ylabel("# Pickups")
    plt.plot(first_x,smooth16[i][:4464], color=uniqueish_color(), label='2016 Jan
    plt.legend()
    plt.show()
```

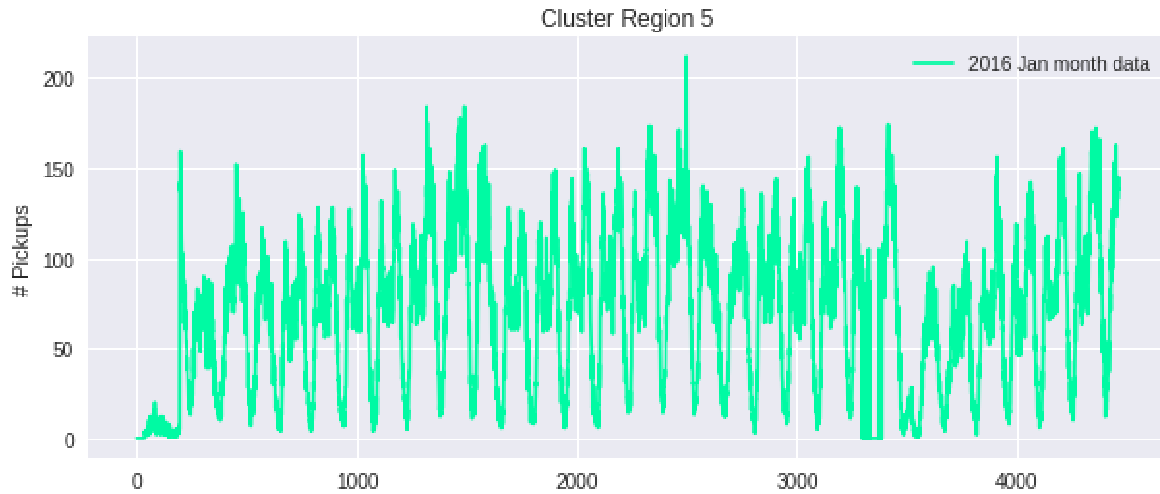


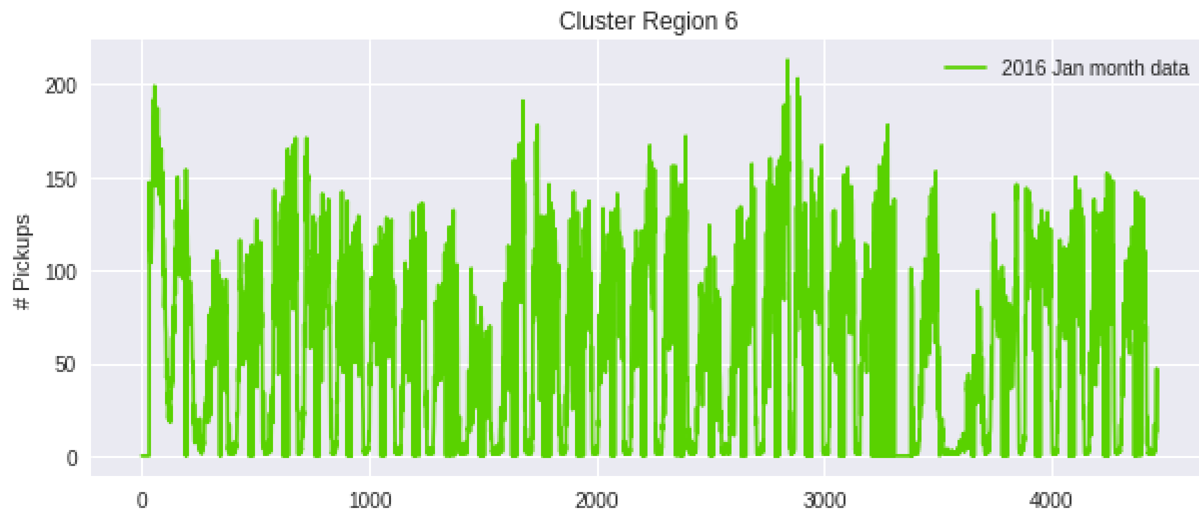


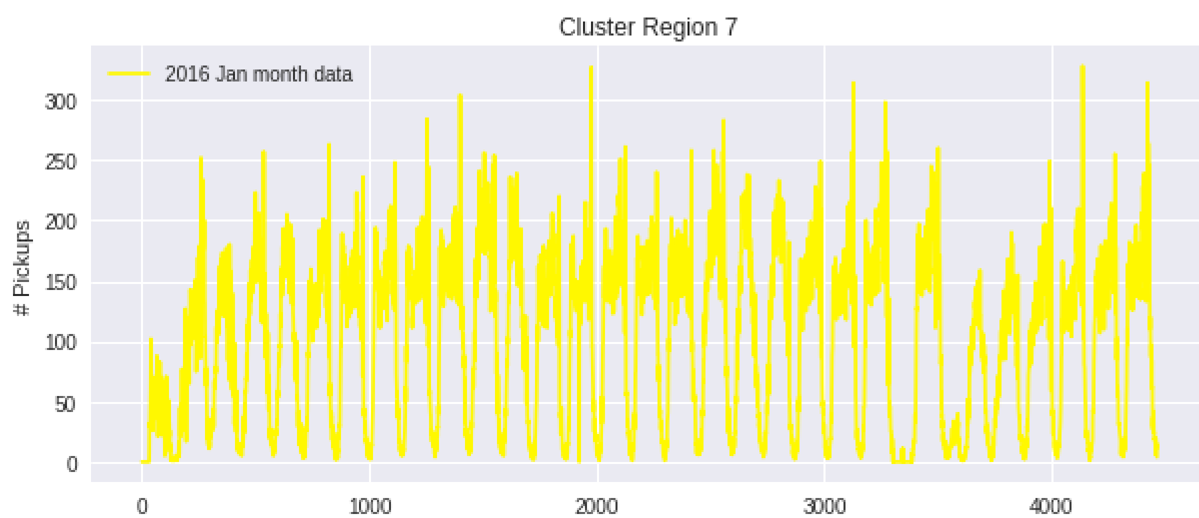

Cluster Region 2
2016 Jan month data
Pickups
0
50
100
150
200
250
0
1000
2000
3000
4000

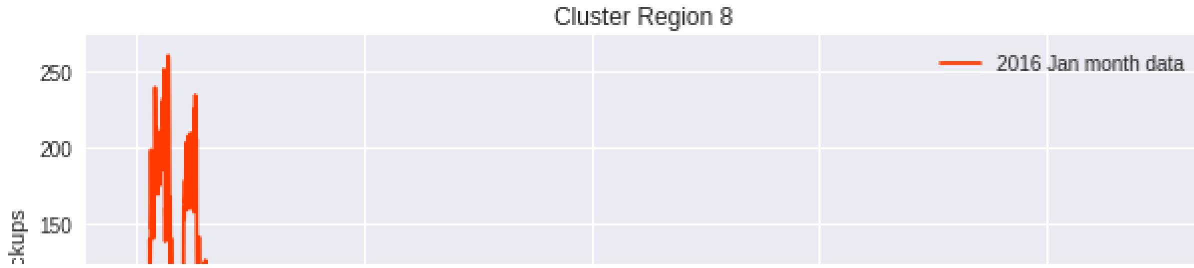

Cluster Region 3
2016 Jan month data
Pickups
0
25
50
75
100
125
150
175
0
1000
2000
3000
4000

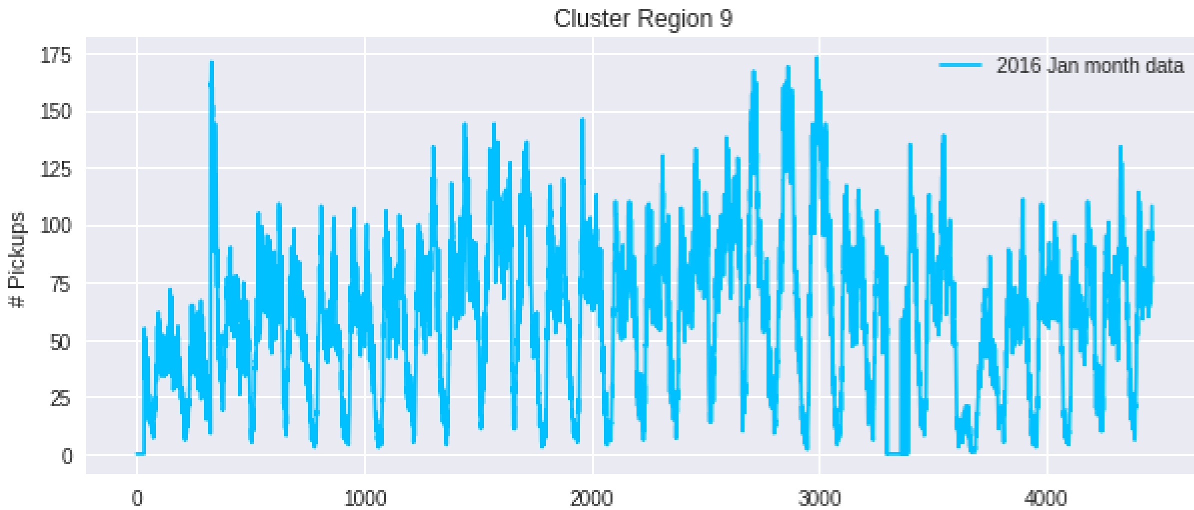

Cluster Region 4
2016 Jan month data
Pickups
0
20
40
60
80
100
120
140
0
1000
2000
3000
4000

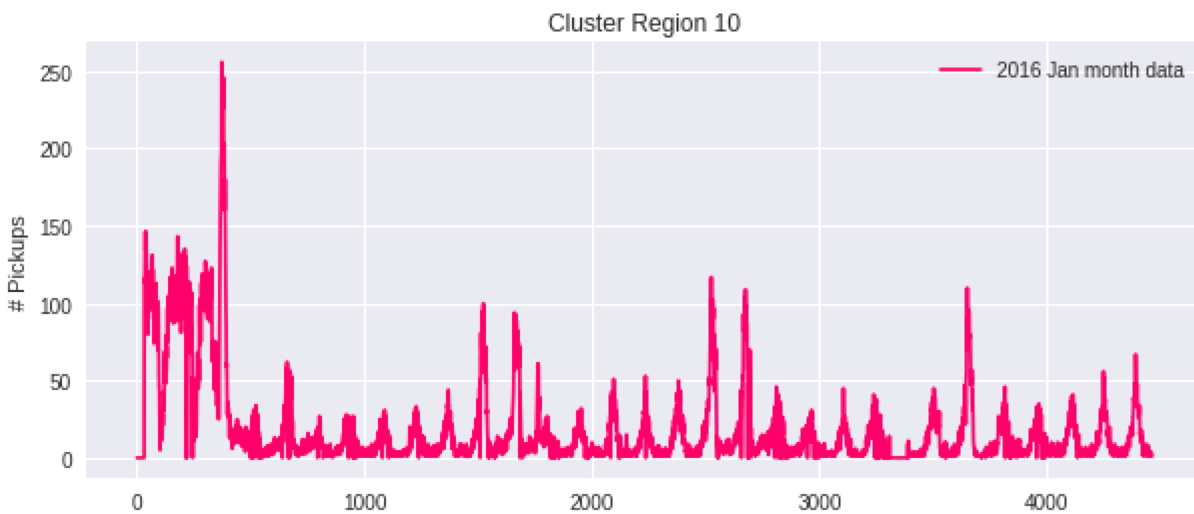

Cluster Region 5
2016 Jan month data
Pickups
0
50
100
150
200
0
1000
2000
3000
4000

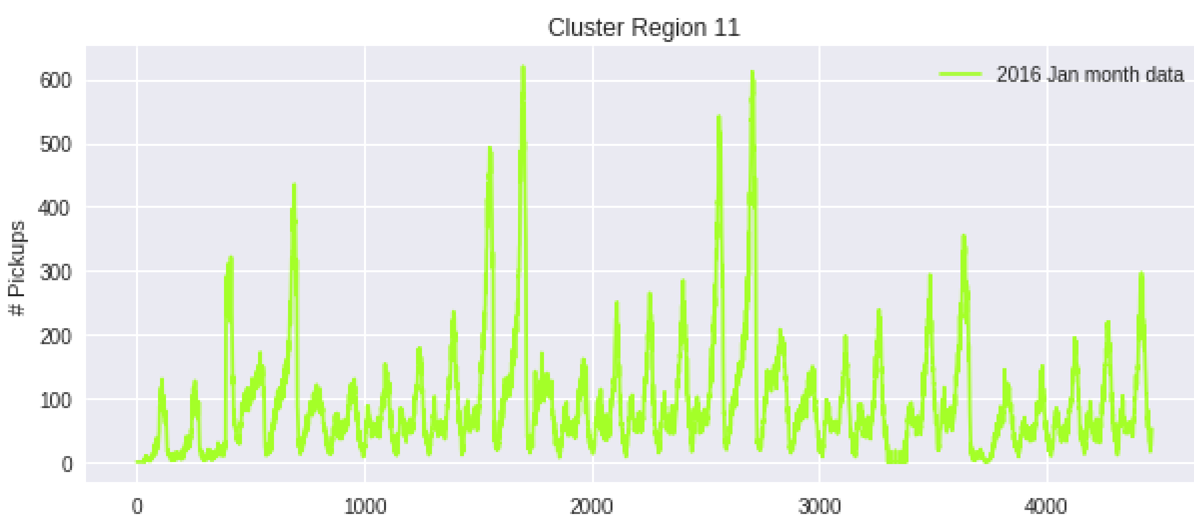

Cluster Region 6
2016 Jan month data
Pickups
0
50
100
150
200
0
1000
2000
3000
4000

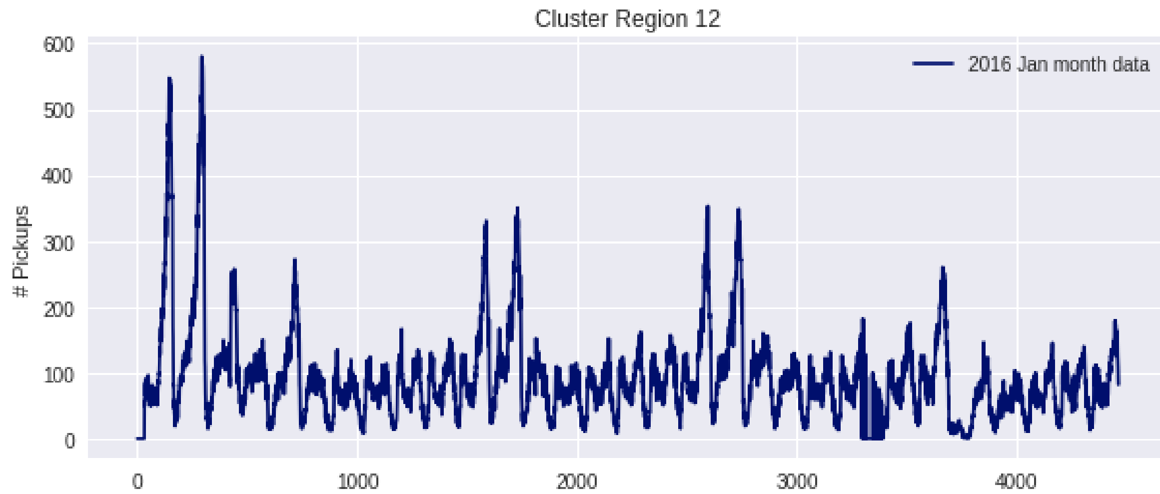

Cluster Region 7
2016 Jan month data
Pickups
0
50
100
150
200
250
300
0
1000
2000
3000
4000


Cluster Region 8
2016 Jan month data
250
200
150

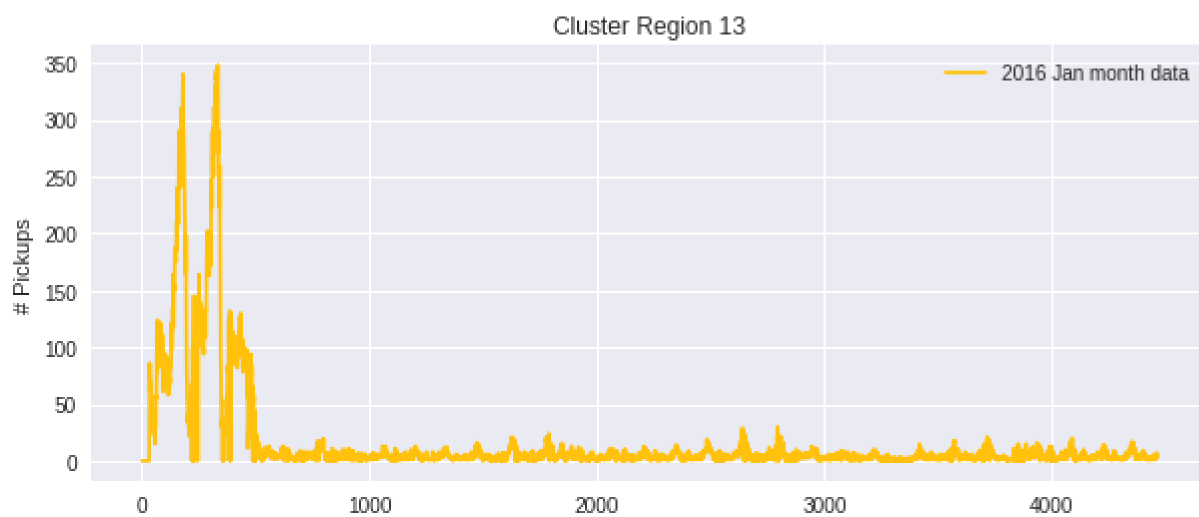

Cluster Region 9
2016 Jan month data
Pickups
175
150
125
100
75
50
25
0
0
1000
2000
3000
4000


Cluster Region 10
2016 Jan month data
Pickups
250
200
150
100
50
0
0
1000
2000
3000
4000

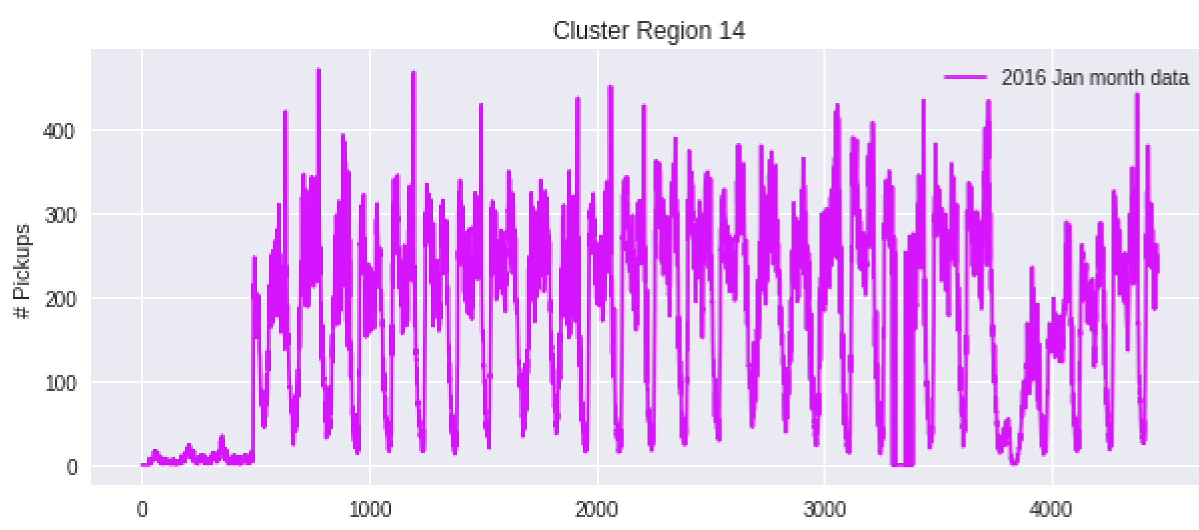

Cluster Region 11
2016 Jan month data
Pickups
600
500
400
300
200
100
0
0
1000
2000
3000
4000



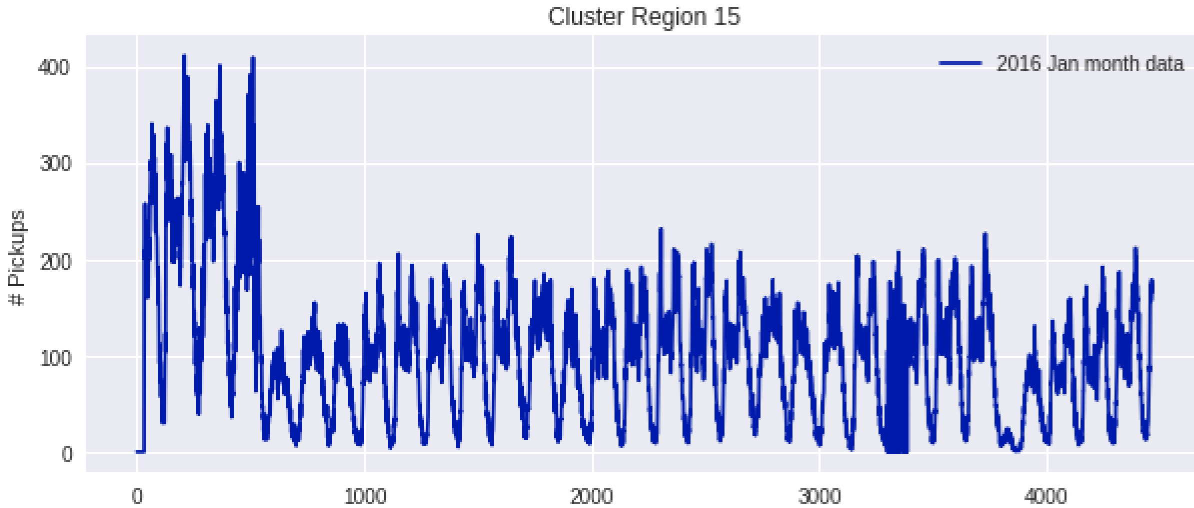
Cluster Region 12
2016 Jan month data
Pickups
600
500
400
300
200
100
0
0
1000
2000
3000
4000

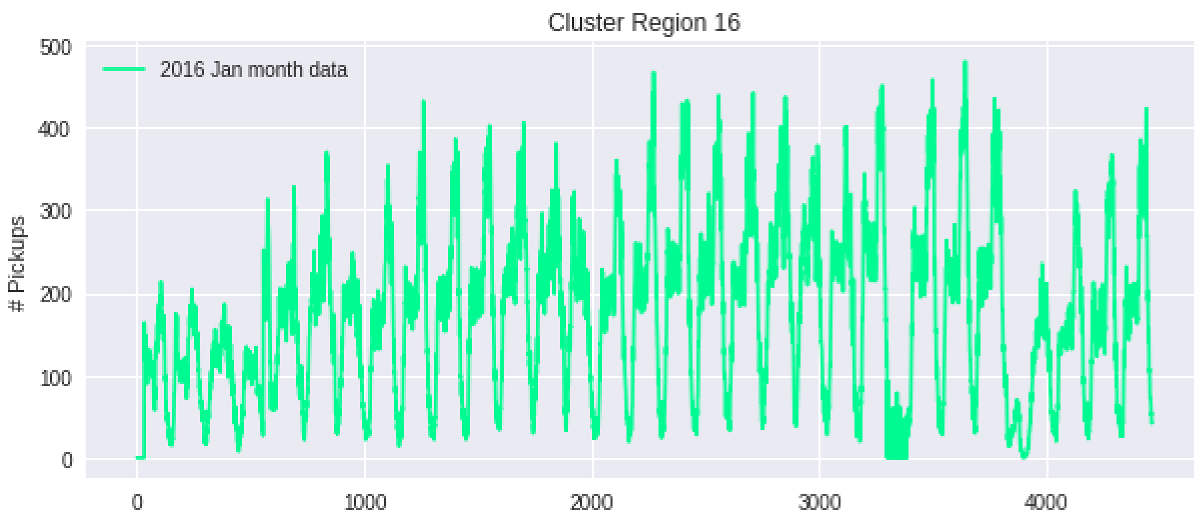


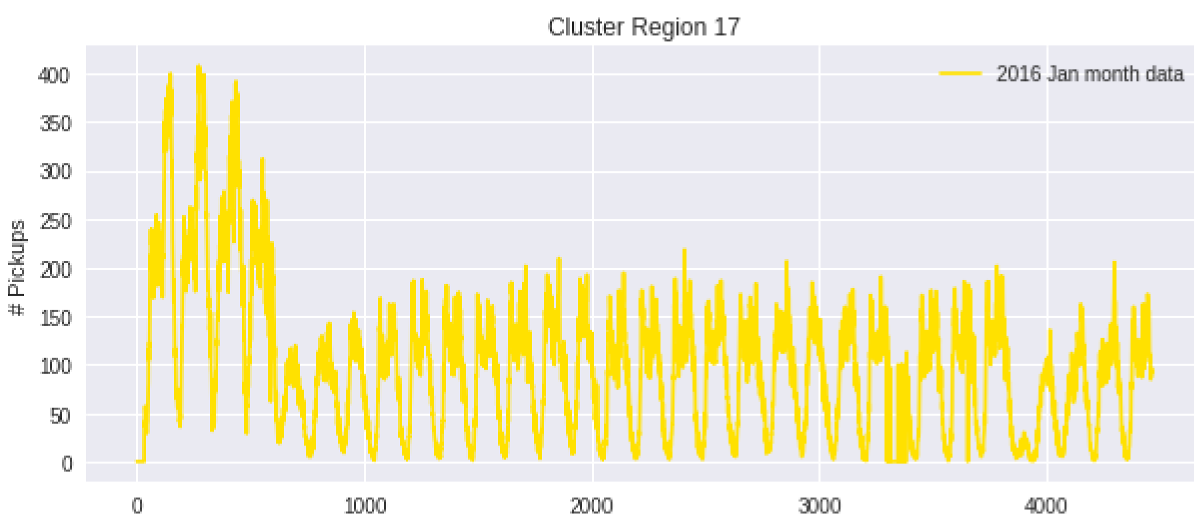
Cluster Region 13
2016 Jan month data
Pickups
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000

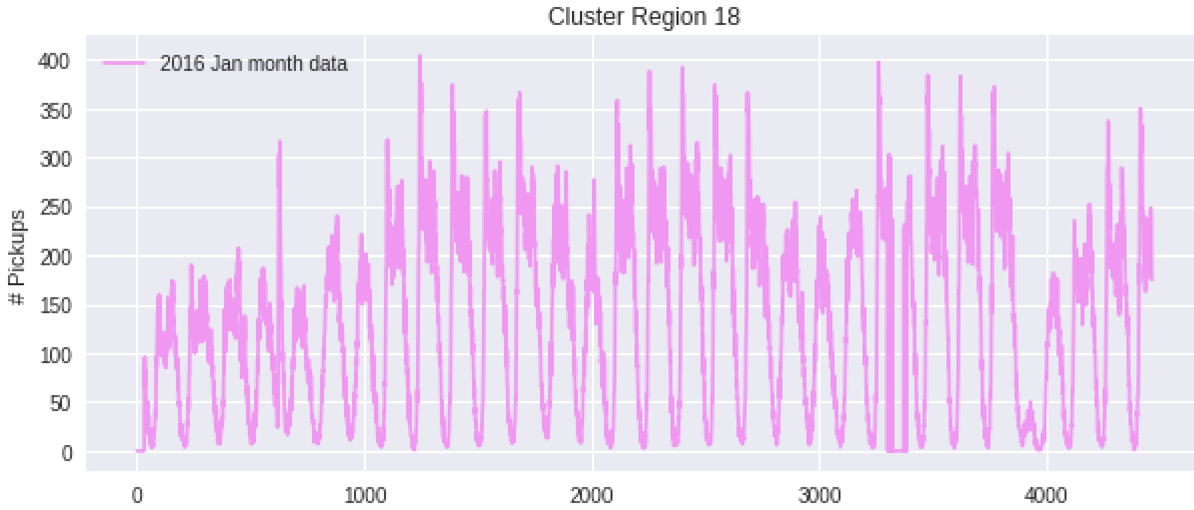


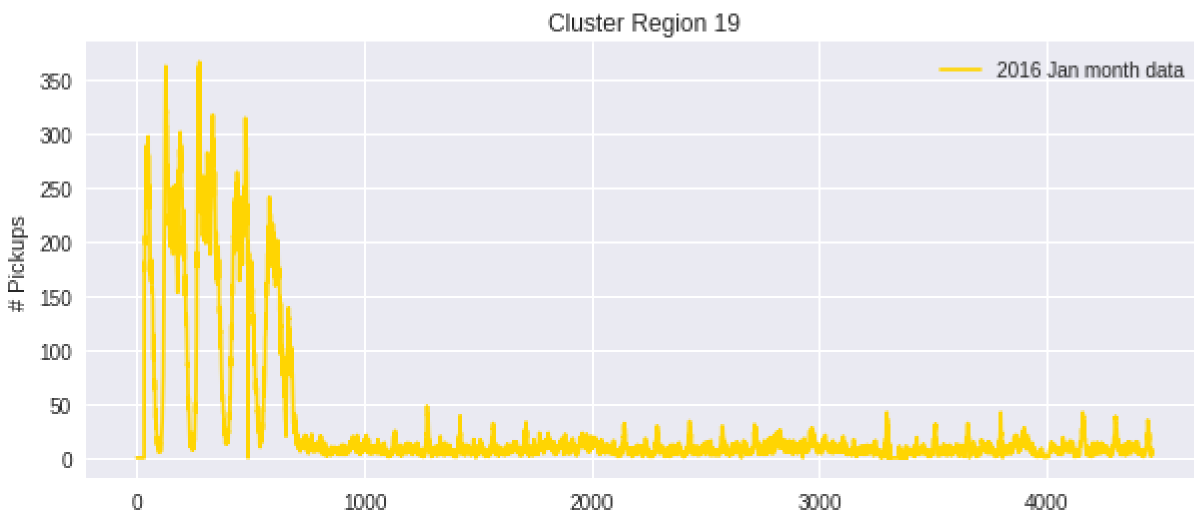
Cluster Region 14
2016 Jan month data
Pickups
400
300
200
100
0
0
1000
2000
3000
4000


Cluster Region 15
2016 Jan month data
Pickups
0
100
200
300
400
0
1000
2000
3000
4000

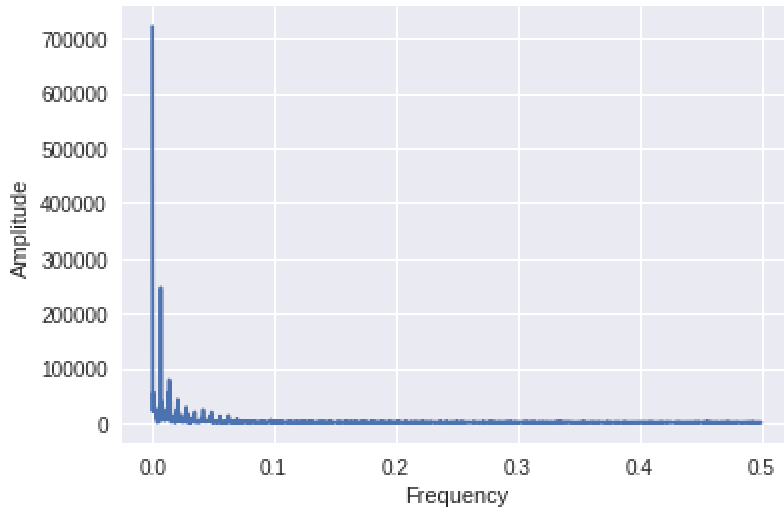

Cluster Region 16
2016 Jan month data
Pickups
0
100
200
300
400
500
0
1000
2000
3000
4000


Cluster Region 17
2016 Jan month data
Pickups
0
50
100
150
200
250
300
350
400
0
1000
2000
3000
4000

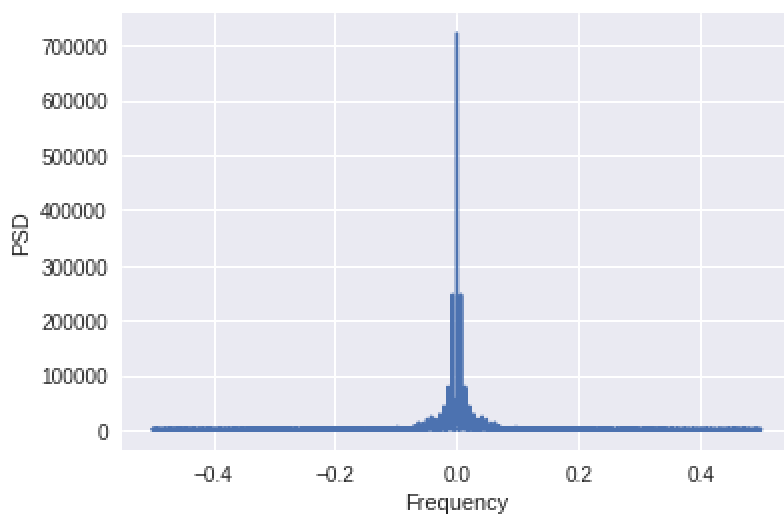

Cluster Region 18
2016 Jan month data
Pickups
400
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000


Cluster Region 19
2016 Jan month data
Pickups
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000

In [86]:
```
# getting peaks: https://blog.ytotech.com/2015/11/01/findpeaks-in-python/
# read more about fft function : https://docs.scipy.org/doc/numpy/reference/genera
Y = np.fft.fft(np.array(jan_2016_smooth)[0:4460])
# read more about the fftfreq: https://docs.scipy.org/doc/numpy/reference/generat
freq = np.fft.fftfreq(4460, 1)
n = len(freq)
plt.figure()
plt.plot( freq[:int(n/2)], np.abs(Y)[:int(n/2)] )
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.show()
```



In [87]:
```
# ploting by taking PSD = absolute( complex valued amplitude)
plt.figure()
plt.plot( freq, np.abs(Y) )
plt.xlabel("Frequency")
plt.ylabel("PSD")
plt.show()
```



Observation

1. In above Amplitude/frequency plot the first peak at index 0, is the DC component,DC component just means the average of positive and negative half cycles is not zero.
2. D is the DC component. It shifts the function up or down the y-axis. Note that it is independant of the function variable t. we will not consider it's amplitude and frequency. We will start taking frequency and amplitudes from the second peak onwards.
3. In Second plot: A[0] contains the zero-frequency term (the sum of the signal), which is always purely real for real inputs. A[1:n/2] contains the positive-frequency terms A[n/2+1:] contains the negative-frequency terms

In [0]:

```
def process_freq(freq,Y1):
    '''The Amplitude spectrum in frequency domian is a complex space
       so take absolute values of amplitude i.e PSD.

       The amplitude values are symmetric with y axis acting as the mirror so hal
       frequency space is sufficient to record all the frequency peaks'''
    n = len(freq) # x is freq

    f = np.abs(freq)[:int(n/2)]
    a = np.abs(Y1)[:int(n/2)]

    return f,a
```

In [98]:

```
!pip install peakutils
import peakutils
def gets_peaks(amp_val1,t):
    '''returns incices of the peaks'''
    indices = peakutils.indexes(amp_val1, thres=t, min_dist=1,thres_abs=True)
    return indices
```

```
Collecting peakutils
  Downloading https://files.pythonhosted.org/packages/2a/e0/a45948450946a87dae4
4d936ea7646d862e1014753c496468a05f20e95c5/PeakUtils-1.3.2.tar.gz (https://file
s.pythonhosted.org/packages/2a/e0/a45948450946a87dae44d936ea7646d862e1014753c49
6468a05f20e95c5/PeakUtils-1.3.2.tar.gz)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
 (from peakutils) (1.14.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
 (from peakutils) (1.1.0)
Building wheels for collected packages: peakutils
  Building wheel for peakutils (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/6d/52/9c/94cff100c9dd4ec0c727629
47b8d5da6f6c0762cd5312b04ec
Successfully built peakutils
Installing collected packages: peakutils
Successfully installed peakutils-1.3.2
```

In [0]:
```
def freqT(month_all):
    '''Discrete frequency transformation using fast fourier tranform'''
    '''Each cluster is transformed and processed separatly'''
    '''Returns top 5 amp and corresponding freq values for each cluster'''
    psd_y = []
    freq_x = []
    for clust_i in range(30):
        amp = np.fft.fft(month_all[i][:]) # returns complex values
        f = np.fft.fftfreq(1304,1)

        fre,ampli = process_freq(f,amp)

        t1=10000 # peak threshold
        peak_index = gets_peaks(ampli,t1)

        # sorting decending order , returns indices
        sorted_index = np.argsort(-(ampli[peak_index]))
        top5 = sorted_index[0:5]

        top5_amp = list(ampli[top5])
        top5_freq = list(fre[top5])

        psd_y.append(top5_amp)
        freq_x.append(top5_freq)
    return psd_y,freq_x
```

In [0]:
```
# 'psds' and 'frequencies' top 5 peak PSD values
# contains 30 lists corresponding to each cluster for 1st 3 months of 2016 data
# each of the 30 list is of size 5

psds,frequencies = freqT(smooth16)
```

In [101]:
```
print('number of clusters',len(psds))
print('num of top values',len(psds[0]))
```

```
number of clusters 30
num of top values 5
```

In [0]:
```
# Preparing data to be split into train and test, The below prepares data in cumu
# number of 10min indices for jan 2015= 24*31*60/10 = 4464
# number of 10min indices for jan 2016 = 24*31*60/10 = 4464
# number of 10min indices for feb 2016 = 24*29*60/10 = 4176
# number of 10min indices for march 2016 = 24*31*60/10 = 4464
# regions_cum: it will contain 20 lists, each list will contain 4464+4176+4464 va
# that are happened for three months in 2016 data


previous_bins = 5 # number of previous 10min intravels to consider

##################################################################################
# The following variables will be used to store 30 lists
# each internal list will store 13104-5= 13099 values
# Ex: [[cluster0 13099times],[cluster1 13099times], [cluster2 13099times].... 30
##################################################################################

output = [] # to store number of pickups 13104-5 = 13099 for each cluster

lat = [] # stores 13099 lattitude values for every cluster

lon = [] # stores 13099 longitude values for every cluster

weekday = [] # stores day coded as sun= 0, mon=1, tue= 2, wed=3, thur=4, fri=5, s

##################################################################################



# its an numpy array, of shape (523960, 5)
# each row corresponds to an entry in out data
# for the first row we will have [f0,f1,f2,f3,f4] fi=number of pickups happened i
# the second row will have [f1,f2,f3,f4,f5]
# and so on...
ts_feature = [0]*previous_bins


for i in range(0,30):
    lat.append([kmeans.cluster_centers_[i][0]]*13099)
    lon.append([kmeans.cluster_centers_[i][1]]*13099)

    # jan 1st 2016 is Friday, so we start our day from 5: "(int(k/144))%7+5"
    # prediction start from 5th bin using previous 5 bins

    weekday.append([(((k//144)%7)+5)%7 for k in range(5,4464+4176+4464)])

    # smooth16 is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 30
    ts_feature = np.vstack((ts_feature, [smooth16[i][r:r+previous_bins]\
                                         for r in range(0,len(smooth16[i])-pr
    output.append(smooth16[i][5:])
ts_feature = ts_feature[1:]
```

In [103]:
```
# sanity check
len(lat[0])*len(lat) == ts_feature.shape[0] == len(weekday)*len(weekday[0])== 30*
```

Out[103]:
```
True
```

In [104]:
```
ts_feature
```

Out[104]:
```
array([[ 0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0],
       ...,
       [14,  9, 13, 21, 18],
       [ 9, 13, 21, 18, 20],
       [13, 21, 18, 20, 22]])
```

In [0]:
```
# Getting the predictions of exponential moving averages to be used as a feature

# upto now we computed 8 features for every data point that starts from 50th min 
# 1. cluster center lattitude
# 2. cluster center longitude
# 3. day of the week
# 4. freq_1: number of pickups that are happened previous t-1th 10min intravel
# 5. freq_2: number of pickups that are happened previous t-2th 10min intravel
# 6. freq_3: number of pickups that are happened previous t-3th 10min intravel
# 7. freq_4: number of pickups that are happened previous t-4th 10min intravel
# 8. freq_5: number of pickups that are happened previous t-5th 10min intravel

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

alpha=0.3

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

# it is similar like lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6
predict_list = []
flat_exp_avg = []
for r in range(0,30):
    for i in range(0,13104):
        if i==0:
            predicted_value= smooth16[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(smooth16[r][i])
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

In [106]:
```
print(len(psds))
print(len(frequencies))
print(len(psds[0]))
```

```
30
30
5
```

Fourier Transform

In [0]:
```
#frequencies and amplitudes are same for all the points a cluster
psd_feat = [0]*30
freq_feat = [0]*30

for cl in range(30):
    p_i = []
    f_i = []

    for k in range(13104):
        p_i.append(psds[cl])
        f_i.append(frequencies[cl])

    psd_feat[cl]=p_i
    freq_feat[cl]=f_i
```

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 [108]:
```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 mo
# and split it such that for every region we have 70% data in train and 30% in te
# ordered date-wise for every region
print("size of train data :", int(13099*0.7))
print("size of test data :", int(13099*0.3))
```

```
size of train data : 9169
size of test data : 3929
```

In [0]:
```
# Extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for
train_features =  [ts_feature[i*13099:(13099*i+9169)] for i in range(0,30)]

test_features = [ts_feature[(13099*(i))+9169:13099*(i+1)] for i in range(0,30)]
```

In [110]:
```
print("Train data # Regions = ",len(train_features), \
      "\nNumber of data points", len(train_features[0]), \
      "\n Each data point contains", len(train_features[0][0]),"features\n")

print("Test data # Regions = ",len(train_features), \
      "\nNumber of data points in test data", len(test_features[0]), \
      "\nEach data point contains", len(test_features[0][0]),"features")
```

```
Train data # Regions =  30
Number of data points 9169
 Each data point contains 5 features

Test data # Regions =  30
Number of data points in test data 3930
Each data point contains 5 features
```

In [0]:
```
# the above contains values in the form of list of lists (i.e. list of values of 
# here we make all of them in one list
train_new_features = []
for i in range(0,30):
    train_new_features.extend(train_features[i])

test_new_features = []
for i in range(0,30):
    test_new_features.extend(test_features[i])
```

In [113]:
```
len(train_new_features)
```

Out[113]:
```
275070
```

In [0]:
```
train_fourier_psd = [psd_feat[i][5:9169+5] for i in range(30)]
test_fourier_psd = [psd_feat[i][9169+5:] for i in range(30)]
train_fourier_freq = [freq_feat[i][5:9169+5] for i in range(30)]
test_fourier_freq = [freq_feat[i][9169+5:] for i in range(30)]
```

In [0]:
```
# converting lists of lists into single list i.e flatten
train_psds = sum(train_fourier_psd, [])
test_psds  = sum(test_fourier_psd, [])

train_freqs = sum(train_fourier_freq, [])
test_freqs   = sum(test_fourier_freq, [])
```

In [0]:
```
train_f_lat = [i[:9169] for i in lat]
train_f_lon = [i[:9169] for i in lon]
train_f_weekday = [i[:9169] for i in weekday]
train_f_output = [i[:9169] for i in output]
train_f_exp_avg = [i[:9169] for i in predict_list]
```

In [0]:
```
# 3930 points to test
test_f_lat = [i[9169:] for i in lat]
test_f_lon = [i[9169:] for i in lon]
test_f_weekday = [i[9169:] for i in weekday]
test_f_output = [i[9169:] for i in output]
test_f_exp_avg = [i[9169:] for i in predict_list]
```

In [0]:
```
# 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_lat = sum(train_f_lat, [])
train_lon = sum(train_f_lon, [])
train_weekday = sum(train_f_weekday, [])
train_output = sum(train_f_output, [])
train_exp_avg = sum(train_f_exp_avg,[])
```

In [0]:
```
# converting lists of lists into sinle list i.e flatten
test_lat = sum(test_f_lat, [])
test_lon = sum(test_f_lon, [])
test_weekday = sum(test_f_weekday, [])
test_output = sum(test_f_output, [])
test_exp_avg = sum(test_f_exp_avg, [])
```

In [0]:
```
train_FT = np.hstack((train_new_features, train_psds, train_freqs))
test_FT = np.hstack((test_new_features, test_psds,test_freqs))
```

In [123]:
```
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1','a1','a2','a3','a4','a5',\
           'f1','f2','f3','f4','f5']

df_train = pd.DataFrame(data=train_FT, columns=columns)
df_train['lat'] = train_lat
df_train['lon'] = train_lon
df_train['weekday'] = train_weekday
df_train['exp_avg'] = train_exp_avg


print(df_train.shape)
```

```
(275070, 19)
```

In [124]:
```
df_test = pd.DataFrame(data=test_FT, columns=columns)
df_test['lat'] = test_lat
df_test['lon'] = test_lon
df_test['weekday'] = test_weekday
df_test['exp_avg'] = test_exp_avg
print(df_test.shape)
```

```
(117900, 19)
```

In [125]:
```
# final test dataframe
df_test.head()
```

Out[125]:

	ft_5	ft_4	ft_3	ft_2	ft_1	a1	a2	a3	a4	
0	271.0	270.0	238.0	269.0	260.0	22790.263173	329663.192557	831171.0	396741.604335	25437.
1	270.0	238.0	269.0	260.0	281.0	22790.263173	329663.192557	831171.0	396741.604335	25437.
2	238.0	269.0	260.0	281.0	264.0	22790.263173	329663.192557	831171.0	396741.604335	25437.
3	269.0	260.0	281.0	264.0	286.0	22790.263173	329663.192557	831171.0	396741.604335	25437.
4	260.0	281.0	264.0	286.0	280.0	22790.263173	329663.192557	831171.0	396741.604335	25437.

In [0]:
```
import pickle
pickle_out = open("df_train","wb")
pickle.dump(df_train, pickle_out)
pickle_out.close()

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

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

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

In [5]:
```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving df_test to df_test
Saving df_train to df_train
Saving tsne_test_output to tsne_test_output
Saving tsne_train_output to tsne_train_output
```

In [0]:
```
import pickle
pickle_in = open("df_train","rb")
df_train = pickle.load(pickle_in)
pickle_in.close()

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

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

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

Models

Using Linear Regression

In [0]:
```
def LR_reg(df_train,df_test, train_output):
    s = StandardScaler()
    df_train1 = s.fit_transform(df_train)
    df_test1 = s.transform(df_test)

    LR = SGDRegressor(loss="squared_loss")


    alp =  [0.00001,0.000001,0.000002,0.000005]
    ite = [400,500,600]

    c_param = {"alpha": alp, "max_iter":ite}
    opti_model = GridSearchCV(LR, param_grid= c_param, scoring = "neg_mean_absolu

    opti_model.fit(df_train1, train_output)

    y_pred =  opti_model.best_estimator_.predict(df_train1)
    lr_train_predictions = [round(value) for value in y_pred]

    y_pred = opti_model.best_estimator_.predict(df_test1)
    lr_test_predictions = [round(value) for value in y_pred]

    print(opti_model.best_params_)
    return lr_train_predictions, lr_test_predictions
```

In [137]:

```
lr_train_predictions,lr_test_predictions = LR_reg(df_train,df_test, train_output)
```

```
{'alpha': 1e-05, 'max_iter': 600}
```

In [141]:

```
# Calculating the error metric values
train_mse_sgd = mean_squared_error(train_output,lr_train_predictions)
train_mpe_sgd = mean_absolute_error(train_output,lr_train_predictions)/(sum(train_
test_mse_sgd = mean_squared_error(test_output,lr_test_predictions)
test_mpe_sgd = mean_absolute_error(test_output,lr_test_predictions)/(sum(test_out

print(train_mpe_sgd*100)
print(test_mpe_sgd*100)
```

```
12.516377357076564
11.902378748321238
```

Using Random Forest Regressor

In [0]:

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

def RF_reg(df_train,df_test,train_output):
    n_est = sp_randint(400,600)
    max_dep = sp_randint(10, 20)
    min_split = sp_randint(8, 15)
    start = [False]
    min_leaf = sp_randint(8, 15)
    c_param = {'n_estimators':n_est ,'max_depth': max_dep,'min_samples_split':min_
               'min_samples_leaf':min_leaf ,'warm_start':start }

    RF_reg = RandomForestRegressor(max_features='sqrt', n_jobs=4)

    model2 =  RandomizedSearchCV(RF_reg, param_distributions= c_param, scoring =

    model2.fit(df_train, train_output)

    y_pred = model2.best_estimator_.predict(df_test)
    rndf_test_predictions = [round(value) for value in y_pred]
    y_pred = model2.best_estimator_.predict(df_train)
    rndf_train_predictions = [round(value) for value in y_pred]
    print(model2.best_params_)
    return rndf_train_predictions,rndf_test_predictions
```

In [145]:

```
# Predicting on test data using our trained random forest model
rndf_train_predictions,rndf_test_predictions = RF_reg(df_train,df_test,train_outp
```

```
{'max_depth': 14, 'min_samples_leaf': 8, 'min_samples_split': 9, 'n_estimator
s': 511, 'warm_start': False}
```

In [147]:
```
# Calculating the error metric values
train_mse_rf = mean_squared_error(train_output,rndf_train_predictions)
train_mpe_rf = mean_absolute_error(train_output,rndf_train_predictions)/(sum(trai
test_mse_rf = mean_squared_error(test_output,rndf_test_predictions)
test_mpe_rf = mean_absolute_error(test_output,rndf_test_predictions)/(sum(test_ou

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

```
11.451104543509478
11.637367792191494
```

Using XgBoost Regressor

In [0]:
```
from scipy import stats
def xg_reg(df_train,df_test,train_output):

    c_param={'learning_rate' :stats.uniform(0.01,0.2),
      'n_estimators':sp_randint(100,1000),
      'max_depth':sp_randint(1,10),
      'min_child_weight':sp_randint(1,8),
      'gamma':stats.uniform(0,0.02),
      'subsample':stats.uniform(0.6,0.4),
      'reg_alpha':sp_randint(0,200),
      'reg_lambda':stats.uniform(0,200),
      'colsample_bytree':stats.uniform(0.6,0.3)}

    xreg= xgb.XGBRegressor(nthread = 4)
    model3 = RandomizedSearchCV(xreg, param_distributions= c_param, scoring = "ne

    model3.fit(df_train, train_output)

    y_pred = model3.predict(df_test)
    xgb_test_predictions = [round(value) for value in y_pred]
    y_pred = model3.predict(df_train)
    xgb_train_predictions = [round(value) for value in y_pred]
    print(model3.best_params_)

    return xgb_train_predictions,xgb_test_predictions
```

In [151]:
```
# predicting with our trained Xg-Boost regressor

xgb_train_predictions,xgb_test_predictions=xg_reg(df_train,df_test,train_output)
```

```
{'colsample_bytree': 0.7403721329380182, 'gamma': 0.006302676247616212, 'learni
ng_rate': 0.0303033149457079, 'max_depth': 5, 'min_child_weight': 5, 'n_estimat
ors': 404, 'reg_alpha': 35, 'reg_lambda': 61.103073600823606, 'subsample': 0.88
17814327558107}
```

In [152]:
```
# Calculating the error metric values
train_mse_xgb = mean_squared_error(train_output,xgb_train_predictions)
train_mpe_xgb = mean_absolute_error(train_output,xgb_train_predictions)/(sum(trai
test_mse_xgb = mean_squared_error(test_output,xgb_test_predictions)
test_mpe_xgb = mean_absolute_error(test_output,xgb_test_predictions)/(sum(test_ou

print(train_mpe_xgb*100)
print(test_mpe_xgb*100)
```

```
12.239956300453478
11.73770605820867
```

Calculating the error metric values for various models

In [0]:
```
# Store MAPE SCORES
train_mape=[0]*5
test_mape=[0]*5
# Base Line Model MAPE
train_mape[0]=(mean_absolute_error(train_output,df_train['ft_1'].values))/(sum(tr
train_mape[1]=(mean_absolute_error(train_output,df_train['exp_avg'].values))/(sum

# Exponential Averages Forecasting MAPE
test_mape[0]= (mean_absolute_error(test_output, df_test['ft_1'].values))/(sum(tes
test_mape[1]= (mean_absolute_error(test_output, df_test['exp_avg'].values))/(sum(
```

Error Metric Matrix

In [156]:
```
print ("Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)")
print ("-------------------------------------------------------------------------
print ("Baseline Model -                          Train(%): ",train_mape[0]*10
print ("Exponential Averages Forecasting -        Train(%): ",train_mape[1]*10
print ("Linear Regression -                       Train(%): ",train_mpe_sgd*10
print ("Random Forest Regression -                Train(%): ",train_mpe_rf*100
print ("XgBoost Regression -                      Train(%): ",train_mpe_xgb*10
print ("-------------------------------------------------------------------------
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE(%)
-------------------------------------------------------------------------------
-------------------------
Baseline Model -                            Train(%):  13.005473783252741
Test(%):  12.462006969436612
Exponential Averages Forecasting -          Train(%):  12.494239827303064
Test(%):  11.944317081772379
Linear Regression -                         Train(%):  12.516377357076564
Test(%):  11.902378748321238
Random Forest Regression -                  Train(%):  11.451104543509478
Test(%):  11.637367792191494
XgBoost Regression -                        Train(%):  12.239956300453478
Test(%):  11.73770605820867
-------------------------------------------------------------------------------
-------------------------
```

Procedure

1. First step was data pre-processing on various features ex. lattitude & longitude, trip duration, speed, distance etc.
2. Second step was to cluster regions by using Kmeans algorithm.
3. Third step is to try various baseline models like Simple moving averages, Weighted moving averages etc. and their comparision.
4. In fourth step we calculated the fourier features and merged them with the previous five features.
5. At last we applied different models like Linear Regression, Random Forest, XGBoost.
6. Compared the model performances based on MAPE(mean absolute percentage error) and MSE(mean squared error) metric.

Reference:

1. https://stackoverflow.com/questions/27546476/what-fft-descriptors-should-be-used-as-feature-to-implement-classification-or-cl (https://stackoverflow.com/questions/27546476/what-fft-descriptors-should-be-used-as-feature-to-implement-classification-or-cl)
2. https://dsp.stackexchange.com/questions/10062/when-should-i-calculate-psd-instead-of-plain-fft-magnitude-spectrum (https://dsp.stackexchange.com/questions/10062/when-should-i-calculate-psd-instead-of-plain-fft-magnitude-spectrum)

Conclusion:

1. After seeing the results we can conclude that all the models are performing somewhat equally.
2. Randon forest regressionis performing well amoung all the models with train MAPE 11.45% and with test MAPE 11.63
3. Tree based models are outperforming other models slightly.

In [0]: