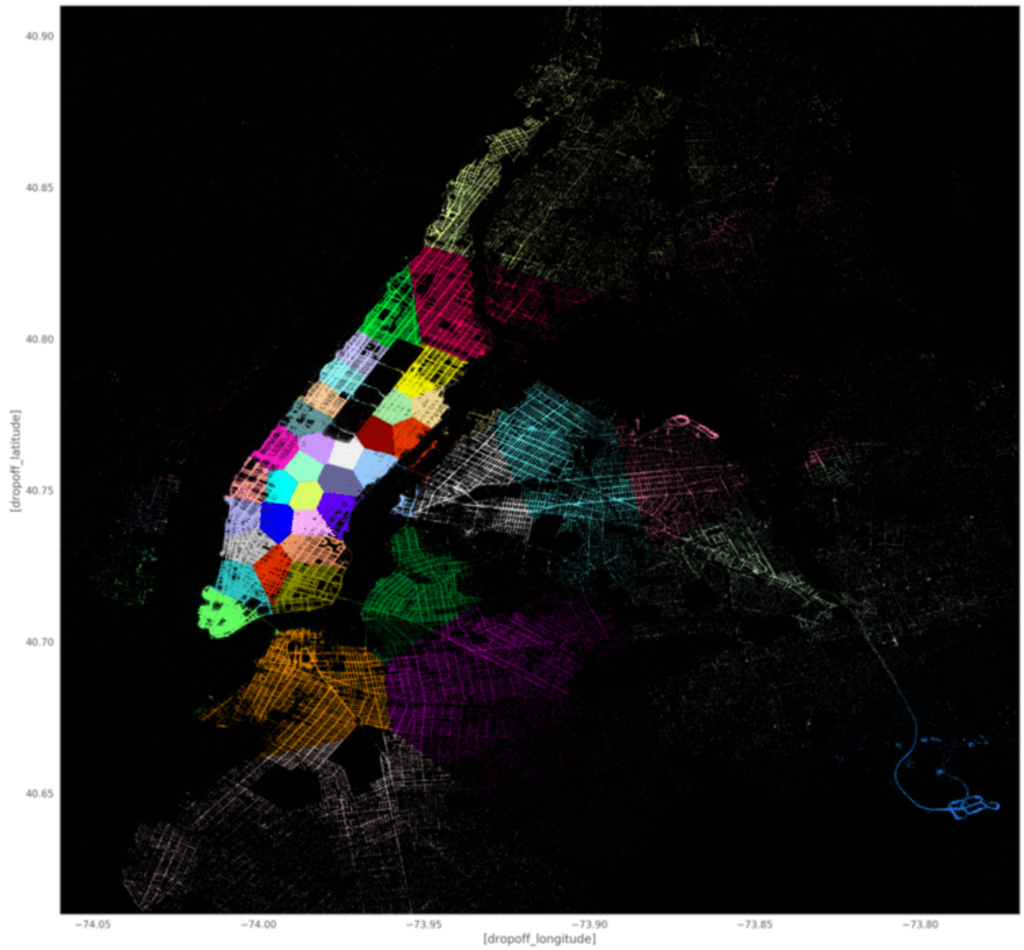
Taxi demand prediction in New York City



In [22]:
```
!pip install gpxpy
```

```
Collecting gpxpy
  Downloading https://files.pythonhosted.org/packages/6e/d3/ce52e67771929de455e76655365a4935a2f369f76dfb0d70c20a308ec463/gpxpy-1.3.5.tar.gz (https://files.pythonhosted.org/packages/6e/d3/ce52e67771929de455e76655365a4935a2f369f76dfb0d70c20a308ec463/gpxpy-1.3.5.tar.gz) (105kB)
    100% |████████████████████████████████| 112kB 3.2MB/s
Building wheels for collected packages: gpxpy
  Building wheel for gpxpy (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/d2/f0/5e/b8e85979e66efec3eaa0e47fbc5274db99fd1a07befd1b2aa4
Successfully built gpxpy
Installing collected packages: gpxpy
Successfully installed gpxpy-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 [24]:

```
!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) (3.0.0)
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: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->kaggle) (2.6)
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[24]:

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

In [25]:
```
!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
100% 1.75G/1.76G [00:20<00:00, 40.5MB/s]
100% 1.76G/1.76G [00:20<00:00, 92.4MB/s]
df_test   kaggle.json  taxidemand.zip    tsne_train_output
df_train  sample_data  tsne_test_output
```

In [26]:
```
!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) (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)

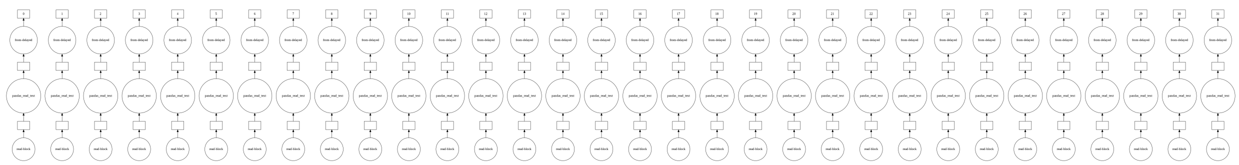
file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	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 [27]:
```
#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')
```

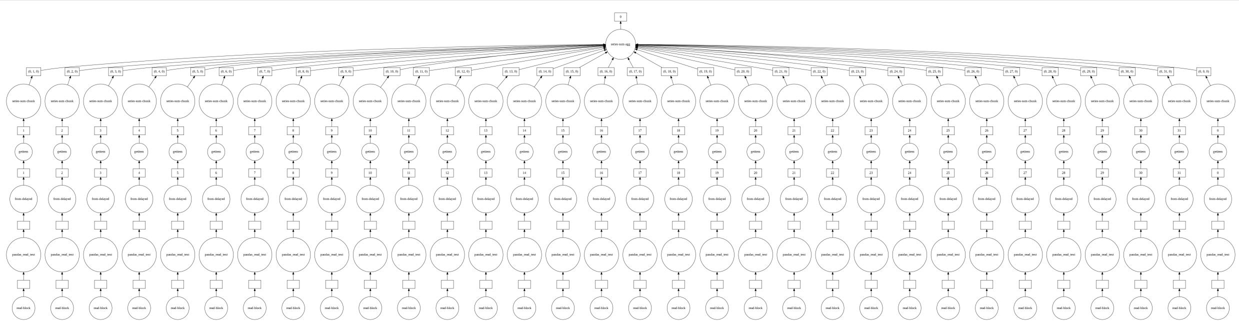
In [0]:
```
# However unlike Pandas, operations on dask.dataframes don't trigger immediate co
# instead they add key-value pairs to an underlying Dask graph. Recall that in th
# circles are operations and rectangles are results.

# 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]:
```
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 [28]:
```
#table below shows few datapoints along with all our features
jan_2015.head(5)
```

Out[28]:

	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 [31]:
```
# 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
```

Out[31]:

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 [0]:
```
# 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
```

Out[9]:

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

3. Trip Durations:

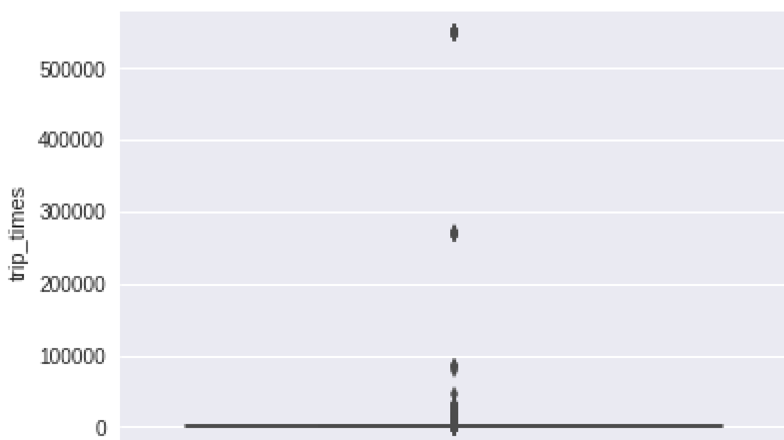
According to NYC Taxi & Limousine Commision Regulations **the maximum allowed trip duration in a 24 hour interval is 12 hours.**

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 [0]:
```
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
```

In [0]:
```
%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 [0]:
```
#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 [0]:
```
#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])
```

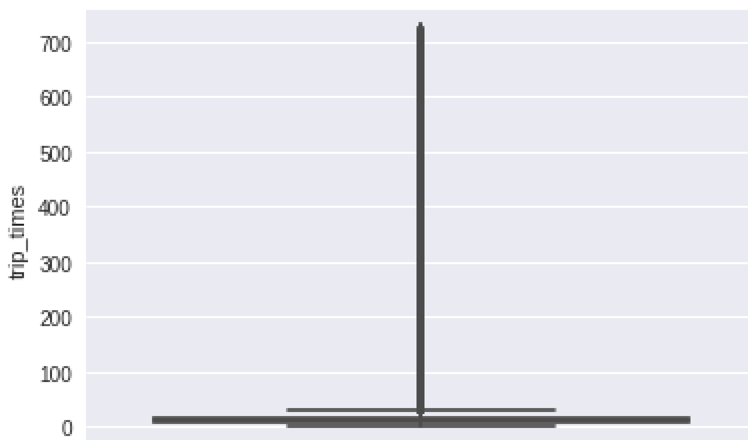
```
90 percentile value is 23.45
91 percentile value is 24.35
92 percentile value is 25.383333333333333
93 percentile value is 26.55
94 percentile value is 27.933333333333334
95 percentile value is 29.583333333333332
96 percentile value is 31.683333333333334
97 percentile value is 34.46666666666667
98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

In [0]:
```
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])
```

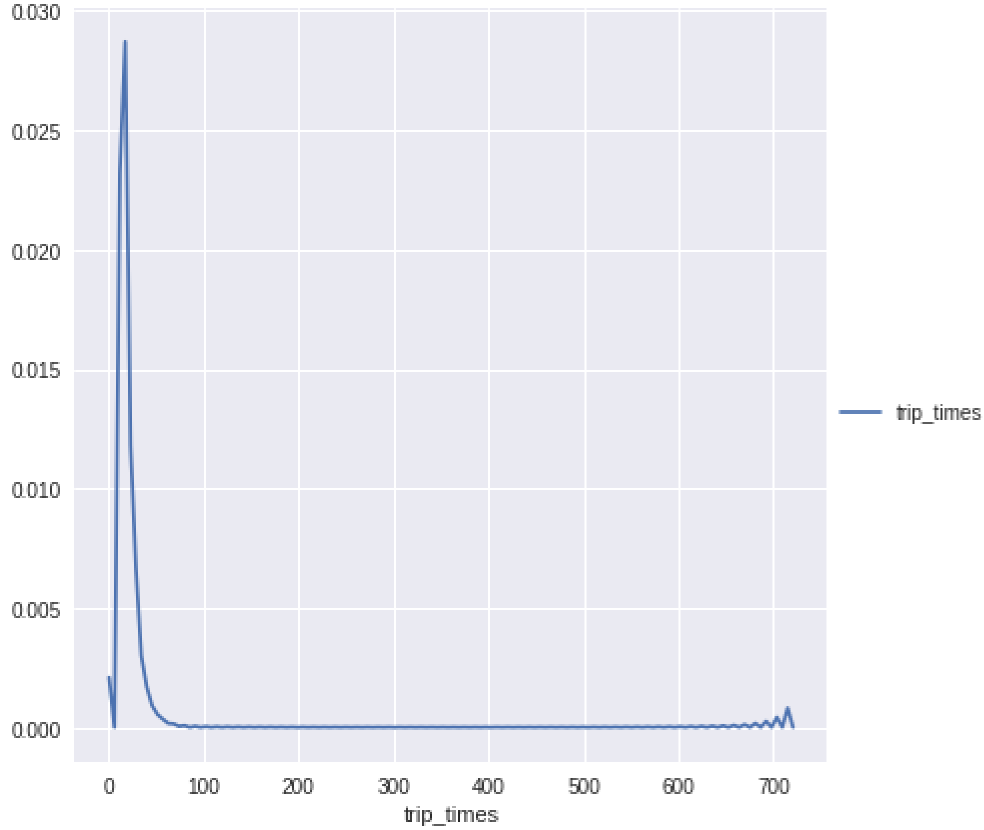
```
99.0 percentile value is 46.75
99.1 percentile value is 48.06666666666667
99.2 percentile value is 49.56666666666667
99.3 percentile value is 51.28333333333333
99.4 percentile value is 53.31666666666667
99.5 percentile value is 55.833333333333336
99.6 percentile value is 59.13333333333333
99.7 percentile 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 [0]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_times", data = updated_duration_of_trip)
plt.show()
```

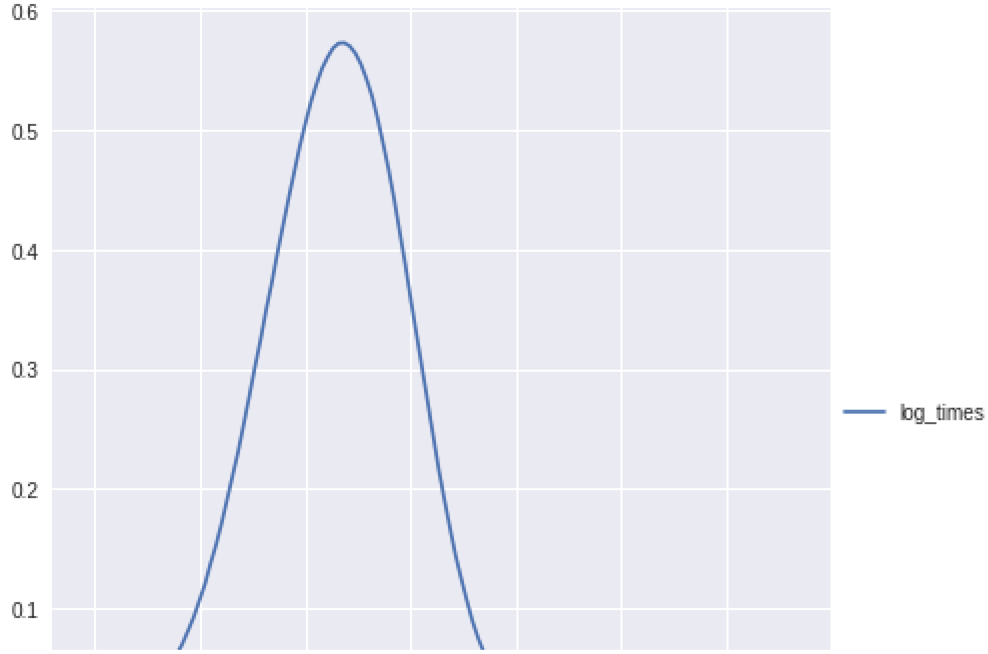


In [0]:
```
#pdf of trip-times after removing the outliers
sns.FacetGrid(updated_duration_of_trip,size=6) \
      .map(sns.kdeplot,"trip_times") \
      .add_legend();
plt.show();
```

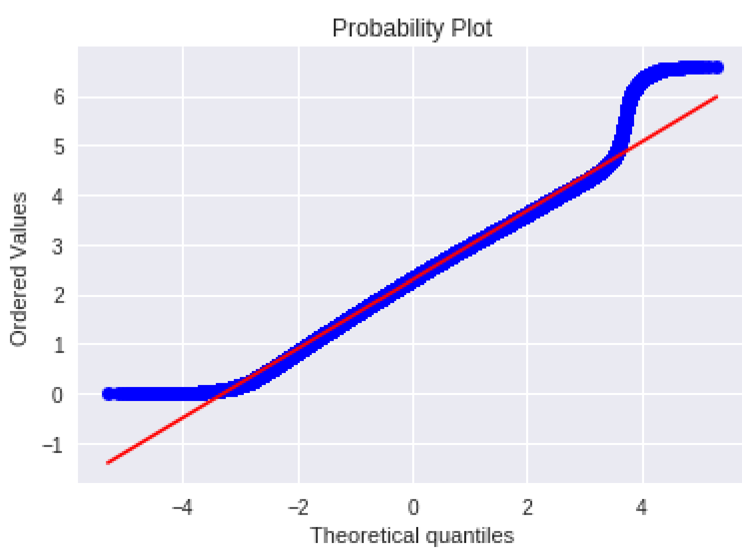


In [0]:
```
#converting the values to log-values to chec for log-normal
import math
updated_duration_of_trip['log_times']=[math.log(i) for i in updated_duration_of_t
```

In [0]:
```
#pdf of log-values
sns.FacetGrid(updated_duration_of_trip,size=6) \
      .map(sns.kdeplot,"log_times") \
      .add_legend();
plt.show();
```

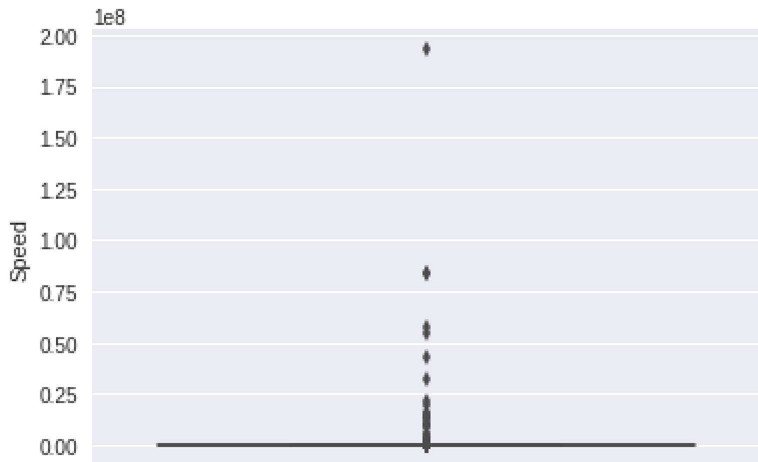


In [0]:
```
#Q-Q plot for checking if trip-times is log-normal
import scipy.stats
import matplotlib.pyplot as plt
scipy.stats.probplot(updated_duration_of_trip['log_times'].values, plot=plt)
plt.show()
```



4. Speed

In [0]:
```python
# 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 [0]:
```python
#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 [0]:
```
#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 [0]:
```
#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
```

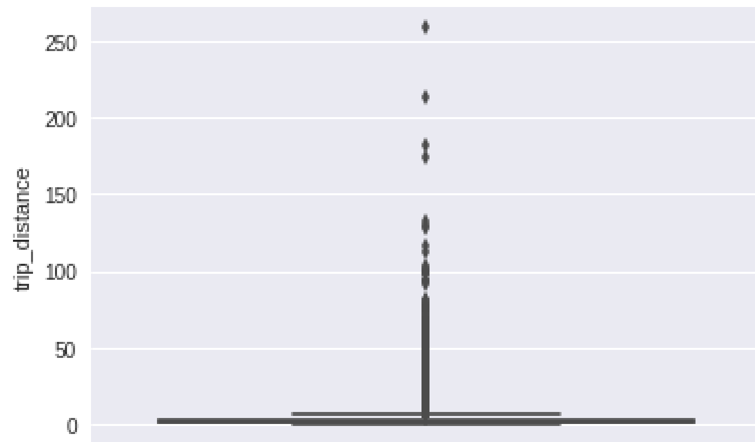
In [0]:
```
#avg.speed of cabs in New-York
sum(updated_duration_of_trip["Speed"]) / float(len(updated_duration_of_trip["Spee
```

Out[38]:
```
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 [0]:
```
# 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 [0]:
```
#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 [0]:
```
#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])
```

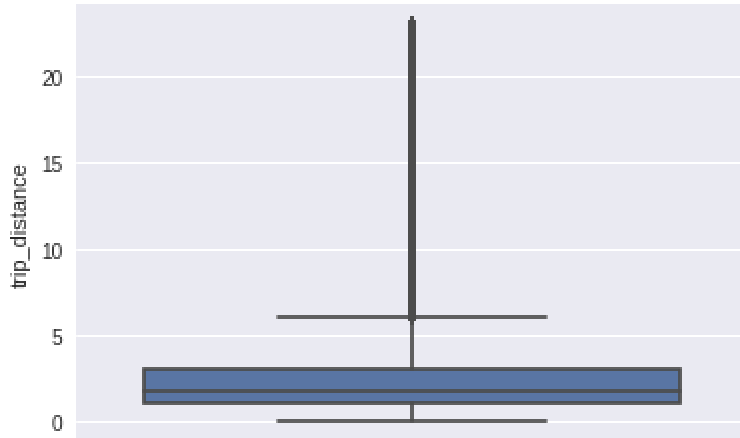
```
90 percentile value is 5.98
91 percentile value is 6.47
92 percentile value is 7.09
93 percentile value is 7.87
94 percentile value is 8.74
95 percentile value is 9.6
96 percentile value is 10.6
97 percentile value is 12.1
98 percentile value is 16.06
99 percentile value is 18.18
100 percentile value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

In [0]:
```
#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
```

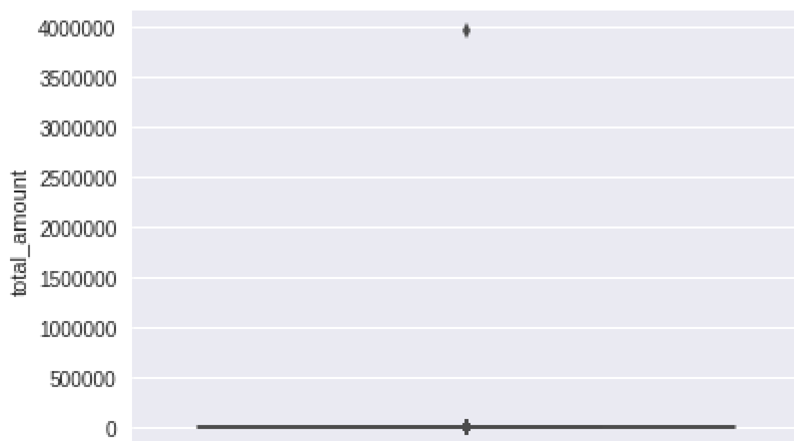
In [0]:
```
#removing further outliers based on the 99.9th percentile value
updated_duration_of_trip = updated_duration_of_trip[(updated_duration_of_trip.tri
```

In [0]:
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = updated_duration_of_trip)
plt.show()
```



5. Total Fare

In [0]:
```
# 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 [0]:
```
#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 [0]:
```
#calculating total fare amount values at each percntile 90,91,92,93,94,95,96,97,98
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 [0]:
```
#calculating total fare amount values at each percntile 99.0,99.1,99.2,99.3,99.4,9
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)/10
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_frame):


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


    temp_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 7
    c = temp_frame.shape[0]
    print ("Number of outliers from trip times analysis:",(a-c))


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

    temp_frame = new_frame[(new_frame.Speed <= 45.31) & (new_frame.Speed >= 0)]
    e = temp_frame.shape[0]
    print ("Number of outliers from speed analysis:",(a-e))

    temp_frame = new_frame[(new_frame.total_amount <86.6) & (new_frame.total_amou
    f = temp_frame.shape[0]
    print ("Number of outliers from fare analysis:",(a-f))


    new_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.d
                    (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropo
                    ((new_frame.pickup_longitude >= -74.15) & (new_frame.picku
                    (new_frame.pickup_longitude <= -73.7004) & (new_frame.pick

    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 <86.6) & (new_frame.total_amoun
    # in real world passenger count in new york cannot be more than 6.
    new_frame = new_frame[new_frame.passenger_count < 6]

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

In [0]:
```
print ("Removing outliers in the month of Jan-2015")
print ("----")
cleaned_data = remove_outliers(frame_with_durations)
print("fraction of data points that remain after removing outliers", float(len(cl
```

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12748986
Number of outlier coordinates lying outside NY boundaries: 293919
Number of outliers from trip times analysis: 23889
Number of outliers from trip distance analysis: 92597
Number of outliers from speed analysis: 36690
Number of outliers from fare analysis: 27561
Total outliers removed 828606
---
fraction of data points that remain after removing outliers 0.9350061251930154
```

Data-preperation

Clustering/Segmentation

In [0]:
```
#trying different cluster sizes to choose the right K in K-means
coords = cleaned_data[['pickup_latitude', 'pickup_longitude']].values
neighbours=[]

def find_min_distance(cluster_centers, cluster_len):
    less2 = []
    more2 = []
    min_dist=1000
    for i in range(0, cluster_len):
        nice_points = 0
        wrong_points = 0
        for j in range(0, cluster_len):
            if j!=i:
                distance = gpxpy.geo.haversine_distance(cluster_centers[i][0], cl
                min_dist = min(min_dist,distance/(1.60934*1000))
                # changing distance into miles
                if (distance/(1.60934*1000)) < 2:
                    nice_points +=1
                else:
                    wrong_points += 1
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Cluster

def find_clusters(increment):
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=
    cleaned_data['pickup_region'] = kmeans.predict(cleaned_data[['pickup_latitude
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

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

```
On choosing a cluster size of  10
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
8.0
Min inter-cluster distance =  1.0531913702962918
---
On choosing a cluster size of  20
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2):
15.0
Min inter-cluster distance =  0.5226035044177118
---
On choosing a cluster size of  30
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2):
```

```
8.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
22.0
Min inter-cluster distance =  0.48155848644119775
---
On choosing a cluster size of  40
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
0.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
30.0
Min inter-cluster distance =  0.4445335318347697
---
On choosing a cluster size of  50
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
1.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
39.0
Min inter-cluster distance =  0.2853861460481325
---
On choosing a cluster size of  60
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
5.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
45.0
Min inter-cluster distance =  0.2982545533880061
---
On choosing a cluster size of  70
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 1
7.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
53.0
Min inter-cluster distance =  0.33001352903406134
---
On choosing a cluster size of  80
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2
0.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
60.0
Min inter-cluster distance =  0.2576278654651188
---
On choosing a cluster size of  90
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 2
2.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 
68.0
Min inter-cluster distance =  0.21800469558780927
---
```

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 20

In [0]:
```
# so we choose 20 clusters for solve the further problem
# Getting 20 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=20, batch_size=10000,random_state=0).fit(coor
cleaned_data['pickup_region'] = kmeans.predict(cleaned_data[['pickup_latitude', '
```

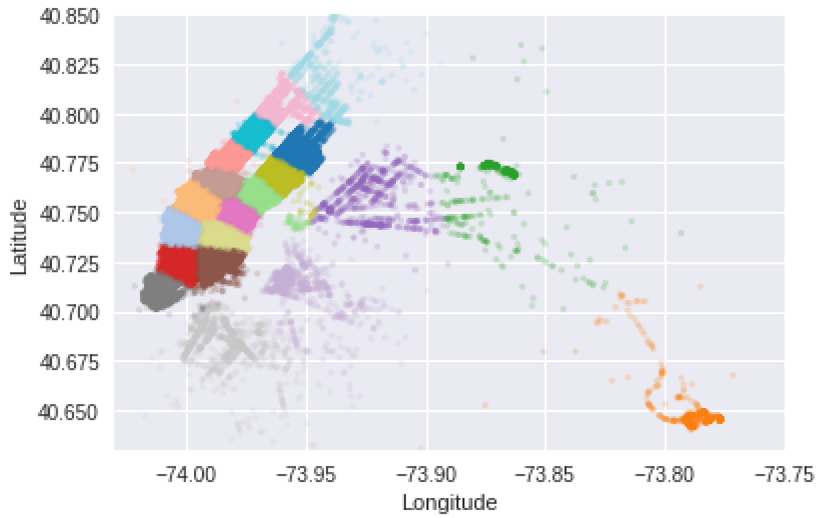
Plotting the cluster centers:

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

Out[53]:

Plotting the clusters:

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

plot_clusters(cleaned_data)
```



Time-binning

In [0]:
```
#Refer:https://www.unixtimestamp.com/
# 1420070400 : 2015-01-01 00:00:00
# 1422748800 : 2015-02-01 00:00:00
# 1425168000 : 2015-03-01 00:00:00
# 1427846400 : 2015-04-01 00:00:00
# 1430438400 : 2015-05-01 00:00:00
# 1433116800 : 2015-06-01 00:00:00

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

def add_pickup_bins(frame,month,year):
    unix_pickup_times=[i for i in frame['pickup_times'].values]
    unix_times = [[1420070400,1422748800,1425168000,1427846400,1430438400,143311б
                    [1451606400,1454284800,1456790400,1459468800,1462060800,14647

    start_pickup_unix=unix_times[year-2015][month-1]
    # https://www.timeanddate.com/time/zones/est
    # (int((i-start_pickup_unix)/600)+33) : our unix time is in gmt to we are con
    tenminutewise_binned_unix_pickup_times=[(int((i-start_pickup_unix)/600)+33) f
    frame['10min_bins'] = np.array(tenminutewise_binned_unix_pickup_times)
    return frame
```

In [0]:
```
# clustering, making pickup bins and grouping by pickup cluster and pickup bins
import math
# Taking log of fare_amount and trip distance and creating new features
cleaned_data['log_fare'] = np.log(cleaned_data['total_amount'])
cleaned_data['log_dist'] = np.log(cleaned_data['trip_distance'])

jan_2015_frame = add_pickup_bins(cleaned_data,1,2015)
jan_2015_groupby = jan_2015_frame[['pickup_region','10min_bins','trip_distance']]
```

In [0]:
```
# 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[57]:

	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 [0]:
```
# 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[58]:

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

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 datapreparation(month,kmeans,month_no,year_no):

    print ("Return with trip times..")

    frame_with_durations = return_with_trip_times(month)

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

    print ("Estimating clusters..")
    frame_with_durations_outliers_removed['pickup_region'] = kmeans.predict(frame_
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] = kmeans.predic

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,m
    final_groupby_frame = final_updated_frame[['pickup_region','10min_bins','trip_

    return final_updated_frame,final_groupby_frame

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

jan_2016_frame,jan_2016_pickups = datapreparation(month_jan_2016,kmeans,1,2016)
feb_2016_frame,feb_2016_pickups = datapreparation(month_feb_2016,kmeans,2,2016)
mar_2016_frame,mar_2016_pickups = datapreparation(month_mar_2016,kmeans,3,2016)
```

```
Return with trip times..
Remove outliers..
Number of pickup records =  10906858
Number of outlier coordinates lying outside NY boundaries: 214677
Number of outliers from trip times analysis: 27190
Number of outliers from trip distance analysis: 79742
Number of outliers from speed analysis: 31018
Number of outliers from fare analysis: 29544
Total outliers removed 664287
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  11382049
Number of outlier coordinates lying outside NY boundaries: 223161
Number of outliers from trip times analysis: 27670
```

```
Number of outliers from trip distance analysis: 81902
Number of outliers from speed analysis: 31866
```

Smoothing

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

In [0]:
```python
# for every month we get all indices of 10min intravels in which atleast one picku

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

#feb
feb_2016_unique_10min_bin = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique_10min_bin = return_unq_pickup_bins(mar_2016_frame)
```

In [0]:
```
# for each cluster number of 10min intravels with 0 pickups
for i in range(20):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  18
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  32
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  116
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  4 th cluster number of 10min intavels with zero pickups:  38
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  21
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  82
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  27
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  23
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
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:  40
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  27
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  20
------------------------------------------------------------
```

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):
    smoothed_regions=[]
    ind=0
    for r in range(0,20):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    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(0,20):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already 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 the left-limit or t
                            continue
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to b
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0
                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known va
                        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: When we have the first/first few values are found to
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]:
                            continue
                        else:
                            right_hand_limit=j
                            break
```

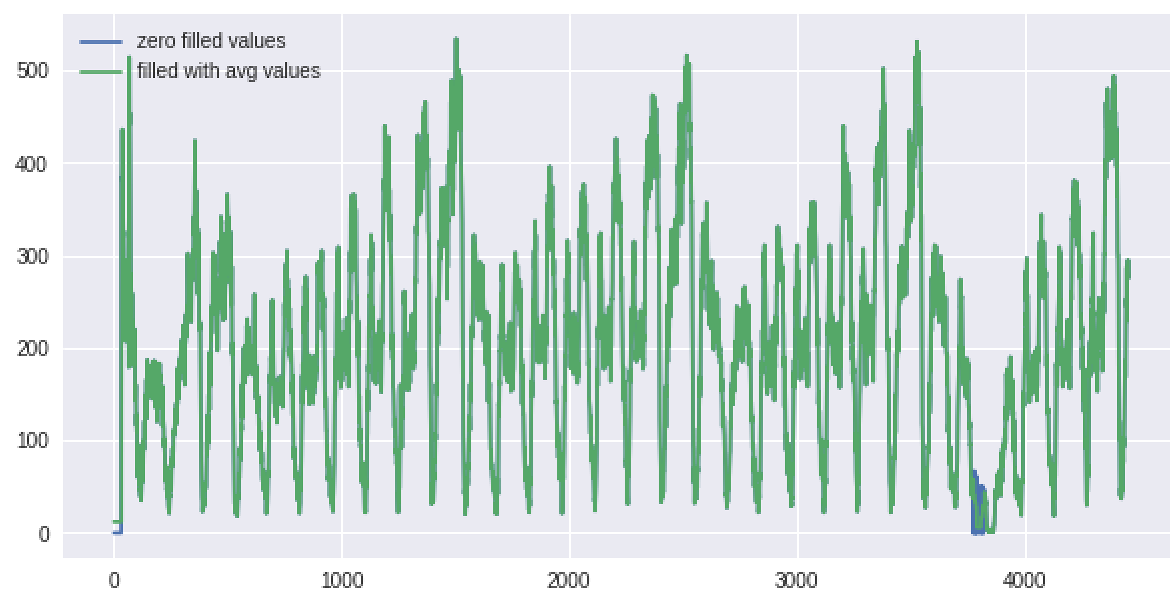
```
                    smoothed_value=count_values[ind]*1.0/((right_hand_limit-i)+1)
                    for j in range(i,right_hand_limit+1):
                        smoothed_bins.append(math.ceil(smoothed_value))
                    repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)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

In [0]:
```
#Filling Missing values of Jan-2015 with 0
# here in jan_2015_groupby dataframe the trip_distance represents the number of p
jan_2015_filledzero = fill_missing(jan_2015_groupby['trip_distance'].values,jan_2

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

In [0]:
```
# 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_filledzero
```

```
number of 10min intravels among all the clusters  89280
```

In [0]:
```
# Smoothing vs Filling
# sample plot that shows two variations of filling missing values
# we have taken the number of pickups for cluster region 2
plt.figure(figsize=(10,5))
plt.plot(jan_2015_filledzero[4464:8920], label="zero filled values")
plt.plot(jan_2015_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```



In [0]:
```
# why we choose, these methods and which method is used for which data?

# Ans: consider we have data of some month in 2015 jan 1st, 10 _ _ _ 20, i.e there
# 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_2016_smooth = fill_missing(jan_2016_pickups['trip_distance'].values,jan_2016_
feb_2016_smooth = fill_missing(feb_2016_pickups['trip_distance'].values,feb_2016_
mar_2016_smooth = fill_missing(mar_2016_pickups['trip_distance'].values,mar_2016_

# Making list of all the values of pickup data in every bin for a period of 3 mon
three_month_pickups_2016 = []

# 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(0,20):
    three_month_pickups_2016.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_s
```

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 $R_t = P^{2016}_t / P^{2015}_t$
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):
            predicted_ratio=(ratios['Ratios'].values)[0]
            error=[]
            predicted_values=[]
            window_size=3
            predicted_ratio_values=[]
            for i in range(0,4464*20):
                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*20):
        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*20):
        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*20):
        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*20):
        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*20):
        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
    # Modified MAPE
    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]*10
median_err=[0]*10
ratios_jan,mean_err[0],median_err[0]=MA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[1],median_err[1]=MA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[2],median_err[2]=WA_R_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[3],median_err[3]=WA_P_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[4],median_err[4]=EA_R1_Predictions(ratios_jan,'jan')
ratios_jan,mean_err[5],median_err[5]=EA_P1_Predictions(ratios_jan,'jan')
```

Comparison between baseline models

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

In [0]:
```
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:  0.162102472591521
66        MSE:  2826.229939516129
Moving Averages (2016 Values) -                        MAPE:  0.114786428679305
04         MSE:  482.2712813620072
--------------------------------------------------------------------------------
-------------------------
Weighted Moving Averages (Ratios) -                    MAPE:  0.161845579965894
27        MSE:  2348.6768705197132
Weighted Moving Averages (2016 Values) -               MAPE:  0.111624049177146
93        MSE:  448.2545138888889
--------------------------------------------------------------------------------
-------------------------
Exponential Moving Averages (Ratios) -              MAPE:  0.1622831185474423
MSE:  2389.792417114695
Exponential Moving Averages (2016 Values) -         MAPE:  0.1117581651141183
MSE:  447.24315636200714
```

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

From the above matrix it is inferred that the best forecasting model for our prediction would be:-

$P'_t = \alpha * P_{t-1} + (1 - \alpha) * P'_{t-1}$ i.e Exponential Moving Averages using 2016 Values

Regression Models

Train-Test Split

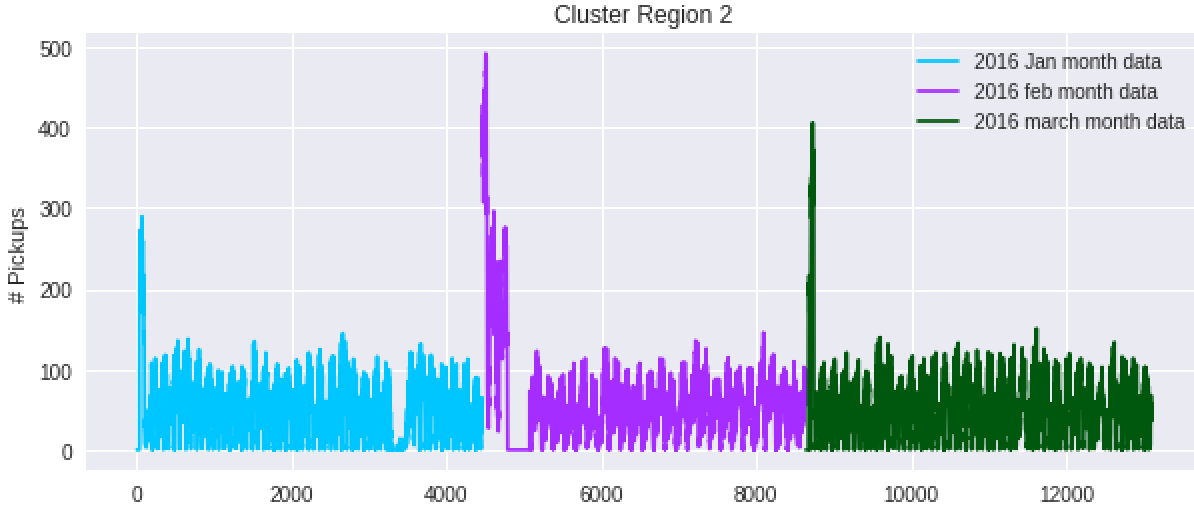
Before we start predictions using the tree based regression models we take 3 months of 2016 pickup data and split it such that for every region we have 70% data in train and 30% in test, ordered date-wise for every region

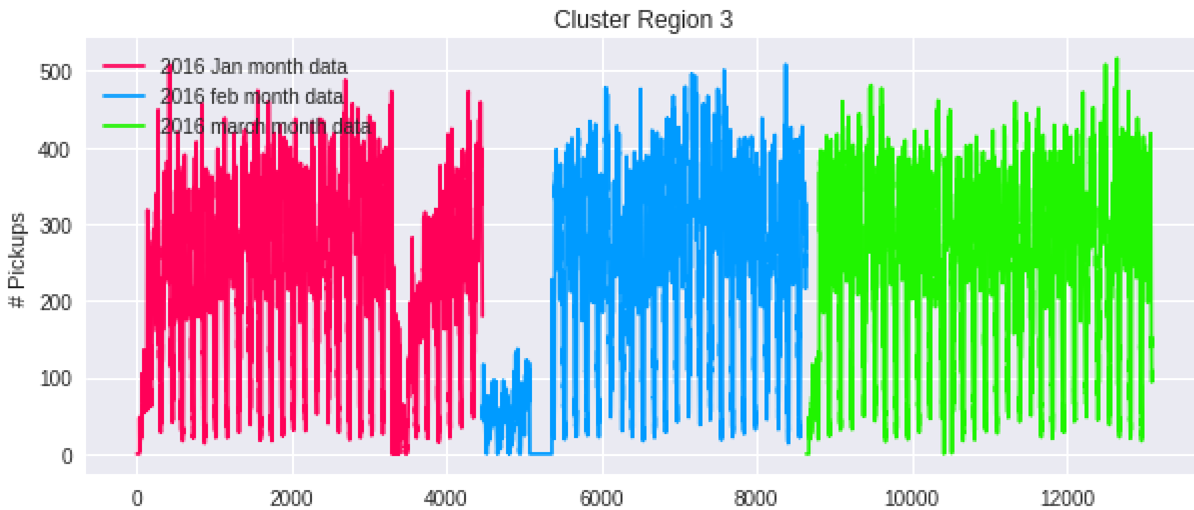
Time series and Fourier Transforms

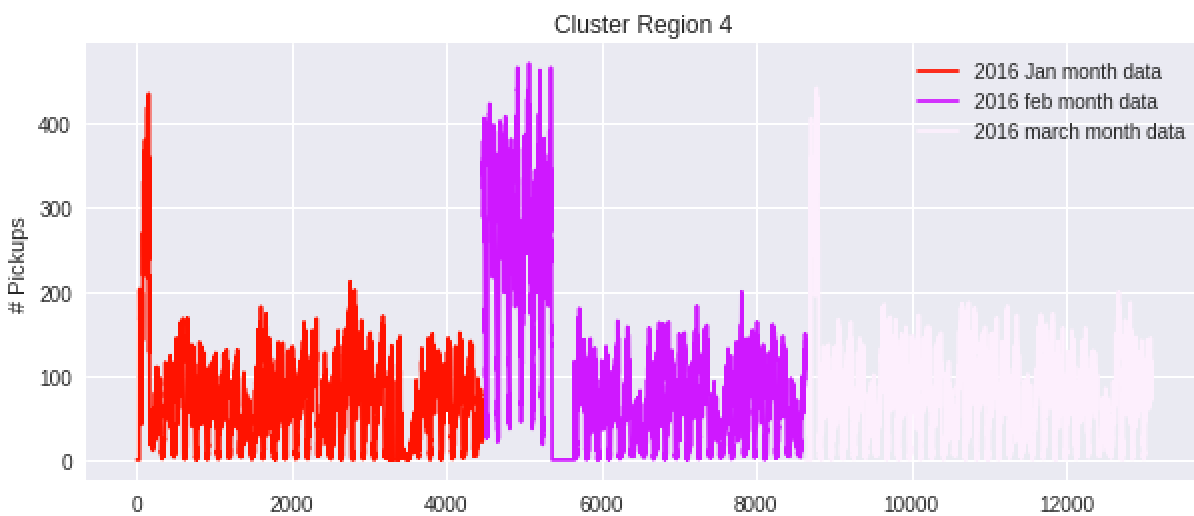
Ploting time series data

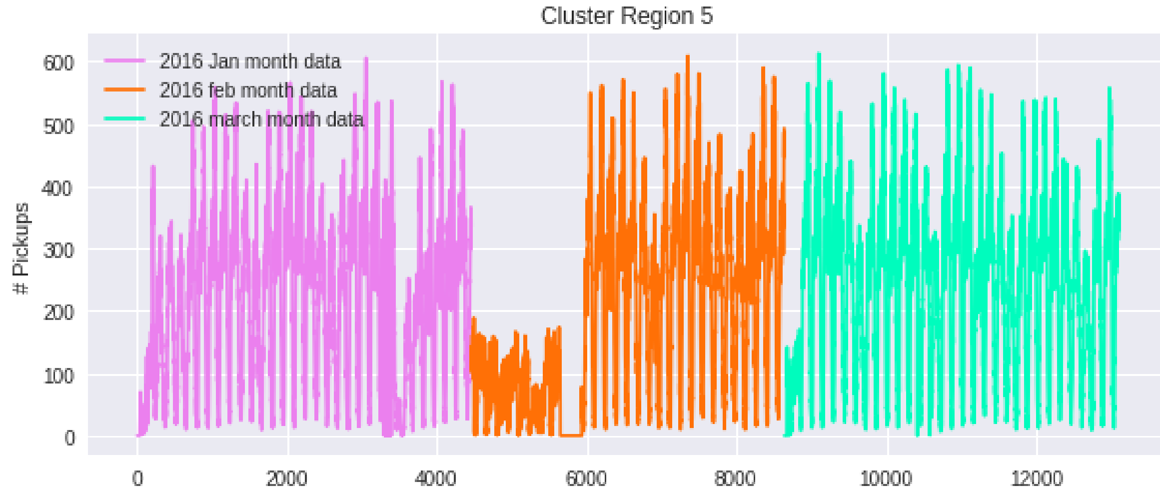
Plot and observe patterns,for each region and month to decide if Fourier Transform is useful

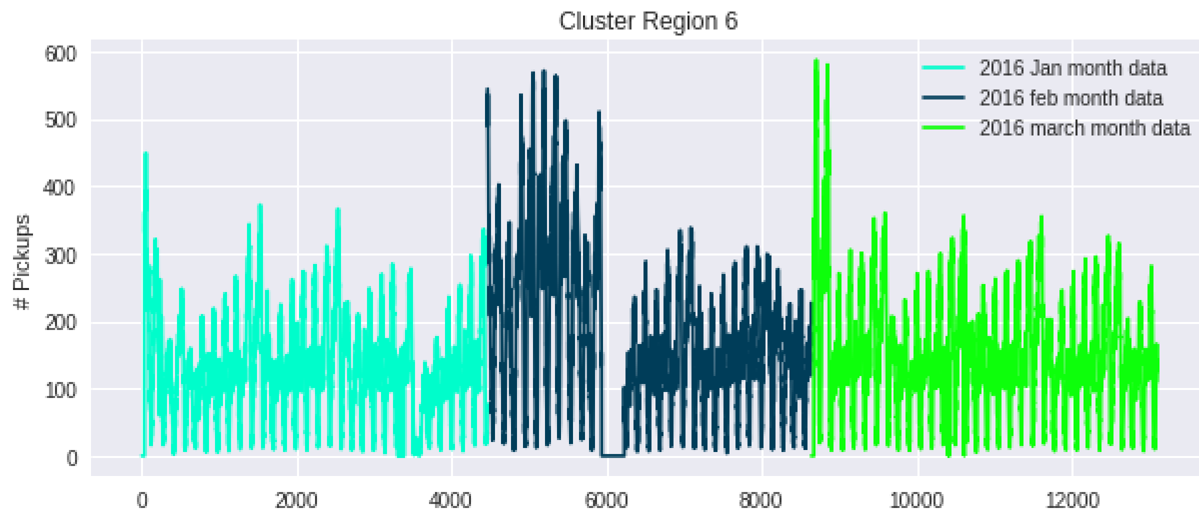
In [0]:
```
def uni_color():
    return plt.cm.gist_ncar(np.random.random())

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, three_month_pickups_2016[i][:4464], color=uni_color(), labe
    plt.plot(second_x, three_month_pickups_2016[i][4464:8640], color=uni_color(),
    plt.plot(third_x, three_month_pickups_2016[i][8640:], color=uni_color(), labe
    plt.legend()
    plt.show()
```

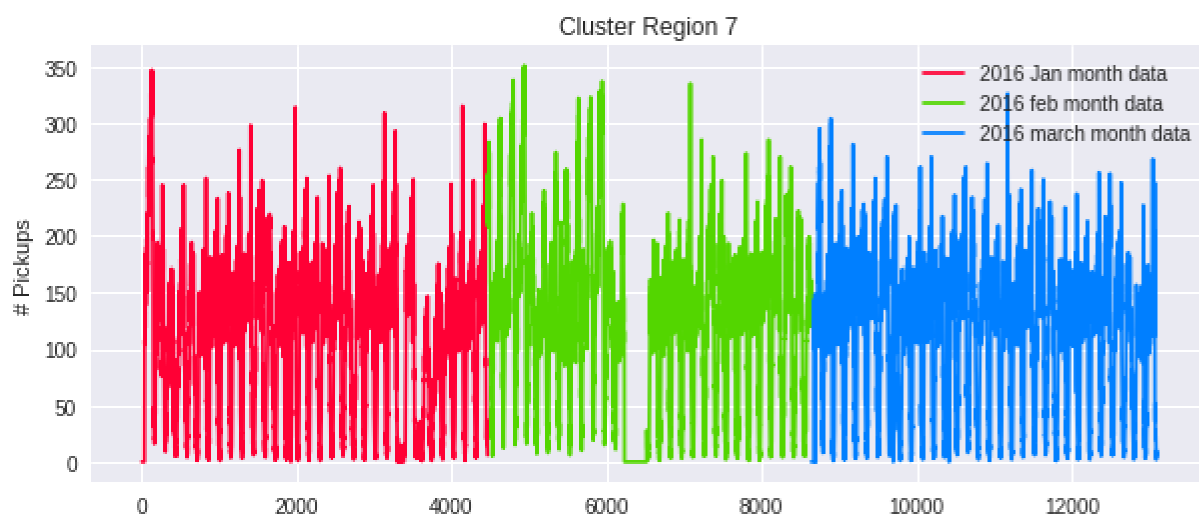

Cluster Region 2
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data

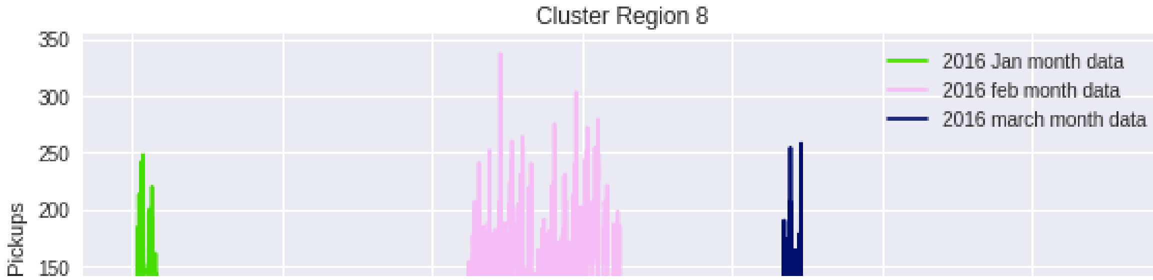

Cluster Region 3
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data

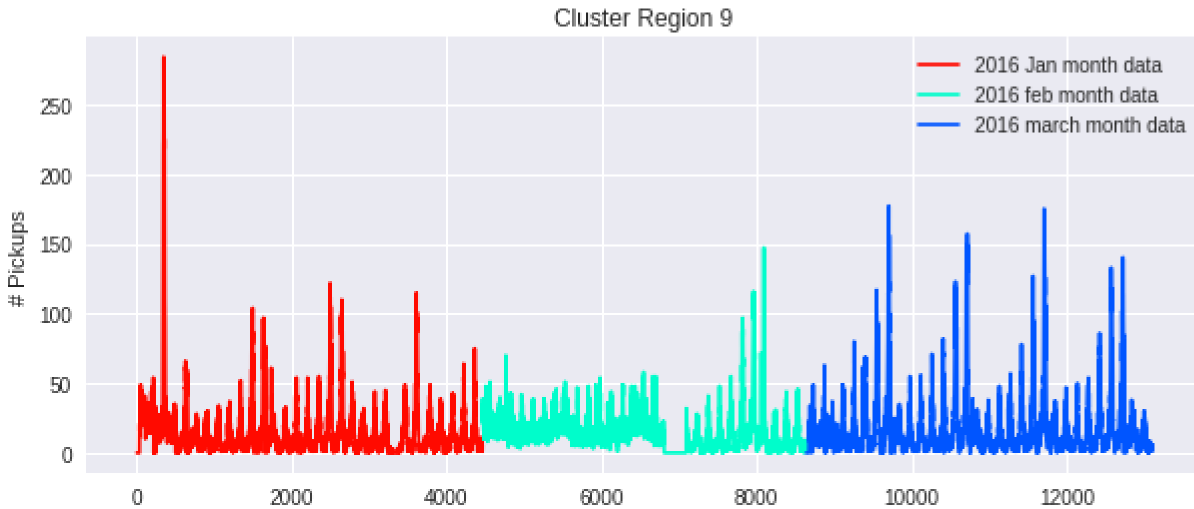

Cluster Region 4
Pickups
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000
2016 Jan month data
2016 feb month data
2016 march month data

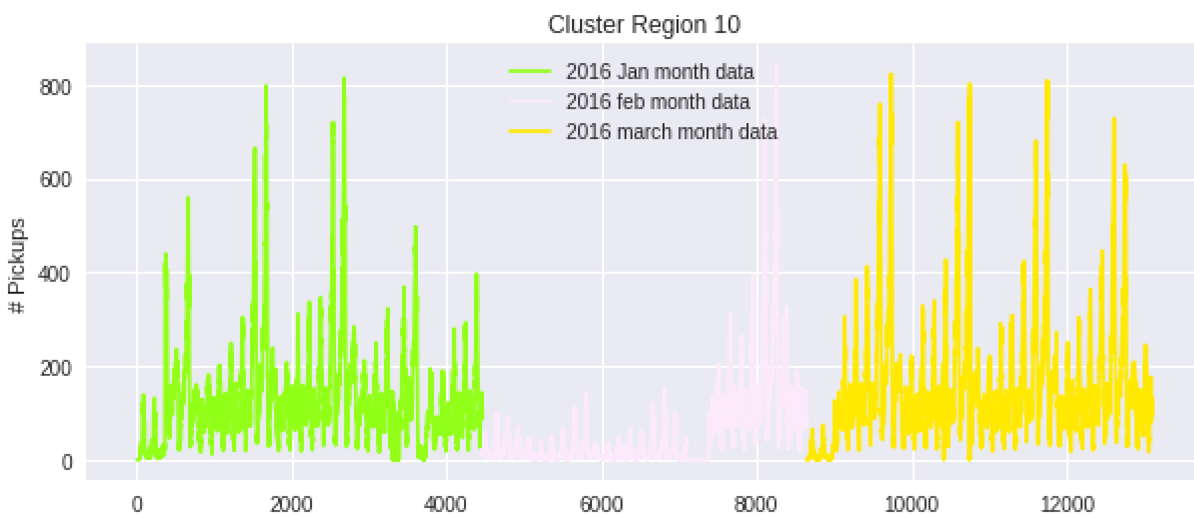

Cluster Region 5
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
600
0
2000
4000
6000
8000
10000
12000

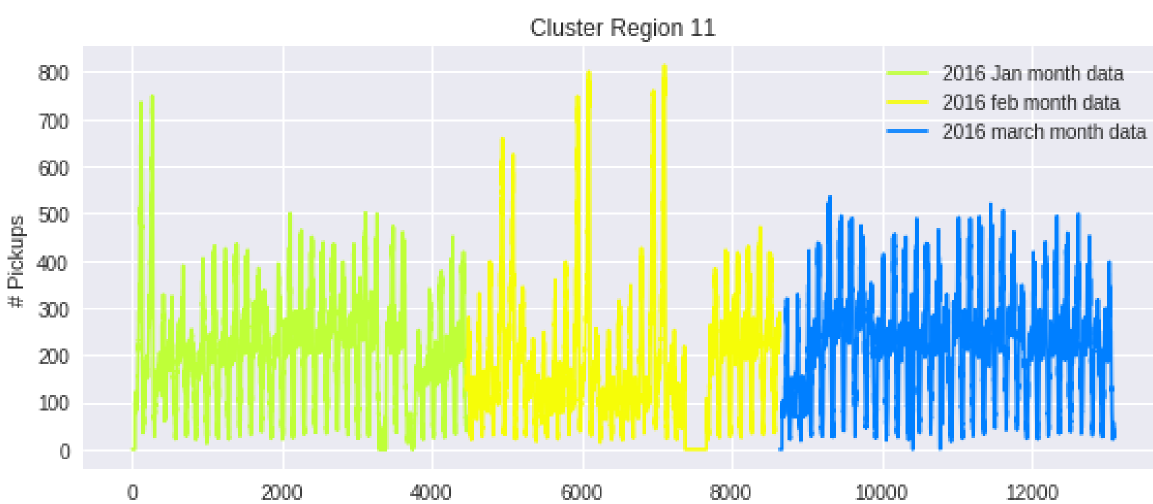

Cluster Region 6
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
600
0
2000
4000
6000
8000
10000
12000


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
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
350
300
250
200
150

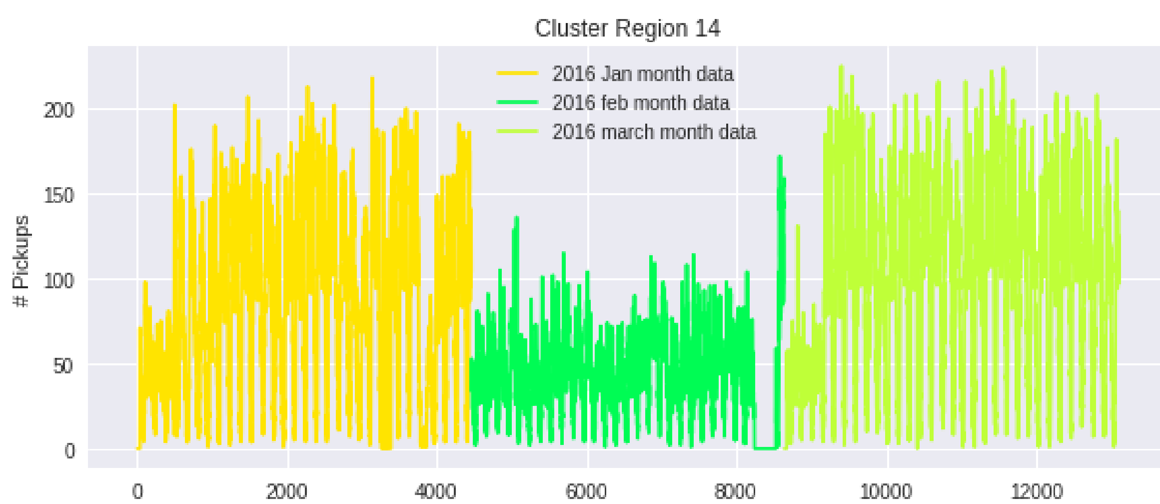

Cluster Region 9
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000

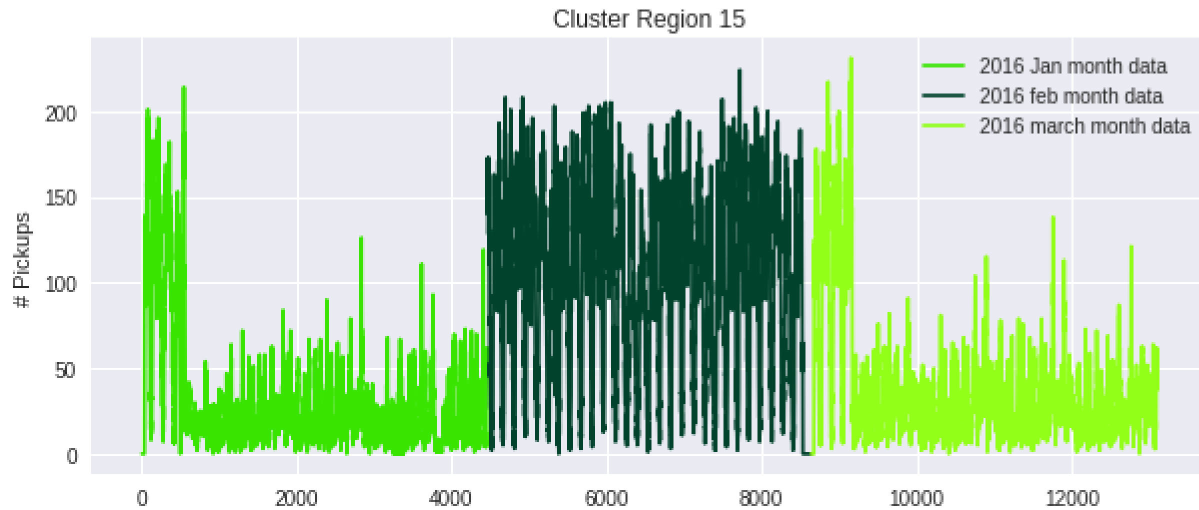

Cluster Region 10
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
800
600
400
200
0
0
2000
4000
6000
8000
10000
12000

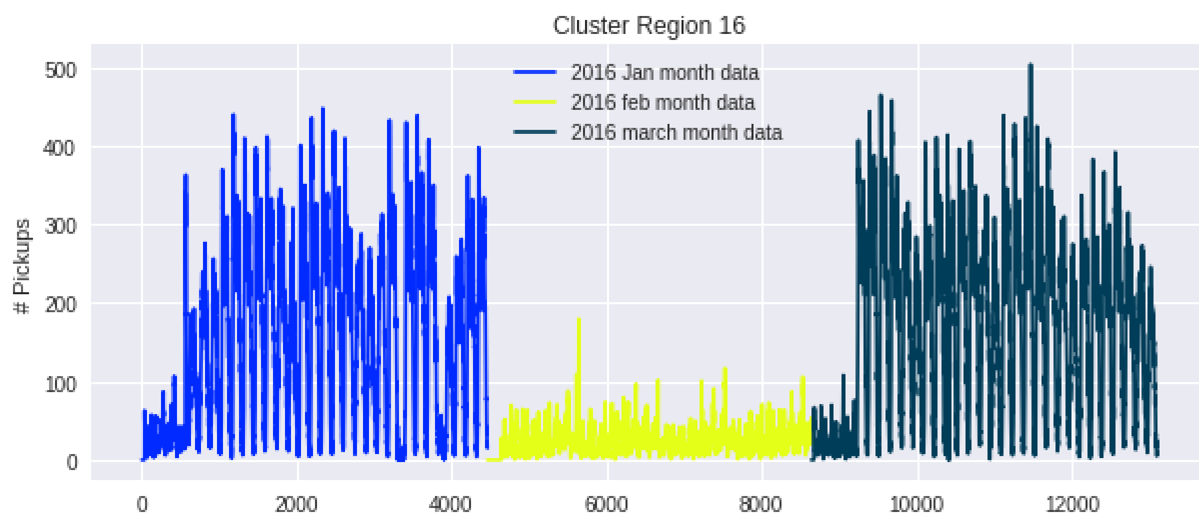

Cluster Region 11
Pickups
2016 Jan month data
2016 feb month data
2016 march month data
800
700
600
500
400
300
200
100
0
0
2000
4000
6000
8000
10000
12000

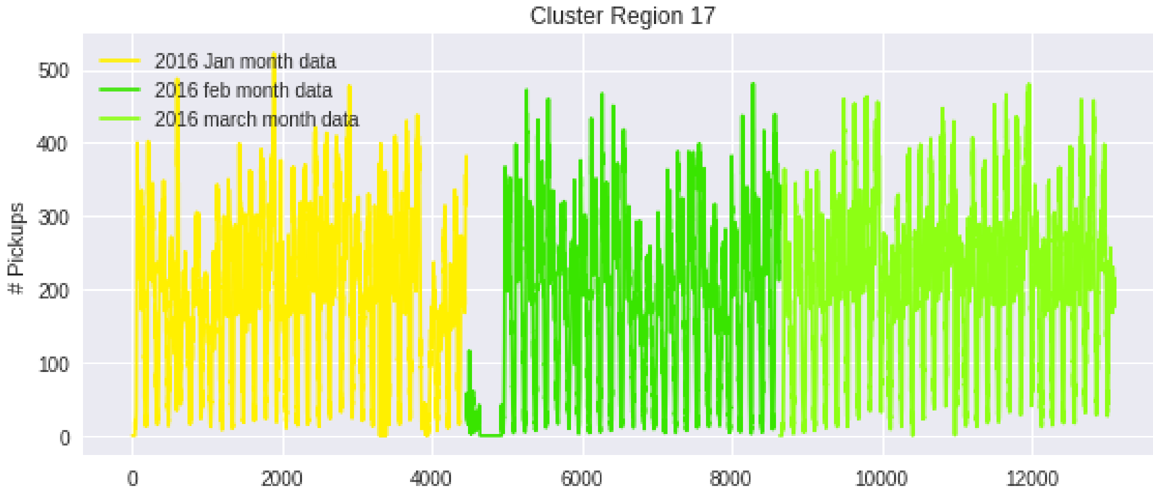

Cluster Region 12
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000

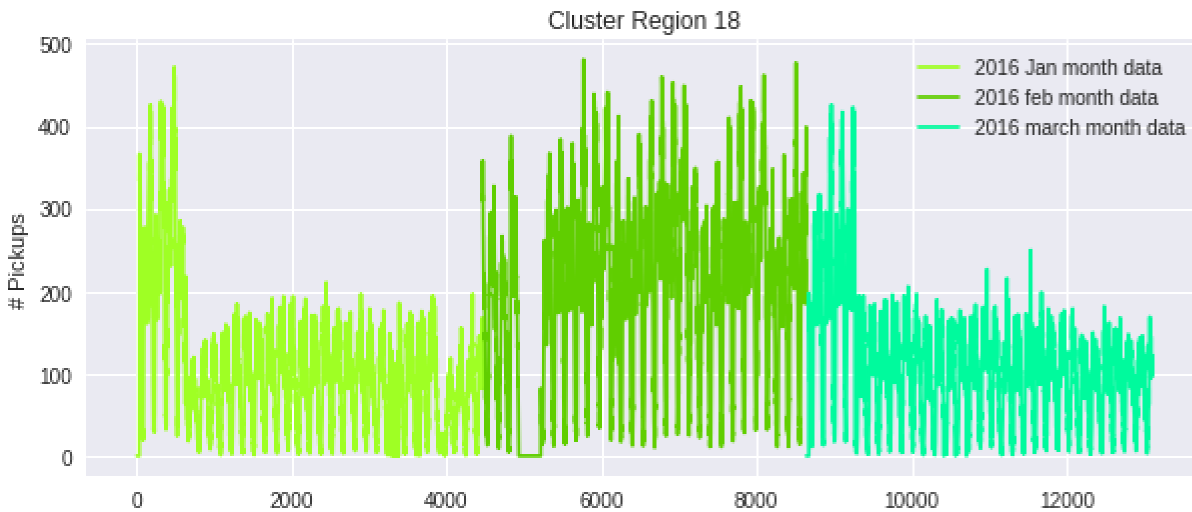

Cluster Region 13
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000

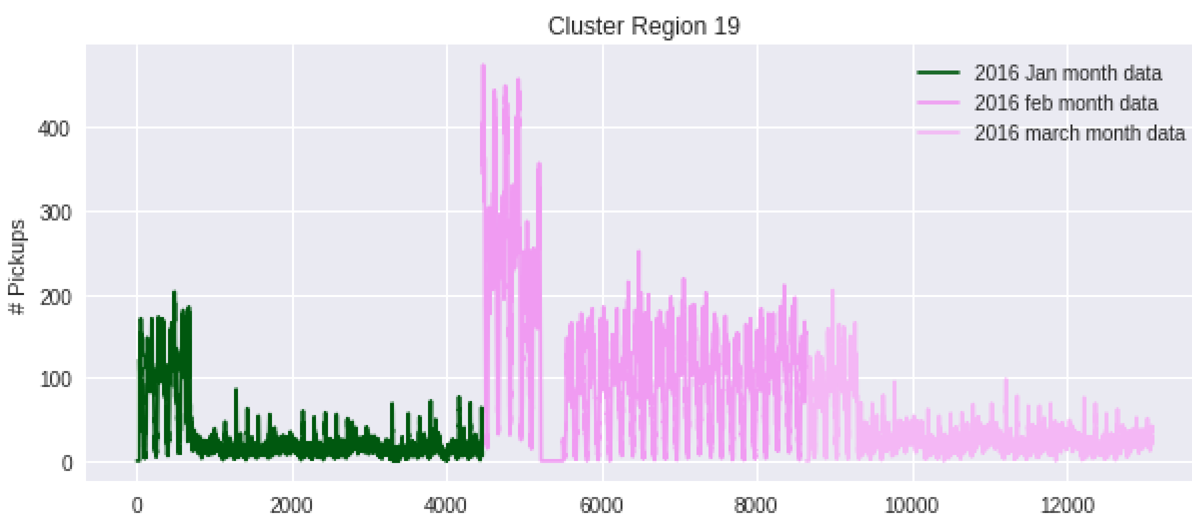

Cluster Region 14
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000


Cluster Region 15
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

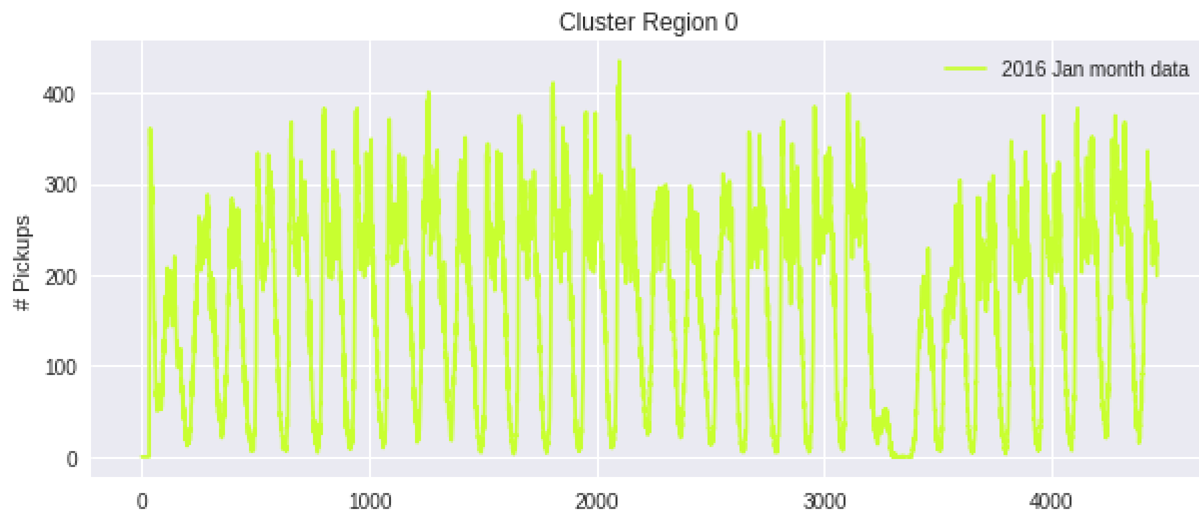

Cluster Region 16
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000

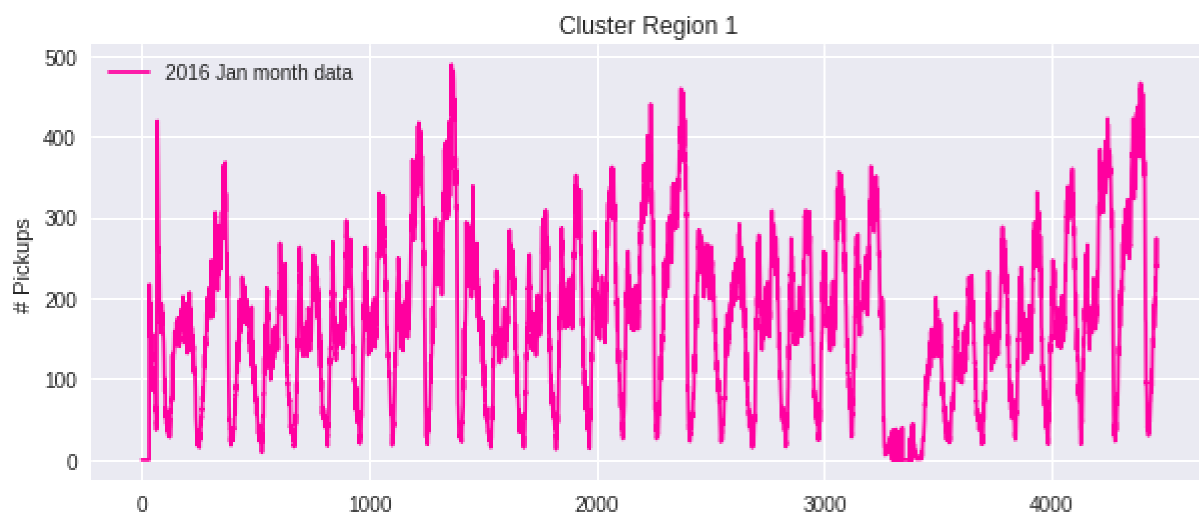

Cluster Region 17
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000

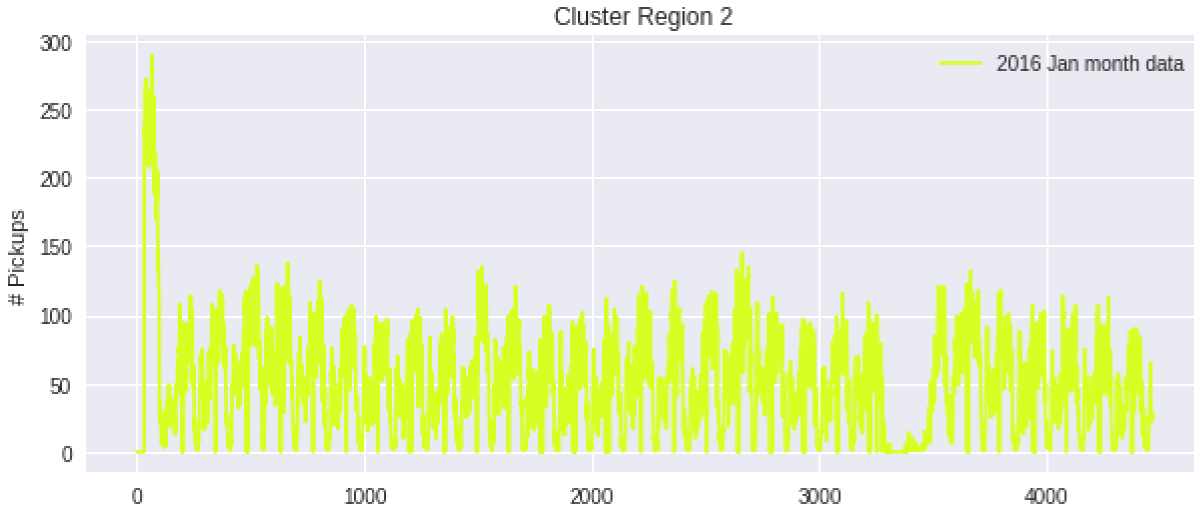

Cluster Region 18
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
12000

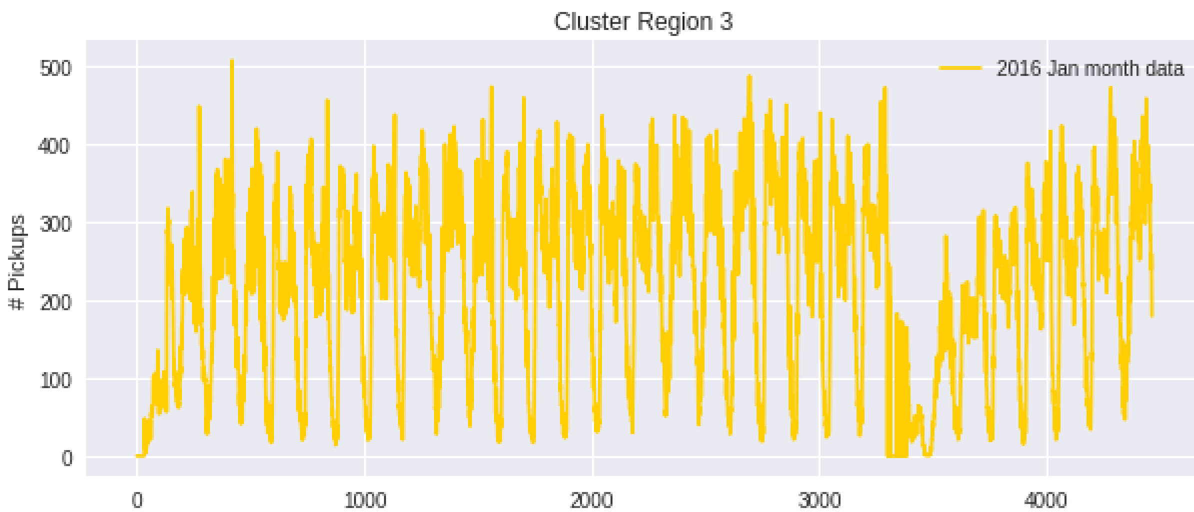

Cluster Region 19
2016 Jan month data
2016 feb month data
2016 march month data
Pickups
0
100
200
300
400
0
2000
4000
6000
8000
10000
12000

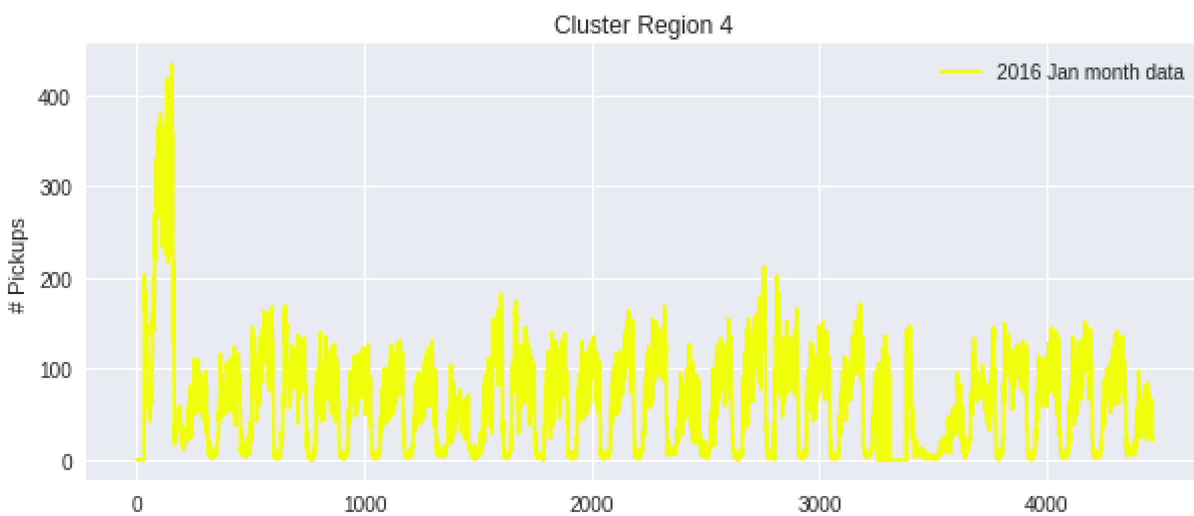
In [0]:
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(20):
    plt.figure(figsize=(10,4))
    plt.title("Cluster Region "+str(i))
    plt.ylabel("# Pickups")
    plt.plot(first_x,three_month_pickups_2016[i][:4464], color=uniqueish_color(),
    plt.legend()
    plt.show()
```

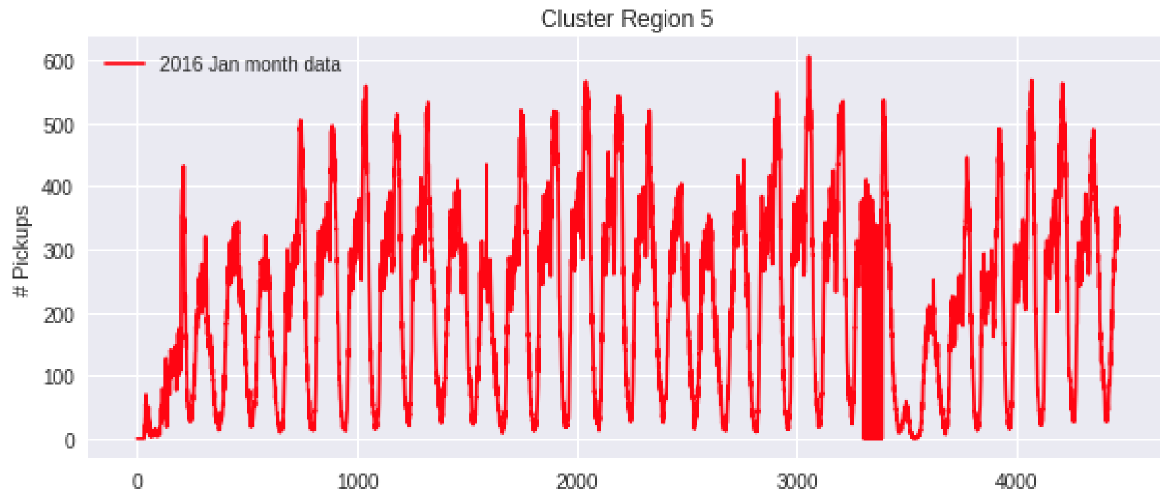


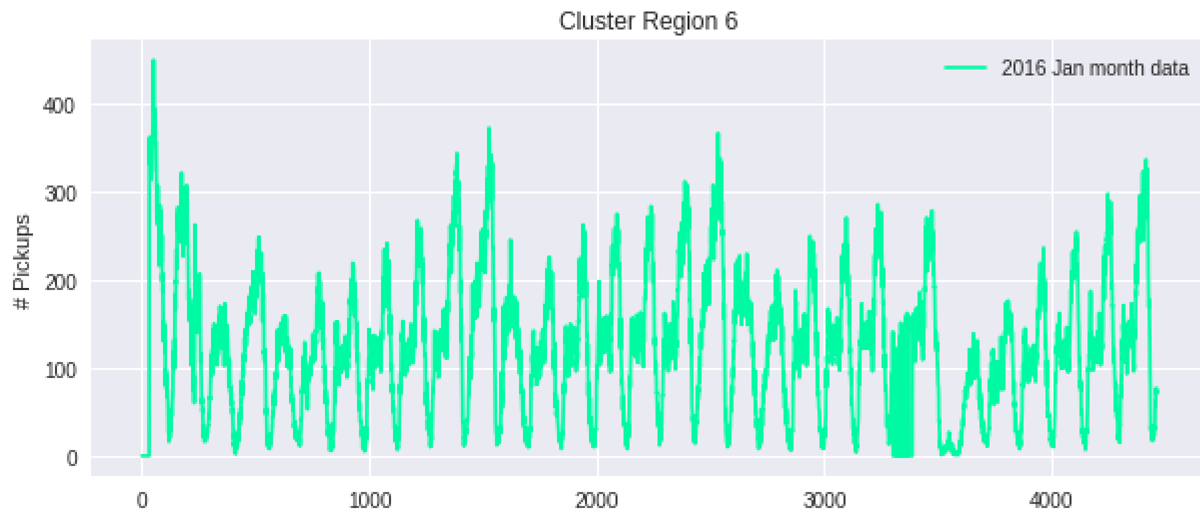


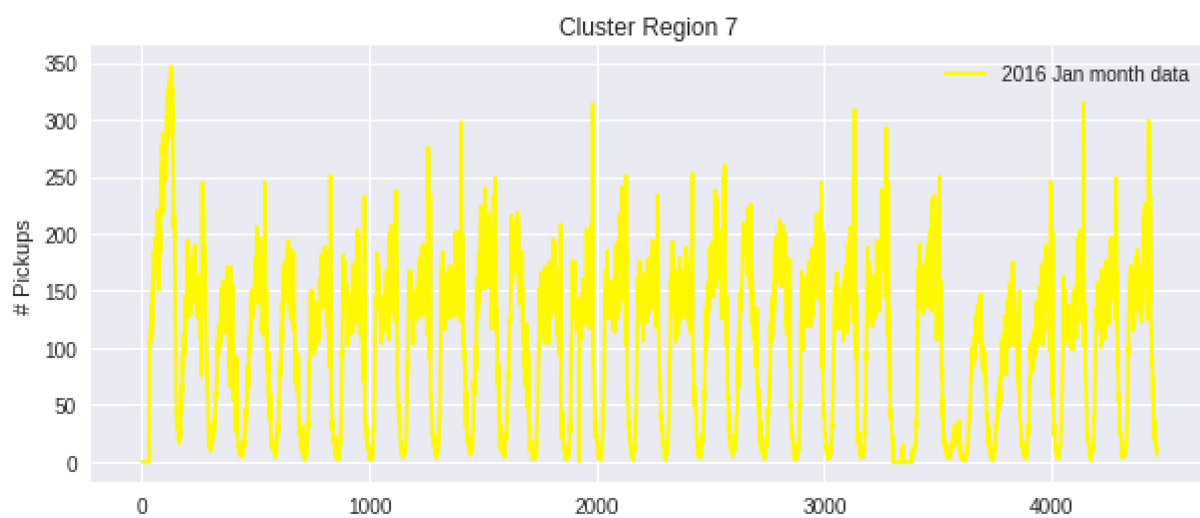

Cluster Region 2
2016 Jan month data
Pickups
300
250
200
150
100
50
0
0
1000
2000
3000
4000

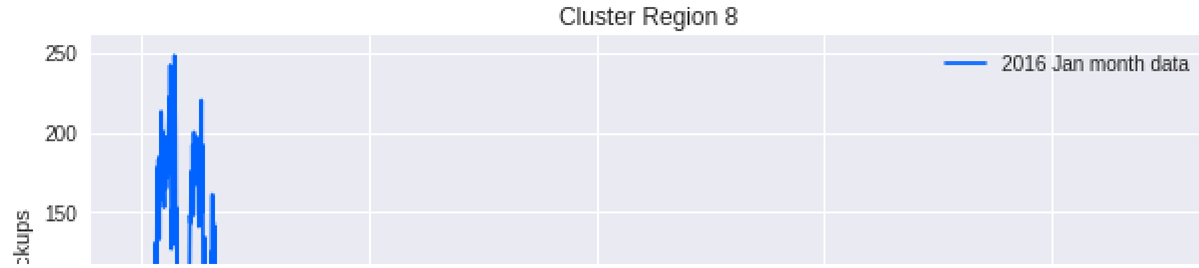

Cluster Region 3
2016 Jan month data
Pickups
500
400
300
200
100
0
0
1000
2000
3000
4000

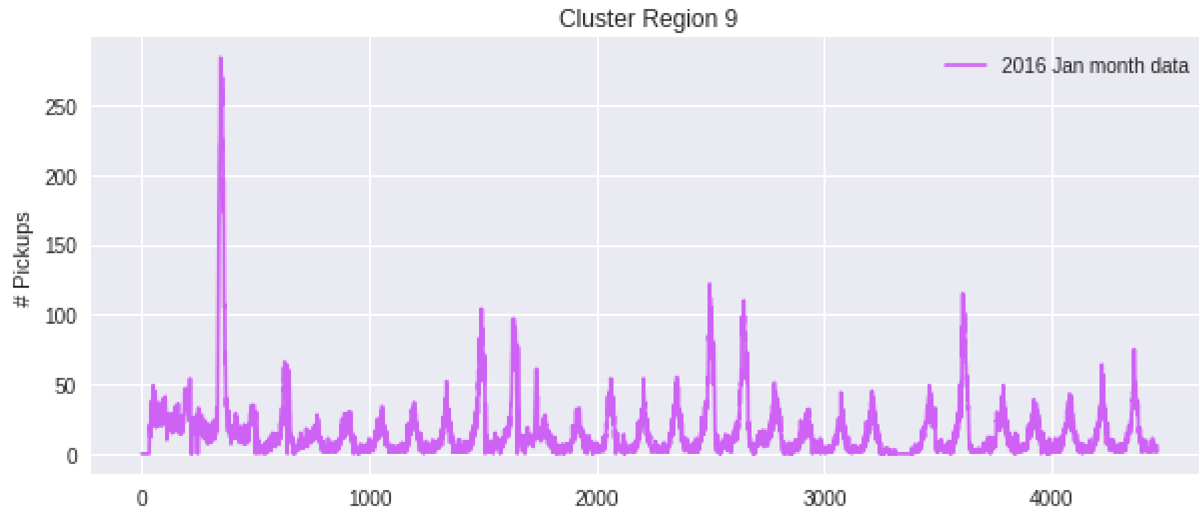

Cluster Region 4
2016 Jan month data
Pickups
400
300
200
100
0
0
1000
2000
3000
4000

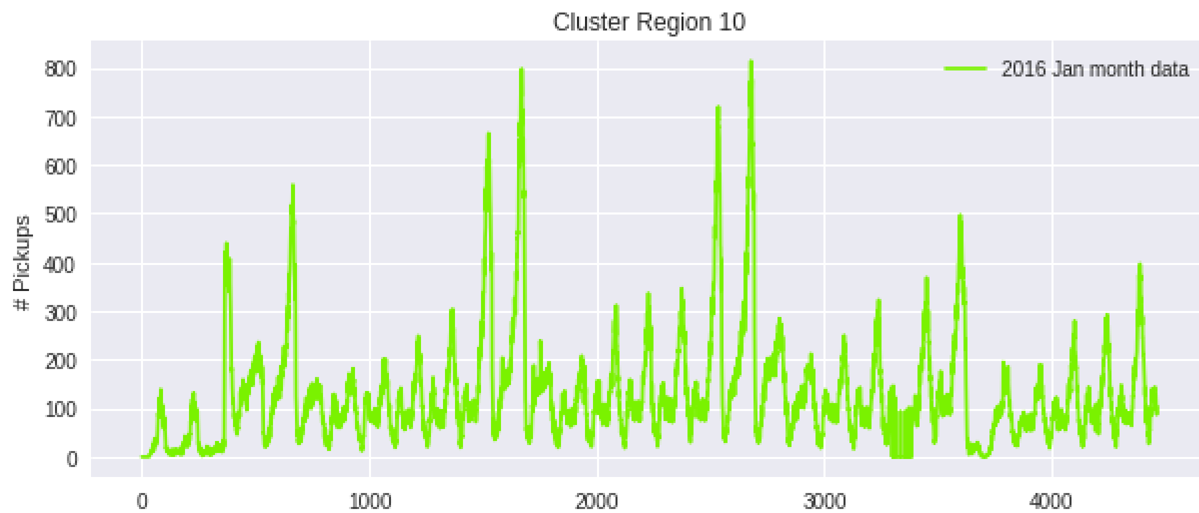

Cluster Region 5
2016 Jan month data
Pickups
600
500
400
300
200
100
0
0
1000
2000
3000
4000

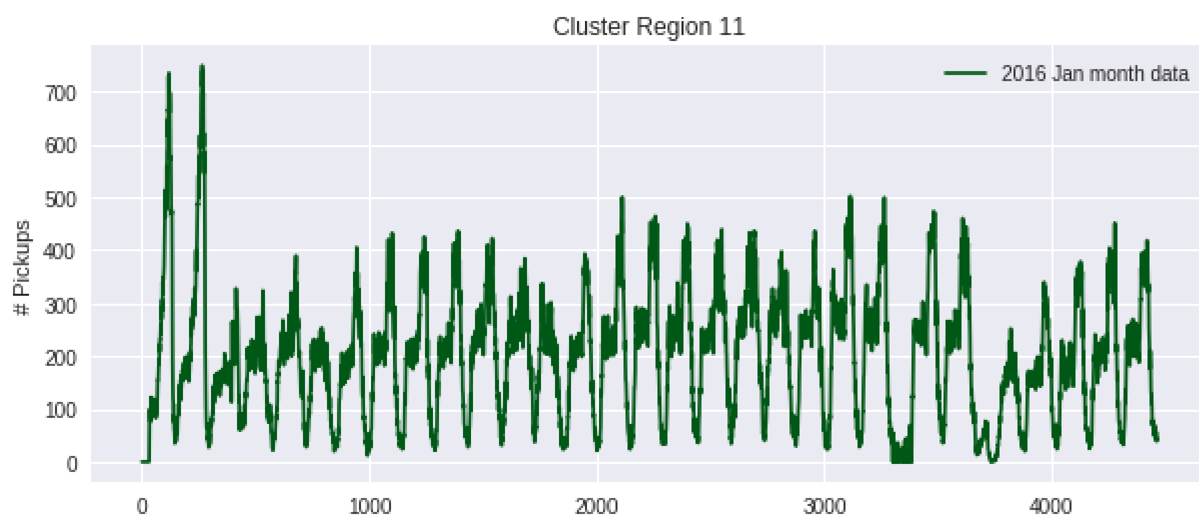

Cluster Region 6
2016 Jan month data
Pickups
400
300
200
100
0
0
1000
2000
3000
4000

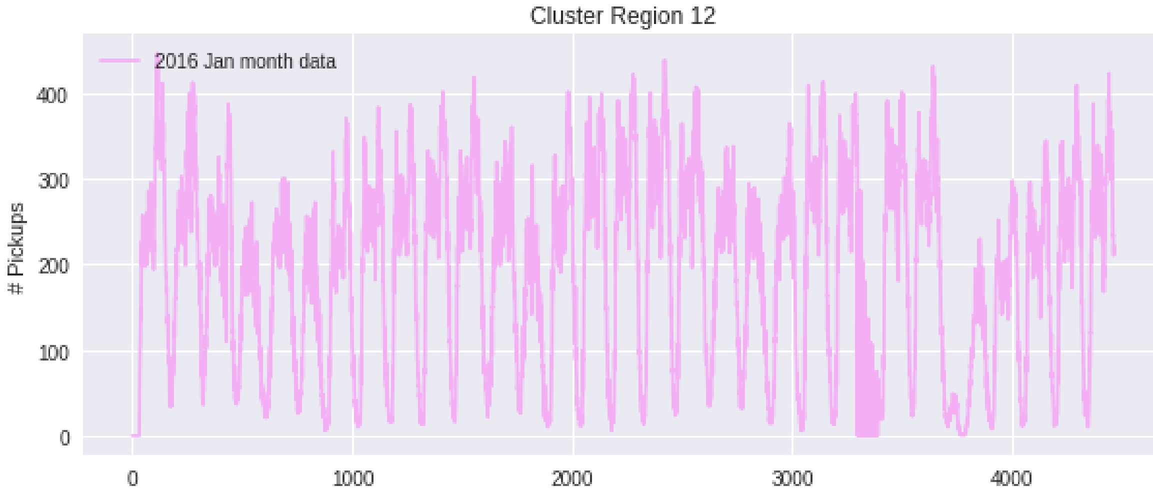

Cluster Region 7
2016 Jan month data
Pickups
350
300
250
200
150
100
50
0
0
1000
2000
3000
4000


Cluster Region 8
2016 Jan month data
250
200
150
ckups

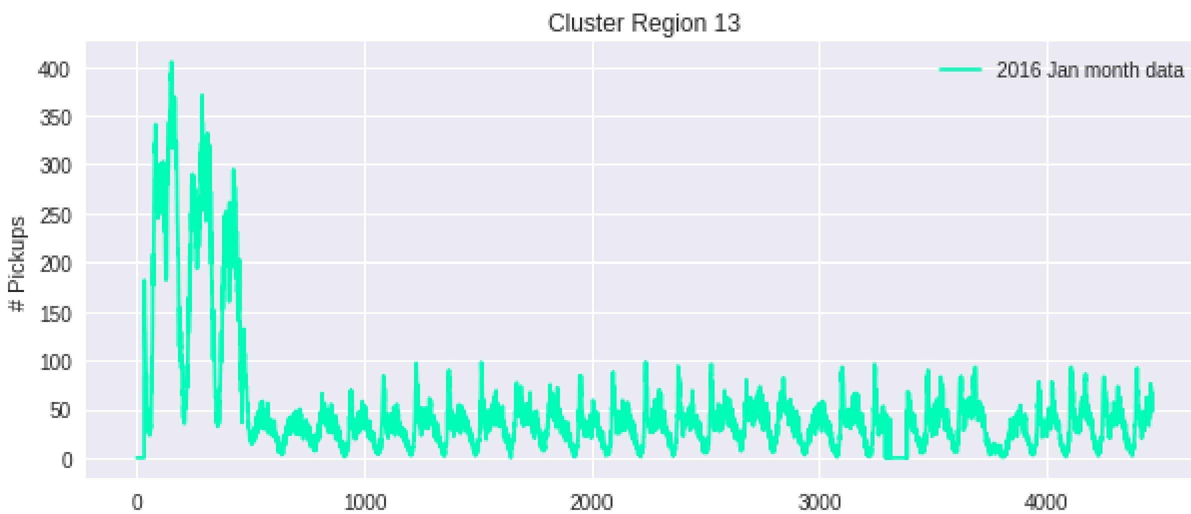

Cluster Region 9
2016 Jan month data
Pickups
250
200
150
100
50
0
0
1000
2000
3000
4000


Cluster Region 10
2016 Jan month data
Pickups
800
700
600
500
400
300
200
100
0
0
1000
2000
3000
4000

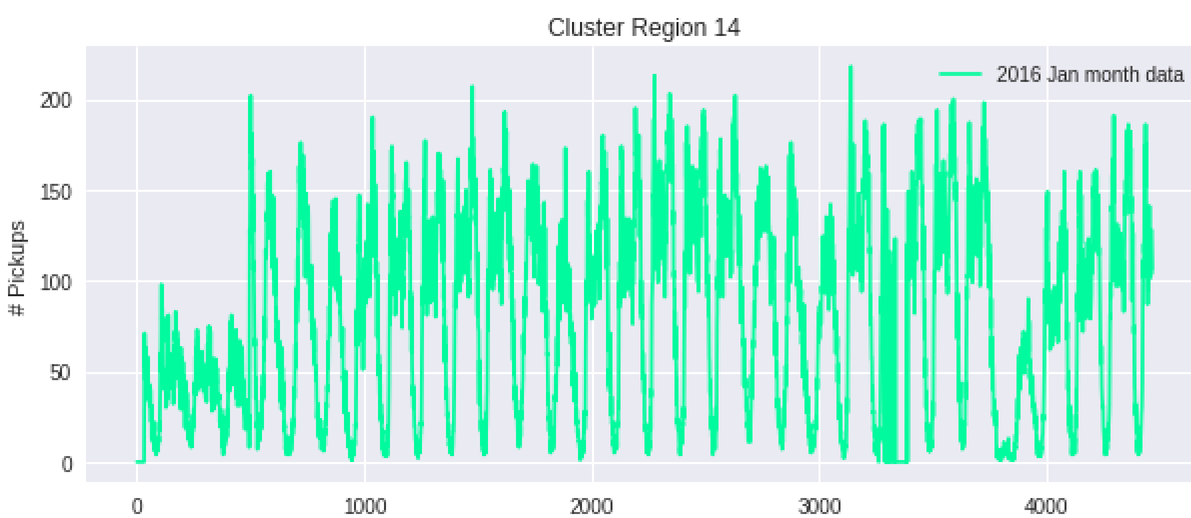

Cluster Region 11
2016 Jan month data
Pickups
700
600
500
400
300
200
100
0
0
1000
2000
3000
4000



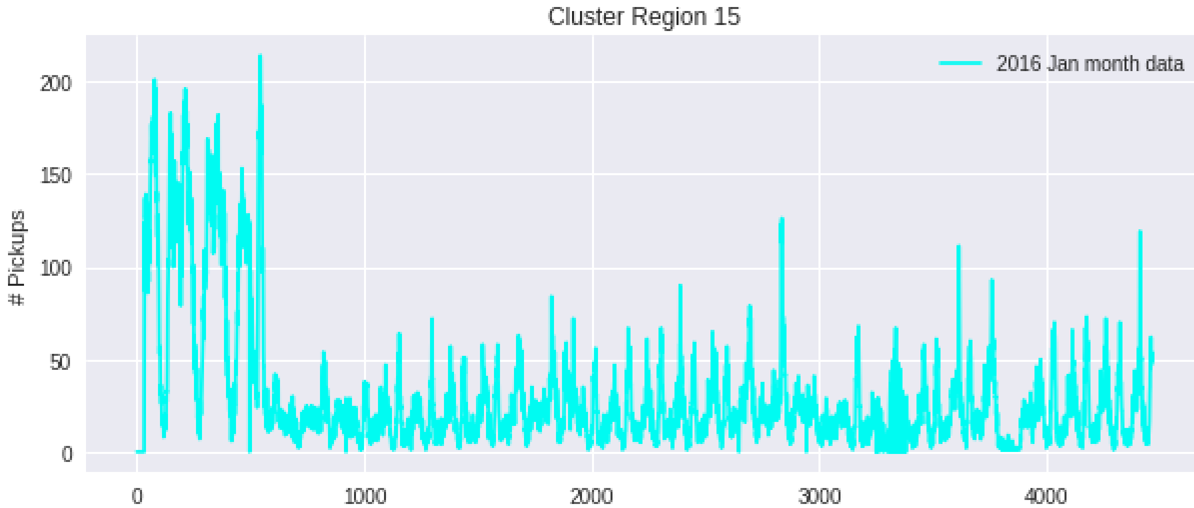
Cluster Region 12
2016 Jan month data
Pickups
0
100
200
300
400
0
1000
2000
3000
4000

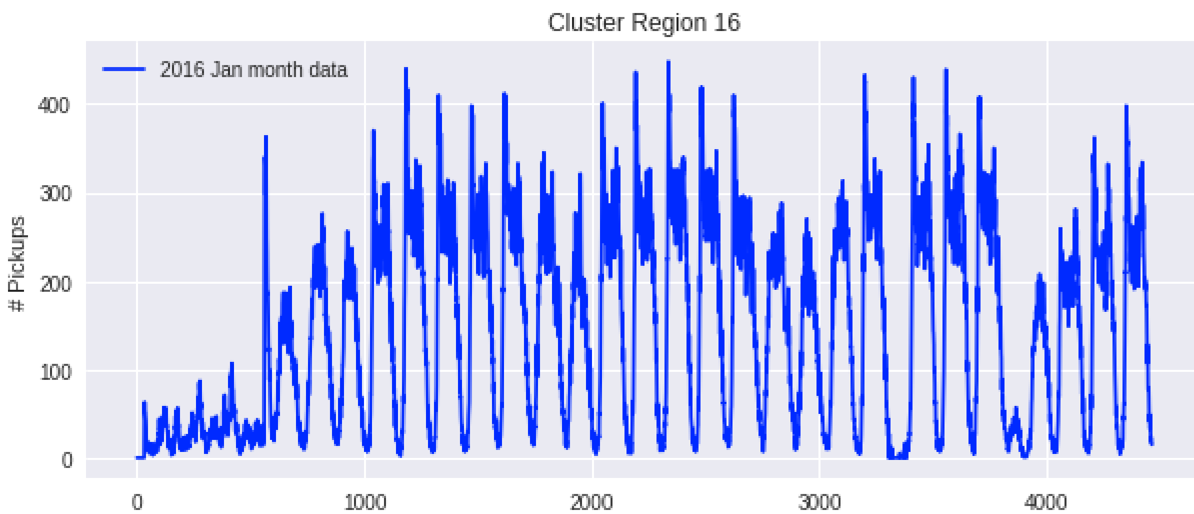


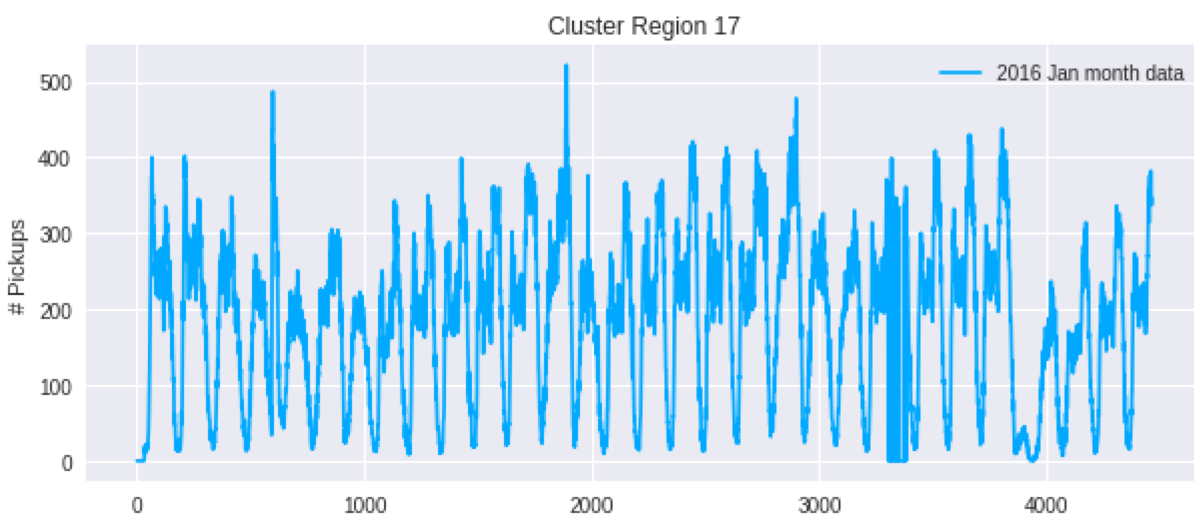
Cluster Region 13
2016 Jan month data
Pickups
0
50
100
150
200
250
300
350
400
0
1000
2000
3000
4000

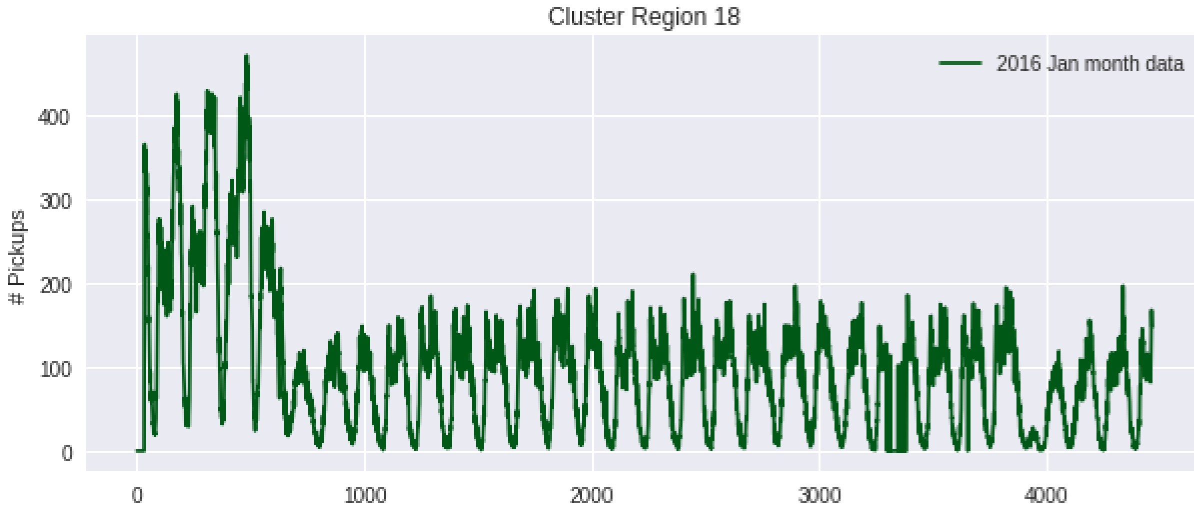


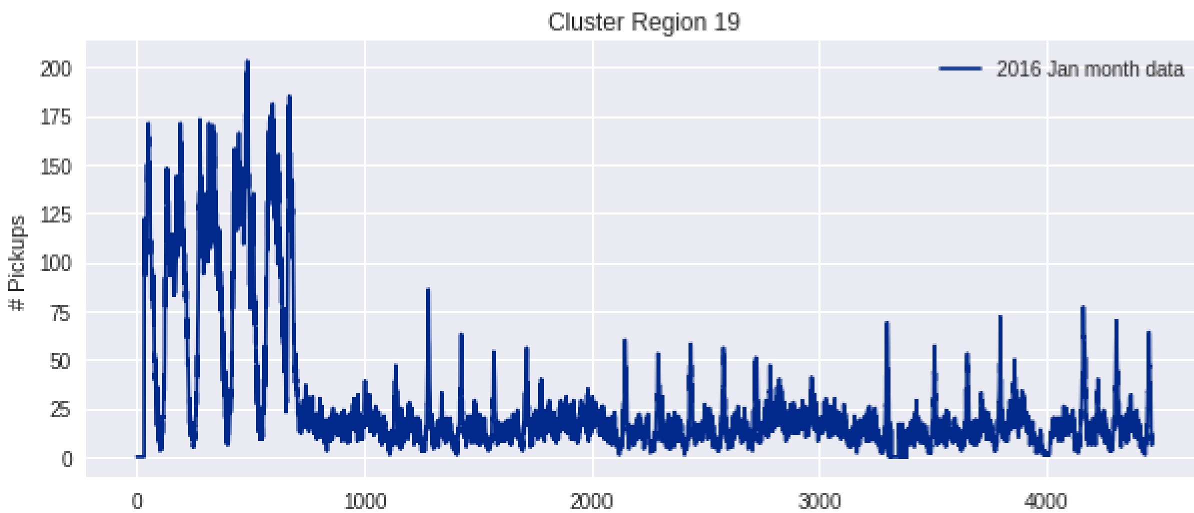
Cluster Region 14
2016 Jan month data
Pickups
0
50
100
150
200
0
1000
2000
3000
4000


Cluster Region 15
2016 Jan month data
Pickups
0
50
100
150
200
0
1000
2000
3000
4000

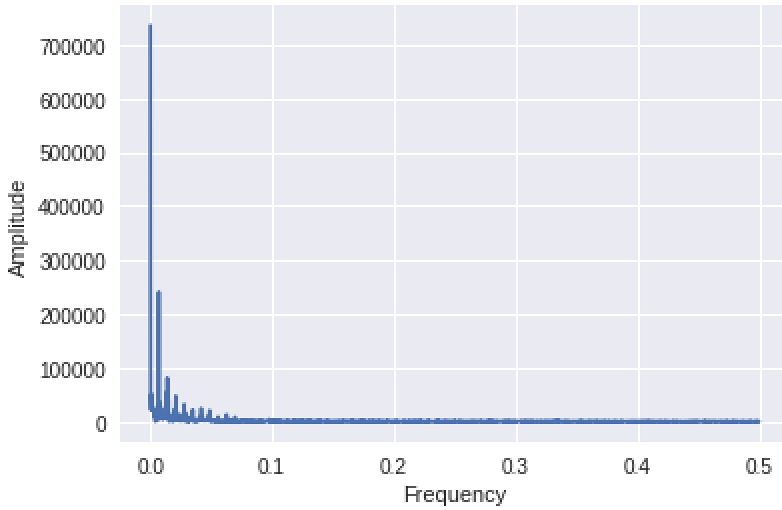

Cluster Region 16
2016 Jan month data
Pickups
0
100
200
300
400
0
1000
2000
3000
4000


Cluster Region 17
2016 Jan month data
Pickups
0
100
200
300
400
500
0
1000
2000
3000
4000

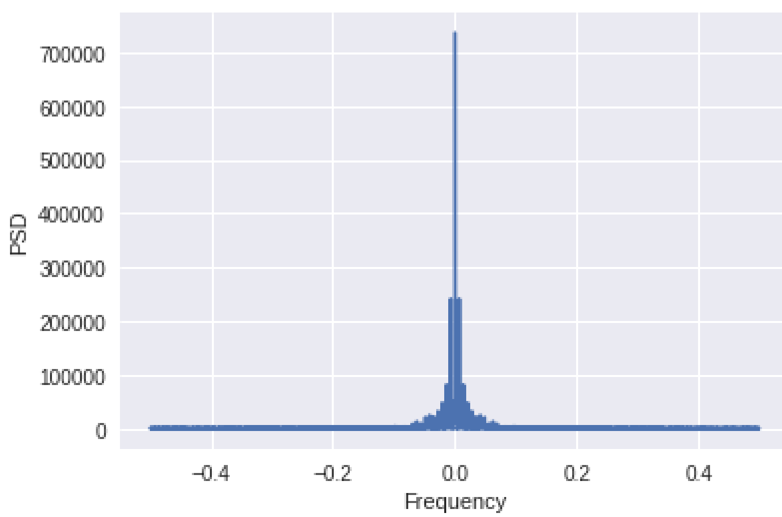

Cluster Region 18
2016 Jan month data
Pickups
0
100
200
300
400
0
1000
2000
3000
4000


Cluster Region 19
2016 Jan month data
Pickups
0
25
50
75
100
125
150
175
200
0
1000
2000
3000
4000

In [0]:
```
# 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/generate
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 [0]:
```
# 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]:
```
# 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


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

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


# tsne_lat will contain 13104-5=13099 times lattitude of cluster center for every
# Ex: [[cent_lat 13099times],[cent_lat 13099times], [cent_lat 13099times].... 20
# it is list of lists
tsne_lat = []


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

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

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


tsne_feature = [0]*number_of_time_stamps
for i in range(0,20):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*4459)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*4459)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number o
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464)])
    # three_month_pickups_2016 is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3.
    tsne_feature = np.vstack((tsne_feature, [three_month_pickups_2016[i][r:r+numb
    output.append(three_month_pickups_2016[i][5:4464])
tsne_feature = tsne_feature[1:]
```

In [0]:
```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(
```

Out[159]:
```
True
```

In [0]:
```
tsne_feature
```

Out[160]:
```
array([[ 0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0],
       ...,
       [28, 38, 24, 20,  9],
       [38, 24, 20,  9, 11],
       [24, 20,  9, 11,  6]])
```

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

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,20):
    for i in range(0,4464):
        if i==0:
            predicted_value= three_month_pickups_2016[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(three_month_pic
    predict_list.append(predicted_values[5:4464])
    predicted_values=[]
```

Fourier Transform

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

In [0]:
```
# 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
tr_size = int(4459*0.7)
print("size of train data :", tr_size)
print("size of test data :", 4459 - tr_size)
```

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

In [0]:
```
# extracting first 9169 timestamp values i.e 70% of 4459 (total timestamps) for o
train_features =  [tsne_feature[i*4459:(4459*i+3121)] for i in range(0,20)]
test_features = [tsne_feature[(4459*(i))+3121:4459*(i+1)] for i in range(0,20)]
```

In [0]:
```
train_top_freq = [frequency[4459*i:(4459*i+3121)] for i in range(0,20)]
train_top_amp = [amplitude[4459*i:(4459*i+3121)] for i in range(0,20)]
test_top_freq = [frequency[4459*i + 3121: 4459*(i+1)] for i in range(0,20)]
test_top_amp = [amplitude[4459*i:4459*(i+1)] for i in range(0,20)]
```

In [0]:
```
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 =  20
Number of data points 3121
 Each data point contains 5 features

Test data # Regions =  20
Number of data points in test data 1338
Each data point contains 5 features
```

In [0]:
```
tsne_train_flat_lat = [i[:3121] for i in tsne_lat]
tsne_train_flat_lon = [i[:3121] for i in tsne_lon]
tsne_train_flat_weekday = [i[:3121] for i in tsne_weekday]
tsne_train_flat_output = [i[:3121] for i in output]
tsne_train_flat_exp_avg = [i[:3121] for i in predict_list]
```

In [0]:
```
tsne_test_flat_lat = [i[3121:] for i in tsne_lat]
tsne_test_flat_lon = [i[3121:] for i in tsne_lon]
tsne_test_flat_weekday = [i[3121:] for i in tsne_weekday]
tsne_test_flat_output = [i[3121:] for i in output]
tsne_test_flat_exp_avg = [i[3121:] for i in predict_list]
```

In [0]:
```
# the above contains values in the form of list of lists (i.e. list of values of 
train_new_features = []
for i in range(0,20):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,20):
    test_new_features.extend(test_features[i])
```

In [0]:
```
(train_top_freq[0][1][0])
```

Out[174]:
```
0.006889763779527559
```

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

In [0]:
```
train_new_freq[0]
```

Out[176]:
```
[0.006889763779527559,
 0.007135826771653543,
 0.014025590551181102,
 0.013779527559055118,
 0.007874015748031496]
```

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

In [0]:
```
train_new_amp[0]
```

Out[178]:
```
[12.229314032345926,
 11.146058795074413,
 11.016417768361388,
 10.87650917769881,
 10.870259914445603]
```

In [0]:
```
columns_frequency = ['f1','f2','f3','f4','f5']
columns_amplitude = ['a1','a2','a3','a4','a5']
df_freq_train = pd.DataFrame(train_new_freq,columns = columns_frequency)
df_amp_train = pd.DataFrame(train_new_amp,columns = columns_amplitude)
df_freq_test = pd.DataFrame(test_new_freq,columns = columns_frequency)
df_amp_test = pd.DataFrame(test_new_amp,columns = columns_amplitude)
```

In [0]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

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

In [0]:
```
# converting lists of lists into sinle list i.e flatten
# a  = [[1,2,3,4],[4,6,7,8]]
# print(sum(a,[]))
# [1, 2, 3, 4, 4, 6, 7, 8]

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

In [0]:
```
# Preparing the data frame for our train data
columns = ['ft_5','ft_4','ft_3','ft_2','ft_1']
df_train = pd.DataFrame(data=train_new_features, columns=columns)
df_train = pd.merge(df_train,df_freq_train,left_index = True,right_index = True)
df_train = pd.merge(df_train,df_amp_train,left_index = True,right_index = True)
df_train['lat'] = tsne_train_lat
df_train['lon'] = tsne_train_lon

df_train['weekday'] = tsne_train_weekday
df_train['exp_avg'] = tsne_train_exp_avg

print(df_train.shape)
```

```
(62420, 19)
```

In [0]:
```
# final train dataframe
df_train.head()
```

Out[186]:

	ft_5	ft_4	ft_3	ft_2	ft_1	f1	f2	f3	f4	f5	a1	a2
0	0	0	0	0	0	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
1	0	0	0	0	0	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
2	0	0	0	0	0	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
3	0	0	0	0	0	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
4	0	0	0	0	0	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059

In [0]:
```
# Preparing the data frame for our train data
df_test = pd.DataFrame(data=test_new_features, columns=columns)
df_test = pd.merge(df_test,df_freq_test,left_index = True,right_index = True)
df_test = pd.merge(df_test,df_amp_test,left_index = True,right_index = True)
df_test['lat'] = tsne_test_lat
df_test['lon'] = tsne_test_lon
df_test['weekday'] = tsne_test_weekday
df_test['exp_avg'] = tsne_test_exp_avg
print(df_test.shape)
```

```
(26760, 19)
```

In [0]:
```
# final test dataframe
df_test.head()
```

Out[188]:

	ft_5	ft_4	ft_3	ft_2	ft_1	f1	f2	f3	f4	f5	a1	a2
0	248	256	218	248	263	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
1	256	218	248	263	273	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
2	218	248	263	273	262	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
3	248	263	273	262	266	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059
4	263	273	262	266	251	0.00689	0.007136	0.014026	0.01378	0.007874	12.229314	11.146059

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

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

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

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

Using Linear Regression

In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error , mean_absolute_error
import warnings
warnings.filterwarnings("ignore")

# Standardization
train_std = StandardScaler().fit_transform(df_train)
test_std = StandardScaler().fit_transform(df_test)

# Hyperparameter tuning using gridsearch
values = [10**-8, 10**-6, 10**-4, 10**-2, 10**0, 10**2, 10**4, 10**6, 10**8]
params = {"alpha": values}
sgd = SGDRegressor(loss="squared_loss", penalty = "l2")
gsv = GridSearchCV(sgd, params, scoring = 'neg_mean_absolute_error', cv = 3)
gsv.fit(train_std, tsne_train_output)
alpha  = gsv.best_params_['alpha']
#Optimal alpha
print(alpha)
```

```
0.01
```

In [0]:
```
# fitting model using best hyperparameter
lr_reg = SGDRegressor(alpha=alpha, loss='squared_loss')
lr_reg.fit(train_std,tsne_train_output)

# prediction
y_pred = lr_reg.predict(test_std)
test_predictions = [round(value) for value in y_pred]
y_pred = lr_reg.predict(train_std)
train_predictions = [round(value) for value in y_pred]
```

In [0]:
```
# Calculating the error metric values
train_mse_sgd = mean_squared_error(tsne_train_output,train_predictions)
train_mpe_sgd = mean_absolute_error(tsne_train_output,train_predictions)/(sum(tsn
test_mse_sgd = mean_squared_error(tsne_test_output,test_predictions)
test_mpe_sgd = mean_absolute_error(tsne_test_output,test_predictions)/(sum(tsne_t

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

```
10.867323065655603
15.71258550442399
```

Using Random Forest Regressor

In [0]:
```
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

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

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

print(n_estimator)
print(max_depth)
```

```
435
```

In [0]:
```
# fitting model with optimal hyperparameter
rf = RandomForestRegressor(n_estimators=n_estimator, max_depth= max_depth)
rf.fit(df_train, tsne_train_output)

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

In [0]:
```
# Calculating the error metric values
train_mse_rf = mean_squared_error(tsne_train_output, y_pred_train )
train_mpe_rf = mean_absolute_error(tsne_train_output, y_pred_train )/(sum(tsne_tr
test_mse_rf = mean_squared_error(tsne_test_output, y_pred_test )
test_mpe_rf = mean_absolute_error(tsne_test_output, y_pred_test )/(sum(tsne_test_

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

```
9.311795898596177
12.226112523256715
```

In [0]:
```
#feature importances
print (df_train.columns)
print (rf.feature_importances_)
```

```
Index(['ft_5', 'ft_4', 'ft_3', 'ft_2', 'ft_1', 'f1', 'f2', 'f3', 'f4', 'f5',
       'a1', 'a2', 'a3', 'a4', 'a5', 'lat', 'lon', 'weekday', 'exp_avg'],
      dtype='object')
[2.00632867e-03 1.48276500e-03 1.54579290e-03 1.57630338e-03
 3.80817015e-01 0.00000000e+00 3.03742644e-05 1.08978023e-04
 2.05730340e-05 4.68703747e-05 1.01542954e-04 4.35927142e-05
 4.25734046e-05 4.67438076e-05 4.41045338e-05 4.33122727e-04
 6.28952996e-04 4.06519933e-04 6.10617846e-01]
```

Using XgBoost Regressor

In [0]:
```
import xgboost as xgb

# Hyperparameter tuning using gridsearch
param = {'n_estimators':sp_randint(100,1000), 'max_depth':sp_randint(1,10)}
xgb = xgb.XGBRegressor()
rsv = RandomizedSearchCV(xgb, param, scoring = 'neg_mean_absolute_error', cv =3)
rsv.fit(df_train,tsne_train_output)

estimator = rsv.best_params_['n_estimators']
depth = rsv.best_params_['max_depth']

print(estimator)
print(depth)
```

```
136
3
```

In [0]:
```
import xgboost as xgb

# fitting model with optimal hyperparameter
xgb = xgb.XGBRegressor(n_estimators = estimator, max_depth = depth)
xgb.fit(df_train,tsne_train_output)

y_pred_test = xgb.predict(df_test)
y_pred_train = xgb.predict(df_train)
```

In [0]:
```
# Calculating the error metric values
train_mse_xgb = mean_squared_error(tsne_train_output,y_pred_train )
train_mpe_xgb = mean_absolute_error(tsne_train_output,y_pred_train)/(sum(tsne_tra
test_mse_xgb = mean_squared_error(tsne_test_output,y_pred_test)
test_mpe_xgb = mean_absolute_error(tsne_test_output,y_pred_test)/(sum(tsne_test_o

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

```
10.525021348579894
12.355185546923463
```

In [0]:
```
#feature importances based on analysis using xgboost
xgb.feature_importances_
```

Out[16]:
```
array([0.16384777, 0.08562368, 0.09090909, 0.14270613, 0.24524313,
       0.        , 0.00739958, 0.0179704 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.02748414, 0.04439746, 0.01374207, 0.16067654], dtype=float32)
```

Calculating the error metric values for various models

In [0]:
```
train_mape=[]
test_mape=[]

train_mape.append((mean_absolute_error(tsne_train_output,df_train['ft_1'].values)
train_mape.append((mean_absolute_error(tsne_train_output,df_train['exp_avg'].valu

test_mape.append((mean_absolute_error(tsne_test_output, df_test['ft_1'].values))/
test_mape.append((mean_absolute_error(tsne_test_output, df_test['exp_avg'].values
```

Error Metric Matrix

In [0]:
```
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(%):  11.122132292003503
Test(%):  12.363389587682795
Exponential Averages Forecasting -           Train(%):  10.82730417781128
Test(%):  12.006055502245257
Linear Regression -                          Train(%):  10.867323065655603
Test(%):  15.71258550442399
Random Forest Regression -                   Train(%):  9.311795898596177
Test(%):  12.226112523256715
XgBoost Regression -                         Train(%):  10.525021348579894
Test(%):  12.355185546923463
--------------------------------------------------------------------------------
-------------------------
```

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 9.31% and with test MAPE 12.22
3. Tree based models are outperforming other models slightly.

In [0]:

```

```