Taxi demand prediction in New York City

In [1]:

```
!pip install gpxpy
```

```
Collecting gpxpy
  Downloading
https://files.pythonhosted.org/packages/dd/23/a1c04fb3ea8d57d4b46cf2956c99a62dfbe009bbe091babeef90c
ef6/gpxpy-1.4.2.tar.gz (105kB)
     |████████████████████████████████| 112kB 3.5MB/s
Building wheels for collected packages: gpxpy
  Building wheel for gpxpy (setup.py) ... done
  Created wheel for gpxpy: filename=gpxpy-1.4.2-cp36-none-any.whl size=42546
sha256=7fdbd385b9132685e59629253887021c4d555585cd6a4c369c8a7408dcc0640b
  Stored in directory:
/root/.cache/pip/wheels/d9/df/ed/b52985999b3967fa0ef8de22b3dc8ad3494ce3380d5328dd0f
Successfully built gpxpy
Installing collected packages: gpxpy
Successfully installed gpxpy-1.4.2
```

In [2]:

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

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

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

import time #Convert to unix time

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

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

# this lib is used while we calculate the stight line distance between two (lat,lon) pairs in mile
s

import gpxpy.geo #Get the haversine distance

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

# download migwin: https://mingw-w64.org/doku.php/download/mingw-builds
# install it in your system and keep the path, migw path ='installed path'
```

```
#mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev0\\mingw64\\bin'
#os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

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

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Data Information

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

Information on taxis:

Yellow Taxi: Yellow Medallion Taxicabs

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

For Hire Vehicles (FHVs)

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

Green Taxi: Street Hail Livery (SHL)

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

Credits: Quora

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

Data Collection

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

file name	file name size	number of records	number of features
yellow_tripdata_2016-01	1. 59G	10906858	19
yellow_tripdata_2016-02	1. 66G	11382049	19
yellow_tripdata_2016-03	1. 78G	12210952	19
yellow_tripdata_2016-04	1. 74G	11934338	19
yellow_tripdata_2016-05	1. 73G	11836853	19
yellow_tripdata_2016-06	1. 62G	11135470	19
yellow_tripdata_2016-07	884Mb	10294080	17
yellow_tripdata_2016-08	854Mb	9942263	17
yellow_tripdata_2016-09	870Mb	10116018	17

yellow_tripdata_2016-10	933Mb	10854626	17
yellow_tripdata_2016-11	868Mb	10102128	17
yellow_tripdata_2016-12	897Mb	10449408	17
yellow_tripdata_2015-01	1.84Gb	12748986	19
yellow_tripdata_2015-02	1.81Gb	12450521	19
yellow_tripdata_2015-03	1.94Gb	13351609	19
yellow_tripdata_2015-04	1.90Gb	13071789	19
yellow_tripdata_2015-05	1.91Gb	13158262	19
yellow_tripdata_2015-06	1.79Gb	12324935	19
yellow_tripdata_2015-07	1.68Gb	11562783	19
yellow_tripdata_2015-08	1.62Gb	11130304	19
yellow_tripdata_2015-09	1.63Gb	11225063	19
yellow_tripdata_2015-10	1.79Gb	12315488	19
yellow_tripdata_2015-11	1.65Gb	11312676	19
yellow_tripdata_2015-12	1.67Gb	11460573	19

In [3]:

```
from google.colab import drive
drive.mount('/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /gdrive
```

In [4]:

```
#Looking at the features
# dask dataframe  : # https://github.com/dask/dask-tutorial/blob/master/07_dataframe.ipynb
month = dd.read_csv('/gdrive/My Drive/TaxiPrediction/Copy of yellow_tripdata_2015-01.csv')
print(month.columns) #reading the coloumns in our dataframe
```

```
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 [5]:

```
month.dtypes
```

Out[5]:

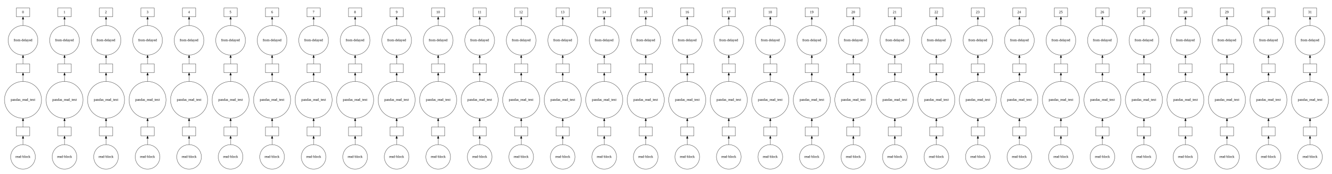
```
VendorID                  int64
tpep_pickup_datetime     object
tpep_dropoff_datetime    object
passenger_count           int64
trip_distance           float64
pickup_longitude        float64
pickup_latitude         float64
RateCodeID                int64
store_and_fwd_flag       object
dropoff_longitude       float64
dropoff_latitude        float64
payment_type              int64
fare_amount             float64
```

```
extra                        float64
mta_tax                      float64
tip_amount                   float64
tolls_amount                 float64
improvement_surcharge        float64
total_amount                 float64
dtype: object
```

In [6]:

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

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

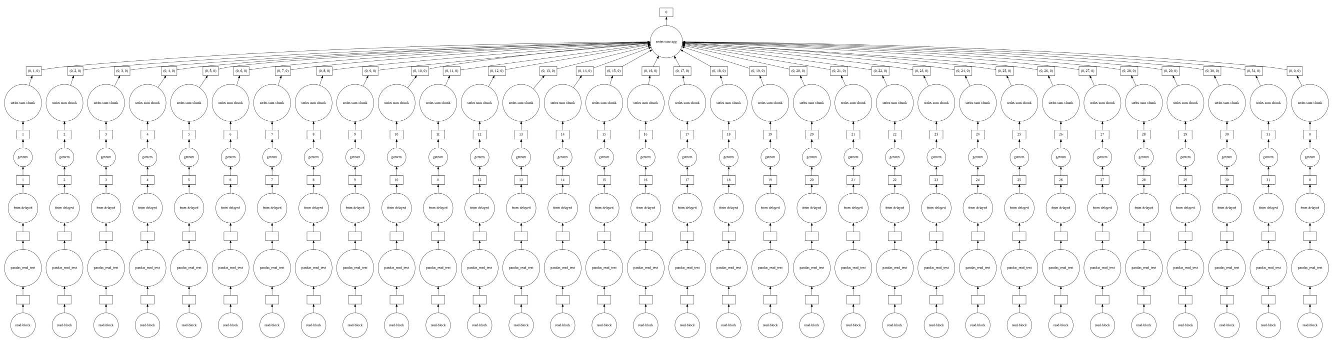
Out[6]:



In [7]:

```
month.fare_amount.sum().visualize() #this is to show how a certain operation like sum is performed using dask
```

Out[7]:



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 [8]:

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

Out[8]:

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

Pickup Latitude and Pickup Longitude

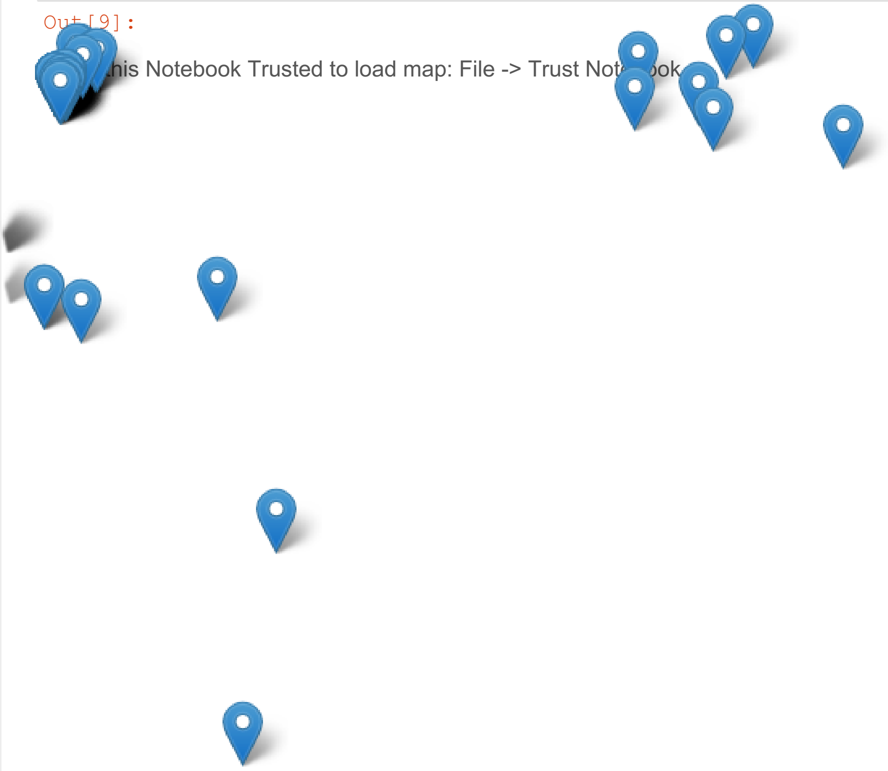
It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bounded by the location cordinates(lat,long) - **(40.5774, -74.15)** & **(40.9176,-73.7004)** so hence any cordinates not within these cordinates are not considered by us as we are only concerned with pickups which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

Removing all those areas which are outside the new york area using lat,longitude

In [9]:

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

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

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

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

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

Out[9]:



Observation:- As you can see above that there are some points just outside the boundary but there are also some points which are outrageous like some points are in **water** some are near **cuba** and **colombia** South America and these are certainly outliers

Dropoff Latitude & Dropoff Longitude

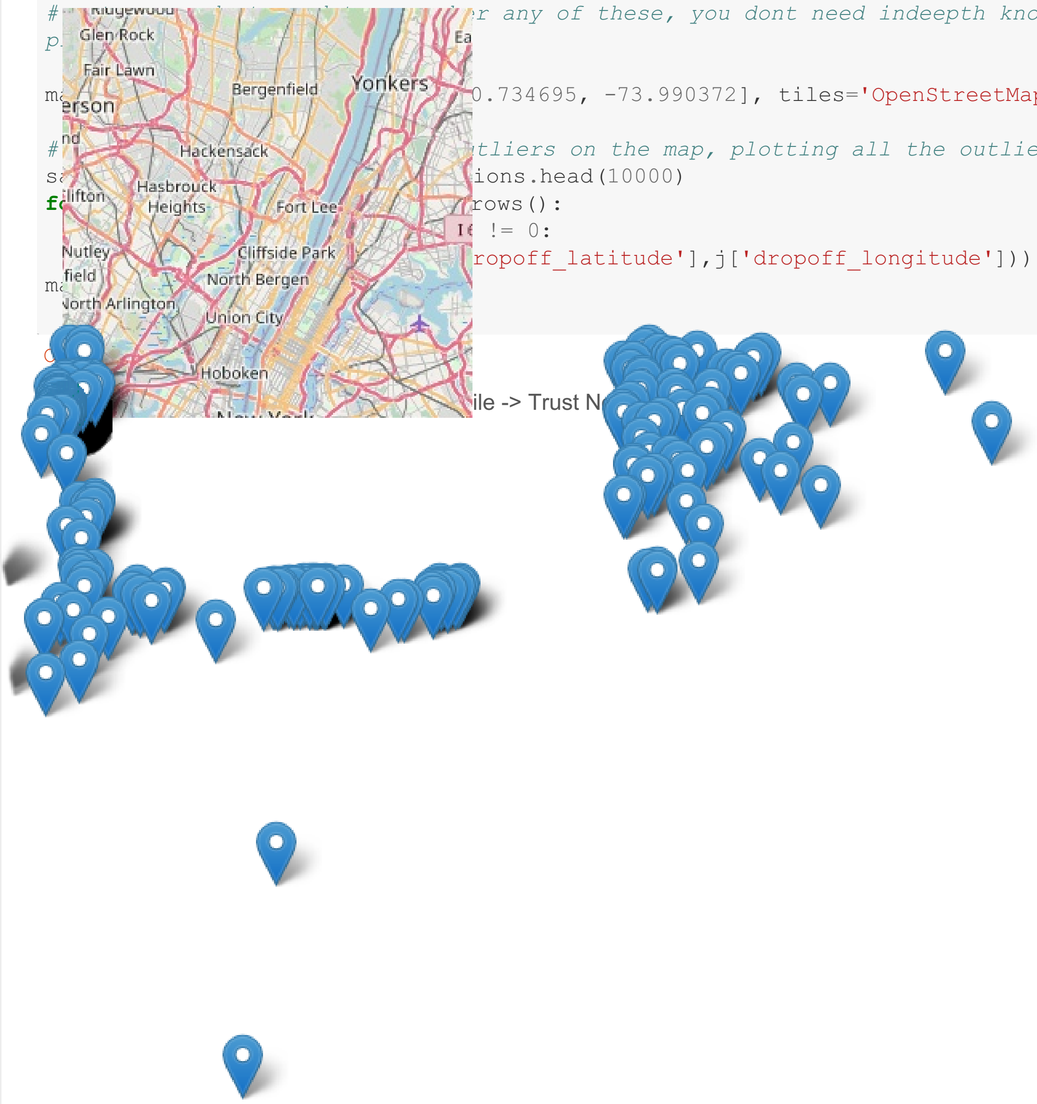
It is inferred from the source https://www.flickr.com/places/info/2459115 that New York is bounded by the location cordinates(lat,long) - (40.5774, -74.15) & (40.9176,-73.7004) so hence any cordinates not within these cordinates are not considered by us as we are only concerned with dropoffs which are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

In [10]:

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

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

```
# ...r any of these, you dont need indeepth knowledge on these maps and
p...

ma...0.734695, -73.990372], tiles='OpenStreetMap')

#...tliers on the map, plotting all the outliers will take more time
s...ions.head(10000)
fo...rows():
    ... != 0:
        ...ropoff_latitude'],j['dropoff_longitude']))).add_to(map_osm)
ma...
```

...ile -> Trust N...



Observation:- The observations here are similar to those obtained while analysing pickup latitude and longitude..some points are outraegous like some in near **spain** some in **sea** etc

Trip Durations(in miles) :

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

In [11]:

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

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

#trip duration = driopoff time -  pickup time.. since time is in YYYYY/MM/DD/H/Min/S there we need
to convert it into desired format
#therefore we are using convert_to_unix function which converts it into unix timestamp

# 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
```

```python
# 5.'dropoff_longitude' : self explanatory
# 6.'dropoff_latitude' : self explanatory
# 7.'total_amount' : total fair that was paid
# 8.'trip_times' : duration of each trip
# 9.'pickup_times : pickup time converted into unix time
# 10.'Speed' : velocity of each trip
def return_with_trip_times(month):
    duration = month[['tpep_pickup_datetime','tpep_dropoff_datetime']].compute()
    #what is dask.compute - https://distributed.dask.org/en/latest/manage-computation.html
    #pickups and dropoffs to unix time
    duration_pickup = [convert_to_unix(x) for x in duration['tpep_pickup_datetime'].values] #passin
g to function to convert the time into unix
    duration_drop = [convert_to_unix(x) for x in duration['tpep_dropoff_datetime'].values]
    #calculate duration of trips
    durations = (np.array(duration_drop) - np.array(duration_pickup))/float(60)#dividing by 60 to g
et the time in mins

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

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

    return new_frame

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

In [12]:

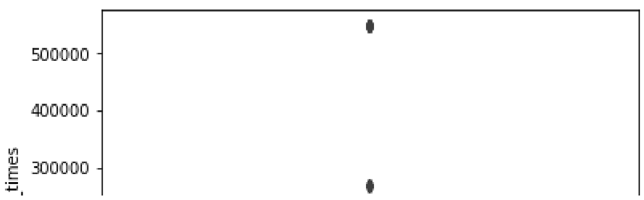
```python
frame_with_durations.head()
```

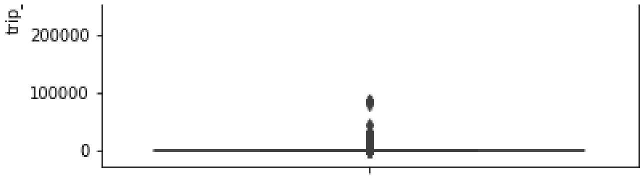
Out[12]:

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

In [13]:

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





We can't gain much informaation from the above box plot as all the percentile values are very close to each other so now will see the percentile values numerically

In [14]:

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

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

we can the that 0th percentile value is negative which is impossible and also the 100th percentile value, now investigate between **0th to 10th** and **90th to 100th**

In [15]:

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

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

In [16]:

```
#looking further from the 99th percecntile
for i in range(0,11):
    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 ("10 percentile value is ",var[9])
```

```
0 percentile value is -1211.0166666666667
1 percentile value is 1.2166666666666666
2 percentile value is 1.8833333333333333
3 percentile value is 2.2666666666666666
4 percentile value is 2.5833333333333335
5 percentile value is 2.8333333333333335
```

```
6 percentile value is 3.066666666666667
7 percentile value is 3.2666666666666666
8 percentile value is 3.466666666666667
9 percentile value is 3.65
10 percentile value is 3.8333333333333335
```

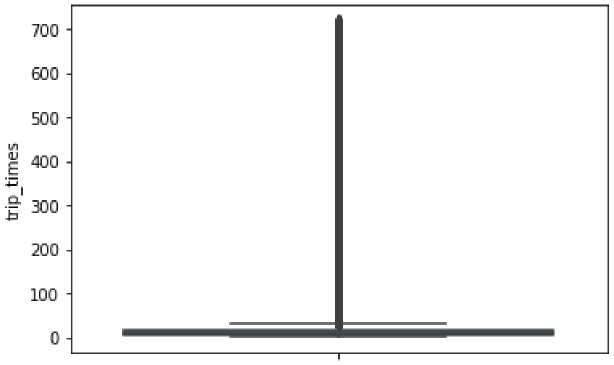
0th percentile value and the 100th percentile value are outliers

In [17]:

```
#removing data based on our analysis and TLC regulations
#as the maximum allowed duration is less than 720mins therefore we are checking for values beyond allowed range
frame_with_durations_modified=frame_with_durations[(frame_with_durations.trip_times>1) & (frame_with_durations.trip_times<720)]
```

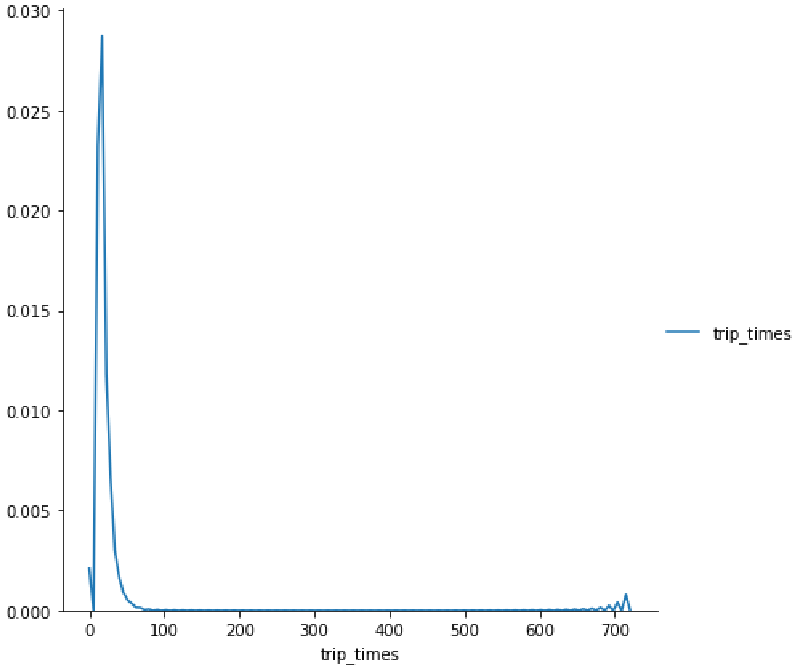
In [18]:

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



In [19]:

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



The distribution of the trip-duration looks somewhat to **log-normal** .. let's see if it's really a log-normal distribution or not.

Why we want to check if its a log-normnal or not? Ans- If we somehow find the it is a log-normal distribution then taking a log of these features will be useful because algorithms like logistic regression tends to perform better when the featues are normally distributed

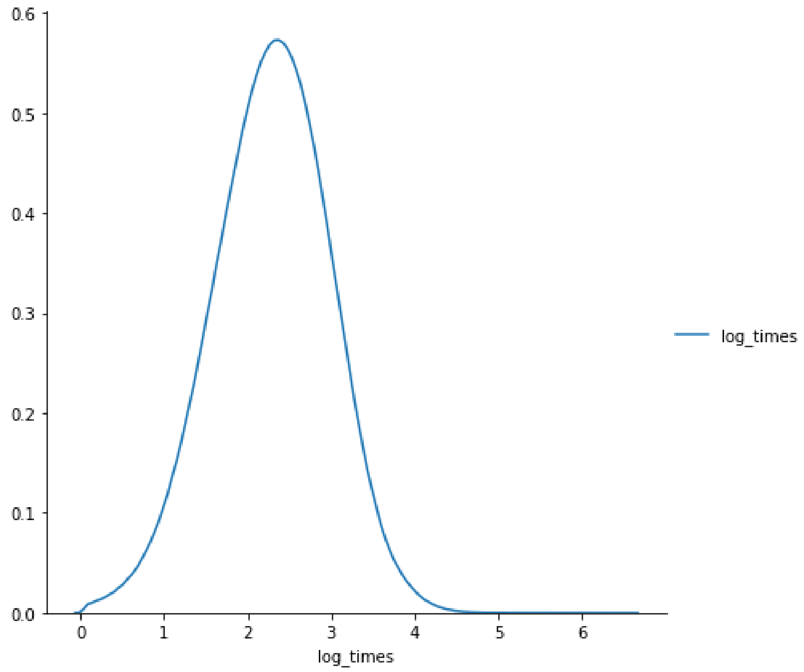
We can see that some points are beyond the 700 limit

In [20]:

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

In [21]:

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

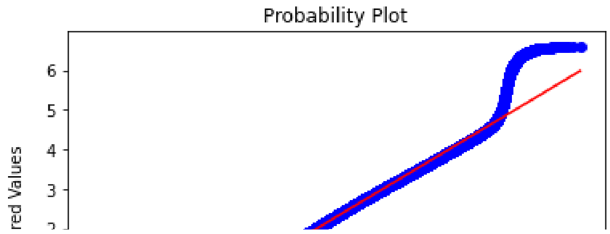


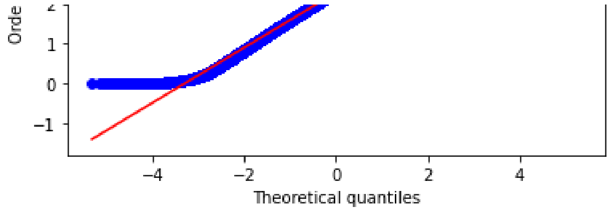
The curve looks somewhat same to normal distribution except the right tale which is long so we plot a **Q-Q** plot to find out.

A q-q plot is a plot of the quantiles of the first data set against the quantiles of the second data set. By a quantile, we mean the fraction (or percent) of points below the given value. ... If the two sets come from a population with the same distribution, the points should fall approximately along this reference line.

In [22]:

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



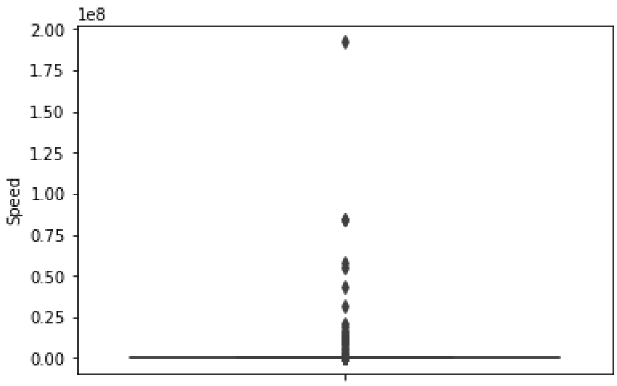


We can see that beyond the **-3 and +3** standard deviation the plot is not behaving like a gaussian plot and hence we can say that its not a gaussian distributon

Speed(miles/hr)

In [23]:

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



In [24]:

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

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

In [25]:

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

```
90 percentile value is 20.186915887850468
```

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

In [26]:

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

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

Speed with 192857142 is certainly an outlier

In [27]:

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

In [28]:

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

Out[28]:

```
12.450173996027528
```

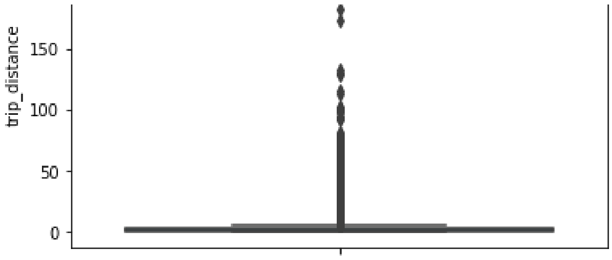
The avg speed in NewYork speed is 12.45miles/hr, so a cab driver can travel 2 miles per 10min on avg.

The reason we choose 10mins bin beacuse we need a distance which the driver can go with much of trouble i.e if we choose the distance to large , driver might not want to go that far to take the pickups

Trip Distance(in miles)

In [29]:

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



In [30]:

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

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

In [31]:

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

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

In [32]:

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

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

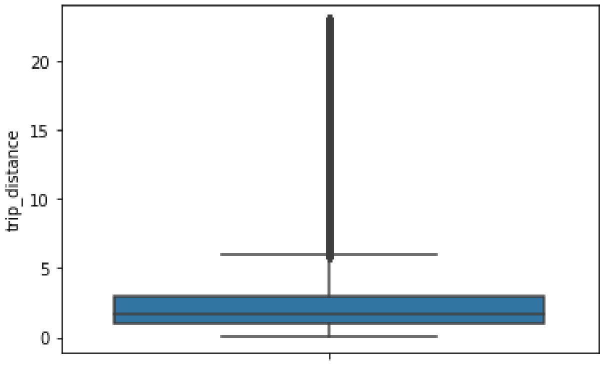
```
99.9 percentile value is 22.57
100 percentile value is  258.9
```

In [33]:

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

In [34]:

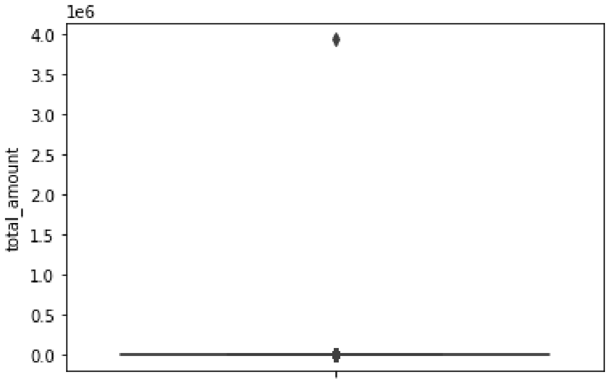
```
#box-plot after removal of outliers
sns.boxplot(y="trip_distance", data = frame_with_durations_modified)
plt.show()
```



Total Fare(in Dollars)

In [35]:

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



In [36]:

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

```
0 percentile value is -242.55
10 percentile value is 6.3
20 percentile value is 7.8
30 percentile value is 8.8
40 percentile value is 9.8
50 percentile value is 11.16
```

```
60 percentile value is 12.8
70 percentile value is 14.8
80 percentile value is 18.3
90 percentile value is 25.8
100 percentile value is  3950611.6
```

100th percentile value of **3950611** is certainly an outlier becuase no one will pay this much of amount for a trip

In [37]:

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

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

In [38]:

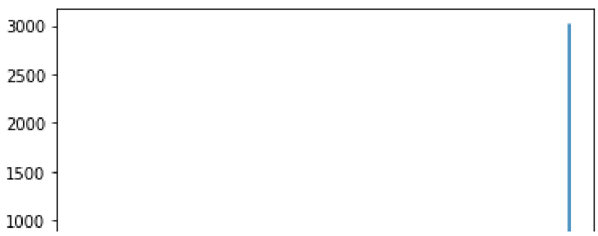
```
#calculating total fare amount values at each percntile
99.0,99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100
for i in np.arange(0.0, 1.0, 0.1):
    var = frame_with_durations_modified["total_amount"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))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

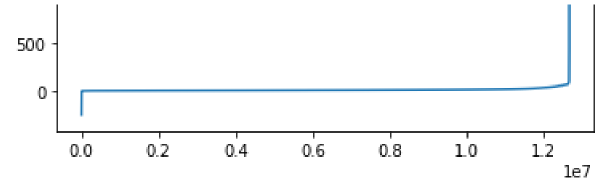
```
99.0 percentile value is 66.13
99.1 percentile value is 68.13
99.2 percentile value is 69.6
99.3 percentile value is 69.6
99.4 percentile value is 69.73
99.5 percentile value is 69.75
99.6 percentile value is 69.76
99.7 percentile value is 72.58
99.8 percentile value is 75.35
99.9 percentile value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

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

In [39]:

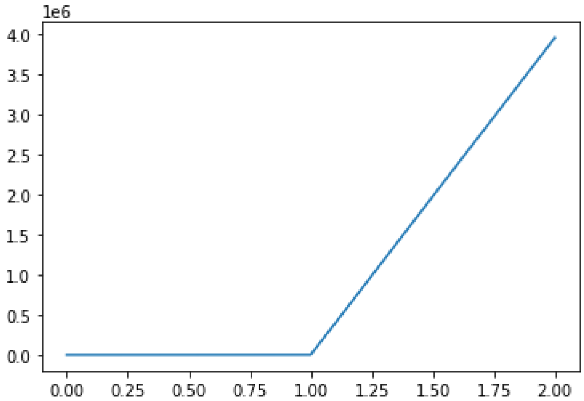
```
#below plot shows us the fare values(sorted) to find a sharp increase to remove those values as ou
tliers
# plot the fare amount excluding last two values in sorted data
plt.plot(var[:-2])
plt.show()
```





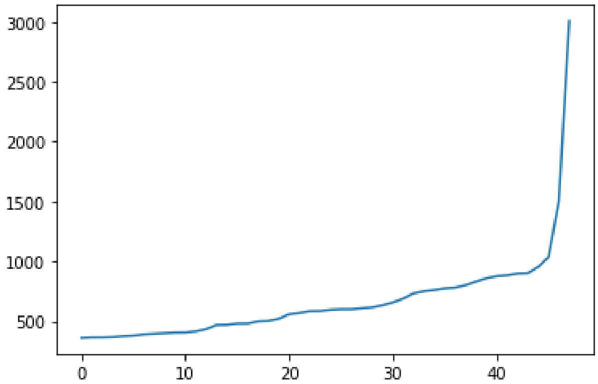
In [40]:

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



In [41]:

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



passenger_count

In [42]:

```
frame_with_durations_modified.passenger_count.value_counts().sort_index()
```

Out[42]:

```
0        6209
1     8920254
2     1804243
3      525701
4      251957
5      695112
6      452899
7           7
8           3
```

```
9           4
Name: passenger_count, dtype: int64
```

what's the point of having 0 passenger and also the maximum allowed passengers in U.S is 7 and there we will remove those which are beyond those https://www.tripadvisor.in/ShowTopic-g60763-i5-k1461038-How_many_passengers_can_REALLY_fit_into_a_taxi-New_York_City_New_York.html

In [43]:

```
frame_with_durations_modified =
frame_with_durations_modified[(frame_with_durations_modified.passenger_count>0) &
(frame_with_durations_modified.passenger_count<7)]
```

In [44]:

```
frame_with_durations_modified.head()
```

Out[44]:

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

Remove all outliers/erronous points.

In [45]:

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


    a = new_frame.shape[0] #first we find the no. of rows before removal of outliers
    print ("Number of pickup records = ",a)
    temp_frame = new_frame[((new_frame.dropoff_longitude >= -74.15) & (new_frame.dropoff_longitude
<= -73.7004) &\
                       (new_frame.dropoff_latitude >= 40.5774) & (new_frame.dropoff_latitude <=
40.9176)) & \
                       ((new_frame.pickup_longitude >= -74.15) & (new_frame.pickup_latitude >=
40.5774)& \
                       (new_frame.pickup_longitude <= -73.7004) & (new_frame.pickup_latitude <=
40.9176))] #removed all those points which are outside the nyc lat, long
    b = temp_frame.shape[0] #caluclating no of rows left
    print ("Number of outlier coordinates lying outside NY boundaries:",(a-b)) #printing rows remov
ed using (orignial-new)


    temp_frame = new_frame[(new_frame.trip_times > 0) & (new_frame.trip_times < 720)] #removing poi
nts which are outside the
    c = temp_frame.shape[0]
    print ("Number of outliers from trip times analysis:",(a-c))


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

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

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

```
    temp_frame = new_frame[(new_frame.passenger_count >0) & (new_frame.passenger_count <7)]
    g = temp_frame.shape[0]
    print ("Number of outliers from passenger count:",(a-g))


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

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

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

In [46]:

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

```
Removing outliers in the month of Jan-2015
----
Number of pickup records =  12748986
Number of outlier coordinates lying outside NY boundaries: 293919
Number of outliers from trip times analysis: 23889
Number of outliers from trip distance analysis: 92597
Number of outliers from speed analysis: 24473
Number of outliers from fare analysis: 5275
Number of outliers from passenger count: 6595
Total outliers removed 377910
---
fraction of data points that remain after removing outliers 0.9703576425607495
```

3% of the data is outliers or errornous and therefore removed

Data-preperation

Clustering/Segmentation

In [47]:

```
#trying different cluster sizes to choose the right K in K-means
coords = frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']].values
#taking all the values within the newyork by using lat lng values
neighbours=[]

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

```
,cluster_centers[j][0], cluster_centers[j][1])
                min_dist = min(min_dist,distance/(1.60934*1000)) # 1 mile = 1.60934 and min dist = 
1000
                if (distance/(1.60934*1000)) <= 2: #checking if the points are within 2km the it cc
mes under nice else wrong
                    nice_points +=1
                else:
                    wrong_points += 1
        less2.append(nice_points)
        more2.append(wrong_points)
    neighbours.append(less2)
    print ("On choosing a cluster size of ",cluster_len,"\nAvg. Number of Clusters within the vici
nity (i.e. intercluster-distance < 2):", np.ceil(sum(less2)/len(less2)), "\nAvg. Number of 
Clusters outside the vicinity (i.e. intercluster-distance > 2):", np.ceil(sum(more2)/len(more2)),"
\nMin inter-cluster distance = ",min_dist,"\n---")

def find_clusters(increment):
    #k-means finds clusters of same size(no of points)
    #more pickups cluster size smaller and vice-versa because we want to have same probabiltiy of
pickups
    kmeans = MiniBatchKMeans(n_clusters=increment, batch_size=10000,random_state=42).fit(coords)
    frame_with_durations_outliers_removed['pickup_cluster'] = 
kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    cluster_centers = kmeans.cluster_centers_
    cluster_len = len(cluster_centers)
    return cluster_centers, cluster_len

# we need to choose number of clusters so that, there are more number of cluster regions
#that are close to any cluster center
# and make sure that the minimum inter cluster should not be very less
for increment in range(10, 100, 10): #taking 10-10 values
    cluster_centers, cluster_len = find_clusters(increment) #finding cluster centers(centroids) anc
length of each cluster of every cluster size of 10 till 100
    find_min_distance(cluster_centers, cluster_len)
```

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

```
Avg. Number of Clusters within the vicinity (i.e. intercluster-distance < 2): 21.0
Avg. Number of Clusters outside the vicinity (i.e. intercluster-distance > 2): 69.0
Min inter-cluster distance =  0.18257992857033273
---
```

Inference:

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

In [48]:

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

# Getting 40 clusters using the kmeans
kmeans = MiniBatchKMeans(n_clusters=40, batch_size=10000,random_state=0).fit(coords)
frame_with_durations_outliers_removed['pickup_cluster'] =
kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
```

Plotting the cluster centers:

In [49]:

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

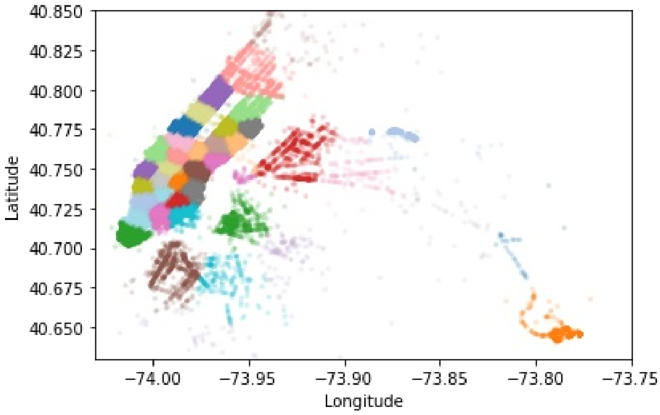
Out[49]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Plotting the clusters:

In [50]:

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

plot_clusters(frame_with_durations_outliers_removed)
```



Time-binning

In [51]:

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

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

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

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

In [52]:

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

```
bins']).count()
```

In [53]:

```
# 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[53]:

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

In [54]:

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

Out[54]:

		trip_distance
pickup_cluster	**pickup_bins**	
0	**33**	104
	34	200
	35	208
	36	141
	37	155

In [55]:

```
jan_2015_frame.shape
```

Out[55]:

```
(12371076, 12)
```

In [56]:

```
# 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_cluster'] =
kmeans.predict(frame_with_durations_outliers_removed[['pickup_latitude', 'pickup_longitude']])
    #frame_with_durations_outliers_removed_2016['pickup_cluster'] =
kmeans.predict(frame_with_durations_outliers_removed_2016[['pickup_latitude',
'pickup_longitude']])

    print ("Final groupbying..")
    final_updated_frame = add_pickup_bins(frame_with_durations_outliers_removed,month_no,year_no)
    final_groupby_frame = final_updated_frame[['pickup_cluster','pickup_bins','trip_distance']].gro
upby(['pickup_cluster','pickup_bins']).count()

    return final_updated_frame,final_groupby_frame

month_jan_2016 = dd.read_csv('/gdrive/My Drive/TaxiPrediction/Copy of yellow_tripdata_2016-01.csv'
)
month_feb_2016 = dd.read_csv('/gdrive/My Drive/TaxiPrediction/Copy of yellow_tripdata_2016-02.csv'
)
month_mar_2016 = dd.read_csv('/gdrive/My Drive/TaxiPrediction/Copy of yellow_tripdata_2016-03.csv'
)

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

```
Return with trip times..
Remove outliers..
Number of pickup records =  10906858
Number of outlier coordinates lying outside NY boundaries: 214677
Number of outliers from trip times analysis: 27190
Number of outliers from trip distance analysis: 79742
Number of outliers from speed analysis: 21047
Number of outliers from fare analysis: 4991
Number of outliers from passenger count: 591
Total outliers removed 297784
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  11382049
Number of outlier coordinates lying outside NY boundaries: 223161
Number of outliers from trip times analysis: 27670
Number of outliers from trip distance analysis: 81902
Number of outliers from speed analysis: 22437
Number of outliers from fare analysis: 5476
Number of outliers from passenger count: 577
Total outliers removed 308177
---
Estimating clusters..
Final groupbying..
Return with trip times..
Remove outliers..
Number of pickup records =  12210952
Number of outlier coordinates lying outside NY boundaries: 232444
Number of outliers from trip times analysis: 30868
Number of outliers from trip distance analysis: 87318
Number of outliers from speed analysis: 23889
Number of outliers from fare analysis: 5859
Number of outliers from passenger count: 678
Total outliers removed 324635
---
Estimating clusters..
Final groupbying..
```

Smoothing

In [57]:

```
jan_2015_frame.head()
```

Out[57]:

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

In [ ]:

```
#x=jan_2015_frame[jan_2015_frame["pickup_cluster"]==2]
#y = list(set(x["pickup_bins"]))
#print(y)
#y.sort()
```

In [58]:

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

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

In [59]:

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

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

#feb
feb_2016_unique = return_unq_pickup_bins(feb_2016_frame)

#march
mar_2016_unique = return_unq_pickup_bins(mar_2016_frame)
```

In [60]:

```
# for each cluster number of 10min intravels with 0 pickups
#4464 = in 1 hr there will be 6 10min intervals and in one day there are 24hrs and for jan we have
31days
#31(days)*24(1day = 24hrs)*6(1hr = 60mins = 6 10min intervals)
for i in range(40):
    print("for the ",i,"th cluster number of 10min intavels with zero pickups: ",4464 -
len(set(jan_2015_unique[i])))
    print('-'*60)
```

```
for the  0 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  1 th cluster number of 10min intavels with zero pickups:  1985
------------------------------------------------------------
for the  2 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  3 th cluster number of 10min intavels with zero pickups:  354
------------------------------------------------------------
```

```
for the  4 th cluster number of 10min intavels with zero pickups:  37
------------------------------------------------------------
for the  5 th cluster number of 10min intavels with zero pickups:  153
------------------------------------------------------------
for the  6 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  7 th cluster number of 10min intavels with zero pickups:  34
------------------------------------------------------------
for the  8 th cluster number of 10min intavels with zero pickups:  117
------------------------------------------------------------
for the  9 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  10 th cluster number of 10min intavels with zero pickups:  25
------------------------------------------------------------
for the  11 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
for the  12 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  13 th cluster number of 10min intavels with zero pickups:  28
------------------------------------------------------------
for the  14 th cluster number of 10min intavels with zero pickups:  26
------------------------------------------------------------
for the  15 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  16 th cluster number of 10min intavels with zero pickups:  40
------------------------------------------------------------
for the  17 th cluster number of 10min intavels with zero pickups:  58
------------------------------------------------------------
for the  18 th cluster number of 10min intavels with zero pickups:  1190
------------------------------------------------------------
for the  19 th cluster number of 10min intavels with zero pickups:  1357
------------------------------------------------------------
for the  20 th cluster number of 10min intavels with zero pickups:  53
------------------------------------------------------------
for the  21 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  22 th cluster number of 10min intavels with zero pickups:  29
------------------------------------------------------------
for the  23 th cluster number of 10min intavels with zero pickups:  163
------------------------------------------------------------
for the  24 th cluster number of 10min intavels with zero pickups:  35
------------------------------------------------------------
for the  25 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  26 th cluster number of 10min intavels with zero pickups:  31
------------------------------------------------------------
for the  27 th cluster number of 10min intavels with zero pickups:  214
------------------------------------------------------------
for the  28 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  29 th cluster number of 10min intavels with zero pickups:  41
------------------------------------------------------------
for the  30 th cluster number of 10min intavels with zero pickups:  1180
------------------------------------------------------------
for the  31 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  32 th cluster number of 10min intavels with zero pickups:  44
------------------------------------------------------------
for the  33 th cluster number of 10min intavels with zero pickups:  43
------------------------------------------------------------
for the  34 th cluster number of 10min intavels with zero pickups:  39
------------------------------------------------------------
for the  35 th cluster number of 10min intavels with zero pickups:  42
------------------------------------------------------------
for the  36 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  37 th cluster number of 10min intavels with zero pickups:  321
------------------------------------------------------------
for the  38 th cluster number of 10min intavels with zero pickups:  36
------------------------------------------------------------
for the  39 th cluster number of 10min intavels with zero pickups:  43
------------------------------------------------------------
```

Above we counted the number of 0pickups in each cluster 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 [61]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add 0 to the smoothed data
# we finally return smoothed data
def fill_missing(count_values,values):
    smoothed_regions=[]
    ind=0
    for r in range(0,40):
        smoothed_bins=[]
        for i in range(4464):
            if i in values[r]:
                smoothed_bins.append(count_values[ind])
                ind+=1
            else:
                smoothed_bins.append(0)
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [62]:

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

# for every 10min intravel(pickup_bin) we will check it is there in our unique bin,
# if it is there we will add the count_values[index] to smoothed data
# if not we add smoothed data (which is calculated based on the methods that are discussed in the
above markdown cell)
# we finally return smoothed data
def smoothing(count_values,values):
    smoothed_regions=[] # stores list of final smoothed values of each reigion
    ind=0
    repeat=0
    smoothed_value=0
    for r in range(0,40):
        smoothed_bins=[] #stores the final smoothed values
        repeat=0
        for i in range(4464):
            if repeat!=0: # prevents iteration for a value which is already visited/resolved
                repeat-=1
                continue
            if i in values[r]: #checks if the pickup-bin exists
                smoothed_bins.append(count_values[ind]) # appends the value of the pickup bin if it
exists
            else:
                if i!=0:
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]: #searches for the left-limit or the pickup-bin
value which has a pickup value
                            continue
```

```
                        else:
                            right_hand_limit=j
                            break
                    if right_hand_limit==0:
                    #Case 1: When we have the last/last few values are found to be missing,hence we
have no right-limit here
                        smoothed_value=count_values[ind-1]*1.0/((4463-i)+2)*1.0

                        for j in range(i,4464):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(4463-i)
                        ind-=1
                    else:
                    #Case 2: When we have the missing values between two known values
                        smoothed_value=(count_values[ind-1]+count_values[ind])*1.0/((right_hand_lim
t-i)+2)*1.0

                        for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                        smoothed_bins[i-1] = math.ceil(smoothed_value)
                        repeat=(right_hand_limit-i)
                else:
                    #Case 3: When we have the first/first few values are found to be missing,hence
we have no left-limit here
                    right_hand_limit=0
                    for j in range(i,4464):
                        if  j not in values[r]:
                            continue
                        else:
                            right_hand_limit=j
                            break
                    smoothed_value=count_values[ind]*1.0/((right_hand_limit-i)+1)*1.0
                    for j in range(i,right_hand_limit+1):
                            smoothed_bins.append(math.ceil(smoothed_value))
                    repeat=(right_hand_limit-i)
            ind+=1
        smoothed_regions.extend(smoothed_bins)
    return smoothed_regions
```

In [63]:

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

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

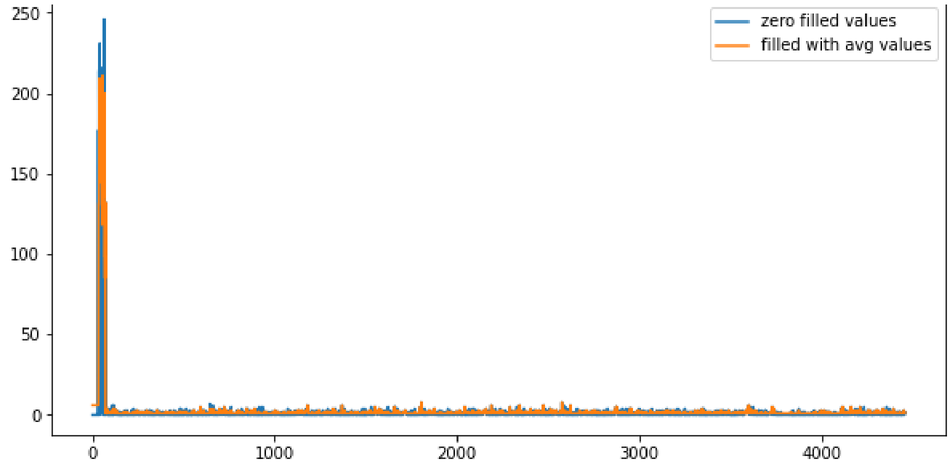
In [64]:

```
# 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 40*4464 = 178560 (length of the
jan_2015_fill)
print("number of 10min intravels among all the clusters ",len(jan_2015_fill))
```

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

In [65]:

```
# 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_fill[4464:8920], label="zero filled values")
plt.plot(jan_2015_smooth[4464:8920], label="filled with avg values")
plt.legend()
plt.show()
```



Smoothing takes care of the presence of 0values at any 10min time bin by distributing the pickups equally

In [66]:

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

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

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

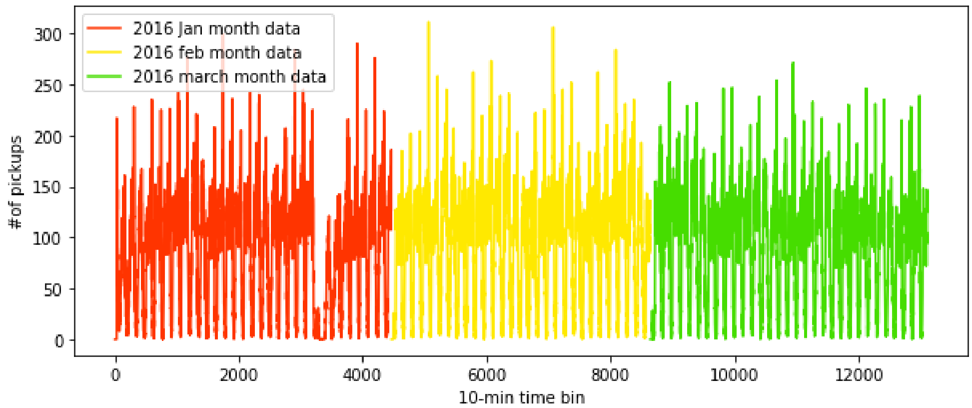
In [67]:

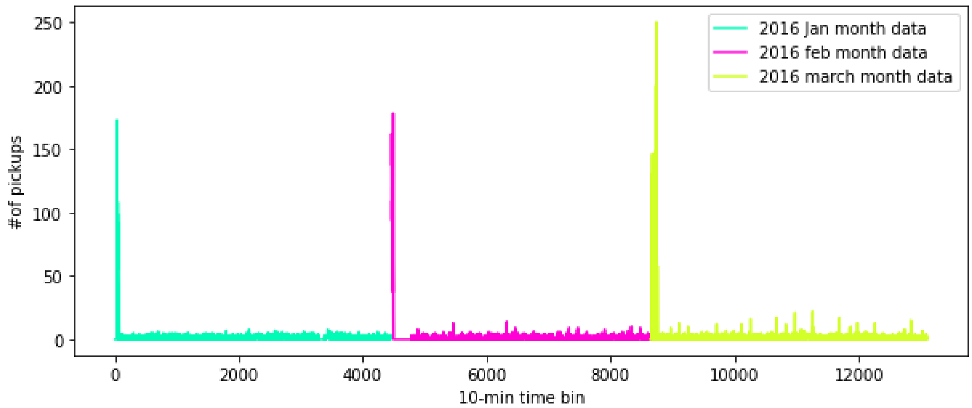
```
# Jan-2015 data is smoothed, Jan,Feb & March 2016 data missing values are filled with zero
jan_2015_smooth = smoothing(jan_2015_groupby['trip_distance'].values,jan_2015_unique)
jan_2016_smooth = fill_missing(jan_2016_groupby['trip_distance'].values,jan_2016_unique)
feb_2016_smooth = fill_missing(feb_2016_groupby['trip_distance'].values,feb_2016_unique)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

# Making list of all the values of pickup data in every bin for a period of 3 months and storing t
hem region-wise
regions_cum = []

# 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 40 lists, each list will contain 4464+4176+4464 values which repres
ents the number of pickups
# that are happened for three months in 2016 data

for i in range(0,40):
    regions_cum.append(jan_2016_smooth[4464*i:4464*(i+1)]+feb_2016_smooth[4176*i:4176*(i+1)]+mar_20
16_smooth[4464*i:4464*(i+1)])

# print(len(regions_cum))
# 40
# print(len(regions_cum[0]))
# 13104
```

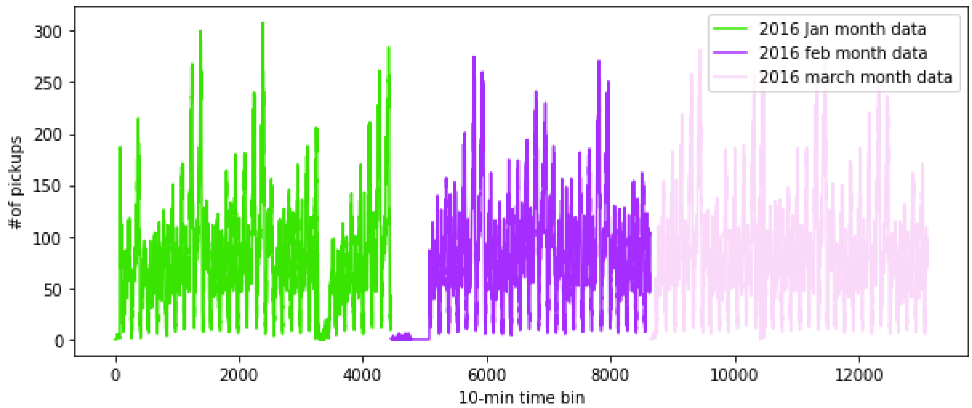
Time series and Fourier Transforms

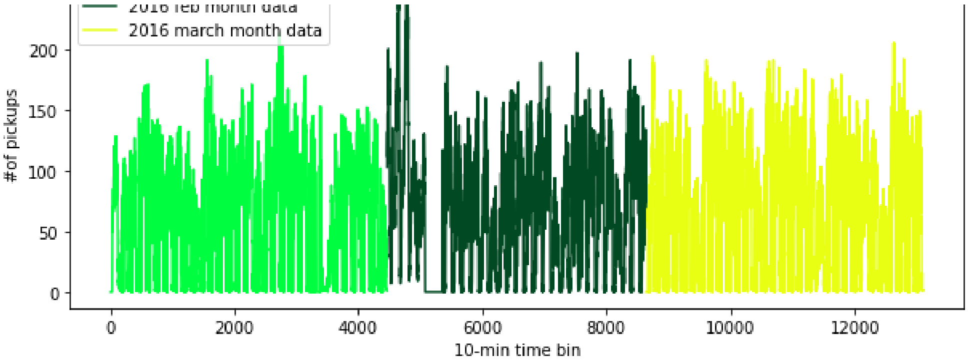
In [68]:

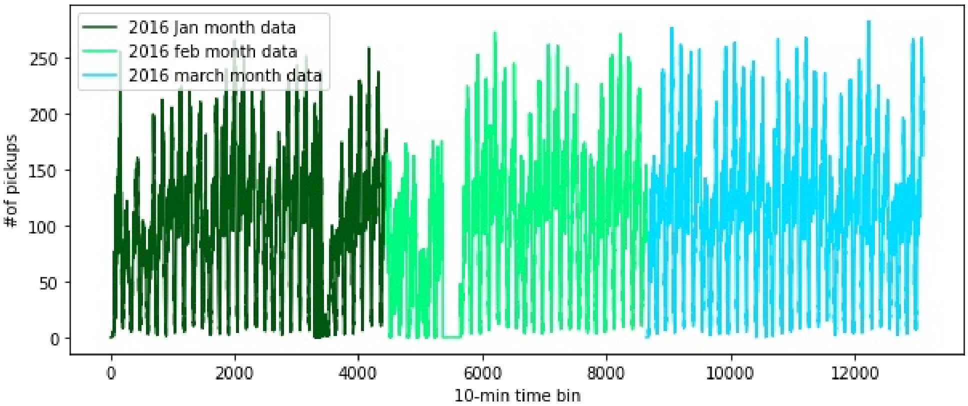
```
def uniqueish_color():
    """There're better ways to generate unique colors, but this isn't awful."""
    return plt.cm.gist_ncar(np.random.random())
first_x = list(range(0,4464))
second_x = list(range(4464,8640))
third_x = list(range(8640,13104))
for i in range(40): #because we have choosen optimal no of cluster as 40 therefore we are
iterating over each clusters
    plt.figure(figsize=(10,4))
    plt.plot(first_x,regions_cum[i][:4464], color=uniqueish_color(), label='2016 Jan month data')
    plt.plot(second_x,regions_cum[i][4464:8640], color=uniqueish_color(), label='2016 feb month dat
a')
    plt.plot(third_x,regions_cum[i][8640:], color=uniqueish_color(), label='2016 march month data')
    plt.xlabel("10-min time bin")
    plt.ylabel("#of pickups")
    plt.legend()
    plt.show()
```

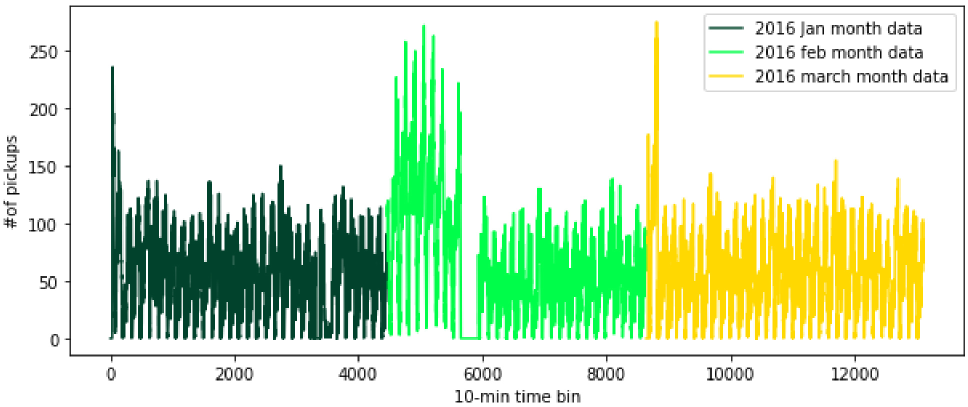


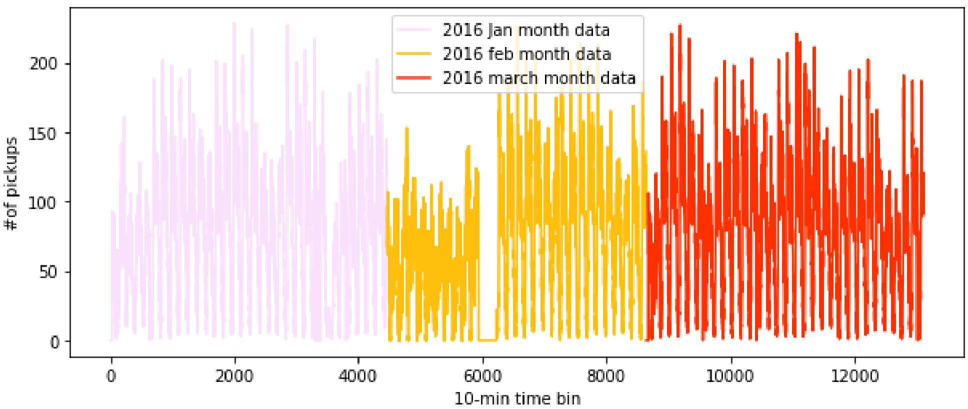


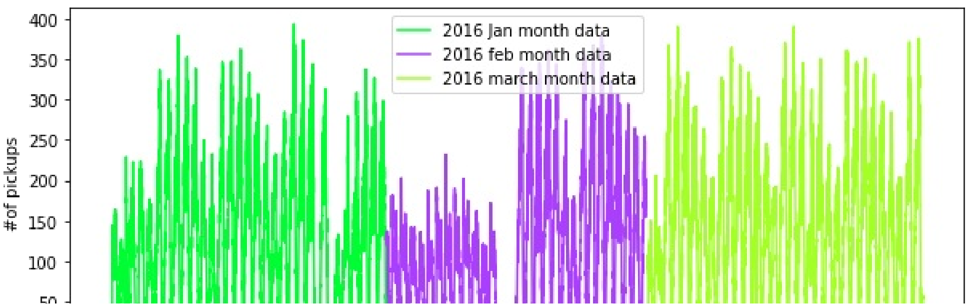


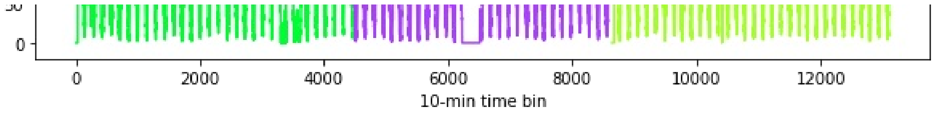

2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

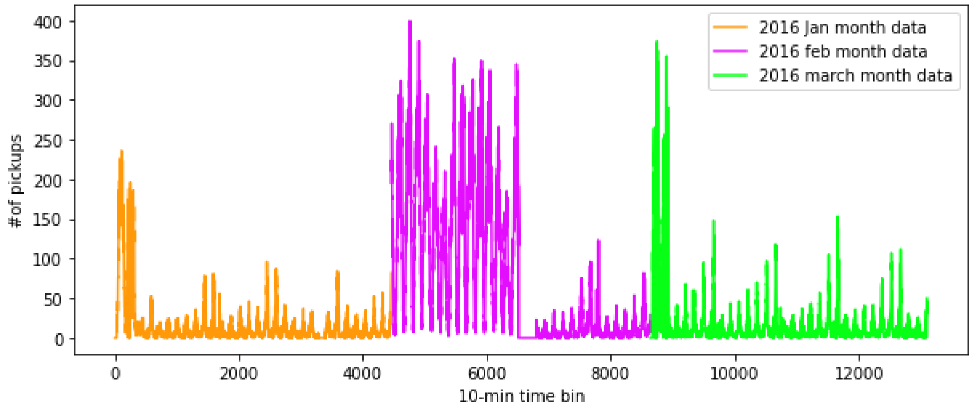

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

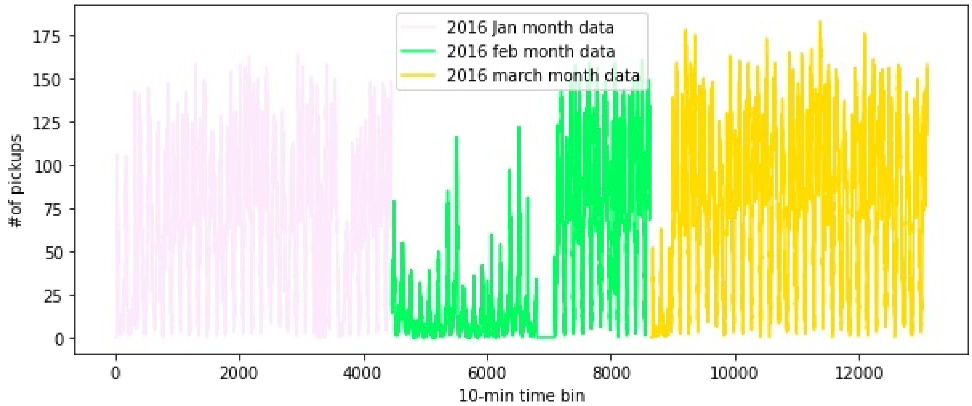

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups



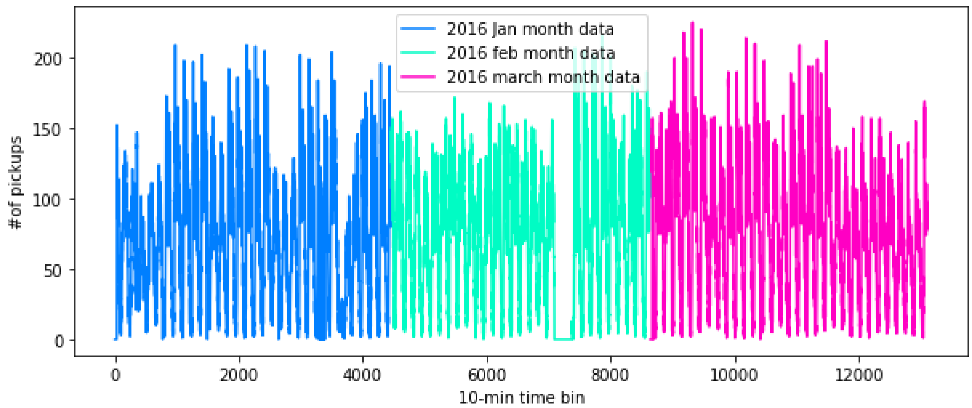
0
2000
4000
6000
8000
10000
12000
10-min time bin



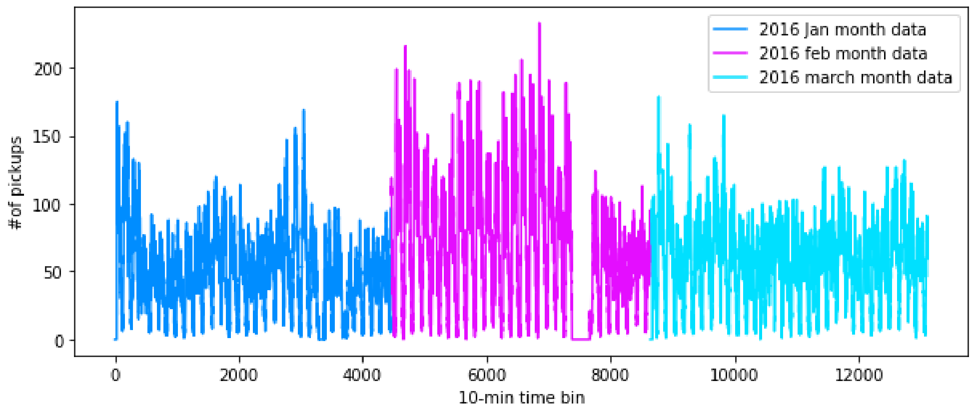
2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin



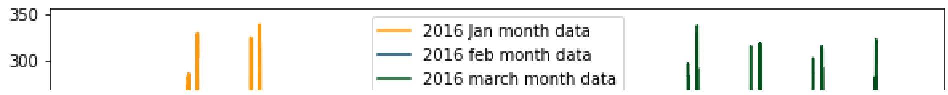
2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin



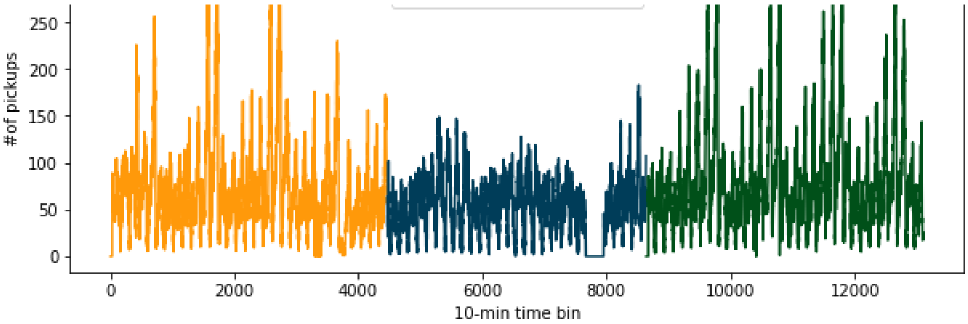
2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

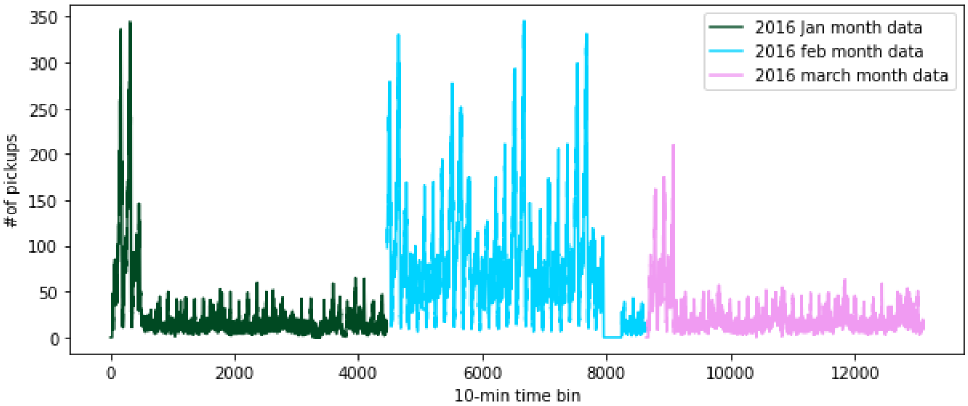


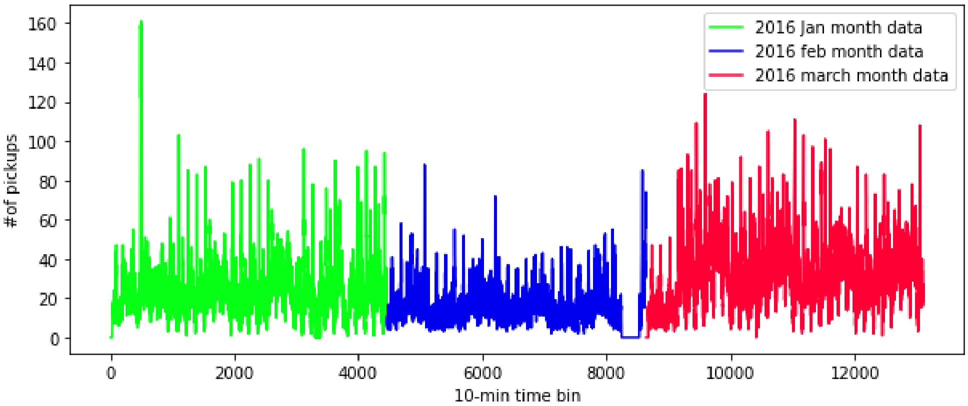
2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

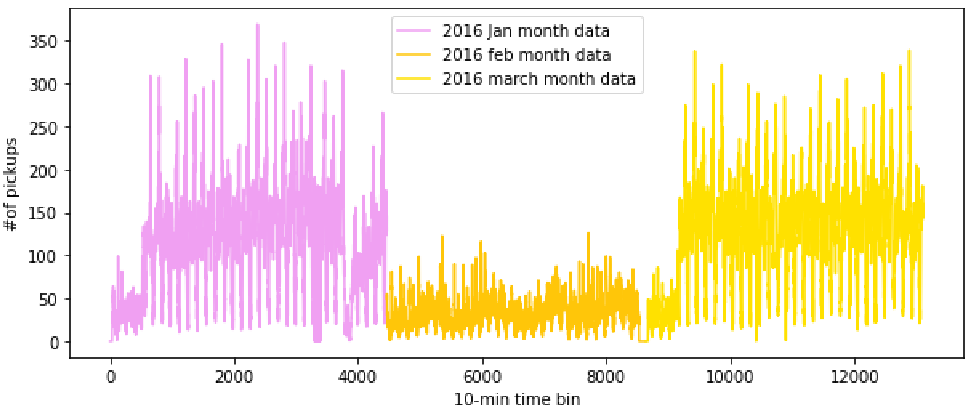


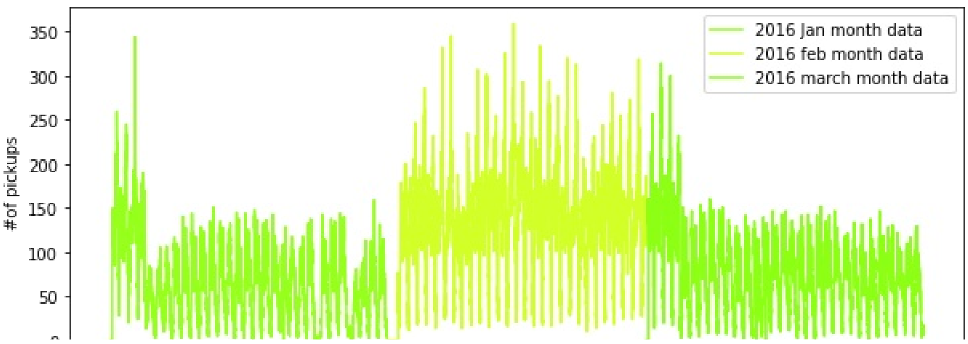
2016 Jan month data
2016 feb month data
2016 march month data

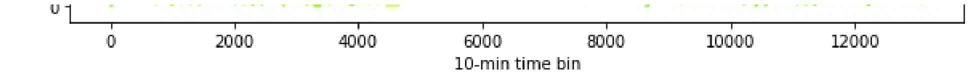

#of pickups
10-min time bin

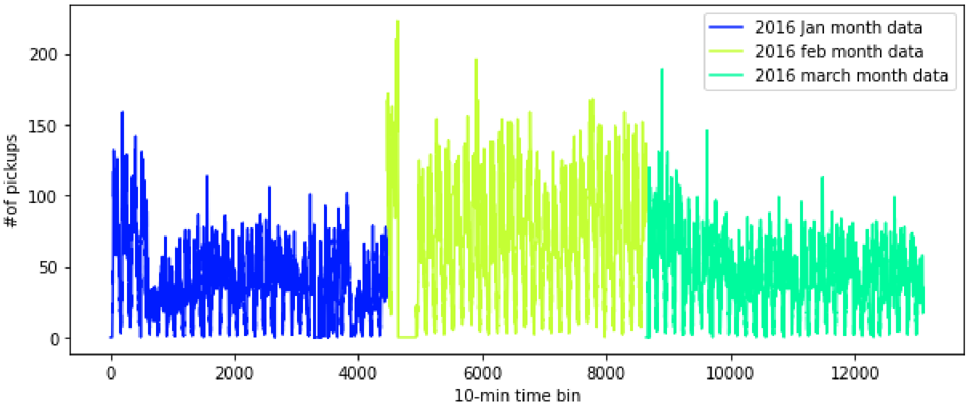

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

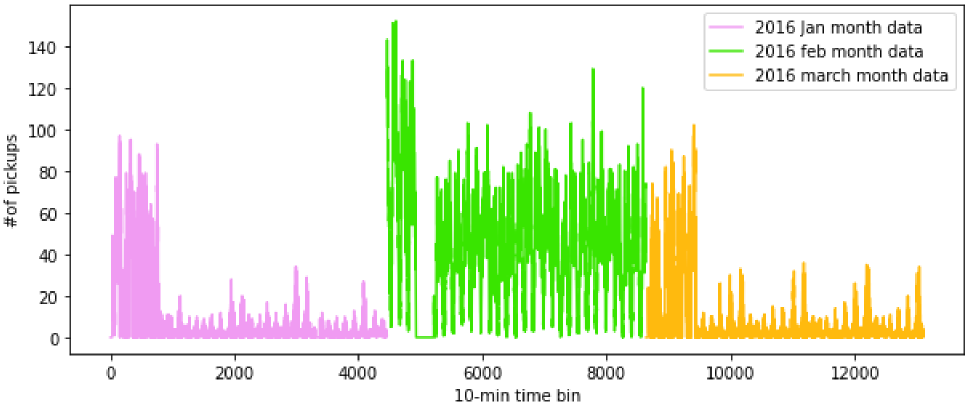

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

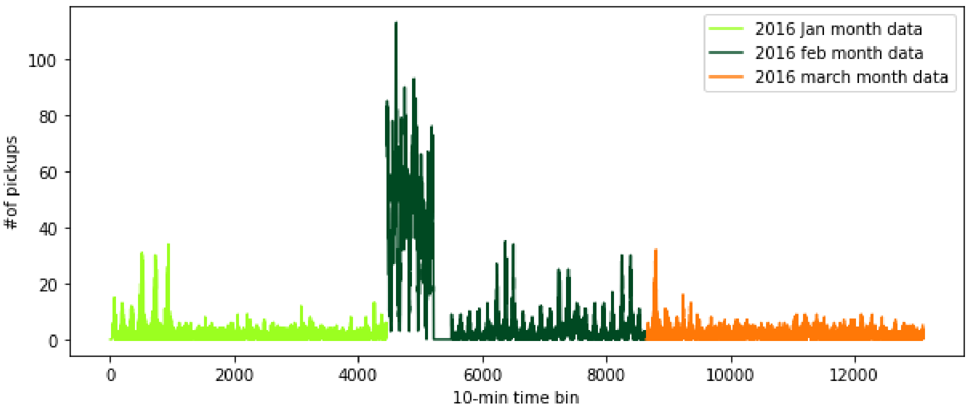

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

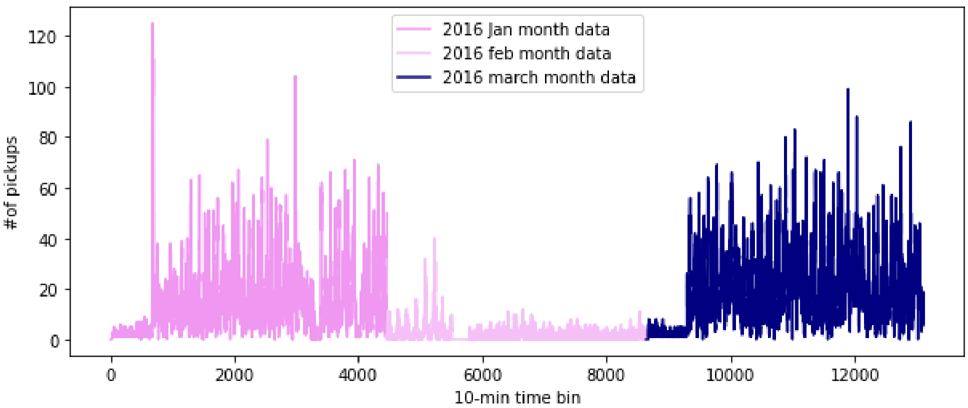

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups


10-min time bin

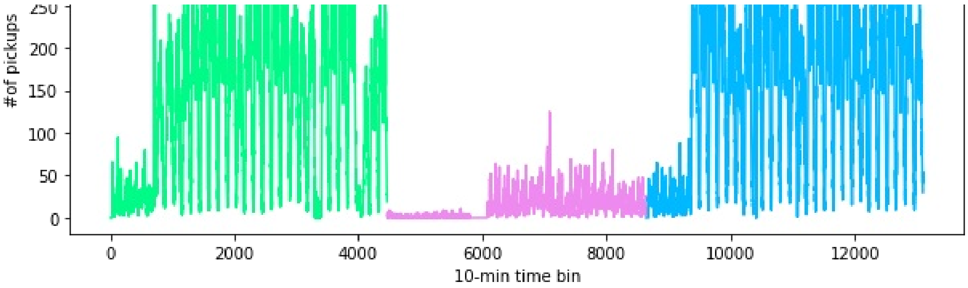

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

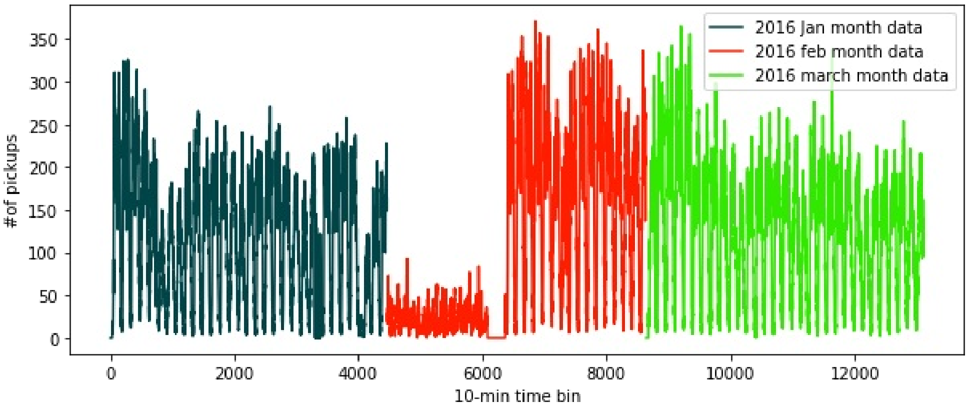

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

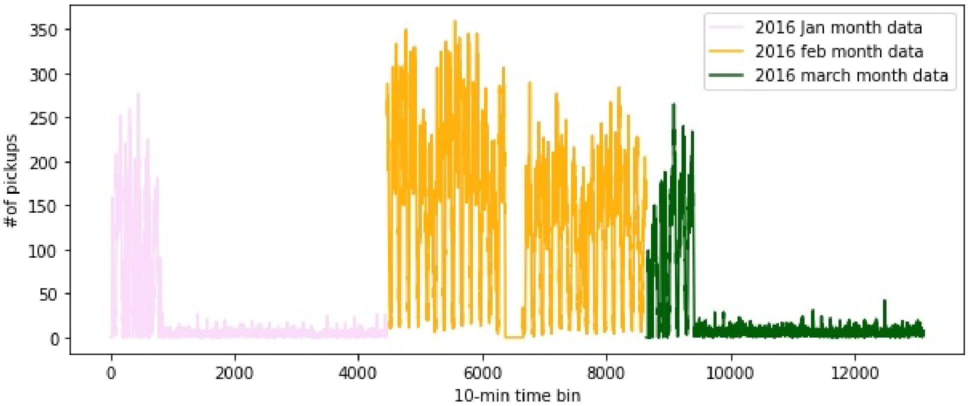

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

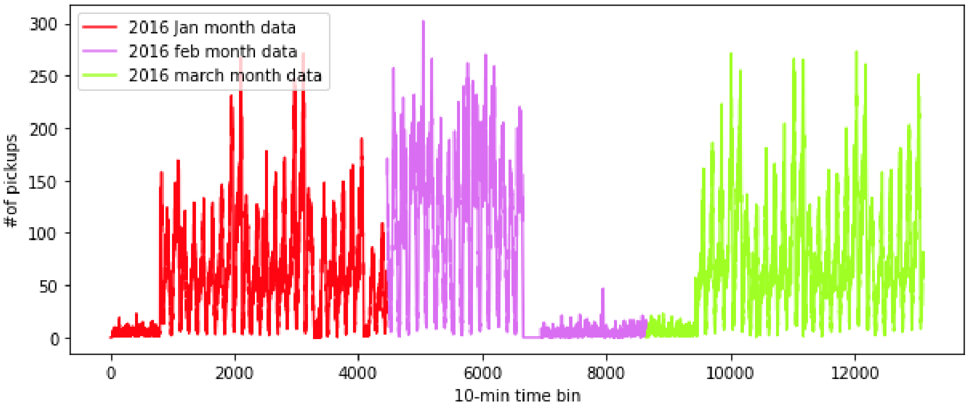

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

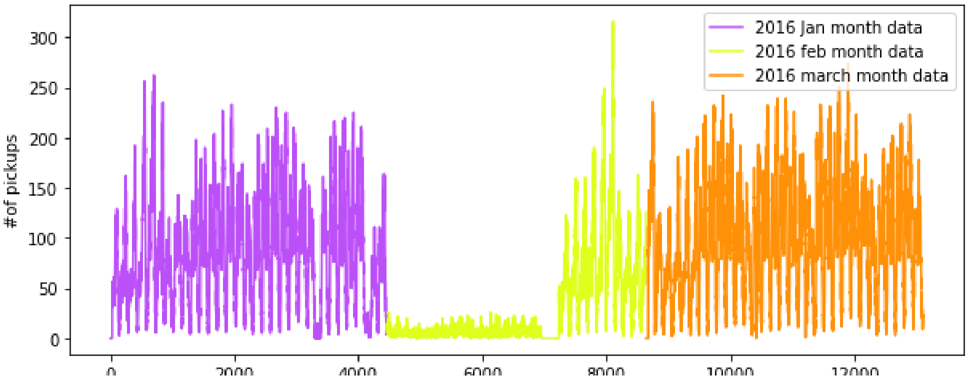

2016 Jan month data
2016 feb month data
2016 march month data


#of pickups
0 50 100 150 200 250
0 2000 4000 6000 8000 10000 12000
10-min time bin

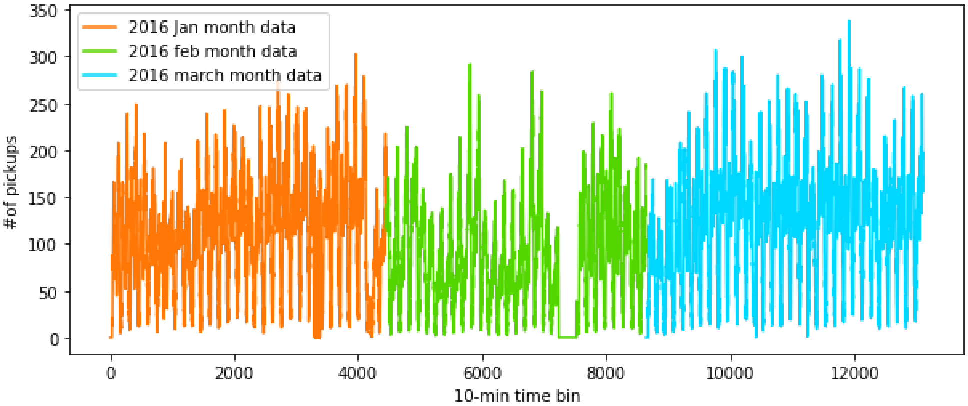

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
0 50 100 150 200 250 300 350
0 2000 4000 6000 8000 10000 12000
10-min time bin

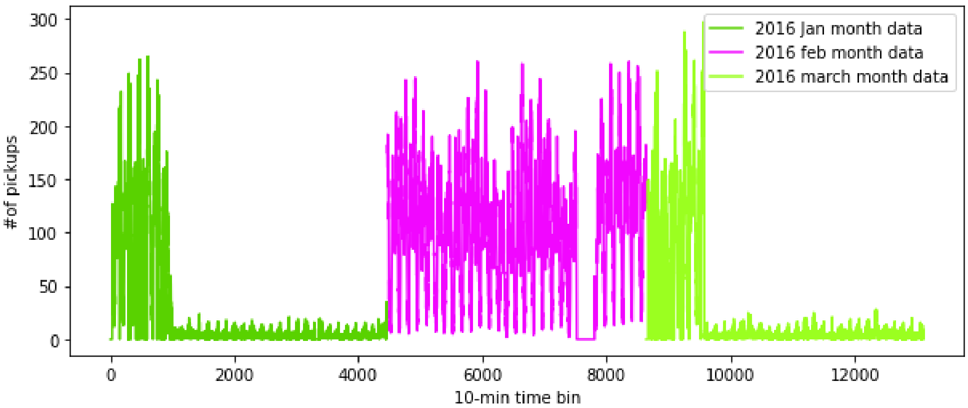

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
0 50 100 150 200 250 300 350
0 2000 4000 6000 8000 10000 12000
10-min time bin

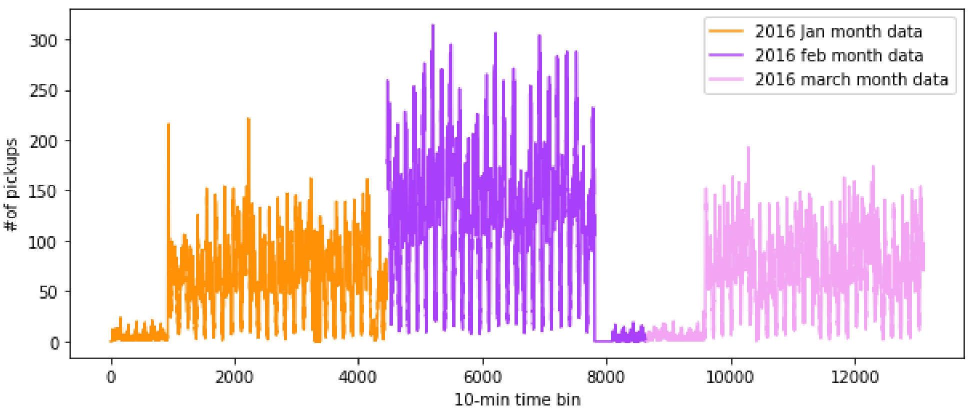

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
0 50 100 150 200 250 300
0 2000 4000 6000 8000 10000 12000
10-min time bin

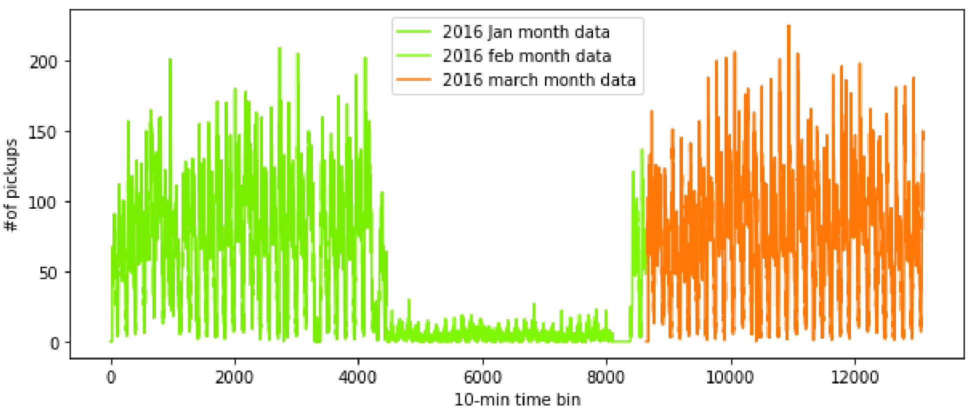

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
0 50 100 150 200 250 300

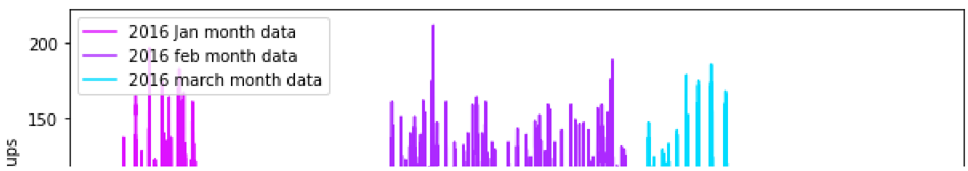

10-min time bin

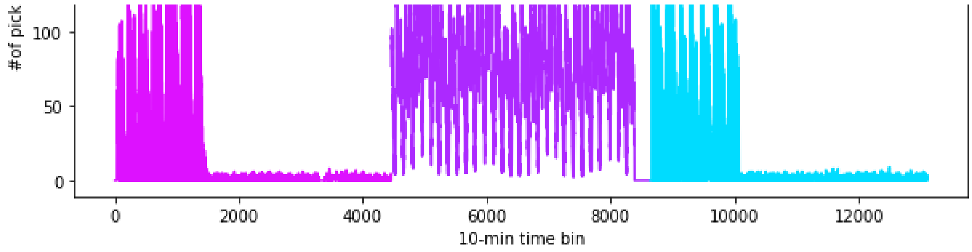

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

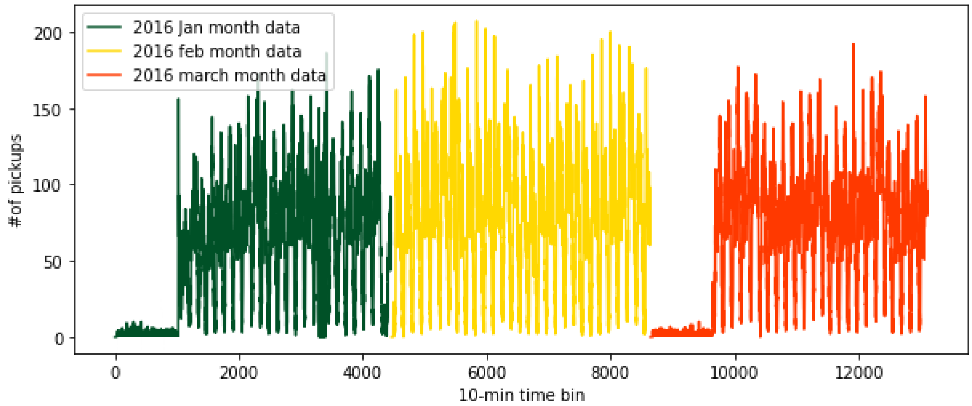

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

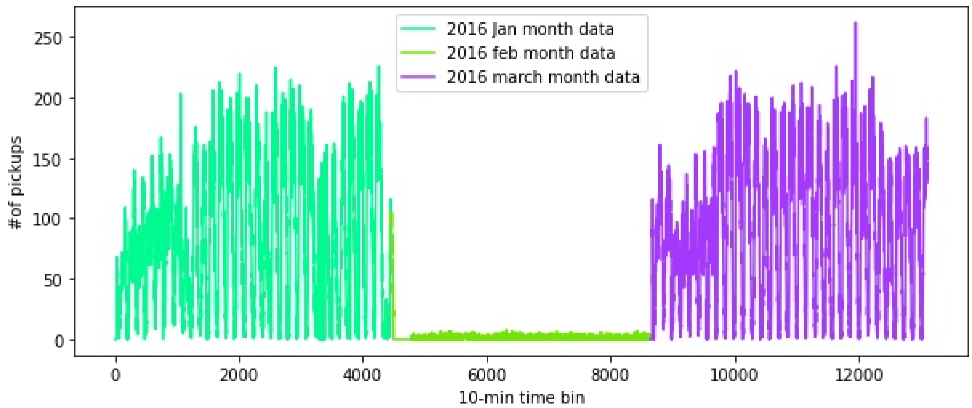

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

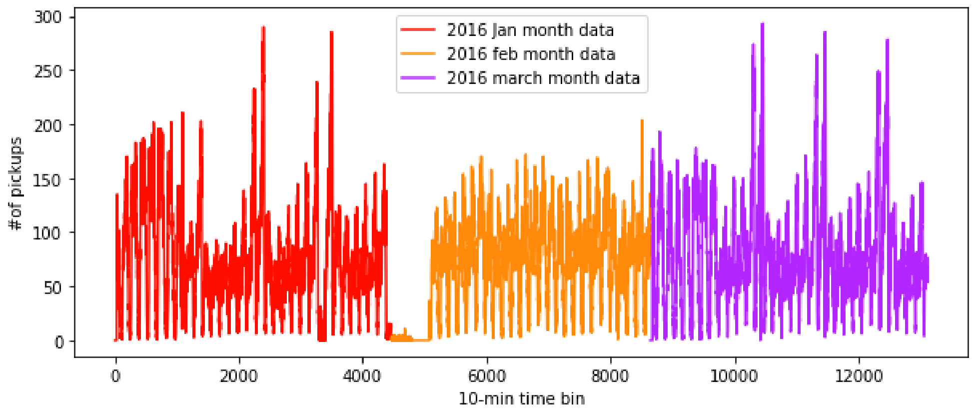

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

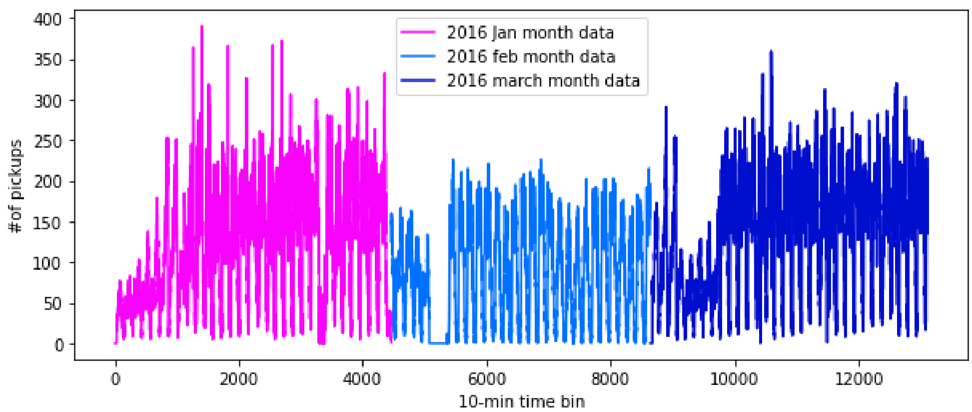

2016 Jan month data
2016 feb month data
2016 march month data

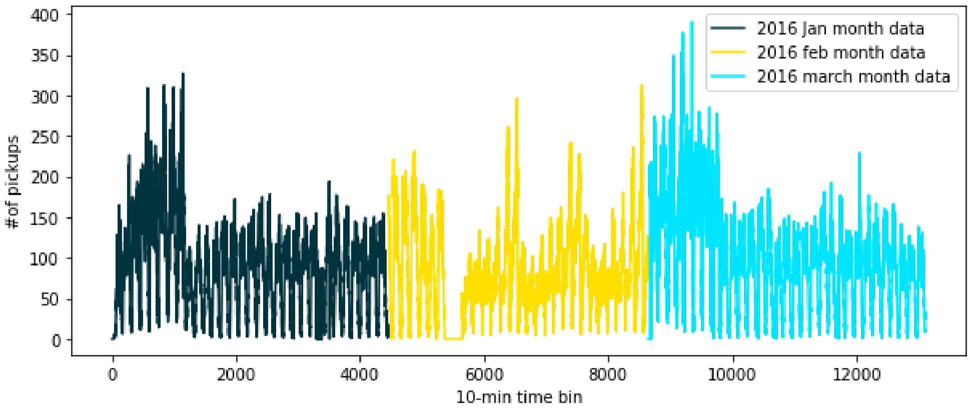

#of pick
10-min time bin

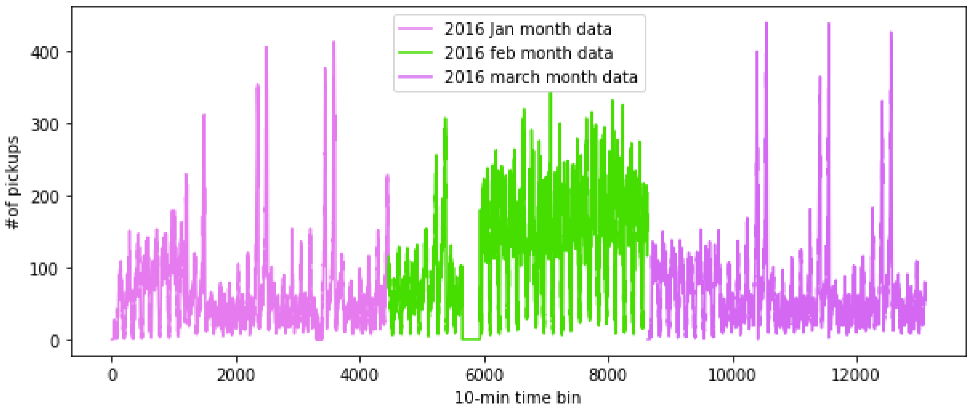

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

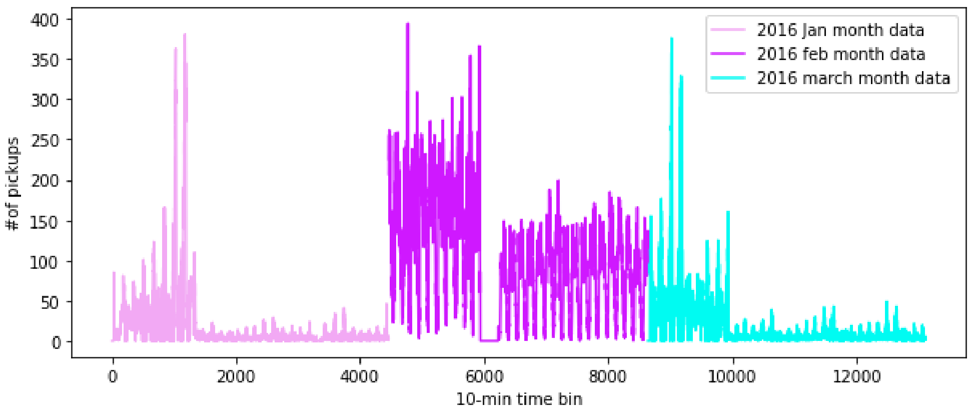

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

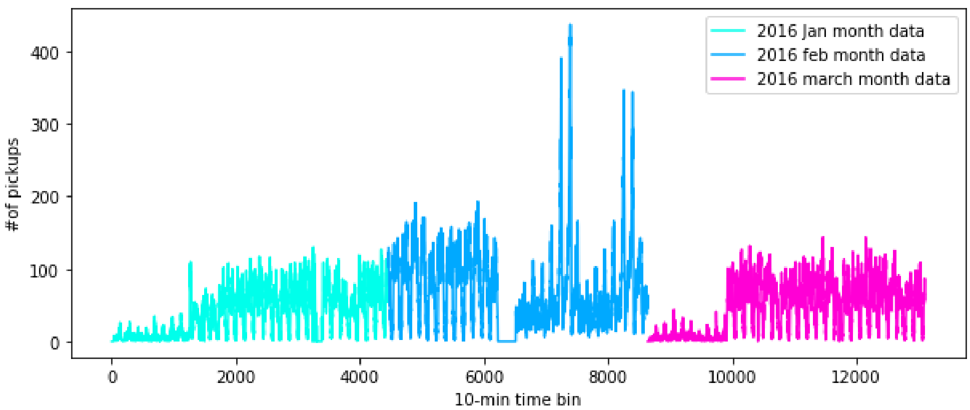

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

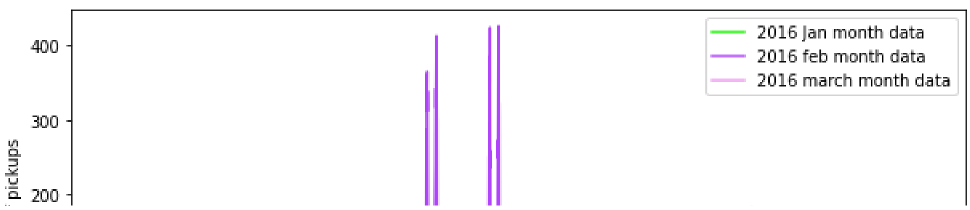

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin

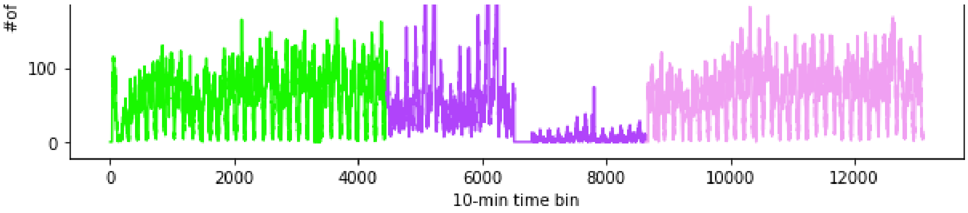

2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data
#of pickups
10-min time bin


2016 Jan month data
2016 feb month data
2016 march month data



on analysing the graph we observe that the graph is repeating itself after 144units/1440 min interval and therefore we can apply fourier transform where frequency = 1/T

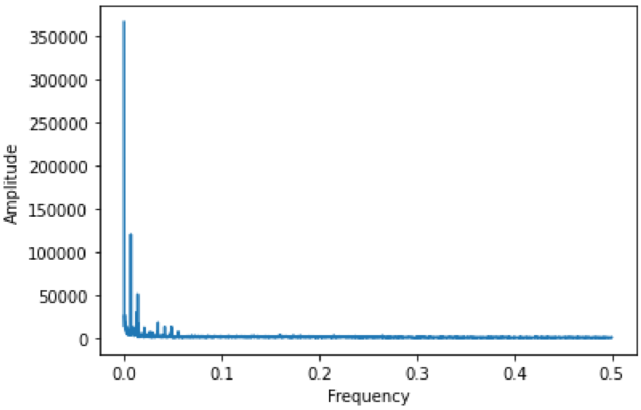
T= 144

freq = 1/144

Therefore plotting the fft of the above graph we get

In [69]:

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



In [70]:

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

Modelling: Baseline Models

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

1. Using Ratios of the 2016 data to the 2015 data i.e $\begin{align} R_{t} = P^{2016}_{t} / P^{2015}_{t} \end{align}$ where **p^2016 is the no of pickups in 2016 and p^2015 is the no of pickups in 2015**

underline assumption is the if we want to predict the value of 2016 at any point it will be dependent on the previous year value at the same time X the ratio between the two

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 - $\begin{align}R_{t} = (R_{t-1} + R_{t-2} + R_{t-3} .... R_{t-n})/n \end{align}$

In [71]:

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


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

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

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

In [72]:

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

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

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

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 - $\begin{align}R_{t} = (N*R_{t-1} + (N-1)*R_{t-2} + (N-2)*R_{t-3} .... 1*R_{t-n})/(N*(N+1)/2) \end{align}$

In [73]:

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

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

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

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

In [74]:

```
def WA_P_Predictions(ratios,month):
    predicted_value=(ratios['Prediction'].values)[0]
    error=[]
    predicted_values=[]
    window_size=2
    for i in range(0,4464*40):
```

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

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

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

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

Exponential Weighted Moving Averages

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

In exponential moving averages we use a single hyperparameter alpha $\begin{align}(\alpha)\end{align}$ 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 $\begin{align}\alpha=0.9\end{align}$ then the number of days on which the value of the current iteration is based is~$\begin{align}1/(1-\alpha)=10\end{align}$ i.e. we consider values 10 days prior before we predict the value for the current iteration. Also the weights are assigned using $\begin{align}2/(N+1)=0.18\end{align}$,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.

$\begin{align}R^{'}_{t} = \alpha*R_{t-1} + (1-\alpha)*R^{'}_{t-1} \end{align}$

In [75]:

```
(ratios_jan["Ratios"].values)[0]
```

Out[75]:

```
0.0
```

In [76]:

```
def EA_R1_Predictions(ratios,month):
    predicted_ratio=(ratios['Ratios'].values)[0]
    alpha=0.6
    error=[]
    predicted_values=[]
    predicted_ratio_values=[]
    for i in range(0,4464*40):
        if i%4464==0:
            predicted_ratio_values.append(0)
            predicted_values.append(0)
            error.append(0)
            continue
        predicted_ratio_values.append(predicted_ratio)
```

```
        predicted_values.append(int(((ratios['Given'].values)[i])*predicted_ratio))
        error.append(abs((math.pow(int(((ratios['Given'].values)[i])*predicted_ratio)-(ratios['Pred
iction'].values)[i],1))))
        predicted_ratio = (alpha*predicted_ratio) + (1-alpha)*((ratios['Ratios'].values)[i])

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

$\begin{align}P^{'}_{t} = \alpha*P_{t-1} + (1-\alpha)*P^{'}_{t-1} \end{align}$

In [77]:

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

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

In [78]:

```
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 [79]:

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

```
print ("-------------------------------------------------------------------------------------------------")
print ("Exponential Moving Averages (Ratios) -           MAPE: ",mean_err[4],"      MSE: ",median_err[4])
print ("Exponential Moving Averages (2016 Values) -      MAPE: ",mean_err[5],"      MSE: ",median_err[5])
```

```
Error Metric Matrix (Forecasting Methods) - MAPE & MSE
--------------------------------------------------------------------------------------------------------
Moving Averages (Ratios) -                          MAPE:  0.22785156353133512       MSE:  1196.953853046595
Moving Averages (2016 Values) -                     MAPE:  0.15583458712025738        MSE:  254.6309363799283
--------------------------------------------------------------------------------------------------------
Weighted Moving Averages (Ratios) -                 MAPE:  0.22706529144871415       MSE:  1053.083529345878
Weighted Moving Averages (2016 Values) -            MAPE:  0.1479482182992932       MSE:  224.81054547491038
--------------------------------------------------------------------------------------------------------
Exponential Moving Averages (Ratios) -           MAPE:  0.2275474636148534       MSE:  1019.3071012544802
Exponential Moving Averages (2016 Values) -      MAPE:  0.1475381297798153       MSE:  222.35159610215055
```

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

Regression Models

Train-Test Split

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

In [80]:

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

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

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

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


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

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

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

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


tsne_feature = [0]*number_of_time_stamps
for i in range(0,40):
    tsne_lat.append([kmeans.cluster_centers_[i][0]]*13099)
    tsne_lon.append([kmeans.cluster_centers_[i][1]]*13099)
    # jan 1st 2016 is thursday, so we start our day from 4: "(int(k/144))%7+4"
    # our prediction start from 5th 10min intravel since we need to have number of pickups that ar
e happened in last 5 pickup bins
    tsne_weekday.append([int(((int(k/144))%7+4)%7) for k in range(5,4464+4176+4464)])
    # regions_cum is a list of lists [[x1,x2,x3..x13104], [x1,x2,x3..x13104], [x1,x2,x3..x13104],
[x1,x2,x3..x13104], [x1,x2,x3..x13104], .. 40 lsits]
    tsne_feature = np.vstack((tsne_feature, [regions_cum[i][r:r+number_of_time_stamps] for r in ran
ge(0,len(regions_cum[i])-number_of_time_stamps)]))
    output.append(regions_cum[i][5:])
tsne_feature = tsne_feature[1:]
```

In [81]:

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

```
13099
40
```

In [82]:

```
len(tsne_lat[0])*len(tsne_lat) == tsne_feature.shape[0] == len(tsne_weekday)*len(tsne_weekday[0]) =
= 40*13099 == len(output)*len(output[0])
```

Out[82]:

```
True
```

In [83]:

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

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

# from the baseline models we said the exponential weighted moving avarage gives us the best error
# we will try to add the same exponential weighted moving avarage at t as a feature to our data
# exponential weighted moving avarage => p'(t) = alpha*p'(t-1) + (1-alpha)*P(t-1)
alpha=0.3
```

```
alpha=0.5

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

# it is similar like tsne_lat
# it is list of lists
# predict_list is a list of lists [[x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5,x6,x7..x13104], [x5
,x6,x7..x13104], [x5,x6,x7..x13104], .. 40 lsits]
predict_list = []
tsne_flat_exp_avg = []
for r in range(0,40):
    for i in range(0,13104):
        if i==0:
            predicted_value= regions_cum[r][0]
            predicted_values.append(0)
            continue
        predicted_values.append(predicted_value)
        predicted_value =int((alpha*predicted_value) + (1-alpha)*(regions_cum[r][i]))
    predict_list.append(predicted_values[5:])
    predicted_values=[]
```

In [ ]:

```
# Fourier Transformed Features
```

In [84]:

```
amplitude = [] #for storing amplitudes
frequency = [] #for storing frequencies
for i in range(40): #iterating over each clusters
    amp  = np.abs(np.fft.fft(regions_cum[i][0:13104]))#amplitude calculation
    freq = np.abs(np.fft.fftfreq(13104, 1)) #frequencies calculation
    amp_indices = np.argsort(-amp)[1:]     #sorting amplitude
    amp_values = []
    freq_values = []
    for j in range(0, 9, 2):   #taking top 5 amplitudes and frequencies
        amp_values.append(amp[amp_indices[j]])
        freq_values.append(freq[amp_indices[j]])
    for k in range(13104):    #those top 5 frequencies and amplitudes are same for all the points
in one cluster
        amplitude.append(amp_values)
        frequency.append(freq_values)
```

Holt-Winters Forecasting

Double Exponential Smoothing

https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-ii/

In [85]:

```
#intializing the trend
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen
```

In [86]:

```
def double_exponential_smoothing(series, slen, alpha, beta, n_preds):
    result = []
    #seasonals = initial_seasonal_components(series, slen)
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
```

```
            result.append(series[0])
            continue
        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append(smooth + m*trend)# + seasonals[i%slen])
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val) + (1-alpha)*(smooth+trend)
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            #seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            result.append(smooth+trend)
    return result
```

In [87]:

```
alpha = 0.2
beta = 0.15
season_len = 24 #becuase we see the same trend each day and each day has 24hrs

predict_values_1 =[]
predict_list_1 = []
tsne_flat_exp_avg = []
for r in range(0,40):
    predict_values_1 = double_exponential_smoothing(regions_cum[r][0:13104], season_len, alpha, bet
a, 0)
    predict_list_1.append(predict_values_1[5:])
```

In [88]:

```
print(len(predict_list_1[0]))
print(len(predict_list_1))
```

```
13099
40
```

Triple Exponential Smoothing

https://grisha.org/blog/2016/02/17/triple-exponential-smoothing-forecasting-part-iii/

In [89]:

```
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen

def initial_seasonal_components(series, slen):
    seasonals = {}
    season_averages = []
    n_seasons = int(len(series)/slen)
    # compute season averages
    for j in range(n_seasons):
        season_averages.append(sum(series[slen*j:slen*j+slen])/float(slen))
    # compute initial values
    for i in range(slen):
        sum_of_vals_over_avg = 0.0
        for j in range(n_seasons):
            sum_of_vals_over_avg += series[slen*j+i]-season_averages[j]
        seasonals[i] = sum_of_vals_over_avg/n_seasons
    return seasonals


def triple_exponential_smoothing(series, slen, alpha, beta, gamma, n_preds):
    result = []
    seasonals = initial_seasonal_components(series, slen)
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
            result.append(series[0])
```

```
            continue
        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append((smooth + m*trend) + seasonals[i%slen])
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-alpha)*(smooth+trend)
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            result.append(smooth+trend+seasonals[i%slen])
    return result



alpha = 0.2
beta = 0.15
gamma = 0.2
season_len = 24

predict_values_2 =[]
predict_list_2 = []
tsne_flat_exp_avg_2 = []
for r in range(0,40):
    predict_values_2 = triple_exponential_smoothing(regions_cum[r][0:13104], season_len, alpha, bet
a, gamma, 0)
    predict_list_2.append(predict_values_2[5:])
```

In [90]:

```
print(len(predict_list_2))
print(len(predict_list_2[0]))
```

```
40
13099
```

Splitting

In [91]:

```
# train, test split : 70% 30% split
# Before we start predictions using the tree based regression models we take 3 months of 2016 pick
up data
# and split it such that for every region we have 70% data in train and 30% in test,
# ordered date-wise for every region
print("size of train data :", int(13099*0.7))
print("size of test data :", int(13099*0.3))
```

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

In [92]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
#finding frequencies for train and test
train_freq =  [frequency[i*13099:(13099*i+9169)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_freq = [frequency[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

In [93]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
#finding amplitude for train and test
train_amp =  [amplitude[i*13099:(13099*i+9169)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_amp = [amplitude[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

In [94]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
```

```
train_features =  [tsne_feature[i*13099:(13099*i+9169)] for i in range(0,40)]
# temp = [0]*(12955 - 9068)
test_features = [tsne_feature[(13099*(i))+9169:13099*(i+1)] for i in range(0,40)]
```

In [95]:

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

```
Number of data clusters 40 Number of data points in trian data 9169 Each data point contains 5 fea
tures
Number of data clusters 40 Number of data points in test data 3930 Each data point contains 5 feat
ures
```

In [96]:

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

```
Number of data clusters 40 Number of data points in trian data 9169 Each data point contains 5 fre
quency
Number of data clusters 40 Number of data points in test data 3930 Each data point contains 5 freq
uency
```

In [97]:

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

```
Number of data clusters 40 Number of data points in trian data 9169 Each data point contains 5 amp
litude
Number of data clusters 40 Number of data points in test data 3930 Each data point contains 5 ampl
itude
```

In [98]:

```
print(f"{len(tsne_lat)}\n{len(tsne_lon)}\n{len(tsne_weekday)}\n{len(output)}\n{len(predict_list)}\
n{len(predict_list_1)}\n{len(predict_list_2)}\n")
```

```
40
40
40
40
40
40
40
```

In [99]:

```
# extracting first 9169 timestamp values i.e 70% of 13099 (total timestamps) for our training data
tsne_train_flat_lat = [i[:9169] for i in tsne_lat]
tsne_train_flat_lon = [i[:9169] for i in tsne_lon]
tsne_train_flat_weekday = [i[:9169] for i in tsne_weekday]
tsne_train_flat_output = [i[:9169] for i in output]
tsne_train_flat_exp_avg = [i[:9169] for i in predict_list]
tsne_train_flat_double_avg = [i[:9169] for i in predict_list_1]
tsne_train_flat_triple_avg = [i[:9169] for i in predict_list_2]
```

In [100]:

```
len(tsne_train_flat_lat)
```

Out[100]:

```
40
```

In [101]:

```
len(tsne_train_flat_double_avg)
```

Out[101]:

```
40
```

In [102]:

```
# extracting the rest of the timestamp values i.e 30% of 12956 (total timestamps) for our test dat
a
tsne_test_flat_lat = [i[9169:] for i in tsne_lat]
tsne_test_flat_lon = [i[9169:] for i in tsne_lon]
tsne_test_flat_weekday = [i[9169:] for i in tsne_weekday]
tsne_test_flat_output = [i[9169:] for i in output]
tsne_test_flat_exp_avg = [i[9169:] for i in predict_list]
tsne_test_flat_double_avg = [i[9169:] for i in predict_list_1]
tsne_test_flat_triple_avg = [i[9169:] for i in predict_list_2]
```

In [103]:

```
len(tsne_test_flat_lat)
```

Out[103]:

```
40
```

In [104]:

```
len(tsne_test_flat_triple_avg)
```

Out[104]:

```
40
```

In [105]:

```
len(tsne_test_flat_double_avg)
```

Out[105]:

```
40
```

In [107]:

```
# the above contains values in the form of list of lists (i.e. list of values of each region), her
e we make all of them in one list
train_new_features = []
for i in range(0,40):
    train_new_features.extend(train_features[i])
test_new_features = []
for i in range(0,40):
    test_new_features.extend(test_features[i])
```

In [108]:

```
train_frequency = []
test_frequency = []
train_amplitude = []
test_amplitude = []
for i in range(0,40):
    train_frequency.extend(train_freq[i])
    test_frequency.extend(test_freq[i])#we are using extend instead of append becuase train_freq,
```

```
test_freq itself is a list
    train_amplitude.extend(train_amp[i])
    test_amplitude.extend(test_amp[i])
```

In [109]:

```
#horizontal stacking new features + amplitude and frequency
train_brand_new_features=np.hstack((train_new_features,train_frequency,train_amplitude))
test_brand_new_features=np.hstack((test_new_features,test_frequency,test_amplitude))
```

In [110]:

```
# 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,[])
tsne_train_double=sum(tsne_train_flat_double_avg,[])
tsne_train_triple=sum(tsne_train_flat_triple_avg,[])
```

In [111]:

```
len(tsne_train_triple)
```

Out[111]:

```
366760
```

In [112]:

```
# 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,[])
tsne_test_double=sum(tsne_test_flat_double_avg,[])
tsne_test_triple=sum(tsne_test_flat_triple_avg,[])
```

In [113]:

```
len(tsne_test_triple)
```

Out[113]:

```
157200
```

In [114]:

```
# Preparing the data frame for our train data
#creating new dataframe from our features and data where coloumns name will be equal to the name w
e pass in the coloumn list
columns = ['ft5','ft4','ft3','ft2','ft1','freq1','freq2','freq3','freq4','freq5','Amp1','Amp2','Amp
3','Amp4','Amp5']
train_df = pd.DataFrame(data=train_brand_new_features, columns=columns)
train_df['lat'] = tsne_train_lat
train_df['lon'] = tsne_train_lon
train_df['weekday'] = tsne_train_weekday
train_df['exp_avg'] = tsne_train_exp_avg
train_df['exp_double_avg'] = tsne_train_double
train_df['exp_triple_avg'] = tsne_train_triple
```

```
print(train_df.shape)
print(train_df.head())
```

```
(366760, 21)
   ft5  ft4  ft3  ft2  ...  weekday  exp_avg  exp_double_avg  exp_triple_avg
0  0.0  0.0  0.0  0.0  ...        4        0       11.369697        8.245475
1  0.0  0.0  0.0  0.0  ...        4        0       11.299744        7.045487
2  0.0  0.0  0.0  0.0  ...        4        0       10.904790        5.408308
3  0.0  0.0  0.0  0.0  ...        4        0       10.261683        2.349447
4  0.0  0.0  0.0  0.0  ...        4        0        9.439347        3.437864

[5 rows x 21 columns]
```

In [115]:

```
# Preparing the data frame for our test data
columns = ['ft5','ft4','ft3','ft2','ft1','freq1','freq2','freq3','freq4','freq5','Amp1','Amp2','Amp
3','Amp4','Amp5']
test_df = pd.DataFrame(data=test_brand_new_features, columns=columns)
test_df['lat'] = tsne_test_lat
test_df['lon'] = tsne_test_lon
test_df['weekday'] = tsne_test_weekday
test_df['exp_avg'] = tsne_test_exp_avg
test_df['exp_double_avg'] = tsne_test_double
test_df['exp_triple_avg'] = tsne_test_triple
print(test_df.shape)
```

```
(157200, 21)
```

In [116]:

```
test_df.head()
```

Out[116]:

	ft5	ft4	ft3	ft2	ft1	freq1	freq2	freq3	freq4	freq5	Amp1	Amp2	Amp3	
0	143.0	145.0	119.0	113.0	124.0	0.006944	0.013889	0.012897	0.034722	0.007937	364029.703039	181600.695635	83398.440676	67
1	145.0	119.0	113.0	124.0	121.0	0.006944	0.013889	0.012897	0.034722	0.007937	364029.703039	181600.695635	83398.440676	67
2	119.0	113.0	124.0	121.0	131.0	0.006944	0.013889	0.012897	0.034722	0.007937	364029.703039	181600.695635	83398.440676	67
3	113.0	124.0	121.0	131.0	110.0	0.006944	0.013889	0.012897	0.034722	0.007937	364029.703039	181600.695635	83398.440676	67
4	124.0	121.0	131.0	110.0	116.0	0.006944	0.013889	0.012897	0.034722	0.007937	364029.703039	181600.695635	83398.440676	67

Using Linear Regression

Hyperparameter Tuning using GridSearch

https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/

In [117]:

```
# find more about LinearRegression function here http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# -------------------------
# default paramters
# sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1
)

# some of methods of LinearRegression()
# fit(X, y[, sample_weight]) Fit linear model.
# get_params([deep]) Get parameters for this estimator.
```

```
# predict(X) Predict using the linear model
# score(X, y[, sample_weight]) Returns the coefficient of determination R^2 of the prediction.
# set_params(**params) Set the parameters of this estimator.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1-2-copy-8/
# -----------------------

from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV

sgd = SGDRegressor(loss = "squared_loss", penalty = "l2")

param_grid={
    "alpha" : [10**-4, 10**-2, 10**0, 10**2, 10**3, 10**4]
}

best = GridSearchCV(sgd, param_grid, scoring = "neg_mean_absolute_error", cv = 3)
best.fit(train_df, tsne_train_output)
```

Out[117]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=SGDRegressor(alpha=0.0001, average=False,
                                    early_stopping=False, epsilon=0.1,
                                    eta0=0.01, fit_intercept=True,
                                    l1_ratio=0.15, learning_rate='invscaling',
                                    loss='squared_loss', max_iter=1000,
                                    n_iter_no_change=5, penalty='l2',
                                    power_t=0.25, random_state=None,
                                    shuffle=True, tol=0.001,
                                    validation_fraction=0.1, verbose=0,
                                    warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.01, 1, 100, 1000, 10000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

In [118]:

```
alpha = best.best_params_["alpha"]
print(alpha)
```

```
0.01
```

In [119]:

```
clf = SGDRegressor(loss = "squared_loss", penalty = "l2", alpha = alpha)
clf.fit(train_df, tsne_train_output)
y_pred = clf.predict(test_df)
lr_test_predictions = [round(value) for value in y_pred]
y_pred = clf.predict(train_df)
lr_train_predictions = [round(value) for value in y_pred]
```

In [ ]:

```
type(lr_train_predictions)
```

Out[ ]:

```
list
```

Using Random Forest Regressor

Hyperparameter Tuning using RandomizedSearchCV

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

In [122]:

```
# find more about LinearRegression function here http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# -------------------------
# default paramters
# sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)

# some of methods of LinearRegression()
# fit(X, y[, sample_weight]) Fit linear model.
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict using the linear model
# score(X, y[, sample_weight]) Returns the coefficient of determination R^2 of the prediction.
# set_params(**params) Set the parameters of this estimator.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1-2-copy-8/
# -----------------------

from sklearn.model_selection import RandomizedSearchCV

regr1=RandomForestRegressor(max_features ='sqrt')


param_grid={
    "n_estimators": [40,60,80,100],
    "min_samples_split": [2,3,4,5,6],
    "max_depth": [3, None],
    "min_samples_leaf":[2,3,4]
}

best = RandomizedSearchCV(regr1,param_grid, scoring = "neg_mean_absolute_error",cv=2)
best.fit(train_df, tsne_train_output)
```

Out[122]:

```
RandomizedSearchCV(cv=2, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='sqrt',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_score=False,
                                                   random_state=None, verbose=0,
                                                   warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [3, None],
                                        'min_samples_leaf': [2, 3, 4],
                                        'min_samples_split': [2, 3, 4, 5, 6],
                                        'n_estimators': [40, 60, 80, 100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [123]:

```
best.best_params_
```

Out[123]:

```
{'max_depth': None,
 'min_samples_leaf': 4,
 'min_samples_split': 6,
 'n_estimators': 100}
```

In [124]:

```
regr1=RandomForestRegressor(max_features ='sqrt',max_depth=None, min_samples_split=6 ,
min_samples_leaf= 4,n_estimators=100)

regr1.fit(train_df, tsne_train_output)
y_pred = regr1.predict(test_df)
rndf_test_predictions = [round(value) for value in y_pred]
y_pred = regr1.predict(train_df)
rndf_train_predictions = [round(value) for value in y_pred]
```

In [125]:

```
#feature importances based on analysis using random forest
print (train_df.columns)
print (regr1.feature_importances_)
```

```
Index(['ft5', 'ft4', 'ft3', 'ft2', 'ft1', 'freq1', 'freq2', 'freq3', 'freq4',
       'freq5', 'Amp1', 'Amp2', 'Amp3', 'Amp4', 'Amp5', 'lat', 'lon',
       'weekday', 'exp_avg', 'exp_double_avg', 'exp_triple_avg'],
      dtype='object')
[0.03799344 0.05867348 0.0770907  0.09798597 0.14795531 0.00093226
 0.0003446  0.00038325 0.00037733 0.00046309 0.00372578 0.00243386
 0.00277193 0.00528795 0.00481332 0.00067146 0.00055067 0.00069605
 0.16200418 0.17201399 0.22283138]
```

Using XgBoost Regressor

In [126]:

```
#https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-p
ython/
xg = xgb.XGBRegressor(objective ='reg:squarederror')

param_grid={
    "learning_rate": [0.001,0.01,0.1],
    "n_estimators": [50,100,150,500,1000],
    "min_child_weight":[2,3,4],
}
best = RandomizedSearchCV(xg,param_grid, scoring = "neg_mean_absolute_error",random_state=5,cv=2)
best.fit(train_df, tsne_train_output)
```

Out[126]:

```
RandomizedSearchCV(cv=2, error_score=nan,
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None,
                                          objective='reg:squarederror',
                                          random_state=0, reg_...
                                          reg_lambda=1, scale_pos_weight=1,
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'learning_rate': [0.001, 0.01, 0.1],
                                        'min_child_weight': [2, 3, 4],
                                        'n_estimators': [50, 100, 150, 500,
                                                         1000]},
                   pre_dispatch='2*n_jobs', random_state=5, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [127]:

```
best.best_params_
```

Out[127]:

```
{'learning_rate': 0.1, 'min_child_weight': 3, 'n_estimators': 1000}
```

In [128]:

```python
xg = xgb.XGBRegressor(learning_rate =0.1,
n_estimators=1000,
min_child_weight = 3,
max_depth=3,
gamma=0,
subsample=0.8,
reg_alpha=200, reg_lambda=200,
colsample_bytree=0.8)
xg.fit(train_df, tsne_train_output)
```

```
[06:32:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
```

Out[128]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=3, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=200, reg_lambda=200, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.8, verbosity=1)
```

In [129]:

```python
y_pred = xg.predict(test_df)
xgb_test_predictions = [round(value) for value in y_pred]
y_pred = xg.predict(train_df)
xgb_train_predictions = [round(value) for value in y_pred]
```

In [130]:

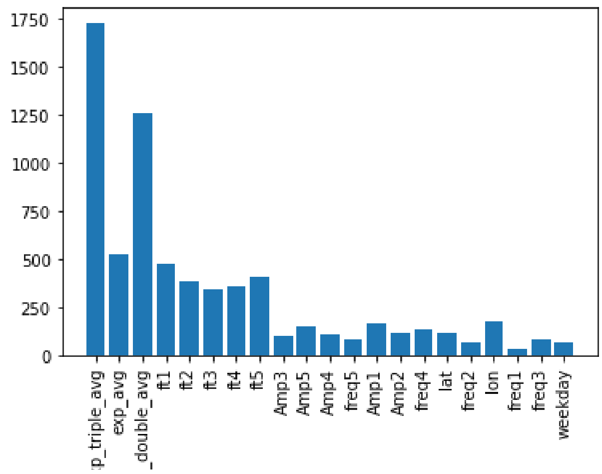
```python
#feature importances
#how to get feature importance in xgboost
#since xg.booster() gives error - https://github.com/TeamHG-Memex/eli5/issues/252
lst = xg.get_booster().get_score(importance_type='weight')
```

In [131]:

```python
k= lst.keys()
v=lst.values()
plt.bar(k,v)
plt.xticks(rotation=90)
```

Out[131]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
 <a list of 21 Text major ticklabel objects>)
```



ex exp.

Top 3 most important features are exp_avg, ft1 and ft5

Calculating error

In [132]:

```
#we are create two empty list which will store the train and test mape values
train_mape=[]
test_mape=[]

train_mape.append((mean_absolute_error(tsne_train_output,train_df['ft5'].values))/(sum(tsne_train_o
utput)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,train_df['exp_avg'].values))/(sum(tsne_tra
in_output)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,rndf_train_predictions))/(sum(tsne_train_c
utput)/len(tsne_train_output)))
train_mape.append((mean_absolute_error(tsne_train_output,
xgb_train_predictions))/(sum(tsne_train_output)/len(tsne_train_output)))

test_mape.append((mean_absolute_error(tsne_test_output, test_df['ft5'].values))/(sum(tsne_test_outp
ut)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output,
test_df['exp_avg'].values))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output,
rndf_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
test_mape.append((mean_absolute_error(tsne_test_output,
xgb_test_predictions))/(sum(tsne_test_output)/len(tsne_test_output)))
```

In [133]:

```
print("Error Metric Matrix (Tree Based Regression Methods) -  MAPE")
print("---------------------------------------------------------------------------------------------------
---------")
print("Baseline Model -                          Train: ",train_mape[0],"      Test: ",test_mape
0])
print("Exponential Averages Forecasting -        Train: ",train_mape[1],"      Test: ",test_mape
[1])
print("Random Forest Regression -                Train: ",train_mape[2],"     Test: ",test_mape[
2])
print("XgBoost Regression -                      Train: ",train_mape[3],"      Test: ",test_mape
3])
print("---------------------------------------------------------------------------------------------------
---------")
```

```
Error Metric Matrix (Tree Based Regression Methods) -  MAPE
--------------------------------------------------------------------------------------------------------
-
Baseline Model -                          Train:  0.23851987580892342       Test:
0.22953669590736295
Exponential Averages Forecasting -        Train:  0.14121603560900353       Test:
0.13490049942819257
Random Forest Regression -                Train:  0.06854967914079894      Test:
0.09474236283577367
XgBoost Regression -                      Train:  0.10018875757094267       Test:  0.09749414162
79181
--------------------------------------------------------------------------------------------------------
-
```

Imported various required libraries, some of the important libraries are like **dask** which is used to optimally load such a large data and perform various operations, **folium** used for interactive plotting, etc

We visualize how computation occurs in dask using **graphviz** library

Described the problem statement and defined key performance indicator(KPI)

Performed data cleaning on various features like **longitude**, **latitude**, **trip distance** etc and found outliers points in the data, we also plotted **Q-Q plot** to check wether its a log-normal distribution or not

Then we break the NewYork region into different clusters based on number of picks in each cluster using **k-means**, desired cluster **minimizes the inter-cluster points to 2miles** because we found that on a average a person can travel 2miles within 10mins

Then we converted our time into 10min bins

Loaded our 2016 data using dask dataframe

Performed **smoothing** to take care of presence of 0 pickups in at any point in any region.

Then we break our time series data into frequencey and amplitude using **fourier transform**.

Caluclated time series features like **simple moving averages , weighted moving average, exponential weighted moving averages** and compared our KPI(mean absolute percentage error and mean squared error) for each of these features and found that **exponential moving average has the lowest mean absolute error**

We split our data into 70% and 30% ratio in such a way that the newest data occurs in test data and older data in train data

Models like **linear regression , Random Forest and XGBoost** are applied calculated mean absolute error (MAPE) for each of the models, compared and found the **XBGoost** performs best with the test MAPE of **9%**

In [ ]:

```

```