Analysis of Uber Pickups in New York City Using K-Means Clustering

Problem statement

- Description :
- Dataset : Uber Trip Data 2014 of New York City
- Uber is a peer-to-peer ride sharing platform. Uber platform connects the cab drivers who can drive to the customer location. For calculating pricing to finding the optimal positioning of cars to maximize profits we need to build machine learning model. By using public dataset of Uber trips in K-Means clustering helps Uber in optimal pricing, the optimal position of cars in order to serve their customer faster and grow their business.

Import Required Libraries

In [32]:
```
import numpy as np
import pandas as pd
```

Read The DataSet

In [33]:
```
df1 = pd.read_csv("uber-raw-data-apr14.csv")
df2 = pd.read_csv("uber-raw-data-aug14.csv")
df3 = pd.read_csv("uber-raw-data-jul14.csv")
df4 = pd.read_csv("uber-raw-data-jun14.csv")
df5 = pd.read_csv("uber-raw-data-may14.csv")
df6 = pd.read_csv("uber-raw-data-sep14.csv")
```

Merge All Dataset into One

In [34]:
```
data_full = pd.concat([df1, df2, df3, df4, df5, df6])
```

In [35]:
```
data_full.shape
```

Out[35]:
```
(4534327, 4)
```

In [36]:
```
data_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4534327 entries, 0 to 1028135
Data columns (total 4 columns):
Date/Time    object
Lat          float64
Lon          float64
Base         object
dtypes: float64(2), object(2)
memory usage: 173.0+ MB
```

In [37]:
```
data_full.describe()
```

Out[37]:

	Lat	Lon
count	4.534327e+06	4.534327e+06
mean	4.073926e+01	-7.397302e+01
std	3.994991e-02	5.726670e-02
min	3.965690e+01	-7.492900e+01
25%	4.072110e+01	-7.399650e+01
50%	4.074220e+01	-7.398340e+01
75%	4.076100e+01	-7.396530e+01
max	4.211660e+01	-7.206660e+01

Selecting features

In [38]:
```
clus = data_full[['Lat', 'Lon']]
clus.dtypes
```

Out[38]:
```
Lat     float64
Lon     float64
dtype: object
```

In [39]:
```
data_full.head()
```

Out[39]:

	Date/Time	Lat	Lon	Base
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512

Apply K-Means Clustering On DataSet

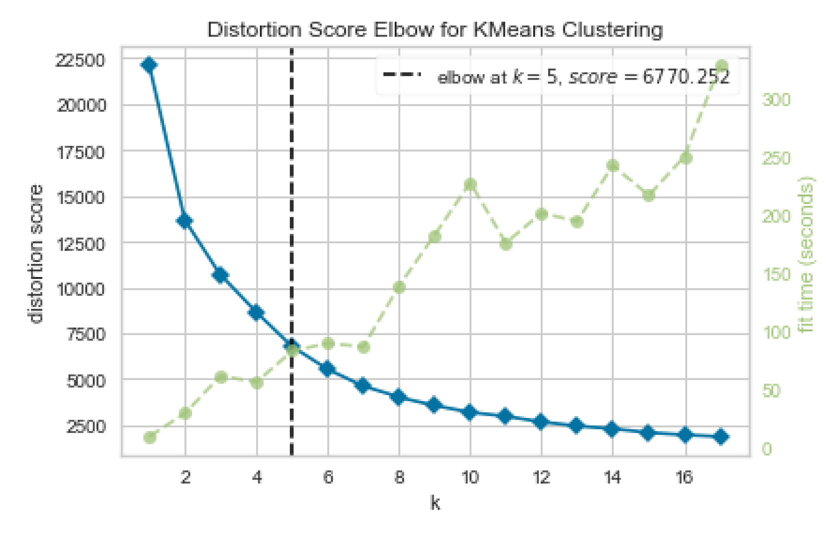
In [40]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
```

In [41]:
```
model = KMeans()
```

Select the Value of K using Elbow Method

- First step for any K-Means Clustering an unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered.
- The Elbow Method is one of the most popular methods to determine this optimal value of k.
- Distortion: It is calculated as the average/mean of the squared distances from the cluster centers of the respective clusters. Typically, the Euclidean distance metric is used.

In [8]:
```
visualizer = KElbowVisualizer(model, k = (1, 18))
visualizer.fit(clus)
visualizer.show()
```



Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2460a73e4c8>
```

Assigning a number of cluster in K-Means Algorithim

In [43]:
```
kmeans = KMeans(n_clusters = 5, random_state = 0)
kmeans.fit(clus)
```

Out[43]:
```
KMeans(n_clusters=5, random_state=0)
```

Stroring the Cluster Centroids

In [44]:
```
centroids = kmeans.cluster_centers_
centroids
```

Out[44]:
```
array([[ 40.71600413, -73.98971408],
       [ 40.66573796, -73.76418117],
       [ 40.79662299, -73.87899073],
       [ 40.76235269, -73.97687068],
       [ 40.69504708, -74.20164878]])
```

In [45]:
```
clocation = pd.DataFrame(centroids, columns = ['Latitude', 'Longitude'])
```

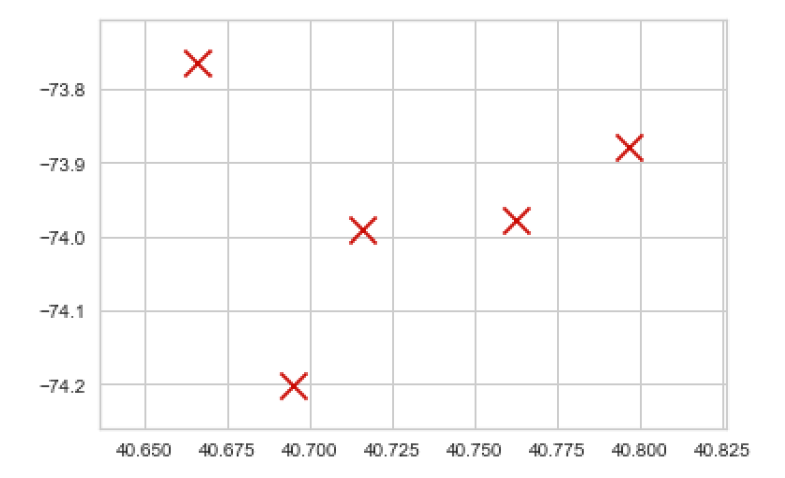
In [46]:
```
clocation.head()
```

Out[46]:

	Latitude	Longitude
0	40.716004	-73.989714
1	40.665738	-73.764181
2	40.796623	-73.878991
3	40.762353	-73.976871
4	40.695047	-74.201649

In [47]:
```
plt.scatter(clocation['Latitude'], clocation['Longitude'], marker = "x", color = 'R', s = 200)
```

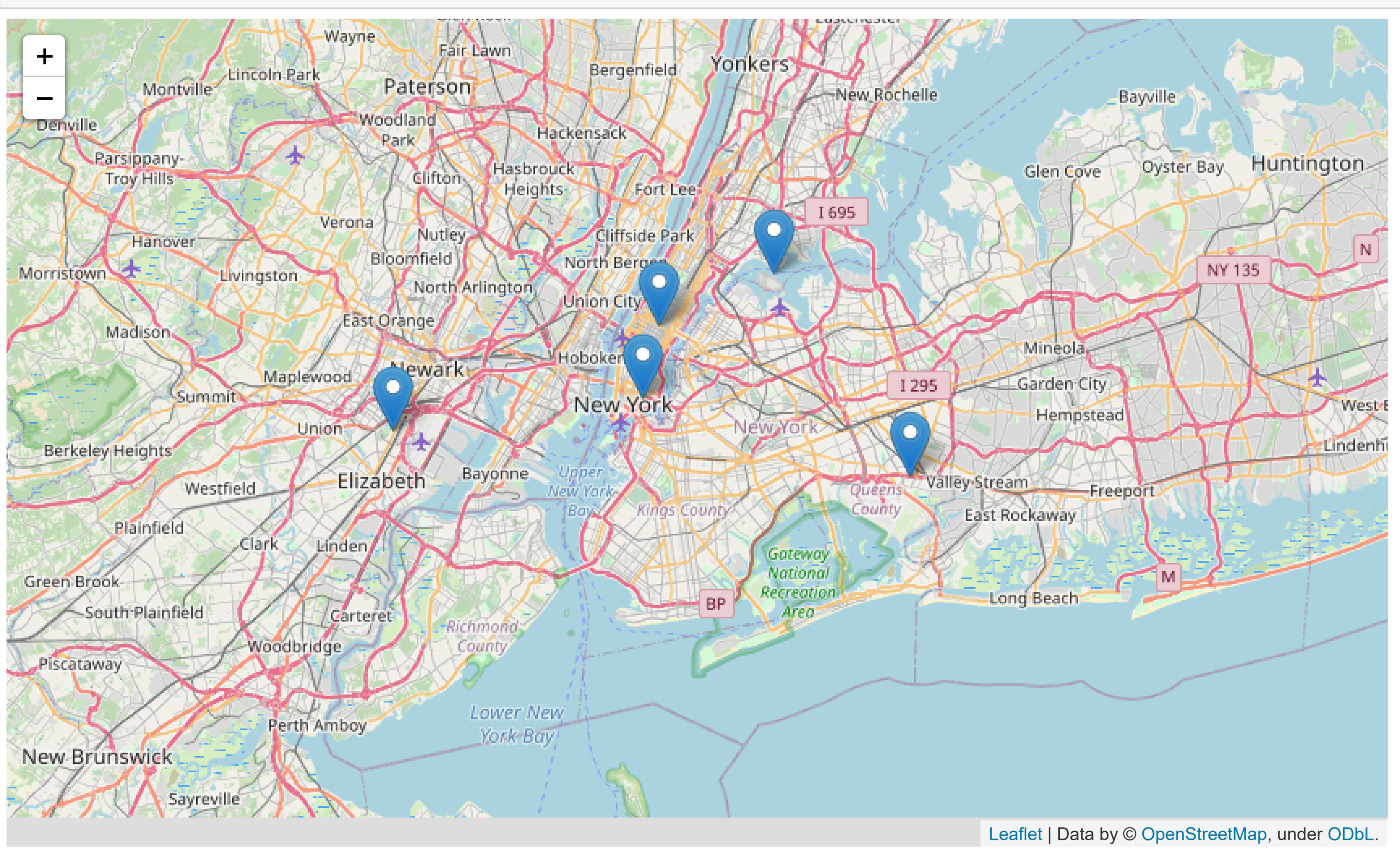
Out[47]:
```
<matplotlib.collections.PathCollection at 0x246102b3fc8>
```



Visualize Points on Map

In [48]:
```
import folium
centroid = clocation.values.tolist()

map = folium.Map(location = [40.71600413400166, -73.98971408426613], zoom_start = 10)
for point in range(0, len(centroid)):
    folium.Marker(centroid[point], popup = centroid[point]).add_to(map)

map
```

Out[48]:



In [49]:
```
label = kmeans.labels_
label
```

Out[49]:
```
array([3, 0, 0, ..., 2, 0, 0])
```

In [50]:
```
data_new = data_full
data_new['Clusters'] = label
data_new
```

Out[50]:

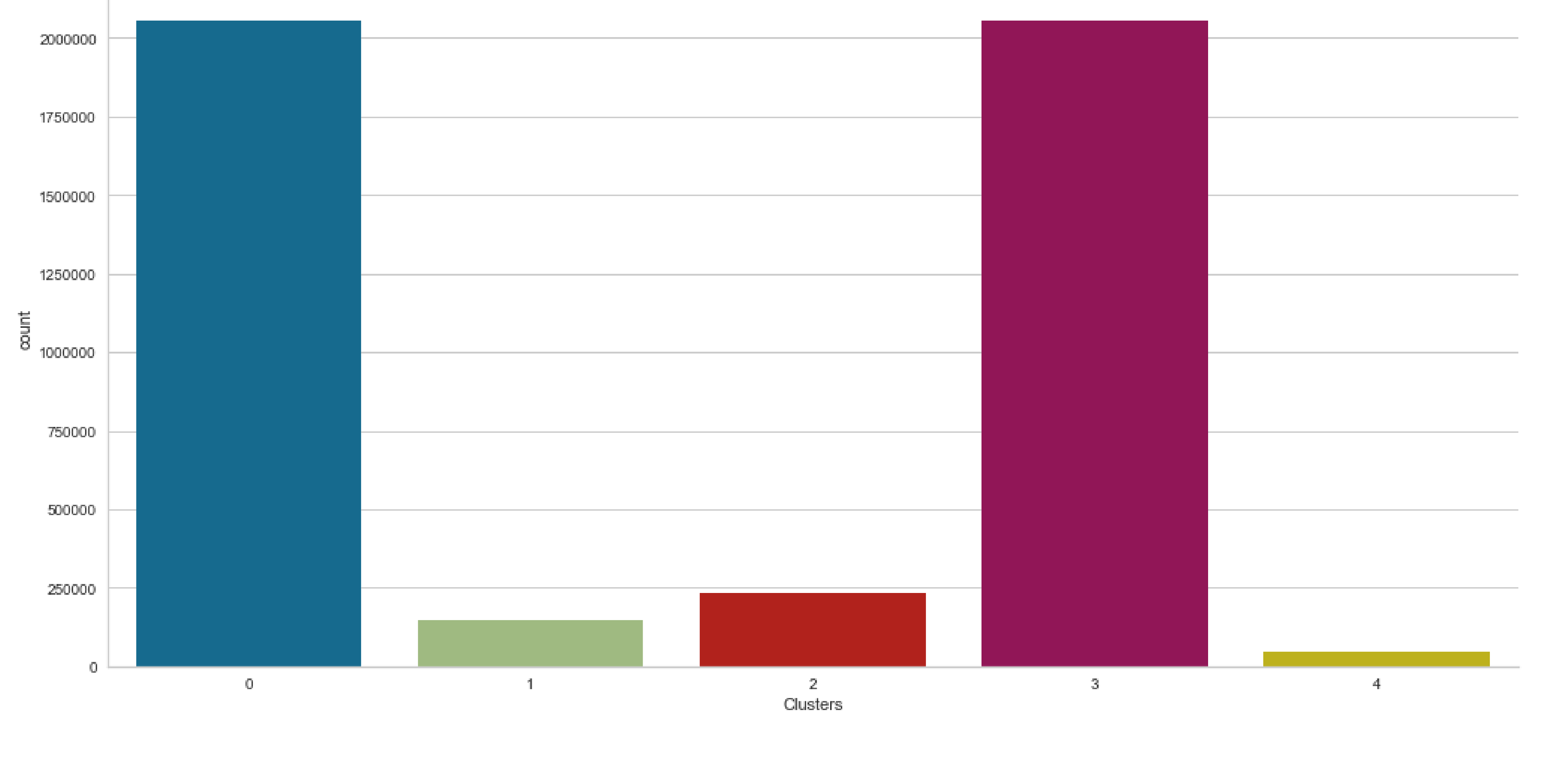
	Date/Time	Lat	Lon	Base	Clusters
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512	3
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512	0
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512	0
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512	3
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512	3
...	...	...	...	...	...
1028131	9/30/2014 22:57:00	40.7668	-73.9845	B02764	3
1028132	9/30/2014 22:57:00	40.6911	-74.1773	B02764	4
1028133	9/30/2014 22:58:00	40.8519	-73.9319	B02764	2
1028134	9/30/2014 22:58:00	40.7081	-74.0066	B02764	0
1028135	9/30/2014 22:58:00	40.7140	-73.9496	B02764	0

4534327 rows × 5 columns

Which cluster receives maximum ride request?

In [57]:
```
import seaborn as sb
sb.factorplot(data = data_new, x = "Clusters", kind = "count", size = 7, aspect = 2)
```

Out[57]:
```
<seaborn.axisgrid.FacetGrid at 0x2460673e888>
```



Compare the cluster

In [52]:
```
count_3 = 0
count_0 = 0
for value in data_new['Clusters']:
    if value == 3:
        count_3 += 1
    if value == 0:
        count_0 += 1
print(count_0, count_3)
```

```
2052540 2054339
```

Predict cluster for new location

In [53]:
```
new_location = [(40.86, -75.56)]
kmeans.predict(new_location)
```

Out[53]:
```
array([4])
```

Analysis of Uber Pickups in New York City Using K-Means Clustering

Data Visualization

In [11]:
```
import pandas
import seaborn
```

1. Importing data file

In [12]:
```
data = pandas.read_csv('uber-raw-data-apr14.csv')
```

In [13]:
```
data.head()
```

Out[13]:

	Date/Time	Lat	Lon	Base
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512

2. cleaning the data

In [14]:
```
data['Date/Time'] = data['Date/Time'].map(pandas.to_datetime)
```

In [15]:
```
data.head()
```

Out[15]:

	Date/Time	Lat	Lon	Base
0	2014-04-01 00:11:00	40.7690	-73.9549	B02512
1	2014-04-01 00:17:00	40.7267	-74.0345	B02512
2	2014-04-01 00:21:00	40.7316	-73.9873	B02512
3	2014-04-01 00:28:00	40.7588	-73.9776	B02512
4	2014-04-01 00:33:00	40.7594	-73.9722	B02512

In [16]:
```
data['Date/Time']
```

Out[16]:
```
0         2014-04-01 00:11:00
1         2014-04-01 00:17:00
2         2014-04-01 00:21:00
3         2014-04-01 00:28:00
4         2014-04-01 00:33:00
5         2014-04-01 00:33:00
6         2014-04-01 00:39:00
7         2014-04-01 00:45:00
8         2014-04-01 00:55:00
9         2014-04-01 01:01:00
10        2014-04-01 01:19:00
11        2014-04-01 01:48:00
12        2014-04-01 01:49:00
13        2014-04-01 02:11:00
14        2014-04-01 02:25:00
15        2014-04-01 02:31:00
16        2014-04-01 02:43:00
17        2014-04-01 03:22:00
18        2014-04-01 03:35:00
19        2014-04-01 03:35:00
20        2014-04-01 03:41:00
21        2014-04-01 04:11:00
22        2014-04-01 04:15:00
23        2014-04-01 04:19:00
24        2014-04-01 04:20:00
25        2014-04-01 04:26:00
26        2014-04-01 04:27:00
27        2014-04-01 04:38:00
28        2014-04-01 04:47:00
29        2014-04-01 04:49:00
                  ...
564486   2014-04-30 22:25:00
564487   2014-04-30 22:25:00
564488   2014-04-30 22:25:00
564489   2014-04-30 22:26:00
564490   2014-04-30 22:27:00
564491   2014-04-30 22:27:00
564492   2014-04-30 22:28:00
564493   2014-04-30 22:29:00
564494   2014-04-30 22:32:00
564495   2014-04-30 22:35:00
564496   2014-04-30 22:36:00
564497   2014-04-30 22:42:00
564498   2014-04-30 22:46:00
564499   2014-04-30 22:47:00
564500   2014-04-30 22:50:00
564501   2014-04-30 22:51:00
564502   2014-04-30 22:56:00
564503   2014-04-30 22:57:00
564504   2014-04-30 22:58:00
564505   2014-04-30 22:58:00
564506   2014-04-30 23:00:00
564507   2014-04-30 23:04:00
564508   2014-04-30 23:05:00
564509   2014-04-30 23:15:00
564510   2014-04-30 23:18:00
564511   2014-04-30 23:22:00
564512   2014-04-30 23:26:00
564513   2014-04-30 23:31:00
564514   2014-04-30 23:32:00
564515   2014-04-30 23:48:00
Name: Date/Time, Length: 564516, dtype: datetime64[ns]
```

In [17]:
```
def get_dom(dt):
    return dt.day

data['dom'] = data['Date/Time'].map(get_dom)
```

In [19]:
```
def get_weekday(dt):
    return dt.weekday()

data['weekday'] = data['Date/Time'].map(get_weekday)
```

In [20]:
```
def get_hour(dt):
    return dt.hour

data['hour'] = data['Date/Time'].map(get_hour)
```

In [21]:
```
data.head()
```

Out[21]:

	Date/Time	Lat	Lon	Base	dom	weekday	hour
0	2014-04-01 00:11:00	40.7690	-73.9549	B02512	1	1	0
1	2014-04-01 00:17:00	40.7267	-74.0345	B02512	1	1	0
2	2014-04-01 00:21:00	40.7316	-73.9873	B02512	1	1	0
3	2014-04-01 00:28:00	40.7588	-73.9776	B02512	1	1	0
4	2014-04-01 00:33:00	40.7594	-73.9722	B02512	1	1	0

In [22]:
```
data.tail()
```

Out[22]:

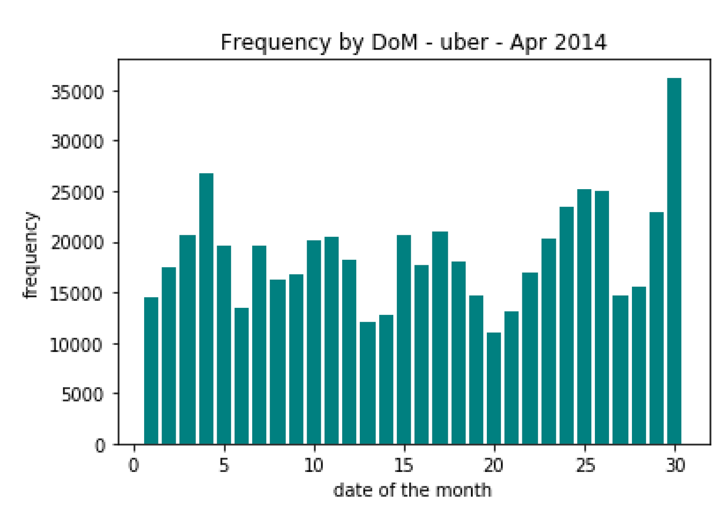
	Date/Time	Lat	Lon	Base	dom	weekday	hour
564511	2014-04-30 23:22:00	40.7640	-73.9744	B02764	30	2	23
564512	2014-04-30 23:26:00	40.7629	-73.9672	B02764	30	2	23
564513	2014-04-30 23:31:00	40.7443	-73.9889	B02764	30	2	23
564514	2014-04-30 23:32:00	40.6756	-73.9405	B02764	30	2	23
564515	2014-04-30 23:48:00	40.6880	-73.9608	B02764	30	2	23

3. data analysis

In [49]:
```
%pylab inline
hist(data.dom, bins=30, rwidth=.8, color="teal", range=(0.5, 30.5))
xlabel('date of the month')
ylabel('frequency')
title('Frequency by DoM - uber - Apr 2014')
```

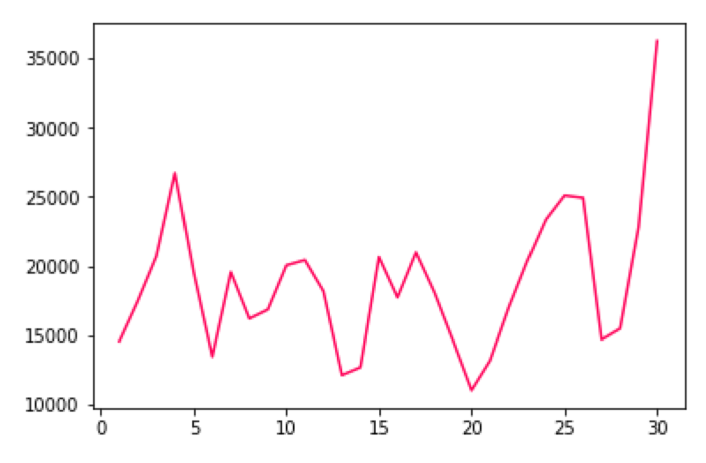
Populating the interactive namespace from numpy and matplotlib

Out[49]: Text(0.5, 1.0, 'Frequency by DoM - uber - Apr 2014')



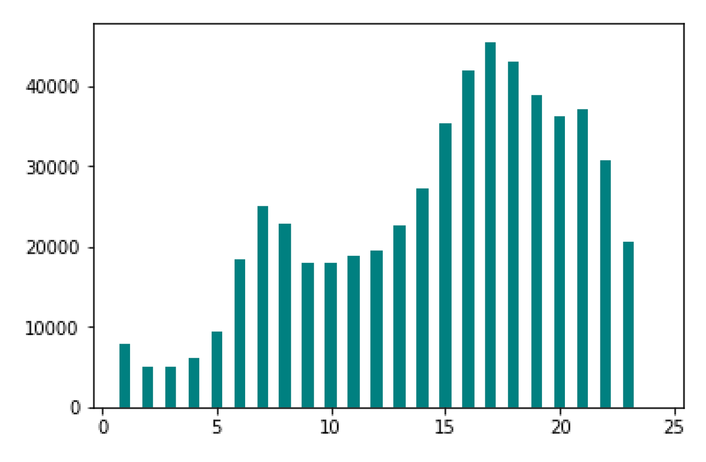
In [62]:
```
#for k, rows in data.groupby('dom'):
#    print((k, len(rows)))

def count_rows(rows):
    return len(rows)

by_date = data.groupby('dom').apply(count_rows)
plot(by_date, color='#FF0059')
```

Out[62]: [<matplotlib.lines.Line2D at 0x7fe6c73a66a0>]



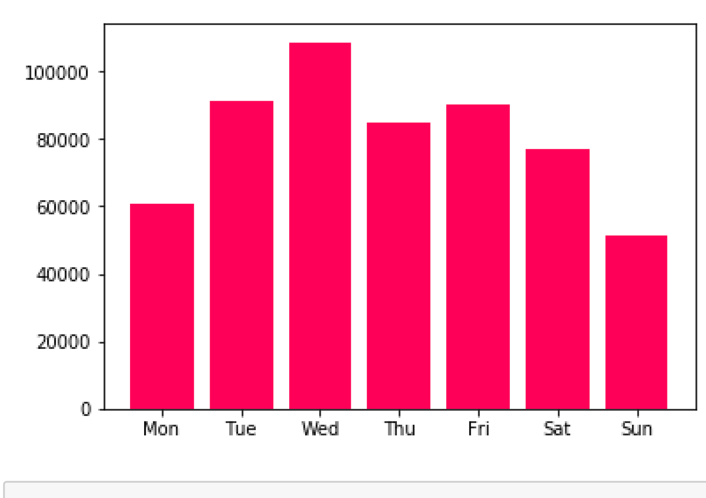
In [67]:
```
hist(data.hour, rwidth=0.5, bins=24, color='teal', range=(0.5, 24.5))
;
```

Out[67]: ''



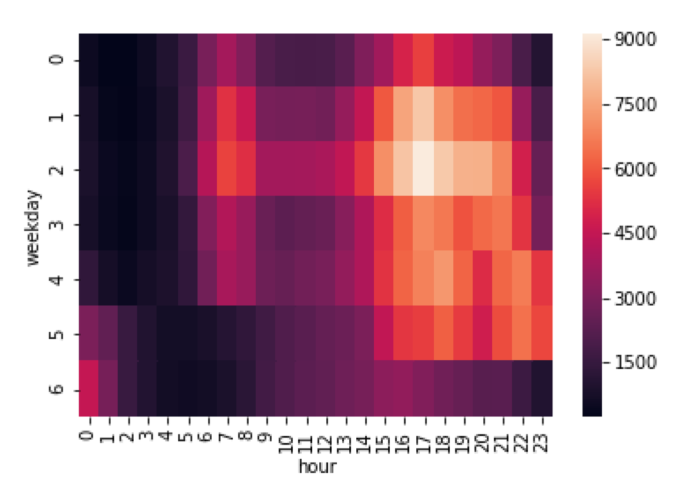
In [68]:
```
hist(data.weekday, bins=7, range =(-0.5,6.5), rwidth=.8, color='#FF0059')
xticks(range(7), 'Mon Tue Wed Thu Fri Sat Sun'.split())
;
```

Out[68]: ''



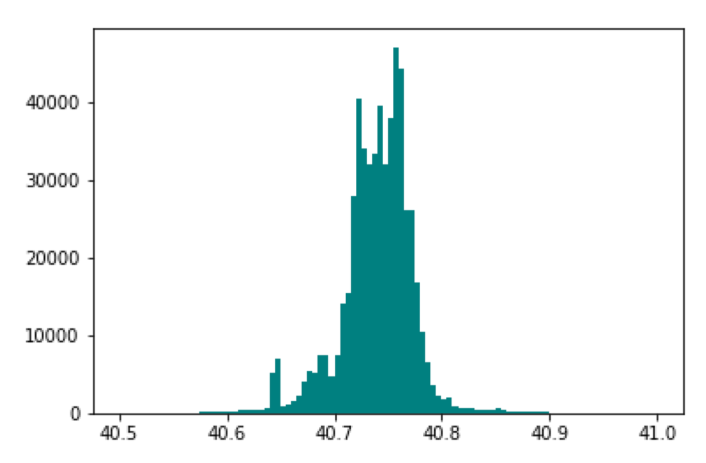
In [65]:
```
by_cross = data.groupby('weekday hour'.split()).apply(count_rows).unstack()
seaborn.heatmap(by_cross)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe6c632c2b0>



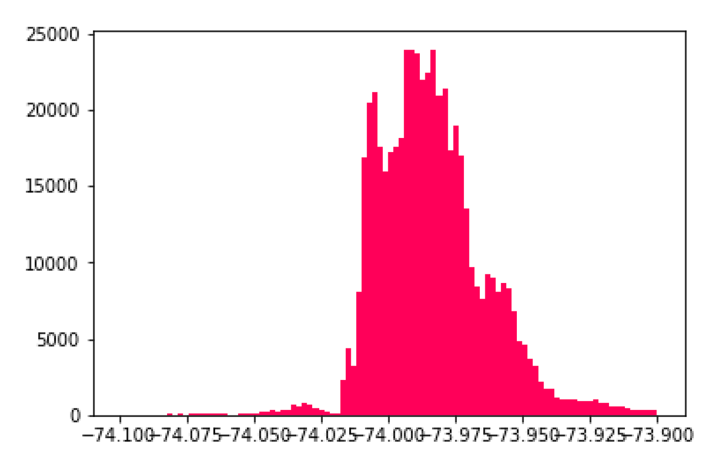
In [71]:
```
hist(data['Lat'], bins=100,color = 'teal', range = (40.5, 41))
;
```

Out[71]: ''



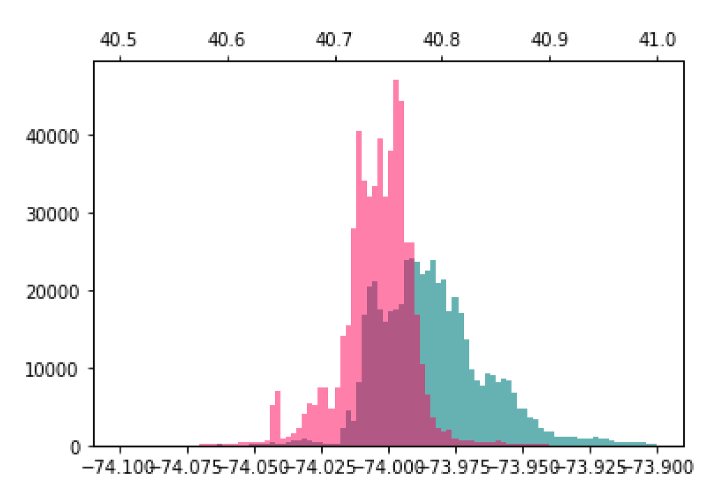
In [73]:
```
hist(data['Lon'], bins=100,color = '#FF0059', range = (-74.1, -73.9))
;
```

Out[73]: ''



In [87]:
```
hist(data['Lon'], bins=100, range = (-74.1, -73.9), color='teal', alpha=.6)
twiny()
hist(data['Lat'], bins=100, range = (40.5, 41), color='#FF0059', alpha=.5)
;
```

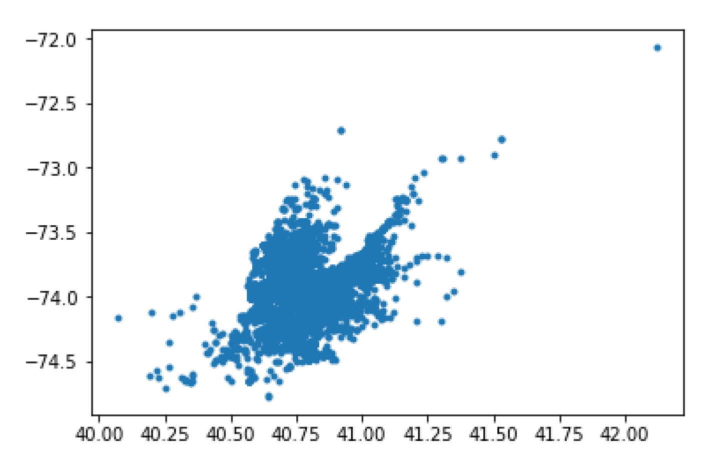
Out[87]: ''



4. Working on Longitude and Lattitude

In [111]:
```
plot(data['Lat'],data['Lon'], '.')
```

Out[111]: [<matplotlib.lines.Line2D at 0x7fe6cb278b38>]



In [ ]: