Hot Routes

Niger Little-Poole

I've completed the following uses python and non propreitary python libraries.

I like to start by importing someuseful libraries and just getting a general overview of what is in the dataset. I've transitioned to using mpld3 in conjunction with matplotlib for graphing since at Tumblr it allows us to build D3 graphs directly in python. Being as I am the only team member who writes D3 javascript, this has been a great timesaver.

In [1]:
```
import matplotlib as mpl
mpl.use('Agg')
```

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import mpld3
import random
import math
import copy
import pandas as pd
from sklearn.cluster import KMeans
```

In [3]:
```
rides = pd.read_csv('rides.csv')
print rides.head()
```

```
                       hack_license      pickup_datetime  start_la
t  \
0  58894369838B3783D30EEAA92B364BEB  2013-08-11 10:16:00  40.76103
2
1  6AC79A2B688191545021B8CF96ED385F  2013-08-11 10:00:00  40.77390
7
2  C52F793C850CD7CBCC1875FD79A9414C  2013-08-11 10:04:00  40.76189
0
3  5946C99EE9D4216C0A7395E525F2ED5F  2013-08-11 10:04:00  40.76514
1
4  7C228290C05CE8E37E0FE4C1B806DF88  2013-08-11 10:09:00  40.73115
2

   start_lng     dropoff_datetime    end_lat    end_lng  distanc
e_miles  \
0 -73.987091  2013-08-11 10:19:00  40.758083 -73.985519
0.34
1 -73.873360  2013-08-11 10:15:00  40.648796 -73.782616
12.11
2 -73.986977  2013-08-11 10:15:00  40.818638 -73.951485
5.56
3 -73.980568  2013-08-11 10:13:00  40.780441 -73.972839
1.44
4 -73.982193  2013-08-11 10:12:00  40.730576 -73.975960
0.62

   duration_secs  passenger_count
0            180                1
1            900                1
2            660                1
3            540                6
4            180                1
```

In [5]:
```
rides.drop_duplicates().count()
```

Out[5]:
```
hack_license        2835720
pickup_datetime     2835720
start_lat           2835720
start_lng           2835720
dropoff_datetime    2835720
end_lat             2835712
end_lng             2835712
distance_miles      2835720
duration_secs       2835720
passenger_count     2835720
dtype: int64
```

Exploratory Analysis

In [4]:
```
rides.describe()
```

Out[4]:

	start_lat	start_lng	end_lat	end_lng	dista
count	2835723.000000	2835723.000000	2835715.000000	2835715.000000	28357
mean	40.749596	-73.975251	40.750119	-73.974743	2.967
std	0.027221	0.036386	0.030818	0.035539	3.431
min	40.080708	-74.673485	40.055046	-74.772507	0.000
25%	40.735657	-73.992622	40.734795	-73.991821	1.000
50%	40.752319	-73.982330	40.752762	-73.981117	1.940
75%	40.765671	-73.969101	40.766369	-73.966469	3.250
max	41.758438	-71.757393	41.756645	-71.757393	50.00

So I'm pretty sure these lat, long ranges limit the dataset to just the New York City metro area as drawing a rectangular box with the min/max of the lng,lat reveals New York City, long island, southern Conneticut, and Eastern New Jersey (all regions of the NYC Metro) . At latitude 41, a degree of latitude corresponds to 69 miles while a degree of longitude corresponds to 52 miles. Given that $\alpha = 84.2$ and $\beta = 111.2$, this makes sense since it appears that these parameters are attemtping to scale the two parts of the coordinate to equal weight. This was confirmed after asking the question to the team.

The duration and time information is cool but doesn't seem to be extremely necessary at the moment. Currently the prompt is solely to find the 5 hot routes that maximize expected # of rides. In reality I assume there would be a need to also try to minimize trip durations & etc but being as that isn't the task at hand I can focus on the lat,lng pairs.

Now that I have an idea of what the dataset looks like, my first intution is to try to graphically see what the rides look like geometrically and see if I can naively spot any patterns. I'll take a random sample of 1000 rides(without replacement) and graph them as vectors to see what I can find.

In [14]:
```
n = 1000
test = rides.sample(n=n)
x = test.as_matrix(columns=["start_lat","end_lat"])
y = test.as_matrix(columns=["start_lng","end_lng"])

fig = plt.figure()
plt.grid(b=True, which='major', color='b', linestyle='-', alpha =
.2)
ax = fig.add_subplot(111)
for i in range(n):
    plt.plot(x[i],y[i], lw = '1', dash_joinstyle='round',solid_caps
tyle="round")
plt.title('Ride Vectors', fontsize=26)
plt.rc('axes', color_cycle=['r', 'g', 'b', 'y'])

mpld3.display(fig)
plt.close()
```

Without actually representing these routes on a real map its hard to see geographically what the trends are. However there is a dense concentration of rides in a certain region of the map which I'm fairly sure intuitively are rides occuring closer to the more population dense city. Its hard to visually see any patterns in the super dense region of the center area, however there does seem to be some similarity in vectors between some of the subsets of vectors in the outer rim. Its quite possible that there are hot routes to be discovered here.

Methodology

My approach to solving this problem will be to cluster the rides into 5 similar groups. Then I will I try to find the optimal hot route for each subset of the rides. To cluster the groups, I'll use the K Means algorithm to create 5 centroids that represent 5 groups. I'll use euclidian distance, scaling lng and lat with alpha and beta beforehand, in order to cluster. Other distance functions aren't guarenteed to converge so I don't want to complicate things by introducing them.

I'm choosing this methodology because I'm not currently aware of a deterministic way to solve this problem with something like a Linear/Integer Program. In order to complete this on time and in a way I could explain, I decided to try a more heuristic approach.

Clustering

In [6]:
```
# scaling values
scaled = rides
scaled['start_lat'] = scaled['start_lat'].apply( lambda x: x * 11
1.2)
scaled['end_lat'] = scaled['end_lat'].apply( lambda x: x * 111.2)
scaled['start_lng'] = scaled['start_lng'].apply( lambda x: x * 8
4.2)
scaled['end_lng'] = scaled['end_lng'].apply( lambda x: x * 84.2)
scaled=scaled.dropna()
print scaled.head()
print scaled.describe()
```

```
                       hack_license      pickup_datetime    star
t_lat  \
0  58894369838B3783D30EEAA92B364BEB  2013-08-11 10:16:00  4532.626
758
1  6AC79A2B688191545021B8CF96ED385F  2013-08-11 10:00:00  4534.058
458
2  C52F793C850CD7CBCC1875FD79A9414C  2013-08-11 10:04:00  4532.722
168
3  5946C99EE9D4216C0A7395E525F2ED5F  2013-08-11 10:04:00  4533.083
679
4  7C228290C05CE8E37E0FE4C1B806DF88  2013-08-11 10:09:00  4529.304
102

     start_lng     dropoff_datetime       end_lat      end_lng  dis
tance_miles  \
0 -6229.713062  2013-08-11 10:19:00  4532.298830 -6229.580700
0.34
1 -6220.136912  2013-08-11 10:15:00  4520.146115 -6212.496267
12.11
2 -6229.703463  2013-08-11 10:15:00  4539.032546 -6226.715037
5.56
3 -6229.163826  2013-08-11 10:13:00  4534.785039 -6228.513044
1.44
4 -6229.300651  2013-08-11 10:12:00  4529.240051 -6228.775832
0.62

   duration_secs  passenger_count
0            180                1
1            900                1
2            660                1
3            540                6
4            180                1
           start_lat       start_lng         end_lat          en
d_lng  \
count  2835715.000000  2835715.000000  2835715.000000  2835715.000
000
mean      4531.355093    -6228.716174     4531.413273    -6228.673
392
std          3.026978        3.063662        3.426983        2.992
419
min       4456.974730    -6287.507437     4454.121115    -6295.845
089
25%       4529.805058    -6230.178772     4529.709204    -6230.111
328
50%       4531.657873    -6229.312186     4531.707134    -6229.210
051
75%       4533.142615    -6228.198304     4533.220233    -6227.976
690
max       4643.538306    -6041.972491     4643.338924    -6041.972
491

       distance_miles   duration_secs  passenger_count
count  2835715.000000  2835715.000000   2835715.000000
```

```
mean          2.967329     747.646067         1.743620
std           3.431377     541.307216         1.414145
min           0.000000     -10.000000         0.000000
25%           1.000000     367.000000         1.000000
50%           1.940000     600.000000         1.000000
75%           3.250000     960.000000         2.000000
max          50.000000    7200.000000         6.000000
```

In [7]:

```
vals = [ [x[2],x[3], x[5],x[6]] for x in scaled.values if x[2] and
x[3] and  x[5] and x[6] ]
print vals[0]
```

```
[4532.6267584, -6229.7130622, 4532.2988296, -6229.5806998]
```

In [88]:

```
k = 5
model = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_ini
t=5)
model.fit(vals)
```

Out[88]:

```
KMeans(copy_x=True, init='k-means++', max_iter=100, n_clusters=5,
n_init=5,
    n_jobs=1, precompute_distances='auto', random_state=None, to
l=0.0001,
    verbose=0)
```

In [89]:

```
model.cluster_centers_
centroids = model.cluster_centers_
print centroids
```

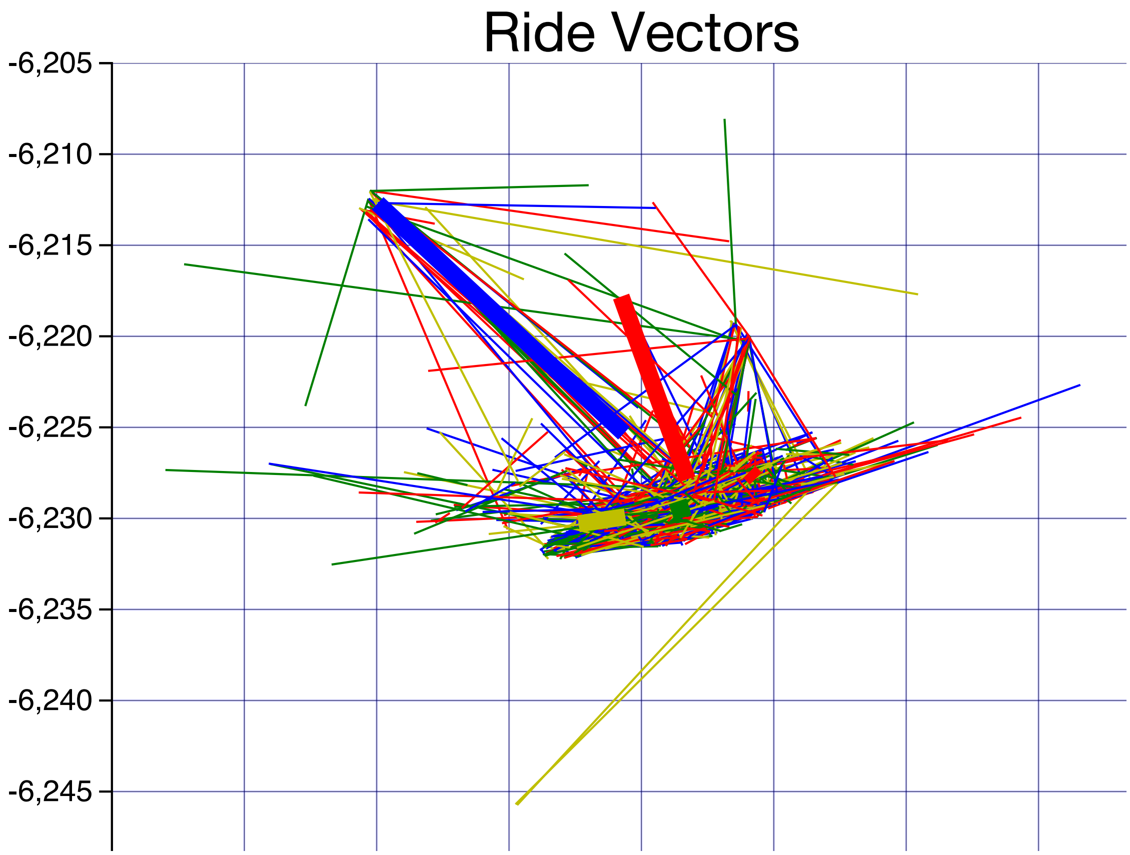
```
[[ 4533.90806137 -6227.41797144  4534.60337671 -6226.98976947]
 [ 4529.20173715 -6230.0400732   4527.96141902 -6230.21365891]
 [ 4531.53886892 -6227.36170965  4523.15257028 -6214.20040078]
 [ 4520.07849722 -6212.71357612  4529.42755611 -6225.43933437]
 [ 4531.55819733 -6229.4269058   4531.81580472 -6229.32689426]]
```

In [31]:

```
a = []
b = []
for x in centroids:
    temp = []
    a.append([x[0],x[2]])
    b.append([x[1],x[3]])
print a
print b
n = 1000
test = rides.sample(n=n)
x = test.as_matrix(columns=["start_lat","end_lat"])
y = test.as_matrix(columns=["start_lng","end_lng"])

fig = plt.figure()
plt.grid(b=True, which='major', color='b', linestyle='-', alpha =
.2)
ax = fig.add_subplot(111)

for i in range(n):
    plt.plot(x[i],y[i], lw = '1', dash_joinstyle='round',solid_caps
tyle="round")
for i in range(5):
    plt.plot(a[i],b[i], lw = '8', dash_joinstyle='round',solid_caps
tyle="round")

plt.title('Ride Vectors', fontsize=26)
plt.rc('axes', color_cycle=['r', 'g', 'b', 'y'])
mpld3.display(fig)
```

```
[[4534.0741444788964, 4534.3235098290188], [4531.1405582491279, 45
31.7986205697971], [4520.0357313140157, 4529.3408882680551], [452
9.388575098802, 4527.6253774878451], [4531.7358561955361, 4529.232
1095675106]]
[[-6227.4337582313829, -6227.8014903434541], [-6229.6597251170069,
-6229.3411812964487], [-6212.6897650214923, -6225.3105438793546],
[-6229.9010468762253, -6230.2831155560089], [-6227.8420562119563,
-6217.8073001368275]]
```

Out[31]:



The centroids from K means give an idea of what the 5 most average set of rides are(the thick lines). Now that I have each ride assigned to a group, I can begin to create a optimal hot route for each group. The sum of the expected values for each hot route should represent the expected number of rides for the set of hot routes. If the clusters are made well, each group should represent riders that would pick the same hot route given the set of hot routes, since their rides were closer together in starting and destination points.

Generating Hot Routes

Stochastic Gradient Descent

Now that I have the 5 groups/centroids. I'll attempt to maximize expected # of rides using stochastic gradient descent. The expected # of rides for a group is the sum of the $\Sigma f(H_j , r_k)$ for all k since the expected value for one ride is just the probability of the ride multiplied by the # of rides which is 1.

$\Sigma f(H_j , r_k)$ increases as the manhattan distance between H_j *and* r_k decreases. If we label the negative manhattan distance between the hot route and the ideal route as the error e, such that $e = 0 - dist$, then we can make use of a convex loss function, specifically $1/2e^2$ since minimizing e would maximize the probability. $1/2e^2$ is simply the mean squared error multiplied by a half so that when taking later derivatives the squared power cancels out. Using a convex loss function ensures that the stochastic gradient descent algorithm will eventually converge to a minima. In accordance with the stochastic gradient descent algorithm, for each point in the group, we can take the partial derivative of e in relation to each feature in the r_k and use that to update H_j. The larger the manhattan difference, the more the gradient descent will try to update H_j to compensate and vice versa. In this case the two partial derivatives are $\pm e$ for each feature in each set of coordinates.

Stochastic gradient descent also needs a step size in order to do the updates. I've chosen a step size of .1 as after fiddling around for a bit these seemed to give decent results.

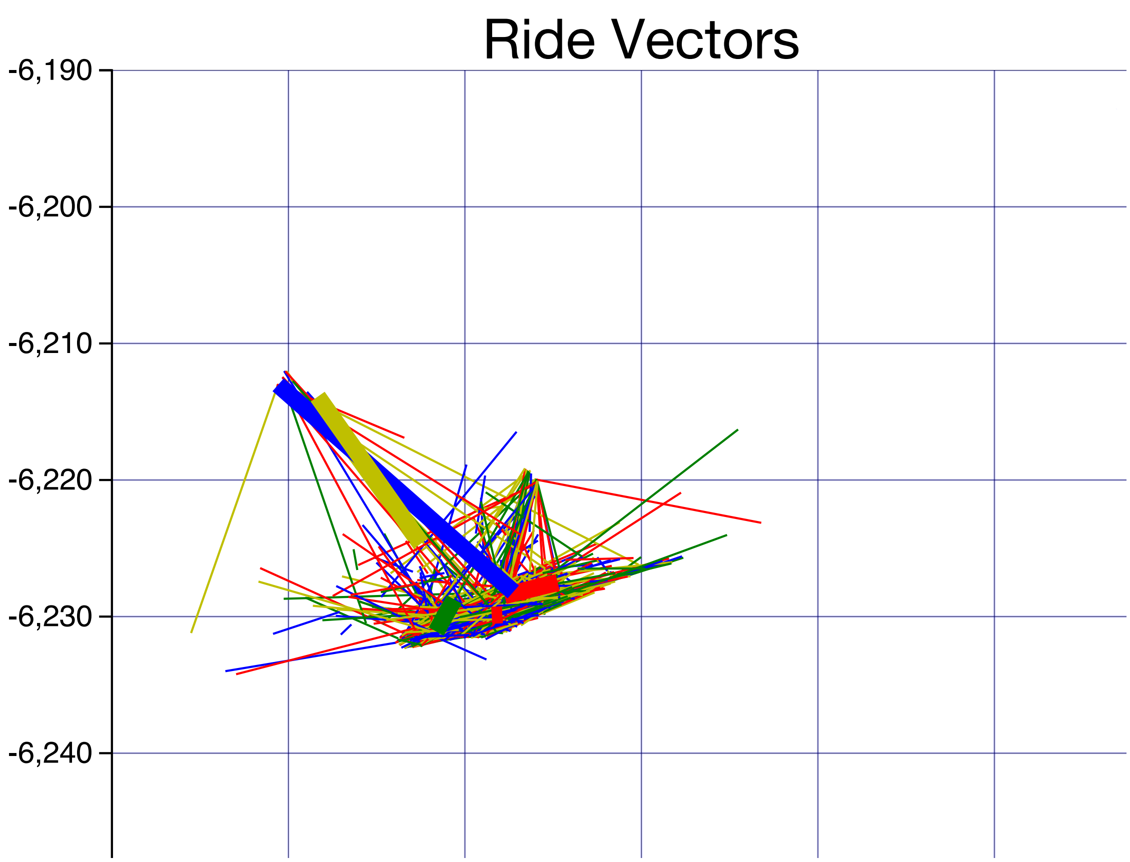
In [116]:

```
def manhattan_dist(h,r):
    return abs(h[0] - r[0]) + abs(h[1] - r[1]) + abs(h[2] - r[2]) +
abs(h[3] - r[3])
def euclid_dist(h,r):
    return math.sqrt ( math.pow(h[0] - r[0],2) + math.pow(h[1] -
r[1],2) + math.pow(h[2] - r[2],2) + math.pow(h[3] - r[3],2) )
def ride_probability(h, r):
    dist = manhattan_dist(h,r)
    return math.exp(-1*dist)
def gradient(h,r):
    dist = manhattan_dist(h,r)
    return [-1*dist if h[i] >= r[i] else dist  for i in range(4)]
def add(a,b,step=1):
    res = []
    for i in range(4):
        res.append(a[i]*step + b[i])
    return res
def stochastic_descent(cens, values, iterations = 1):
    cens = copy.deepcopy(cens)
    for i in range(iterations):
        for v in values:
            index = model.predict(v)[0]
            g = gradient(cens[index],v)
            f = add(g,cens[index],.1)
            cens[index] = f
    return cens
routes = stochastic_descent(centroids,vals,5)
print routes
expected = sum([ ride_probability(routes[model.predict(v)[0]], v) f
or v in vals] )
print expected
```

```
[[ 4532.24307248 -6228.43493141  4535.25537398 -6227.47456626]
 [ 4529.39056109 -6228.71224716  4528.37240248 -6231.06808402]
 [ 4532.7394233  -6228.17067918  4519.41836737 -6213.00232298]
 [ 4521.65809928 -6213.87543332  4527.52405372 -6224.61782518]
 [ 4532.12268577 -6229.83737004  4531.51585362 -6229.88359966]]
85463.7713953
```

In [113]:
```
a = []
b = []
for x in routes:
    temp = []
    a.append([x[0],x[2]])
    b.append([x[1],x[3]])
n = 1000
test = rides.sample(n=n)
x = test.as_matrix(columns=["start_lat","end_lat"])
y = test.as_matrix(columns=["start_lng","end_lng"])

fig = plt.figure()
plt.grid(b=True, which='major', color='b', linestyle='-', alpha =
.2)
ax = fig.add_subplot(111)

for i in range(n):
    plt.plot(x[i],y[i], lw = '1', dash_joinstyle='round',solid_caps
tyle="round")
for i in range(5):
    plt.plot(a[i],b[i], lw = '8', dash_joinstyle='round',solid_caps
tyle="round")

plt.title('Ride Vectors', fontsize=26)
plt.rc('axes', color_cycle=['r', 'g', 'b', 'y'])
mpld3.display(fig)
```

Out[113]:



The hot routes computed are signifigantly different than the average vector of the centroids. Stochastic gradient descent relatively minimizes the error, in the case the manhattan distance, between the hot route and elements of the centroids set. The expected value laid out isn't exact as I didn't run gradient descent until convergence but rather on a fixed # of iterations so that it wouldn't take too long to compute. This is an instance where having some faster tools on hand such as Spark or Map Reduce on a couple of clusters would be nice.

In [112]:
```
hot_routes = [["start_lat", "start_lng","end_lat","end_lng"]]
for route in routes:
    temp = []
    temp.append(route[0] / 111.2)
    temp.append(route[1] / 84.2)
    temp.append(route[2] / 111.2)
    temp.append(route[3] / 84.2)
    hot_routes.append(temp)
print hot_routes
with open('hot_routes.csv', 'w') as f:
    a = csv.writer(f, delimiter=',')
    a.writerows(hot_routes)
```

```
[['start_lat', 'start_lng', 'end_lat', 'end_lng'], [40.75758158702
9804, -73.971911299368855, 40.784670629276384, -73.96050553757098
1], [40.731929506228632, -73.975204835573294, 40.722773403601799,
-74.003183895713065], [40.762045173536833, -73.968772911870772, 4
0.642251505169462, -73.788626163674635], [40.66239297916016, -73.7
98995645120669, 40.715144368028142, -73.926577496165322], [40.7564
98972736793, -73.988567340099181, 40.751041849092232, -73.98911638
5477814]]
```

Results

I used google maps to determine where these five routes actually map to and from. I uncovered the following:

- Route 1: 51st & Park Ave to 90th Street & 5th Ave
- Route 2: Stuytown to Broome & West Broadway
- Route 3: 60th and Lexington to JFK Airport
- Route 4: JFK to East Williamsburg
- Route 5: 42nd between 7th & 8th (Times Square) and Penn Station

Given that I'm a life long New Yorker, I'll use some of my personal domain knowledge to interpret these results. Since the hot routes are just representing trends, its not so much that these exact starting and ending destinations are key points, but rather there is a lot of traffic with these trends. My thoughts are the following:

The first route is both midtown to central park east as well as connecting midtown to Mt Sinai. These areas are connected by the Lexington Ave subway but this is the most congested line in the system and this route is actually spanning the most congested part of that line.

The second route connects Stuyvesant Town and the west part of Soho. The former is a large residential complex with numerous NYU students and young post grads while the latter is an upscale fashion and restaurant area. These areas are not directly connected by subway and one of the lines that helps connect them (L) is considered the worst service in the subway system.

The third hot route connects midtown to the airport, which makes a lot of sense. Midtown is the commercial center of Manhattan and JFK is the largest and international airport. It is quite possible that there are also tourists making this journey as midtown has a lot of Manhattan's hotels.

The fourth route connects JFK to East Williamsburg. This region of brooklyn has the least subway access and is once again serviced by the infamous L. It is quite likely that this pattern is emerging from a lack of good trasnportation options between the airport and East Brooklyn, as before the introduction of green cabs, cab drivers also prefered to do trips to Manhattan and not the outer boroughs. Even now, there are signifigantly less green cabs than yellow cabs

The fifth route connects Times Square region with Penn Station. These are the two of the most population dense areas of Manhattan, so it make sense that there is a lot of service in between. Two of the largest transportation hubs (Penn Station and Port Authority) are also connected by this route. Additionally the theatres on Broadway are being connected to the shopping/restaurants on 34th. New Yorkers don't really live in either of these areas and if they do aren't likely to be taking cabs through midtwon for such short distances. This is very likely tourist driven traffic as it conects transportation, tourist landmmarks, shopping, dining , and other service activities.

Intuitive Take aways:

None of these are conclusive or even proven, but if I were to look into next steps it would be confirming these hypothesis:

- Route 1: servicing midtown east to the upper east side, compensating for congested/minimal subway access. Likely tourist and hospital commuters
- Route 2: servicing Lower East side to Soho, compensating for lack/inconvenint subway. Likely serviced by young adult crowd
- Route 3: servicing Midtown to the airport, typically cab journey. Likely servicing businessmen or tourist
- Route 4: servicing JFK to east brooklyn, compensating for poor subway service, likely serviced by locals
- Route 5: Times Square to Penn Station, likely tourists moving between landmarks or transportation hubs
- None of the routes seem unexpected, which gives a bit of assurance that these results are somewhat valid. Though in some cases that could just be coincidence and creat false confidence.

After writing all of this code I have generated a csv (hot_routes.csv) with the lat,lng pairs for each hot route. These hot routes maximize expected # of rides by minimizing the manhattan distances for the corresponding historical rides.The hot_routes generated generally don't have high probabilities of rides(< 20%) for each historacal ride, but since the number of rides is very large the expected value is still relatively high. The expected number of rides for the result is about 3%-4% of the historical # of rides. Now depending on what the hot routes are being used for, this is a positive or negative result. If one is trying to replace free chosen rides with 5 fixed hot routes, then this is not great because it signifigantly reduces the # of rides. However if one is contemplating introducing a Lyft Shuttle service, than this indicates that there are 5 routes that would have potentially a lot of demand. This exercise is also useful in uncovering potential service patterns within the historical data.

Assumptions

The following relies on a couple of assumptions:

- Dataset is clean and accurate, pandas might have caught a single digit amount of duplicates but I ignored that sinc its relatively insignifigant to the 2.8 millino rides in the set
- After optimizing the hot routes for each centroid group, the closest hot route will still match the original centroid from k means. This essentially relies on the centroids not being very close together so that optimizing the hot route doesn't change the label of rides on the outer boundary of clustered region
- I assume longitude and latittude accurately represent the distance of travel. In the city this is relatively okay but in areas that have elevation and terrain differences, even such as New Jersey, there are other factors other than two dimensional euclidian/manhattan distance that are going to be influences.
- I assume time of day is not important to the hot routes, its very possible that the optimal hot routes change at different points of the day/day of the week due to different travel patterns. The prompt did not ask for this level of complexity so I didn't address it

- I assume duration is not important to the probability someone will take a ride or is represented by geographic distance. Its completely possible that factors such as traffic(especially in New York) make it so that the distance is not sufficient to measure rider satisfaction.
- Seasonality isn't a factor, all the data comes from a short period in August 2 years ago. Its entirely possible these trends are outdated or only valid within a certain season.

This is not to say that the following conclusions aren't useful, there are just aspects of the problem that the feature set is not capturing at that reduces the precision of the results.