Data_102_Project_02

December 14, 2019

0.0.1 1 Bandits

1. Formalizing the problem as a multi-armed bandits problem First formalize the problem as a multi-armed bandits problem to maximize the number of flyers given out at the street corner, over some number of days for which the promotion will take place.

- What are the arms? What are the rewards? Would you model the rewards as subgaussian or as bounded?
- What is the time horizon?
- What are the modeling assumptions?
- Which of your assumptions do you expect to be reasonable? Which might you not expect to hold? Why? Which assumptions can you test by gathering data?
- What notion of regret are you considering? In particular, which strategy (possibly with perfect knowledge) would you compare your adaptive strategy to?

To formalize the problem as a multi-armed bandits problem in order to maximize the number of flyers given out at the street corner over a number of days for the promotion, I think we could model the arms of the bandit problem as the street corners and the rewards as the number of flyers given out at the street corner. The distributions for the rewards for the street corners would be bounded in my opinion and not subgaussian because there is a limit to how little flyers that can be passed out, 0, and there is also a physical max of flyers that can be passed out by the person determined by what they can carry or what is available to them. The time horizon in this case would be the number of promotional days since those are the only days where we will be passing out flyers. The assumptions that we are making in this modeling is that each street corner will be able to allow for a certain number of flyers to be passed out according to a certain bounded probability distribution. I think that this assumption is reasonable because the number of flyers that are passed out for a specific place can vary and if they are bounded there is that limitation on the physical maximum and minimum number of flyers that can be passed out. Typically in this scenario we expect the amount of flyers to follow a certain probability distribution like the normal distribution or something else and this might not be very reasonable to hold. We could sample some of the street corners and see if they seem to fit a certain distribution, but it's not guaranteed. I also think that the assumption that each of the probability distributions of the arms or the street corners in this case are independent might not hold and should be tested because I think the different street corners are related in some way if there's a popular overall location some corners may be more popular if they are in that overall popular section. Another thing that I remember being an assumption we made for a particular problem in class was that the distributions had the same variance. After plotting some of the distributions for different stations below, I'm not sure if this holds or if this is necessary for UCB. In terms of regret, I think I would consider the difference

in the average number of flyers that would be collected by the best street corner and the average number of flyers that would be collected by the street corner that we visited.

[2]:
```
#libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

[2]:
```
ny = pd.read_csv('ny.csv')
dc = pd.read_csv('dc.csv')
chicago = pd.read_csv('chicago.csv')
```

[3]:
```
start_dates_ny = ny['starttime'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
stop_dates_ny = ny['stoptime'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
ny['startday'] = start_dates_ny
ny['stopday'] = stop_dates_ny
ny_startday_count = ny[['startday', 'start station name', 'tripduration']].
 ↪rename(columns = {'start station name': 'station', 'startday':'day'}).
 ↪groupby(['day', 'station']).count()
ny_stopday_count = ny[['stopday', 'end station name', 'tripduration']].
 ↪rename(columns = {'end station name': 'station', 'stopday':'day'}).
 ↪groupby(['day', 'station']).count()
ny_station_counts = pd.DataFrame(pd.merge(ny_startday_count, ny_stopday_count,␣
 ↪how = 'outer', left_index = True, right_index = True).fillna(0).sum(axis =␣
 ↪1)).reset_index().rename(columns = {0:'count'})
```

[4]:
```
start_dates_dc = dc['Start date'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
stop_dates_dc = dc['End date'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
dc['startday'] = start_dates_dc
dc['stopday'] = stop_dates_dc
dc_startday_count = dc[['startday', 'Start station', 'Duration (ms)']].
 ↪rename(columns = {'Start station': 'station', 'startday': 'day'}).
 ↪groupby(['day', 'station']).count()
dc_stopday_count = dc[['stopday', 'End station', 'Duration (ms)']].
 ↪rename(columns = {'End station': 'station', 'stopday': 'day'}).
 ↪groupby(['day', 'station']).count()
dc_station_counts = pd.DataFrame(pd.merge(dc_startday_count, dc_stopday_count,␣
 ↪how = 'outer', left_index = True, right_index = True).fillna(0).sum(axis =␣
 ↪1).reset_index().rename(columns = {0:'count'}))
```

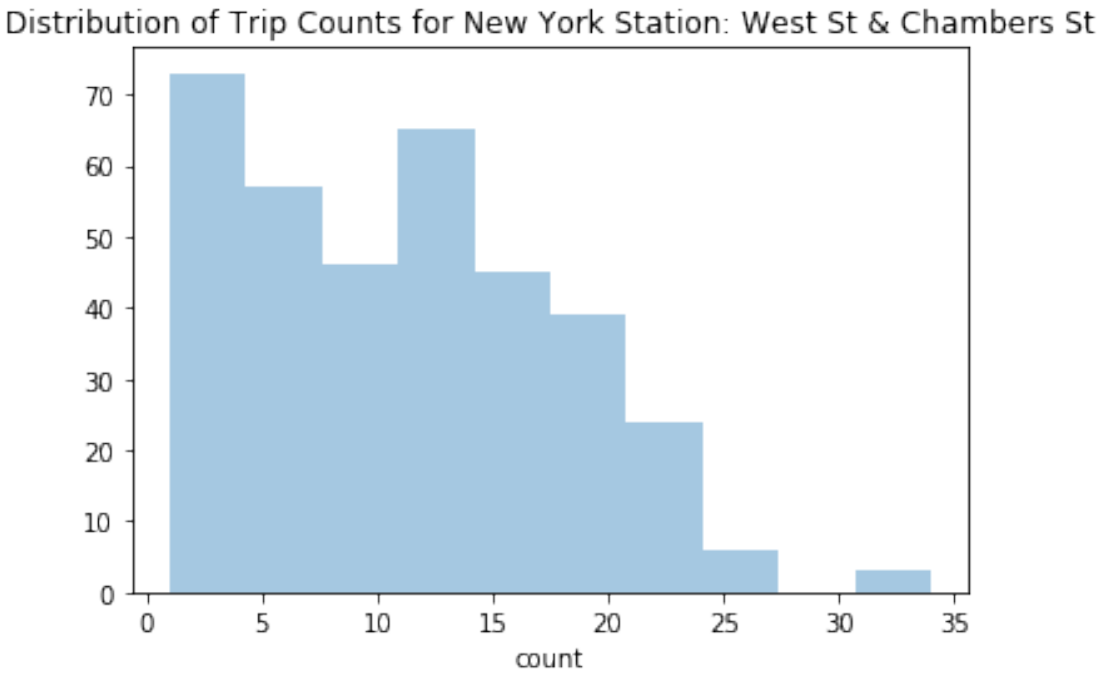
[5]:
```
start_dates_ch = chicago['starttime'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
```

```
stop_dates_ch = chicago['stoptime'].apply(lambda x: re.findall(r'[0-9]{1,2}\/
 ↪[0-9]{1,2}\/[0-9]{4}', x)[0])
chicago['startday'] = start_dates_ch
chicago['stopday'] = stop_dates_ch
chicago_startday_count = chicago[['startday', 'from_station_name',␣
 ↪'tripduration']].rename(columns = {'from_station_name':'station', 'startday':
 ↪'day'}).groupby(['day', 'station']).count()
chicago_stopday_count = chicago[['stopday', 'to_station_name', 'tripduration']].
 ↪rename(columns = {'to_station_name':'station', 'stopday':'day'}).
 ↪groupby(['day', 'station']).count()
chicago_station_counts = pd.DataFrame(pd.merge(chicago_startday_count,␣
 ↪chicago_stopday_count, how = 'outer', left_index = True, right_index = True).
 ↪fillna(0).sum(axis = 1).reset_index().rename(columns = {0:'count'}))
```

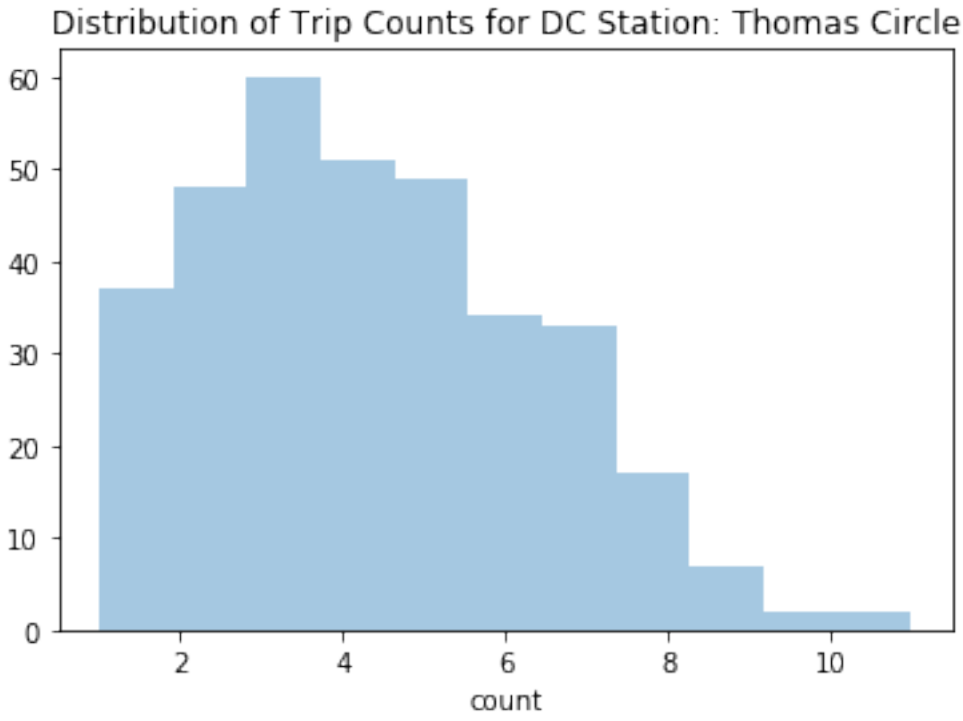
[129]:
```
top_10_ny = list(ny_station_counts[['station', 'count']].groupby('station').
 ↪sum().reset_index().sort_values('count', ascending = False).
 ↪iloc[range(10)]['station'])
top_10_dc = list(dc_station_counts[['station', 'count']].groupby('station').
 ↪sum().reset_index().sort_values('count', ascending = False).
 ↪iloc[range(10)]['station'])
top_10_ch = list(chicago_station_counts[['station', 'count']].
 ↪groupby('station').sum().reset_index().sort_values('count', ascending =␣
 ↪False).iloc[range(10)]['station'])
```

[7]:
```
ny_station_counts_top = ny_station_counts[ny_station_counts['station'].
 ↪apply(lambda s: s in top_10_ny)]
dc_station_counts_top = dc_station_counts[dc_station_counts['station'].
 ↪apply(lambda s: s in top_10_dc)]
ch_station_counts_top =␣
 ↪chicago_station_counts[chicago_station_counts['station'].apply(lambda s: s␣
 ↪in top_10_ch)]
```

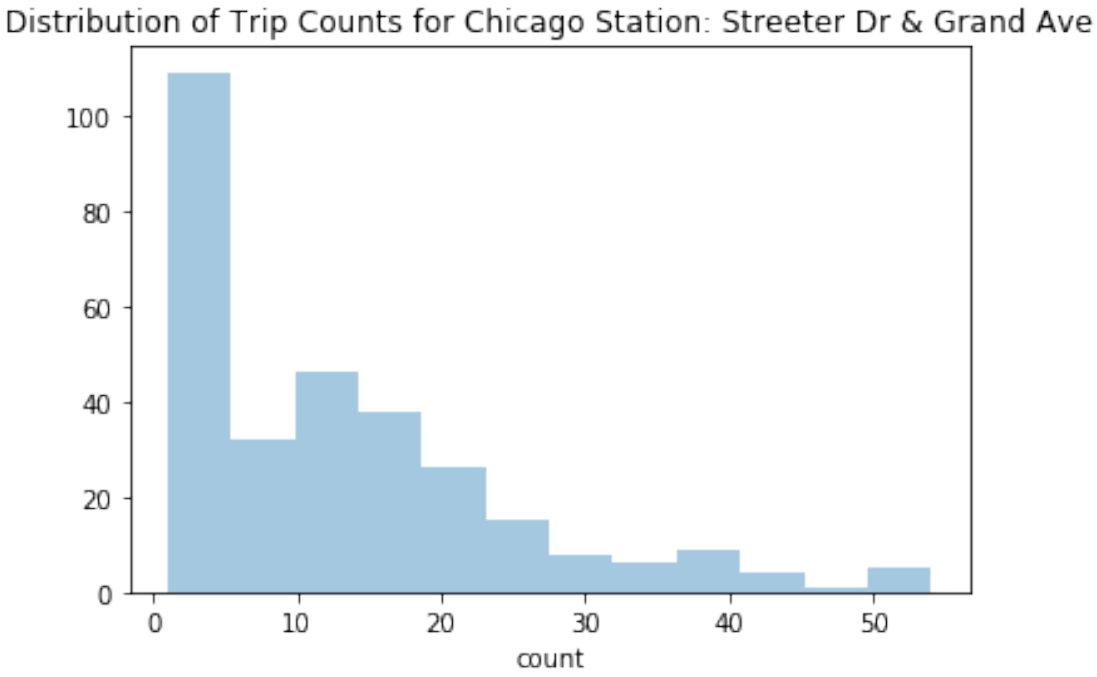
[81]:
```
sns.distplot(ny_station_counts_top[ny_station_counts_top['station'] ==␣
 ↪top_10_ny[3]]['count'], kde = False)
plt.title('Distribution of Trip Counts for New York Station: ' + top_10_ny[3]);
```



[82]:
```
sns.distplot(dc_station_counts_top[dc_station_counts_top['station'] ==␣
 ↪top_10_dc[7]]['count'], kde = False)
plt.title('Distribution of Trip Counts for DC Station: ' + top_10_dc[7]);
```



[83]:
```
sns.distplot(ch_station_counts_top[ch_station_counts_top['station'] ==␣
 ↪top_10_ch[0]]['count'], kde = False)
plt.title('Distribution of Trip Counts for Chicago Station: ' + top_10_ch[0]);
```



2. Simulate UCB strategy using past data

[56]:
```
def upper_confidence_bound(time, stations, flyers):
    confidence_bounds = []
    means = []
    for st in stations:
        rewards = flyers[st]
        if len(rewards) == 0:
            confidence_bounds.append(np.inf)
            means.append(np.inf)
        else:
            current_sample_mean = np.mean(rewards)
            means.append(current_sample_mean)
            add_int = np.sqrt((3 * np.log(time)) / (2 * len(rewards)))
            confidence_bounds.append(current_sample_mean + add_int)
    pull = stations[np.argmax(confidence_bounds)]
    return pull, confidence_bounds, means
```

[57]:
```
def pull_arm(pull, flyers, table):
    pulled = np.random.choice(table[table['station'] == pull]['count'])
    flyers[pull] = flyers[pull] + [pulled]
    return flyers
```

1.2.1 Implementation and Results

Assuming boundedness of the rewards. The way that we are starting the exploration part of the process of determining the best street corner to pass flyers out to using the number of people either starting or ending their bike ride at a certain station as a proxy for determining which station would be best to pass flyers out to. Since I'm not assuming subgaussian rewards, the way that we are 'pulling arms' is by randomly sampling the number of people that either start or end their trip at the station from our data. I did this by sorting our counts table to only include the station or "arm" that we are pulling and then randomly sampling with np.random.choice from the counts from the station. At each step we calculate means for each "arm" or station by finding the average of past rewards up to that point and the upper confidence bound for that station by using the sample mean added with the $\sqrt{\frac{3log(t)}{2T_i(t-1)}}$ where t is the time step we are computing confidence bounds for and $T_i(t)$ is the number of times that the "arm" i has been pulled up to time t. If the arm has never been pulled then the confidence bound for that station is represented by infinity and when multiple arms have been pulled the first station in our list of stations will be the "arm" that is pulled. In order to implement this process I created two functions one that returns the "arm" to pull, the upper confidence bounds for each station using the process that I explained above, and the sample mean of the pulled "arms". I implemented this with a time integer variable, list of station names, and a dictionary with station names as indexes and lists of the rewards we have received from that station with unvisited stations with empty lists. The other function "pulls" an arm by randomly sampling from the counts that we have for the station from the data. This function would need to know the arm being pulled, the dictionary containing the station as keys and the previous pulls, and the table that contains all of the data that we have for each station and returns the reward observed and then an updated dictionary of the rewards at each station. The formula for finding the upper confidence width was chosen in a way that the width depends on what time step we are on, but also the amount of times that we have pulled and therefore "explored" the given "arm".

[58]:
```
top_10_ny_means = np.array([])
for st in top_10_ny:
    top_10_ny_means = np.append(top_10_ny_means, np.
 ↪mean(ny_station_counts_top[ny_station_counts_top['station'] == st]['count']))
top_ny_station = top_10_ny[np.argmax(top_10_ny_means)]
```

[59]:
```
top_10_dc_means = np.array([])
for st in top_10_dc:
    top_10_dc_means = np.append(top_10_dc_means, np.
 ↪mean(dc_station_counts_top[dc_station_counts_top['station'] == st]['count']))
top_dc_station = top_10_dc[np.argmax(top_10_dc_means)]
```

[12]:
```
top_10_ch_means = np.array([])
for st in top_10_ch:
    top_10_ch_means = np.append(top_10_ch_means, np.
 ↪mean(ch_station_counts_top[ch_station_counts_top['station'] == st]['count']))
```

```
top_ch_station = top_10_ch[np.argmax(top_10_ch_means)]
```

[13]:
```
ch_flyers = {}
for st in top_10_ch:
    ch_flyers[st] = []
```

[14]:
```
top_10_ny_array = np.array(top_10_ny)
top_10_dc_array = np.array(top_10_dc)
top_10_ch_array = np.array(top_10_ch)
```

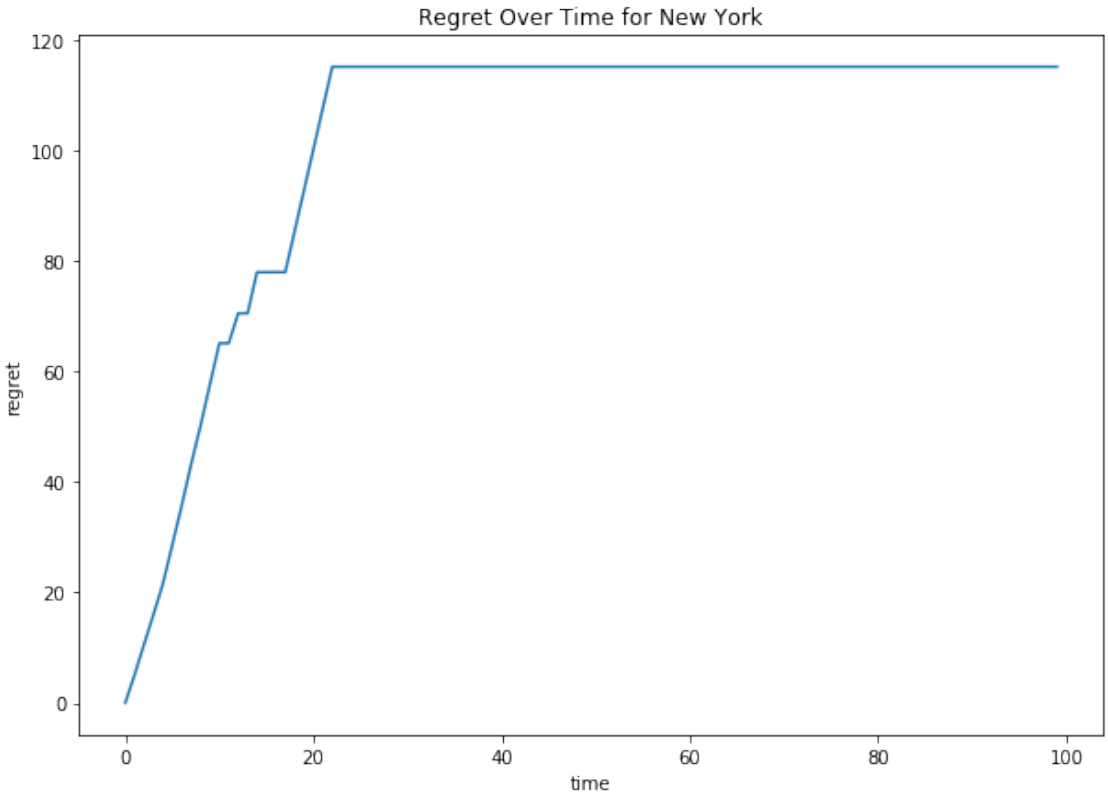
[15]:
```
station_means_strings_ch = [str(m) for m in top_10_ch_means]
```

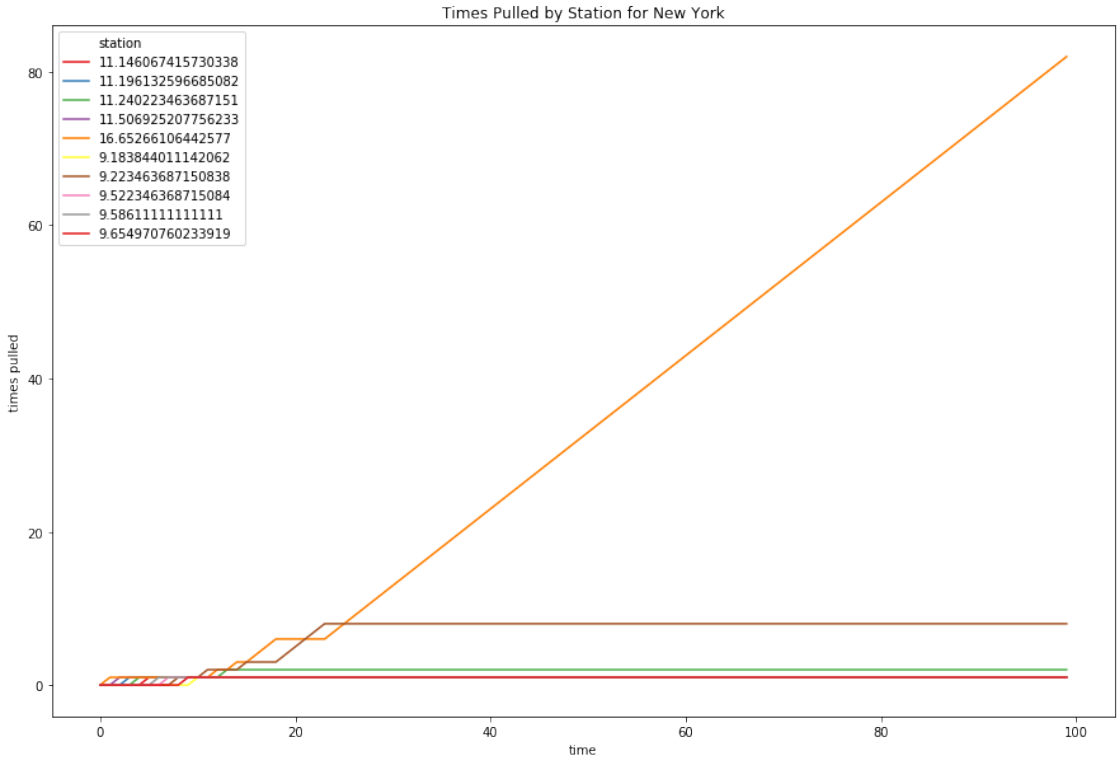
[16]:
```
ch_regret = []
ch_number_pulls = []
ch_estimated_means = []
ch_estimated_bounds = []
```

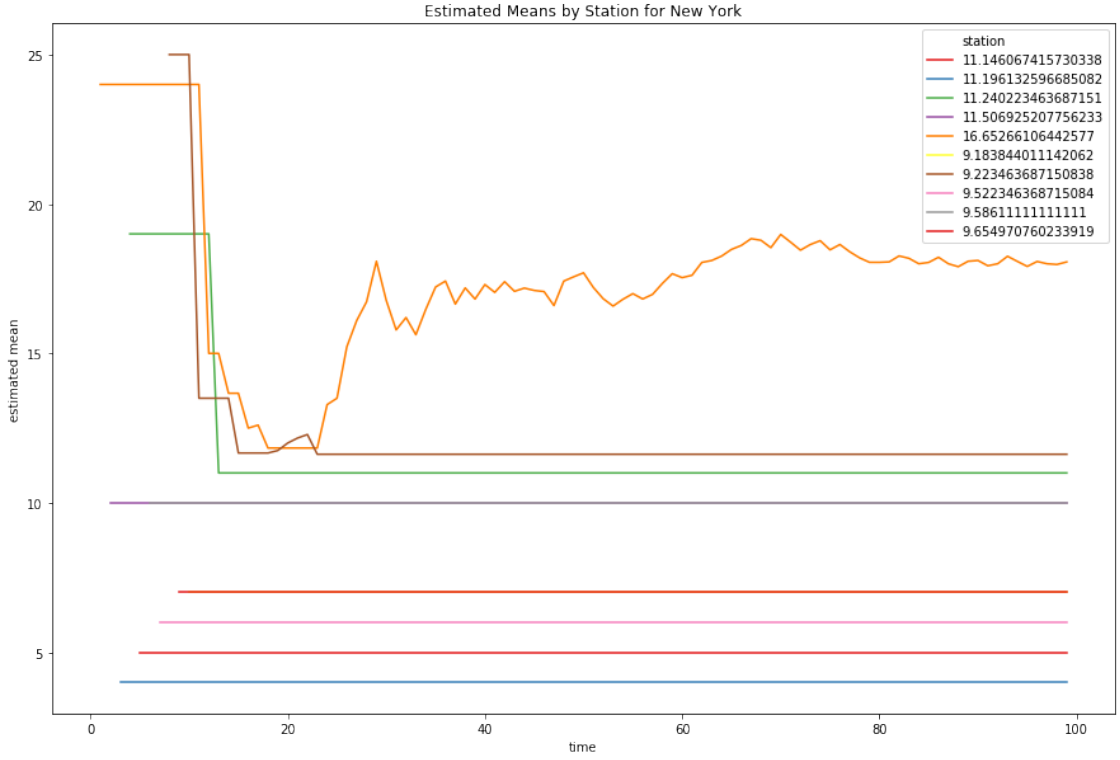
[77]:
```
def plot_graphs(station_list, counts_table, steps, city):
    flyers_dict = {}
    for st in station_list:
        flyers_dict[st] = []
    means_array = np.array([])
    for st in station_list:
        means_array = np.append(means_array, (np.
 ↪mean(counts_table[counts_table['station'] == st]['count'])))
        means_strings_array = [str(m) for m in means_array]
    regret = []
    number_pulls = []
    estimated_means = []
    estimated_bounds = []
    stations_array = np.array(station_list)
    for t in np.arange(steps):
        pull, bounds, means = upper_confidence_bound(t, station_list,␣
 ↪flyers_dict)
        for i in np.arange(10):
            number_pulls.append([t, means_strings_array[i],␣
 ↪len(flyers_dict[station_list[i]])])
            estimated_bounds.append([t, means_strings_array[i], bounds[i]])
            estimated_means.append([t, means_strings_array[i], means[i]])
        flyers_dict = pull_arm(pull, flyers_dict, counts_table)
        regret.append((np.max(means_array) - means_array[stations_array ==␣
 ↪pull])[0])
    fig, ax = plt.subplots(figsize = (10, 7))
    sns.lineplot(np.arange(steps), np.cumsum(regret), ax = ax)
    plt.title('Regret Over Time for ' + city)
    plt.xlabel('time')
    plt.ylabel('regret')
    pulls_df = pd.DataFrame(number_pulls, columns = ['time', 'station', 'times␣
 ↪pulled'])
```

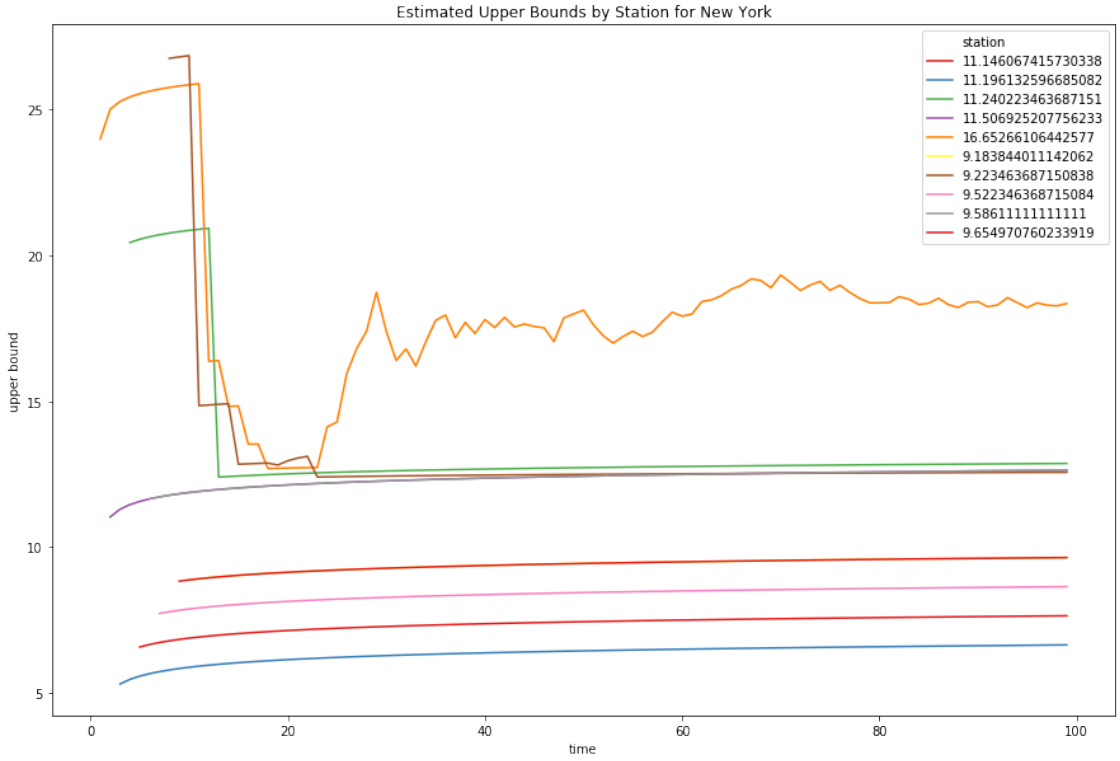
```
    fig, ax = plt.subplots(figsize = (15, 10))
    sns.lineplot(x = 'time', y = 'times pulled', hue = 'station', palette=sns.
↪color_palette("Set1", pulls_df.station.nunique()), data = pulls_df)
    plt.title('Times Pulled by Station for ' + city)
    means_df = pd.DataFrame(estimated_means, columns = ['time', 'station',␣
↪'estimated mean'])
    fig, ax = plt.subplots(figsize = (15, 10))
    sns.lineplot(x = 'time', y = 'estimated mean', hue = 'station', palette=sns.
↪color_palette("Set1", pulls_df.station.nunique()), data = means_df)
    plt.title('Estimated Means by Station for ' + city)
    bounds_df = pd.DataFrame(estimated_bounds, columns = ['time', 'station',␣
↪'upper bound'])
    fig, ax = plt.subplots(figsize = (15, 10))
    sns.lineplot(x = 'time', y = 'upper bound', hue = 'station', palette=sns.
↪color_palette("Set1", pulls_df.station.nunique()), data = bounds_df)
    plt.title('Estimated Upper Bounds by Station for ' + city)
```

[125]:
```
plot_graphs(top_10_ny, ny_station_counts_top, 100, 'New York')
```



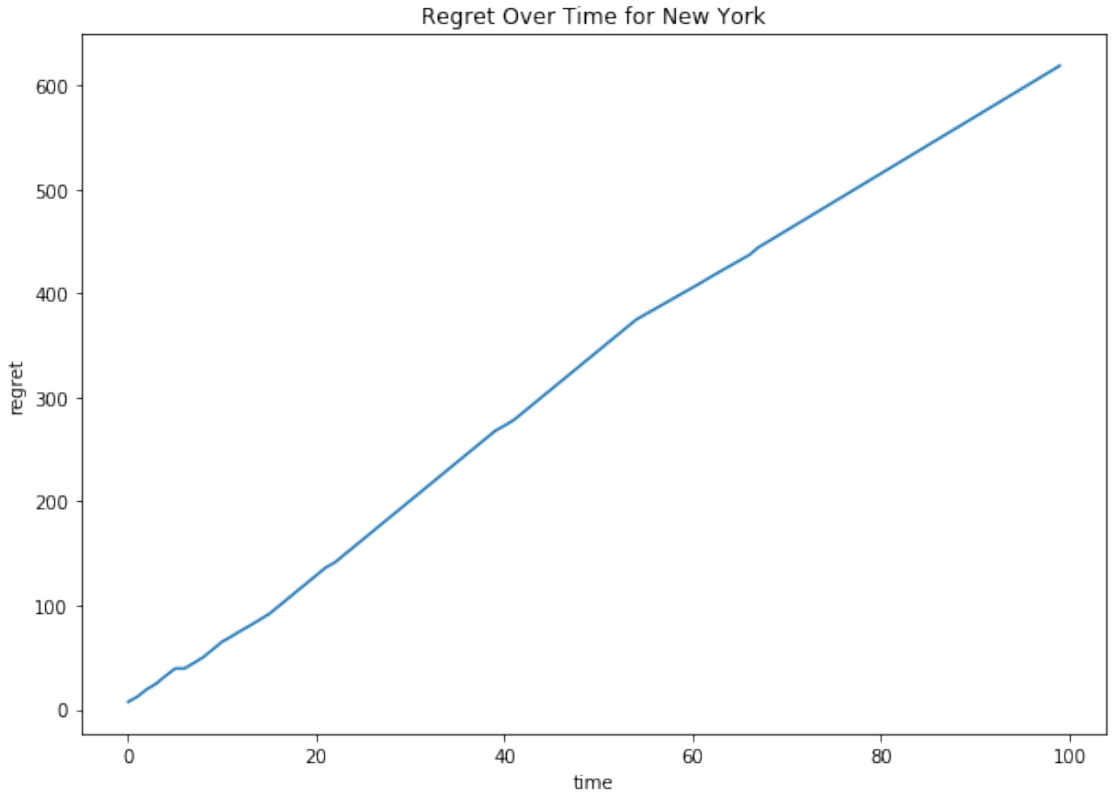

Times Pulled by Station for New York
station
11.146067415730338
11.196132596685082
11.240223463687151
11.506925207756233
16.65266106442577
9.183844011142062
9.223463687150838
9.522346368715084
9.58611111111111
9.654970760233919
times pulled
time
0
20
40
60
80
100

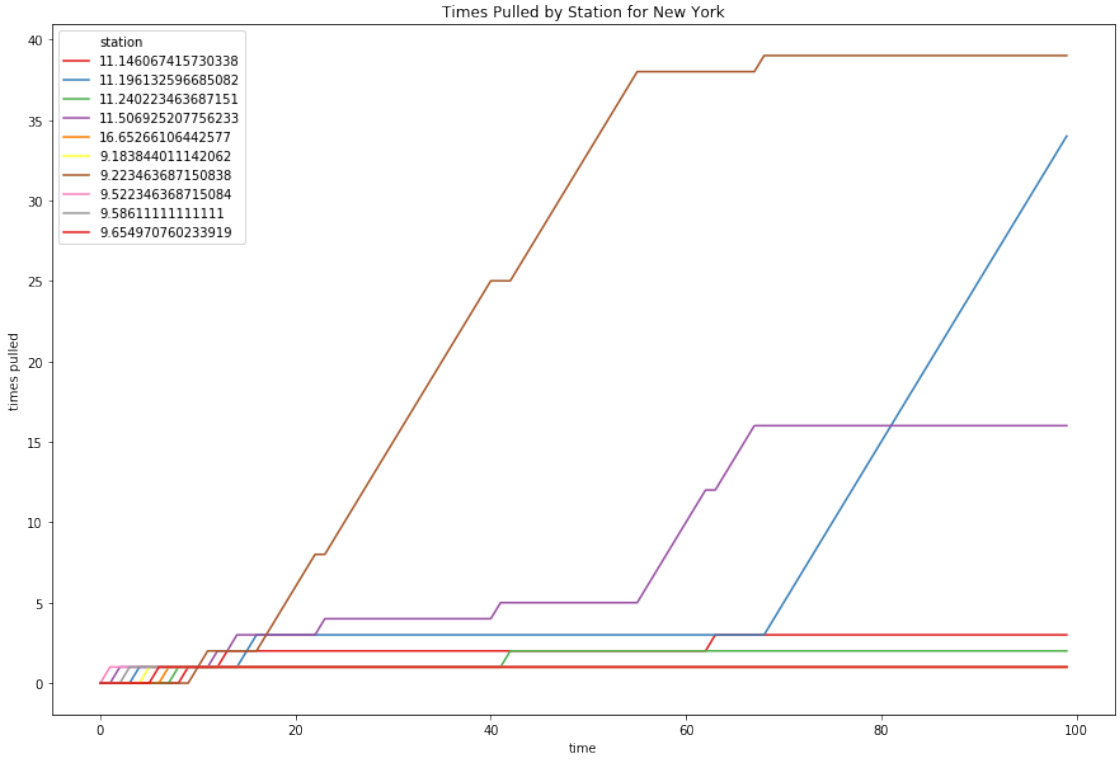

Estimated Means by Station for New York
station
11.146067415730338
11.196132596685082
11.240223463687151
11.506925207756233
16.65266106442577
9.183844011142062
9.223463687150838
9.522346368715084
9.58611111111111
9.654970760233919
estimated mean
time
5
10
15
20
25
0
20
40
60
80
100

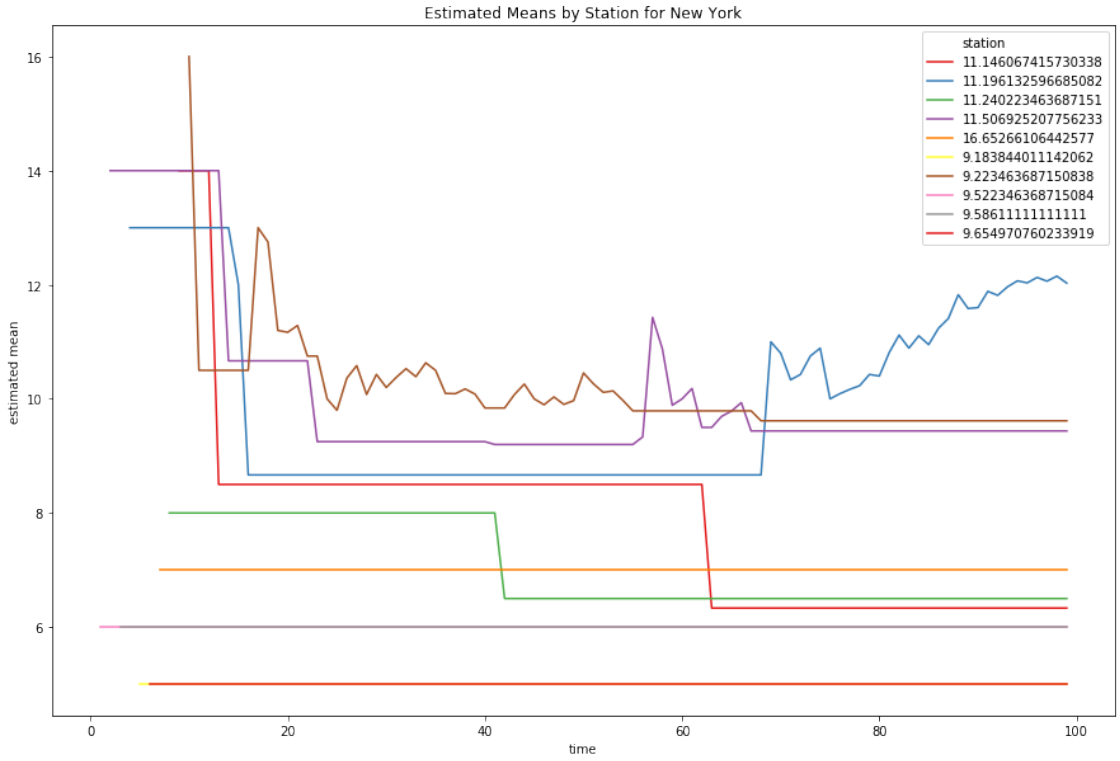


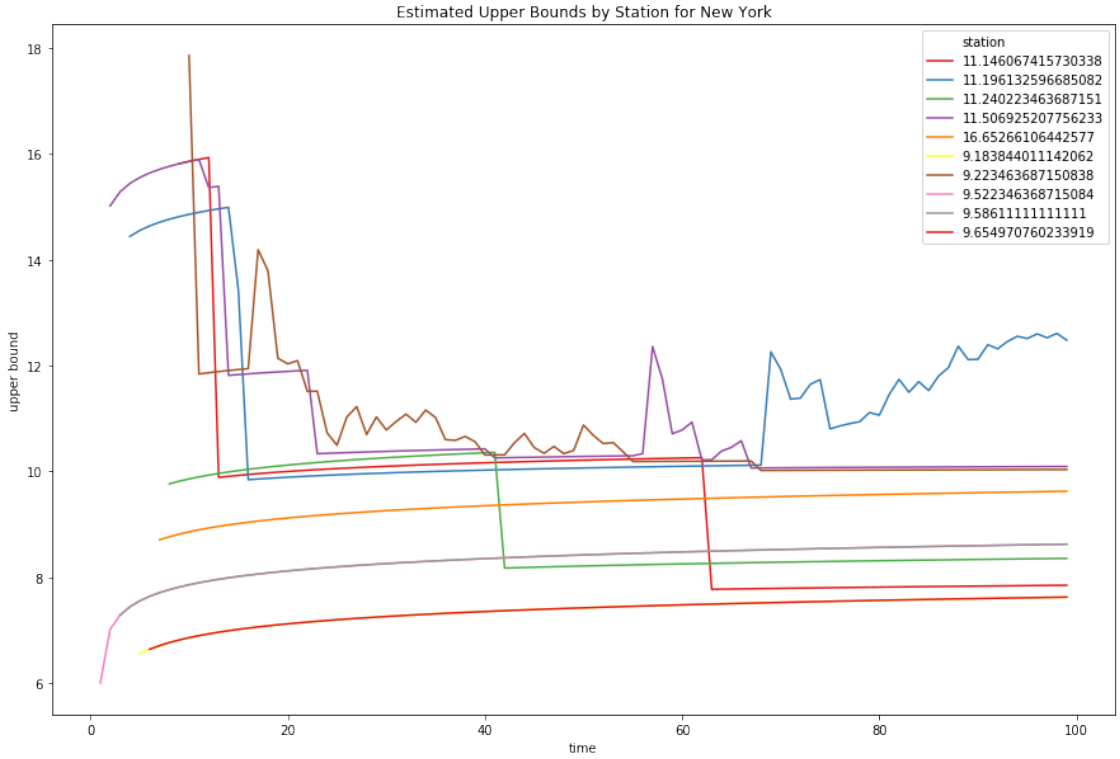
[126]:
```
np.random.shuffle(top_10_ny)
```

[127]:
```
plot_graphs(top_10_ny, ny_station_counts_top, 100, 'New York')
```

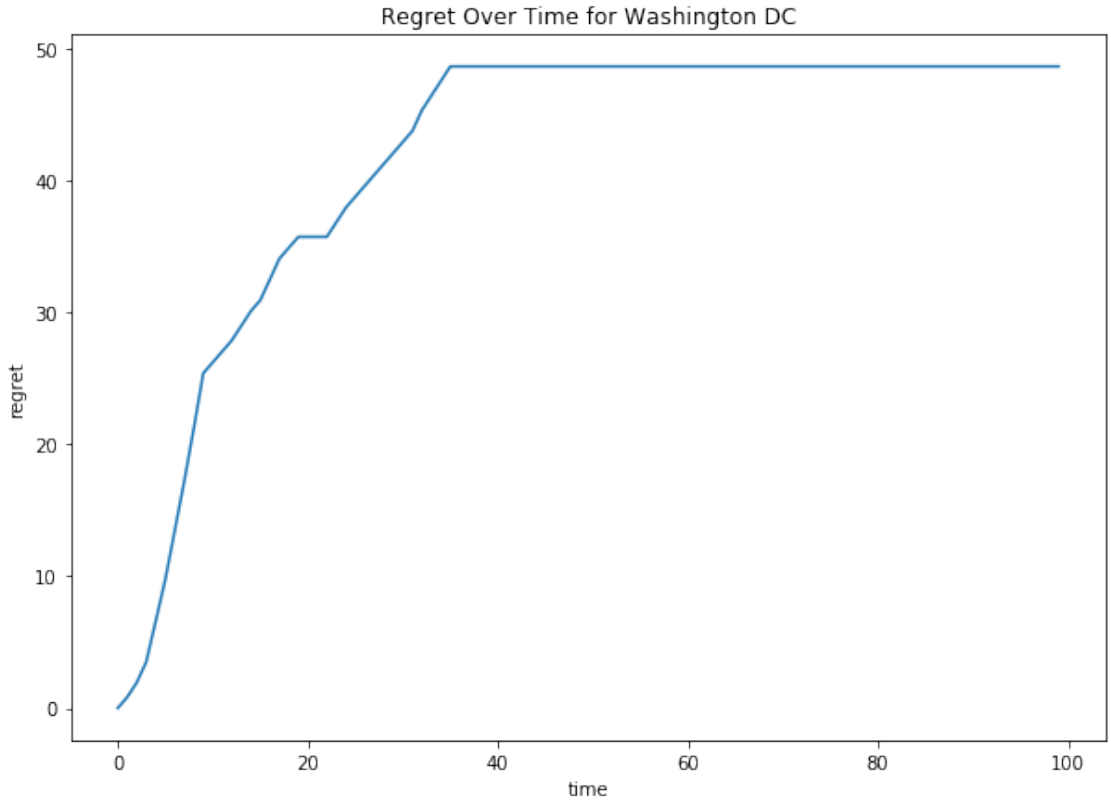

Regret Over Time for New York
regret
time
0
100
200
300
400
500
600
0
20
40
60
80
100

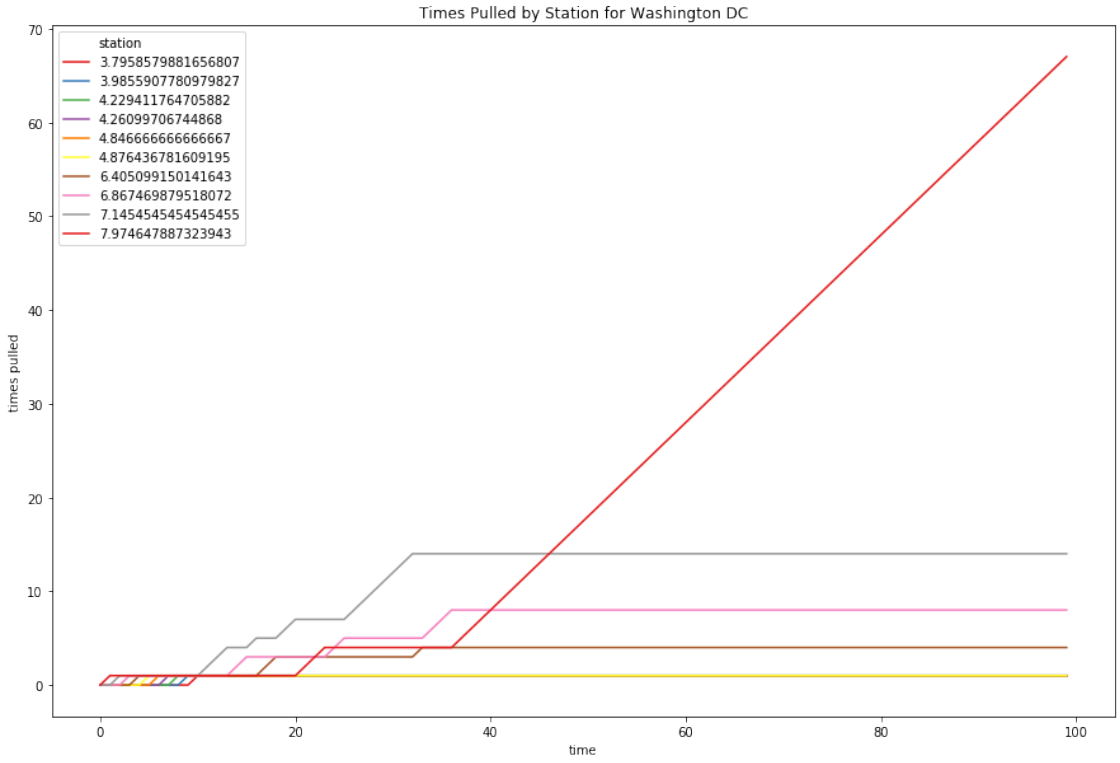

Times Pulled by Station for New York
station
11.146067415730338
11.196132596685082
11.240223463687151
11.506925207756233
16.65266106442577
9.183844011142062
9.223463687150838
9.522346368715084
9.58611111111111
9.654970760233919
times pulled
time
0
5
10
15
20
25
30
35
40
0
20
40
60
80
100

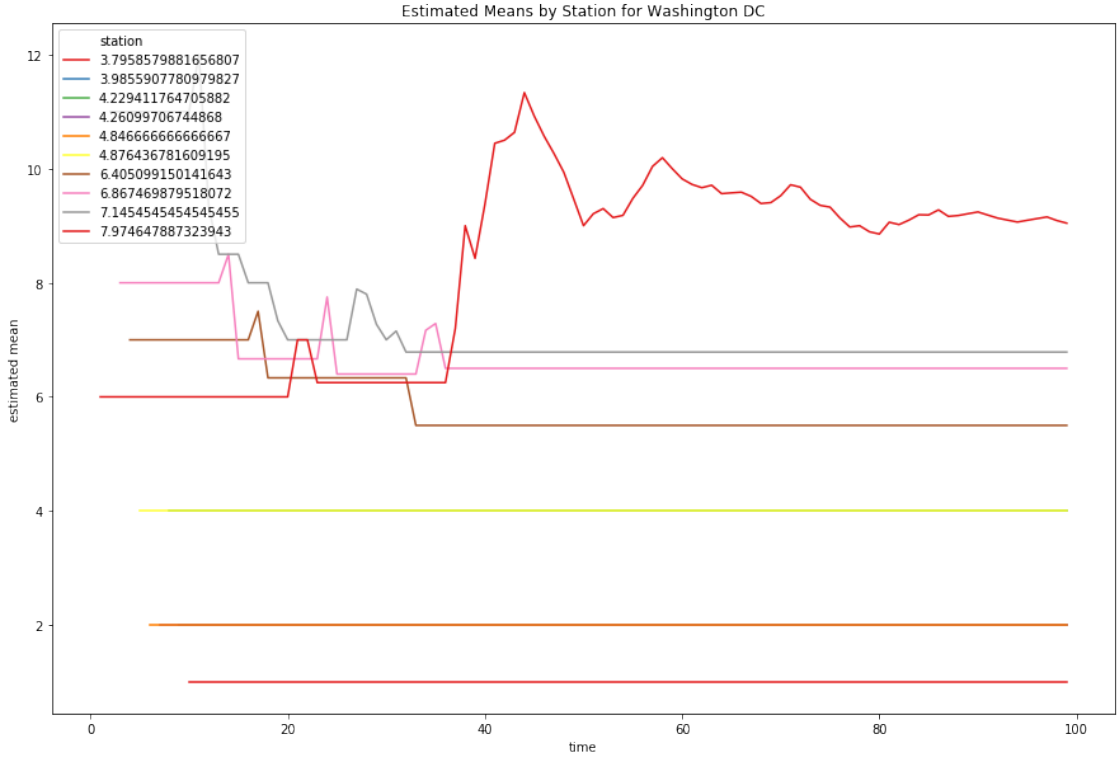

Estimated Means by Station for New York
station
11.146067415730338
11.196132596685082
11.240223463687151
11.506925207756233
16.65266106442577
9.183844011142062
9.223463687150838
9.522346368715084
9.58611111111111
9.654970760233919
estimated mean
time
0
20
40
60
80
100
6
8
10
12
14
16

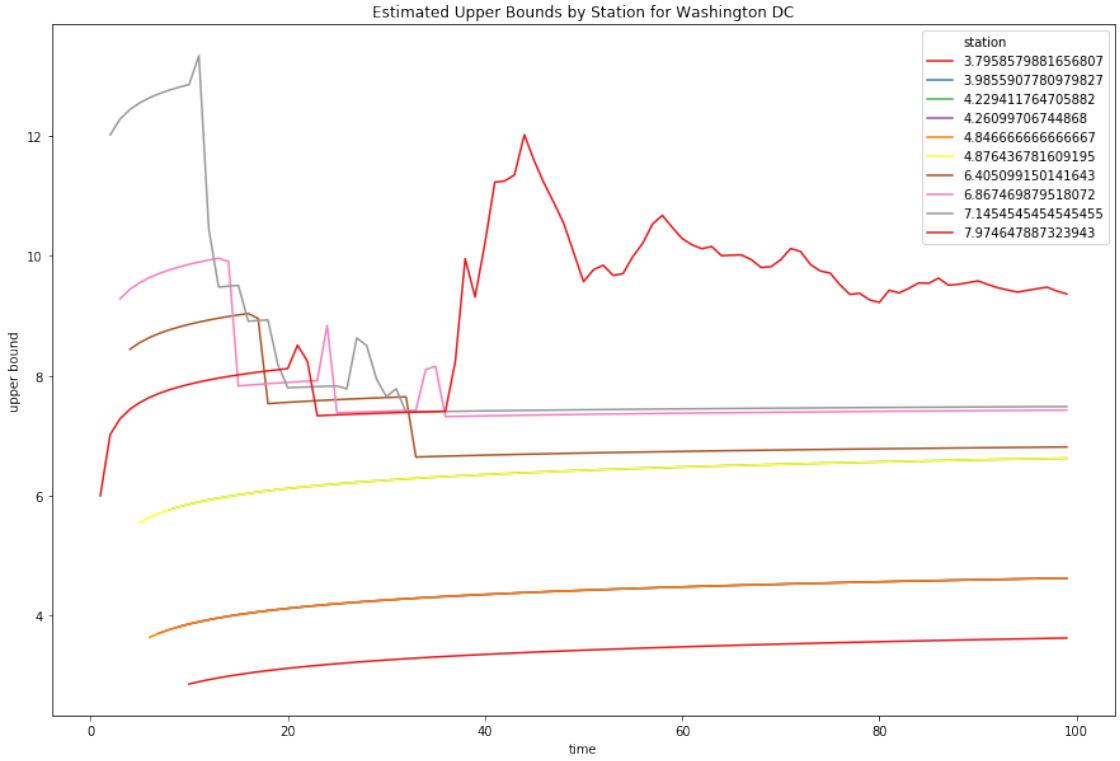

Estimated Upper Bounds by Station for New York
station
11.146067415730338
11.196132596685082
11.240223463687151
11.506925207756233
16.65266106442577
9.183844011142062
9.223463687150838
9.522346368715084
9.58611111111111
9.654970760233919
upper bound
time
0
20
40
60
80
100
6
8
10
12
14
16
18

[142]:
```
plot_graphs(top_10_dc, dc_station_counts_top, 100, 'Washington DC')
```



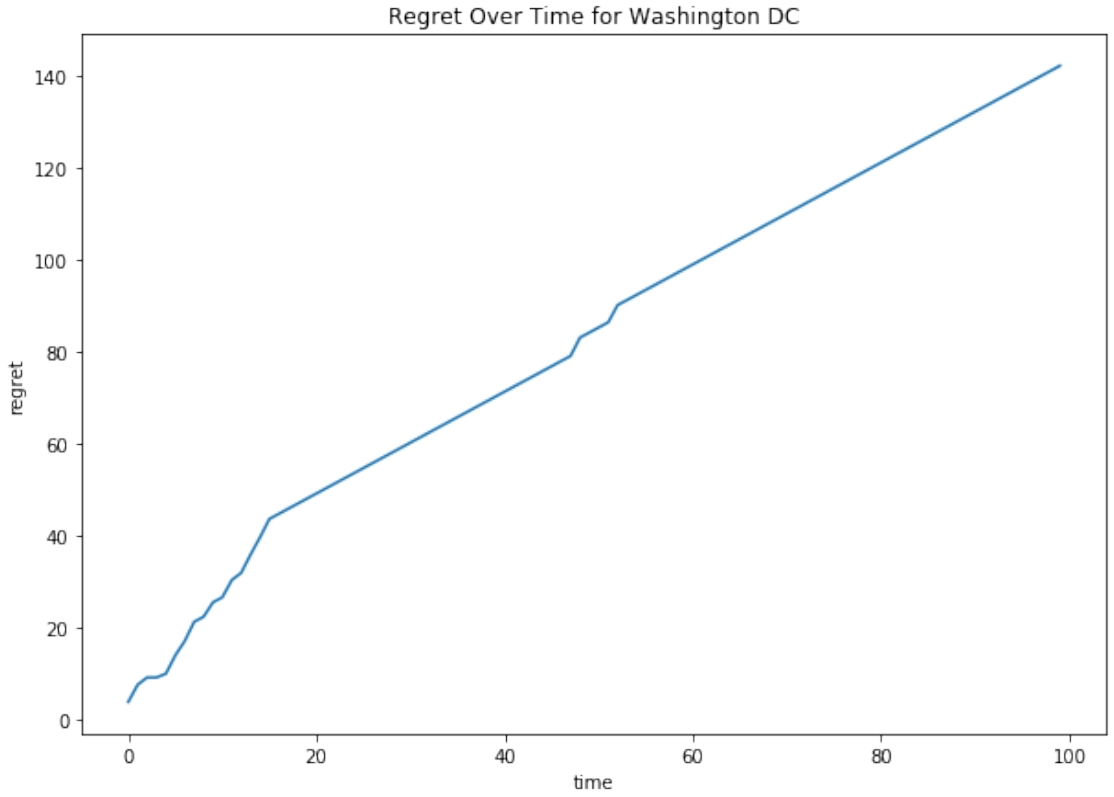

Times Pulled by Station for Washington DC
station
3.7958579881656807
3.9855907780979827
4.229411764705882
4.26099706744868
4.846666666666667
4.876436781609195
6.405099150141643
6.867469879518072
7.1454545454545455
7.974647887323943
times pulled
time
0
10
20
30
40
50
60
70
0
20
40
60
80
100

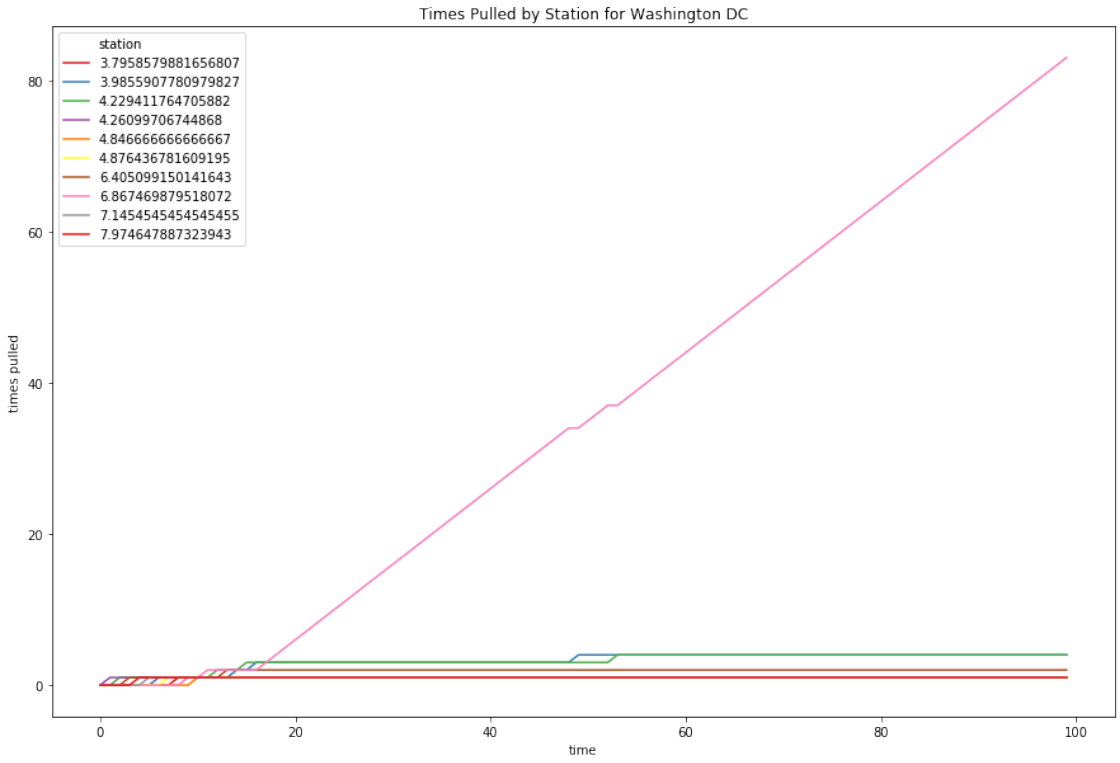

Estimated Means by Station for Washington DC
station
3.7958579881656807
3.9855907780979827
4.229411764705882
4.26099706744868
4.846666666666667
4.876436781609195
6.405099150141643
6.867469879518072
7.1454545454545455
7.974647887323943
estimated mean
time
2
4
6
8
10
12
0
20
40
60
80
100

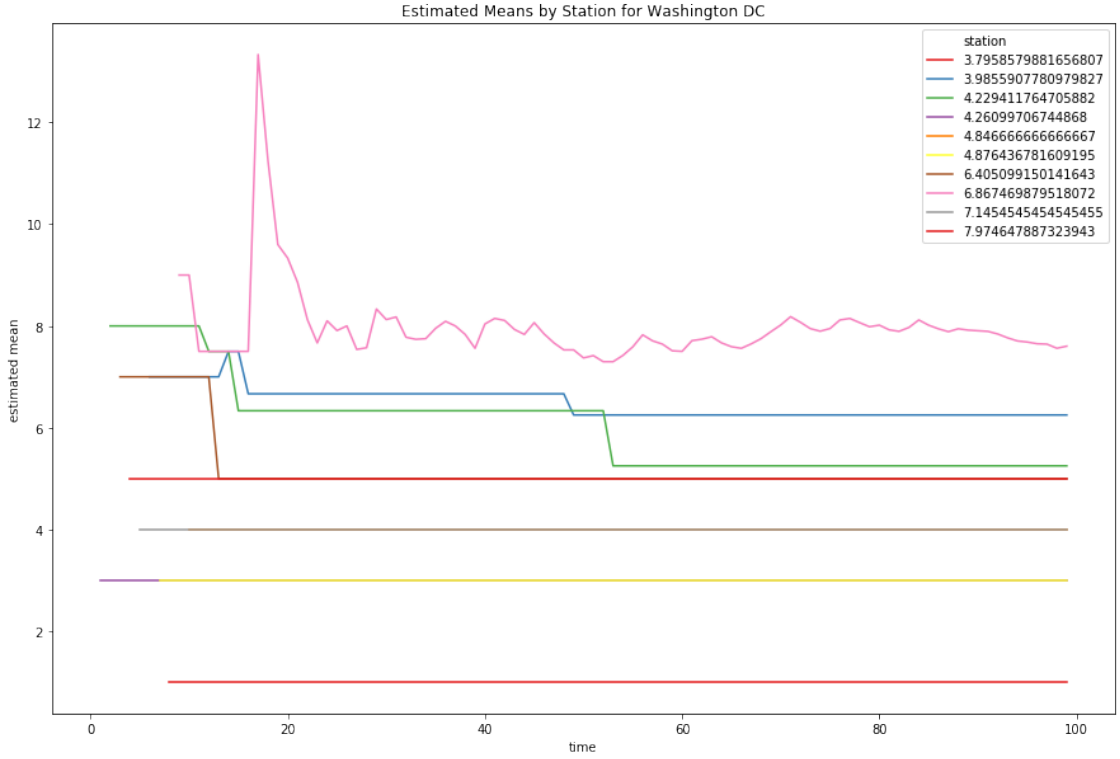


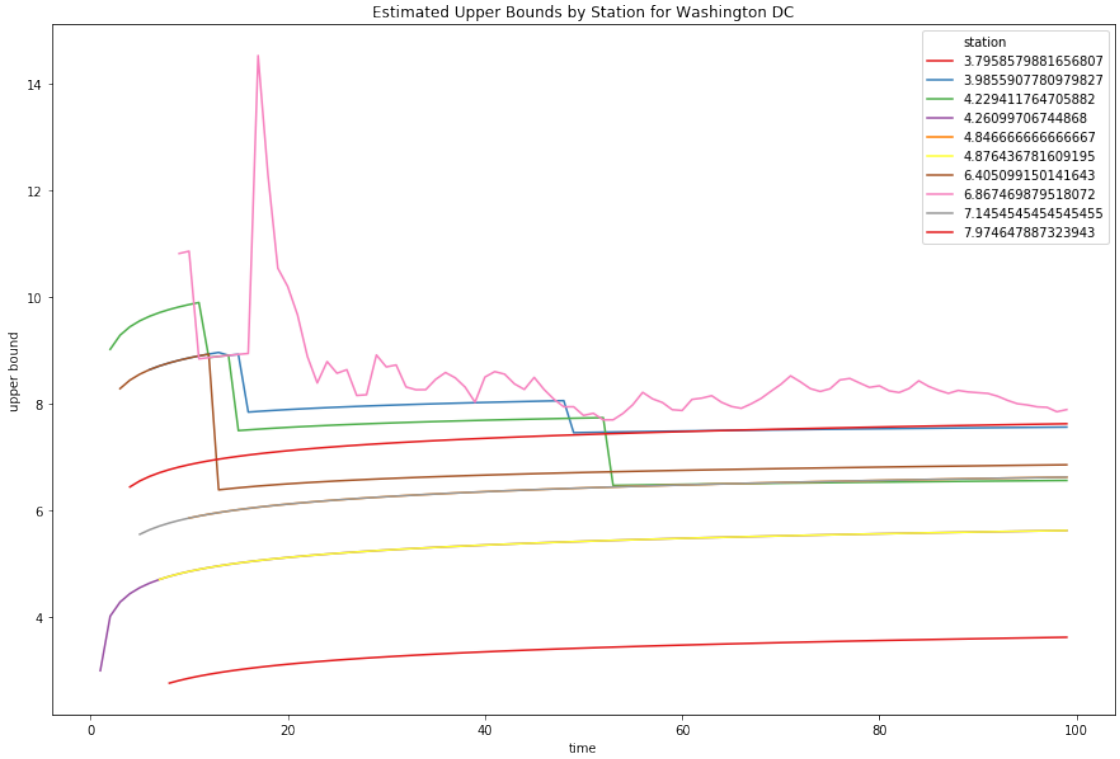
[143]:
```
np.random.shuffle(top_10_dc)
```

[144]:
```
plot_graphs(top_10_dc, dc_station_counts_top, 100, 'Washington DC')
```

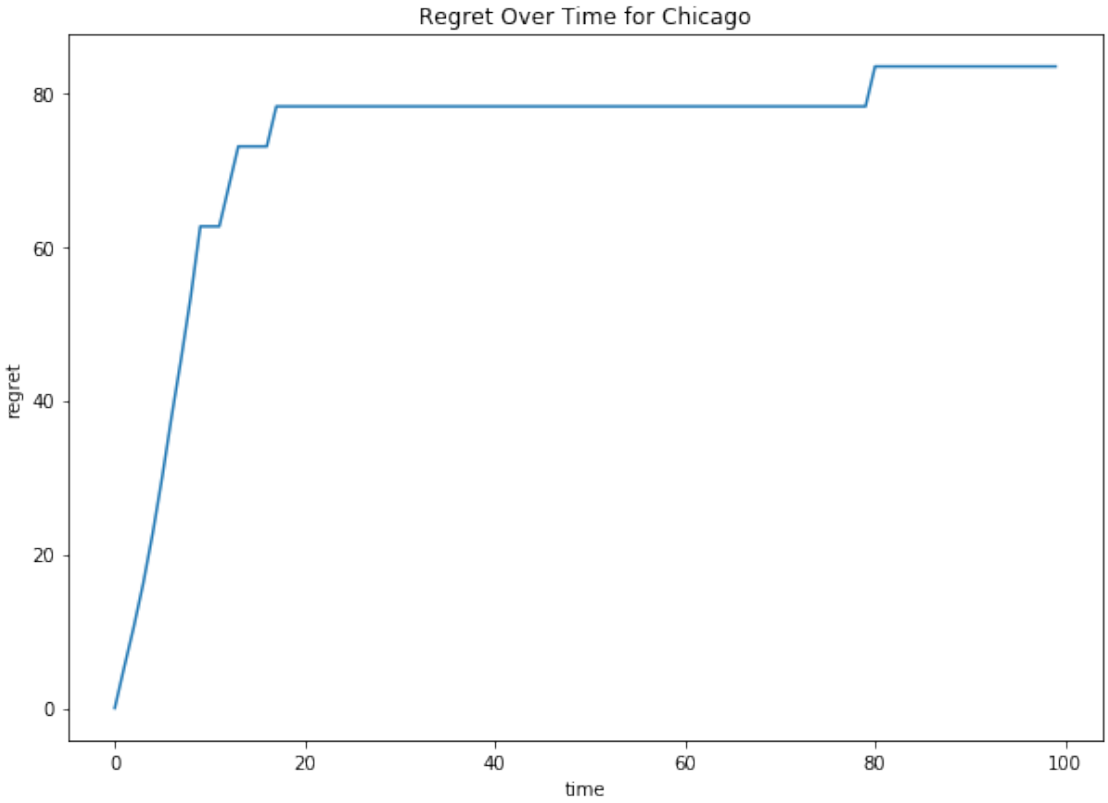

Regret Over Time for Washington DC
regret
time
0
20
40
60
80
100
120
140

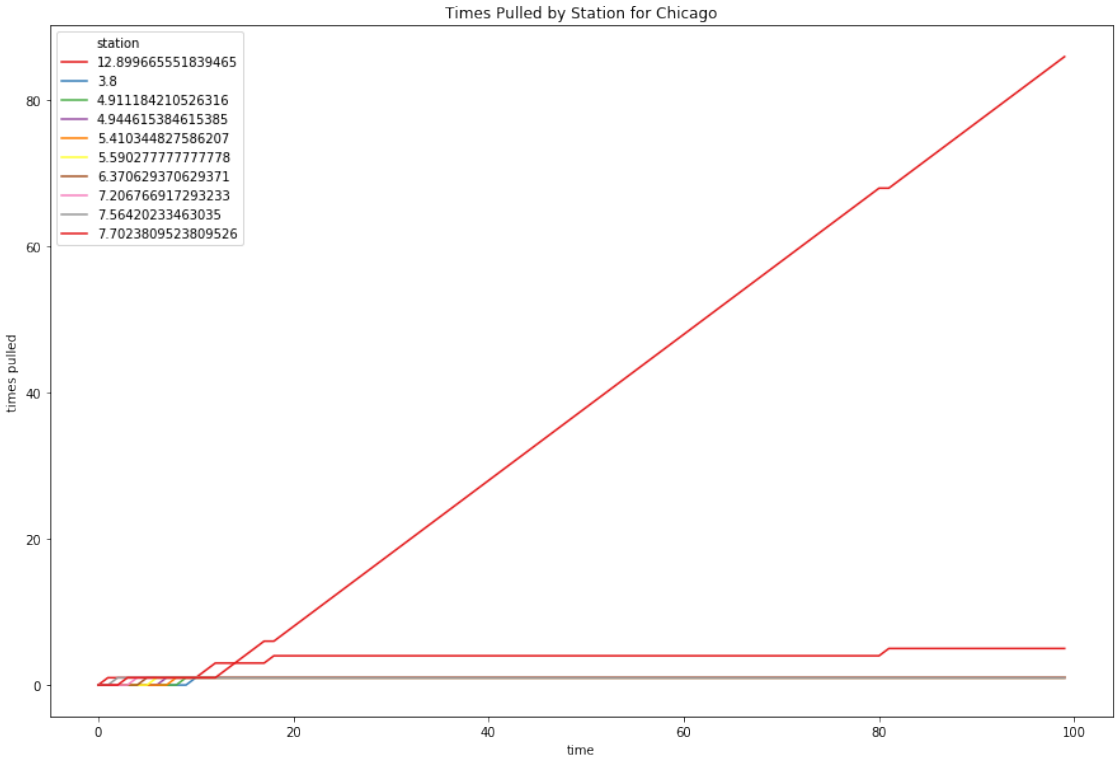

Times Pulled by Station for Washington DC
station
3.7958579881656807
3.9855907780979827
4.229411764705882
4.26099706744868
4.846666666666667
4.876436781609195
6.405099150141643
6.867469879518072
7.1454545454545455
7.974647887323943
times pulled
time
0
20
40
60
80
100

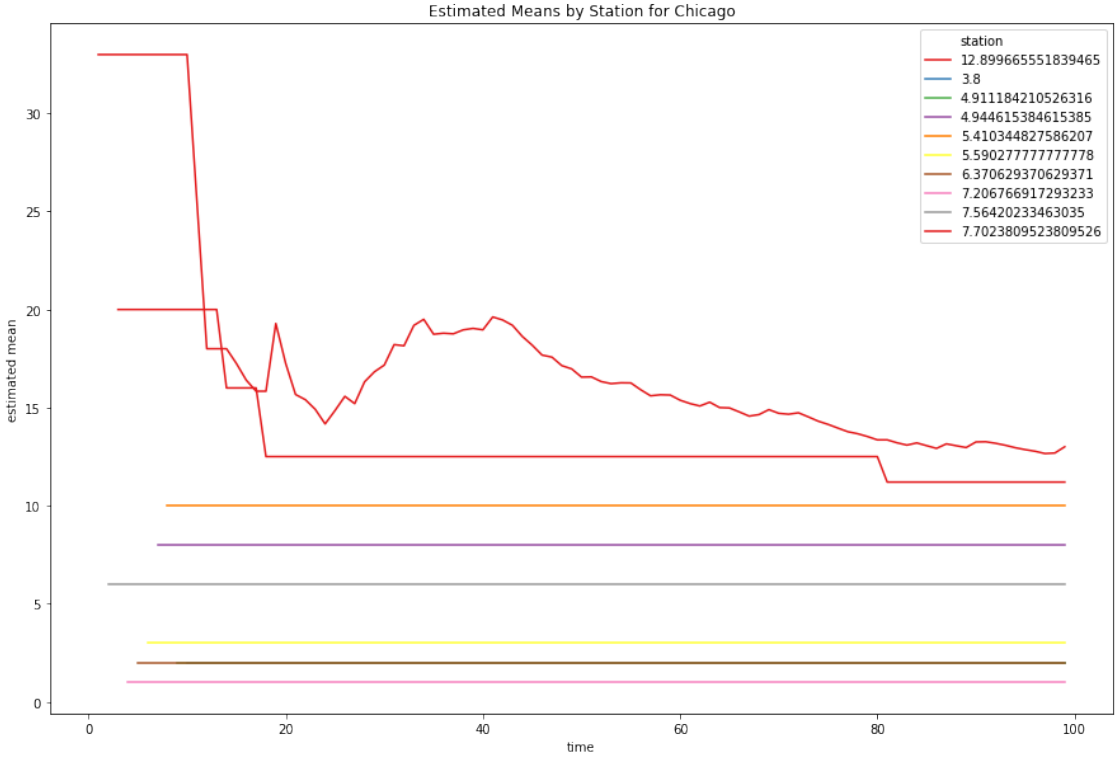

Estimated Means by Station for Washington DC
station
3.7958579881656807
3.9855907780979827
4.229411764705882
4.26099706744868
4.846666666666667
4.876436781609195
6.405099150141643
6.867469879518072
7.1454545454545455
7.974647887323943
estimated mean
time

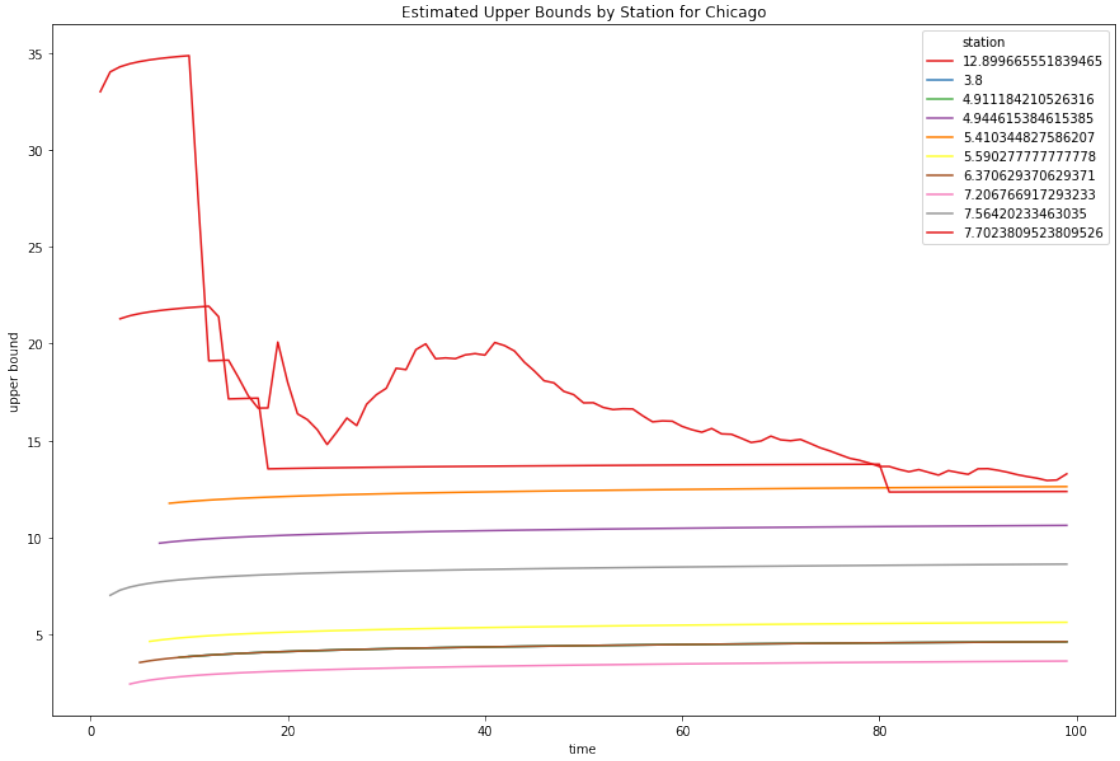

Estimated Upper Bounds by Station for Washington DC
station
3.7958579881656807
3.9855907780979827
4.229411764705882
4.26099706744868
4.846666666666667
4.876436781609195
6.405099150141643
6.867469879518072
7.1454545454545455
7.974647887323943
upper bound
time

[134]:
```
plot_graphs(top_10_ch, ch_station_counts_top, 100, 'Chicago')
```



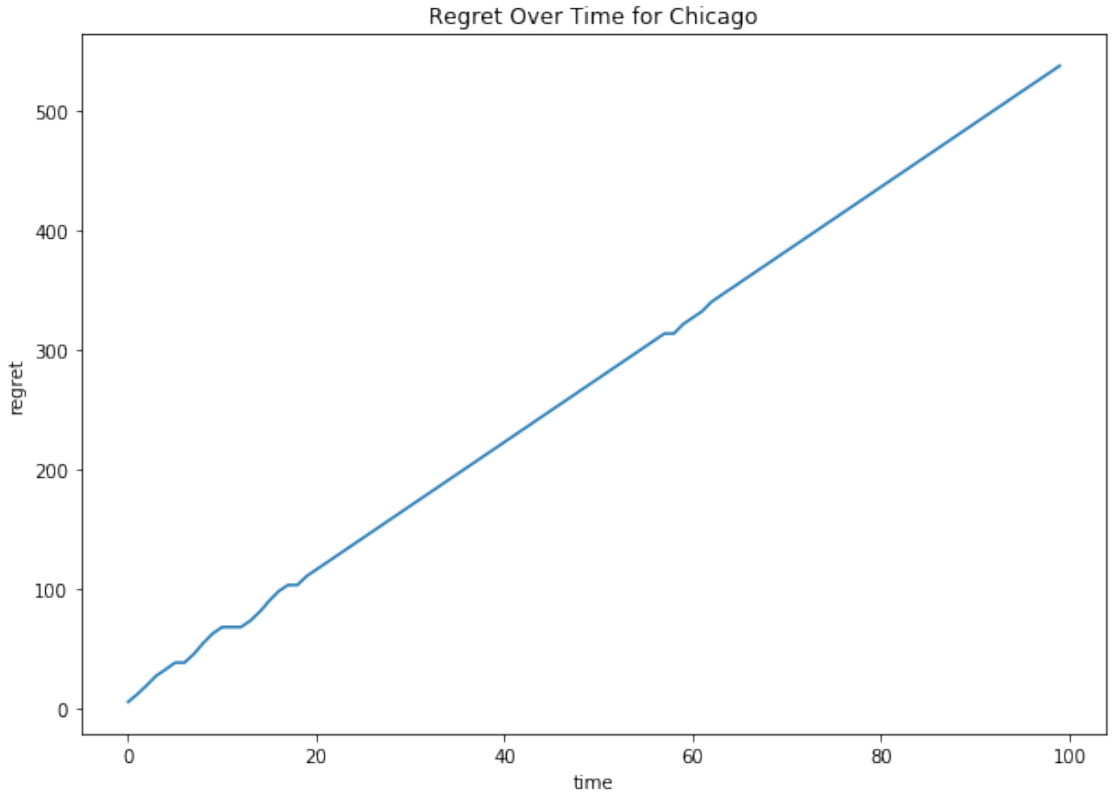

Times Pulled by Station for Chicago
station
12.899665551839465
3.8
4.911184210526316
4.944615384615385
5.410344827586207
5.590277777777778
6.370629370629371
7.206766917293233
7.56420233463035
7.7023809523809526
times pulled
time
0
20
40
60
80
100

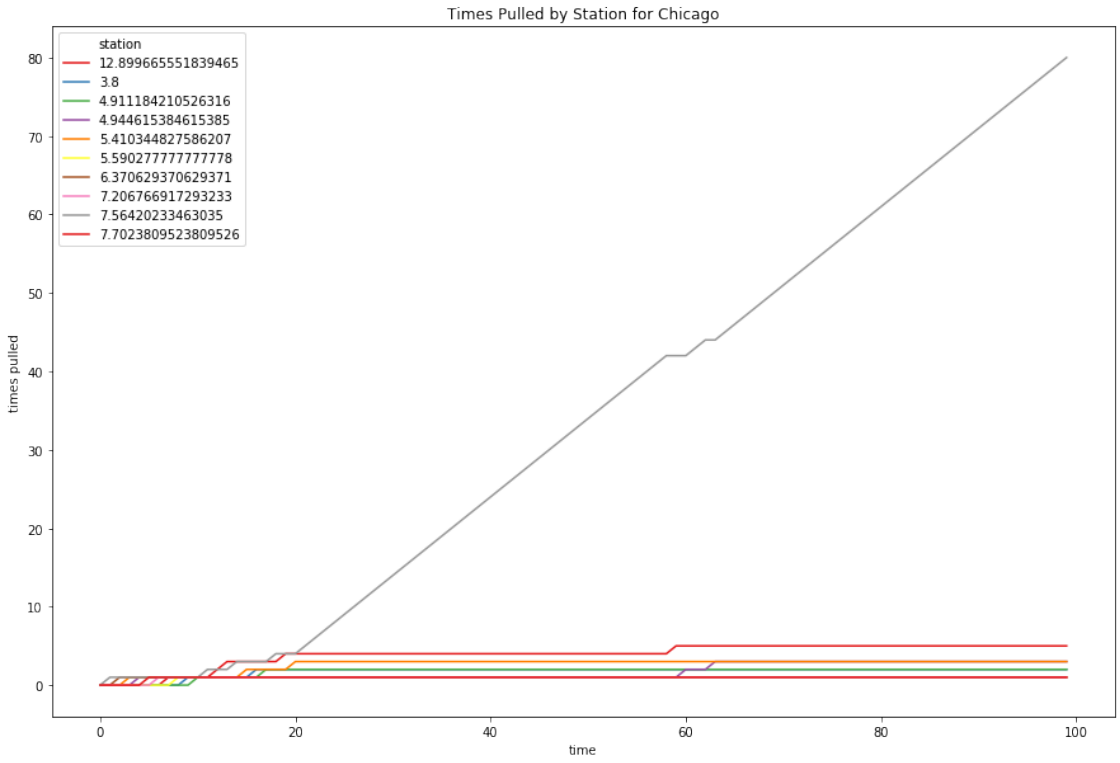

Estimated Means by Station for Chicago
station
12.899665551839465
3.8
4.911184210526316
4.944615384615385
5.410344827586207
5.590277777777778
6.370629370629371
7.206766917293233
7.56420233463035
7.7023809523809526
estimated mean
time
0
5
10
15
20
25
30
0
20
40
60
80
100

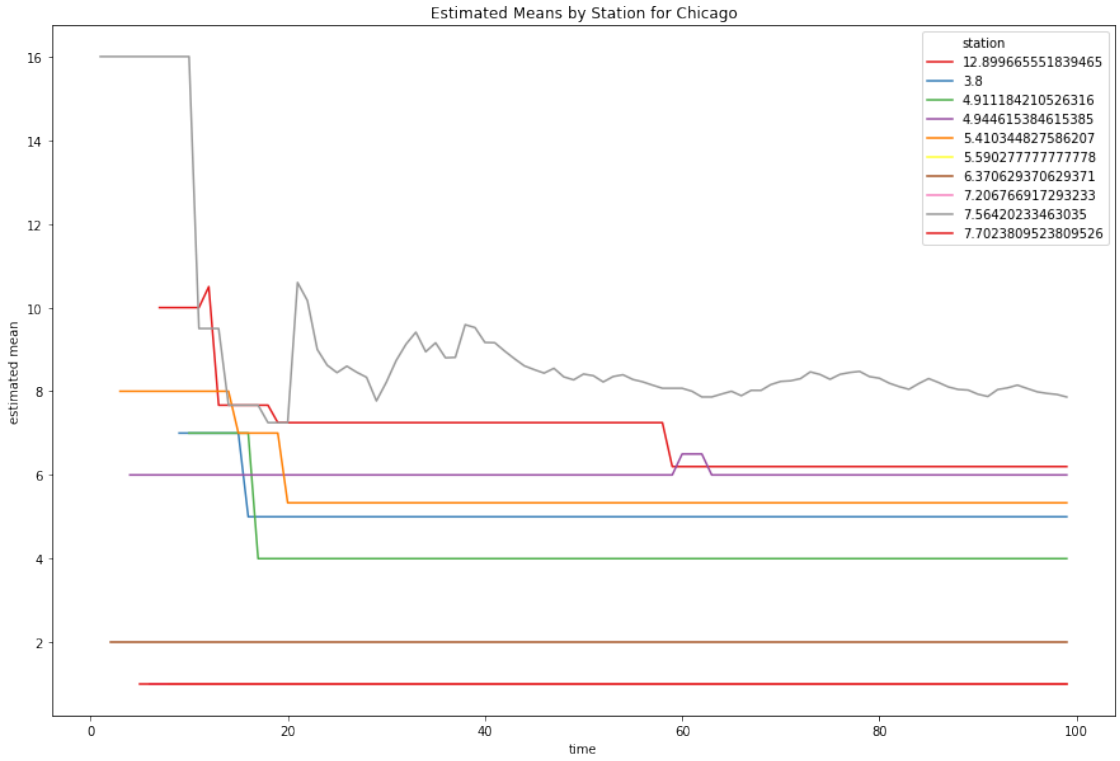


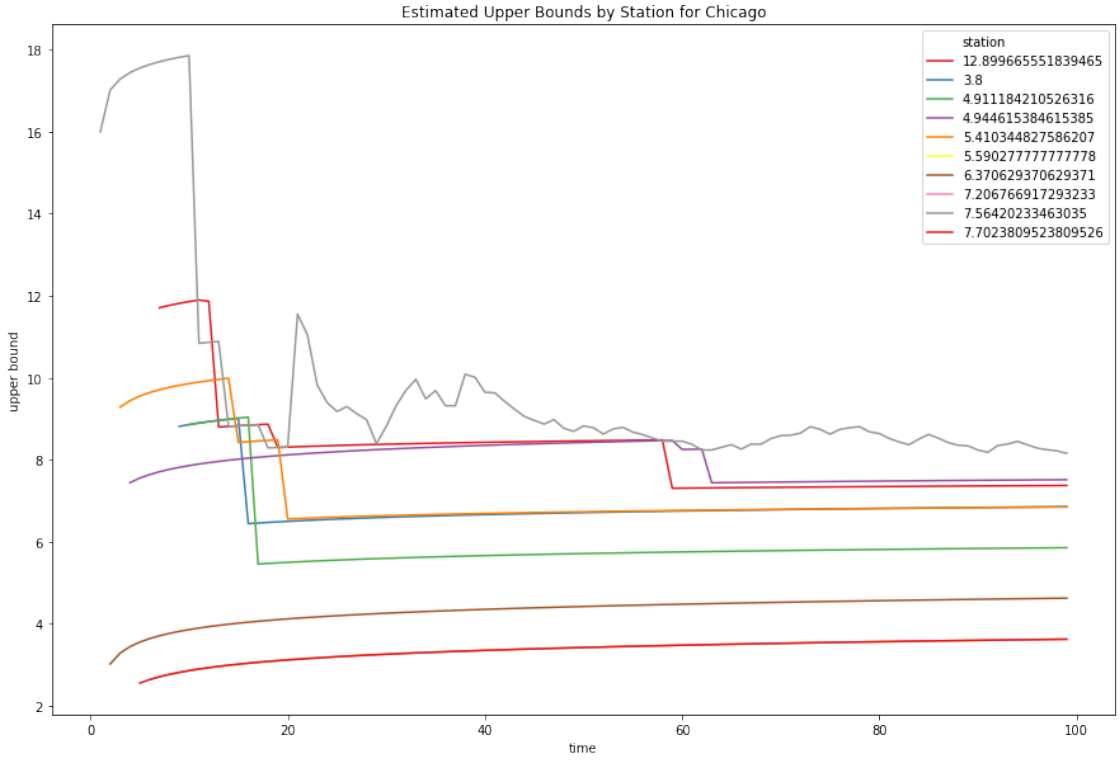
[135]:
```
np.random.shuffle(top_10_ch)
```

[137]:
```
plot_graphs(top_10_ch, ch_station_counts_top, 100, 'Chicago')
```


Regret Over Time for Chicago
regret
time
0
100
200
300
400
500
0
20
40
60
80
100


Times Pulled by Station for Chicago
station
12.899665551839465
3.8
4.911184210526316
4.944615384615385
5.410344827586207
5.590277777777778
6.370629370629371
7.206766917293233
7.56420233463035
7.7023809523809526
times pulled
time
0
10
20
30
40
50
60
70
80
0
20
40
60
80
100


Estimated Means by Station for Chicago
station
12.899665551839465
3.8
4.911184210526316
4.944615384615385
5.410344827586207
5.590277777777778
6.370629370629371
7.206766917293233
7.56420233463035
7.7023809523809526
estimated mean
time


Estimated Upper Bounds by Station for Chicago
station
12.899665551839465
3.8
4.911184210526316
4.944615384615385
5.410344827586207
5.590277777777778
6.370629370629371
7.206766917293233
7.56420233463035
7.7023809523809526
upper bound
time

1.2.2. Discussion

The results of the simulation changes over different orderings of what order to pull the arms in did seem to make a difference, but after running the simulation several times for the same orderings I noticed that the early rewards make a big difference to the way that the "arm" or street corner will be ranked later on. If the street corner with the largest average daily number of riders "rewards" us largely in the beginning, we do end up constantly picking that same street corner over time and our regret plateaus. However, when we get some small values for what we have defined as the best street corner, a different street corner that by random chance pulled larger rewards tends to be the arm that we tend to choose and our regret linearly grows. We have talked about regret in the way that we are always sending our employeee to the location with the highest average number of rides in expectation, but given the structure of the data, it may be better to possibly to somehow reduce the data to not include outliers and then send the employee to the location with the highest average number of rides for the station with the highest average after the removal of outliers. I plotted a couple of the distributions for the count of trips taken to and from a station for a couple of stations and many of them have right handed tails. This seems to pull their mean to be higher than others while maybe some other stations have a more constant slightly smaller average with all data included, but from the procedure of random sampling, picking over time from a distribution with the mass of the probability distribution centered around a slightly smaller average as compared to a distribtion with small probability for high values and the majority of the density around smaller values but with a larger average would probably tend to better results.

1.3 Takeaways

The simulation for the problem posed in section 1.1 does seem somewhat applicable to me. The way that we view each street corner as a bandit make some sense, but from what we say from part 1 of the project, weather seems to make some impact on the amount of people traveling through these stations, so maybe weather would be a better factor than location. The idea of street corners as bandits with the time horizon as the number of promotional days seems to fit as well with looking at the regret as the number of flyers we could have passed out if we stayed at the "best" location versus the expected reward we were to gain from our exploration steps. The way that the simulation seems to be lacking toward answering the question in my opinion is the fact that we are using the number of trips taken to and from a station as a proxy for the number of flyers that we can pass out. I think theoretically this makes sense, the more people there are, the more flyers you can pass out, but if we were to think about the purpose of these flyers I don't think that the simulation holds. In my opinion, I think that if we wanted to advertise using flyers we would want to pass out flyers to the most different people possible, but in the bandit problem, over time we want to continually pick the arm that has the highest mean. I would assume that a lot of the same people come and go at each street corner every day, so although they are there handing them multiple flyers over the course of several days might not be as effective as passing out flyers to maybe less people but not the same people.

The applicability of the multi-armed bandits formulation to the task at hand I think had some problems. I'm not sure if a lot of the traditional assumptions hold such as the independence of the arms since I think popularity of street corners and therefore the number of people arriving and leaving from them might be related in some way. Another assumption is that the variance of the arms is similar (at least this was the assumption that we had in lab) which does not seem to hold very well either after I plotted a couple of the trip counts for different stations the distributions seemed to vary pretty widely. This then caused for the algorithm to be quite unstable during

different runs in choosing the "best" street corner. The random draws that we had from sampling for the data we had early on in the simulation made a huge difference in the long term choice of the algorithm. For the first order of stations for each city, I ran the simulation so that regret plateued, but for a different order regret could really grow linearly as maybe a couple of lucky draws from a "lesser" street corner in the beginning and bad draws from the "best" street corner seemed to seal the fate for the street corner that we end up usually picking.

Ultimately, I don't think that I would recommend using UCB to adaptively place the person handing out promotional flyers. If it were possible I think some how modifying the algorithm to take into account different variances would help in the unstable nature of the algorithm that was experienced in the simulations above. Given my belief that handing out flyers to the same people over and over again is not better than handing out maybe less flyers but to different people would have me suggest a different approach altogether in publicizing the company. Multi armed bandit problems do not seem to have this notion that the more that you pick an arm the reward actually decays as in this case passing out flyers to the same people may not be as helpful as different people as well as the assumptions of equal variance across arms since the number of people at each station vary quite widely as well.