Part 2: Bonus 1

Based on the NY Parking Violation data, given a Black vehicle parking illegally at 34510, 10030, 34050 (streat codes). What is the probability that it will get an ticket? (very rough prediction).

Design

The K-Means algorithm is implemented with PySpark and SparkSQL, with the following steps:

1. Load the csv into a spark context as a Spark DataFrame
2. Subset the DataFrame by filtering with corresponding black cars, and then selecting columns of Street Code. At the sametimes, transform the numerical datatype to float.

```
black_list = ['BK', 'BLK', 'BK/', 'BK.', 'BLK.', 'BLAC', 'Black', 'BCK', 'BC', 'B LAC']
black = df.filter(df['Vehicle Color'].isin(black_list))
data = black.select(black['Street Code1'].cast('float'), black['Street Code2'].cast('float'),
black['Street Code3'].cast('float'))
```

In addition, converted the DataFrame into rdd:

```
rdd = data.rdd.collect()
```

4. K-Means

After preprocessing, I start to implement the Kmeans algorithm.

Firsday, randomly select 4 rows as intial centroid:

```
ini_centroid = data.rdd.takeSample(False, 4
```

Then, we run the K-Means algorithm iteratively. For each data point, we calculate their distances to the 4 initial centroids, and assign them to the cluster of their closest centroid. Next, for each cluster, we recalculate the new centroid by getting the mean of each column. By doing so, we have 4 new centriods, and we recalculate the distance between each data points to the new centroids. We iterate this process until the new centroids equal to the old centroids, or until the maximum times of iteration is reached.

- calculate each data's distance to the centroid
- asign each data to the closet cdntroid
- calculate the new centroid per cluster by finding their mean
- iteration: calculate each data's distance to the NEW centroid
- stop iteration when old_centroids = new_centroids or when the maximum number of iteration is reached

```
i = 0
old_centroid = ini_centroid
data = rdd

for i in range(3):
    out1 = closet_centroid(data, old_centroid)
    center_new = new_centroid(out1)

    if center_new == old_centroid:
        print("Converge at the %s iteration" %(i))
        print(final_centroid)
        break
    else:
        i += 1
        old_centroid = center_new
```

```
        print("Iteration - %s" %(i))

final_cetroid = center_new
print(final_cetroid)
```

The final output is the cluster that we chose to park.