Rishov Dutta
Michael Pradaxay
Jacob Carr
Wednesday, May 2nd, 2018

Chicago Crime Data Analysis

Introduction:

We did our analysis on the Chicago Crime Dataset. We chose this dataset for several reasons. Firstly, Chicago is widely known for its very high crime rate. We were interested to analyze and discover what regions in Chicago typically had the highest crime rates, what types of crime occur in what regions, and potentially interface with the Google Maps APIs to create a “Safe Rides” application. Additionally, the dataset was convenient to analyze. The dataset is updated with new crimes reported every day since 2001, so it is accurate and current. It is in CSV format with columns representing attributes of each crime, so it is easy to parse and analyze using Python. Finally, it is a reasonable size to run on the cluster, at 6.55 million rows, or approximately 3GB.

Description of Dataset:

The Chicago Crime Dataset is structured in the following way. Each row represents a crime, and the columns (shown below) represent attributes of each crime. We found some of these columns to not be particularly useful (for example, we can’t do meaningful analysis on the Case Number or IUCR), but many columns were useful for analysis, including Date, District, Latitude, Longitude, Location, and Year.

Column Name	Description	Type
ID	Unique identifier for the record.	Number
Case Number	The Chicago Police Department RD Number (Records...	Plain Text
Date	Date when the incident occurred. this is sometimes a...	Date & Time
Block	The partially redacted address where the incident oc...	Plain Text
IUCR	The Illinois Unifrom Crime Reporting code. This is dir...	Plain Text
Primary Type	The primary description of the IUCR code.	Plain Text
Description	The secondary description of the IUCR code, a subcat...	Plain Text
Location Description	Description of the location where the incident occurr...	Plain Text
Arrest	Indicates whether an arrest was made.	Checkbox
Domestic	Indicates whether the incident was domestic-related ...	Checkbox
Beat	Indicates the beat where the incident occurred. A bea...	Plain Text
District	Indicates the police district where the incident occurr...	Plain Text
Ward	The ward (City Council district) where the incident occ...	Number
Community Area	Indicates the community area where the incident occ...	Plain Text
FBI Code	Indicates the crime classification as outlined in the FB...	Plain Text
X Coordinate	The x coordinate of the location where the incident o...	Number
Y Coordinate	The y coordinate of the location where the incident o...	Number
Year	Year the incident occurred.	Number
Updated On	Date and time the record was last updated.	Date & Time
Latitude	The latitude of the location where the incident occurr...	Number
Longitude	The longitude of the location where the incident occu...	Number
Location	The location where the incident occurred in a format ...	Location

Frameworks:

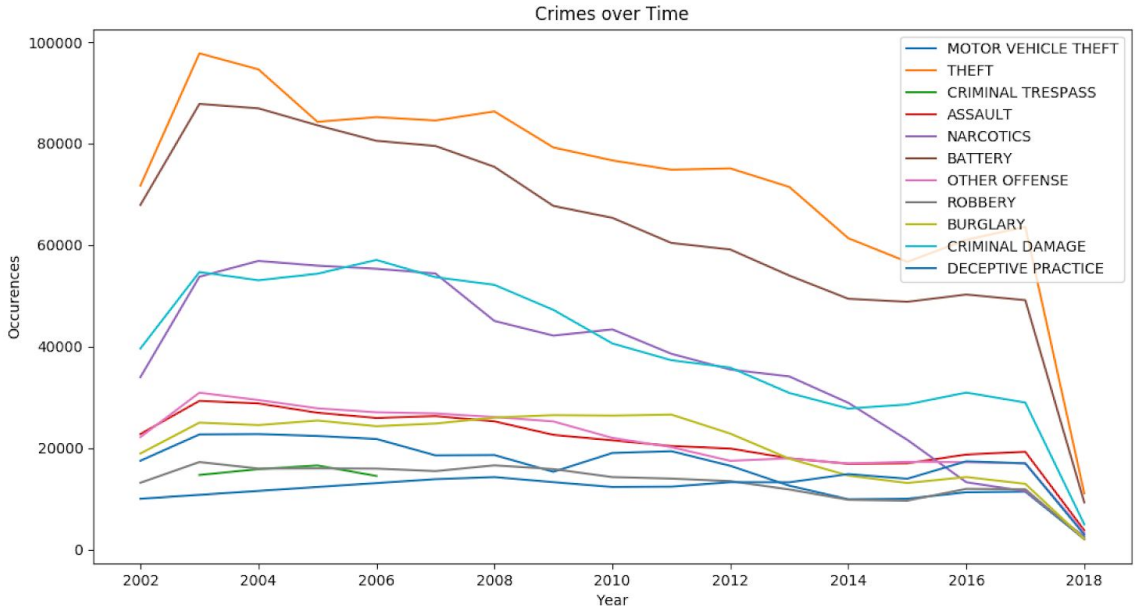
The application makes use of Spark SQL as well as Spark ML/MLLIB. SQL was used for all of the data collection steps as it is easy to use on large data sets. We used ML/MLLIB for k-means clustering based on location to determine the clusters of crimes in the Chicagoland area. The results of these can be found below as well as how we got these results with some code samples.

We also used Folium to generate cluster maps as well as heat maps in order to visualize the data in a more intuitive way. We used marker clustering so that when we zoom in and out, we can see the overall crimes in a zone as a number, then go into more specifics as to where exactly on the streets the crime occurred as well as what the crime was. Folium was also used to generate heat maps based on the number of crimes reported in an area. It is similar to the clustering map but provides a more qualitative view of crimes in certain areas vs. others. Again when zooming in and out of this map we get a more detailed or general look of crimes in the area.

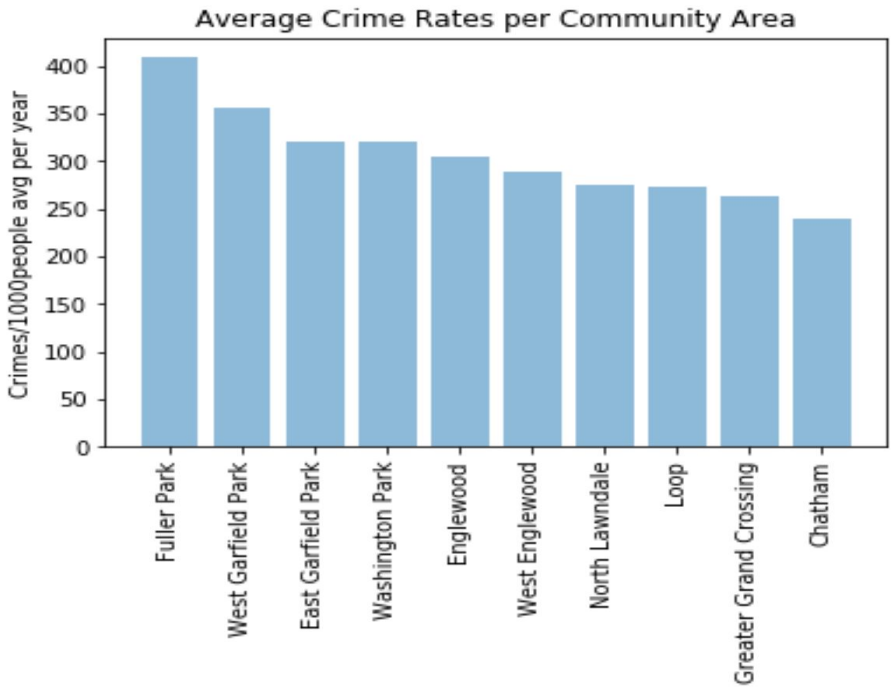
Lastly, we used OpenRouteService to create a GPS like function to avoid certain high density crime areas. These results can be found in route_comparison.txt where we compare a route with avoiding an area to a route without avoiding. Again these results and more details regarding the code are in the results section.

Results:

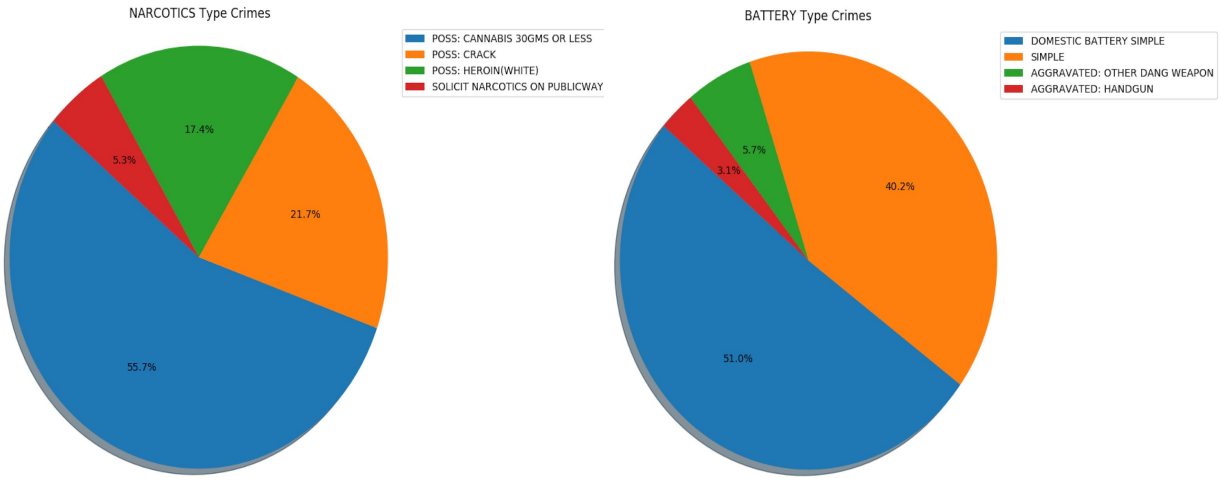
Through the use of PySpark's SQL module, we are able to easily parse the data and retrieve the parts that were important to our research. The Chicago crime data was thankfully organized relatively neatly to allow for us to run our queries without having to do much cleaning up on the raw data. We are then able to produce visualizations such as those below to help us see crime behavior in Chicago. In the line chart below, we can prominently see the four most common types of crime (Theft, Battery, Criminal Damage, and Narcotics) over the time span of the data set. We are also able to clearly see trends in the number of occurrences of these types of crimes over the years. Overall, we can see a downward trend from 2003 to 2017. The sudden drop at 2018 is most likely because that is the current year. A rapid drop in narcotics crime reports can be seen after 2013. Being able to see these trends and to then try to find explanations for them could be important for organizations running things such as anti-drug campaigns to see if a campaign technique is effective over time.



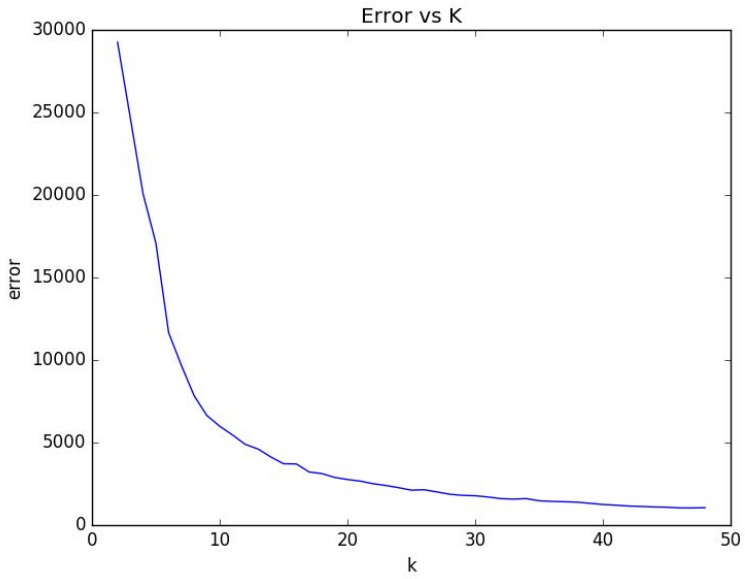
We are also able to see things such as dangerous areas of Chicago as described by the number of crimes. The data set uses numbers to identify community areas, so we had to combine that data with a list of community area numbers and names.



Reports also contain a general description in addition to the primary type of crime and using these we are able can see specifically see what crimes make up these primary types. Information such as this can help point toward what area of crime prevention would need more attention.



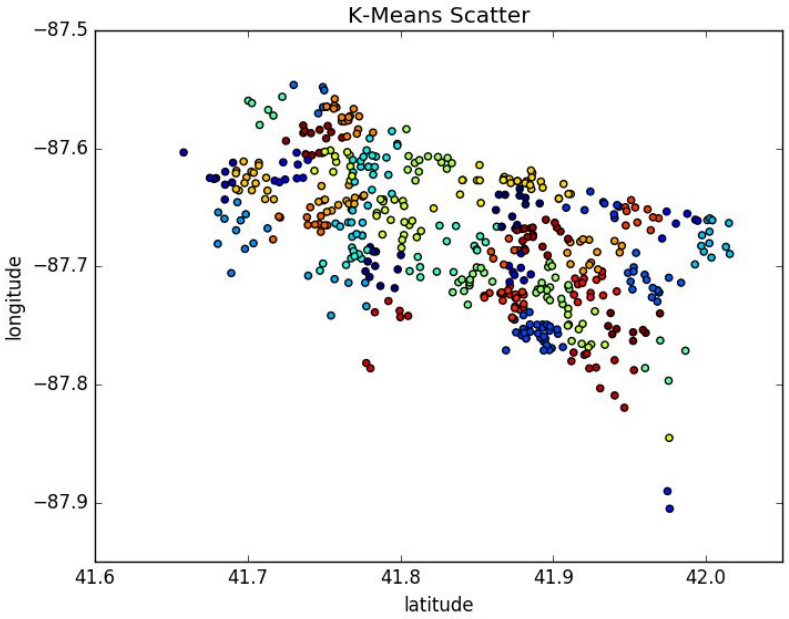
(k_plot.png)



This was generated by running k means clustering in sparkML ~50 times and finding where diminishing returns start for the value k. It was found that after 48, k begins to see diminishing returns so 48 was chosen as the overall k for clustering in the future.

```
cost = [0] * 50
for k in range(0,50):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
    model = kmeans.fit(df_kmeans.sample(False,0.1, seed=42))
    cost[k] = model.computeCost(df_kmeans) # requires Spark 2.0 or later
print (cost)
```

(kmeans_plot.png)

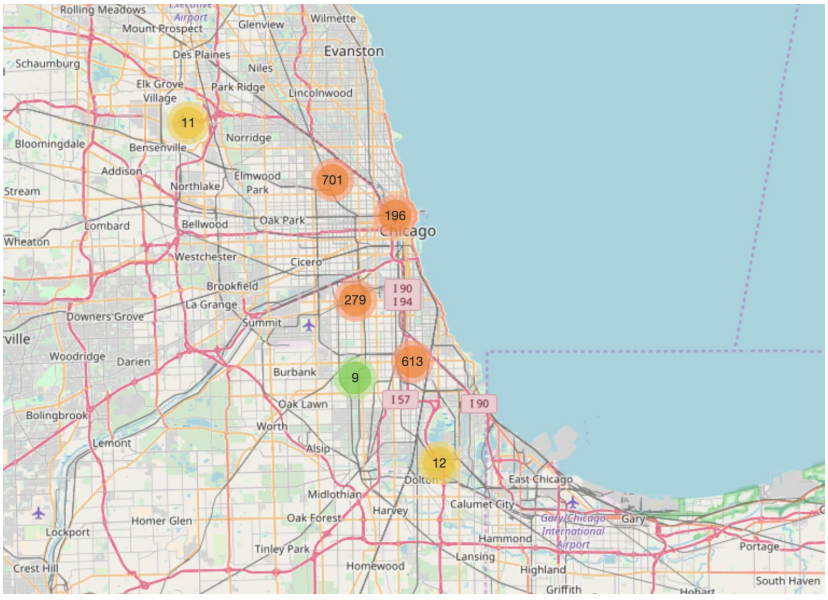


This was generated by collecting the prediction values from the data coloring them different colors and plotting it on a scatter plot.

As we can see, there are more clusters in the areas with a lesser latitude value, indicating there are more crimes the further south in Chicago you go. There are some hotspots in the northern parts where the latitude values are larger, but in general south side Chicago is significantly more dangerous than north side.

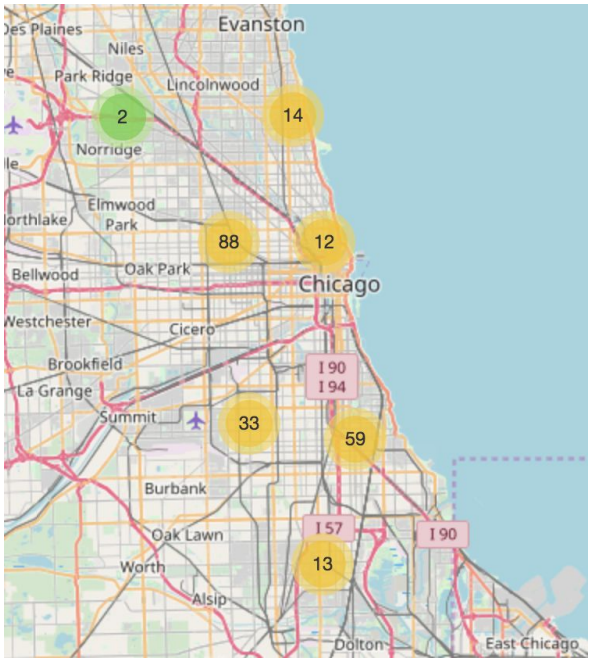
Total Crimes

(total_crimes.html)



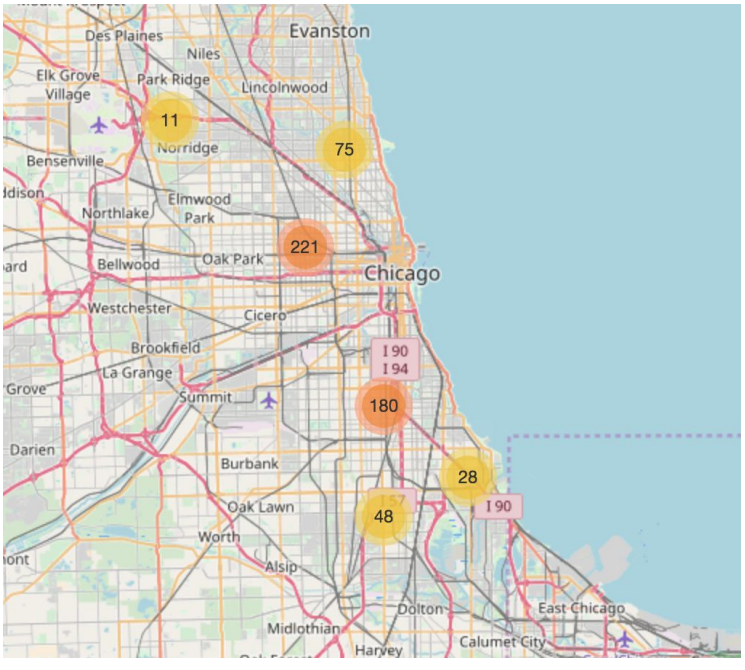
Night

(night.html)



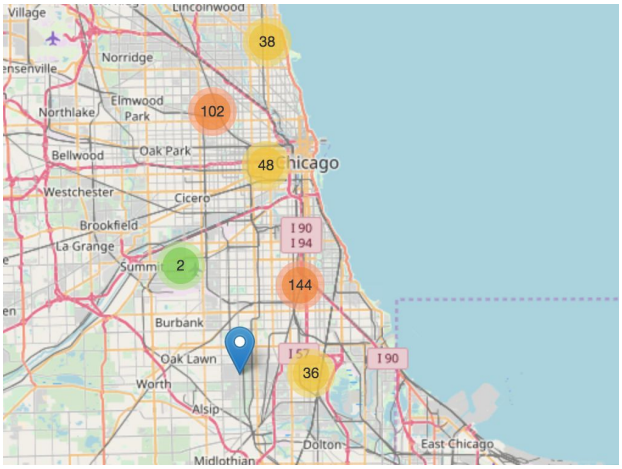
Day

(day.html)



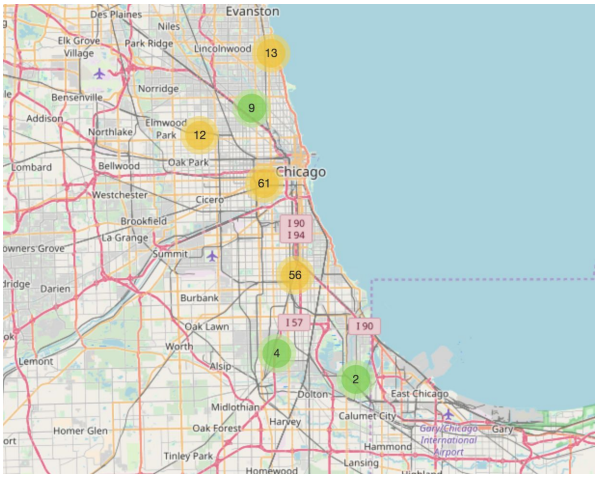
Weekday

(weekday.html)



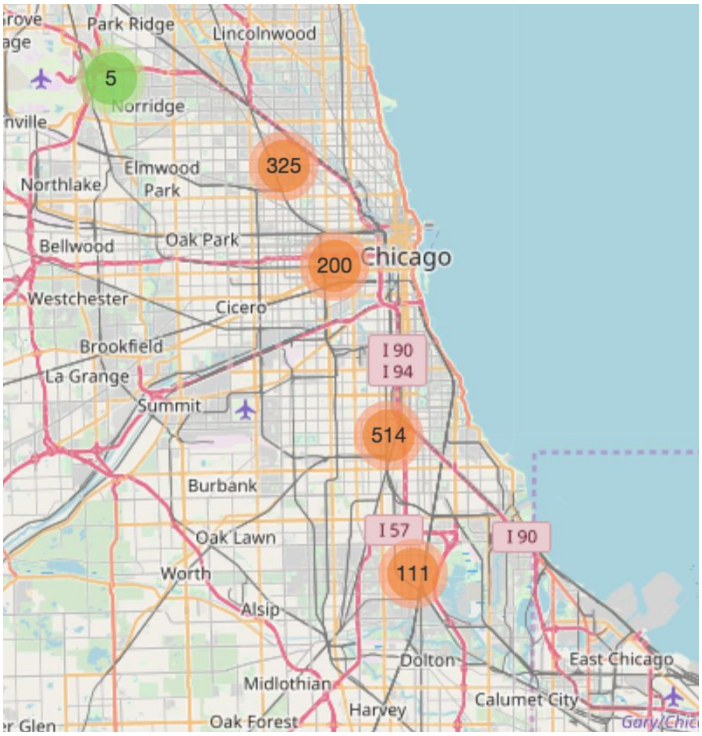
Weekend

(weekend.html)



Severe Crimes

(severe.html)



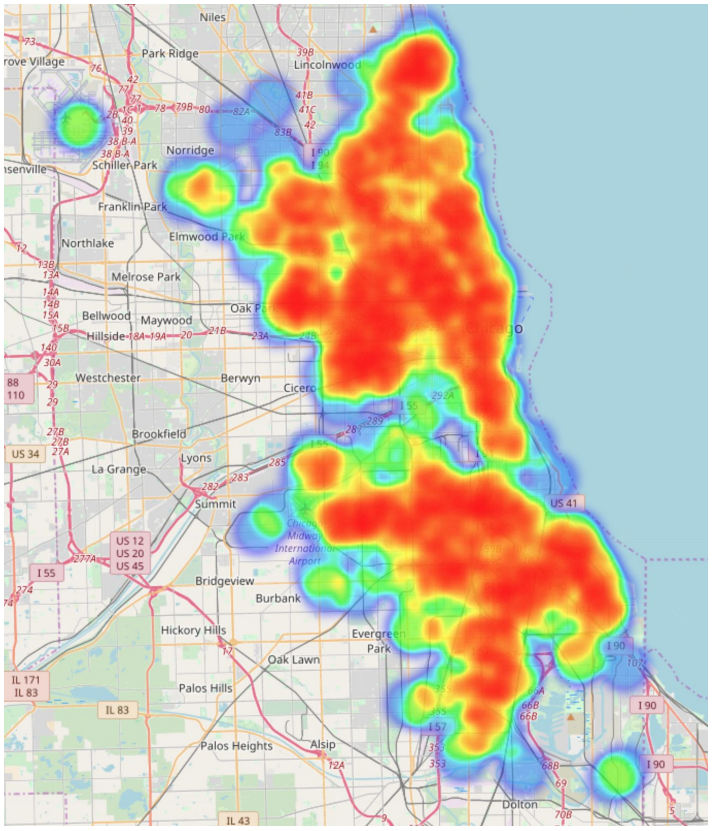
These graphs were all generated from essentially the same code, just changing what values are put into the final rdd.

```
def create_visuals(df):

    latitude = df.select(F.mean('Latitude')).collect()[0][0]
    longitude = df.select(F.mean('Longitude')).collect()[0][0]
    chicago_map = folium.Map(location=[latitude, longitude],
                  zoom_start=10)

    marker_cluster = MarkerCluster().add_to(chicago_map)

    locationslist = df.rdd.map(lambda x: ((x.Description, x.prediction), (x.Latitude, x.Longitude)))
    locationlist = locationslist.collect()
    for index in range(0, len(locationlist)):
        folium.Marker([locationlist[index][1][0], locationlist[index][1][1]], popup=locationlist[index][0][0]).add_to(marker_cluster)
    chicago_map.save('total_crimes.html')
```

As we can see, there are some specific trends in all these graphs. The middle to south side of Chicago has significantly more crimes than northside. This is presumably because people, in general, are wealthier in north side than southside per multiple sources. We also see a direct correlation between severe crimes and location, with southside having significantly more severe crimes in about the same area as northside chicago. These interactive maps can be found in the github repository and can be accessed by just double clicking the html file and accessing in a web browser.

Heatmap

(time_heatmap.py.html)



Similar to the cluster maps above, the heatmap shows that chicago has a lot of crime in its southern and central parts relative to the northern parts. This can be due to many factors such as population, overall income, etc.

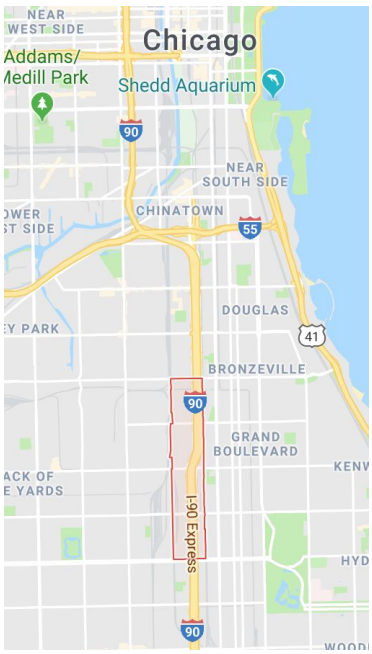
route_comparison

(route_comparison.txt)

```
Route with avoiding
===================
Total time of trip:     172.3 seconds.
Total distance of trip: 1.206 miles.
Total steps of trip:    6 steps.
1. Head south on Dan Ryan Expressway, I 90, I 94 for 0.393 miles (35 seconds).
2. Turn slight right for 0.219 miles (21.1 seconds).
3. Continue straight onto South Wells Street for 0 miles (39.2 seconds).
4. Turn right onto West 57th Street for 0.029 miles (11.3 seconds).
5. Turn right for 0.227 miles (65.7 seconds).
6. Arrive at your destination, on the left.


Route without avoiding
======================
Total time of trip:     79.4 seconds.
Total distance of trip: 0.815 miles.
Total steps of trip:    8 steps.
1. Head south for 0.032 miles (3.1 seconds).
2. Continue straight onto Dan Ryan Expressway, I 90, I 94 for 0.361 miles (32.2 seconds).
3. Turn slight right for 0.219 miles (21.1 seconds).
4. Continue straight onto South Wells Street for 0.086 miles (10 seconds).
5. Turn right onto West Garfield Boulevard for 0.062 miles (5.5 seconds).
6. Turn left onto South Princeton Avenue for 0.025 miles (4.9 seconds).
7. Turn left onto West Garfield Boulevard for 0.029 miles (2.6 seconds).
8. Arrive at West Garfield Boulevard, on the left.


Comparison Statistics
=====================
Avoiding took 92.9 more seconds.
Avoiding took 0.391 more miles.
Avoiding took 2 less steps.
```

Avoiding Area

(avoiding_area.png)



```
#Enter these values to change routing
lat1 = 41.8046
lon1 = -87.6323
lat2 = 41.7943
lon2 = -87.63244
```

The routing service is pictured above. It returns a list of instructions on how to get from point a to point b while avoiding the area pictured above. We chose to avoid Fuller Park because based on our analysis of the data, it was the most criminally active part of Chicago. It is a well defined area and we were unable to figure out how to extrapolate this to automatically determine coordinates that are compatible with the API. Because of this, we manually inputted Fuller Park coordinates. As seen by the code snippet above, we can manually input coordinates to traverse and the output, as seen above, will give two routes, one with avoiding the block area and another without. It will finally return the comparison at the end with distance, time, and total steps difference between the two routes. Obviously, avoiding routes that go into the area in general take more time than not avoiding.

Note: Please refer to the results folder in the repository to access the interactive maps. They're pretty cool.

Issues:

We encountered some minor issues while analyzing the data. One problem we faced was that much of the data was filled with garbage values. For example, some rows in the dataset left no explanation of what the crime was. We worked around this by dropping the garbage values from the table before performing our analysis. Another complication we encountered was that we couldn't use the entire dataset to create our cluster map. There were too many data points, and the job would fail when trying to use all of the data. We solved this problem by only using 0.1% of the data to create the map. This still results in a fairly accurate representation of the data, and is easy to generate with our resources. Finally, a minor problem we faced was that the cluster would occasionally lose executors, cancelling the entire job.

Applications:

As discussed above, we used OpenRouteService to create a GPS like function to avoid certain high density crime areas in Chicago. This functionality could potentially be added into Google Maps or other commercial GPS map systems, to provide safer routes based on current crime data. Furthermore, police departments and law enforcement officials could make decisions about what regions to cover more heavily based on where crime is most likely to occur, and what types of crimes are most likely to occur there.

Performance Report:

We ran the code approximately 15 times and the average runtime turned out to be 5 minutes and 13 seconds. Converting to seconds gives us 313 seconds. With over 6.55 million records, our yield was approximately 20,927 records per second. This is most likely because we create our data visualizations near the end of running our job, which causes the job time to increase greatly since we collect a fraction of the output data sequentially and print it to file.