Xtern
Data Science Report

October 14, 2019

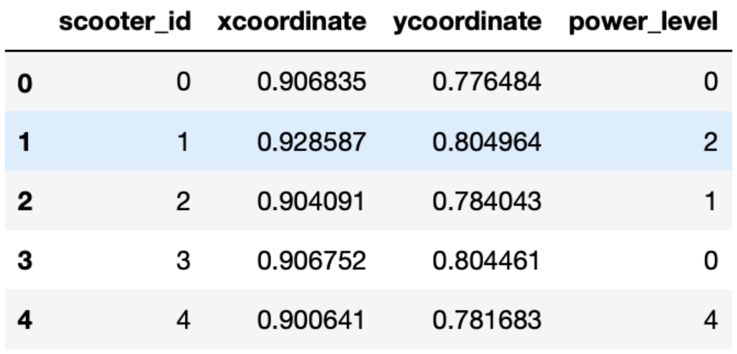
Muskan Uprety

Introduction

For the project, I was given a data set containing the geo-location and the power level of electronic scooters for a e-scooter ride sharing company. The power level of these vehicles ranged from 0 to 5. There is a bus that charges these scooters and it can charge a scooter from zero to five in 5 hours. I have analyzed the data and found some insight, which I have showcased in this report. I am also proposing a strategy to charge all these scooters and have calculated the Operation Time Cost for the process.

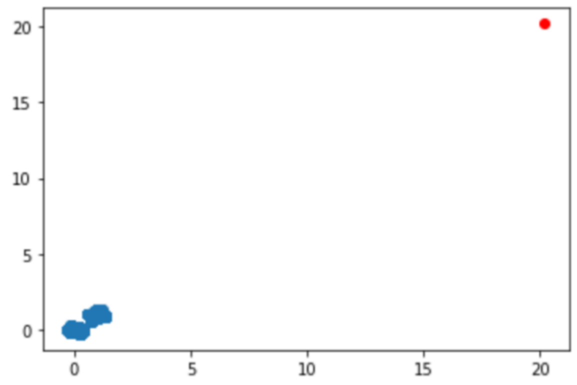
All the findings in this report, and the code used to get these findings is from the python notebook that is supplementing this report.

Initial look at the Data Set:



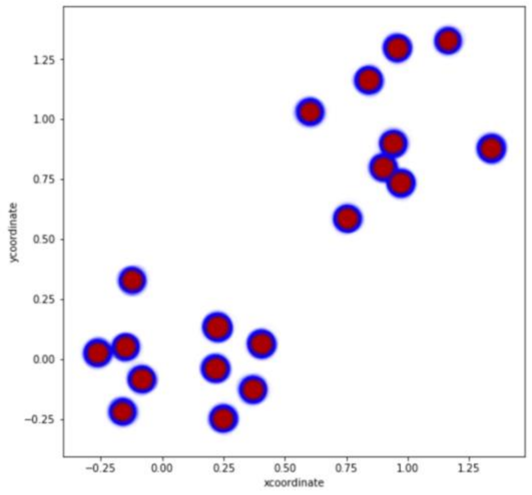
	scooter_id	xcoordinate	ycoordinate	power_level
0	0	0.906835	0.776484	0
1	1	0.928587	0.804964	2
2	2	0.904091	0.784043	1
3	3	0.906752	0.804461	0
4	4	0.900641	0.781683	4

The coordinates of the location where the bus is currently situated is given as (20.19, 20.19). So I graphed the location of all the scooters in Blue, and the location of the bus in red to see how the scooters are spread out and how far they are from the charging bus.



Graph: xcoordinates and ycoordinates of scooters (blue) and the charging bus (red)

Since the location is very spread out, I decided to graph only the scooter locations to get a more clearer picture.



The color blue in the graph represent the scooters with power level of 3 and higher, while the red are scooters with low power levels (0-2).

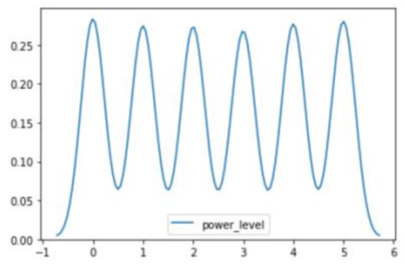
There are 19 circles in this graph which I am assuming are docking stations/ parking area for these scooters. So the company is doing a good job in managing their fleet and ensuring people are not parking scooters too far from the designated areas.

Also, looking at the graph above, there is a clear divide in the middle between these geolocations. It probably shows that these are 2 different major cities. I have labelled them as city 1 (the one at the top right of the graph with 9 circle points) and city 2 (one at the bottom left with 10 circle points).

There are a total of 25,668 scooters which are spread out pretty evenly in power level. The count of scooters and their power level is given below:

Power Level	No. of scooters
0	4388
1	4248
2	4245
3	4160
4	4284
5	4343

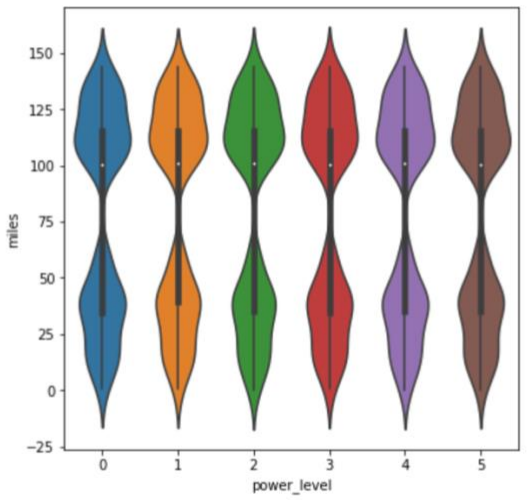
Kernel Density estimate showcasing the information in the table:



Distance Calculation

I created another column labelled 'Miles' that calculated how far each scooter is from the parked bus location. The minimum value of this column, i.e., the scooter closest to the bus is 1824.47 miles away from the parked bus. This figure is important because the bus has to first get there before starting to charge the scooters, and the fastest it can start is by getting to the one that is closest.

After calculating the miles column, I subtracted 1824.47 from each row to get a sense of how far they are from each other. I also created a violin plot to analyze the spread of the scooters in miles, by their power levels.



We can clearly see the divide between the 2 'cities' in this plot as the frequency of scooter is nil from 70-80 miles from the first scooter. Like mentioned above, the scooters are evenly spread in terms of their power levels. But the violin plot is a little thicker on the top end than the bottom, suggesting there are more scooters overall in city 2. This could mean that city 2 is bigger, with more customer base.

I removed all the scooters in the data set that has power levels of 5 because they didn't need to be charged. So I omitted them so that the bus doesn't have to go to these coordinates. I also calculated the mean distance (in miles) which was close to 78. So, I divided the total data set into city1 (scooters closer than 78 miles) and city2 (scooters further than 78 miles).

My Strategy

My strategy to charge these scooters was to initially get the bus to the closest scooter, and then start picking up the scooters that are closest to the bus. The bus will pick a scooter, go to the next coordinate, and continue the process until one of the scooters inside the bus is at max power. Then it will drop it off at whichever stop it happened to be in. The bus can only drop a scooter in a location where one scooter already exists. One geo-location can have multiple scooter at the same time as it gets dropped off by the bus.

Some assumptions made:

- Since we don't have any information about the road structure in this hypothetical location, I am making an assumption that there are straight paths going from one location to another.

- After calculating the miles column, I sorted the Data frame based on the mile figures. This was done to figure out which coordinate to go to next. The distance between 2 coordinates may not be the smallest just because they are equally further away from a given point (parked bus location in this case) but this was the simplest approach, and the calculation was pretty accurate.

- We don't know the capacity of the bus to hold scooters. So, I am assuming it can hold scooters as needed. I haven't accounted for a situation where the bus is at max capacity and cannot take any more scooters.

- The bus is a MEGA Bus, so I have assumed that it does not require any rest, refueling, or any other overhead of the sort. (Hopefully it is self-driving so no person is being worked like this)

- We have not taken into account how long it takes to get to a location, load, and unload scooters.

Process

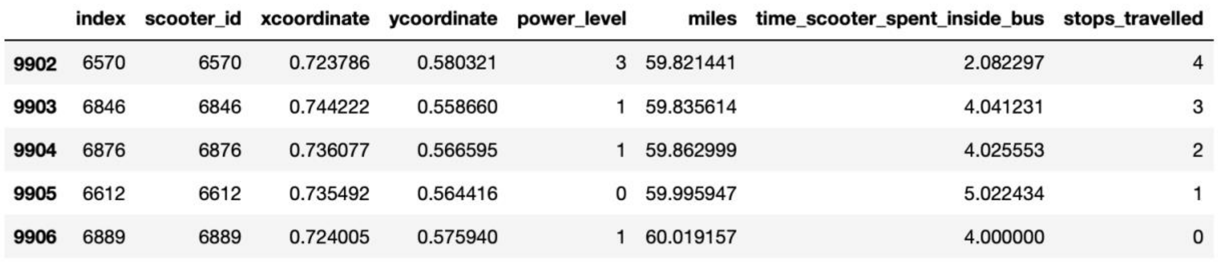
I added 2 columns in the dataset for city1 called 'time_scooter_spent_inside_bus' and 'stops_travelled'.

The first row shows how long the scooter has been in the bus, and the second row tells us how many stops each scooter had to go through before it was fully charged.

First, I created a dictionary that stored the time taken to travel from one consecutive row to another. These rows are sorted by the miles, so I made a strategy to go to the next closest scooter.

The function charge_scooter uses this dictionary and calculates values for the rows mentioned above. It continues to look at the next stops until there are next scooters available. If next coordinates are not available, the function assumes that the bus is just going to drive around until all the remaining scooters are fully charged. It increments the time by half an hour each time it runs through the loop. We can tailor this for desired accuracy. The function will return the extra time the bus had to drive around because there were no next stations to charge the remaining scooters in the bus.

The function took too long to process through the entire dataset (I ran it for about an hour before stopping it) so I reran the function only for city 1. Checking the tail of the data frame tells us that it ran to its entirety.

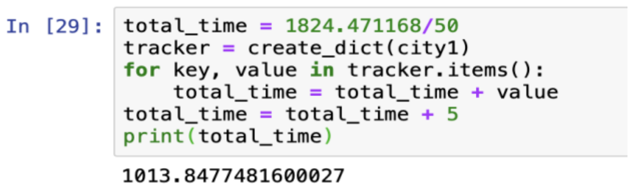


	index	scooter_id	xcoordinate	ycoordinate	power_level	miles	time_scooter_spent_inside_bus	stops_travelled
9902	6570	6570	0.723786	0.580321	3	59.821441	2.082297	4
9903	6846	6846	0.744222	0.558660	1	59.835614	4.041231	3
9904	6876	6876	0.736077	0.566595	1	59.862999	4.025553	2
9905	6612	6612	0.735492	0.564416	0	59.995947	5.022434	1
9906	6889	6889	0.724005	0.575940	1	60.019157	4.000000	0

Extra time required for city 1 returned 5 hours. It makes sense since the second to last scooter had a power level of 0 and it only had one more stop to go through which was less than an hour away. So, it needed the full extra 5 hours to charge. Since the bus has to be parked so far away, we cannot use this time to return as doing so would take these scooters away from the city.

CITY1

Before reaching the first scooter, the bus had to travel 1824.47 miles. The speed of the bus is 50 miles/hr. So We have to calculate the time taken to get there first to compute OTC. I used the following code to calculate the total time taken just for city1



```
In [29]: total_time = 1824.471168/50
         tracker = create_dict(city1)
         for key, value in tracker.items():
             total_time = total_time + value
         total_time = total_time + 5
         print(total_time)

         1013.8477481600027
```

I could not do it for the whole data set due to the limitations mentioned above. However, better computational resources, and possibly working on optimizing the function would allow us to calculate the overall time overhead of the entire project.