Analysis and Optimization for NYC Public Transportation Alternatives

|  |
| --- |
| Hongzhi Shi  Columbia University hs3194@columbia.edu |

Abstract

*NYC has a lot to offer when it comes to public transportation alternatives. They not only bring convenience and variety for commuters, but also complements the city public transportation system. Understanding the overall trend and dynamics is key for decision maker to better design and implement the future city transportation system. More importantly, knowing the pros and cons of these services and having some useful tool to help navigate this space is critical for individuals to make better choices, which in the long has a significant impact on both money and time spent.*

# Introduction

*As larger volume of urban data are captured and become available, more data-driven analysis is now possible, which can lead to improvements through evidence-based decision making and policies. The sheer amount of data, however, presents its own challenges. One the one hand, people are always interested and driven by the desire of gaining a macro view of how things are going. The larger the data volume is, the harder it becomes to run any meaningful analysis. The old model of running queries on a relational database on a single machine can no longer suffice in the big data era, which calls for new ways of thinking and new tools that leverage more parallel and distributed computing models. In this project, I used Google cloud platform and Pyspark to perform various analysis on the big data set. On the other hand, it’s very tempting and easy to dive deep into a very narrow and specific area as the granularity and amount of data available in some particular area is unprecedented. For example, there were various publications on how Lyft compares to Uber in NYC and How Juno is bring new challenges to the incumbents. In this project, I was trying to hit a middle ground by focusing not on a particular tool or service, but rather on different competing components in the same space. The rationale behind this is that people are not particular about a specific tool, rather they’re going to choose the best one to perform the task. Specifically, my target space is the public transportation alternatives for commuting in New York City and within this space, there’re three key components: Taxi service, for hire vehicle and Citi bike. Each component serve a different purpose. Taxi, being the incumbent, used to be the only choice and still have the most availability and is the majority on the street. For hire vehicles provide the convenience of hailing a car from wherever you are and not having to go outside. This is especially attractive in some bad weather conditions. They are also more user friendly by providing transparency of the vehicle locations and up front charges. Citi bike, being the most environmentally and budget friendly, is expanding rapidly in the past couple of years and gaining popularity among more commuters largely due to its easy access and free of traffic congestions impact.*

# Related Work

There have been plethora of publications on the landscape of the for hire vehicles in NYC in the past couple of years following the surge of popularity of Uber, Lyft and etc. One example is this article titled ‘Taxi, Uber and Lyft usage in New York City’ from Todd W. Schneider, where the author detailed some usage statistics of Taxi, Uber, Lyft, Juno, Via and Gett with a focus on the overall trend of these services starting from 2016. But the major concern of this analysis if how each service compares to the others in the ride hailing space. It didn’t compare Taxi and for fire vehicles as a whole, neither did it include Citi bike in the analysis. Another article from Business Insider titled ‘The Biggest different between Uber and Lyft, the two biggest ride hailing apps’ takes a deep dive into these two specific applications and did a side by side comparison from pricing, user interface, rush hour price surge, vehicle options perspective. This analysis is very focused on the for hire vehicle space and only two major players in this space. The way this analysis is conducted is mostly by one person experiencing these two apps multiple time throughout the day and is very subjective to personal taste since there isn’t much data backing the author’s statements. There are lots of other related analysis, but most of them are from the perspective of comparing one particular service to another. I can hardly find any publication or article that analyze the different NYC public transportation alternatives from a commuter’s perspective with a very data driven approach, hence the motivation of this project.

# Datasets

## Past 3 years of aggregate data. From TLC we can get a single csv file with monthly indicators for yellow/green taxi as well as high volume for hire vehicles. This file contains records dating back to 2011 and has more fields than we are actually interested in, for example, columns like ‘Unique drivers’, ‘Percent of trips paid with credit card’ are not of our concern, so we filtered out both the uninteresting columns and the rows beyond our 3 year time frame. Moreover, since our analysis will treat yellow and green taxi as one category, we also grouped these two together to aggregate them into one record for each month.

## Past 3 years of trip data from Citi bike. We need these data for two purpose. First, since Citi bike doesn’t publish any monthly aggregate, we need to do it manually from the trip data. Second, we need this to calculate the estimate of how long it takes to get from one neighborhood to another. For the first purpose, we strip off bunch of columns like ‘bikeid’, ’usertype’, ’year’, ’gendar’ and we also filtered out the trips that starts and ends at the same station with a duration of less than one minute since this mostly likely is due to some user error. Trip data is organized per month into a csv file with millions of records, we gathered roughly 6G of data.

## Past 3 years of trip data from TLC for Taxi. Similar to what we need to the Citi bike trip data, we filtered out bunch of unnecessary columns and since this is only for the second part of the project, we also convert the timestamp to a time bucket like ‘8:00AM – 11:00AM’ and ’11:00AM – 4:00PM’.

## NYC borough map file from NYC Open Data. It’s available for download in different format and we used shapefile format.

## NYC Taxi zone file from TLC in a GeoJSON format. We need to join the Citi bike station with these Taxi zone using BigQuery GIS, which support GeoJSON natively.

# Methods

## Data analysis

Producing some aggregate report from transactional data isn’t something new. Overall there are three options and in this section, we’ll examine each of these in more detail.

### Relational database like PostgreSQL. Until the past decade, this was the default and only option. Based on bunch of normalized tables, each with a consistent schema that governs how data looks like and what kind of data can be inserted into a particular table. Lots of work have been done to make the query optimizer better and faster. The Relational database is one piece fit all model and takes care of all the data storage, transaction and retrieval with a strong ACID semantics. We think this is great for applications with small to moderate volume of data since this one tool can basically cover the end to end of entire applications data flow and analysis. However, there’re two major issues that prevent us from using this method. First and foremost, it doesn’t scale very well across machines and clusters for huge amount of data. Even though all the data can fit into the database, the nature of how SQL handles JOIN is eventually going to be a bottleneck for this type of big data analysis. The other drawback is that most of the queries on a relational database is written once and run multiple times. In the contrast, we want to run some analysis and based on how the result looks like, further decide what we need to run. This type of iteratively revising and chaining query and analysis is not very well supported in relational database.

### Map Reduce. Unlike relational database, which bundles both the data storage and the computation and analysis of the data together, Map-reduce decouples these two and focuses on the parallel computation of data and makes the analysis on large scale data possible. Map reduce parallelize thousands of jobs by splitting the input into a large number of small chunks and assigns each chunk to a local job. The job tracker keeps track of the states of each job and retries once a failure is detected. As a computation engine, it employees a very simple model of map and reduce. Surprisingly, the two seemingly simple operator map and reduce can be chained to express some of the very complex queries and computation. Map reduce provides a new way of thinking in the big data era and makes data analysis on large scale data that cannot be handled in relational database possible. However, there are couple of drawbacks that prevented us from using map reduce for this project. First of all, though chaining map and reduce in multiple stages can be pretty expressive, it’s still not easy to create an equivalent of some ad hoc functions. Also, the map reduce is some functional programming concept, for most of the programmers whose first language is object oriented, it’s not intuitive to think and write queries in that model. It would be good if one can write query in a programming language that he/she is most comfortable with. Secondly, granted that map reduce overcomes the limitation of most relational database in the sense that it makes analysis on large scale data possible, it’s still not very performant largely due to that fact that map and reduce are two separate phase and the second phase cannot be started until the first phase is fully finished. That being said, even though one can maximize the parallelism in a particular phase, resource cannot be shared across different phases. If one map job encounters some error and keeps failing, even all other map jobs are done, all the resource has to sit idle waiting for that one map job to be retried and completed until anything can be started in the reduce phase. This, in practice, can result in a significant slowdown especially given that any non-trivial query involves dozens of map reduce chained together. Thirdly, map reduce was built on top of a distributed file system and doesn’t fully utilize the faster main memory. All the input files, intermediary files and results are read from and written to this distributed file system, which incurs a lot of IO cost. The tradeoff here is that we get an easier implementation for its fault tolerance, at the cost of the overall performance. Fourthly, debugging, revising and optimizing the query is hard. Because everything is written to the file system and nothing is in the memory for inspection and that one has to wait for the entire previous phase to

finish before the next phase can be kicked off, writing a perfect query can be a very long and tedious process. The fact that multiple map and reduce are often chained only makes writing the query more difficult.

### Spark. Similar to map reduce, Spark is focus on providing a distributed computation engine but does so in a more efficient way and it overcomes some of map reduce limitation by employing a totally different model. It no longer forces programmer to think and write queries in the map and reduce fashion. Rather, one can choose his/her own language and write any ad hoc queries as they want. Spark doesn’t rely on a file system to achieve fault tolerance, instead, it embeds the concept of lineage into its basic building block RDD. A lineage is a piece of metadata that tells how the current RDD can be constructed like where are all the dependencies located and what’ the function to run on the dependent data set to produce this current RDD as well as how a RDD is partitioned. In the case of a partition failure, it needs only rerun the function on the dependency of this particular partition without blocking the rest of the partitions and other RDDs. Spark also realizes that the cost of main memory has been steadily decreasing in the past decade, so it tries to fit all the data into the main memory across machine in the cluster, the spill over goes to the disk but for most of today’s use case, majority of the data can be fit into the main memory when configured correctly. The design of RDD and its reliance on memory instead of disk gives Spark a significant performance boots, in some case one hundred times faster than map reduce for the same query. In addition, spark is much more programmer friendly compared with map reduce. With PySpark, one can write all the queries in python in a Jupyter notebook. This make iterative development and most of the data science workflow much smooth and easy. One can now try some analysis, get a glimpse of what it looks like and then decide what to do next. Caching dataset in the main memory also makes trying different functions and queries much faster by loading the data once. What’s more, besides the set of the most popular data science libraries in Python, Spark has also built on top of its core a rich set of high level tools like streaming, machine learning, graph that makes it the one stop shop for most of the big data analysis. Overall, Spark is a better tool geared towards today’s problem by being high performant, user friendly and proving a rich and easy to use toolbox. We’ve hence decided to use Spark for our data analysis.

## Data visualization

Data visualization is a key part in today’s data analysis because it provides some unique advantages. By presenting information using charts, graph and maps, data visualization tools provides an accessible way to see and understand trends, outliers and patterns, making it an essential part to analyzing massive amount of information and making data-driven decisions. Our eyes and brains are drawn to colors and patterns. The same information presented in plain text and graph could have drastically different effects. While they might convey the same amount of information, we are more prone to see individual records in the plain text versus the overall trends and highlight in a graph. It’s easier for human brain to consume the information in a visual format and make connections of the related data and make more sense of what’s going on. Listed below are some of the visualization tools we considered for our project:

### Matplotlib. This is one of the most stable, rich and popular plotting library in Python. Since most of our data analysis is done in Python, this is the first tool that we considered. It works seamlessly with Pandas and NumPy data structure, making it very easy to use with Python. It’s also highly customizable, making it easy to create complex plots and visualization. The obvious downside is its tight coupling with one specific language, making it very hard for a project to switch over to a different language or even embrace a variety of different languages within a same project. For this very reason, we decided not to use Matplotlib as our visualization tool.

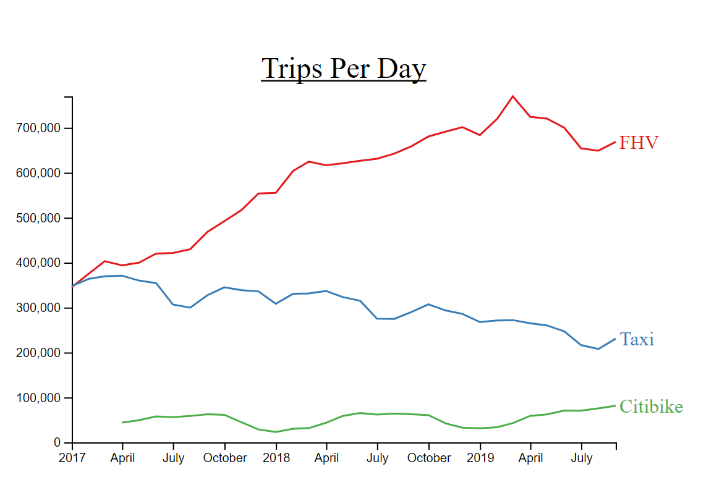
### Tableau. This is a very effective and high level tool to create interactive data visualization very quickly. Designed for both developer and non-developers, it is very user friendly with drag and drop interface. Unlike Matplotlib, it supports multiple language so user won’t get locked into one language. It has very good support for map and geocoding, making it very attractive for our project. However, this product is built and supported by a commercial company. Not only is it not open sourced, it also comes with a hefty price tag for its personal and professional edition. We wanted something that’s accessible for everyone and is well supported by the open source community, so we ended up not using this tool.

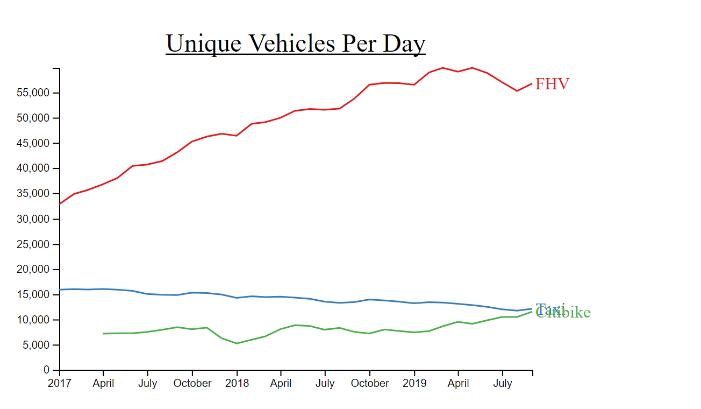
### D3 and related libs. This is an open source JavaScript library. Since JavaScript is the language for the web and is well supported by all modern browsers, visualizations built using d3 can reach the widest range of audience. It works

very well with the existing web stacks like CSS, SVG and HTML with no limitation of any proprietary frameworks. The clear decoupling of data production and data visualization makes it suitable for working with any other backend framework or language. We eventually settled with this tool for our data visualization. We used d3 directly for producing some line charts to show the overall trends and another library called Vega, build on top of D3, to do the interactive map and display the spatial data.

# Experiments

### Overall trend of the Taxi, for hire vehicle and Citi bike. First we want to get the aggregate statistics of the overall usage for Taxi and high volume for hire vehicles like Uber, Lyft and etc. To do that, we downloaded the monthly indicator file from TLC and stored them into the cloud storage in a csv format, we then use PySpark to load them into memory and do the data cleansing. The monthly indicator contains some license types like ‘Black car’, ‘Livery’, ‘Lux Limo’, which are not a typical choice for commuting, so we filtered them out. The monthly indicator also group ‘Yellow Taxi’ and ‘Green Taxi’ into two separate buckets, so we run a map function on the RDD to convert them into the same class. We then group them by time and the newly converted license type and aggregate all the records by summing them up to get the total number of vehicles as well as the total trips for each group for the past 3 years month by month. Then we want to get the aggregate statistics for Citi bike and we have to do it manually from the much more granular trip data since they don’t publish any aggregate. To do that, we first downloaded the past 3 years of trip history for all the Citi bike trips using a script and stored them into cloud storage. We then load them all into PySpark and select only the columns that are of our interest and strip off those unnecessary columns. We did two different group by to get the aggregate of the total number of unique bikes as well as the number of trips incurred. We also append a column to mark that all of these records are for the Citi bike. Finally we take a union of the above two RDD to form a new RDD which contains all of the three types: Taxi, for hire vehicles as well as Citi bike. We plotted them out using D3 to give a very clear picture of how each type is doing compared to each other. Below is a sample of the D3 graph:

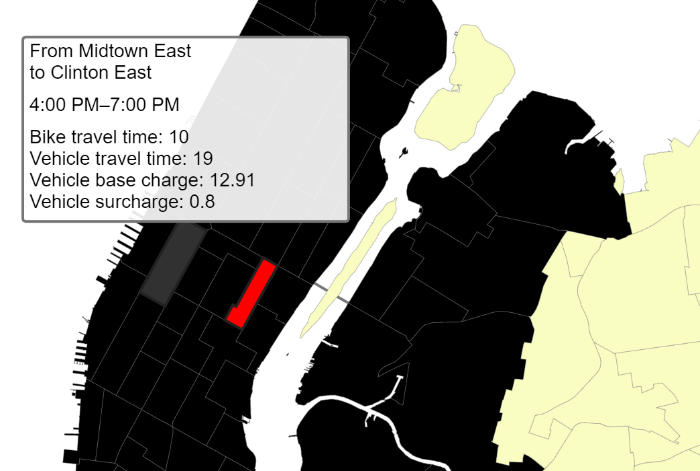
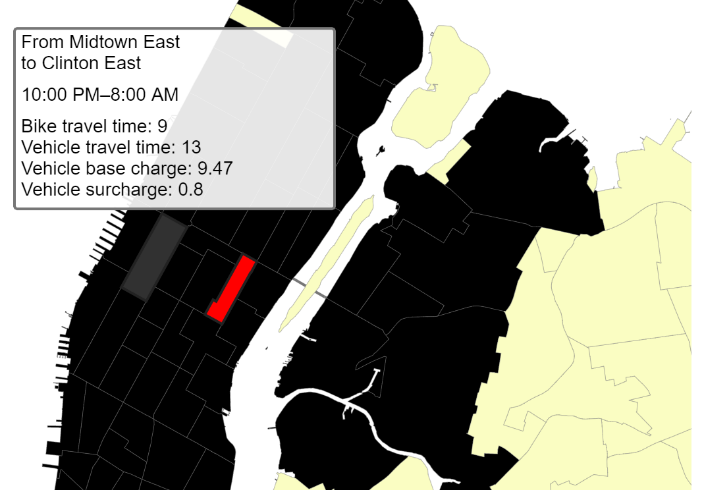




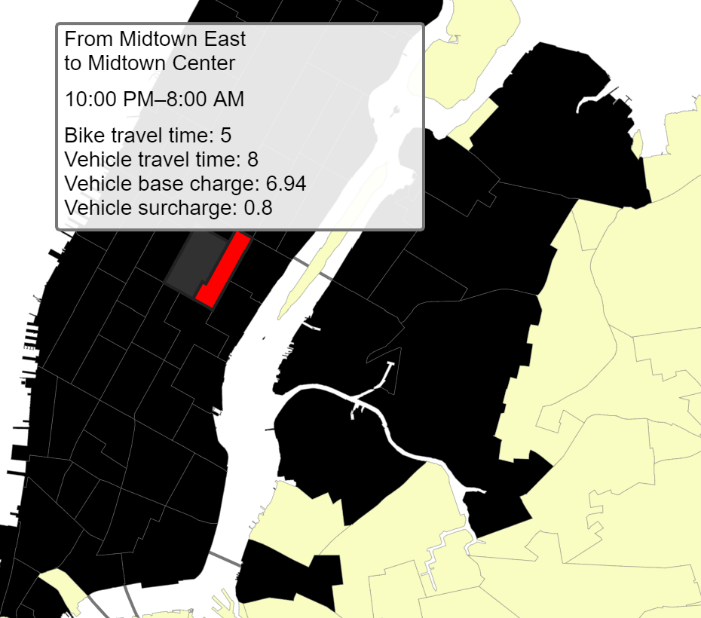
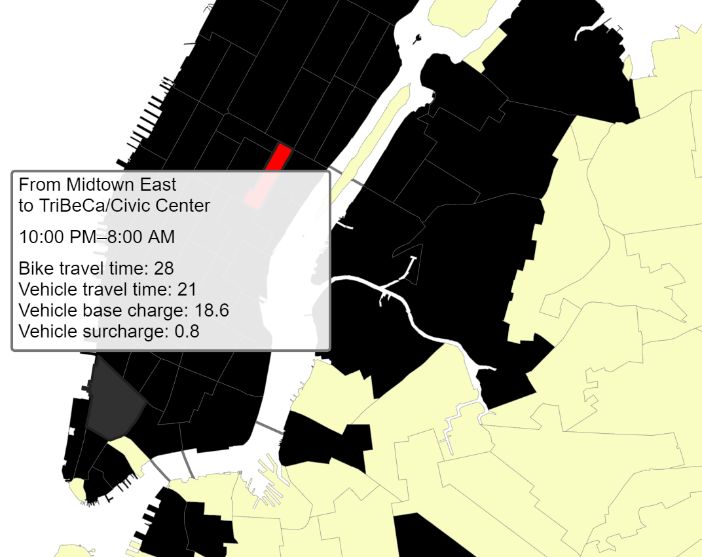
### Estimates of time and money costs from one neighborhood to another using vehicle versus bike. The idea is that we want to build an interactive tool where user can view these spatial data right on the map and when they click or hover the mouse to change their start and end location or the time of the day, these data would change accordingly. To achieve that, we first need to produce all the estimates. We did this by calculating the average time and cost from the past 3 years of data. We downloaded all the trip data from TLC and Citi bike and upload them into two BigQuery tables. For the Taxi trips, we first select the columns like trip duration, trip start and end time, start and end zone etc and leave out the rest, we then convert the timestamp into a time bucket to map each trip into morning rush hour, middle of the day, afternoon rush hour and midnight. Since our objective is to gather insights for the commuting, these buckets make more sense and give us enough data within each bucket to produce some meaningful estimate. The TLC trips are marked by start and end zone and each zone is basically a polygon if we were to draw it out on a map. However, the Citi bike trips are marked by start and end station id, which is a combination of the longitude and latitude of a bike station. This incongruity makes it impossible to compare the two as is. To solve this problem, we leverage the BigQuery GIS feature, where we upload

the NYC Taxi zone file in a geojson format to BigQuery to create a Taxi zone table. We then use the GIS feature to join each trip with this Taxi zone table twice to figure out which taxi zone each bike station corresponds to. Once we have all the Citi bike trips represented in start and end taxi zone, we can then merge this table with the Taxi trips table and group them based on the start, end zone and the time bucket, we then run the average on the aggregate of each group to get an estimate. For the visualization part, we used Vega.js, which is a JavaScript library built on top of D3. It provides some higher level abstraction to make data binding an event processing easier and more flexible. Below are some sample graph we snapped from the interactive map:

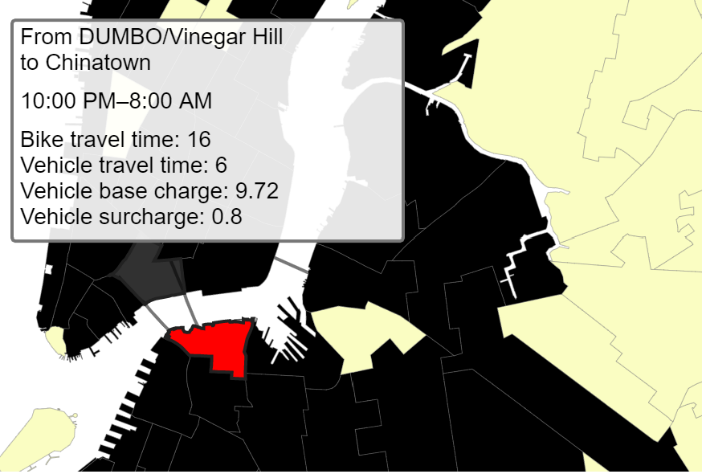
These two graph shows the rush hour impact on travel time for bike and car:



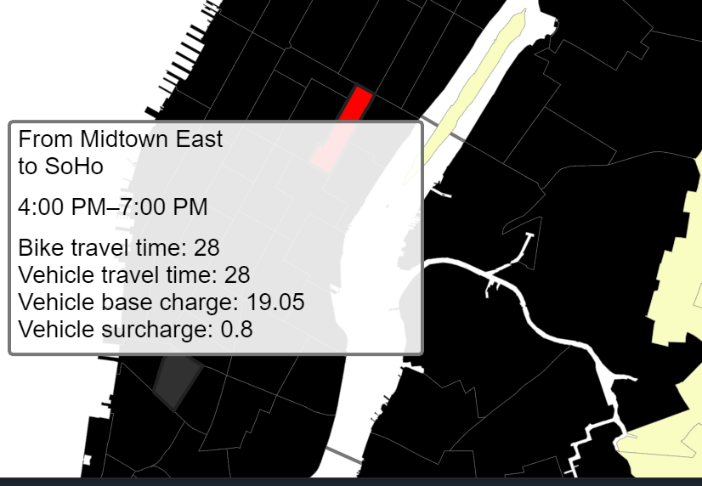
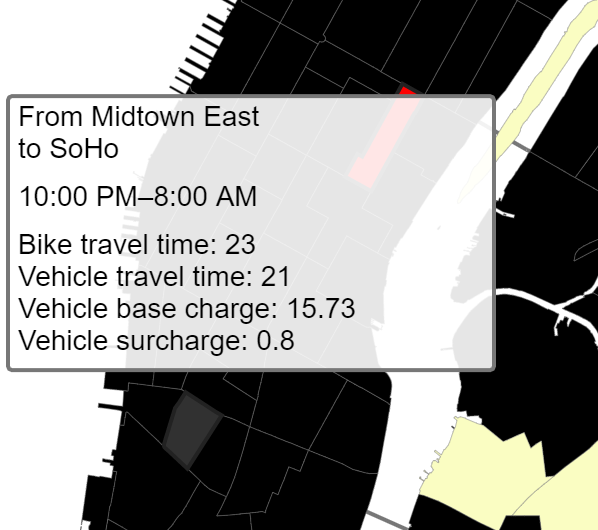
These two graph show the distance impact on travel time



This one shows the cross bridge impact:



And these two shows the fare difference duration and off rush hour:



# System Overview

Most of the tools we use in this project are from Google Cloud platform. The rich offering and on-demand provision of computing and storage resource comes very handy. We used cloud storage to store all the input files, intermediary files as well as the final results. Most of the data cleansing, filtering, joining and aggregation to produce the aggregate report are done in PySpark from DataProc with a cluster of 5 machines. We used Python in Jupyter Notebook to do the iterative and interactive development. For the second part where we built a map to display the spatial data, all the estimate are produced by running standard SQL query with GIS feature support on BigQuery. The web app was done in Django and run locally. All the visualization are done in D3 and Vega.Js. In terms of data processing, both Spark and BigQuery are built with scalability in mind so we can potentially scale it horizontally by adding more resource like memory and CPU. So this likely is not going to be a bottleneck when the data volume gets larger. However, in terms of quality and functionality, there’s some improvements that can be done. Currently all the analysis are produced from historical data without an easy way to ingest current and real time data into the system, which means the analysis are static that doesn’t involve automatically. Overtime these estimates can deviate further from the reality. To address this issue, we can either run a batch job to periodically re-run the analysis on a rolling past 3 year window or a better way would be to build a machine learning pipeline that continuously runs through the historical data set and compare the prediction with the current data as they come in and adjust the model by incorporating the feedback we get from real life.

# Conclusion

Overall, we think each of these three services namely Taxi, for hire vehicle and Citi bike plays a different role. There’s a trend that people are using for hire vehicles and Citi bikes more for their commuting though Taxi still dominates in this space based on the number of trips and number of unique vehicles on the street. With for hire vehicles and Citi bike expanding at the current pace, we expect a gradually diminishing gap between those two and Taxi. On the other hand, if one were to commute from one neighborhood to another in NYC, it’s a good idea to consider Citi bike first for short trips and rush hour trips since Citi bike is both faster and much more budget friendly. However, Taxi and for hire vehicles are still better choices for longer trips and cross bridge trips.