Studying the city of Chicago's bikeshare system using network theory, machine learning and linear programming

Paul Chen | chen3876@purdue.edu
Vivek Rao | rao161@purdue.edu

Introduction

The city of Chicago is one of the most transparent cities in the country. It has a publicly accessible data portal where developers, journalists and those interested can access data related to health code violations in restaurant inspections, property taxes by zip code, street maps and more.

The city's department of transportation began operations for a new bikeshare program in 2013 and quickly racked up riders. The program was a huge success and rideshare company Lyft took over in 2019. The data that powers the system can be publicly accessed via an API, or through monthly data dumps on this link.

For our final project, we decided to analyze data from the third quarter of 2019 using principles we learned in class.

Description of the dataset

The dataset logs information about each individual trip. Here is a sample:

	trip_id	start_time	end_time	bikeid	tripduration	from_station_id	from_station_name	to_station_id	to_station_name	usertype	gender	birthyear
0	23479388	2019-07-01 00:00:27	2019-07-01 00:20:41	3591	1214.0	117	Wilton Ave & Belmont Ave	497	Kimball Ave & Belmont Ave	1	1.0	1992.0
1	23479389	2019-07-01 00:01:16	2019-07-01 00:18:44	5353	1048.0	381	Western Ave & Monroe St	203	Western Ave & 21st St	0	NaN	NaN
2	23479390	2019-07-01 00:01:48	2019-07-01 00:27:42	6180	1554.0	313	Lakeview Ave & Fullerton Pkwy	144	Larrabee St & Webster Ave	0	NaN	NaN
3	23479391	2019-07-01 00:02:07	2019-07-01 00:27:10	5540	1503.0	313	Lakeview Ave & Fullerton Pkwy	144	Larrabee St & Webster Ave	0	NaN	NaN

`trip_id`: an arbitrary (likely sequential) number that uniquely identifies each trip
`start_time`: time when the bike was docked out
`end_time`: time when the bike was docked back in. all times are in Central
`bikeid`: each bike has a unique ID. There are 5,787 unique bikes in this dataset.
`tripduration`: measures duration of each trip in seconds. Some values are extremely high because of docking issues, or because the DoT brought them in for maintenance.
`from_station_xxx`: identifying the station the bike was checked out of
`to_station_xxx`: identifying the station the bike was checked into
`usertype`: a 1 indicates the user is a subscriber and has a pass with Divvy. A 0 indicates the user is using the service for that particular trip.
`birthyear`: this field is not null only when the user is a subscriber

What we learned from this dataset

In our analysis, we wanted to build a narrative around the inventory management process. As of 2014, Divvy had 13,000 daily riders and losing out on a trip is losing revenue for the company. There are also certain nuances when managing inventory for a bikeshare program. For example, more bikes leave residential areas in the morning (as people bike to work) and more bikes leave commercial areas in the evening (as people leave work). Optimal inventory management ensures no station has an excess of bikes because greater bikes mean there's a station out there with fewer bikes and, subsequently, lost revenue.

To kick off this narrative, we focused on the path Divvy takes in the wee hours of the morning on its inventory management rounds.

1. **Traveling through each station exactly once:**
   In our initial idea, we wanted to determine the best way a truck with Divvy bikes could travel through each station and stock the station with bikes before the next morning's rush hour. The best way to do this would be to minimize the number of stations you hit — the lowest number you can do is the exact number of stations there are in the system. Thus, each node would be represented exactly once.

   After conducting research, we learned that this was called the traveling salesman's problem and that networkx had a built-in, greedy function to determine a solution.

   We ran the complete graph through the function and ended with a list of paths/edges between stations where each station was visited exactly once.

2. **Using machine learning to predict the type of consumer:**
   From a business standpoint, it can be extremely lucrative if Divvy could predict which consumers were subscribers and which were one-time users.

   Subscribers are more likely to follow a pattern in use and if big data queries can yield similar insights from non-subscribers, i.e. one-time users, this would lead to cost savings from optimal inventory management and revenue maximization from optimal demand forecasting. Moreover, Divvy can target these one-time users and incentivize them to buy a subscription.

   The predictors we used for this model were gender, birth year and trip duration. However, in the original dataset, the ratio of non-subscribers and subscribers is 30% and 70%. That means even guessing subscribers every time, we can get 70% of the accuracy. To eliminate this problem, we used stratified sampling to get a separate dataset that the

ratio of non-subscribers and subscribers is 50% and 50%. After that we use XGBoost to build our model and get an accuracy of **83.98%** on the testing data**.**

3. **Predicting the number of bikes leaving a station at a particular hour:**
   We built a linear regression model to predict the number of bikes being checked out of Union Station at a user-inputted day and time. While we trained the model on Union Station (Canal St. & Adams St.), it can be easily retrained and generalized on any other station.

   Under this function, Divvy could easily predict the number of bikes that are expected to leave, for example, Broadway and Barry Ave at 8 a.m. on Tuesday and rush to have that number of bikes docked at the station.

   We used two models: one grouped hour as morning, evening, and night, and days as weekends or weekdays; while the other left each value a categorical variable. We found the latter model achieved a much higher R^2 score (63.75%).

4. **Finding the shortest path to every station:**
   We want to provide our customers with some tips to travel around the city. We assume our target customers are tourists. Normally, tourists are chill and won't ride bikes for a long time, so we create a threshold that only edges that are lower than 15 minutes exist. That means we assume most of the customers will only ride for 15 minutes. After filtering out the routes (edges), we can let the customers choose from this station to the destination. How long does it take to get there? And which station they need to go through.

   In order to do that, we create a function with the bellman ford algorithm and let the customers enter the start station. The results will be the time it takes and the routes from the start station to every station.

5. **Using minimum vertex cover via linear programming to find out which node to set up charger station:**
   We want to develop other business models. There is a battery company in Taiwan called the Gogoro Network, it is a modular battery swapping infrastructure designed to be deployed in cities for electric refueling of two-wheel vehicles like scooters and motorcycles. Customers would be able to swap out depleted batteries at a network of kiosks called GoStations for a monthly subscription fee.

   However, setting up those stations is costly. We want to find out only some of the stations that connect all the routes (edges). We use linear programming to build a

minimum vertex cover model to find out how many stations we need to set up the charger stations and which stations.

6. **Utilizing centrality to find the busiest nodes:**
   After we did the minimum vertex cover, we were able to lower down the number of stations to set up. Nevertheless, when we recalculate the cost of setting up the charger stations, we need to further cut down the number of charger stations.

   We use the concepts of centrality, which means the degree of a node is the fraction of nodes it is connected to. We set the threshold to 0.6 to find 25 most busiest stations.