Predicting Shared Dockless Vehicle Time to Activation

MUSA Capstone Mid-Point Draft

Elisabeth Ericson

Introduction and motivation

The initial motivation for this project was my own interest in dockless shared mobility as an emergent urban transportation mode that was virtually unheard of until in the last five years. Dockless scooters introduced to city streets by private companies quickly drew riders, but also backlash; controversy over growth and management has led many cities to impose increasingly onerous restrictions on the scooters' deployment and operation. Simultaneously, scooter advocates point to the potential for shared micromobility to provide a first- and last-mile car alternative for urban neighborhoods poorly served by transit and traditional bikeshare.

I need to connect this introduction to my revised question.

Clear question definition

My initial intent was to study the spatial distribution of dockless bikeshare and scooter trips across urban neighborhoods, particularly in relation to transit accessibility. Unfortunately, the low spatial resolution of published trip-level data made this analysis less interesting: public bikeshare data tends to round origin and destination coordinates to two decimal places, roughly equivalent to rounding to the nearest kilometer. Given that urban neighborhoods one kilometer apart can have vastly different built environments and transit accessibility, I ultimately chose not to pursue this avenue.

Instead, I decided to try to answer a more light-hearted question inspired by my own experiences as a rider of shared dockless vehicles: **What are the chances that a bike or scooter listed as "available" in an app will be taken by someone else before I can get to it?** Or in more rigorous terms: **How can I most effectively model how long an inactive bike or scooter will remain idle between trips?**

Brief literature review

In my initial literature review, I found research supporting the hypothesis that dockless bikeshare improves accessibility in underserved neighborhoods compared to traditional docked bikeshare systems. Analyzing data from 32 U.S. cities with both docked and dockless micromobility, Meng and Brown (2021) found that "the distribution of docked systems is extremely unequal, and that dockless systems greatly reduce geographical inequalities relative to docked." In San Francisco, Qian et al. (2020) found that dockless bikeshare provided greater access to bikes in disadvantaged "Communities of Concern" (CoC) than docked bikeshare did, and also helped mitigate the bikeshare usage gap between CoCs and other communities. Lazarus et al. (2020) found that San Francisco

dockless electric bikes "more heavily serviced lower-density areas" – which tend to have lower transit accessibility – than docked bikes did.

Despite that, inequality may still be present in dockless systems: in a study of dockless bikeshare in the Boston suburbs, Gehrke et al. (2021) found that areas with higher shares of rental housing units and minority residents generated more dockless bike trips, yet had lower access to the bikes.

Given that my research question has shifted from a focus on micromobility travel patterns to a more technical methods question, I should refocus the literature review accordingly. I describe one relevant study in the section on survival analysis below, but there is absolutely more I could cover here.

Technical progress to date

I wrote a proof-of-concept scraper in Python. I developed a simple scraper to query an API endpoint for Washington, D.C.'s Capital Bikeshare system every minute, retrieving real-time dockless bike coordinates in General Bikeshare Feed Specification (GBFS) format.

I collected real-time inactive bicycle coordinates every minute for a day and a half. I ran the scraper for about 38 hours to obtain a preliminary dataset that contained more than 398,000 records in total, with a 24-hour period representing more than 250,000 records. I used this initial dataset to conduct exploratory analysis, identify data quality issues, and develop data processing methods to address them.

I identified two significant technical obstacles to using the raw data. First, bicycle IDs reset every 30 minutes, causing each bicycle to "disappear" from the data after 30 minutes even if it stayed inactive for much longer. Second, the same bicycle ID could represent more than one period of inactivity, if a rider activated it, used it, and deactivated it again within the same half-hour period.

I partially solved both problems. For the second problem, I developed logic and code to identify and assign unique IDs to distinct "instances" of the same bicycle, based on the length of time each bicycle ID "disappeared" from the time series before appearing again. For the first problem, I developed logic to – at least theoretically – match inactive vehicles before and after their IDs reset, in order to measure time to activation beyond 30 minutes.

I identified potential survival analysis models to evaluate, and began researching their implementation in Python. In response to feedback on my project proposal, I researched survival analysis as a potential approach to modeling time to vehicle activation. As the name suggests, survival analysis is used in medical research to model how long a patient or organism is expected to survive a disease process, but it can also be applied more generally to model the expected time until an event occurs.

In simplified terms, I grouped the methods I found into three broad approaches to survival modeling: the **statistical approach**, where the semi-parametric Cox proportional hazards regression seems to be the gold standard; the **machine learning approach**, where survival random forests seem most prominent at first glance, but which also include gradient boosted models and survival support vector machines; and the **deep learning approach**, which includes the Cox proportional hazards neural network DeepSurv (Katzman et al. 2016), among others.

I found well-documented Python implementations of models from each group: Cox regression and several machine learning survival models in scikit-survival, an extension of the scikit-learn machine learning library, and DeepSurv in pycox, a package built around the PyTorch deep learning framework.

I also found at least one example of survival analysis applied specifically to idle time prediction for shared mobility. In a brief conference paper, Kostic et al. (2020) compared the performance of the Cox regression model to the DeepSurv neural network in predicting time-to-pickup for shared cars. The authors found that DeepSurv provided more accurate predictions than the Cox model, and that a "two-step" approach combining classification with survival analysis performed better than survival analysis alone.

I researched and sought advice on extract-transform-load (ETL) pipeline design and implementation for larger-scale data collection. This is complex and not something I've done before, and I've probably spent too long researching pros and cons of different databases, cloud providers, and pipeline designs. My goal in the coming week is to start implementing what I've learned and get a minimum viable scraping pipeline running.

Technical next steps

Set up a scheduler to reliably run the scraper every minute. The proof-of-concept scraper ran on a loop with a fixed 60-second delay, without accounting for the time the code took to run or the risk that a problem with one scrape would delay or prevent the next. My next step is to set up Apache Airflow to allow for more precise and reliable scheduling. (If Airflow proves challenging to get working, there are simpler Python scheduling libraries I could use to get the larger-scale data collection going.)

Refine the scraper code and turn the notebook into a proper script. I ran the initial scraper in a Jupyter Notebook for quick and easy prototyping, but the "production" code would be better suited as a Python script. I also want to rewrite the scraper to collect only the fields I need, fix an issue with the timestamps, and generally clean up the code.

Begin collecting "real" data. Ideally, I'd like to collect data continuously for at least two or three weeks, and for more than one dockless vehicle provider.

Decide how to handle vehicles that disappear from the data, then reappear without apparently having moved. Sometimes an ID disappears for a period of time before reappearing in

the same location, or at least within the bounds of normal GPS error. Without additional information, there is no clear way to determine whether the bicycle was truly unavailable during the time it disappeared (if someone reserved a bike but never used it, or checked out a bike, went for a ride, and then returned it to its original location) or if it simply *appeared* unavailable (if its GPS connection temporarily cut out).

Code the bicycle ID matching logic. I've outlined the steps and written out some rough pseudocode that I'll update this draft to add, but I need to translate it into working Python. If it proves too challenging or time consuming to be practical, survival analysis does have the ability to handle censored data, which in this case would refer to incomplete records where I know a vehicle was inactive for *at least* 30 minutes but can't determine how much longer.

Implement at least one survival model. My best-case scenario would be to implement at least one model from each of the three groups – say, a Cox regression, a survival random forest, and DeepSurv. I would also be curious to see if Kostic et al.'s two-step approach is associated with a similar performance boost for micromobility as for carshare. To begin with, though, developing a Cox proportional hazards model using scikit-survival seems like a reasonable first modeling step.