Scraping Real-Time Shared Micromobility Data to Model Vehicle Time to Activation

Elisabeth Ericson
May 2022

Introduction

As an emergent urban transportation mode, dockless shared mobility was virtually unheard of until in the last five years. Since then, shared electric scooters introduced to city streets by private companies have drawn riders, but also backlash; controversy over growth and management has led many cities to impose increasingly onerous restrictions on the vehicles' deployment and operation. Simultaneously, a growing number of urban bikeshare systems now offers dockless electric bicycles interoperable with their existing docking station infrastructure, and dockless vehicle advocates advocates have pointed to 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.

For this project, I was interested in exploring what insights about the spatial distribution of dockless vehicle usage could be derived from publicly available General Bikeshare Feed Specification (GBFS) data. To that end, I scraped real-time dockless vehicle location data from public APIs for the city of Washington, D.C.; built a pipeline that combined cloud and local computing to extract, transform, and load the data; and created a proof-of-concept model using survival analysis to predict vehicle time to activation at the individual vehicle level.

As this work shows, studying micromobility usage patterns is made difficult by data anonymization measures and inconsistent implementation of the GBFS standard. However, collecting real-time data on inactive vehicles can nonetheless provide valuable information – either directly, with respect to the spatial and temporal distribution of access to shared mobility services, or indirectly, if considering time to activation as a proxy for hyperlocal demand, or lack thereof. In this report, I outline a data collection and processing framework that I hope to build on in future work, and find promise in survival analysis as a method to model shared vehicle time to activation.

Question definition

In effect, this project sought to answer two separate but related questions: First, given that public data on shared dockless vehicle use is limited and deidentified, how can I use high-frequency scraping of real-time inactive vehicle feeds to gain insight into the geography of shared micromobility even in the absence of high-resolution origin-destination trip data? Second, how would I use this data to build a model to predict how long an inactive vehicle will remain idle between trips?

For the first question, I was motivated by technical curiosity and a desire to learn new skills by undertaking a more ambitious data gathering project than anything I'd attempted before. (As it turned out, I had no idea at the outset just how far outside my previous experience it would take me.) The second question is one I've most often seen addressed from the perspective of the operator, whose interest is in efficiently allocating vehicles across space in response to demand, but I was inspired by my own experience as a rider of shared dockless vehicles to instead take the perspective of the user, whose interest is in not missing out on a nearby bike or scooter because someone else got to it first.

Literature review

- studies finding higher access to dockless bikeshare than docked bikeshare in lower-income areas
- that one study finding longer inactivity in lower-income areas
- that one study using survival modeling to predict car share time to activation
- studies using scraped data and which ones can't be done anymore
- that one study about inferring trip origins and destinations given various anonymization techniques

Data

The definitive source of public data on dockless micromobility is the patchwork of real-time feeds published by individual vehicle providers at the city or system level in the format set out by the General Bikeshare Feed Specification (GBFS), an open-source data standard for shared mobility in the spirit of public transit's General Transit Feed Specification (GTFS). The full GBFS standard comprises a collection of JSON file structures, several of which are specific to dock-based bikeshare systems. Most relevant to micromobility is the `free_bike_status` feed, which contains real-time coordinates and select metadata for all currently inactive dockless vehicles.

Feed availability varies by jurisdiction, but my adopted hometown of Washington, D.C. is among a handful of cities that require shared mobility operators to maintain public API endpoints in GBFS format as a condition of their operating permits. As of May 2022, the District of Columbia is home to six dockless vehicle providers: the city's bikeshare system (Capital Bikeshare, operated by Lyft), which offers docking-optional electric bicycles alongside classic docked bikes, and five private companies (Bird, Helbiz, Lime, Lyft, and Spin).

To obtain longitudinal data on how long vehicles remained inactive between trips, I wrote a Python script to query each provider's `free_bike_status` API endpoint every 60 seconds. After an initial trial run using only Capital Bikeshare data, I conducted full-scale data collection for all six D.C. dockless vehicle providers between April 18, 2022 and May 1, 2022.

The common data standard ensured that several essential fields were populated for all providers: a vehicle identifier, the vehicle type (bicycle or scooter), latitude, and longitude. I also recorded the vehicle operator and timestamp. However, different vendor implementations and schema versions meant that fields to track reserved or disabled vehicles were populated in some feeds but not others, and only one provider, Bird, reported vehicles' battery level.

Most significantly, the treatment of unique identifiers varied substantially by vendor. Rotating the vehicle ID each time a trip ends became part of the GBFS standard in January 2020, after contributors to the project raised concerns on GitHub that the ability to reconstruct trip origins and destinations from scraped real-time data risked exposing the movements of domestic abuse survivors, minors accessing reproductive health services, and other vulnerable groups. However, as of May 2022, only one of the six operators whose feeds I scraped appeared to adhere to this standard, with implications I describe under “Data exploration” below.

Pipeline design and infrastructure

Designing the extract-transform-load (ETL) pipeline infrastructure required balancing competing considerations: reliable uptime to run the scraper continuously, without interruptions, every minute for a period of up to several weeks, but also the disk space and computing power to store, process, and model the data without incurring cloud charges beyond the scope of a student project. To meet both needs, I created a split cloud-to-local pipeline to ingest data in the cloud but transfer it to my local environment for processing and analysis.

For the cloud component, I created an Amazon Web Services (AWS) EC2 Micro instance running Ubuntu 20.04. Within it, I built a PostgreSQL 14 database to hold the incoming stream of real-time vehicle data along with error logs, if any. To maintain a single compute and storage environment, and to maximize storage space within AWS's free tier, I chose to build my own database within my EC2 instance rather than use AWS Relational Database Services (RDS) to host a managed PostgreSQL database. I managed the cloud environment via a Secure Shell Protocol (SSH) connection from my desktop.

Still inside the EC2 instance, I ran the scraper as a stand-alone Python script scheduled to query each of the six vendors' `free_bike_status` API endpoint every minute at :00, parse the retrieved JSON data, and write the relevant fields to a database table. Error handling within the scraper script ensured that any failed scrapes would log the timestamp, API provider, and traceback details to a separate database table.

Because a single scrape contained nearly 10,000 unique records, amounting to around 1.4 million new table rows per day, I partitioned the primary `vehicles` table by timestamp range, with a new partition for each day scraped. Partitioning the table also facilitated transferring data between the cloud and local environments without interrupting the scraping process.

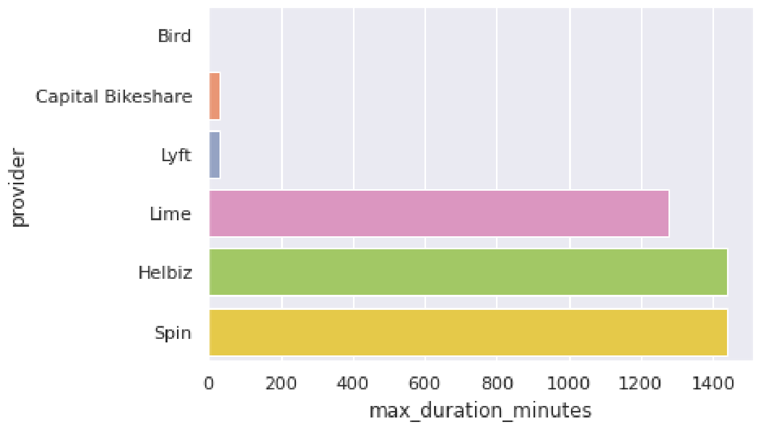
In my local desktop environment, also running Ubuntu 20.04, I built a second PostgreSQL 14 database with a schema mirroring that of the cloud database. In the local version, I also installed the PostGIS extension to allow spatial manipulation and querying directly in the database, which my research suggested would be more performant than Python's GeoPandas for spatial operations on tens of millions of records.

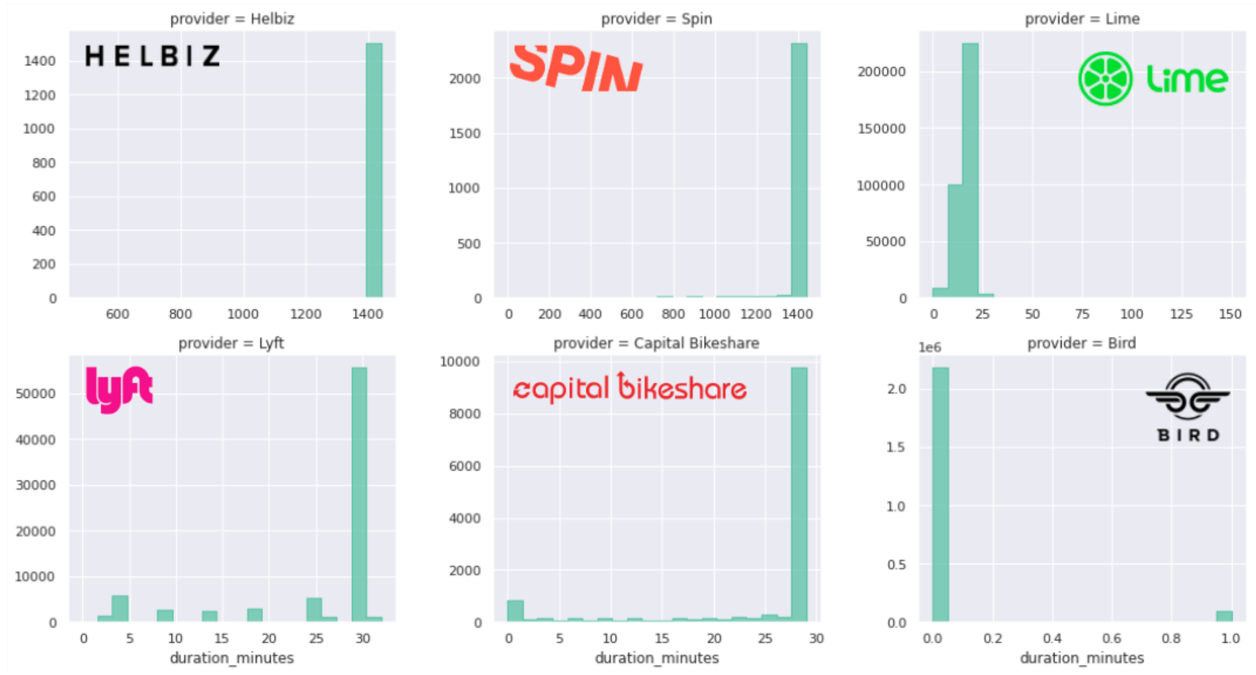
To transfer data between the two environments, I detached the oldest complete partition from the cloud database `vehicles` table, then used secure copy protocol (`scp`) to transfer the data from the EC2 instance to my desktop, where I attached it as a new partition to the local database's `vehicles` table.

Lastly, to conduct further analysis and modeling in Python, I used the Psycopg 3.0 Python package to connect to and query the database from within a Jupyter Notebook. For performance reasons, I kept data manipulation and aggregation within the SQL query to the extent possible, using both PostgreSQL and PostGIS functions, before moving to Python for exploratory analysis, visualization, and model development.

Data exploration

- by provider
- mention both 'hard problems' and their conceptual solutions, but leave actual implementation for 'later work'
    - also note eugene's paper??
- for lime specifically





Company logos for illustrative purposes only. This is an academic project undertaken solely for educational purposes, and I have no affiliation with the above-mentioned entities.

- Visualizations/exploratory
  - Map distribution of available vehicles by vendor
    - How to handle time?
    - Include map of priority/underserved areas?
      - Dockless Equity Emphasis Areas: https://opendata.dc.gov/datasets/DCGIS::dockless-equity-emphasis-areas/
  - Spatial autocorrelation?
  - Histogram of time to activation
  - Vehicles per capita by neighborhood?
    - By priority vs. non-priority area?
  - Correlate availability with income?
  - Correlate time to activation with availability/saturation?

Survival modeling to predict time to activation

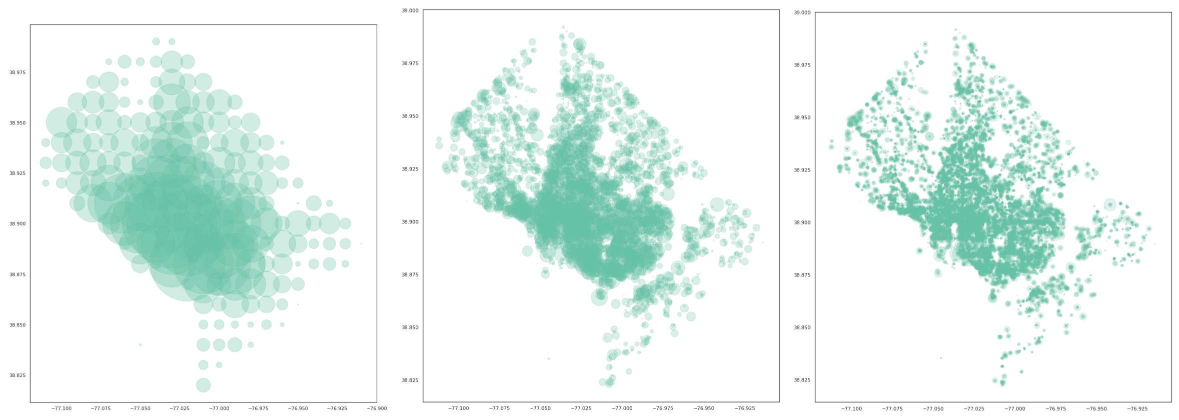
Much more than I had intended, this became a data engineering project with a brief note on predictive modeling, rather than a modeling project per se. As such, this section should be read as an early exploration of survival analysis as a modeling framework to predict shared dockless vehicle time to activation, with the bulk of the model development work yet to come.

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. An advantage of survival analysis is that it allows for the inclusion of so-called censored data, which refers to data whose status is known until a certain point, but unknown thereafter.

In my review, I found 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.

- Kaplan-Meier estimators

I used the scikit-survival Python package to implement a proof-of-concept Cox proportional hazards regression model using only the simplest spatial, temporal, and vehicle features: point **location** rounded to two decimal places, a spatial resolution of approximately 1.1 kilometers; **day** of the week; **hour** of the day; and **vehicle type**. I chose these features for expediency far more than efficacy, and improving spatial precision would be my first priority for the next iteration of this model.



I limited the model to data from Lime, the only operator that appeared to meet the GBFS recommendation to assign a new ID each time a vehicle was parked, but maintain a consistent ID for the duration of inactivity. I coded vehicles as right-censored — that is, with an unknown activation time — if they were present at the last timestamp in the dataset, beyond which their status was unknown.

Lastly, I limited the model to vehicles that remained in a single location, reported to four decimal places (a precision of approximately 11 meters). I was unable to immediately determine why some Lime vehicle IDs appeared in more than one location, but speculated that rebalancing activity — where the company relocates inactive vehicles in response to demand — could cause a vehicle to change locations without triggering an ID reset in the way a trip would. If true, and if the likelihood of rebalancing is tied to the length of inactivity, this has the potential to introduce selection bias into the model by omitting vehicles with long idle periods parked in low-demand locations. In theory, this might be at least partially addressed by treating each vehicle position as a separate vehicle, and coding the initial instance as right-censored.

. . …

Results

- Map of actual vs predicted?
  - Scatterplot of actual vs predicted
- Map of model error
- Where did the model perform well/poorly?

Limitations

Future work

Conclusion

References