Predicting Shared Dockless Vehicle Time to Activation

Elisabeth Ericson

MUSA Capstone 2022

The topic: Dockless micromobility

- ➔ New transportation mode; emerged in last several years
- ➔ Potential to address first- and last-mile problem for urban neighborhoods poorly served by transit and traditional bikeshare
- ➔ Controversy over growth and management has led to regulation, onerous restrictions

I considered a range of very serious questions…

- ➔ Does dockless micromobility improve accessibility for neighborhoods poorly served by transit and traditional bikeshare?
- ➔ Can cluster analysis of origin-destination data tell us anything interesting about where people travel using dockless modes?
- ➔ What neighborhood characteristics predict high rates of dockless bikeshare adoption relative to traditional bikeshare?
- ➔ How do micromobility providers compare on equity? What companies are best and worst at expanding access to low-income or non-white neighborhoods?
- ➔ Is it possible to infer precise trip start and end coordinates by combining anonymized trip history data with real-time inactive bike location data?
- ➔ How does deep learning for origin-destination demand prediction work, and how could I adapt it to this data?

…but decided to go with something more lighthearted

addressing a problem that, if you’ve used scooters or bikeshare, you might be able to relate to

What are the chances that a bike or scooter listed as “available” in the app will be taken before I can get to it?

How can I most effectively model how long an inactive bike or scooter will remain idle between trips?

The premise

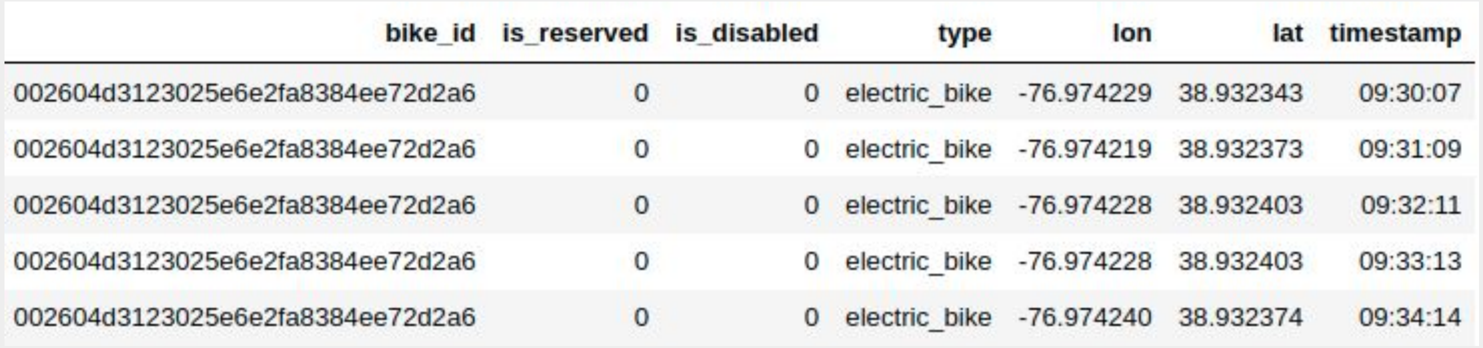
- ➔ Build a model to predict how much longer an inactive bike or scooter is likely to remain available
- ~~➔ Build a simple web app where a user can enter their current location, select a nearby bike or scooter, and see how likely it is to still be available when they get there~~

Bike & scooter location data

- ➔ Real-time API publishes coordinates for all inactive vehicles
- ➔ Standardized format: General Bikeshare Feed Specification (GBFS)
- ➔ Wrote demo Python scraper and collected location data every 60 seconds for 24 hours

Complications: Two hard problems

- ➔ **Same bike, different IDs:** Bicycle IDs reset every 30 minutes
- ➔ **Same ID, different instance:** One ID can represent more than one period of inactivity (if someone activates a bike, rides it, and deactivates it again)

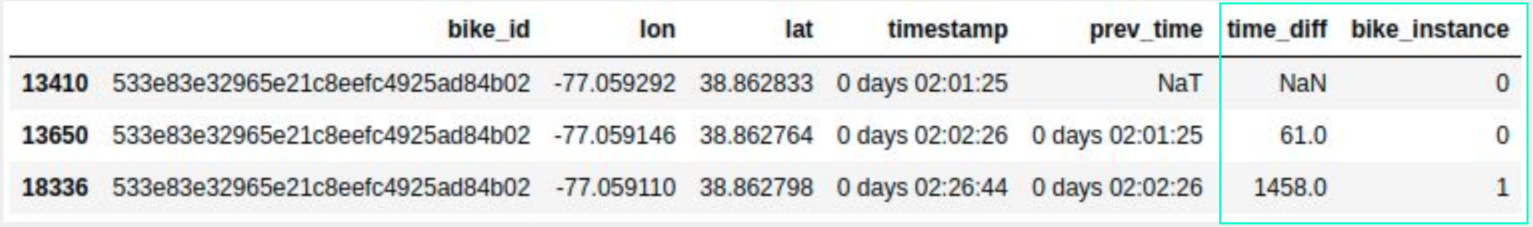


bike_id	is_reserved	is_disabled	type	lon	lat	timestamp
002604d3123025e6e2fa8384ee72d2a6	0	0	electric_bike	-76.974229	38.932343	09:30:07
002604d3123025e6e2fa8384ee72d2a6	0	0	electric_bike	-76.974219	38.932373	09:31:09
002604d3123025e6e2fa8384ee72d2a6	0	0	electric_bike	-76.974228	38.932403	09:32:11
002604d3123025e6e2fa8384ee72d2a6	0	0	electric_bike	-76.974228	38.932403	09:33:13
002604d3123025e6e2fa8384ee72d2a6	0	0	electric_bike	-76.974240	38.932374	09:34:14

Partially solved both problems

➔ **Same ID, different instance:** Developed logic and code to identify and assign unique IDs

- for each bike record, identify previous most recent timestamp for same ID; calculate time difference between records; if >1 min, increment ID counter

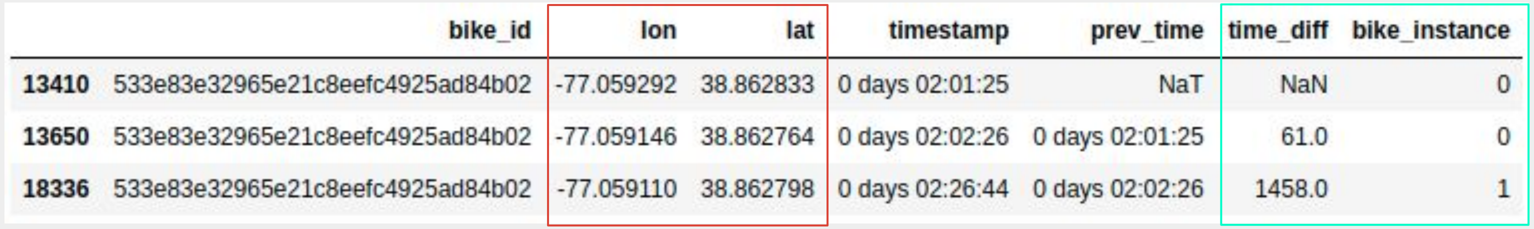


	bike_id	lon	lat	timestamp	prev_time	time_diff	bike_instance
13410	533e83e32965e21c8eefc4925ad84b02	-77.059292	38.862833	0 days 02:01:25	NaT	NaN	0
13650	533e83e32965e21c8eefc4925ad84b02	-77.059146	38.862764	0 days 02:02:26	0 days 02:01:25	61.0	0
18336	533e83e32965e21c8eefc4925ad84b02	-77.059110	38.862798	0 days 02:26:44	0 days 02:02:26	1458.0	1

Partially solved both problems

➔ **Same ID, different instance:** Developed logic and code to identify and assign unique IDs

- for each bike record, identify previous most recent timestamp for same ID; calculate time difference between records; if >1 min, increment ID counter



	bike_id	lon	lat	timestamp	prev_time	time_diff	bike_instance
13410	533e83e32965e21c8eefc4925ad84b02	-77.059292	38.862833	0 days 02:01:25	NaT	NaN	0
13650	533e83e32965e21c8eefc4925ad84b02	-77.059146	38.862764	0 days 02:02:26	0 days 02:01:25	61.0	0
18336	533e83e32965e21c8eefc4925ad84b02	-77.059110	38.862798	0 days 02:26:44	0 days 02:02:26	1458.0	1

Partially solved both problems

- ➔ **Same bike, different IDs:** Developed logic to (theoretically!) match inactive vehicles before and after IDs reset
  - for each pair of minutes before and after ID reset (:59/:00 or :29/:30),
    find nearest neighbor with <10m max distance; eliminate duplicates by distance

Modeling: Survival analysis

- ➔ Also known as “time-to-event” analysis
- ➔ Used in medical research to model life expectancy, but can be used to model time until any event occurs
- ➔ Several approaches and models
  - **Statistical:** Cox proportional hazards regression
  - **Machine learning**: Survival random forests, gradient-boosting models, survival SVMs
  - **Deep learning**: e.g. DeepSurv
- ➔ Python implementations in **scikit-survival** and **pycox**
- ➔ Kostic et al. (2012) used survival analysis to predict time-to-pickup for shared cars; found DeepSurv > Cox regression, and classification + survival analysis > survival analysis alone

ETL and infrastructure

- ➔ Lots of research into ETL pipelines, scheduling, and other infrastructure
- ➔ …I don’t really want to talk about it

Next steps

- ➔ Refine scraper, set up pipeline, and begin collecting “real” data
- ➔ Decide how to handle vehicles that “disappear” and “reappear” in same location
- ➔ Try to code ID matching logic; if too hard, use censored data instead
- ➔ Implement at least one survival model
- ➔ Add features and data sources to the model

Questions?