THE UNIVERSITY OF CHICAGO

Bus pending

Final project for CAPP 30122

Michael Rosenbaum (mrosenbaum) Regina Isabel Medina Rosales (rmedina)
Daniel Muñoz (dmunozbatista) Keling Yue (keling)

Winter 2024

The full code for this project is available in the github repo.

1. Abstract

We visualize inequality in transit delays in Chicago based on CTA's data on bus positions. The Chicago Transit Authority (CTA) regularly reports on bus positions and bus schedules in the city of Chicago. Our project scrapes these reports on a regular interval of one minute to gather bus and train locations and then calculate which stops are regularly delayed. The initial intention for the analysis, was to merge the bus locations with publicly available data on community areas to calculate which areas regularly experience delays. Given the complexity of defining a standardized definition of "delay" for every bus stop at different moments of the day and days of the week, wer where not able to cross-reference bus delays with sociodemografic information in the period of this project. We did manage to visualize which routes experience the most delays and identified that routes crossing from south-to-north face common delays. These data are displayed using a Dash application. The `bus_pending` package also contains resources to schedule and request location data to continue the project. The data linked here represents one week of bus positions equivalent to more than 4.5 million bus minutes, from February 22nd to 28th, 2024.

2. Data Sources

2.1. Bus data

We collect data on bus positions and expected positions from the Chicago Transit Authority's public data. These data report high-frequency current data and schedules, but do not contain historical data.

- CTA bus locations The CTA bus tracker API provides bus locations each minute on their bus tracker API.
- CTA schedule: The CTA provides data in the General Transit Feed Specification (GTFS) that describes scheduled routes.

2.2. GeoData

We display these data on maps constructured using public data from the City of Chicago and the Chicago Transit Authority. These shapefiles are used to form the base of our visualizations:

- CTA Route GeoData: CTA routes are publicly accessible as shapefiles on the City of Chicago's Open Data portal.
- Chicago Community Area GeoData: Chicago community area boundaries are publicly accessible as shapefiles on the City of Chicago's Open Data portal.

2.3. Sociodemographic data

Finally, sociodemographic data on income is added to the map to contextualize where transit delays occur. We draw all income data from the US Census Bureau:

- American Community Survey: We use block-level income and poverty measures from the American Community Survey for our map backgrounds.

3. How to use the software

Bus Pending operates in two modes

1. Visualization: the application cleans scraped data, creates summary measures, and then passes that information to a Dash application to visualize on your local machine. We link one week of bus locations for use here.
2. Scraping: the application creates a database of current bus schedules and bus locations. We include a shell script (`schedule.sh`) to automate calls to the CTA API for bus locations.

We describe details on how to run both components below:

3.1 - Visualization

The Bus Pending project can be run from a machine with Python installed.

1. Clone the Bus Pending Github Repository to your local machine by running `git clone 'https://github.com/RMedina19/bus_pending.git'` to you preferred directory.
2. Run `poetry install` from the top-level directory '/bus-pending' to set up the working environment in Python. Note: this requires installing Python and Poetry, a Python package manager.
3. Download our database of bus positions (826 mb) and place it in a data directory '/bus-pending/data' or scrape data yourself. See below for details on scraping data yourself.
4. Clean and analyze data by running `poetry run python -m bus-pending`
5. Launch the dashboard by changing the working directory to `cd bus_pending/app/` and then running `poetry run python ./app.py`.

- Since the repo contains a copy of the summary statistics used for the plots, this step can be run on its own without the need of running the scrapper or the cleaning and analysis.
- For the `pydeck` plots to render correctly, it is necessary to have a Mapbox API key. The user needs to store their key as a text file named `.apikey` in the directory `bus_pending/app/pages/`. If a key is not provided the bus geodata will render without a map underneath.

3.2 - Scraping

Requesting data from the CTA requires an API key. A key can be requested for free.

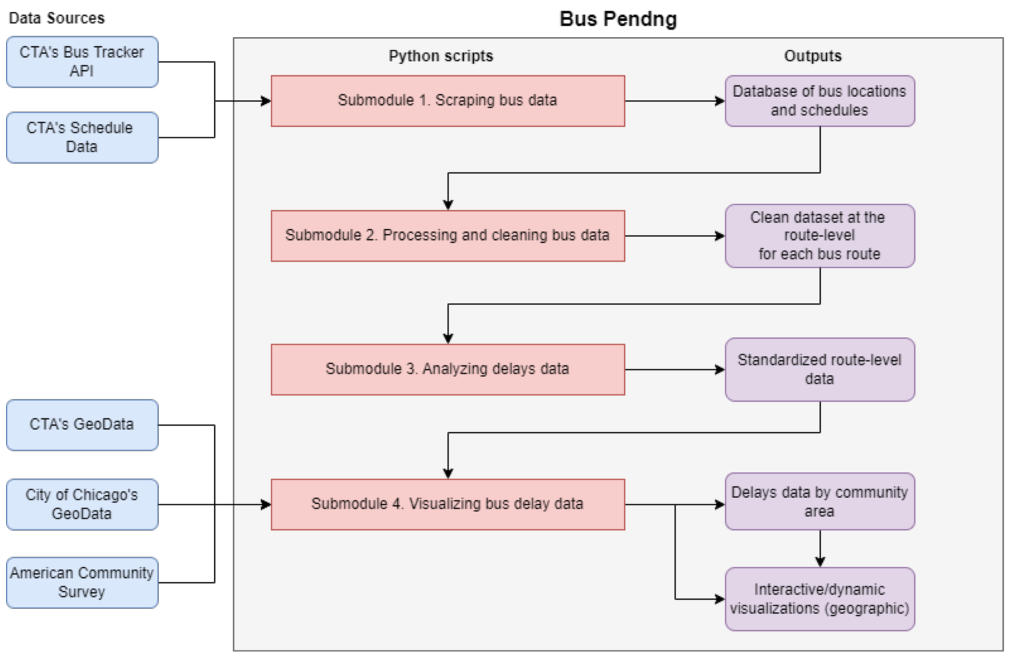
Once you have an API key, we include a shell script to regularly scrape data from the CTA. Although we scraped bus locations each minute, this will exceed the rate limit of 10,000 requests per day, as each minute requires 13 requests to the CTA API for 18,740 requests per day. To scrape the script, do the following once you have the working environment set-up as suggested in Instructions:

1. Put your working directory into the bus_pending file by running `cd/bus_pending/bus_pending`.
2. Create a database using `poetry run python3 -m scraping -makedb`. This will scrape the scheduled routes and provide users the ability to select which components to scrape.
3. Run each iteration of the scraper by running `poetry run python3 -m scraping -quiet`. The –quiet option automatically loads the database if /data/buses.db does not exist and then downloads the full list of routes from the CTA API to scrape without requiring user input.

We provide a shell script `schedule.sh`, in the `./bus-pending/` folder to automate this process.

4. Application Structure

We structure the data workflow in two parts, aligning with the scraping and visualization described in Section 3. This data flow is visualized below:


Data Sources
Bus Pendng
CTA's Bus Tracker API
CTA's Schedule Data
CTA's GeoData
City of Chicago's GeoData
American Community Survey
Python scripts
Outputs
Submodule 1. Scraping bus data
Submodule 2. Processing and cleaning bus data
Submodule 3. Analyzing delays data
Submodule 4. Visualizing bus delay data
Database of bus locations and schedules
Clean dataset at the route-level for each bus route
Standardized route-level data
Delays data by community area
Interactive/dynamic visualizations (geographic)

These lead to a structure:

```
bus_pending/
    analysis/
        analysis.py
        analyze_real_data.py
        compare_schedule_real_data.py
    app/
        app.py
        pages/
          .apikey
          deck_bus_grid.py
          deck_trips.py
          home.py
          plotly_income.py
          plotly_summary_stats.py
    cleaning/
        clean.py
    scraping/
        __main__.py
        make_db.py
        scrape_buses.py
        scrape_routes.py
        scrape_schedules.py
    visualizations/
        acs_data/
        geodata/
        scraped_data/
        preprocessing_for_plotly.py
        preprocessing_for_pydeck.py
    __init__.py
    __main__.py
    shcedule.sh
data/
    instructions.md
    buses.db
tests/
.apikey
routes.txt
```

4.1 - Scraping Data Structure

The scraping elements are a standalone data workflow, meant to be collected once for visualization. These scripts conduct three tasks:

1. `scrape_routes.py` requests and store a list of routes locally from the CTA Bus Tracker API for future requests.
2. `make_db.py` creates a sqlite3 database in an ignored folder `./data` for schedule and bus location data.
3. `scrape_schedules.py` downloads GTFS data and load schedule and helper data into the database to produce expected transit delays. This is approximately ~250 MB of data and takes a few minutes.
4. `scrape_buses.py` scrapes bus location data from the CTA Bus Tracker API.

For projects that want to scrape delay data, (4) can be conducted regularly using scheduled calls to the schedule.sh shell script. The standard API key will allow calls every other minute.

4.2 - Data Processing Structure

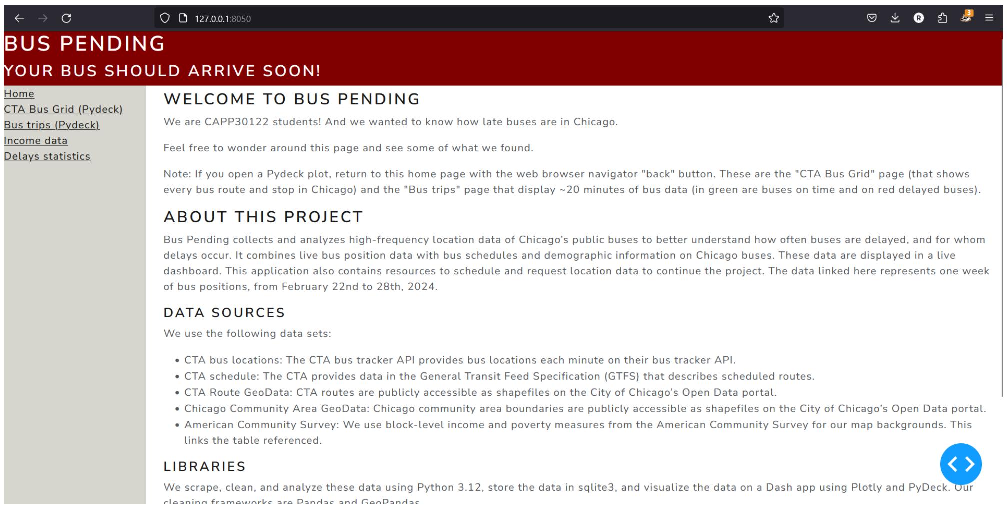
The visualization application is the primary component. Once data are downloaded, it:

1. `clean.py` queries the `./data/buses.db` database to extract bus-minute-level data and returns a route-level dataset with relevant columns as a CSV in the ignored `/data/` folder.
2. `analyze_real_data.py` and `analysis.py` combine the route-level data with expected timing data from the `buses.db` data and determines delays.
3. The `preprocessing_for_plotly.py` and `preprocessing_for_pydeck.py` take the income, scraped and shapefiles raw data and converts it to the formats that are correctly parsed by the Plotly and Pydeck visualization packages. For instance, the visualization of the trips in Chicago uses `Pydeck`'s `PathLayer`. For the layer to parse the scraped data correctly, each trip set of observations had to be collapsed into a single row, CTA's geometries had to be transformed to a list of longitude-latitude pairs, and the time stamps had to be converted to unix standard.

4.3 - Dashboard Structure

The visual outputs of the project are presented in a Dash application with interactive visualizations. The app can be launched without the need of running the scrapper nor the data processing. Because it follows a multi-page design from Dash, the app is built by five "pages" python scripts that are managed and called by `app.py`.

Examples of the visual output included in the Dash App are presented in the following images:



BUS PENDING

YOUR BUS SHOULD ARRIVE SOON!

WELCOME TO BUS PENDING

We are CAPP30122 students! And we wanted to know how late buses are in Chicago.

Feel free to wonder around this page and see some of what we found.

Note: If you open a Pydeck plot, return to this home page with the web browser navigator "back" button. These are the "CTA Bus Grid" page (that shows every bus route and stop in Chicago) and the "Bus trips" page that display ~20 minutes of bus data (in green are buses on time and on red delayed buses).

ABOUT THIS PROJECT

Bus Pending collects and analyzes high-frequency location data of Chicago's public buses to better understand how often buses are delayed, and for whom delays occur. It combines live bus position data with bus schedules and demographic information on Chicago buses. These data are displayed in a live dashboard. This application also contains resources to schedule and request location data to continue the project. The data linked here represents one week of bus positions, from February 22nd to 28th, 2024.

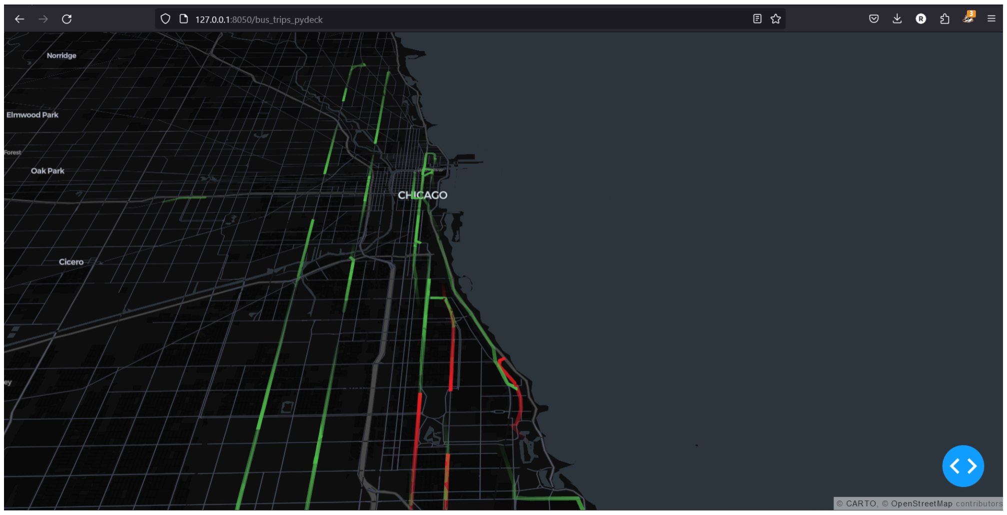
DATA SOURCES

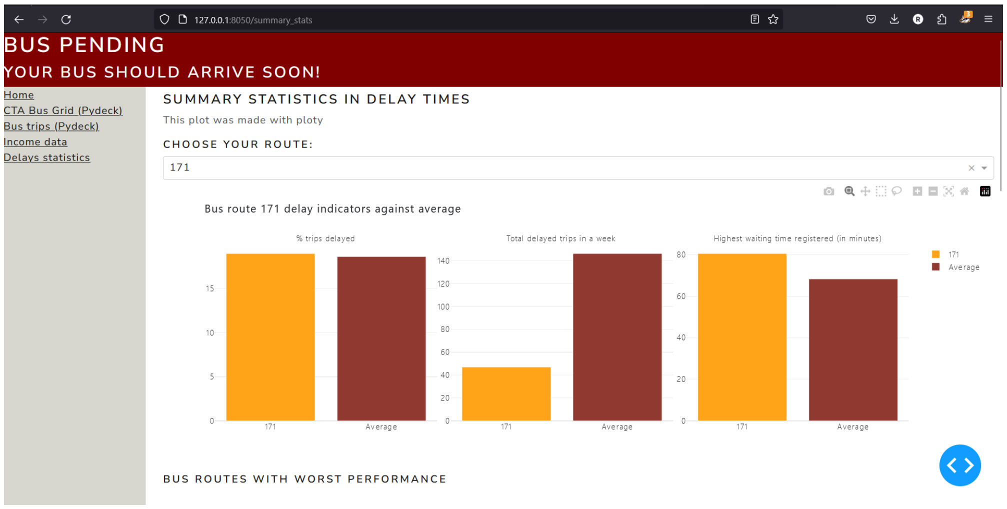
We use the following data sets:

- CTA bus locations: The CTA bus tracker API provides bus locations each minute on their bus tracker API.
- CTA schedule: The CTA provides data in the General Transit Feed Specification (GTFS) that describes scheduled routes.
- CTA Route GeoData: CTA routes are publicly accessible as shapefiles on the City of Chicago's Open Data portal.
- Chicago Community Area GeoData: Chicago community area boundaries are publicly accessible as shapefiles on the City of Chicago's Open Data portal.
- American Community Survey: We use block-level income and poverty measures from the American Community Survey for our map backgrounds. This links the table referenced.

LIBRARIES

We scrape, clean, and analyze these data using Python 3.12, store the data in sqlite3, and visualize the data on a Dash app using Plotly and PyDeck. Our cleaning frameworks are Pandas and GeoPandas


127.0.0.1:8050/bus_trips_pydeck
Norridge
Elmwood Park
Oak Park
CHICAGO
Cicero
© CARTO, © OpenStreetMap contributors


127.0.0.1:8050/summary_stats

BUS PENDING

YOUR BUS SHOULD ARRIVE SOON!

Home
CTA Bus Grid (Pydeck)
Bus trips (Pydeck)
Income data
Delays statistics

SUMMARY STATISTICS IN DELAY TIMES

This plot was made with ploty

CHOOSE YOUR ROUTE:

171

Bus route 171 delay indicators against average
% trips delayed
Total delayed trips in a week
Highest waiting time registered (in minutes)
171
Average

BUS ROUTES WITH WORST PERFORMANCE

5. Members and Responsibilities

Our entire team jointly decided on the idea and approach of analyzing public transit delays. Work on software design and interfaces was decided as a group, including confirming which data would be used and the final units of analysis.

We split our coding responsibilities by each component of the application described above.

Author	GitHub Link	Lead
Michael Rosenbaum	GitHub	Scraping
Keling Yue	GitHub	Cleaning
Daniel Muñoz	GitHub	Analysis
Regina Isabel Medina	GitHub	Visualization and Server

These areas directly map to subfolders in the `./bus_pending` folder:

- Michael wrote all scripts in the `scraping/` subfolder and the schedule.sh script
- Keling wrote all scripts in the `cleaning/` subfolder
- Daniel wrote all scripts in the `analysis/` subfolder, including suggesting analytical decisions such as sample restrictions and pystest files in tests folder.
- Regina wrote all scripts in the `visualizations` and `app` subfolders.

Tests were written for each section as needed. Due to the nature of scraping and visualizing data, we chose to not include automated testing of scraping and visualization scripts, instead reviewing each product as it was produced for quality.

Finally, general documentation was decided on jointly and written by Michael.

6. Conclusions and changes in Process

We initially set out to map administrative burden experienced in Chicago neighborhoods using train and bus access data to measure variation in wait time due to public transport delays. This was an ambitious goal. We accomplished a significant portion of that goal with this application, collecting one week of high-frequency data on bus positions that can be used to measure delays and then analyzing delays by route, and therefore location.

Our approach evolved based on some technical challenges and some focus changes based on interrogating the data more deeply. On the technical front, the CTA Bus Tracker API does not include the next stop, so we realized that we would need to determine when each bus passed each stop based on locating the next most likely stop on a route. This was not feasible given the time constraints of the course. In addition, reported bus data has a number of artifacts such as long duration routes, and short-length routes that suggest CTA bus reporting may not produce accurate data on bus delays at a granular enough level.

Instead, we focused on the route-level, determining how delays accrued over the course of the route. This data allowed us to better describe which buses experience delays regularly, and to plot those delays. Unsurprisingly, these are long local buses including the 4 (King Dr) and 8 (Halsted Ave) buses, which traverse the South Side of Chicago on the North/South axis to downtown.

We encourage you to explore the interactive plots, and better understand how often your local routes are delayed.