Direct train connections dashboard

Yelin Zhang

Data visualization final project report - HS23
SBB GTFS Timetable Data

Source Code: https://github.com/Yelinz/commute-city-triangulation

Motivation

Selecting a new place to call home is a significant decision, one that influences our daily routines, well-being, and overall quality of life. Among the myriad factors to consider, the accessibility and convenience of public transit, but especially direct train connections between target destinations hold paramount importance. Efficient transportation can save valuable time, reduce stress, and contribute to a seamless daily experience.
This is a problem I faced when trying to find a new place to call home. Because using Google Maps or the SBB website for such a task is very cumbersome. As well as there being no existing application to do this task. The desire was to find a location which has a direct train connection to two different destinations, such as school and work. This would be achieved by putting in the desired destinations and then being shown potential origin stations.

Target Audience

The target audience of this data dashboard is individuals who rely on train commuting as a regular mode of transportation and are interested in optimizing their daily travel routes. Whether you're a daily commuter, a frequent traveler, or someone considering a change in residence, this dashboard is designed to provide valuable insights and information. It caters to those seeking to make informed decisions about their daily commute, helping them identify convenient locations, assess travel times, and explore potential changes to their current living arrangements in order to streamline their train-based daily transportation. This audience is looking for data-driven guidance to improve their commuting experience and make more informed choices about their daily routes and residential decisions.

Exploration

Before starting the Project, I was looking around for some tools which already exist with the desired functionality. The problem was there were some similar tools on websites such as https://www.chronotrains.com/ or https://direkt.bahn.guru but they are only for checking one station at a time. Which is not suitable for the task of finding an origin location for multiple destinations. The SBB also provides a map (https://network.sbb.ch/de/) for all their InterCity and InterRegio lines, but excludes a lot of locations which are only served by regional lines such as S-Bahns. Therefore I decided to use the timetable data provided by SBB to find overlapping stations from different destinations. The timetable includes all lines in the entirety of Switzerland.

GTFS Terminology

- Route: A train line e.g. IC1 Geneva - Zürich - St. Gallen.
- Trip: One occasion of a route e.g. IC1 on Monday at 8:17 starting from Geneva.
- Stop time: Every stop which the train serves on a given trip.
- Stop: A train station or platform in the network.

Most of the exploration for the project was reading the GTFS documentation and interpreting it. One of the problems is that the provided GTFS data does not completely fulfill the

specifications and that it contains a lot of data which is just wrong. Which means the routes do not exist in reality or stops which are in the data but not served in reality. To solve this problem I implemented a menu to be able to exclude some routes which are irrelevant or do not exist.

Stops in the data are a mix of the platforms of a train station and the train station itself, this makes working with them a little more difficult, as that information has to be converted back and forth in some processing steps.

Normally there is a shape table in GTFS which the SBB does not provide. It would contain the real shape which the train takes on a given route. Therefore I have to extrapolate where the routes go through based on the stations the route goes through. This approach has a disadvantage for routes which have few stops which are far apart. It causes the lines to be drawn in areas where the train does not go through, such as straight through lakes. Solving this problem would be very difficult, as I would have to scrape OpenStreetMaps for train track information and draw the routes based on that.

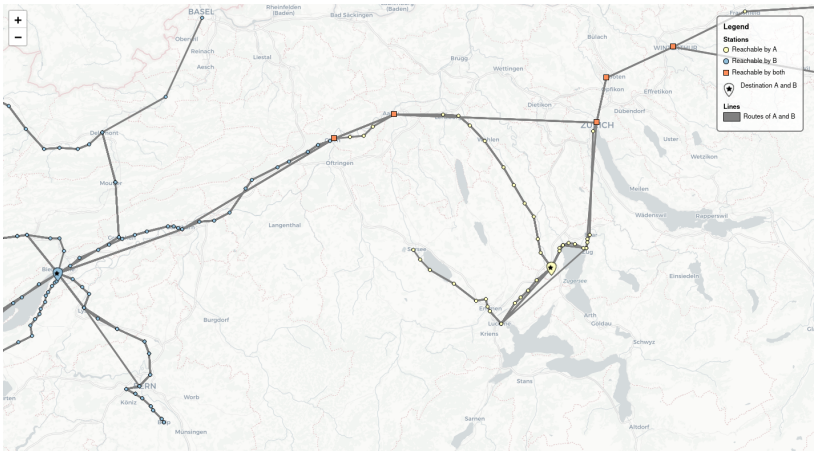
While going through the routes which are available, I discovered a lot of them either do not exist, are specialty routes which only go once a year or are not accessible by the general public. The ones which I definitely know are not normal routes are being filtered out and not even displayed.

Visualization

To visualize the train lines and their stations a map was chosen as the chart type, as the GTFS data provides the coordinates for each station. The map is the right tool for the job as it shows where the stations are and approximately where the train will go through.

At the very top are the filters for the data. Dropdowns to select the destinations, a multiple select dropdown for route and weekday filter and a slider to select relevant date times.
All these filters work together to be able to find the stations which are most relevant to each user.

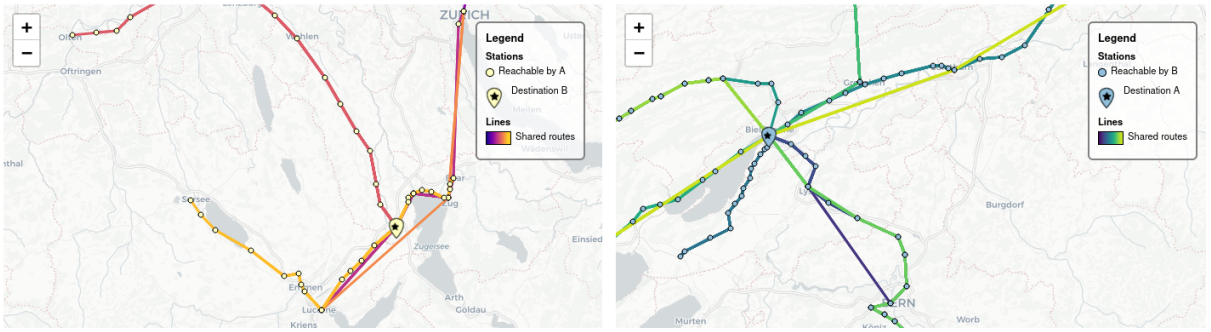
Three divergent colors were chosen (with ColorBrewer) for the stations to signify if they are reachable from destination A or B or are shared by both. Because the shared stations are the most important piece of data, it is additionally encoded with a square shape instead of the usual circle. The shape also helps if the user is color blind, because the data encoding does not only rely on colors. The lines of the map are gray, because that information is not



Shared stations chart

as important as the shared stations. Sometimes it was unclear where the chosen destination was, to prevent that a special marker is placed on the destination location to be able to distinguish them from the normal and shared stops.

To display more information about which lines come from which destination the technique of small multiples was applied. Two small maps are beneath the main map to show the routes which visit the destination. The lines of the small maps are encoded with perceptually uniform sequential colormaps, so that the user is able to distinguish between multiple routes.

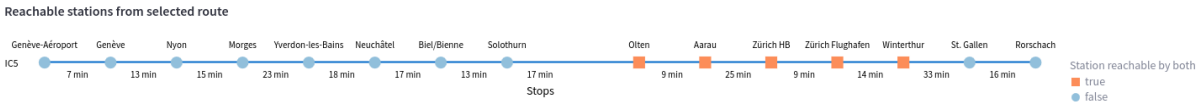


Small multiple of the routes

The legend for the colors and shapes of the routes and stops on the map is built into the map, instead of having it outside as it would disrupt the flow of using the chart. The legend also shows the possible color gradient of the routes, indicating to the user which colors are possible for the lines.

Every route and station also has a tooltip when they are hovered over to show the relevant information. Stations display their names and routes display their name as well and all the stops which are being served.

The visualization for the selected route was done with a graph similar chart. It displays every stop in the route and if they can reach both destinations or not. The nodes represent every station and have the station name above it. The shape of the node is the same as on the map above to show if both chosen destinations are reachable from that stop. The consistency is kept, because when it's a shared station it will be displayed with a square. Then between every stop the time it takes to get between them is shown just underneath.

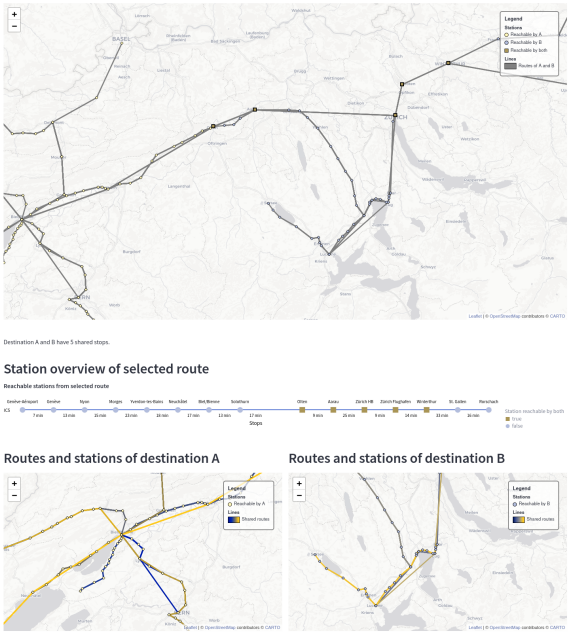


Route detail chart

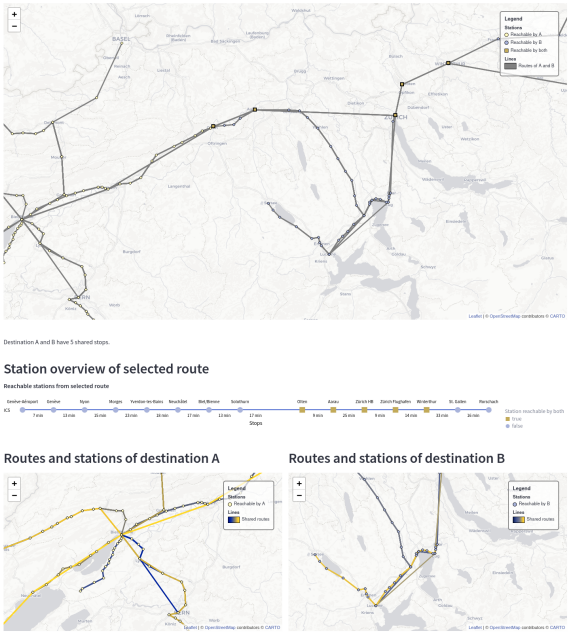
That is also for the sake of balancing how much text is where. Because the top already has the station names, there is no more space to fit more and the time information has to be displayed down below.

Accessibility test

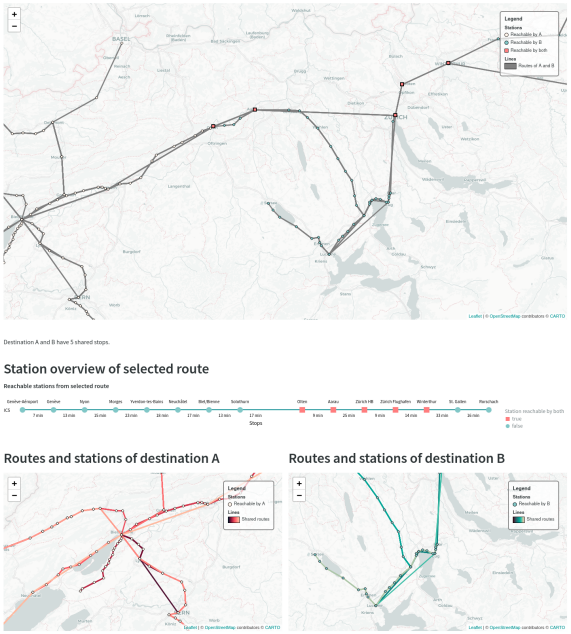
I performed color accessibility tests because some of the information on the dashboard is encoded through colors. This is to see if the chosen colors are sufficiently contrasted. The test was conducted with the Firefox Accessibility Inspector.



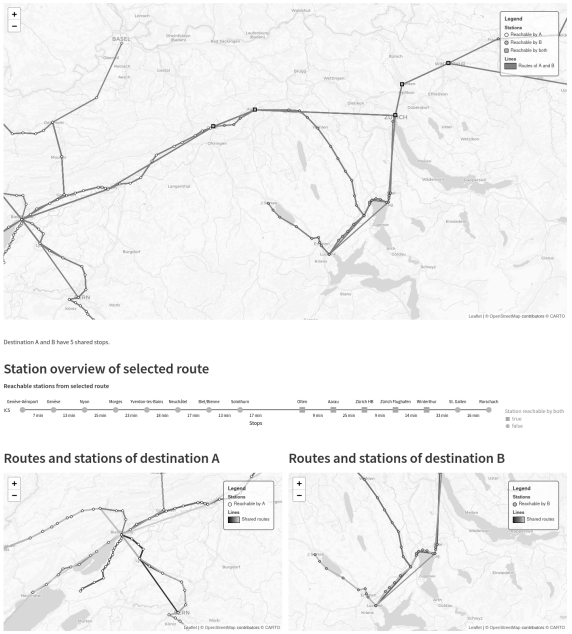
Colors with Protanopia



Colors with Deuteranopia



Colors with Tritanopia



Colors with Achromatopsia

The conclusion after the testing with color blindness filters reveal that the most important aspects of the dashboard: the stops and routes are still distinguishable. By encoding the shared stops with a shape as well, it can even be used by people with Achromatopsia.

Usability test

A lot of valuable feedback can be gathered by conducting a usability test with people who have never used the dashboard. Because they don't know how to use it or have some deeper knowledge of the system beneath. Therefore the feedback is unbiased and more useful than testing it as the creator.

Task

You live in Bern, but you work in Zurich and study in Rotkreuz. You are considering moving somewhere where the commute times are shorter and easier for you.

Feedback

I was able to find a location to consider moving to. Also finding a few other useful informations about other train routes which go through that location.
The loading times are a bit too long.
The color scheme looks good.

Conclusion

The feedback provided is valuable. Considering that I have already implemented and tested several performance enhancements, and no other significant issues have been identified, I can confidently affirm that the dashboard is functioning as intended. It is now ready for use by the target audience, who should find it user-friendly and comprehend the displayed information effectively.

Libraries

- gtfs-kit
- pandas
- seaborn
- streamlit
- streamlit-folium
- altair

gtfs-kit

There are some other python libraries for GTFS data, but gtfs-kit was the choice because it converts the GTFS file into pandas dataframes for further processing. This is the easiest way to work with GTFS data, as it is provided in a tabular manner with foreign keys available.

Streamlit vs Dash Plotly

I chose Streamlit over dash for two main reasons. First of all the interactivity which was needed for the map is not available in dash therefore making it unsuitable. Additionally the

developer experience using Streamlit is better than Dash Plotly. Instead of gatekeeping styles for the app behind an Enterprise subscription like Dash, Streamlit offers all it has in an Open Source manner. Which also suits my personal philosophy more.

Pydeck vs Folium

The reason I chose folium is because the Interactivity which I was planning is not supported by the Streamlit Pydeck implementation. Even though Pydeck is more powerful than Folium, I didn't need the additional features (WebGL) and without Interactivity, I could not implement the planned Features for the dashboard. Pydeck does have a built-in interaction handler, which would work if I built a standalone web app with it. But the integration into Streamlit did not port the interactions into Python. Meanwhile Folium was implemented with integrations in the streamlit-folium package.

Others

Pandas is used for the tabular data manipulation provided by gtfs-kit.
Seaborn is only used to generate the color pallets for the visualizations on the map.
Altair is the default plotting library used by Streamlit.

Sources

Data:
https://opentransportdata.swiss/en/dataset/timetable-2024-gtfs2020
GTFS Reference:
https://opentransportdata.swiss/en/cookbook/gtfs/
Colors:
https://colorbrewer2.org/
https://matplotlib.org/stable/users/explain/colors/colormaps.html#sequential