Technical Documentation: GTFS Transit Data Analyzer

Author: Liz Obst
Last Updated: January 2026
Tech Stack: Python, Streamlit, Pandas, Folium, Plotly, Anthropic API

---

1. Project Overview

Purpose

The **GTFS Transit Data Analyzer** is a web-based tool designed to make public transit data accessible and actionable. Public transportation agencies publish their schedules in a standard format called GTFS (General Transit Feed Specification). While this data is open, the relational text files are difficult to read manually.

I built this application to ingest any valid GTFS feed URL and instantly generate a dashboard showing route frequencies, service maps, and network statistics.

Key Features

- **Universal Ingestion:** Automatically downloads, unzips, and validates GTFS feeds from a user-provided URL.
- **Headway Analysis:** Custom algorithms calculate the time between buses/trains (headways) to determine true service frequency.
- **Geospatial Visualization:** Interactive maps using Folium to plot stops, routes, and station density.
- **AI Analyst Assistant:** An integrated chatbot powered by the Anthropic API that allows users to ask natural language questions about the specific transit network being analyzed.

---

2. High-Level Architecture

The application follows a standard **ETL (Extract, Transform, Load)** pattern, wrapped in a reactive Streamlit interface.

1. **Extraction:** The app downloads the ZIP file from the user-provided URL to a temporary directory using the `requests` library.
2. **Validation & Loading:** It checks for the minimum required files (`agency.txt`, `routes.txt`, `trips.txt`, `stops.txt`, `stop_times.txt`) and loads them into Pandas DataFrames.
3. **Processing:** * **Joining:** Performs multi-table joins to link specific trips to their defined routes and stops.
    - **Cleaning:** Handles common GTFS data issues, such as converting "25:30:00" timestamps (service past midnight) into calculable integers.
4. **Visualization:** The processed data is rendered into Plotly bar charts (for frequency/service span) and Folium maps.

---

3. Data Model

The application relies on the relational nature of GTFS data. I process the following key relationships:

- **Agency:** The transit provider details.
- **Routes:** Definitions of the lines (e.g., "Route 10 - Crosstown").
- **Trips:** Specific instances of a route occurring at a specific time.
- **Stop Times:** The arrival/departure times for every stop on every trip.
- **Stops:** Geographic coordinates (Latitude/Longitude) of the stops.

> ***Note:*** *To ensure performance, the app stores these DataFrames in the Streamlit* `session_state`*. This prevents the app from re-downloading and re-parsing the large datasets every time the user interacts with a widget.*

---

4. Key Logic & Algorithms

4.1 Headway Calculation

One of the most complex parts of the application is calculating "Headways" (the wait time between vehicles). Since GTFS only gives absolute arrival times, I wrote a custom algorithm to:

1. **Group:** Group trips by `route_id` and `direction_id`.
2. **Sort:** Sort all trips chronologically by their start time.
3. **Delta Calculation:** Calculate the difference (in minutes) between consecutive trips.
4. **Filter:** The algorithm filters out "overnight gaps" (e.g., the gap between the last bus at 11:00 PM and the first bus at 5:00 AM) to prevent these large gaps from skewing the average frequency metrics.

4.2 AI Chatbot Integration

The application includes a conversational interface powered by the **Anthropic API**.

- **Context Injection:** When the data is loaded, the system generates a summary of the network (e.g., "This agency has 45 routes, the busiest stop is X...").
- **Prompt Engineering:** When a user asks a question, this data context is appended to the system prompt. This allows the LLM to answer specific questions about the loaded dataset (e.g., "*Which stop is the busiest?*") without hallucinating generic data.

---

5. Challenges & Error Handling

Handling real-world data requires resilient error handling. The application specifically handles:

- **Network Timeouts:** Uses explicit timeouts during the download phase to handle slow agency servers smoothly.
- **Corrupt Feeds:** Validates that the ZIP file allows extraction and contains the minimum required text files before attempting to parse.
- **Data Oddities:**
  - *Timestamps:* GTFS uses "25:00:00" to represent 1:00 AM the following day. The parser converts these strings into minutes-from-midnight integers for accurate sorting.
  - *Missing Coordinates:* If a feed is missing shape data, the map visualizer gracefully falls back to plotting stop markers rather than crashing.

---

6. Future Improvements

- **Database Backend:** Currently, data is held in memory for the session. Integrating a SQL database (like PostgreSQL) would allow for historical analysis and faster retrieval for frequently accessed feeds.
- **Real-Time Support (GTFS-RT):** Adding support for the "Real-Time" extension to show live bus positions compared to the schedule.

Dependencies

- `streamlit` (UI/App Framework)
- `pandas` (Data Analysis)
- `folium` (Mapping)
- `plotly` (Data Visualization)
- `anthropic` (LLM Integration)