Read me

Introduction:

This pipeline creates 2 tables, as asked in the assignment, and simple visualization. In my approach, I tried to utilize modularity and use the full benefits of the separation of concepts. I believe I created an easy-to-follow and customizable script.

Data structure:

I followed the standard ETL approach.

1) Extract: Handles fetching data from our API. I tried to implement robust error handling and retries logic.
2) Transform: Process raw data to our 2 tables and aggregation CSV, taking care of cleaning, standardization, and calculated derivative metrics.
   Functions converting raw data into Pandas df to apply transformation. The first thing that I mentioned when I was working with data was that our connectionTypes column had information about different connectors and tariffs, so I decided to count each of them and separate these values. The next steps were to create metrics (for example, total available chargers, how many are available, etc), change some names, and set the status if any 1 or more charges will be “available”; the status will be available; if not then it will be “not available”, handling duplicate station ID.
3) Load: Loading data. Maintaining data integrity by providing flexible methods to save processed data and load it for further processing or analysis. This step means we don't have to redo the time-consuming extraction and transformation processes every time we want to analyze the data, which makes your workflow much more efficient and keeps our pipeline modular.
4) Validation: works by checking for the presence of essential columns and detecting any missing, duplicate, or anomalous values that might compromise data quality. I set crucial columns and values, and if during the validation stage, some file is missing them, it will return as an error. Additionally, I added geographical coordinates to ensure they fall within the correct range. I also implemented something like a report with comments about values. This is the example:

   2025-04-08 00:23:39,869 - INFO - Utilization status counts: {'Available': 4051, 'FAULTED': 136, 'OutOfOrder': 69, 'Occupied': 67}
   2025-04-08 00:23:39,869 - WARNING - Validation issue: Hourly data issue: Found 59 records (13.8%) where total_connectors doesn't match the sum of status counts. This may be due to connectors with status values not counted in the standard categories.
   2025-04-08 00:23:39,869 - WARNING - Validation issue: Stations data issue: Column 'address' has 13 missing values (3.0%)
   2025-04-08 00:23:39,869 - WARNING - Validation issue: Stations data issue: Column 'description' has 259 missing values (60.7%)
   2025-04-08 00:23:39,869 - WARNING - Validation issue: Stations data issue: Found 427 stations (100.0%) where total_connectors doesn't match sum of specific types. This is expected if there are connectors without specific types assigned.
   2025-04-08 00:23:39,870 - WARNING - Validation issue: Utilization data issue: Found 'FAULTED' status values

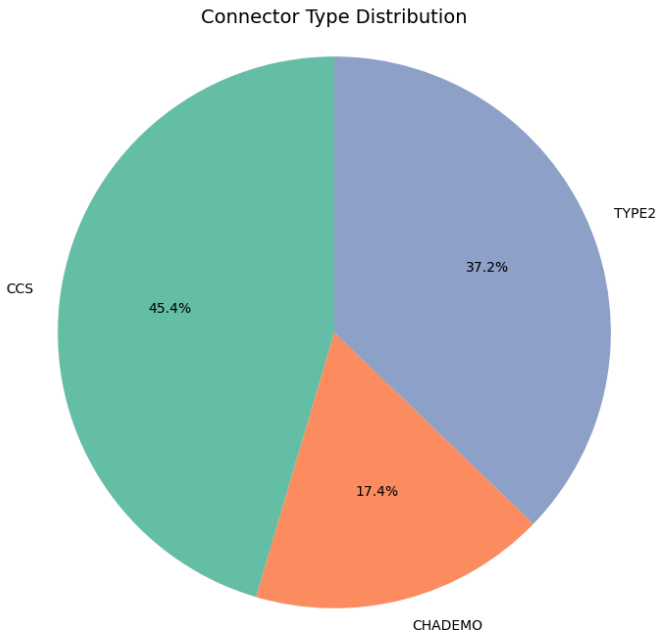
5) I created a file with utility functions that support logging, metadata management, data validation, and summarizing statistics. In addition to logging, the code provides a function to save metadata, where additional contextual information about the data pipeline is stored.
6) Main, the code is an orchestration for this ETL pipeline. Its purpose is to coordinate a series of processes, such as extracting raw data, transforming it into structured formats, validating the data, and saving it for later use or visualization.

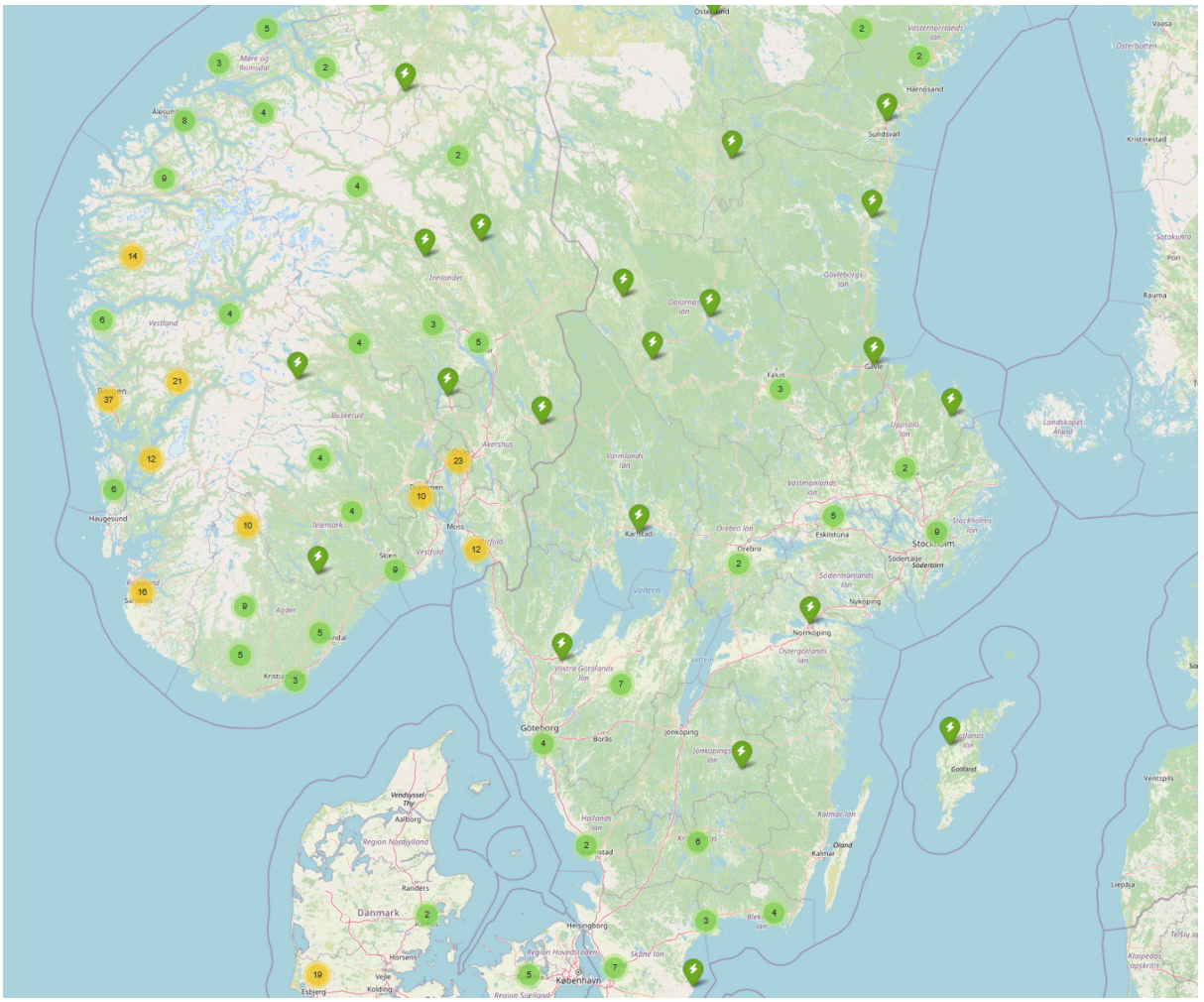
   Two main functions drive the process:

- The **run_single_extraction** function is designed to perform one full cycle of the ETL pipeline. It extracts data from the source, transforms it into structured formats for stations and utilization, aggregates the utilization data on an hourly basis, validates the transformed data to ensure everything looks correct, and finally saves these processed DataFrames as CSV files. This function is useful for on-demand extraction where you want a one-time run of the entire pipeline.

- The **run_continuous_extraction** function, on the other hand, is intended for continuous data collection. It sets a duration (24 hours) and an interval (60 minutes) at which the ETL pipeline runs repeatedly. This function continuously collects fresh data, logs the extraction count and success rate, and then waits until the next scheduled extraction time. It also gracefully handles interruptions and errors during the continuous run.

7) Vizualization. I made 2 simple visualizations: an interactive map of the charging station and a Pie chart of the Types of connectors.





Assumptions

There was a lot of things that I mentioned with such limited time and visualization, one of those was that during this hour (10-11 PM) there were not so many used charging stations, only about 2% of the total which might seems but if we take in consideration nightime it makes sense. What was interesting was that the combined number of faults and out of order was greater than the number of used ones and was equal to 5%. The data was pretty consistent but had some missing values for description more than 60% and 13% for address. What was weird was that total connectors never matched the sum of specific types, which probably means that there are specified types. Another thing about this data is that it sends only the latest data, so for a better analysis, u need to run the pipeline every hour, 30 min, 15, or even in real time.

Improvements

The data I worked with can be highly valuable for analysis like peek hour performance or even predicting the business on stations with depends on amenities, but to perform all of that data, as I mentioned before, need to be ingested in real time to allow 24 hours monitoring.
Adding extensive logging and error handling can benefit any program. Even though I tried to cover as much as I could in my limited time, mine is not an exception.
Applying domain knowledge and creating more robust cleaning and validation will improve the data quality.
Unfortunately, despite all my love for creating user-friendly and story-telling visualizations, I had no time to create those this time. The first thing I would add is hour-to-hour performance, how many chargers were used in total, which regions were more affected, or maybe even particular stations. If the prices are changing constantly I would also plot how it affects the number of cars charging or total electricity consumption.
Another thing that could have been added is setting alarms, improving reports about the quality of the metrics, and if there is a consumption per station or even client, we could do outlier detection using z-scores or IQR.
The last thing is adding data health metrics and connecting them to dashboards.