Final Project: Citibike Data

Data Engineering Practices & Principles, EN.685.652.8VL.SU24

Hannah Haas, Jeremy Hirschler, Isabel Perry, Dan Ansari

Project Description

This repository contains all of the code to:

- **Coallate Data**: coallates data from the Citibike rideshare program in NYC, alongside weather and demographic data
- **Build a Database**: creates a Postgres database of the collected data
- **Run an automated report**: starts an Airflow server which generates+stores an HTML report
- **Deploys an API**: runs an API which allows the user to query Citibike station information

The project has been configured to separate the data coallaction step from the above steps so that:

- The below steps can be followed to build the database from included CSV files of processed data. This allows for more seamless running of the report automation and API
- Scripts to generate each dataset are included separately and have instructions for each respective script to be run

---

Environment Set Up

Docker Image Creation

The below requires Docker Desktop be installed first.

1. Start your Docker application

- **Linux**: `sudo systemctl start docker`
- **Mac/PC**: Start the Docker Desktop application

2. Clone this repo to a directory on your machine

```
git clone https://github.com/ansaridan/jhu-data-engineering-team-5.git
```

3. In a terminal, navigate into the cloned repo directory

```
cd jhu-data-engineering-team-5
```

4. Create the Docker image from the repo's specifications + start the containers

```
docker-compose up --build
```

5. Build the database from within the docker container After confirming that the containers are running (especially the Postgres container), run the ETL script from a CLI in the `jhu-data-enginerring-team-5_jupyter_1` container:

```
python3 etl_process/etl.py
```

This file should output logs along the way, but will take 5-10 minutes to run.

6. Start the Flask API A CLI in the Jupyter container (e.g. accessed through Docker Desktop), run the following command to start the Flask API discussed below.

```
python3 api/api.py
```

Using the data outputs

After ensuring that the docker containers are running, and the ETL script has run successfully, you can interact with our data environment as follows. We have configured our `docker-compose` to start each of the Jupyter Notebooks, Airflow Webserver+Scheduler and API apps, but it's worth checking that they are all up and running.

Running automated reporting in Airflow

- You can access the Airflow webserver at `http://localhost:8080/home` with username/password = admin@example.com/admin
- Inside Airflow, you can run the report by turning on the `run_report_dag`
- The resulting interactive HTML reports will appear in `reporting/reports` and can be viewed in Chrome

Running/Using the Flask API

You can test the Flask API in a number of ways. From within the Jupyter container, it can be started in a CLI with `python3 api/api.py` and then tested at `http://localhost:8001`. There is also a notebook `api/api_extract.ipynb` that can be run in the Jupyter container to show the functionality once the API has been started.

The API has the following commands:

- `/get_station_ids`
- `/read_precip_data`
- `/read_general_data`
- `/read_pressure_data`
- `/read_sky_data`
- `/read_wind_data`
- `/read_station_data`

Running Jupyter for Development

- You can use Jupyter for development at `localhost:8080` with the token provided in the container logs. Click on the "airflow-jupyter-1" container, this will take you to the logs. Select a statement that looks like this: "http://127.0.0.1:8888/lab?token=e8e0df95307f9203b96e93c77ae8dd045263519715da925d" This will open your JupyterLab environment.
- Once you have Jupyter notebook open, open the `api/api_extract.ipynb` notebook, in here, you can apply filters of your choice for each individual command. There are example commands already filled in that you can use.
- For dates, our data only ranges from July 1, 2023 to June 30, 2024, so if you want to apply a date filter, please stay within these bounds.

Deliverables Summary

The project requirements are: **[1] Create at least five separate tables based on the data schema**

- The tables in our project are outlined in the `etl_process/schemas` folder

[2] Create an entity-relationship diagram (ERD) of your schema in PDF format, including any keys or relationships

- The ERD diagram can be found in at `etl_process/ERD.pdf`

[3] Create an automation and an API to export or consume an aggregated view or interesting report of the data.

- See the above section `Running automated reporting in Airflow` and you can find the outputs in `reporting/reports`

[4] Jupyter Notebook(s) or Python script(s) containing the data transformation code.

The following scripts can be run after the environment set up to generate our processed CSV files:

- `etl_process/clean_weather_data.ipynb`. To run this script, load follow the "Running Jupyter for Development" instructions on how to launch your JupyterLab environment. Once it is open, navigate to the above location and open the clean_weather_data.ipynb file. Run the Jupyter-Notebook in succession to properly clean the raw NYC weather data. To run the individual cells, there is an arrow icon towards the top of the notebook. Click that for every cell you enter. Some portions may take a minute or two to complete. Once the script is finished running, the cleaned data is saved off and can be viewed in `etl_process/processed_data/weather/`.
- `etl_process/ride_data.py`. To run this script, load follow the "Running Jupyter for Development" instructions on how to launch your JupyterLab environment. Once it is open, open a new terminal. Use the command `python3 ride_data.py` to run the script. It will download the the raw citibike monthly run data from all of 2023 and until June 2024 and save the transformed clean data into 13 csvs (June 2023 - June 2024). You will be able to track the progress of this etl process, but expect it to take approximately 20 mins. The resulting processed data can also be viewed in etl_process/processed_data/rides/station_traffic.
- `etl_process/data_processing/irs_ETL.py`. This script grabs data directly from the IRS website and transforms it into a more usable csv file containing only data from NYC. Simply run this script in the Docker container terminal and it will create the nyc_irs.csv file.
- `etl_process/data_processing/irs.py`. This script takes the nyc_irs.csv data in CSV format and creates the schema and database tables. Recommended to run the irs_ETL.py file first. To run this script, open a terminal within the Docker container environment and it will create the schema and the proper tables. Note these 2 files should not need to be run when creating the Docker container since the irs_data.yml file will be run instead, and will create all the necessary schema and tables.

[4] Have SQL script containing the table creation and data loading commands.

- The `etl_process.etl.py` script comingles Python and SQL commands around our schema YAML files and processed CSVs to set up the database

[5] Have automation code and a web API implementation in Python

- See above notes on the Flask API

[6] Include required Docker files

- see the repo's `Dockerfile` and `docker-compose.yml`

[7] Include documentation file in PDF format (that can just be a save of this markdown file)

- see the PDF version of this file under `documentation.pdf`