Technical Documentation for Proj3_gans_scooters

A data pipeline that scrapes data about major European cities from Wikipedia, their weather from opeanweathermap.org, their airports and arriving flights from aerodatabox via rapid.api, all using AWS Lambda to AWS RDS.

Owner: The pipeline was written on behalf of the fictional scooter company Gans. This use case was part of the Data Science Bootcamp course, batch 6, by WBS Coding School.

Used since: 2022-06-16

Purpose: The company Gans wants to distribute its scooters more effectively to where they are used. They identified airports as one point of interest. Now they need to know when the weather at airports of big European cities is good and flights with potential users, i.e. backpackers, are arriving.

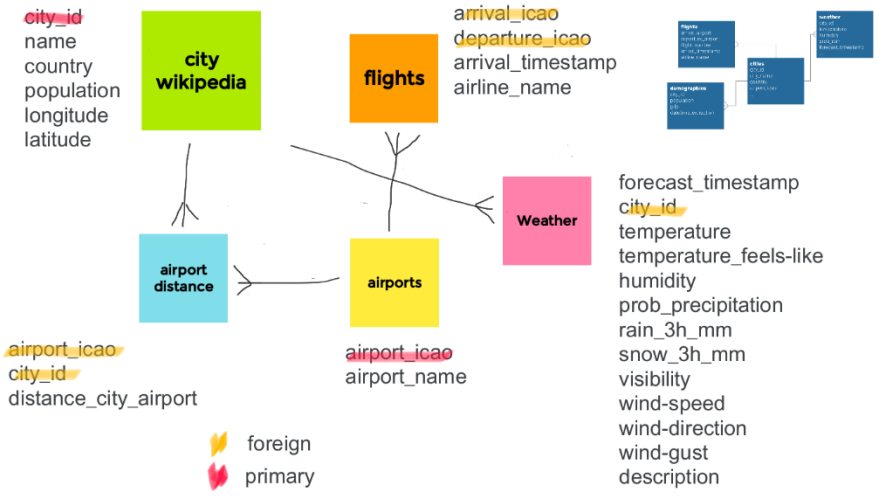
Input data: The whole scraping logic can be found here: https://github.com/circle-ish/DataScienceBC22/blob/main/proj3_gans_scooters/src/scraping.py .

Input pre-processing: This notebook: https://github.com/circle-ish/DataScienceBC22/blob/main/proj3_gans_scooters/proj3_main.ipynb guides through the scraping, data transformation and storage into a MySQL database.

Details

MySQL Connection: The connection is handled entirely by the MyMySQLConnection class in utils.py that takes the credentials (username, password, host address and port) and the database name as an input. Optionally it creates the database. The class has functions to create tables, add pandas DataFrames to the database, and add DataFrames to tables that require a foreign key which the function first fetches.

Database:



One thing that needs to be explained about the database layout is the *airport_distances* table. Since not all airports are directly inside the city, we had to include the surrounding 50km of a city inside the search. This might result in several of our scraped cities returning that airport. Because we did not want to exclude any potential customers based on an arbitrary decision to only take one city airport pair and not all of them, we had to design our database around that fact. An airport belonging to several cities neither fitted within the *airports* nor *cities* tables without violating primary key constraints or having multiple data points in one entry's column. Therefore, we exported that many-to-many relationship between *cities* and *airports* into the new table *airport_distances.*

Credentials: To execute properly the execution tries to find a config file named '.env' at the current path that needs to contain two sections and within:

- [APIs]
    - openweather_key = 33114be……eb9c1a6df4
    - aerodatabox_key = aa3d33d0……jsne163f6847091
- [SQL]
    - user = u……r
    - password = pas……ord
    - hostname = proj3-gans-……al-1.rds.amazonaws.com
    - port = 3306

API keys have caps for calls per minute and calls per month. Check here:
https://openweathermap.org/price
and here:
https://rapidapi.com/aedbx-aedbx/api/aerodatabox/pricing .
SQL parameters specify the credentials to connect to AWS RDS.

Scraping: All the scraping for cities, weather and airports is done with functions in scraping.py. It is a straightforward function chain no forks in the path. So,

- *scrape_wiki_cities()* calls *cleanup_cities()* and *add_lat_lon()*
- *scrape_weather()* calls *get_weather(), weather_json_to_df()* and *cleanup_weather()*
- *icao_airport_codes()* calls *get_icao_args()*

scraping.py also contains *city_airport_distance()* to calculate the distance for the *airport_distances* table. Scraping flights is handled by *flights_scraping_tomorrow()* in proj3_main.ipynb – a late addition that did not make it, screened and approved, into the scraping.py file.

Dependencies: The code uses imports from:

- datetime,
- timezone from pytz,
- requests,
- json,
- typing,
- sys,
- os, as well as:
- RawConfigParser from configparser,
- BeautifulSoup from bs4,

- FlatterDict from flatdict,
- Connection from sqlalchemy.engine.base
- distance from geopy.distance and heavily from:
- pandas

Automation: One AWS Lambda function handles all the scraping and adding to the database. The Lambda function uses three layers:

- AWSDataWrangler-Python39 v4,
- Klayers-p39-SQLAlchemy v3 and
- a custom layer that was created from a python.zip file containing
    - utils.py,
    - scraping.py,
    - flatdict.py from https://github.com/gmr/flatdict
    - geopy .py-holding subfolder from https://pypi.org/project/geopy
    - geographiclib .py-holding subfolder from https://pypi.org/project/geographiclib/

The Lambda trigger is set to execute every day at midnight (cron(0 0 ? * * *)).