Detailed Documentation of the Airbnb ETL Pipeline

Overview

This ETL pipeline is designed to process Airbnb listing data for New York City. The pipeline extracts data from a PostgreSQL database, transforms it to enhance data quality and derive additional insights, and then loads the transformed data back into the database.

Steps in the ETL Pipeline

1. **Data Ingestion:**
    - **Purpose:** Loads raw data from a CSV file (`AB_NYC_2019.csv`) into a PostgreSQL table named `listings`.
    - **Tools Used:**
        - `pandas` (for reading the CSV and creating a DataFrame)
        - `sqlalchemy` (for interacting with the PostgreSQL database)
    - **Process:**
        - The `load_data` function in `src/load_data.py` reads the CSV data into a Pandas DataFrame.
        - The DataFrame is then loaded into the `listings` table using `df.to_sql`.
        - If the table already exists, it is replaced with the new data (`if_exists='replace'`).
        - The script logs successful completion or any errors encountered.
    - **Output:** The raw Airbnb data is populated in the `listings` table in the PostgreSQL database.
2. **Data Extraction:**
    - **Purpose:** Retrieves the raw data from the `listings` table in the database.
    - **Tools Used:**
        - `sqlalchemy`
    - **Process:**
        - The `extract_data` function in `src/extract.py` uses SQLAlchemy to establish a connection to the database.
        - A raw SQL query (`SELECT * FROM listings`) is executed to fetch all data from the table.
        - The data is returned as a Pandas DataFrame.
        - The function includes error handling for database connection errors and invalid queries.
        - The function also has the capability to retrieve data in chunks (using the `chunksize` parameter) to improve efficiency for large datasets.
        - **Output:** A Pandas DataFrame `df` containing the extracted data.
3. **Data Transformation:**
    - **Purpose:** Cleans, normalizes, and enriches the extracted data to prepare it for analysis.
    - **Tools Used:**
        - `pandas`
        - `numpy`
    - **Transformations:**
        - **Data Type Conversion:**

        - `last_review`: Converted to datetime datatype for further date based analysis.
    - **Handling Missing Values:**
        - `reviews_per_month`: Replaced with 0 if missing.
        - `price`, `name`: Dropped rows with missing values for these columns.
    - **Feature Engineering:**
        - `is_superhost`: Binary feature indicating if a host is a superhost (1 or 0).
        - `is_longterm`: Binary feature indicating if a listing requires a minimum stay of more than 7 days (1 or 0).
        - `last_review_year`, `last_review_month`, `last_review_day`: New columns created from the `last_review` column.
    - **Calculate Metrics:**
        - `price_per_person`: Calculated based on room type (divided by 2 for "Private room," otherwise left as is).
        - `avg_price_per_neighbourhood`: Calculated by grouping data by `neighbourhood_group` and averaging the `price`.
        - `listing_age_days`: Calculates the number of days since the last review to capture the listing's age.
    - **Remove Unnecessary Columns:**
        - `host_name` and `last_review`: Dropped from the final DataFrame.
  - **Output:** A Pandas DataFrame `df_transformed` containing the cleaned and transformed data.
4. **Data Loading:**
  - **Purpose:** Loads the transformed data into the `listings_transformed` table in the database.
  - **Tools Used:**
    - `pandas`
    - `sqlalchemy`
  - **Process:**
    - The `load_data` function in `src/load.py` checks if the `listings_transformed` table already exists. If it does, it drops the table to avoid data duplication.
    - It then uses `df.to_sql` to load the transformed DataFrame into the table.
    - The function logs successful completion or any errors encountered.
  - **Output:** The transformed data is stored in the `listings_transformed` table in the PostgreSQL database.

Metaflow Workflow (`airbnb_flow.py`)

- **Purpose:** Orchestrates the entire ETL process using Metaflow, ensuring reproducibility, data versioning, and the potential for future scalability.
- **Structure:** The flow is defined as a Python class (`AirbnbETLFlow`) that inherits from `FlowSpec`.
- **Steps:**
  - `start`: Initializes the database connection.
  - `extract`: Calls the `extract_data` function to get the raw data.

    - `transform`: Calls the `transform_data` function to apply transformations.
    - `load`: Calls the `load_data` function to load the transformed data.
    - `end`: Logs a message indicating the successful completion of the flow.
- **Error Handling:** Each step includes `try-except` blocks to catch and log errors.
- **Execution:** Run the flow using `python airbnb_flow.py run`. The results of each step are stored as artifacts, and Metaflow automatically tracks data and code versions.