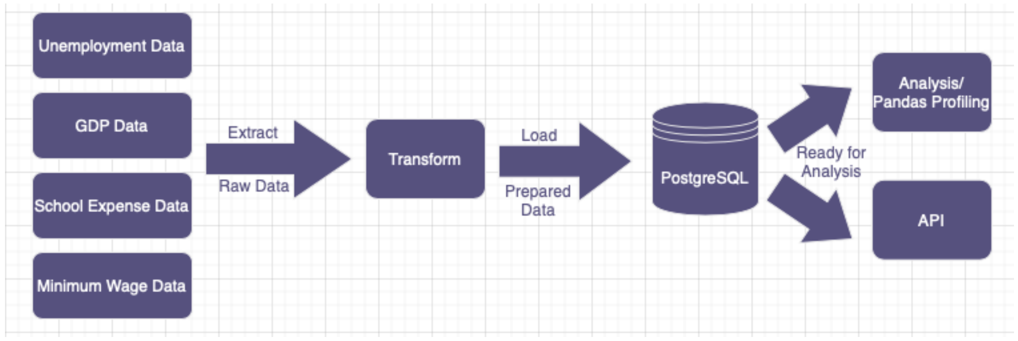
End-to-End Data Pipeline for State Economic Metrics

I. Overview

The purpose of this data pipeline is to streamline the integration, processing, and analysis of economic datasets from various sources like the USDA Economic Research Service and Kaggle. The project aims to deliver a self-contained data pipeline that can be deployed effortlessly using Docker. It automates data ingestion, transforms datasets with Airflow, and stores results in PostgreSQL.

The documentation covers dataset descriptions, normalization into 11 tables, and the generation of pandas profiling reports for comprehensive data exploration. The architecture includes stages for data loading, transformation, and storage, with Airflow managing ETL tasks and PostgreSQL serving as the database backend.

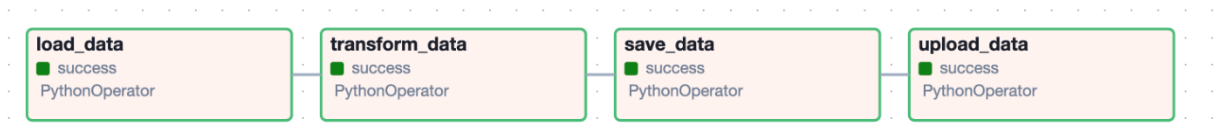


II. Data Sources

The following data sources are utilized in this project. For detailed information and previews of each dataset, please refer to the dataset check-in document submitted previously.

- **Unemployment Rate** - USDA Economic Research Service (Link)
  - Provides unemployment rates (2014-2022) and 2021 median household income by state. The data will be transformed to have years in a single column.
- **Average Cost of Tuition by State** – Kaggle (Link)
  - Displays average tuition and board costs (2013-2021) by state, from different types of postsecondary institutions.
- **Minimum Wage by State** – Kaggle (Link)
  - Contains state minimum wages and Consumer Price Index data (1968-2020).
- **GDP by State** – Kaggle (Link)
  - Provides GDP data by state and industry (1997-2020) from the Bureau of Economic Analysis. The data will be formatted with years in a single column.

III. Data Preprocessing and Ingestion (Airflow/Python/PostgreSQL)



The source tables require normalization to ensure data integrity and eliminate redundancy. The Airflow DAG named `data_cleaning_and_upload_dag` orchestrates data cleaning, transformation, and upload operations for datasets related to unemployment, GDP, school expenses, and minimum wage.

The DAG includes four main tasks:

- `load_data`: Loads datasets from Excel and CSV files, processes them, and pushes processed data to XCom for use in subsequent tasks.
- `transform_data`: Retrieves data from XCom, performs transformations (e.g., melting, column renaming, primary key validation), and pushes transformed data back to XCom.
- `save_data`: Retrieves transformed data from XCom and saves it to CSV files.
- `upload_data`: Retrieves transformed data from XCom, connects to a PostgreSQL database, creates necessary schemas and tables if required, and uploads data by converting each table to CSV and copying it into corresponding PostgreSQL tables.

The DAG's `default_args` dictionary sets parameters such as owner, start date, retry configuration, and delay between retries.

Tasks are implemented using `PythonOperator`, specifying the task ID and corresponding function. Task dependencies are established using the >> operator to ensure sequential execution: load data, transform data, save transformed data to CSV files, and load CSV files into PostgreSQL.

IV. Data Tables

Normalizing the four data sources resulted in the creation of 11 unique tables. Most tables are going to be connected by the year and GeoFIPS columns. An Entity-Relationship Diagram (ERD) can also be referenced and was submitted along with the other files for this project. Below is a description of each table and the columns within it.

- **table_location** – stores geographic data and identifiers
    - *geofips (pk)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
    - *geoname* – location associated with the GeoFIPS code. Data Type: text

- *region* - numeric code categorizing each GeoFIPS code into specific regional areas. Data Type: double precision

- **table_state_min_wage** – stores state minimum wage data for each year, including rates for small and large businesses.
  - *year (pk, fk1)* – Calendar year for which the data is applicable. Data Type: int
  - *geofips (pk, fk2)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
  - *state_min_wage* – the state minimum wage for the given year. The lower number between *small_business_min_wage and large_business_min_wage*. Data Type: double precision
  - *small_business_min_wage* – the minimum wage rate for those working in small businesses for the given year. Data Type: double precision
  - *large_business_min_wage* – the minimum wage rate for those working in big businesses for the given year. Data Type: double precision

- **table_cpi** – Contains average Consumer Price Index (CPI) values for each year
  - *year (pk)* – Calendar year for which the data is applicable. Data Type: int
  - *cpi_average* – the average consumer price index for a given year Data Type: double precision

- **table_fed_min_wage** – tracks the federal minimum wage rates applicable in the US for each year.
  - *year (pk)* – Calendar year for which the data is applicable. Data Type: int
  - *fed_min_wage* – the federal minimum wage for the US for a given year. Data Type: double precision

- **table_inflation** – contains multipliers for adjusting dollar amounts from various years to their equivalent value in 2020 dollars
  - *year (pk)* – Calendar year for which the data is applicable. Data Type: int
  - *inflation_multiplier_2020* – A multiplier for converting dollar amounts from different years to their equivalent value in 2020 dollars. Data Type: double precision

- **table_householdincome2021** – presents information about the median household income for each state (defined by GeoFIPS) in 2021.
  - *geofips (pk, fk1)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
  - *median_household_income_2021* – Median Household Income for each state for the year 202. Data Type: double precision

- **table_unemployment** – contains unemployment rates specific to geographic areas, categorized by year

  - *geofips (pk, fk1)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
  - *year (pk, fk2)* – Calendar year for which the data is applicable. Data Type: int
  - *unemployment_rate* – the state's unemployment rate for a given year. Data Type: double precision

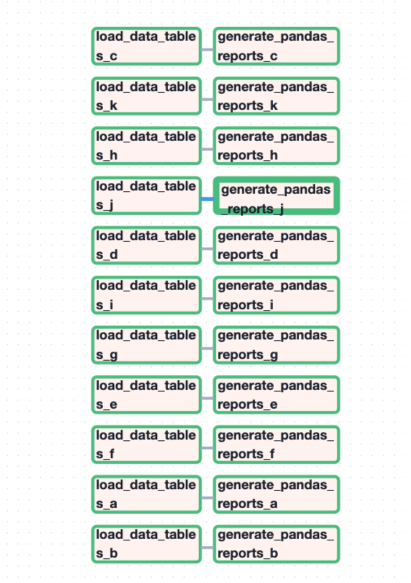
- **table_industry** – contains descriptions for various economic industries
  - *Industry_Code (pk)* – numerical code assigned to each industry Data Type: int
  - *Description* – text description that describes each industry Data Type: text

- **table_gdp** – contains gross domestic product (GDP) by industry, year, and geographic location.
  - *geofips (pk, fk1)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
  - *year (pk, fk2)* – Calendar year for which the data is applicable. Data Type: int
  - *Industry_Code (pk, fk3) – numerical* code assigned to each industry Data Type: int
  - *GDP* – amount of gross domestic product (GDP) in millions, for the given industry, year, and geographical location combination. Data Type: double precision

- **table_schoolexpenses** – Records average school expenses by type, year, and state.
  - *geofips (pk, fk1)* – numeric codes that uniquely identify geographic areas used by the U.S. Census Bureau to tabulate data. Data Type: int
  - *year (pk, fk2)* – Calendar year for which the data is applicable. Data Type: int
  - *school_expense_type_id (pk, fk3)* – unique code assigned to an expense type based on the *school_type, length,* and *expense_type*. Data Type: int
  - *expense_amount* – average amount spent by year and by state for the given school expense type. Data Type: int

- **table_schoolexpensetype** – Defines expense types categorized by school type, duration, and type of expense.
  - *school_expense_type_id (pk)* – unique code assigned to an expense type based on the *school_type, length,* and *expense_type*. Data Type: int
  - *school_type*– indicates if a school is public in-state, public out-of-state or private. Data Type: text
  - *length* – number of years that a degree takes to finish for the given *school_type.* Data Type: text
  - *expense_type* – classifies whether a particular expense is due to fees/tuition or room/board. Data Type: text

V. Data Postprocessing (Airflow/Python)


load_data_tables_c
generate_pandas_reports_c
load_data_tables_k
generate_pandas_reports_k
load_data_tables_h
generate_pandas_reports_h
load_data_tables_j
generate_pandas_reports_j
load_data_tables_d
generate_pandas_reports_d
load_data_tables_i
generate_pandas_reports_i
load_data_tables_g
generate_pandas_reports_g
load_data_tables_e
generate_pandas_reports_e
load_data_tables_f
generate_pandas_reports_f
load_data_tables_a
generate_pandas_reports_a
load_data_tables_b
generate_pandas_reports_b

This Airflow DAG named `generate_pandas_reports` is designed to generate pandas profiling reports for various datasets related to unemployment, GDP, school expenses, and minimum wage. The DAG consists of two main tasks for each dataset: `load_data_tables` and `generate_pandas_reports`. The `default_args` dictionary sets default parameters for the DAG, including the owner, start date, retry configuration, and delay between retries.

The `load_data_tables` function connects to a PostgreSQL database, executes a SQL query to retrieve data from specified tables, and loads the data into pandas DataFrames. The `generate_pandas_reports` function then generates pandas profiling reports for these DataFrames and saves them as HTML files.

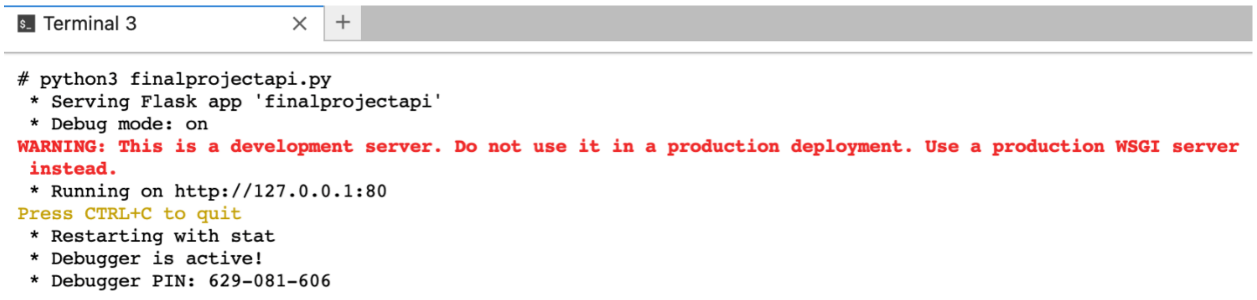
Tasks are defined using `PythonOperator`, specifying the task ID and the function to call. Dependencies between tasks are established using the >> operator, ensuring that the data is first loaded from the PostgreSQL tables and then the reports are generated.

VI. Application Programming Interface (API)

This Application Programming Interface (API) allows users to retrieve data from PostgreSQL tables via HTTP POST requests. Implemented using Flask, it is designed to handle requests efficiently and return queried data in a structured format. Users can specify which table to query and limit the number of records retrieved. Upon receiving a POST request at the `get_data` endpoint, the API extracts the table name and record limit from the JSON payload. A POST request was specifically chosen to have better flexibility with having multiple parameters – in this case, the table name and the number of rows to be returned (limit). It then constructs a corresponding SQL query to fetch the specified data. A connection is established to the PostgreSQL database using psycopg2, a PostgreSQL adapter for Python

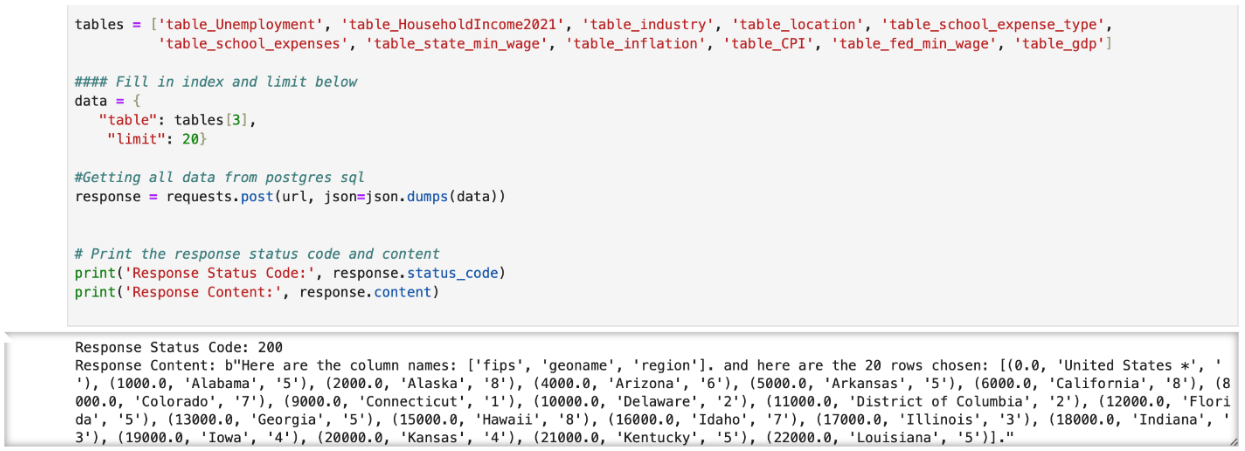
This functionality is particularly useful for applications requiring dynamic data retrieval from multiple tables without hardcoding each query. By leveraging this API, users can streamline their data access processes, ensuring they receive the necessary information promptly and efficiently.

To establish the Flask API connection, run `python3 finalprojectapi.py` in the terminal. A successful connection will be indicated by a response similar to the screenshot below, confirming that the application is running.



```
# python3 finalprojectapi.py
 * Serving Flask app 'finalprojectapi'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 instead.
 * Running on http://127.0.0.1:80
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 629-081-606
```

To fetch data from the API, open the notebook named `FinalProject.ipynb`. Refer to the tables DataFrame and specify the index of the table you wish to connect to in the data DataFrame. Additionally, adjust the limit parameter to modify the number of rows returned as needed.



```
tables = ['table_Unemployment', 'table_HouseholdIncome2021', 'table_industry', 'table_location', 'table_school_expense_type',
          'table_school_expenses', 'table_state_min_wage', 'table_inflation', 'table_CPI', 'table_fed_min_wage', 'table_gdp']

#### Fill in index and limit below
data = {
   "table": tables[3],
    "limit": 20}

#Getting all data from postgres sql
response = requests.post(url, json=json.dumps(data))


# Print the response status code and content
print('Response Status Code:', response.status_code)
print('Response Content:', response.content)
```

```
Response Status Code: 200
Response Content: b"Here are the column names: ['fips', 'geoname', 'region']. and here are the 20 rows chosen: [(0.0, 'United States *', '
'), (1000.0, 'Alabama', '5'), (2000.0, 'Alaska', '8'), (4000.0, 'Arizona', '6'), (5000.0, 'Arkansas', '5'), (6000.0, 'California', '8'), (8
000.0, 'Colorado', '7'), (9000.0, 'Connecticut', '1'), (10000.0, 'Delaware', '2'), (11000.0, 'District of Columbia', '2'), (12000.0, 'Flori
da', '5'), (13000.0, 'Georgia', '5'), (15000.0, 'Hawaii', '8'), (16000.0, 'Idaho', '7'), (17000.0, 'Illinois', '3'), (18000.0, 'Indiana', '
3'), (19000.0, 'Iowa', '4'), (20000.0, 'Kansas', '4'), (21000.0, 'Kentucky', '5'), (22000.0, 'Louisiana', '5')]."
```