Final Project: Designing and Implementing an ETL Workflow

By

Hoan Lam

CIS 660 Data Engineering

Instructor: Dr. Sorio Bolt

Due: 4/26/2025

Project Content

Introduction

For this project, I will examine an e-commerce consumer behavior dataset. The goal is to uncover insights into the datasets that will benefit online business owners. To accomplish this efficiently, I will create an ETL pipeline using Kestra. The data source for this project will be available on my GitHub. After the data is processed, the cleaned data will be sent to PostgreSQL. Both Kestra and PostgreSQL servers will be running via Docker. At this stage, I will retrieve the data from PostgreSQL and import it into my Python notebook on my local computer for further data analysis. Interesting graphs or findings will then be shown via Google Looker Studio for a more user-friendly environment.

Software & Tools Used

- **Visual Studio Code** – For writing and managing project code
- **Jupyter Notebook** – For research, development, and debugging
- **GitHub** – For version control and hosting the dataset
- **Kestra** – For orchestrating the ETL workflow
- **PostgreSQL** – To store cleaned and processed data
- **Docker** – To containerize and run Kestra and PostgreSQL servers
- **pgAdmin 4** – For managing and viewing structured data in PostgreSQL
- **ngrok** – To securely tunnel the PostgreSQL server to the internet
- **Google Looker Studio** – To visualize insights with user-friendly dashboards
- **Kaggle Dataset** – Source of the data - https://www.kaggle.com/datasets/salahuddinahmedshuvo/ecommerce-consumer-behavior-analysis-data/data

Challenges

I began the project by working entirely on Jupiter Notebook. I completed the extraction and transform phases, skipped the loading phases, and created visualizations in my notebook.

Next, I wanted to work on the Kestra, there were a few problems:

1. The installation was challenging since, during class, I could not get the program to work. Even with the help of the professor and some other classmates, Kestra still did not want to work.
2. Using Kestra was challenging because I didn't receive much training and struggled to get it to work. The interface was new, and the navigation was difficult. Sometimes, the server would not be running, and I wouldn't have known. Debugging was challenging due to log and output tabs, as I was unsure what to look for. The program was constantly crashing on me.
3. Extraction was challenging. I tried to input my CVS file to Kestra, but it did not work for security reasons. The transformation step was okay. The load was challenging because the connection to PostgreSQL needed to work. The table's structure must be correct; otherwise, you must constantly delete it from PostgreSQL to recreate it.

The following are the steps taken to solve each problem:

1. After struggling to install Kestra, I identified the problem. I had two PostgreSQL servers running on my machine: one from Docker and one locally. I was trying to connect to either the local PostgreSQL or the Docker PostgreSQL server, which was not up and running then (I was not very familiar with how Docker works at that point). Some of the steps recommended in class worked. I needed to use pgAdmin4 to configure the Docker PostgreSQL server to match the settings in the YAML file, ensuring that Docker is always running to maintain the connection. At this point, the problem of connecting PostgreSQL and Kestra is solved.
2. Now that I have gotten Kestra to work somewhat, I wanted to explore the tool. At first, it was just the flow creation and, eventually, debugging tools. I had to watch several tutorials. The data camp material did not provide much help as it focused on airflow, and no video on Blackboard covered this topic. However, through trial and error, I eventually became more proficient with the tool.
3. For extraction, I uploaded my .csv file to my GitHub page and let Kestra pull from there. Since Kestra is also running in Docker, it has no visual representation on my computer and does not support saving the .csv file locally via the input file feature. At the loading step, my problem was creating the correct table structure, which took some time since I needed to delete the incorrect table from PostgreSQL before attempting the ETL process again. This part was the most time-consuming. Figuring out Kestra syntax and folder structure and troubleshooting also added complexity while configuring these tasks.

Once the data was in PostgreSQL, querying did not raise any issues. I believe this section is intended to double-check that the data was correctly stored in the database.

PostgreSQL to Google Looker Studio was challenging. Since I had already converted my .csv file into the correct table format, I didn't want to convert it back to .csv to load it into Google Looker Studio. I wanted Google Looker Studio to have direct access to my Docker PostgreSQL database (Figure 1). To make the server visible to the internet, I had to install ngrok, which required setting up and verifying all the necessary configurations. It seems that if misused, ngrok can be harmful. It took some time, but I managed to get it to work.

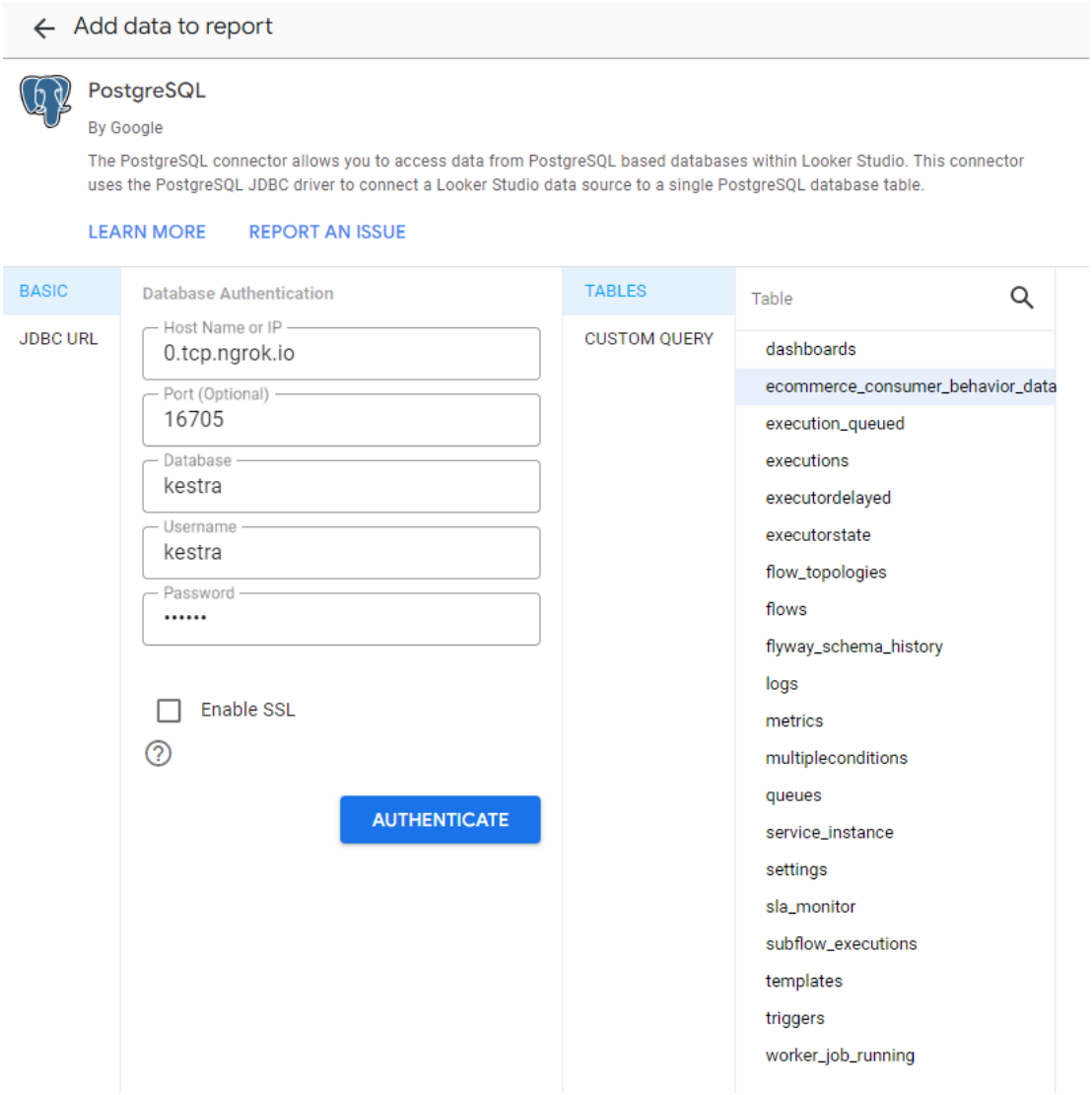

Figure 1. Google Looker Studio Setup

The rest involved creating charts from the selections I had made at the beginning of this process. I would not rate Google Looker Studio as easy to use, as it performs poorly in sorting or axis labeling. However, considering my effort to get here, I can't complain.

Documentation & deliverables

The ETL_R&D notebook discusses the dataset used, which is e-commerce consumer behavior analysis data, along with a link to the Kaggle website and a summary of the dataset and the meaning of each column.

Skipping the Data Extraction part during this stage via Kestra is different.

In the Data Transformation part, there are some discussions on how the data is being cleaned. To handle missing data, I did not simply delete the rows, as there were too many instances of missing data; instead, I created a new category called 'unknown'. To handle currency values, some cleanup is required to convert them to floating-point numbers.

I created some graphs at this stage to visualize some of the data to find outliers but did not find any, so I moved on (I later found out that the dataset was fictitious. I was disappointed when I found this out, but I still wanted to keep working on it since I had put so much work into it already – lesson learned for me) (Figure 2)

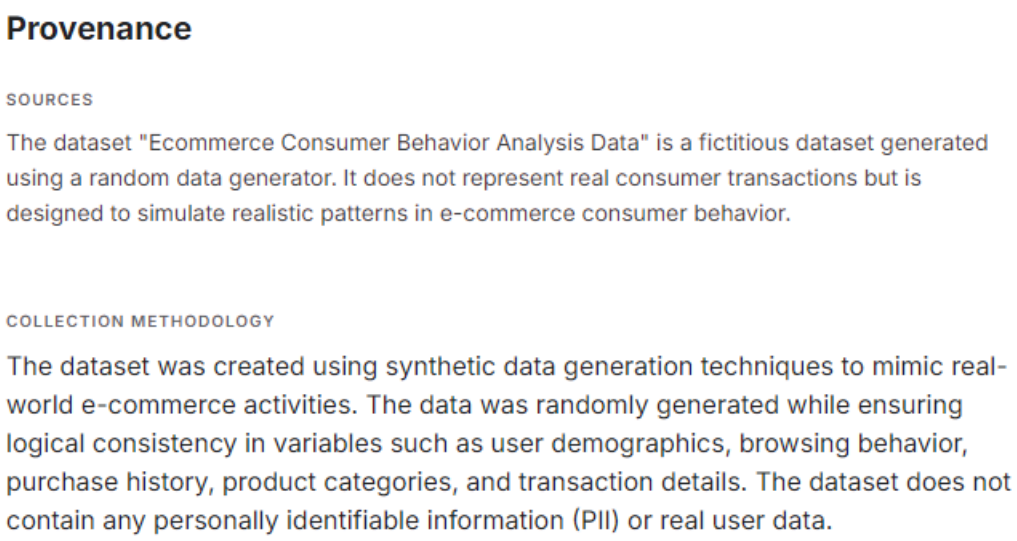

Figure 2. A Fictitious Dataset

Kestra workflow

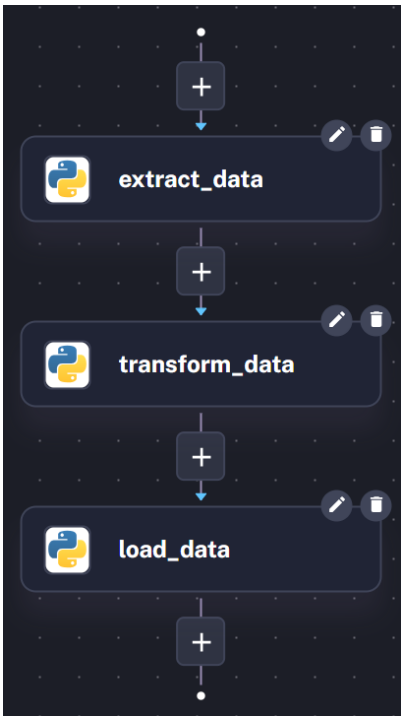

Figure 3. Kestra Pipeline

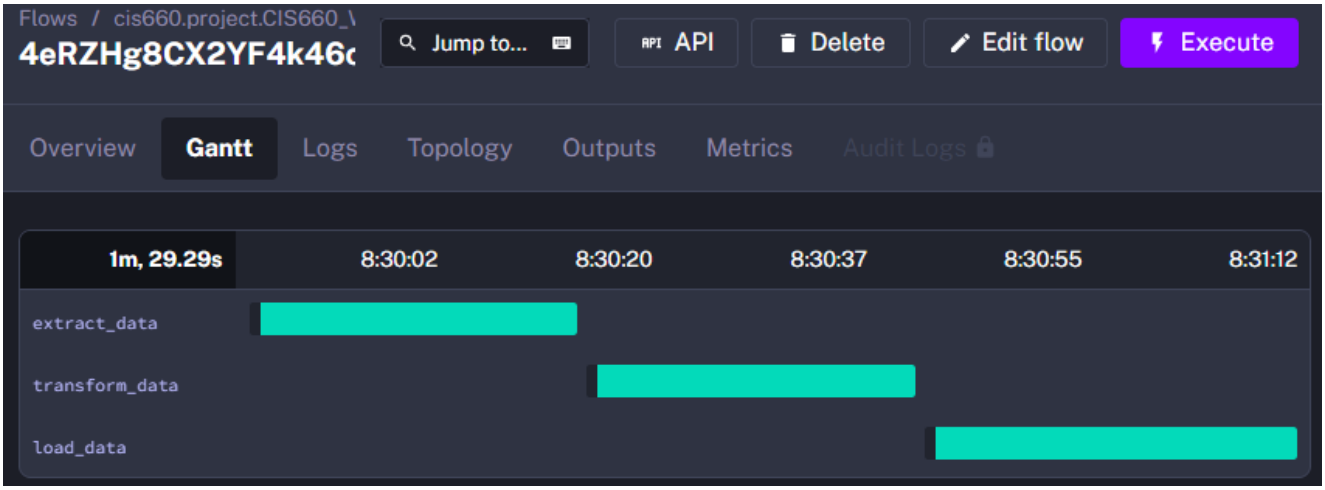

Figure 4. Gantt Chart

The code for this section is included under the appendix Kestra Code section.

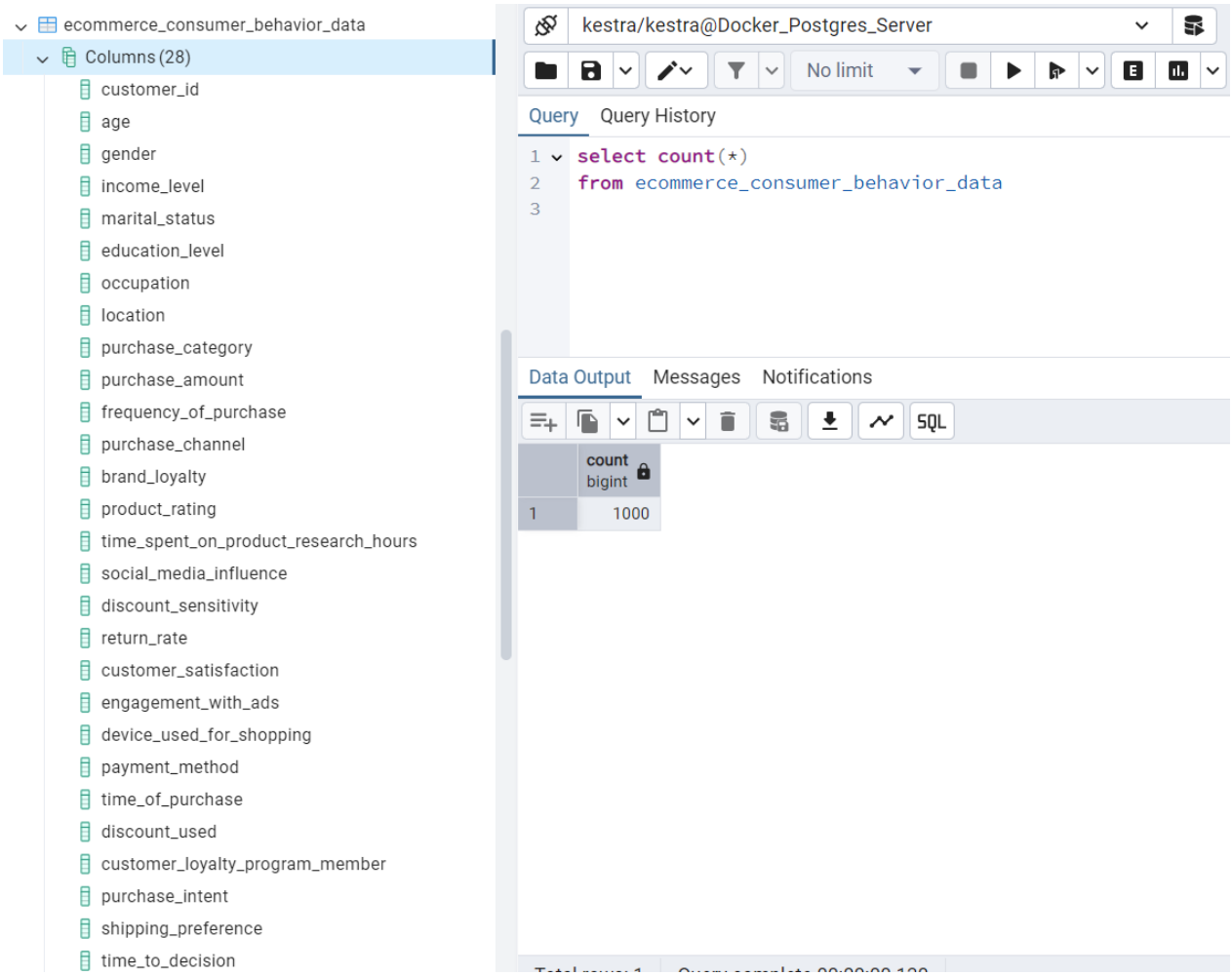

Figure 5. PostgreSQL confirmation query

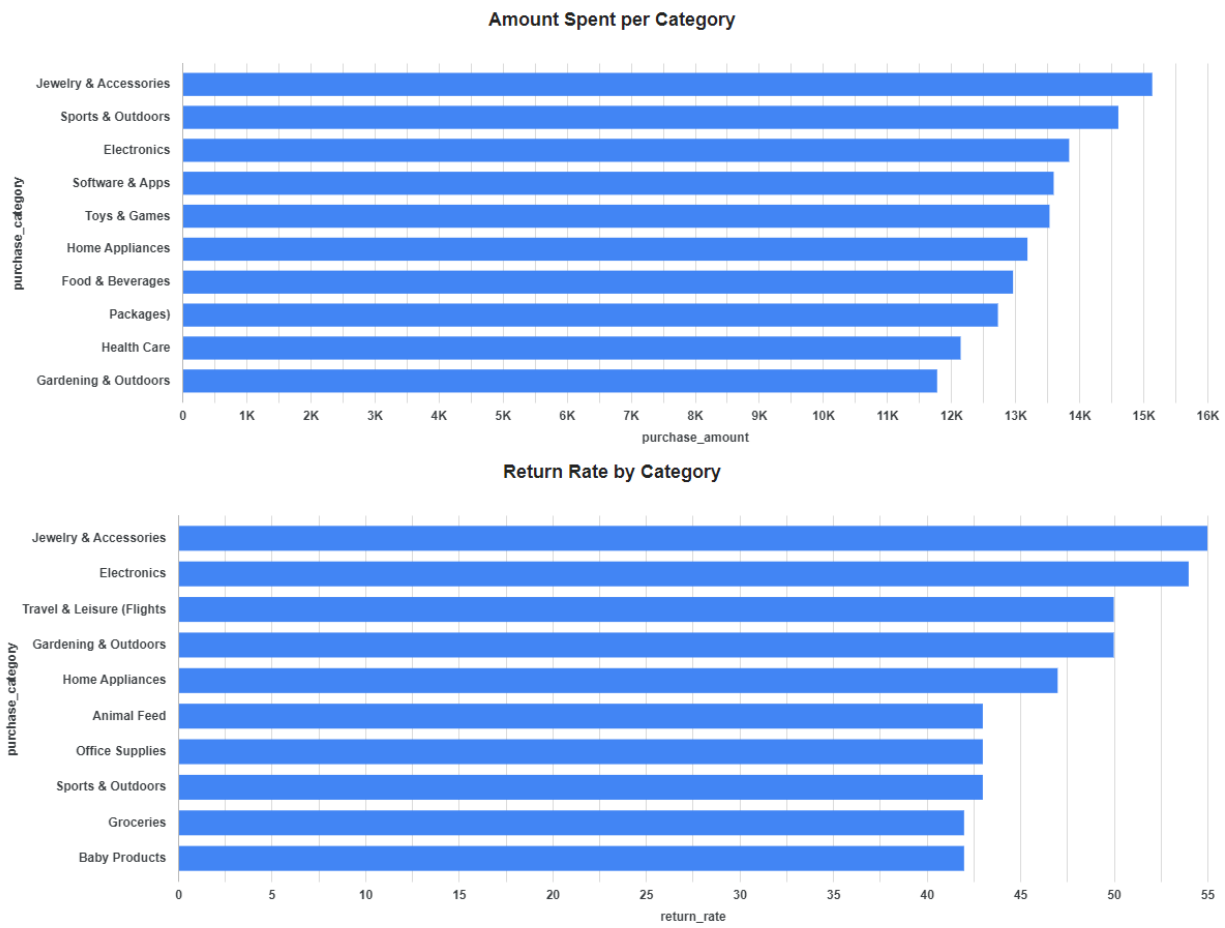

Figure 6. Amount Spent per Category and Return Rate by Category

The above visualization displays the most popular categories among consumers while illustrating each category's return rate. Jewelry & Accessories are the most popular, and at the same time, this category also has the highest return rate (Figure 6).

Figure 7. Consumer information

Women spend more on men; people with a Bachelor's degree spend the most; the 25-29 age group spends the most money (Figure 6).

Conclusion

It was a rough start, but eventually, I could identify and resolve the issues. Throughout the process, I gained a deeper understanding of the ETL pipeline and was able to set it up on my own. I also worked with several tools I hadn't used before—Kestra, Docker, and Google Looker Studio—and learned how to use them more effectively.

This project was challenging but in a good way. It pushed me to problem-solve, explore new technologies, and build something from scratch. I selected the results that made the most sense for the visualizations and helped communicate meaningful insights.

Looking back, I realize I made a mistake in choosing the wrong dataset to focus on. While it appeared intriguing and had a lot of features, I overlooked its credibility. Even so, it turned into a valuable learning experience, and I now have a better understanding of how to evaluate data sources more critically in the future.

Appendix

Kestra Code

```
id: CIS660_Workflow
namespace: cis660.project

tasks:
  - id: extract_data
    type: io.kestra.plugin.scripts.python.Script
    runner: DOCKER
    outputFiles:
      - extract_data.csv
    beforeCommands:
      - python3 -m venv .venv
      - . .venv/bin/activate
      - pip install pandas
    warningOnStdErr: false
    script: |
      import pandas as pd
      url = 'https://raw.githubusercontent.com/clcik-
click/CIS660_Project/refs/heads/main/Ecommerce_Consumer_Behavior_Analysis_Data.csv'
      df = pd.read_csv(url)
      df.to_csv("extract_data.csv", index=False)
      print("☑ Extract complete")

  - id: transform_data
    type: io.kestra.plugin.scripts.python.Script
    runner: DOCKER
    outputFiles:
      - transformed_data.csv
    beforeCommands:
      - python3 -m venv .venv
      - . .venv/bin/activate
      - pip install pandas
    warningOnStdErr: false
    script: |
      import pandas as pd
      df = pd.read_csv('{{ outputs.extract_data.outputFiles["extract_data.csv"] }}')

      # Hanlde missing values
      df['Social_Media_Influence'] = df['Social_Media_Influence'].fillna('Unknown')
      df['Engagement_with_Ads'] = df['Engagement_with_Ads'].fillna('Unknown')

      # Handle currency
      df['Purchase_Amount'] = (
        df['Purchase_Amount']
        .astype(str)
        .str.replace('$', '', regex=False)
        .str.replace(',', '', regex=False)
        .str.strip()
        .astype(float)
      )
```

```
      df.to_csv('transformed_data.csv', index=False)
      print("☑ Transform complete")

  - id: load_data
    type: io.kestra.plugin.scripts.python.Script
    beforeCommands:
      - python3 -m venv .venv
      - . .venv/bin/activate
      - pip install pandas psycopg2-binary
    warningOnStdErr: false
    script: |
      import pandas as pd
      import psycopg2

      df = pd.read_csv('{{ outputs.transform_data.outputFiles["transformed_data.csv"]
}}')

      print("Load - load data complete")

      conn = psycopg2.connect(
          host="host. Docker.internal",
          port=5433,
          database="kestra",
          user="kestra",
          password="k3str4"
      )

      cursor = conn.cursor()
      print("Load - connected to postgres")
      create_table_query = """
      CREATE TABLE IF NOT EXISTS ecommerce_consumer_behavior_data (
          Customer_ID TEXT,
          Age INT,
          Gender TEXT,
          Income_Level TEXT,
          Marital_Status TEXT,
          Education_Level TEXT,
          Occupation TEXT,
          Location TEXT,
          Purchase_Category TEXT,
          Purchase_Amount NUMERIC,
          Frequency_of_Purchase INT,
          Purchase_Channel TEXT,
          Brand_Loyalty INT,
          Product_Rating INT,
          Time_Spent_on_Product_Research_Hours FLOAT,
          Social_Media_Influence TEXT,
          Discount_Sensitivity TEXT,
          Return_Rate INT,
          Customer_Satisfaction INT,
          Engagement_with_Ads TEXT,
          Device_Used_for_Shopping TEXT,
          Payment_Method TEXT,
```

```
            Time_of_Purchase DATE,
            Discount_Used BOOLEAN,
            Customer_Loyalty_Program_Member BOOLEAN,
            Purchase_Intent TEXT,
            Shipping_Preference TEXT,
            Time_to_Decision INT
        )
        """

        cursor.execute(create_table_query)
        conn.commit()
        print("Load - table created")

        insert_query = """
        INSERT INTO ecommerce_consumer_behavior_data (
            Customer_ID, Age, Gender, Income_Level, Marital_Status, Education_Level,
            Occupation, Location, Purchase_Category, Purchase_Amount,
Frequency_of_Purchase,
            Purchase_Channel, Brand_Loyalty, Product_Rating,
Time_Spent_on_Product_Research_Hours,
            Social_Media_Influence, Discount_Sensitivity, Return_Rate,
Customer_Satisfaction,
            Engagement_with_Ads, Device_Used_for_Shopping, Payment_Method,
Time_of_Purchase,
            Discount_Used, Customer_Loyalty_Program_Member, Purchase_Intent,
Shipping_Preference,
            Time_to_Decision
        ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """

        for _, row in df.iterrows():
            cursor.execute(insert_query, tuple(row))

        conn.commit()
        cursor.close()
        conn.close()
        print("☑ Load complete")
```