+ **Task 8 Documentation**

* **Make a visualization on scrapped data ( Pharma Industries Random 10 companies ), then with the use of glue script make a sink table and stored that data into it by doing some transformation in glue script.**

**Step 1: Pulled a docker-compose on docker.**

**Step 2: Made database connection and vault and concourse connection.**

**Step 3: Created a scrapping python script and stored data into postgres.**

**Step 4: Made producer script.**

**Step 5: created audit table, function and triggers for cdc.**

**Step 6: Send data to kafka topic.**

**Step 7: Updated glue script along with sink table updated schema as transformation into data table was done.**

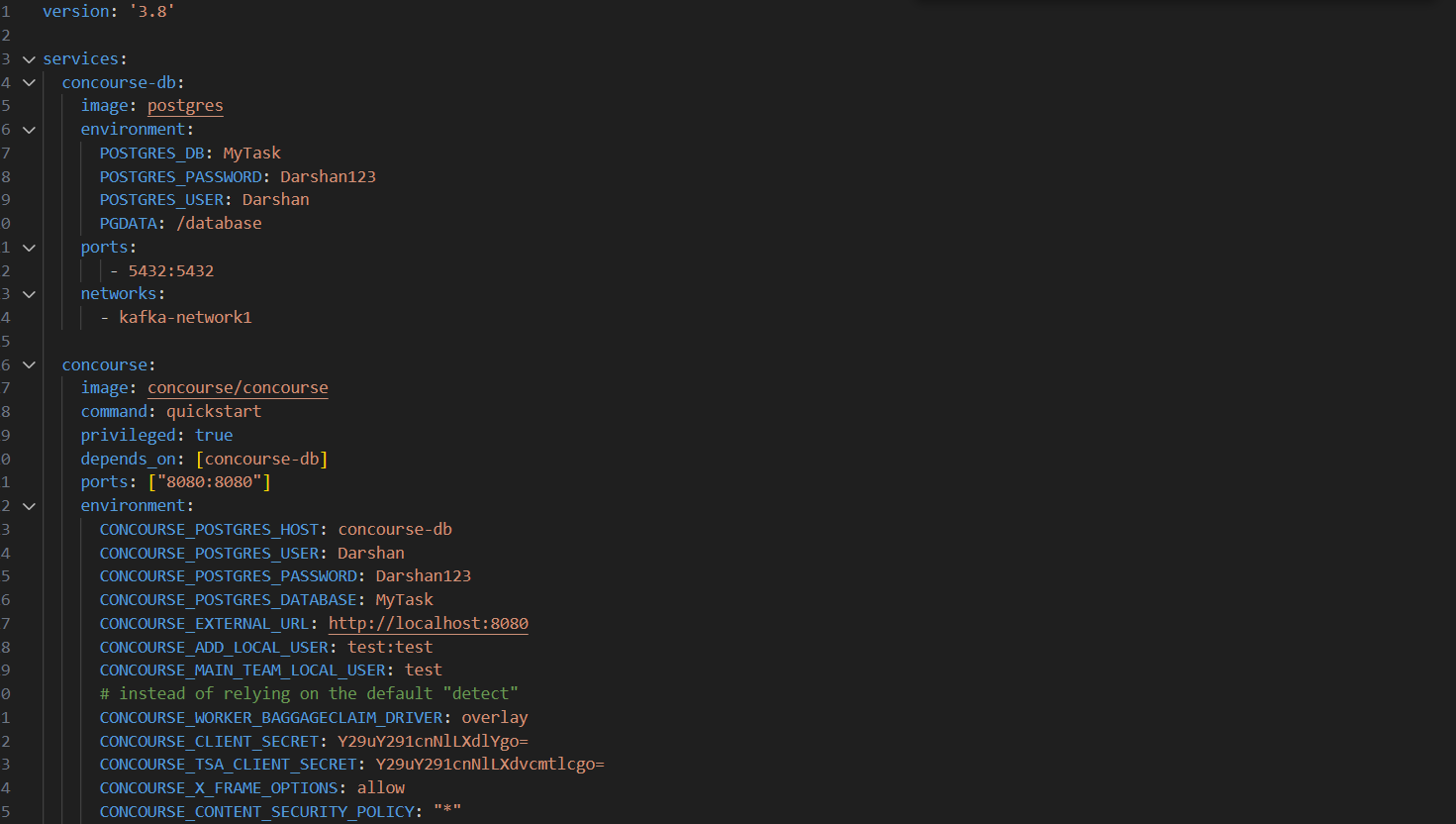
**Step 8: Loaded sink table data into powerbi.**

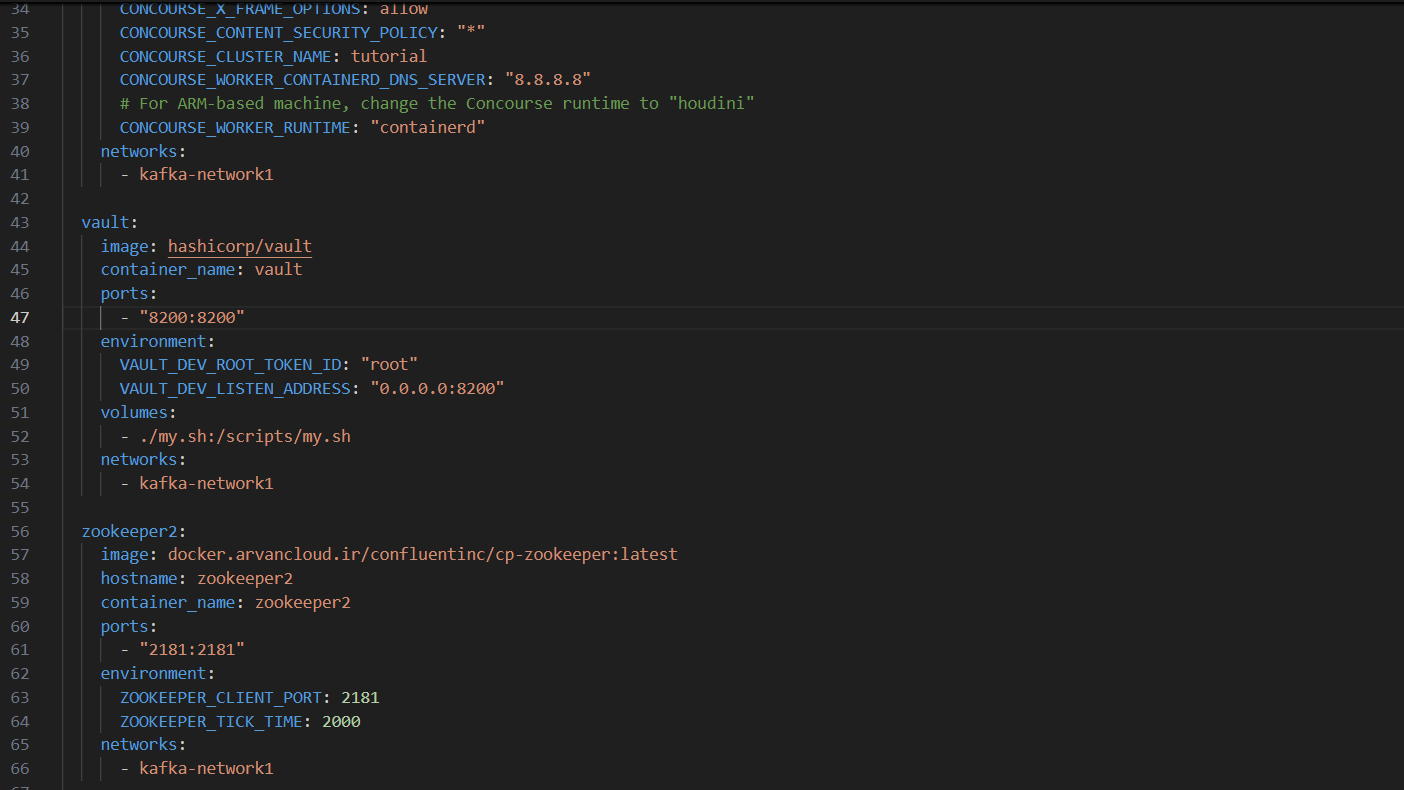
**Step 9: Made a dashboard on Pharmaceutical Industry profit And Loss.**

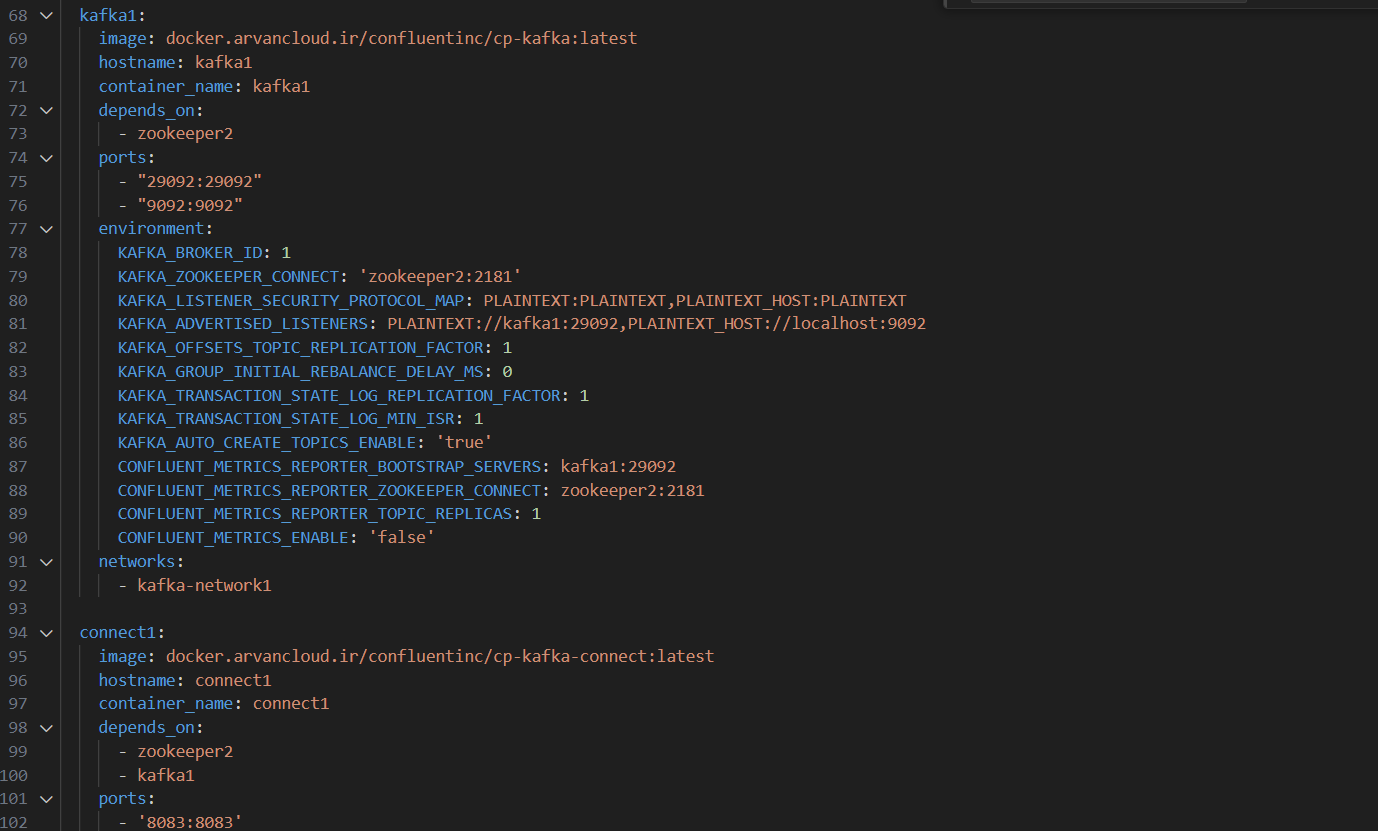
**Step1:**

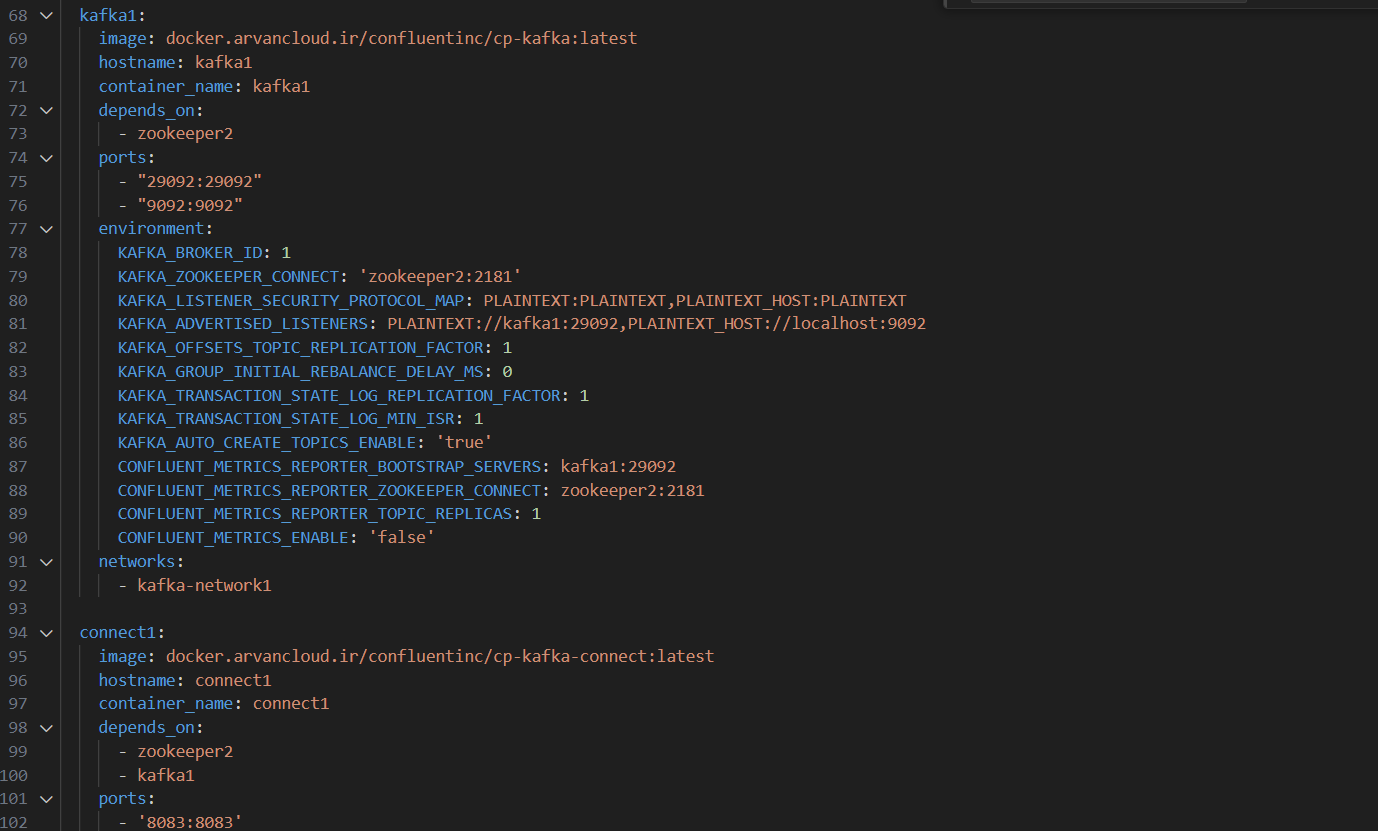
**Update the docker-compose file to making sure that kafka broker and connect, aws-glue attached to it and as images were already there so just used following query.**

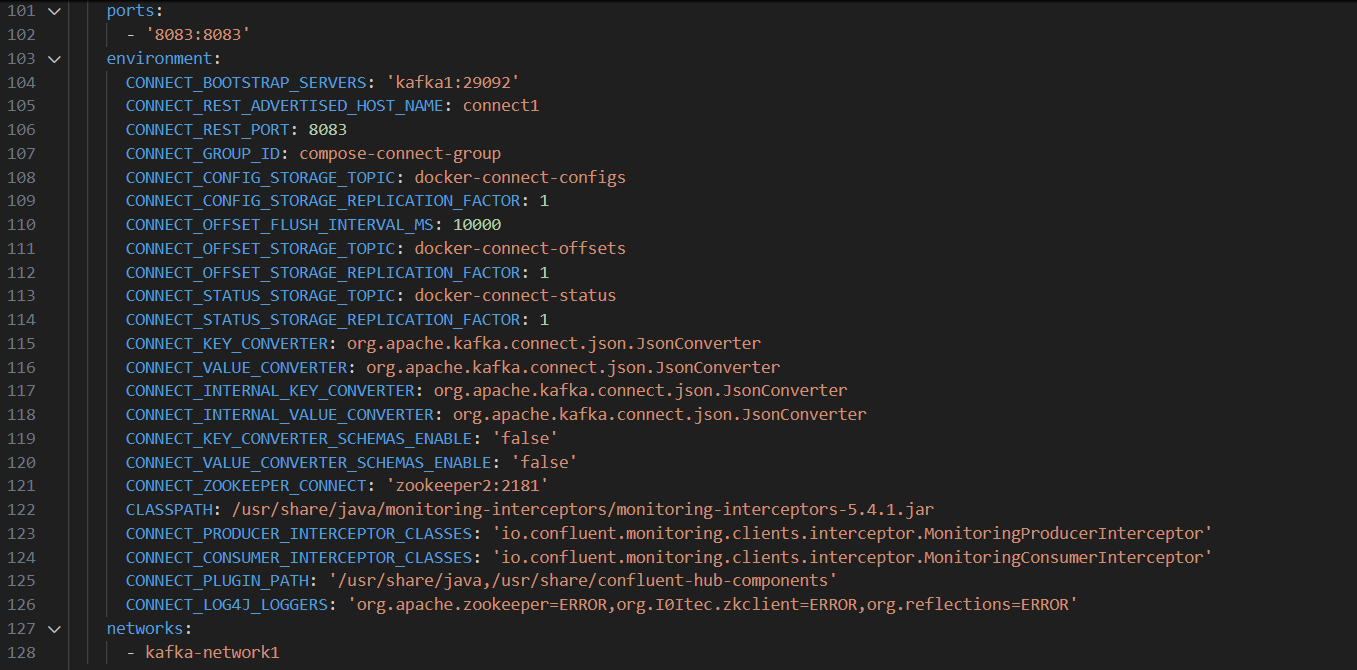
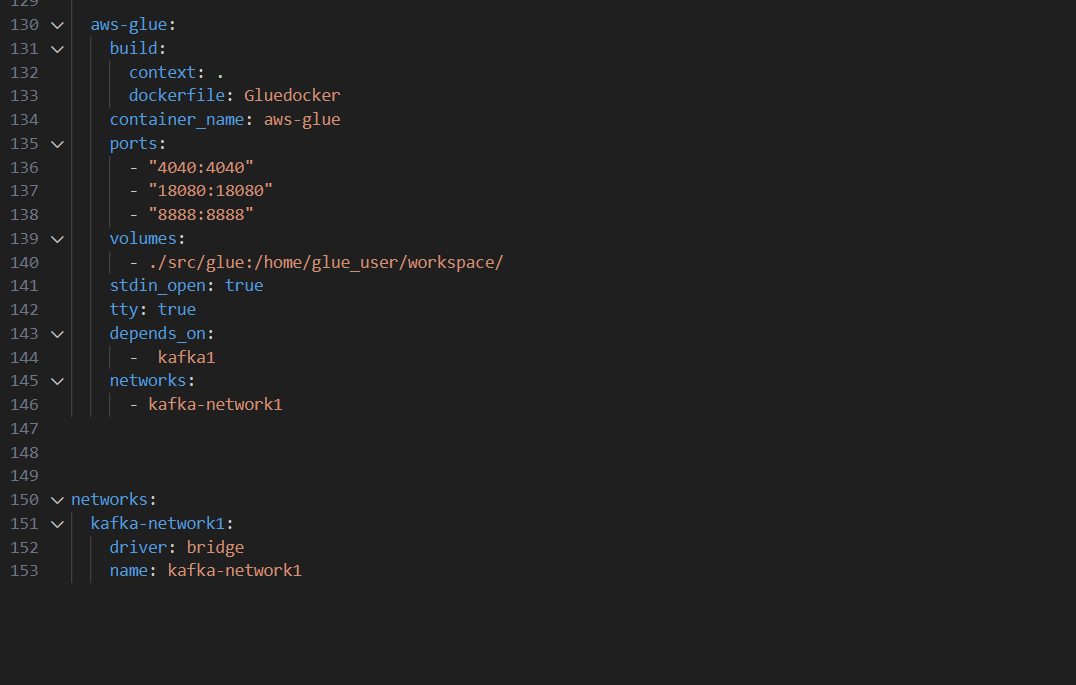
**Docker-compose:**

****

****

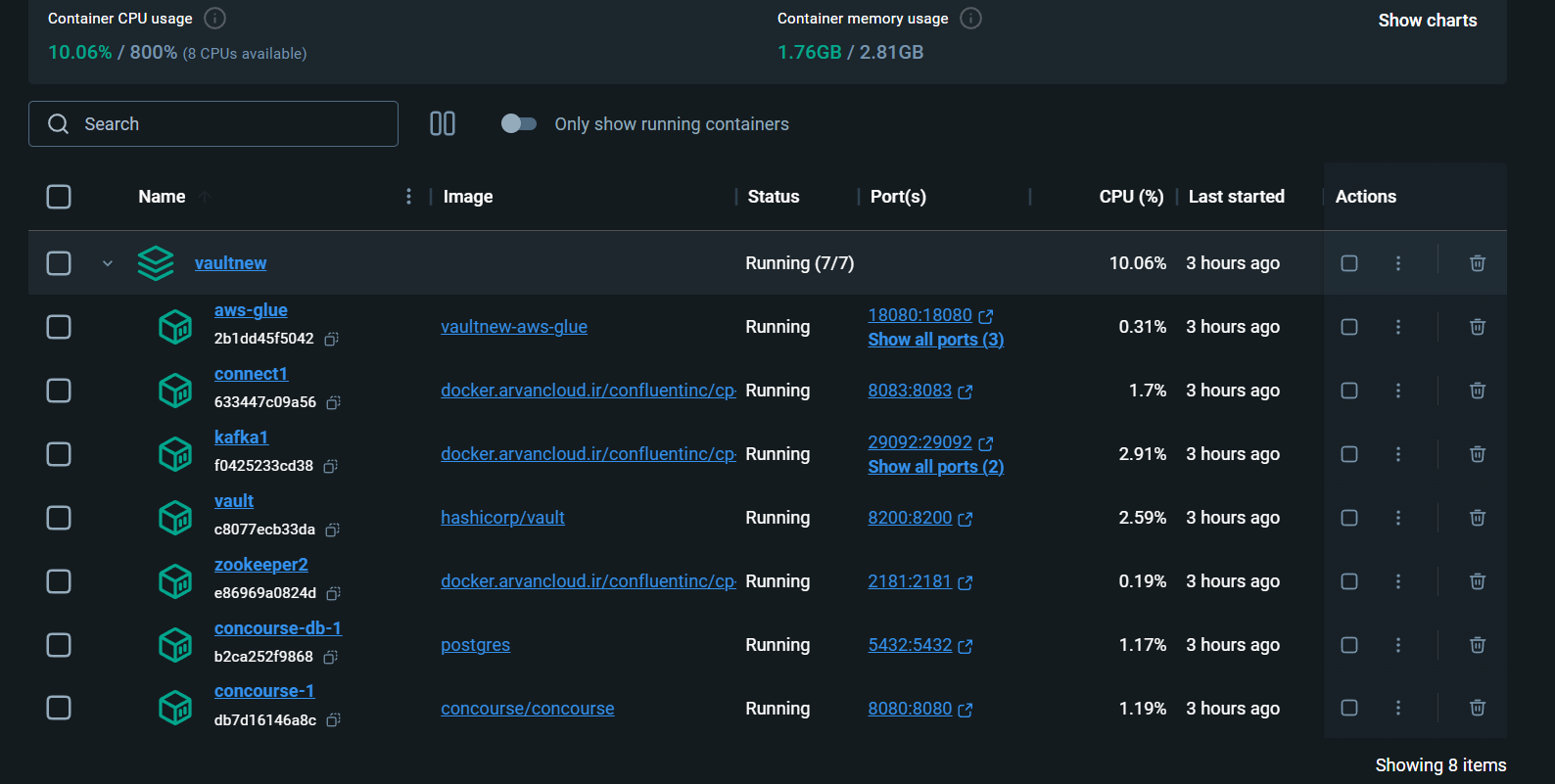




****

**docker-compose up -d –build**

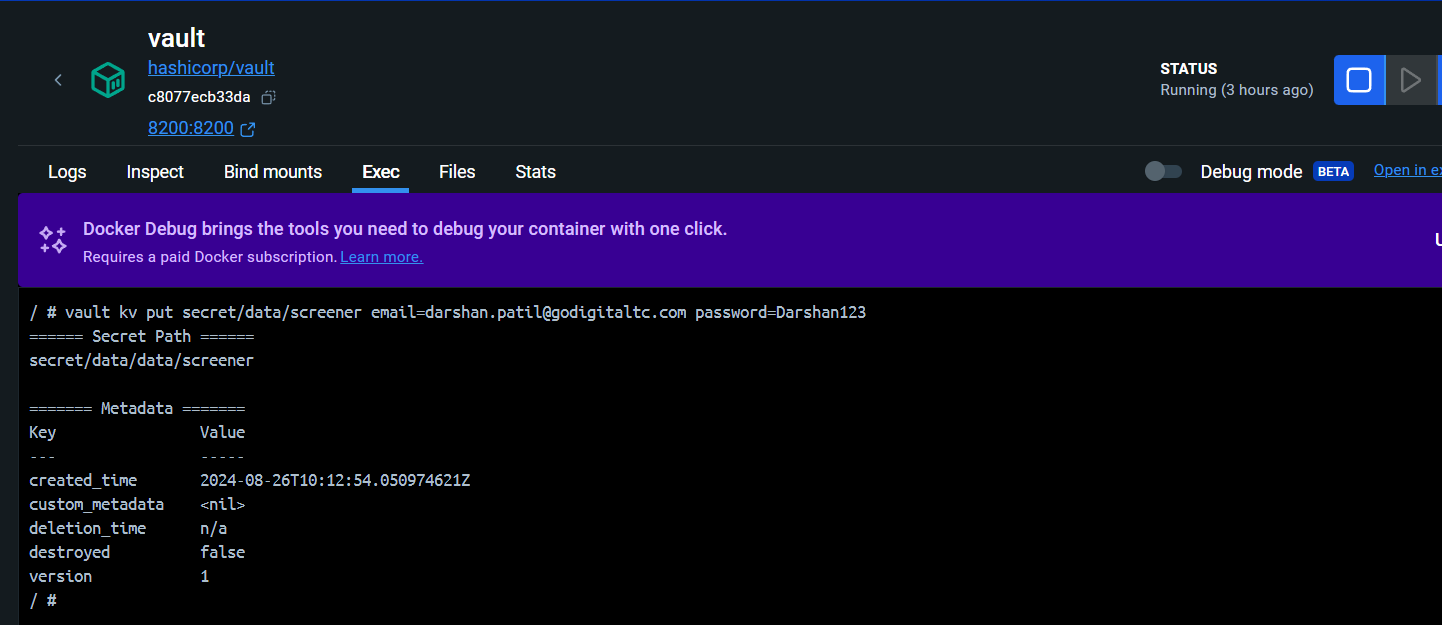
* **Use to rebuild your images in form of containers.**

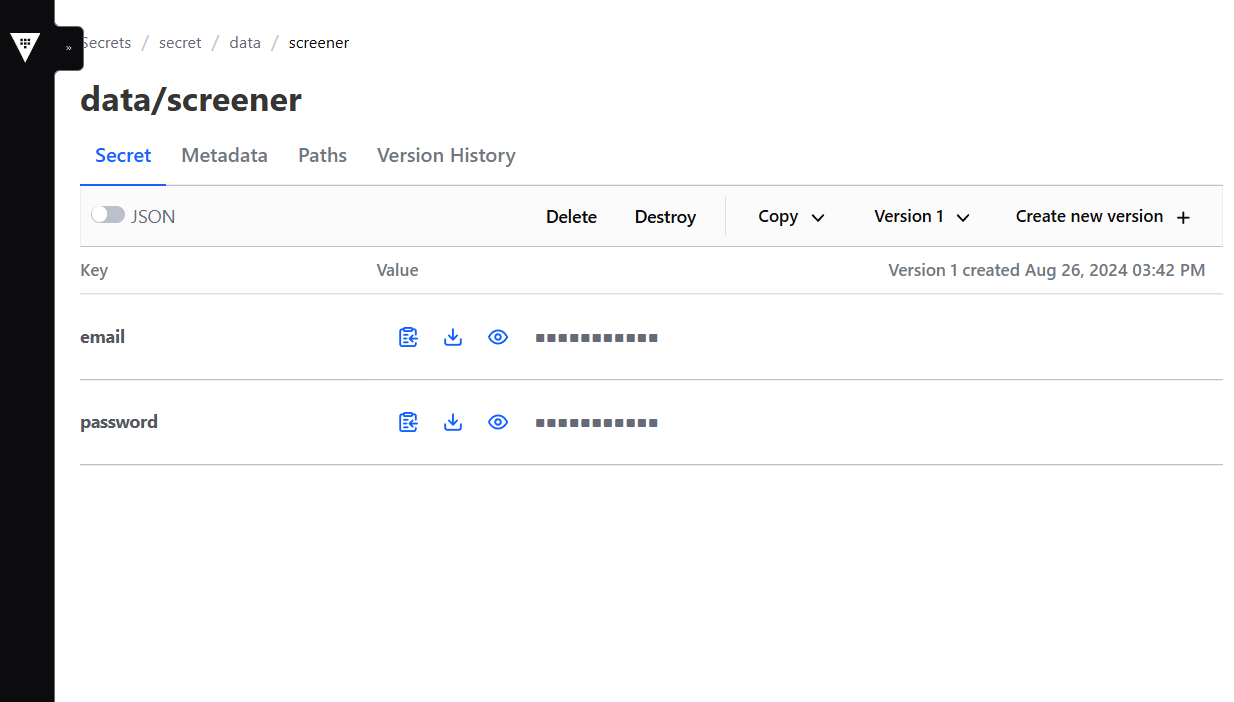
****

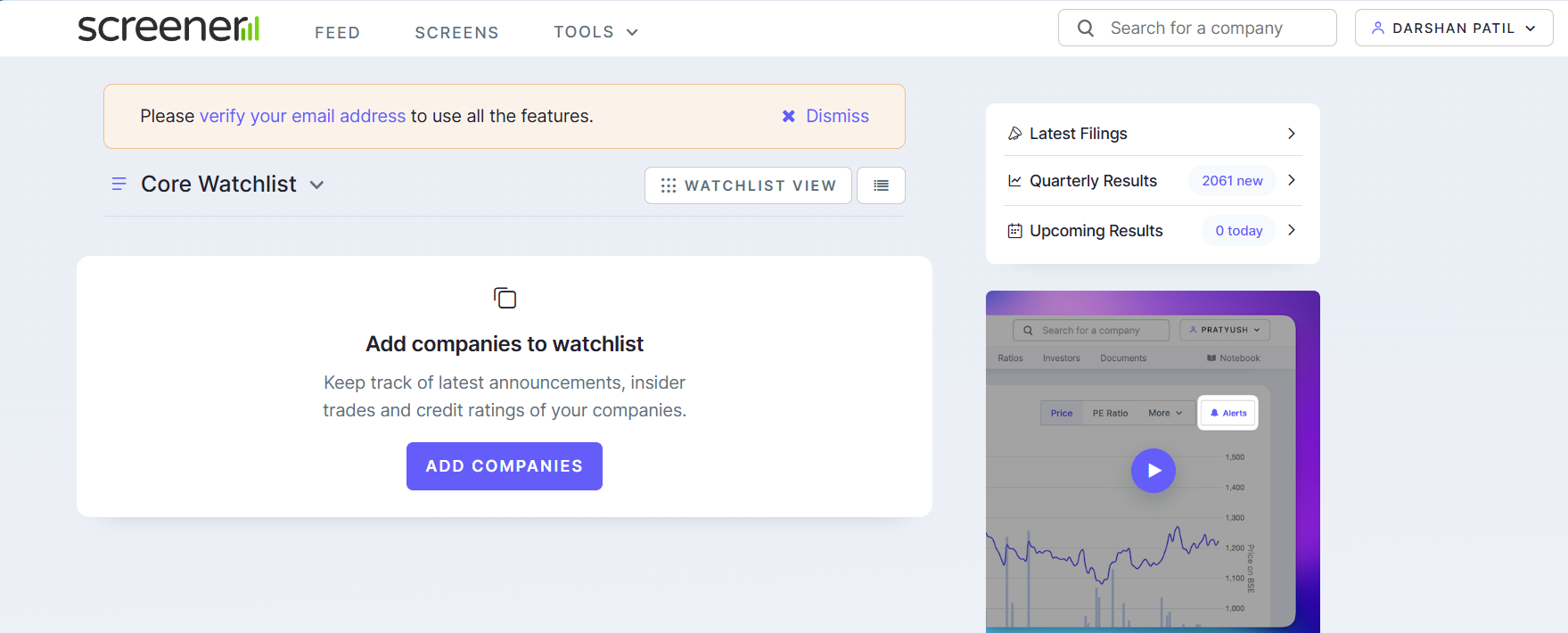
**Step 2:**

**Setup Vault connection:**

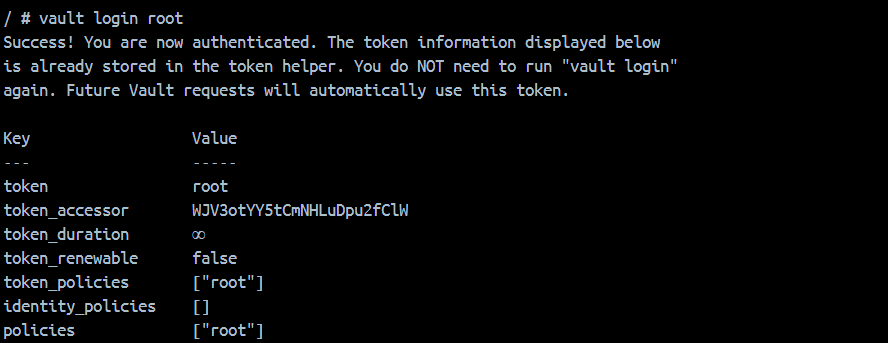
**Made a Vault Container and login into it with credentials and opened the port to url.**

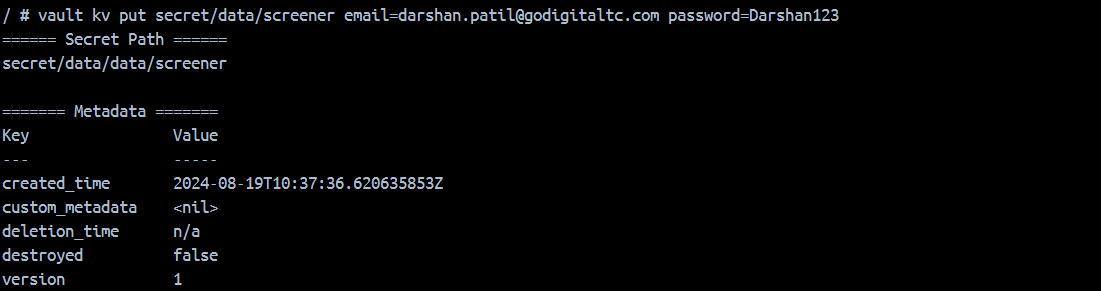
****

****

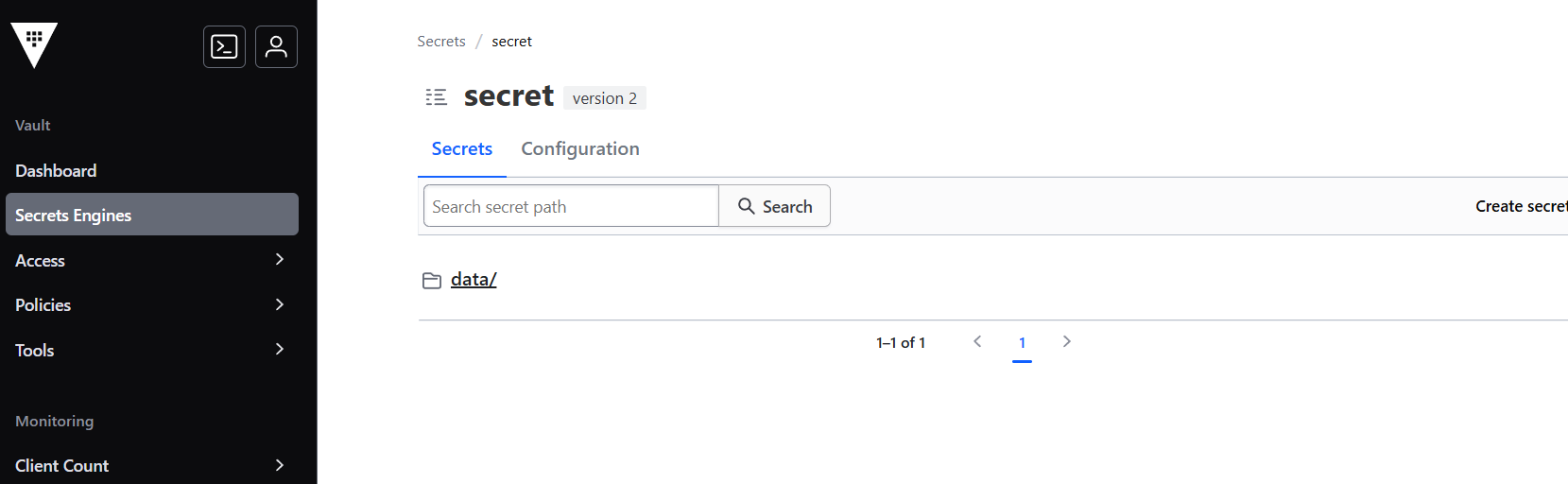


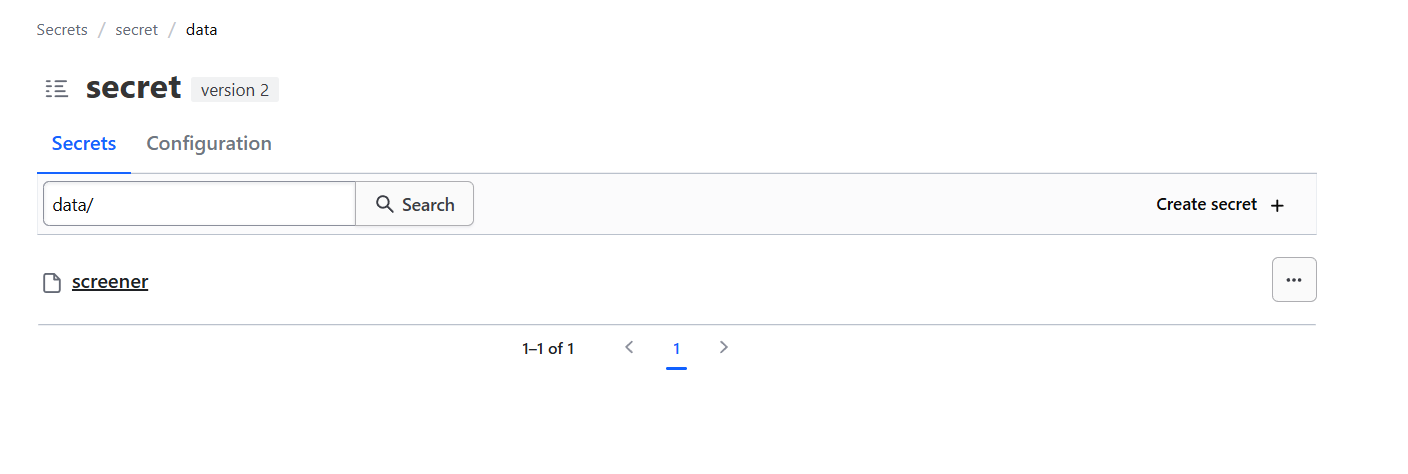
**Now come again in vault shell:**





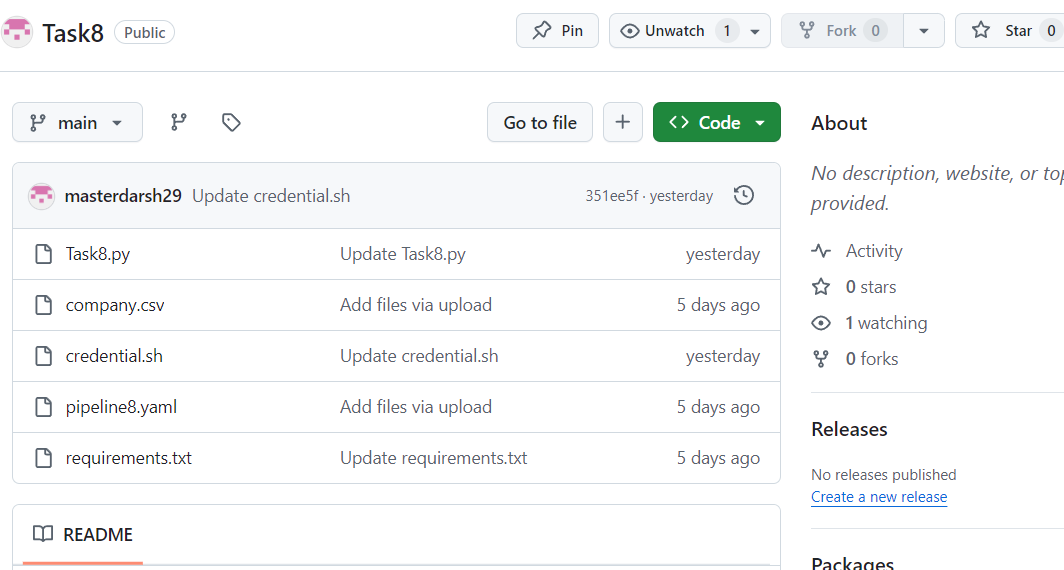
**As a result, I can see those credentials into vault:**



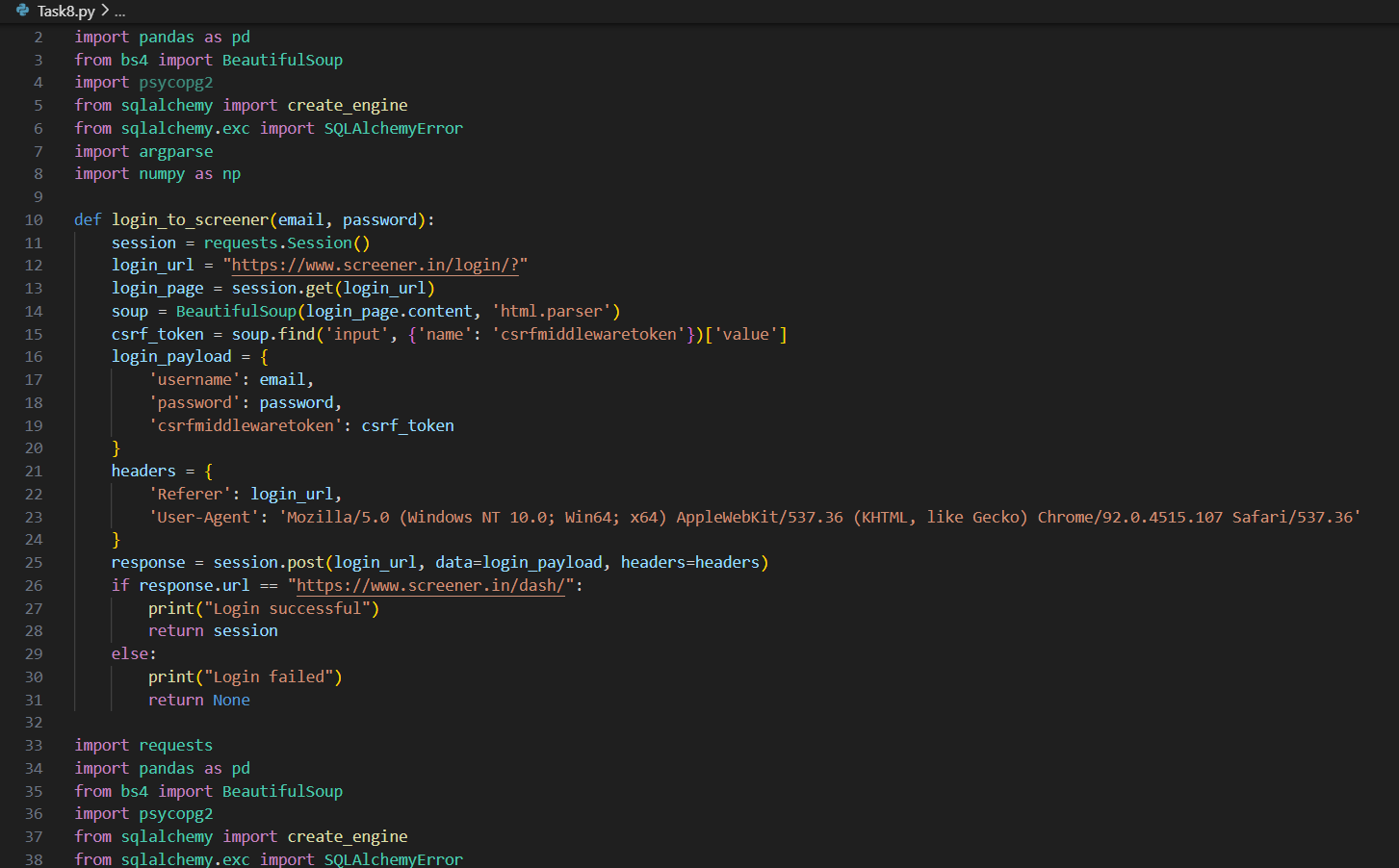


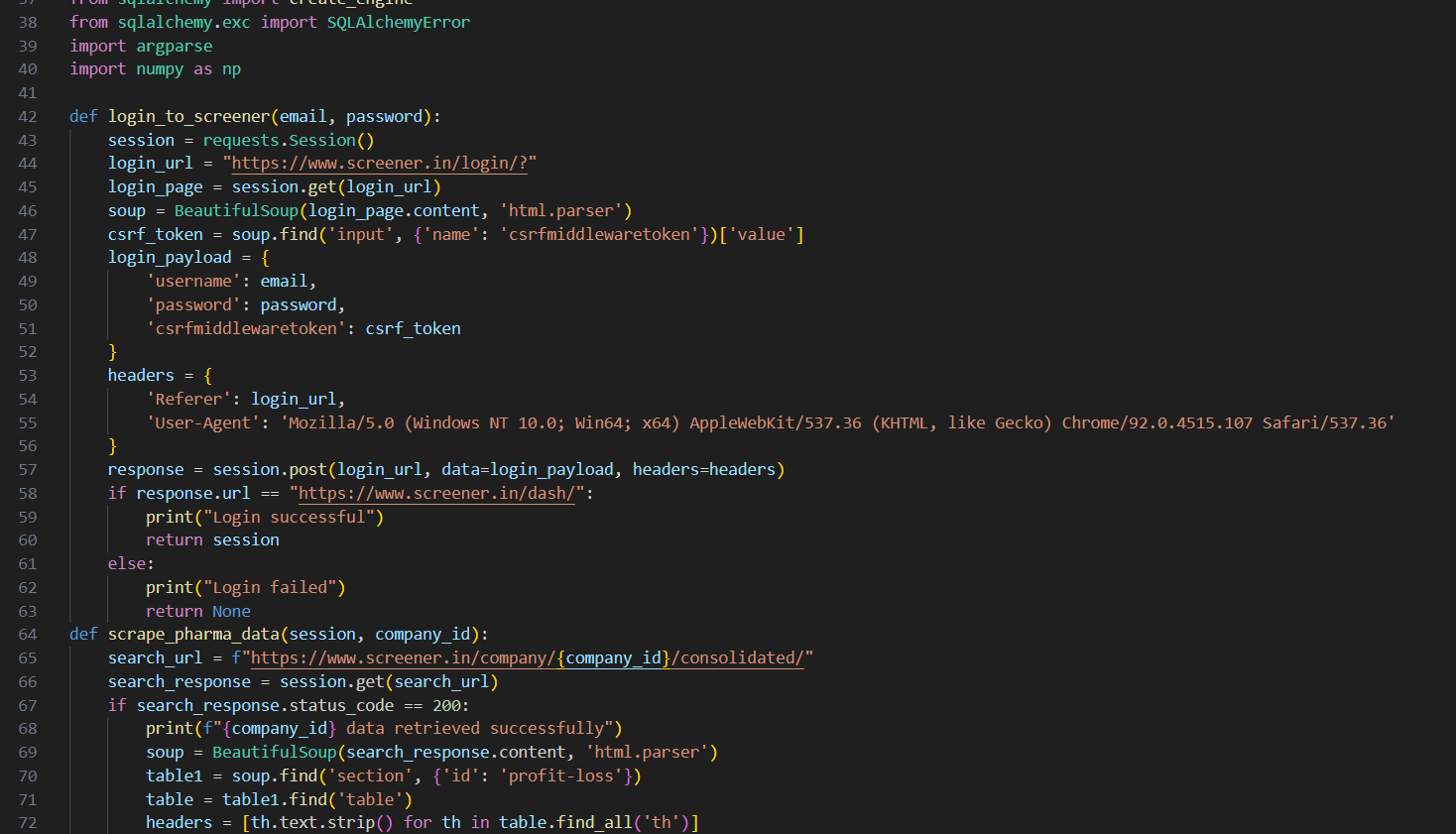
**Step 3:**

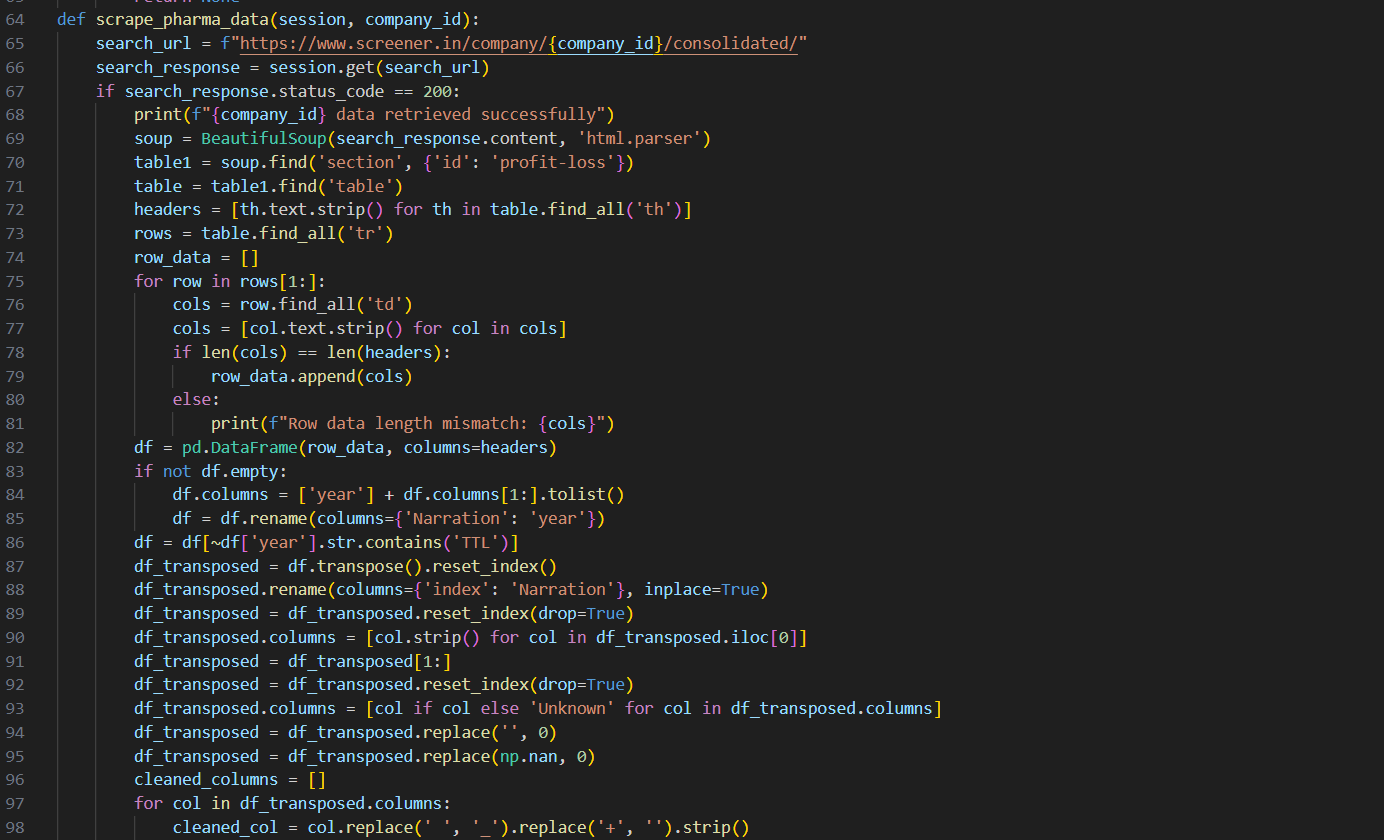
**Create scrapping python script, pipelining, git-repo to store it.**

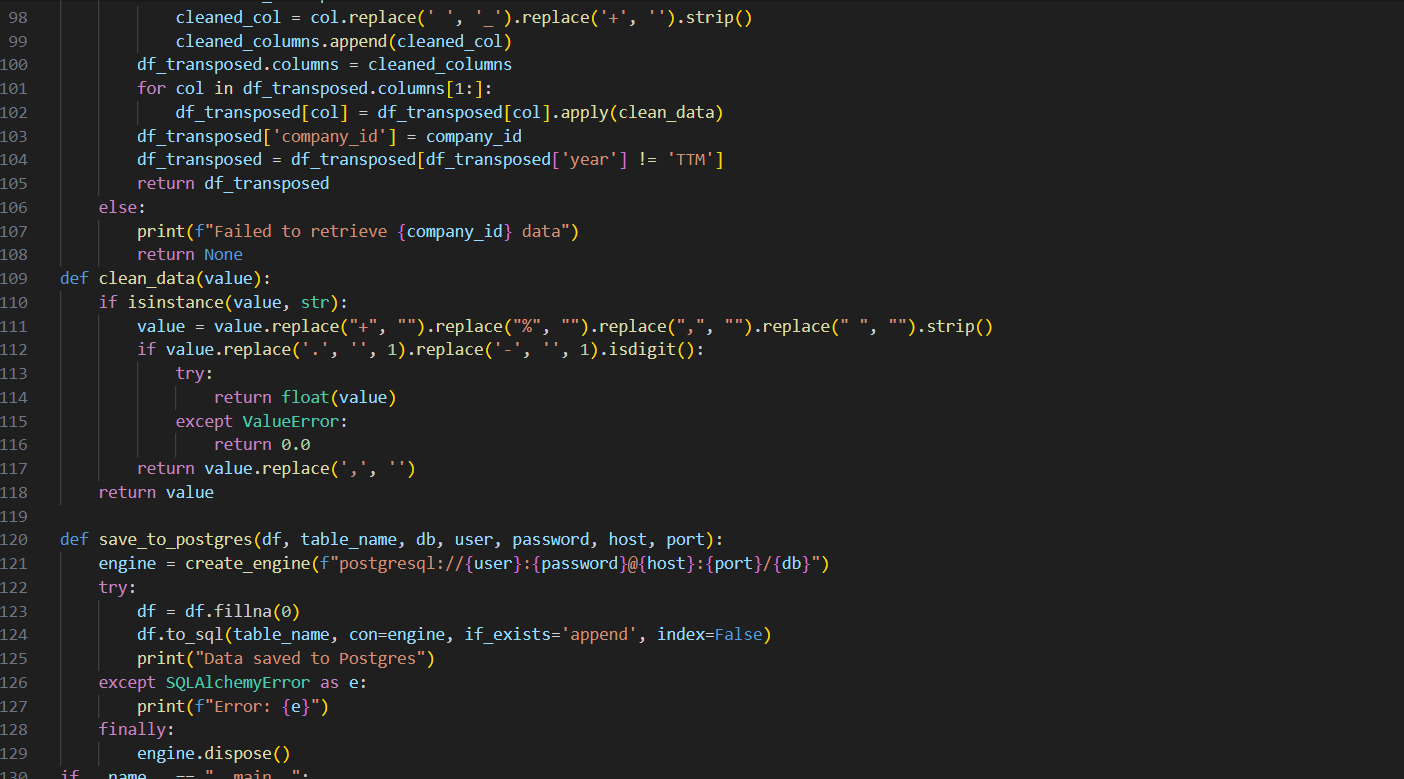


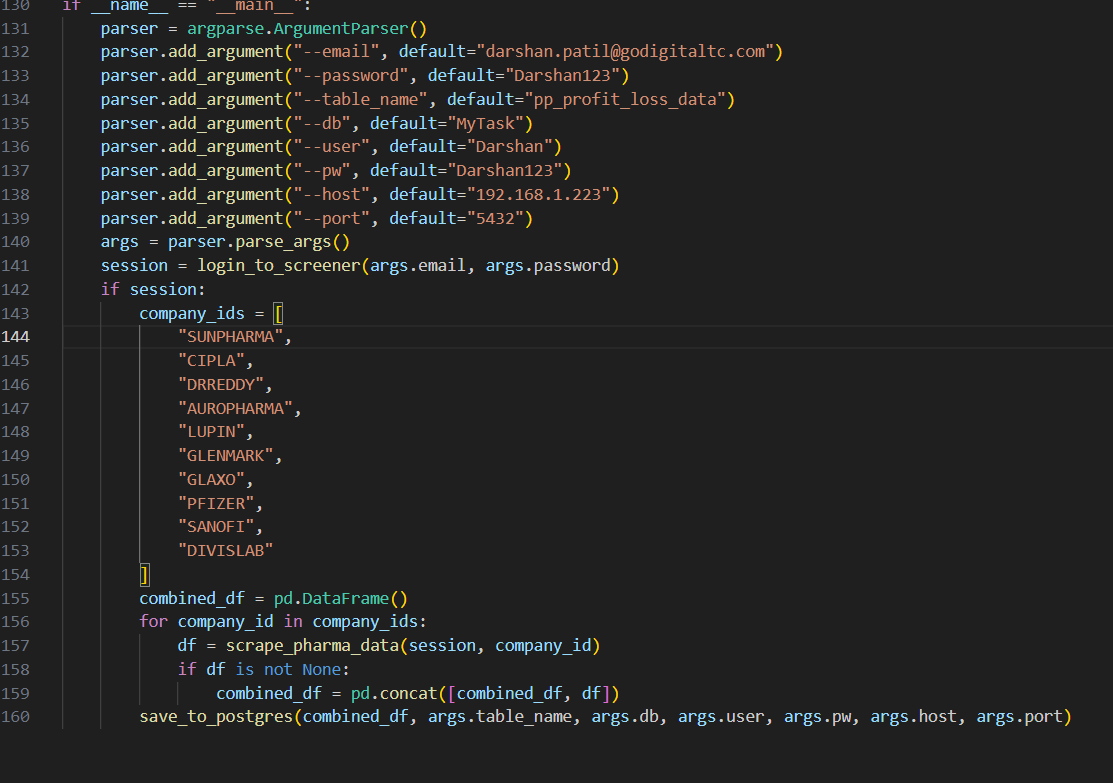
**Scrapping file:**

****

****

****

****

****

This script is designed to:

1. Login to a website (<https://www.screener.in/>) using email and password.
2. Scrape pharmaceutical company data from the website.
3. Clean and transform the scraped data.
4. Save the cleaned data to a PostgreSQL database.

Here's a line-by-line explanation:

Login Function

1. def login\_to\_screener(email, password): - Defines a function to login to the website.
2. session = requests.Session() - Creates a new session for sending HTTP requests.
3. login\_url = "https://www.screener.in/login/?" - Specifies the login URL.
4. login\_page = session.get(login\_url) - Sends a GET request to the login URL.
5. soup = BeautifulSoup(login\_page.content, 'html.parser') - Parses the HTML content of the login page.
6. csrf\_token = soup.find('input', {'name': 'csrfmiddlewaretoken'})['value'] - Extracts the CSRF token from the HTML.
7. login\_payload = {...} - Creates a dictionary with login credentials and CSRF token.
8. response = session.post(login\_url, data=login\_payload, headers=headers) - Sends a POST request to the login URL with login credentials.
9. if response.url == "https://www.screener.in/dash/": - Checks if the login was successful by verifying the redirect URL.

Scrape Pharma Data Function

1. def scrape\_pharma\_data(session, company\_id): - Defines a function to scrape pharmaceutical company data.
2. search\_url = f"https://www.screener.in/company/{company\_id}/consolidated/" - Specifies the URL for scraping company data.
3. search\_response = session.get(search\_url) - Sends a GET request to the company data URL.
4. soup = BeautifulSoup(search\_response.content, 'html.parser') - Parses the HTML content of the company data page.
5. table1 = soup.find('section', {'id': 'profit-loss'}) - Finds the profit-loss table section.
6. table = table1.find('table') - Finds the profit-loss table.
7. headers = [th.text.strip() for th in table.find\_all('th')] - Extracts table headers.
8. rows = table.find\_all('tr') - Finds all table rows.
9. row\_data = [] - Initializes an empty list to store row data.
10. for row in rows[1:]: - Loops through each table row (skipping the header row).
11. cols = row.find\_all('td') - Finds all table cells in the current row.
12. cols = [col.text.strip() for col in cols] - Extracts and strips text from each table cell.
13. if len(cols) == len(headers): - Checks if the number of cells matches the number of headers.
14. row\_data.append(cols) - Appends the row data to the list.
15. df = pd.DataFrame(row\_data, columns=headers) - Creates a Pandas DataFrame from the row data.

Clean Data Function

1. def clean\_data(value): - Defines a function to clean and transform data.
2. if isinstance(value, str): - Checks if the value is a string.
3. value = value.replace("+", "").replace("%", "").replace(",", "").replace(" ", "").strip() - Removes unwanted characters from the string.
4. if value.replace('.', '', 1).replace('-', '', 1).isdigit(): - Checks if the string is a number.
5. try: return float(value) - Tries to convert the string to a float.
6. except ValueError: return 0.0 - Returns 0.0 if the conversion fails.

Save to Postgres Function

1. def save\_to\_postgres(df, table\_name, db, user, password, host, port): - Defines a function to save data to a PostgreSQL database.
2. engine = create\_engine(f"postgresql://{user}:{password}@{host}:{port}/{db}") - Creates a PostgreSQL engine.
3. try: df.to\_sql(table\_name, con=engine, if\_exists='append', index=False) - Saves the DataFrame to the PostgreSQL database.
4. except SQLAlchemyError as e: print(f"Error: {e}") - Catches any SQLAlchemy errors.
5. finally: engine.dispose() - Disposes of the PostgreSQL engine.

Main Script

1. if \_\_name\_\_ == "\_\_main\_\_": - Checks if the script is being run directly.
2. parser = argparse.ArgumentParser() - Creates an argument parser.
3. parser.add\_argument(...) - Adds arguments for email, password, table name, database, user, password, host, and port.
4. args = parser.parse\_args() - Parses the command-line arguments.
5. session = login\_to\_screener(args.email, args.password) - Logs in to the website using the provided email and password.
6. if session: - Checks if the login was successful.
7. company\_ids = [...] - Specifies a list of company IDs to scrape data for.
8. combined\_df = pd.DataFrame() - Initializes an empty DataFrame to store combined data.
9. for company\_id in company\_ids: - Loops through each company ID.
10. df = scrape\_pharma\_data(session, company\_id) - Scrapes data for the current company ID.
11. if df is not None: - Checks if data was scraped successfully.
12. combined\_df = pd.concat([combined\_df, df]) - Appends the scraped data to the combined DataFrame.
13. save\_to\_postgres(combined\_df, args.table\_name, args.db, args.user, args.pw, args.host, args.port) - Saves the combined data to the PostgreSQL database.
14. Explanation of the Script's Flow
15. The script logs in to the website using the provided email and password.
16. It then loops through a list of company IDs and scrapes data for each company.
17. The scraped data is cleaned and transformed using the clean\_data function.
18. The cleaned data is then appended to a combined DataFrame.
19. Finally, the combined data is saved to a PostgreSQL database using the save\_to\_postgres function.

**Step 4:**

**Then use fly-commands to start the concourse pipelining:**

* **commands**

**fly -t tutorial login -c http://localhost:8080 -u test -p test**

**As followed by   
destroy:**

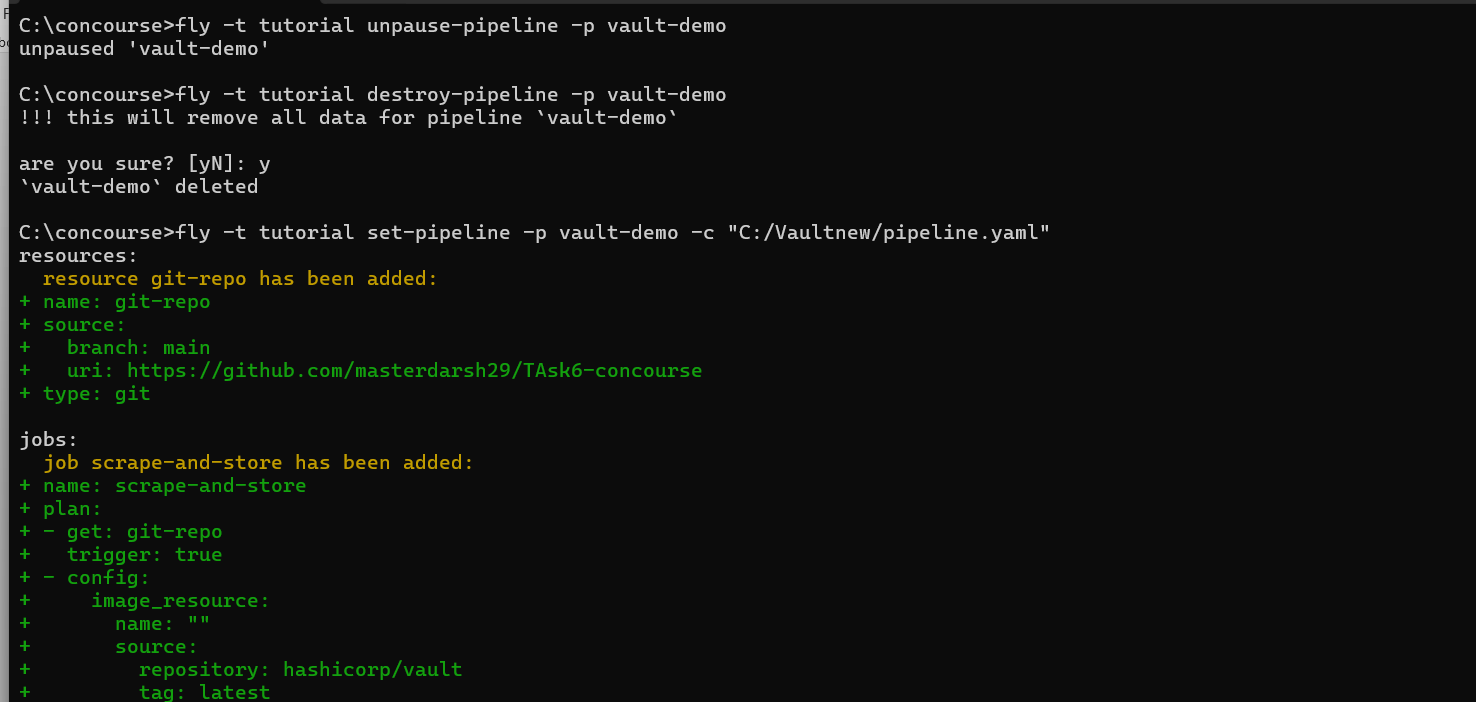
**fly -t tutorial destroy-pipeline -p vault-demo**

**set:**

**fly -t tutorial set-pipeline -p vault-demo -c “C:/Vaultnew/pipeline.yaml”**

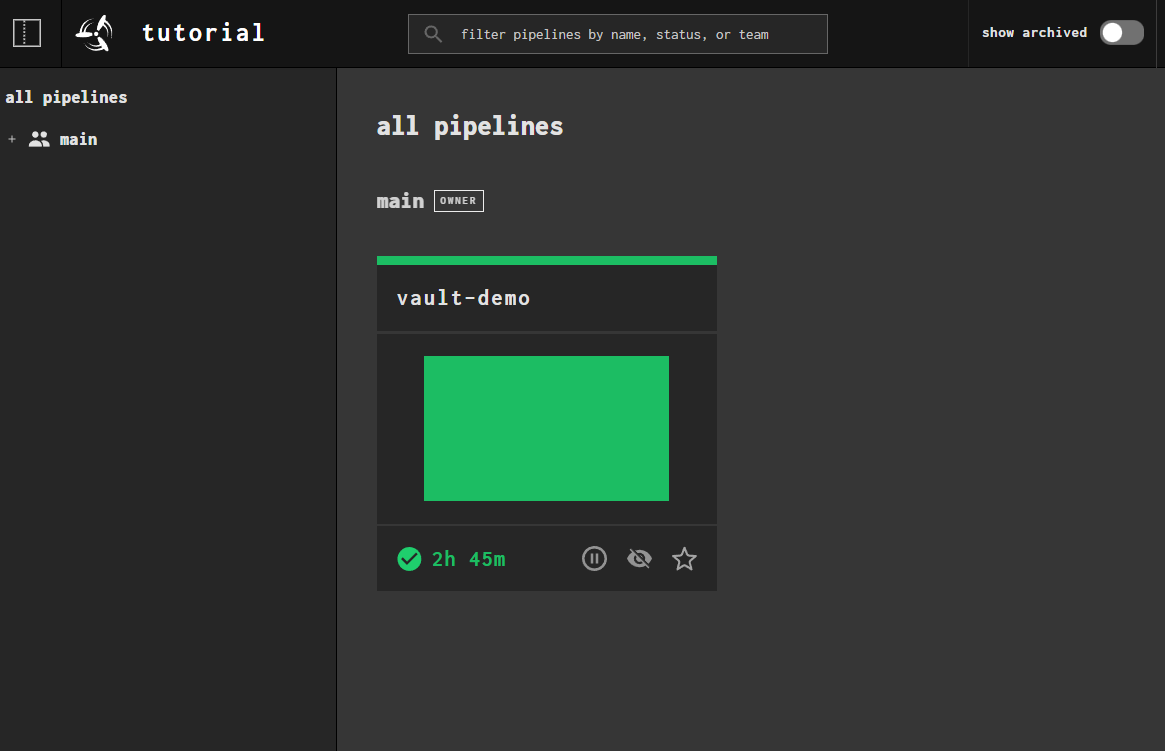
**unpause:**

**fly -t tutorial unpause-pipeline -p vault-demo**

****

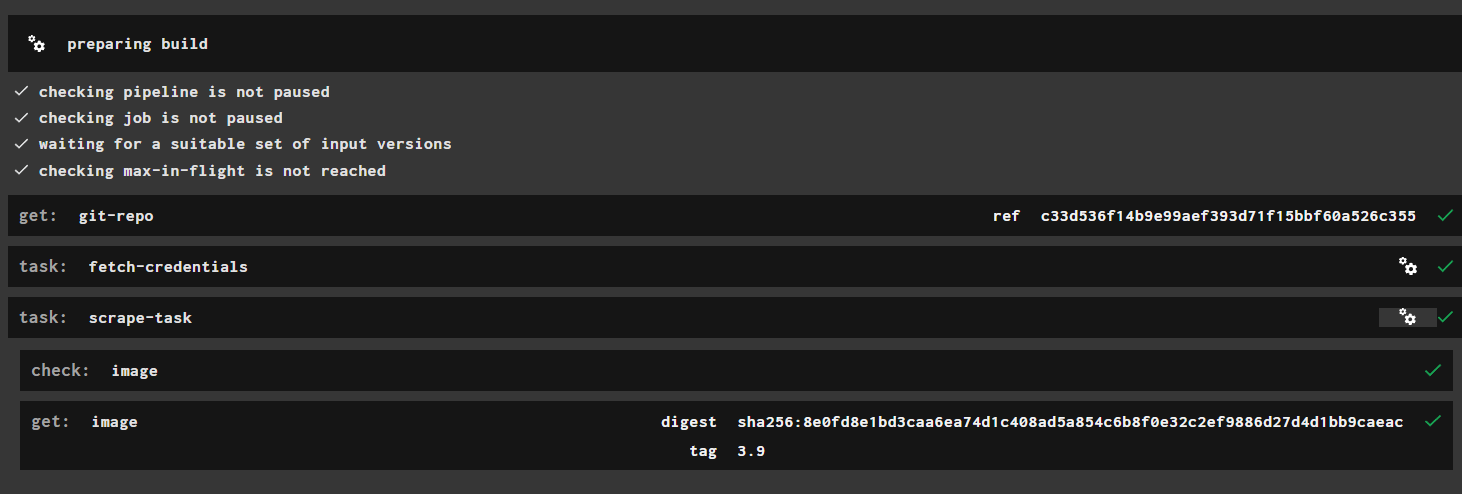
**Step5:**

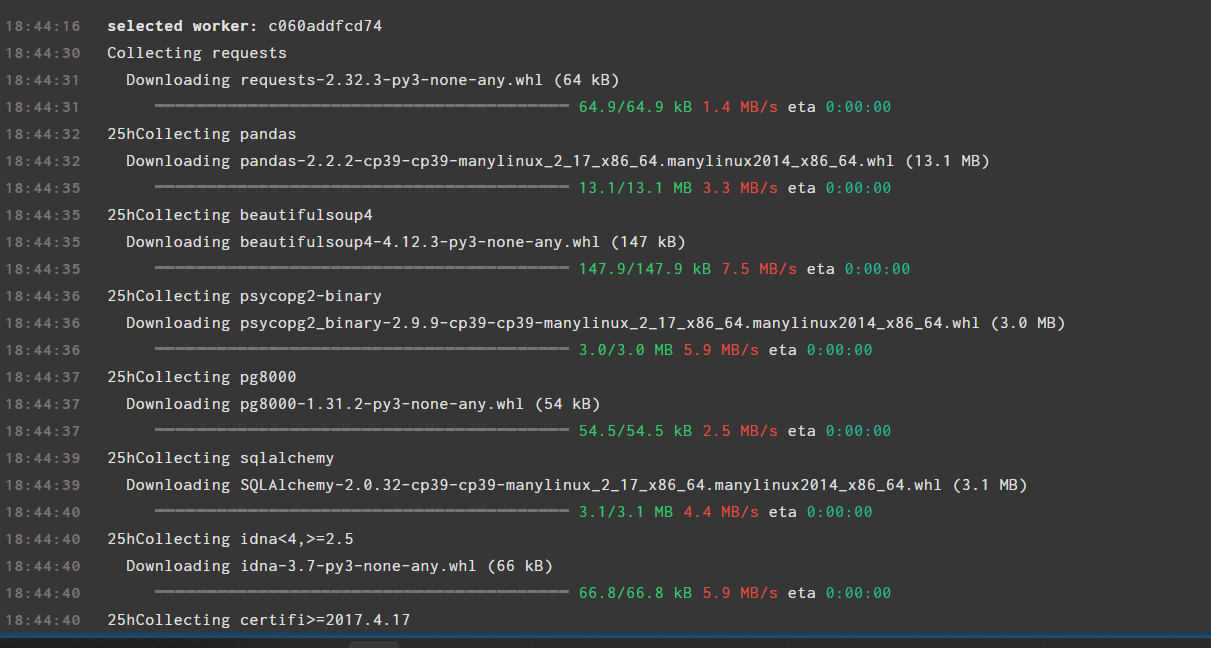
**Now open vault-demo, setup concourse piepline**

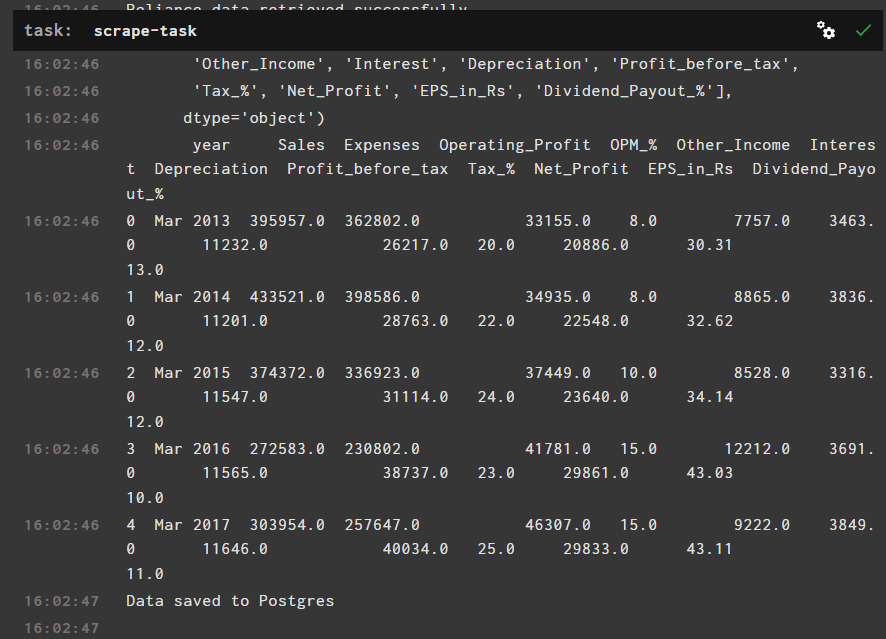




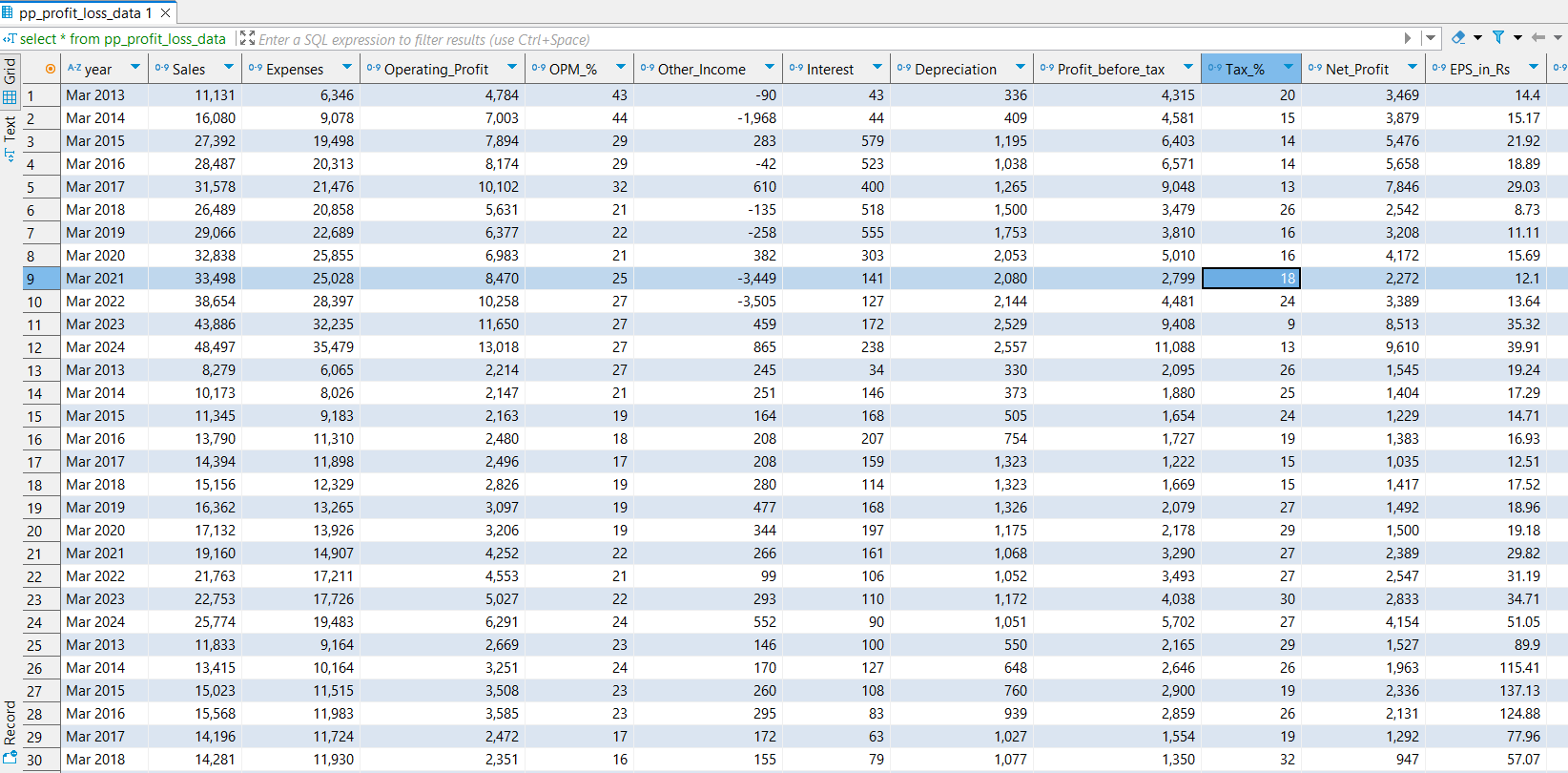
**Try to scrap-and-store your pipeline and fetch data into your postgres connection database.**





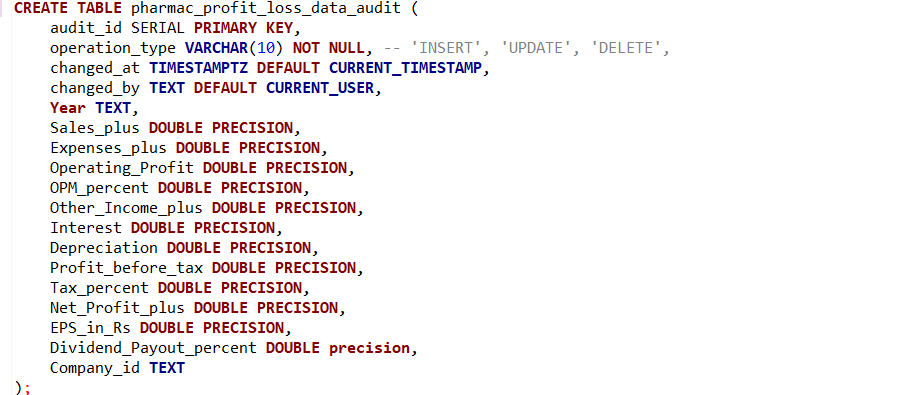


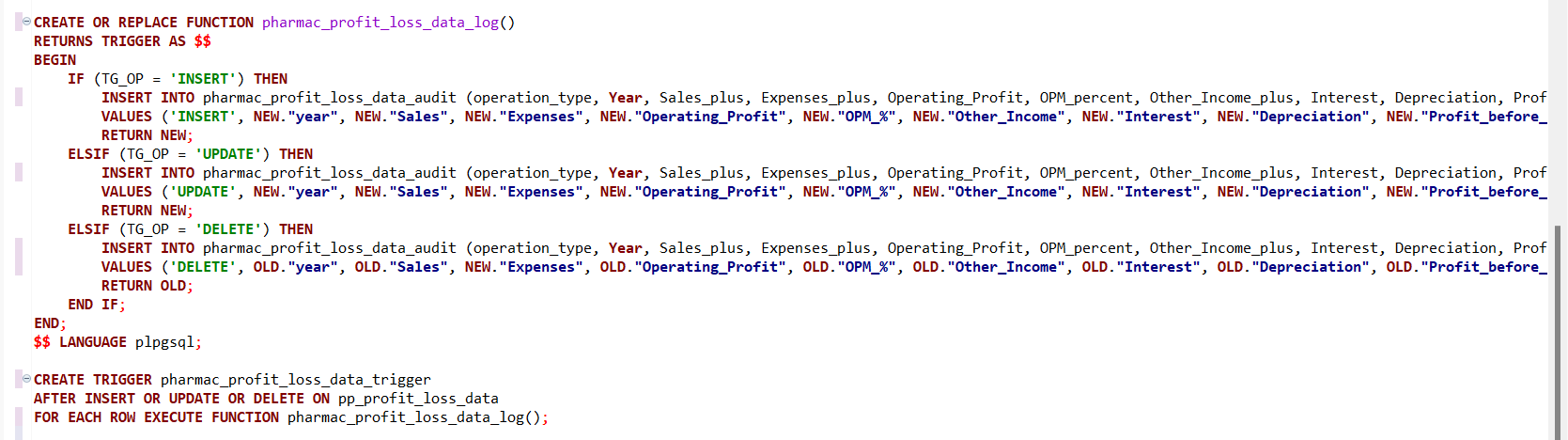
**As I can see data is stored into postgres:**

****

**Step 6:**

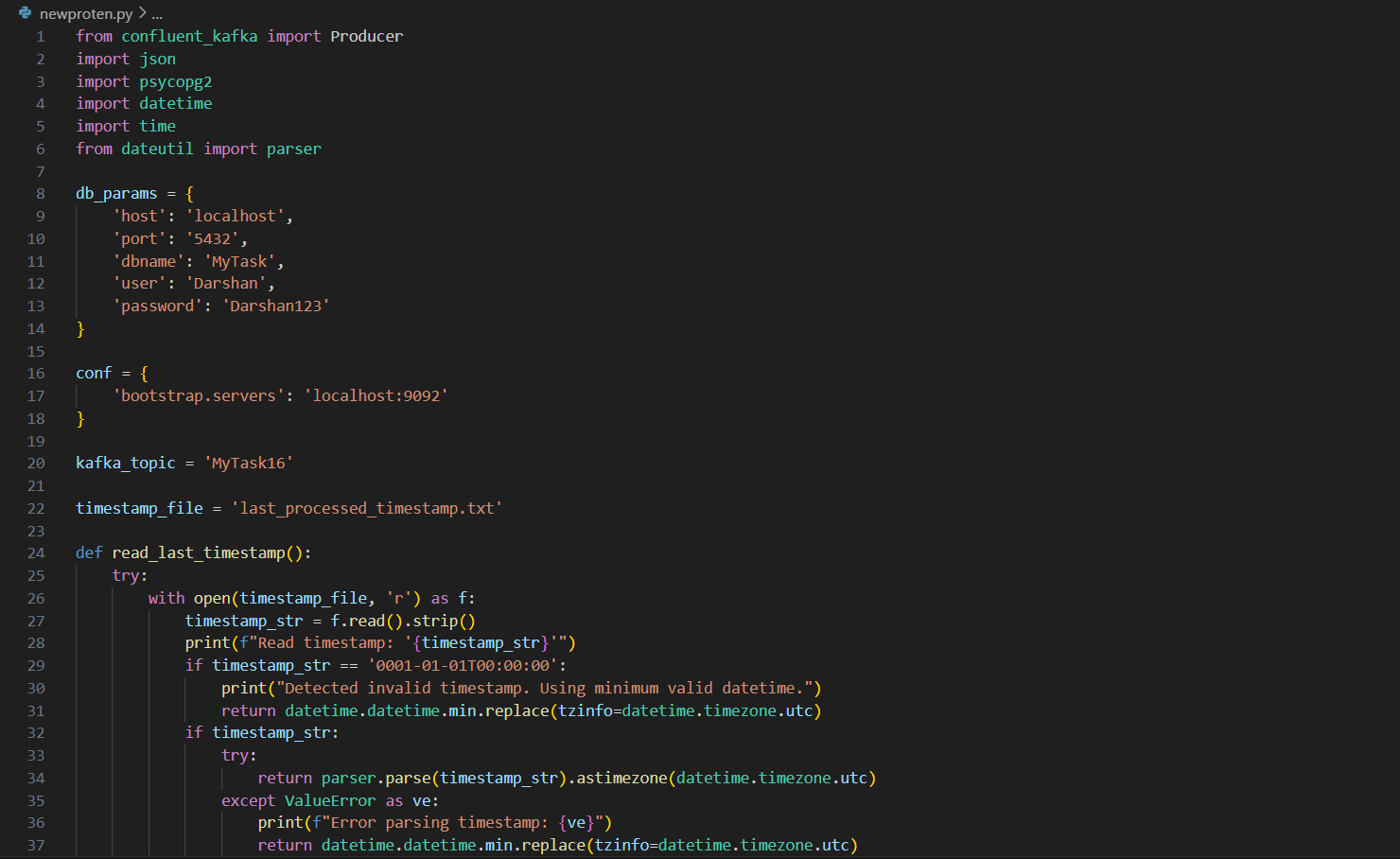
**Now, I have to make trigger along with audit table to see the data into topic and then in glue script will do transformations as I’m going to add some related columns with the use of ready columns.**

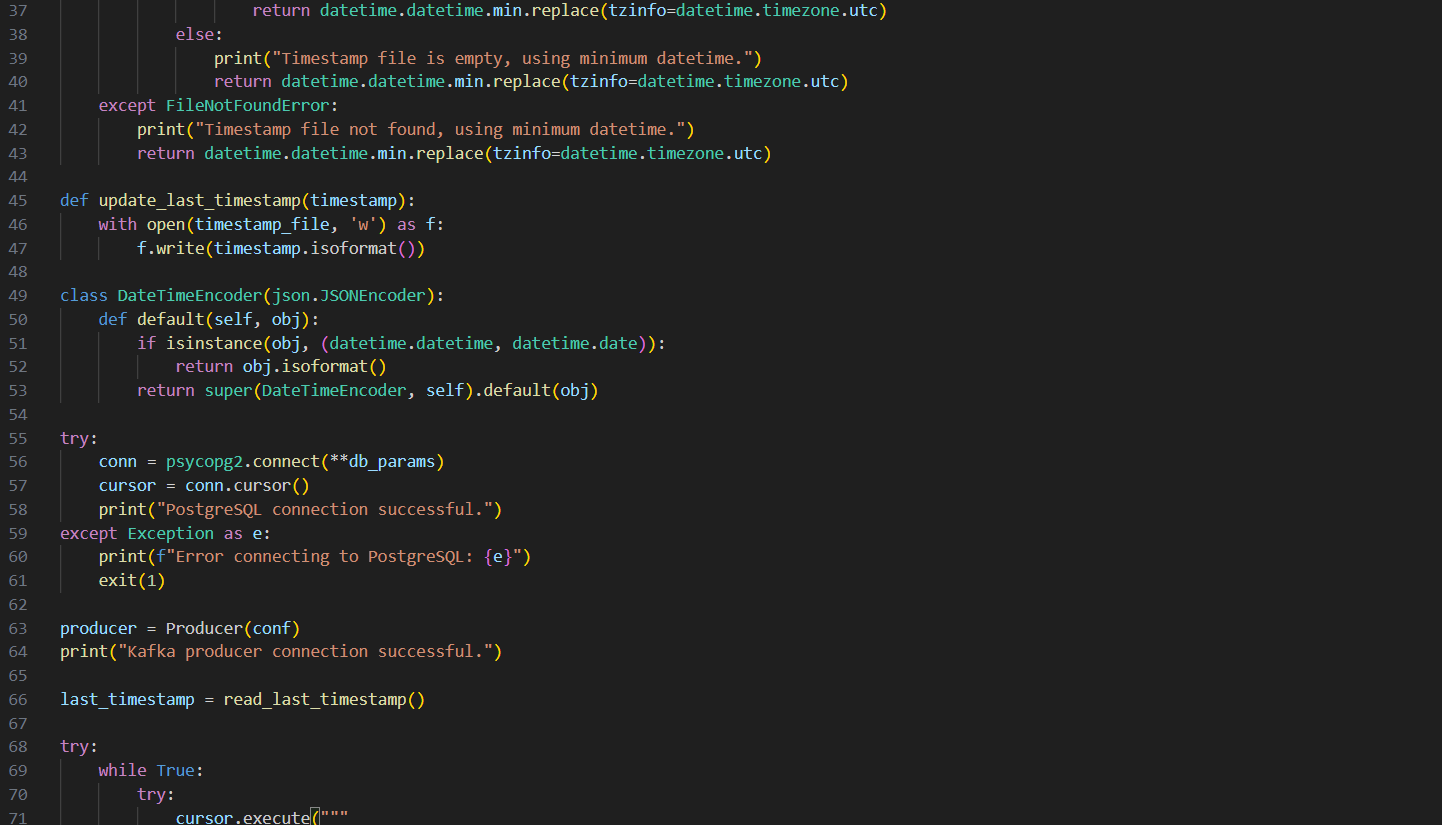
****

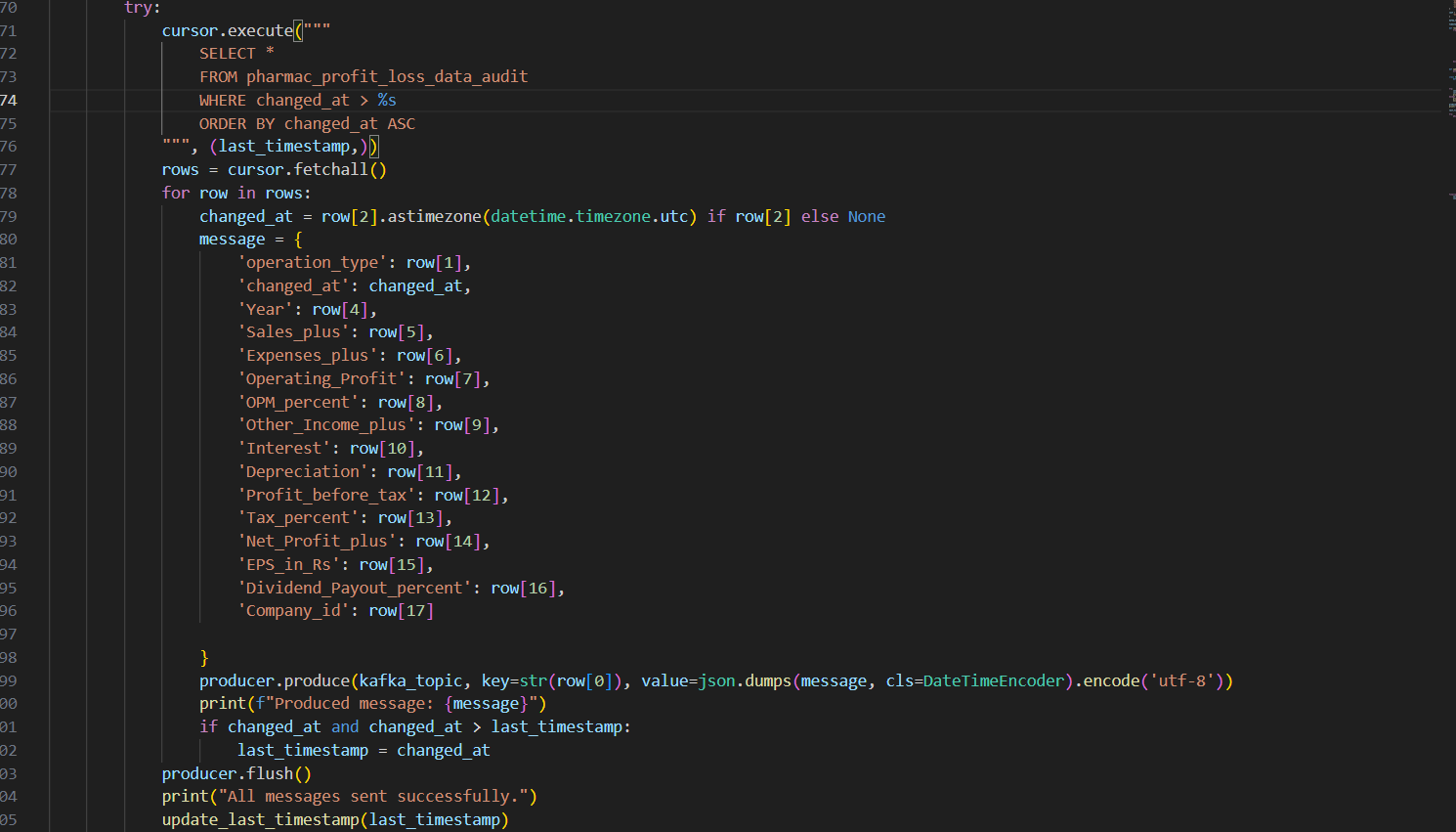
****

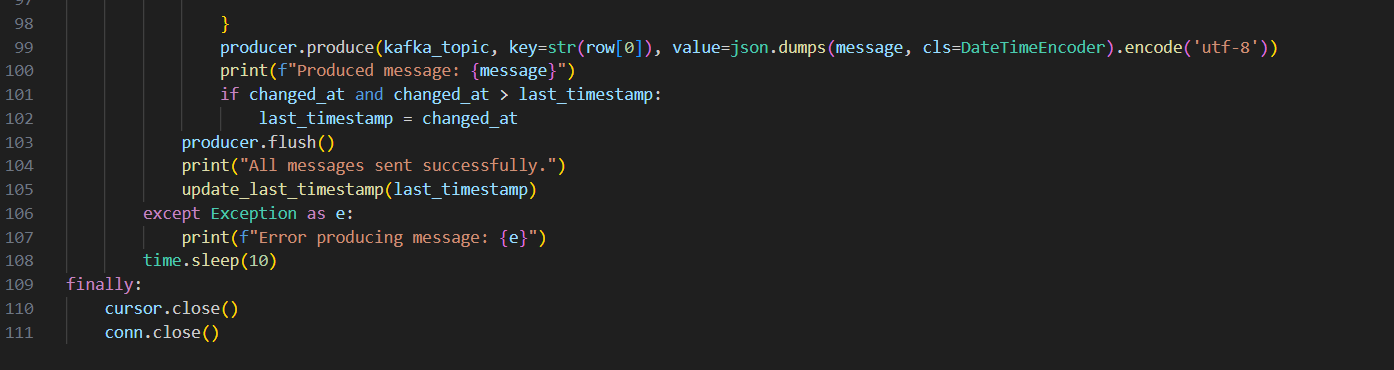
**Step 7:**

**Producer File (for getting data into topic)**

****

****

****

****

1. Connect to a PostgreSQL database and a Kafka server.
2. Read the last processed timestamp from a file.
3. Fetch rows from a PostgreSQL table (pharma\_profit\_loss\_data\_audit) where the timestamp is greater than the last processed timestamp.
4. Convert the fetched data into JSON format and produce it to a Kafka topic.
5. Update the last processed timestamp in the file.

Here's a detailed explanation of the script's components:

PostgreSQL Connection

* The script connects to a PostgreSQL database using the psycopg2 library.
* The database connection parameters are stored in the db\_params dictionary.

Kafka Producer

* The script creates a Kafka producer using the confluent\_kafka library.
* The Kafka producer configuration is stored in the conf dictionary.
* The script produces messages to a Kafka topic named MyTask7.

Timestamp Management

* The script reads the last processed timestamp from a file named last\_processed\_timestamp.txt.
* The timestamp is parsed using the dateutil library.
* The script updates the last processed timestamp in the file after processing each batch of messages.

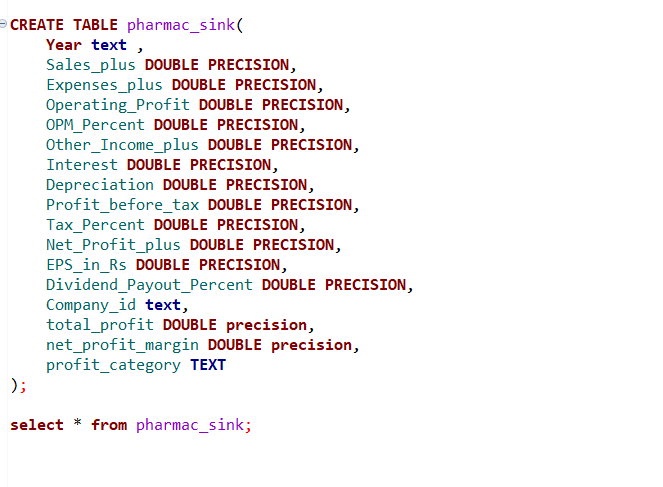
Data Processing

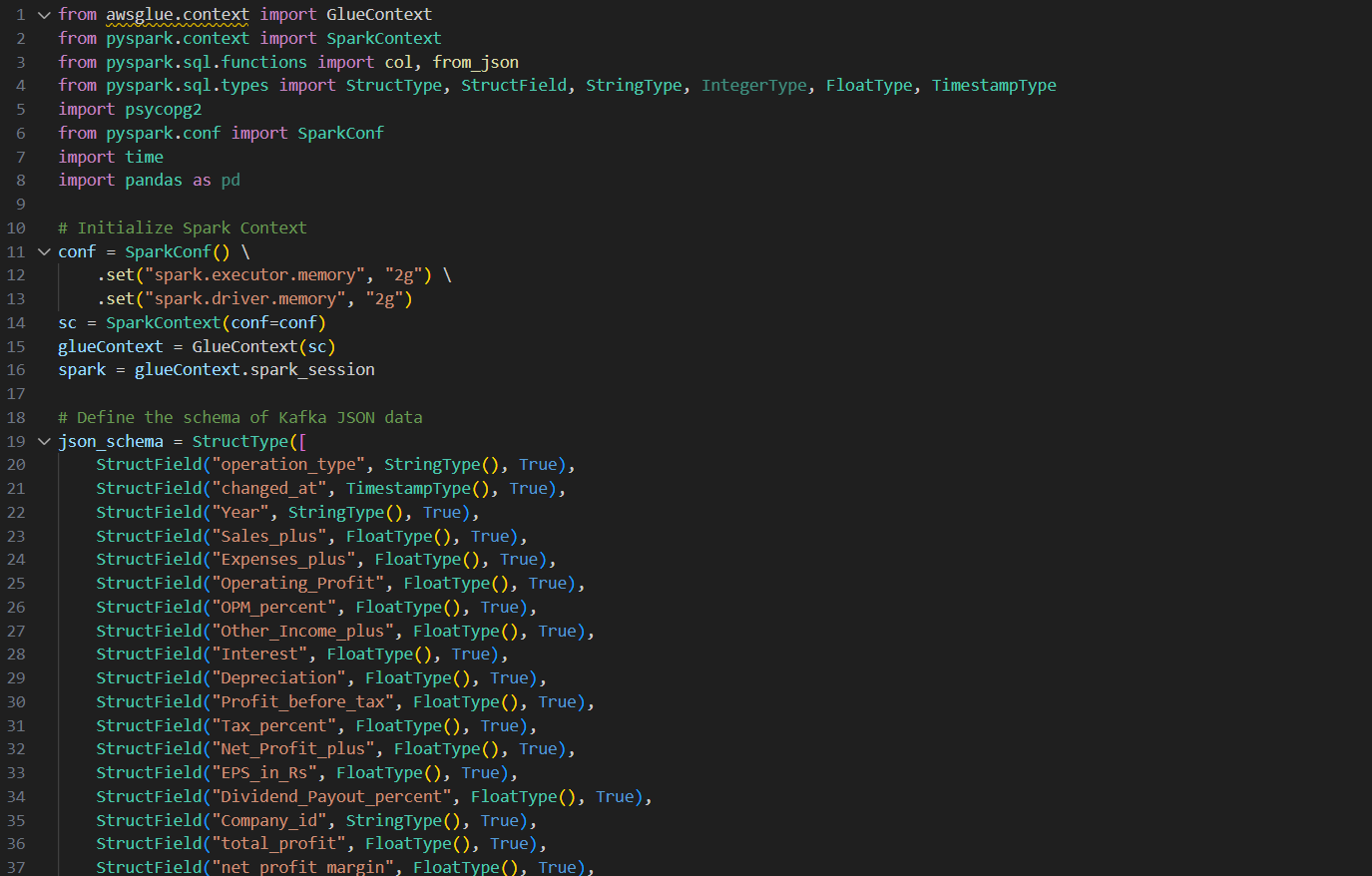
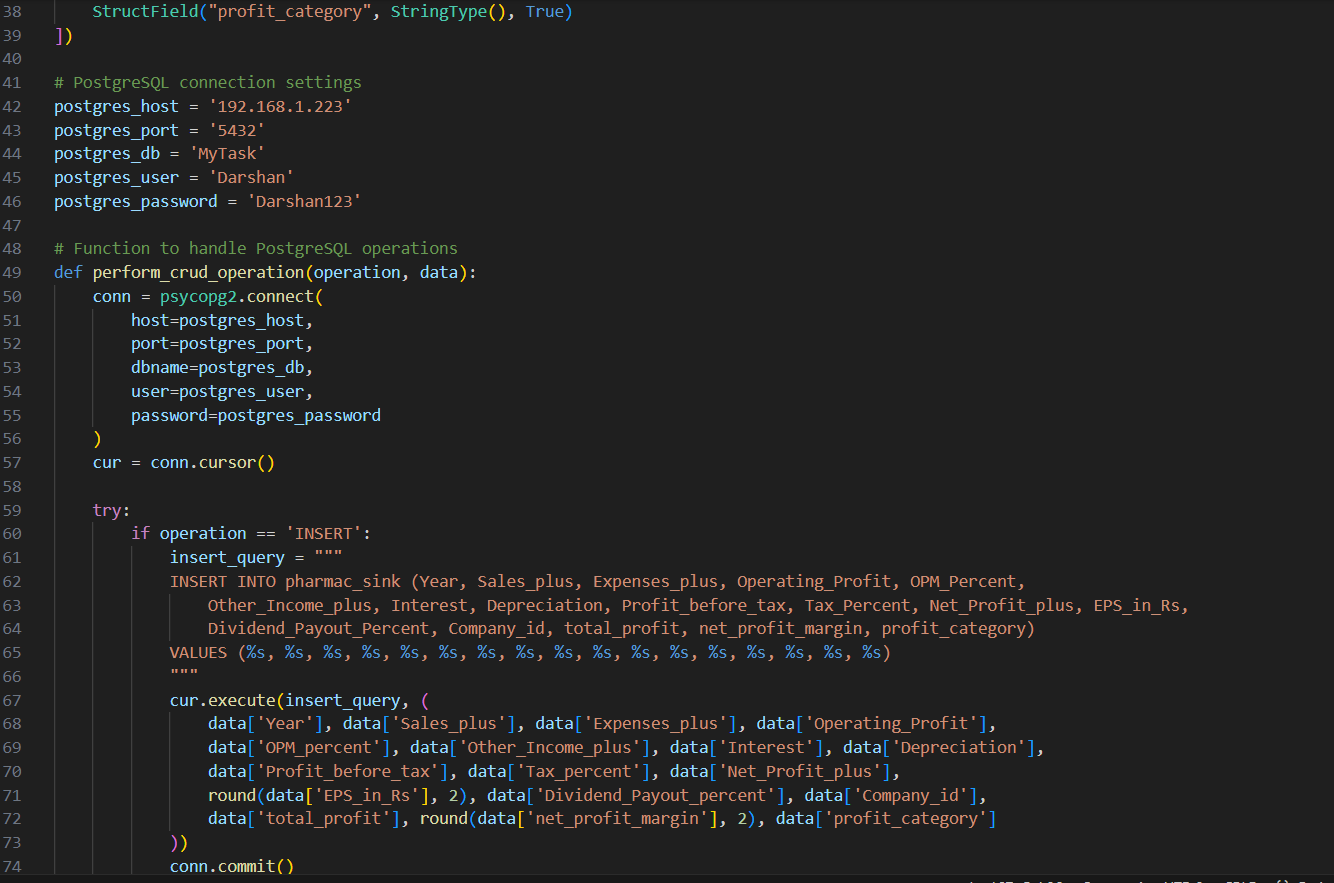
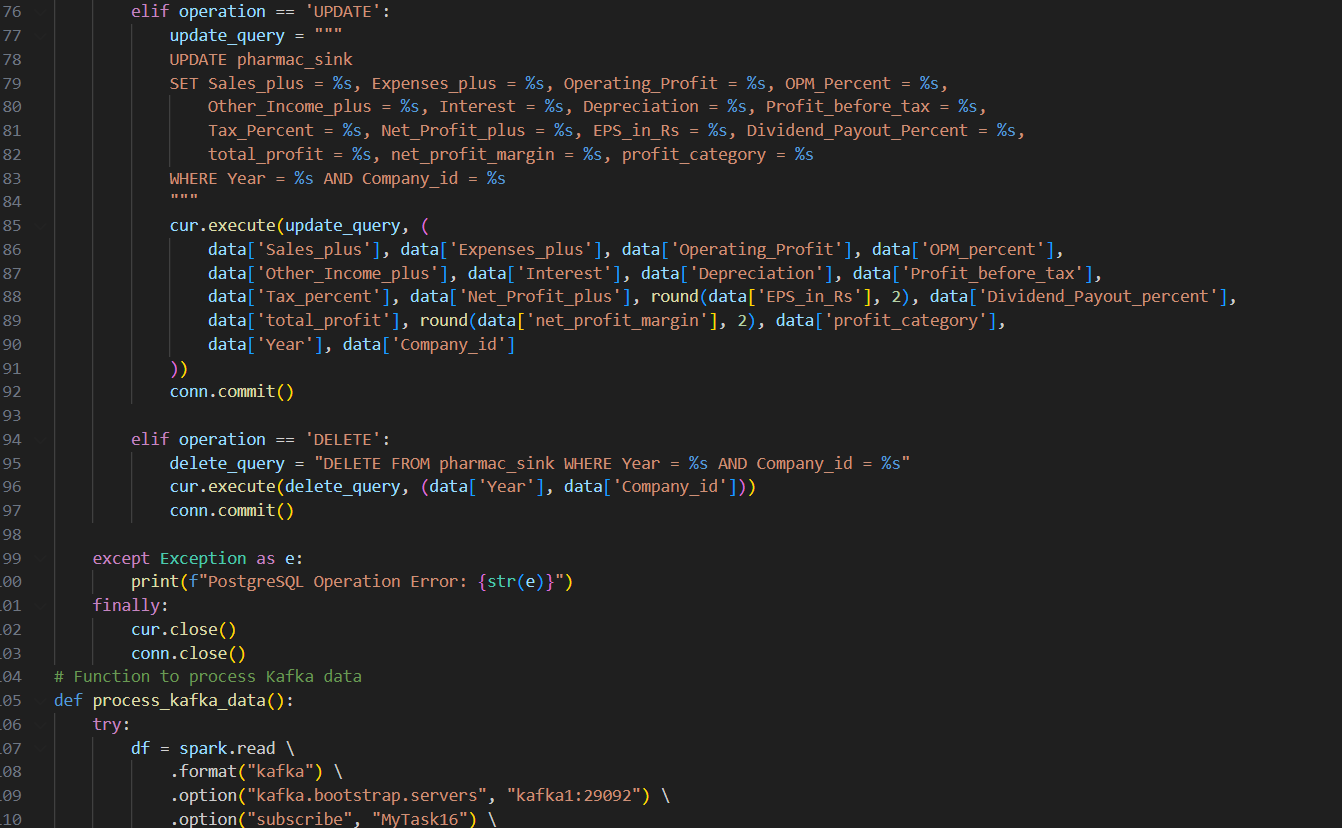
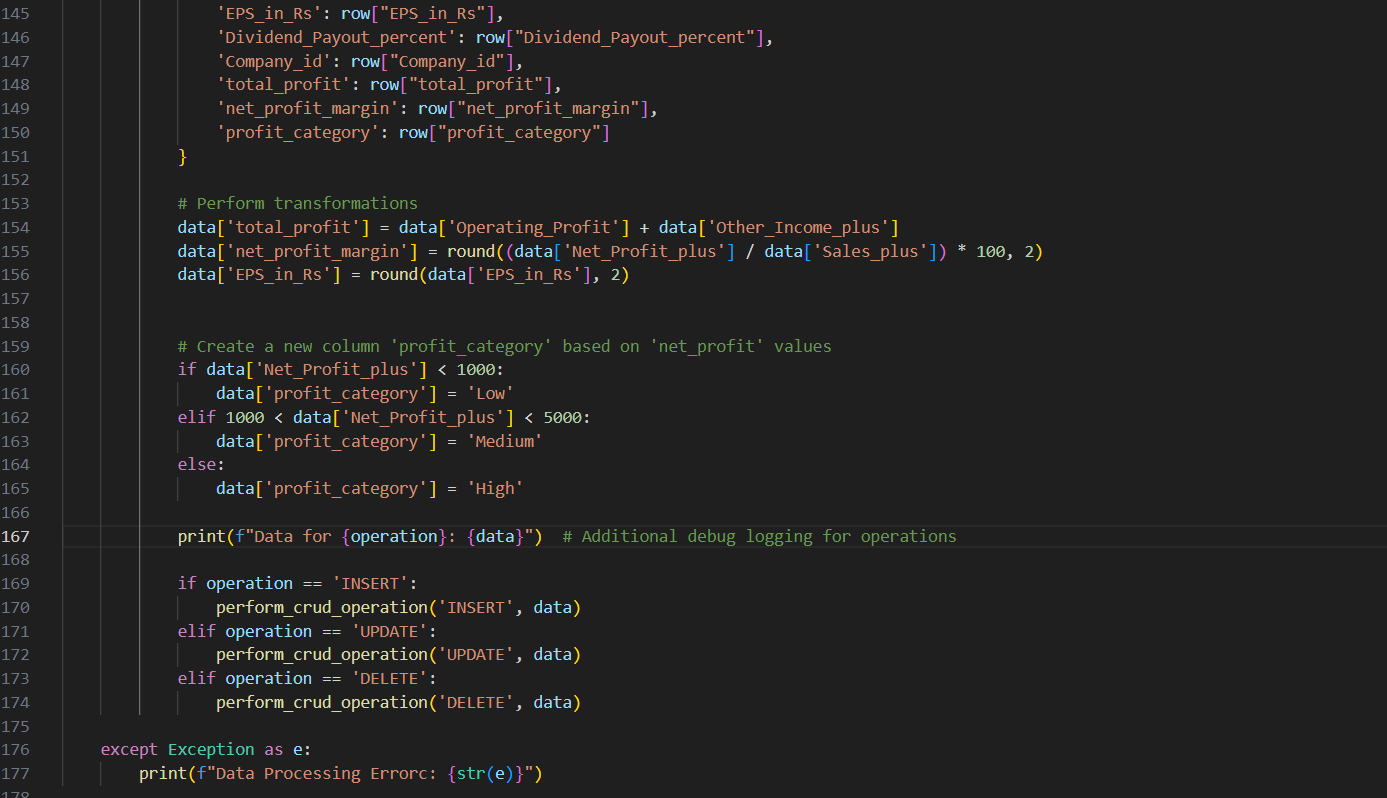
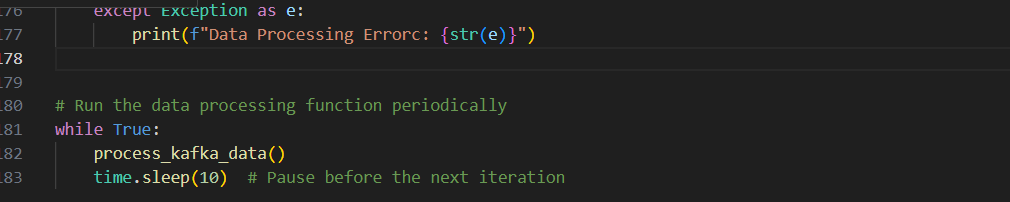
* The script fetches rows from the pharma\_profit\_loss\_data\_audit table where the timestamp is greater than the last processed timestamp.
* The fetched data is converted into JSON format using a custom JSON encoder (DateTimeEncoder) to handle datetime objects.
* The JSON data is produced to the Kafka topic

**Step 8:**

**Then updated a glue script:**

**For transformation of sink table:**

****

**** ****   

This scrpt includes:

1. Connect to a Kafka topic and read JSON data.
2. Process the data by performing CRUD (Create, Read, Update, Delete) operations on a PostgreSQL database.
3. Transform the data by calculating additional fields (e.g., total\_profit, net\_profit\_margin, profit\_category).
4. Run the data processing function periodically using a while loop.

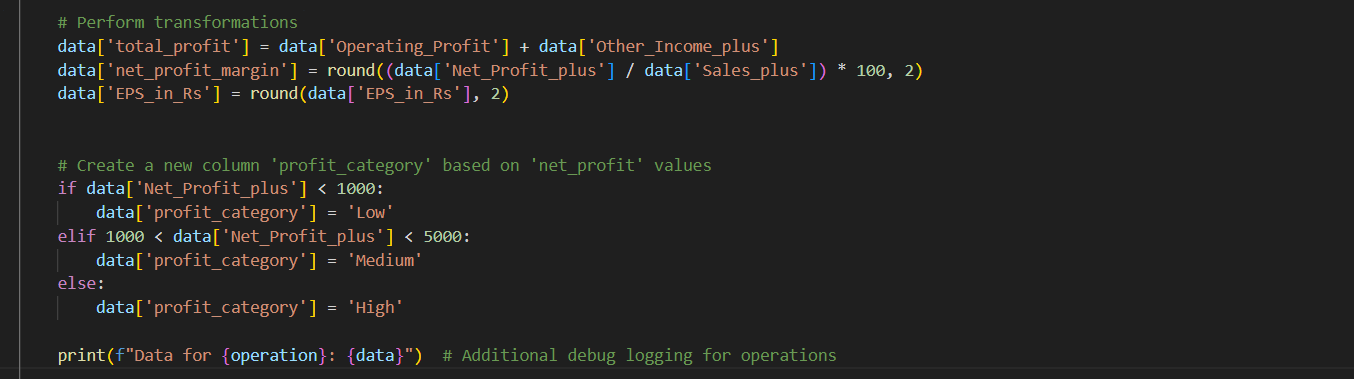
The script uses:

* Apache Spark for data processing
* PySpark for interacting with Spark
* psycopg2 for PostgreSQL connectivity
* Kafka for message streaming

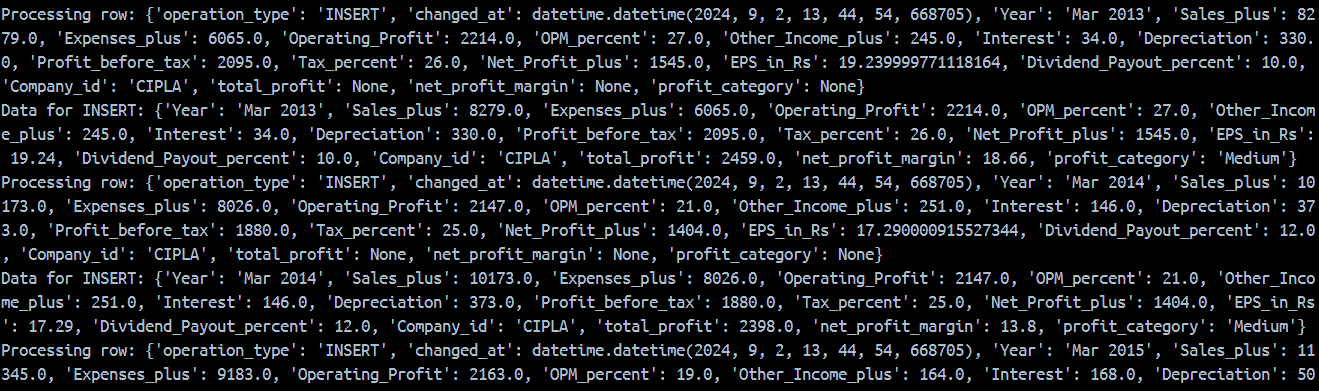
The script performs the following operations:

* Reads Kafka data
* Deserializes JSON data
* Processes data based on the operation type (INSERT, UPDATE, DELETE)
* Performs CRUD operations on the PostgreSQL database
* Transforms data by calculating additional fields
* Runs the data processing function periodically

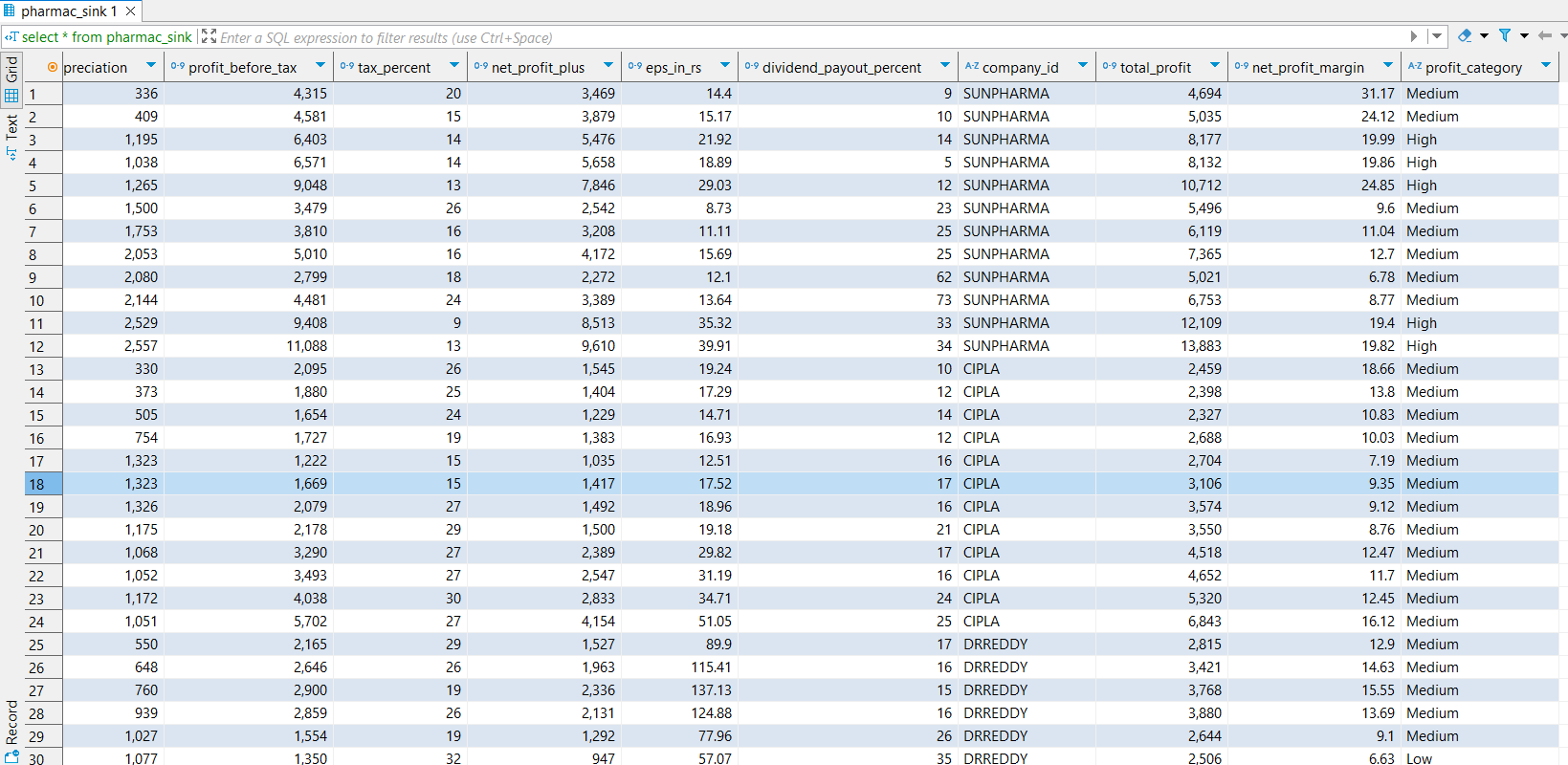
**As there in kafka\_topic Function I have added transforamtions:**



**Data in topic looks like:**



**Then that sink table is looks like this:**

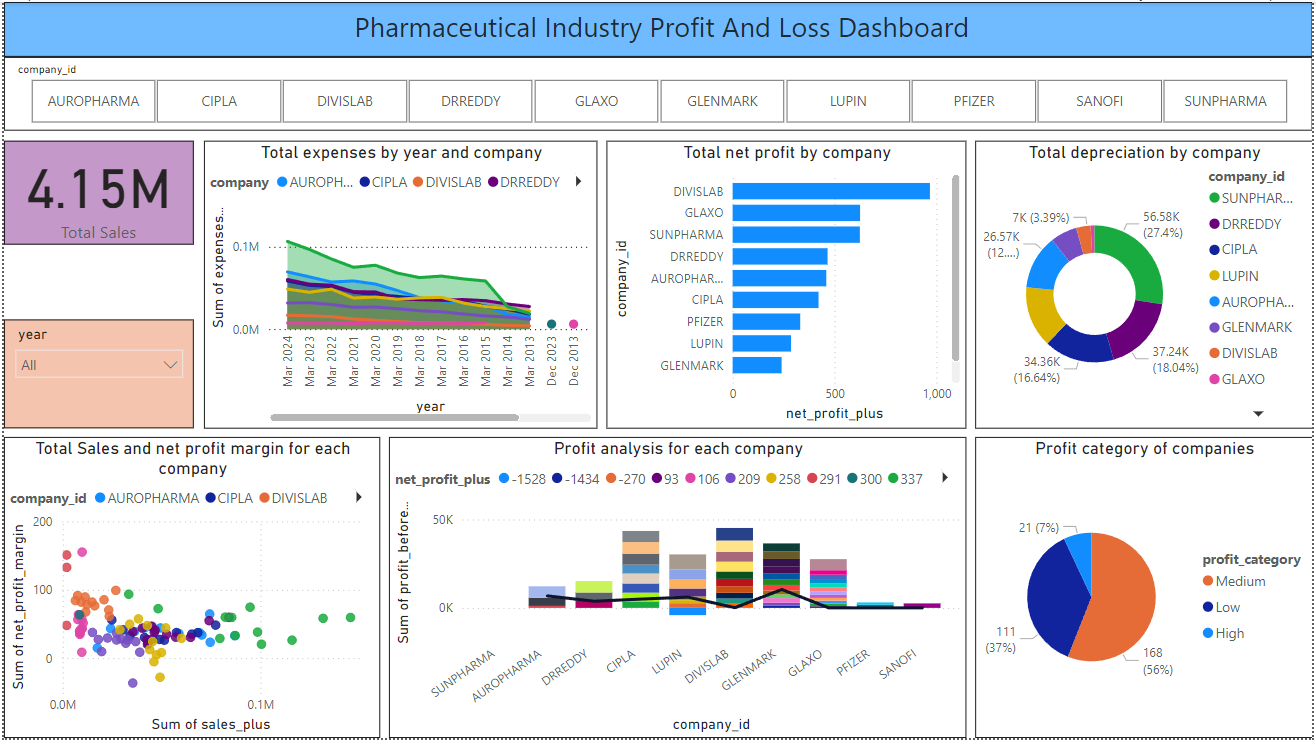


**With previous columns, new columns are also added up here.**

**Step 9:**



**Then sink table data is loaded into powerbi, then done some visualization and made final dashboard as well:**



**As here, the data is of pharma industries random 10 companies for 10 year period of time.**

1. **Made a Area chart for ‘Total expenses by year and company’.**

* **Where, we can see the total expenses made by a company over the years.**

1. **Bar chart for ‘Total net profit by company’.**

* **Total Net profit made by ‘DIVISLAB’ was highest amongst 10 companies.**

1. **Donut chart for depriciation by company.**

**( depreciation is basically as non- cash expenses i.e; network, infrastructure etc.)**

* **It shows the depreciation as like it is non-cash-expenses over the all companies which is for ‘SUNPHARMA’ is highest.**

1. **Pie chart for frofit category for companies.**

* **Where I have placed a margin as such like**

**Less than 1000cr will show the low (total\_net\_profit).**

**Between 1000cr to 5000cr will show medium**

**And above 5000cr it will be high.**

1. **Clustered column and line chart to display the trend of profit and profit\_before-tax basically shows profit analysis for companies.**

* **In which profit\_before\_tax and net\_profit for companies distribution has made.**

1. **Scattered plot to showcase distribution of net\_profit\_margin for particular sales for each company over a period of time.**

* **It shows net\_profit\_margin distribution for each company as sales made by them.**

1. **Added a scaler of company\_id and year as well to see the dynamic distribution of dashboard.**