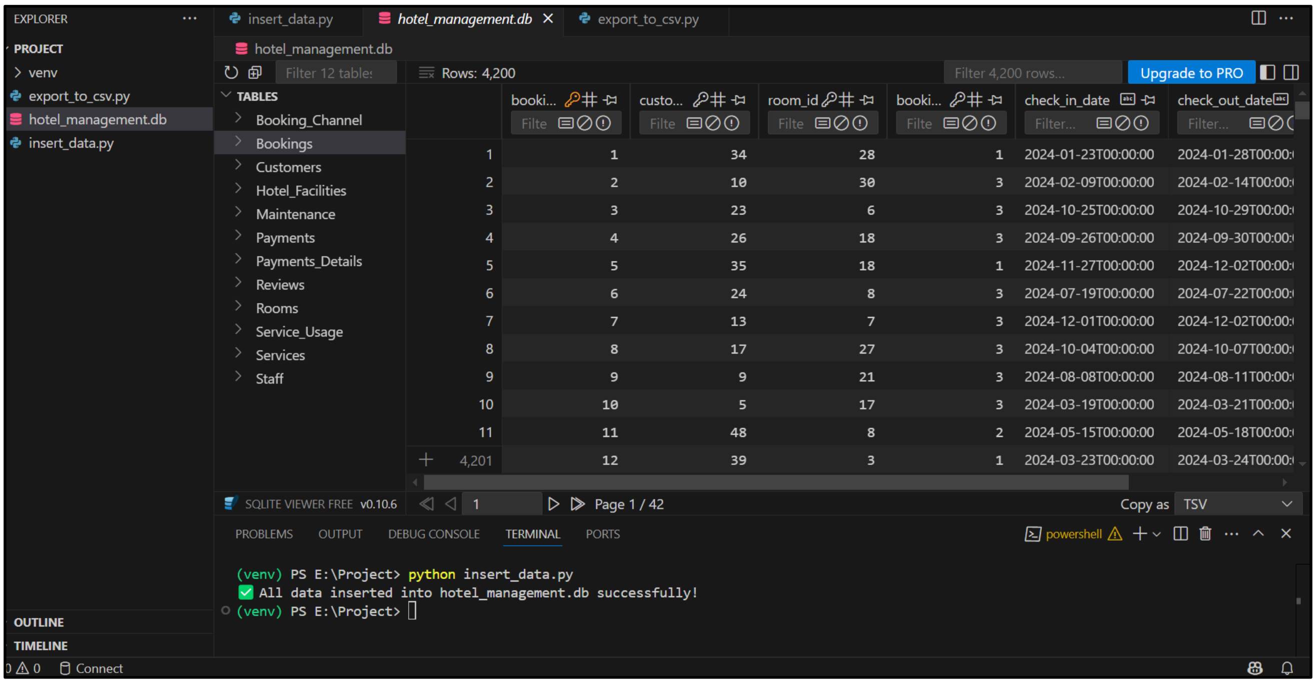
1. Database Setup and Data Insertion

Description:
Created and populated the hotel_management.db SQLite database using insert_data.py.

Command used:
python insert_data.py



hotel_management.db — Rows: 4,200

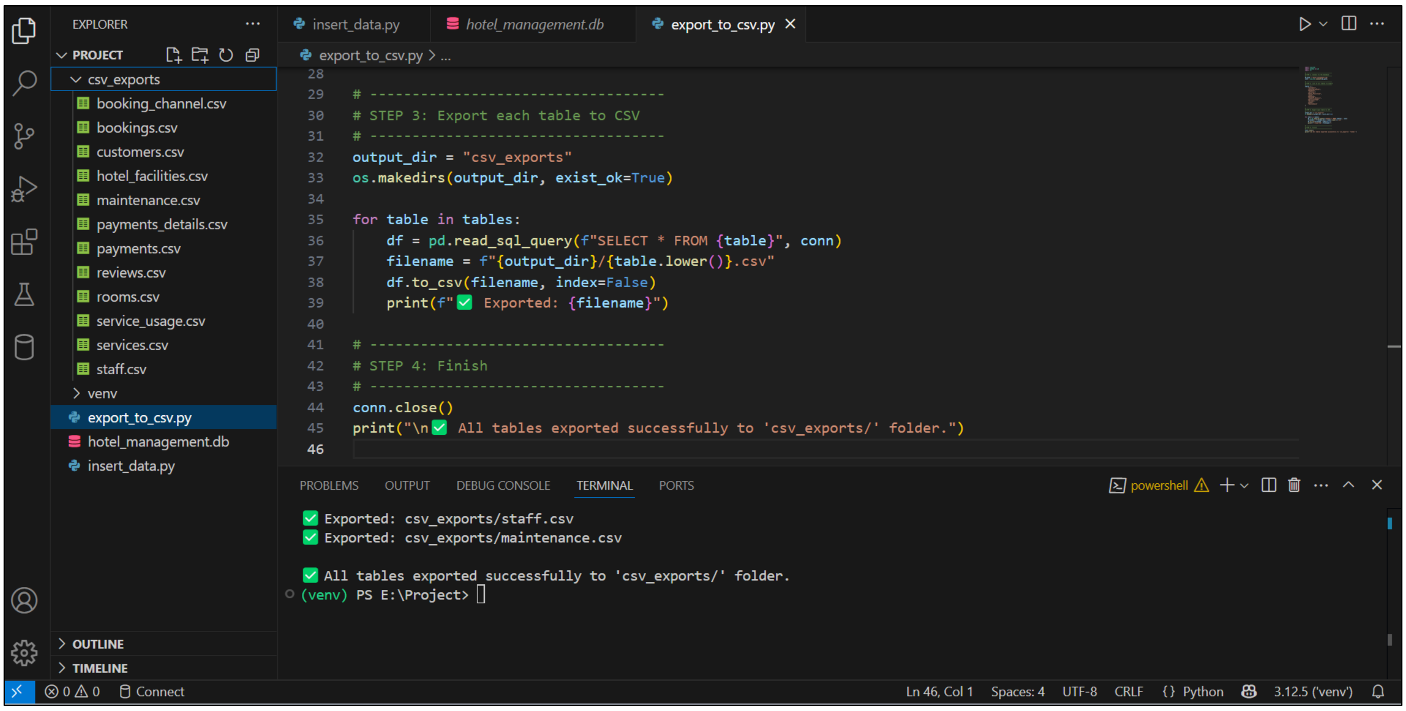
	booki...	custo...	room_id	booki...	check_in_date	check_out_date
1	1	34	28	1	2024-01-23T00:00:00	2024-01-28T00:00:
2	2	10	30	3	2024-02-09T00:00:00	2024-02-14T00:00:
3	3	23	6	3	2024-10-25T00:00:00	2024-10-29T00:00:
4	4	26	18	3	2024-09-26T00:00:00	2024-09-30T00:00:
5	5	35	18	1	2024-11-27T00:00:00	2024-12-02T00:00:
6	6	24	8	3	2024-07-19T00:00:00	2024-07-22T00:00:
7	7	13	7	3	2024-12-01T00:00:00	2024-12-02T00:00:
8	8	17	27	3	2024-10-04T00:00:00	2024-10-07T00:00:
9	9	9	21	3	2024-08-08T00:00:00	2024-08-11T00:00:
10	10	5	17	3	2024-03-19T00:00:00	2024-03-21T00:00:
11	11	48	8	2	2024-05-15T00:00:00	2024-05-18T00:00:
4,201	12	39	3	1	2024-03-23T00:00:00	2024-03-24T00:00:

```
(venv) PS E:\Project> python insert_data.py
✅ All data inserted into hotel_management.db successfully!
(venv) PS E:\Project>
```

2. Exporting Tables to CSV

Description:
All 12 RDBMS tables were exported to the csv_exports/ folder using export_to_csv.py.

Command used:
python export_to_csv.py



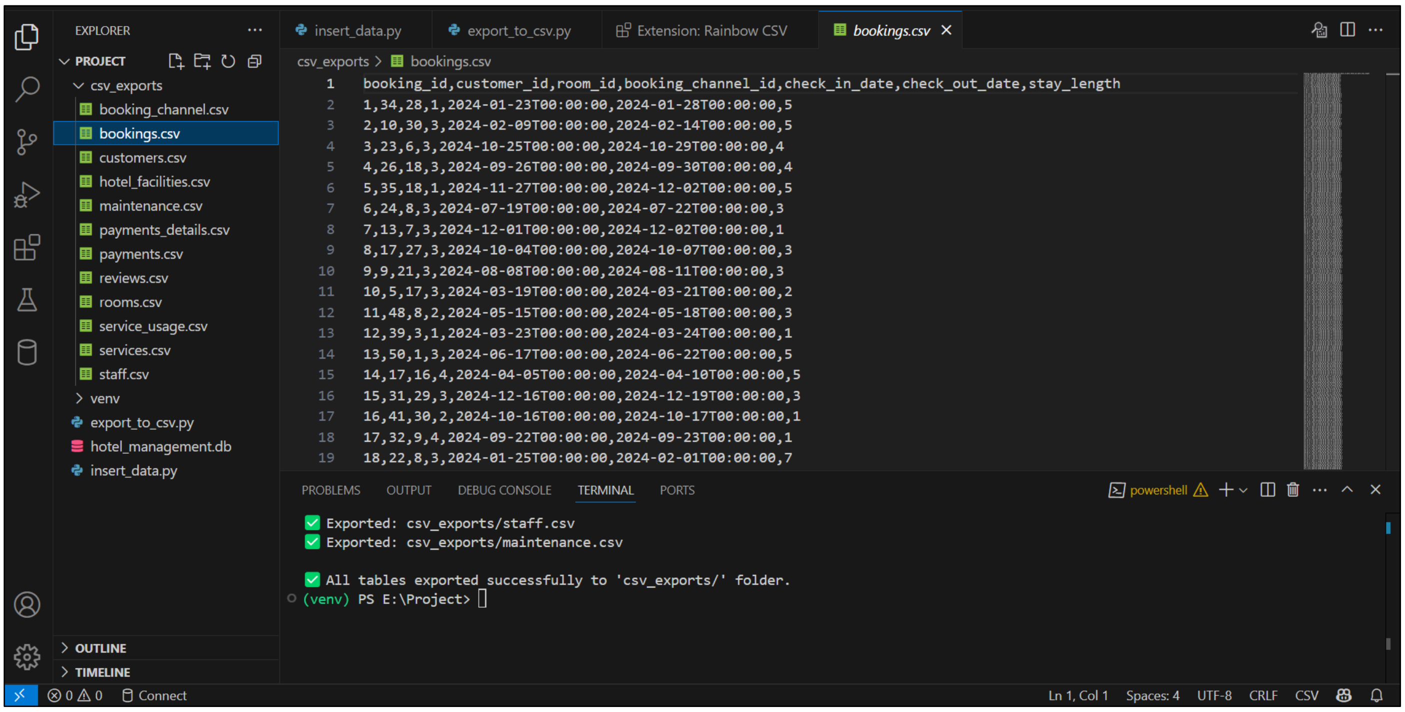
```python
# ---------------------------------
# STEP 3: Export each table to CSV
# ---------------------------------
output_dir = "csv_exports"
os.makedirs(output_dir, exist_ok=True)

for table in tables:
    df = pd.read_sql_query(f"SELECT * FROM {table}", conn)
    filename = f"{output_dir}/{table.lower()}.csv"
    df.to_csv(filename, index=False)
    print(f"✅ Exported: {filename}")

# ---------------------------------
# STEP 4: Finish
# ---------------------------------
conn.close()
print("\n✅ All tables exported successfully to 'csv_exports/' folder.")
```

```
✅ Exported: csv_exports/staff.csv
✅ Exported: csv_exports/maintenance.csv

✅ All tables exported successfully to 'csv_exports/' folder.
(venv) PS E:\Project>
```



```
booking_id,customer_id,room_id,booking_channel_id,check_in_date,check_out_date,stay_length
1,34,28,1,2024-01-23T00:00:00,2024-01-28T00:00:00,5
2,10,30,3,2024-02-09T00:00:00,2024-02-14T00:00:00,5
3,23,6,3,2024-10-25T00:00:00,2024-10-29T00:00:00,4
4,26,18,3,2024-09-26T00:00:00,2024-09-30T00:00:00,4
5,35,18,1,2024-11-27T00:00:00,2024-12-02T00:00:00,5
6,24,8,3,2024-07-19T00:00:00,2024-07-22T00:00:00,3
7,13,7,3,2024-12-01T00:00:00,2024-12-02T00:00:00,1
8,17,27,3,2024-10-04T00:00:00,2024-10-07T00:00:00,3
9,9,21,3,2024-08-08T00:00:00,2024-08-11T00:00:00,3
10,5,17,3,2024-03-19T00:00:00,2024-03-21T00:00:00,2
11,48,8,2,2024-05-15T00:00:00,2024-05-18T00:00:00,3
12,39,3,1,2024-03-23T00:00:00,2024-03-24T00:00:00,1
13,50,1,3,2024-06-17T00:00:00,2024-06-22T00:00:00,5
14,17,16,4,2024-04-05T00:00:00,2024-04-10T00:00:00,5
15,31,29,3,2024-12-16T00:00:00,2024-12-19T00:00:00,3
16,41,30,2,2024-10-16T00:00:00,2024-10-17T00:00:00,1
17,32,9,4,2024-09-22T00:00:00,2024-09-23T00:00:00,1
18,22,8,3,2024-01-25T00:00:00,2024-02-01T00:00:00,7
```

```
✅ Exported: csv_exports/staff.csv
✅ Exported: csv_exports/maintenance.csv

✅ All tables exported successfully to 'csv_exports/' folder.
(venv) PS E:\Project>
```

3. Docker and Airflow Setup

Description:

Docker Desktop was used to containerize the Airflow environment.

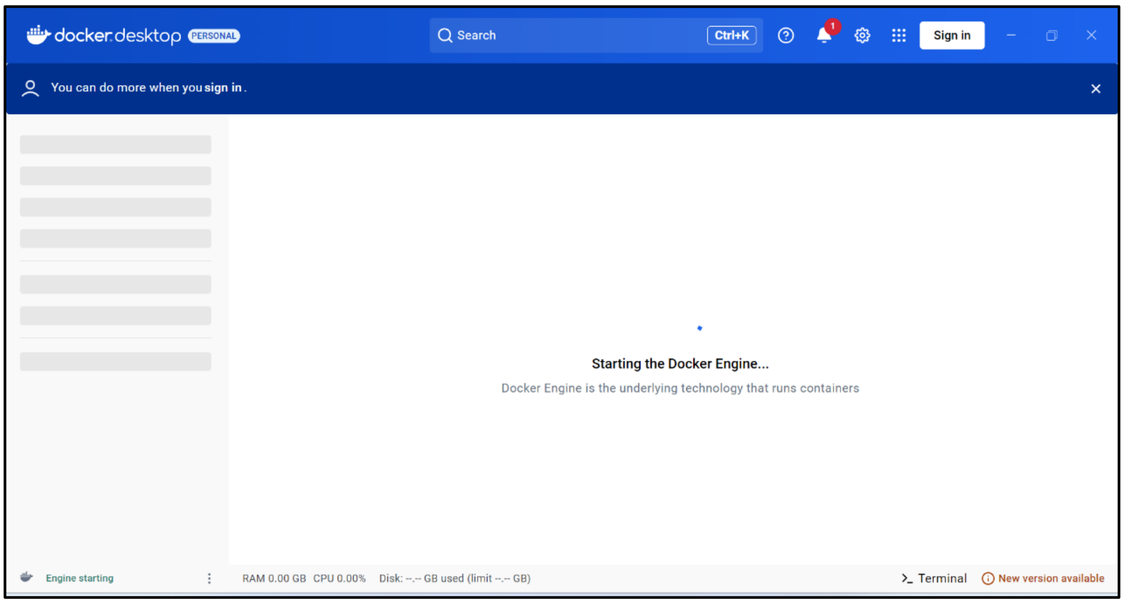
- Docker engine initialized
- Required services defined in docker-compose.yml

Command used:

docker-compose run airflow-init

Images:

- Screenshot (Docker starting)

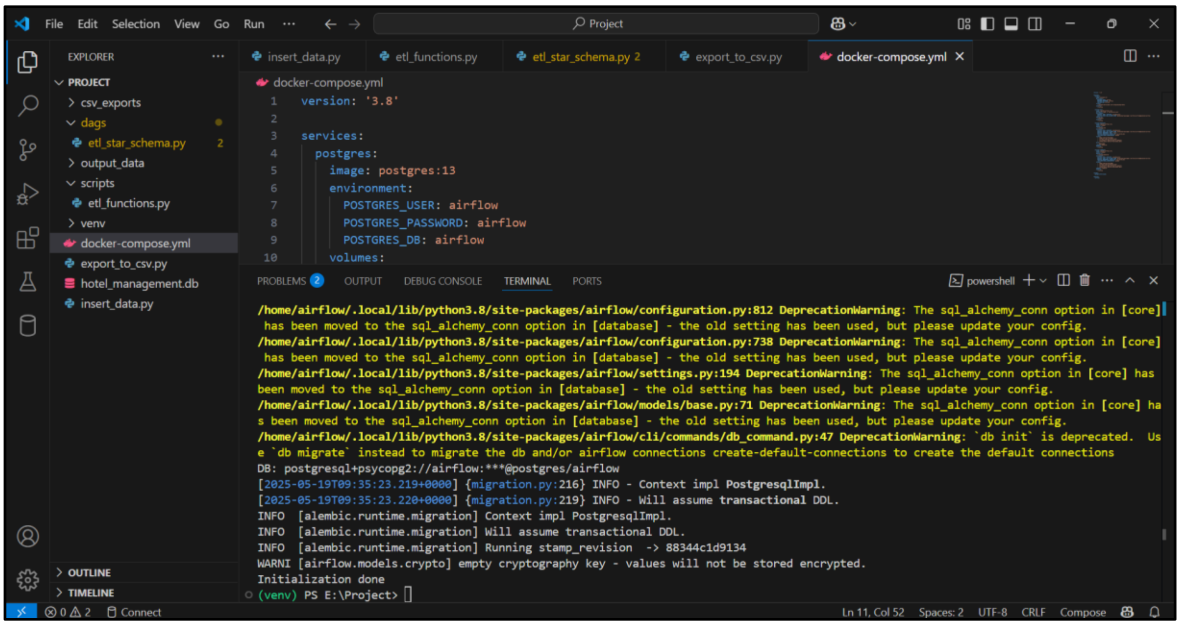


Starting the Docker Engine...

Docker Engine is the underlying technology that runs containers

docker desktop PERSONAL
Search
Ctrl+K
Sign in
You can do more when you sign in.
Engine starting
RAM 0.00 GB CPU 0.00% Disk: --.-- GB used (limit --.-- GB)
Terminal
New version available

- Screenshot (Terminal after airflow-init)



```yaml
version: '3.8'

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
```

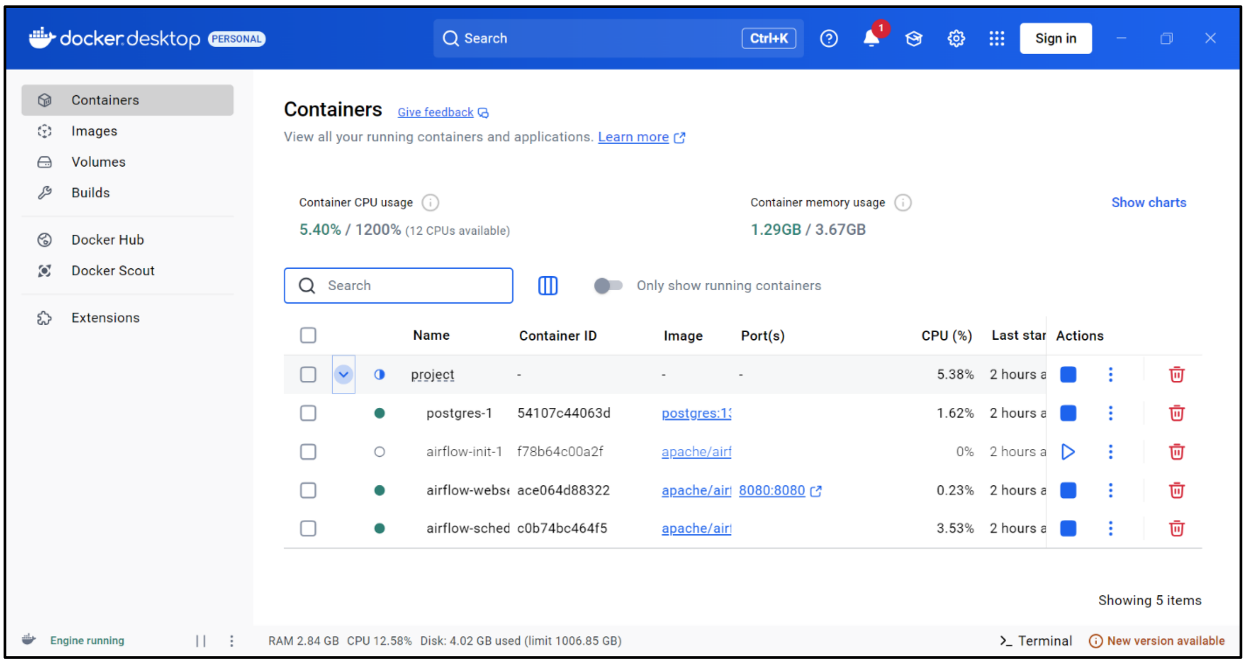
```
/home/airflow/.local/lib/python3.8/site-packages/airflow/configuration.py:812 DeprecationWarning: The sql_alchemy_conn option in [core]
 has been moved to the sql_alchemy_conn option in [database] - the old setting has been used, but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/configuration.py:738 DeprecationWarning: The sql_alchemy_conn option in [core]
 has been moved to the sql_alchemy_conn option in [database] - the old setting has been used, but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/settings.py:194 DeprecationWarning: The sql_alchemy_conn option in [core] has
been moved to the sql_alchemy_conn option in [database] - the old setting has been used, but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/models/base.py:71 DeprecationWarning: The sql_alchemy_conn option in [core] ha
s been moved to the sql_alchemy_conn option in [database] - the old setting has been used, but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/cli/commands/db_command.py:47 DeprecationWarning: `db init` is deprecated.  Us
e `db migrate` instead to migrate the db and/or airflow connections create-default-connections to create the default connections
DB: postgresql+psycopg2://airflow:***@postgres/airflow
[2025-05-19T09:35:23.219+0000] {migration.py:216} INFO - Context impl PostgresqlImpl.
[2025-05-19T09:35:23.220+0000] {migration.py:219} INFO - Will assume transactional DDL.
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running stamp_revision  -> 88344c1d9134
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
Initialization done
(venv) PS E:\Project>
```

4. Launching Airflow Services

Description: Docker containers running for Airflow and Postgres

Command used:

docker-compose up -d



Containers

View all your running containers and applications.

Container CPU usage
5.40% / 1200% (12 CPUs available)

Container memory usage
1.29GB / 3.67GB

Only show running containers

Name	Container ID	Image	Port(s)	CPU (%)	Last star
project	-	-	-	5.38%	2 hours a
postgres-1	54107c44063d	postgres:1		1.62%	2 hours a
airflow-init-1	f78b64c00a2f	apache/airf		0%	2 hours a
airflow-webs	ace064d88322	apache/air	8080:8080	0.23%	2 hours a
airflow-sched	c0b74bc464f5	apache/air		3.53%	2 hours a

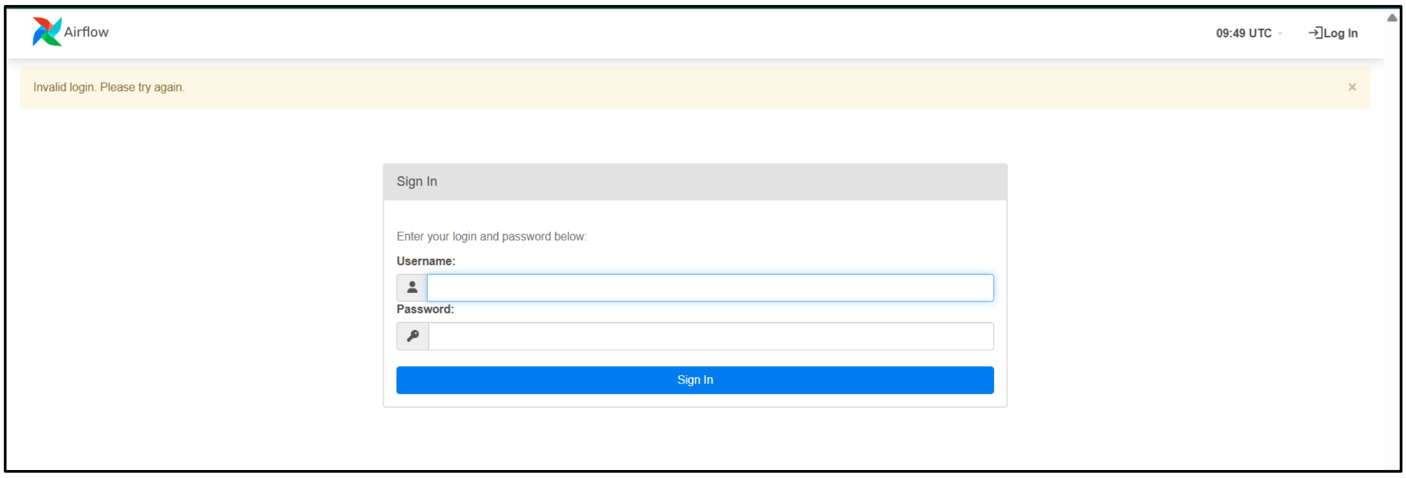
Showing 5 items

Engine running

RAM 2.84 GB CPU 12.58% Disk: 4.02 GB used (limit 1006.85 GB)

5. Accessing Airflow UI

Description:
Accessed Airflow UI at http://localhost:8080. Logged in to monitor and run the DAG.



Invalid login. Please try again.

Sign In

Enter your login and password below:

Username:

Password:

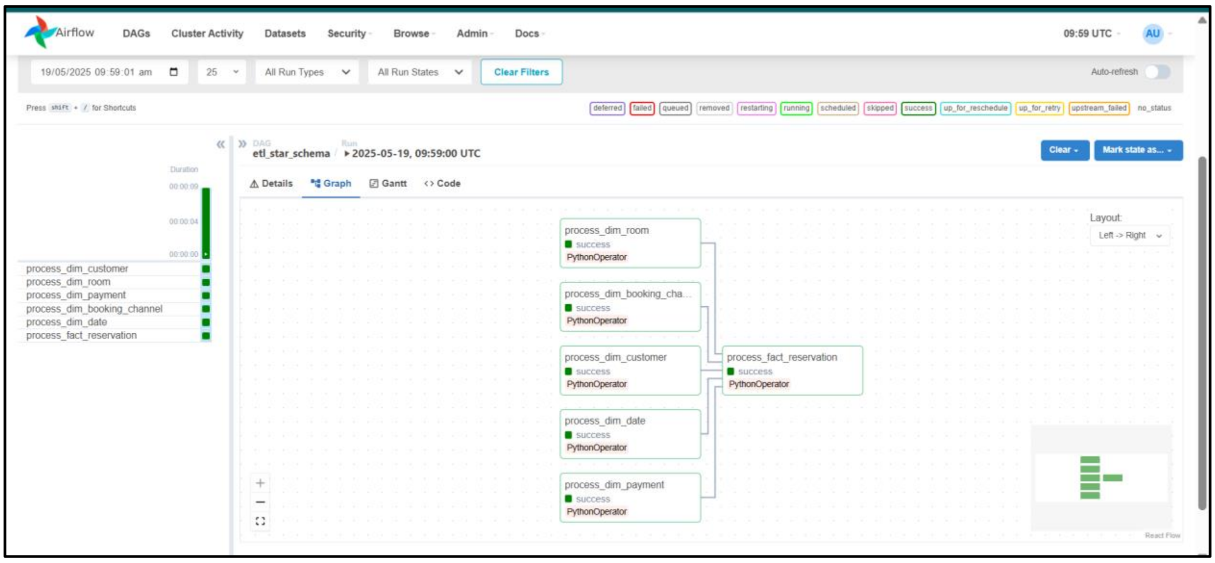
6. DAG Implementation and Execution

Description:
The DAG file etl_star_schema.py defines the ETL workflow. It imports transformation functions from etl_functions.py (stored in the scripts/ folder). Each function processes one table from the star schema (e.g., dim_customer, dim_room, etc.).

The DAG was activated and manually triggered through the Airflow UI. All tasks executed successfully, as shown by the green boxes in the DAG graph view. This confirms that all dimension and fact tables were processed without any errors.

Files referenced:

- dags/etl_star_schema.py
- scripts/etl_functions.py



process_dim_customer
process_dim_room
process_dim_payment
process_dim_booking_channel
process_dim_date
process_fact_reservation

process_dim_room — success — PythonOperator
process_dim_booking_cha... — success — PythonOperator
process_dim_customer — success — PythonOperator
process_dim_date — success — PythonOperator
process_dim_payment — success — PythonOperator
process_fact_reservation — success — PythonOperator

7. Final Output: Transformed Data for the Star Schema

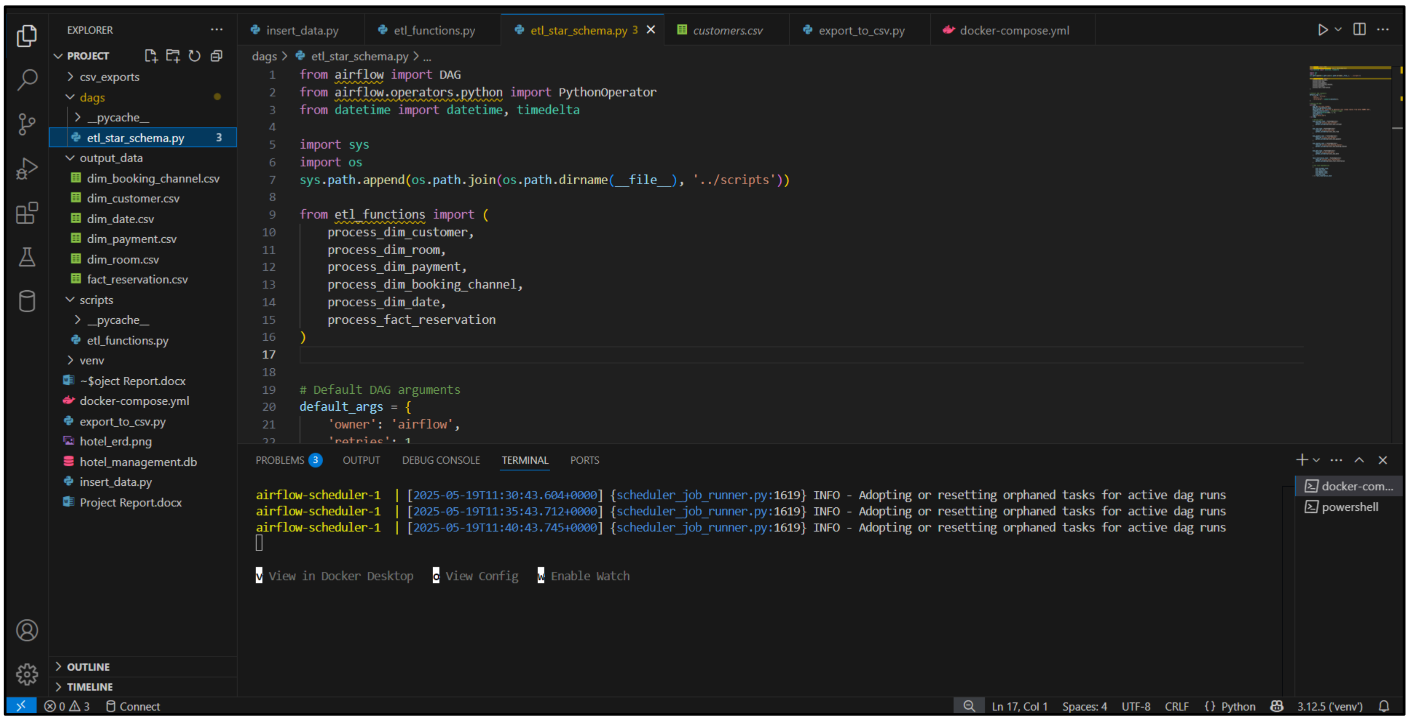
Description:

After the successful execution of the DAG, the transformed data was saved as CSV files in the output_data/ folder. Each file represents a table from the star schema, including all dimension and fact tables required for analytical processing.

These CSVs can now be used for loading into a data warehouse or for dashboard/reporting purposes in tools like Power BI.

Output Files Include:

- dim_customer.csv
- dim_room.csv
- dim_payment.csv
- dim_date.csv
- dim_booking_channel.csv
- fact_reservation.csv



```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta

import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), '../scripts'))

from etl_functions import (
    process_dim_customer,
    process_dim_room,
    process_dim_payment,
    process_dim_booking_channel,
    process_dim_date,
    process_fact_reservation
)


# Default DAG arguments
default_args = {
    'owner': 'airflow',
    'retries': 1
```

```
airflow-scheduler-1  | [2025-05-19T11:30:43.604+0000] {scheduler_job_runner.py:1619} INFO - Adopting or resetting orphaned tasks for active dag runs
airflow-scheduler-1  | [2025-05-19T11:35:43.712+0000] {scheduler_job_runner.py:1619} INFO - Adopting or resetting orphaned tasks for active dag runs
airflow-scheduler-1  | [2025-05-19T11:40:43.745+0000] {scheduler_job_runner.py:1619} INFO - Adopting or resetting orphaned tasks for active dag runs


v View in Docker Desktop   o View Config   w Enable Watch
```

8. Conclusion

This project successfully implemented a complete ETL pipeline for a Hotel Management System using Apache Airflow and Docker. The process began with the creation and population of a relational database, followed by structured extraction and export of data to CSV format. These raw files were then transformed into a star schema suitable for analytical processing.

By leveraging Airflow's DAG-based orchestration, the entire workflow—from reading source data to producing clean, analysis-ready tables—was automated and modular. Each step in the pipeline was carefully verified, and the final outputs are fully ready for integration into business intelligence tools.

This setup provides a scalable and repeatable ETL framework that can be extended or modified for future enhancements, such as incremental loads, error logging, or cloud deployment.