ETL DATA PIPELINE WITH APACHE AIRFLOW FOR ONLINE E-COMMERCE

IMPLEMENTING AN ETL PIPELINE USING APACHE AIRFLOW TO PROCESS AND LOAD AN E-COMMERCE DATASET FROM KAGGLE INTO MYSQL AND POSTGRESQL

Faruq Abdurrahman F

PROJECT DESCRIPTION

PROJECT GOAL

Build an automated ETL pipeline using **Apache Airflow** to download and process data from Kaggle

The dataset used is the **Online E-commerce** dataset from Kaggle, which includes transaction and product information

KEY FEATURES

Automation of the ETL workflow

Downloading data from Kaggle using the Kaggle API

Data transformation for consistency and cleanliness

Loading data into **MySQL** (staging) and **PostgreSQL** (final)

WORKFLOW

Download Data	Use Kaggle API to fetch the dataset from Kaggle
Extract Data	Read CSV files with Pandas
Load Data into MySQL	Filter new data and store it in MySQL
Transform Data	Remove duplicates, change data types, and handle missing values
Load Data into PostgreSQL	Store the final processed data in PostgreSQL

TECHNOLOGIES USED

Apache Airflow	Tool for managing, scheduling, and monitoring the ETL workflow
Kaggle API	Download datasets from Kaggle
MySQL	Store data in the staging area
PostgreSQL	Store transformed data
Pandas	Data manipulation and transformation
Python	Programming for writing ETL scripts
VSCode	Code editor for writing and managing scripts
DBeaver	Tool for managing and visualizing databases

STEP 1 Download Data

Using **Kaggle API** to download the dataset automatically

```python
def download_kaggle_data(**kwargs):
    """
    Download dataset from Kaggle using the Kaggle API and return the file path.
    """
    os.environ['KAGGLE_CONFIG_DIR'] = "/opt/airflow/.kaggle"

    dataset_name = "ayushparwal2026/online-ecommerce"
    download_path = "/opt/airflow/data/online-ecommerce"
    os.makedirs(download_path, exist_ok=True)
    os.system(f"kaggle datasets download -d {dataset_name} -p {download_path} --unzip")

    file_path = os.path.join(download_path, "Online-eCommerce.csv")
    kwargs['ti'].xcom_push(key='file_path', value=file_path)
    print("Data downloaded to:", file_path)

```

Process:

- Configure the environment to access the Kaggle API.
- Download the online-ecommerce dataset from Kaggle and extract it to the target directory.
- Save the dataset file path in XCom for the next process.
- Print the file location.

STEP 2 Extract Data

Read the downloaded CSV file using Pandas for further processing

```python
def extract_data(**kwargs):
    """
    Load CSV data into a pandas DataFrame.
    """
    file_path = kwargs['ti'].xcom_pull(task_ids='download_kaggle_data', key='file_path')
    print(f"Extracting data from file: {file_path}")
    data = pd.read_csv(file_path)
    print("Data extracted successfully")
    kwargs['ti'].xcom_push(key='extracted_data', value=data.to_dict('records'))
```

Process:

- Retrieve the CSV file path from XCom, provided by the dataset download step.
- Read the CSV data using pandas and load it into a DataFrame.
- Store the extracted data in XCom as a list of records for the next step.
- Print a confirmation that the data was successfully extracted.

STEP 3 Load Data to MySQL

Load raw data into the MySQL staging table for temporary storage

```python
def load_data_to_mysql(**kwargs):
    """
    Load new data into the MySQL staging area after checking for duplicates.
    """
    extracted_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='extract_data', key='extracted_data'))

    if 'Order_Number' not in extracted_data.columns:
        raise ValueError("Column 'Order_Number' not found in the provided data.")

    # Use MySqlHook for MySQL connection
    mysql_hook = MySqlHook(mysql_conn_id='mysql_dibimbing')
    engine = mysql_hook.get_sqlalchemy_engine()

    with engine.connect() as connection:
        query = "SELECT Order_Number FROM kaggle_data_staging"
        existing_data = pd.read_sql(query, connection)
        new_data = extracted_data[~extracted_data['Order_Number'].isin(existing_data['Order_Number'])]

    if not new_data.empty:
        new_data.to_sql('kaggle_data_staging', con=engine, if_exists='append', index=False)
        print(f"{len(new_data)} new rows added to MySQL staging area.")
    else:
        print("No new data to add to MySQL.")

    kwargs['ti'].xcom_push(key='staged_data', value=new_data.to_dict('records'))
```

Process:

- Retrieve extracted data from XCom and load it into a DataFrame.
- Check for duplicate data by comparing the **Order_Number** column with existing data in MySQL.
- Insert new data into the MySQL staging table if no duplicates are found.
- Push the loaded data to XCom for the next process.

STEP 4 Transform Data

Transform data to ensure consistency and correct formatting

```python
def transform_data(**kwargs):
    """
    Perform all necessary data transformations including formatting, renaming columns,
    handling missing values, and ensuring data consistency.
    """
    staged_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='load_data_to_mysql', key='staged_data'))

    # Check if 'Order_Date' exists in the data
    if 'Order_Date' not in staged_data.columns:
        print("Warning: 'Order_Date' column is missing!")
        return  # Skip the transformation if the column is missing

    # Remove duplicates
    staged_data.drop_duplicates(inplace=True)

    # Format Order_Date to YYYY-MM-DD
    staged_data['Order_Date'] = pd.to_datetime(staged_data['Order_Date'], format='%d/%m/%Y', errors='coerce').dt.strftime('%Y-%m-%d')

    # Rename columns for PostgreSQL compatibility
    staged_data.rename(columns={"Assigned Supervisor": "Assigned_Supervisor"}, inplace=True)

    # Drop rows with missing critical fields
    staged_data.dropna(subset=['Cost', 'Sales'], inplace=True)
```

```python
# Convert Cost and Sales to numeric values
staged_data['Cost'] = pd.to_numeric(staged_data['Cost'], errors='coerce')
staged_data['Sales'] = pd.to_numeric(staged_data['Sales'], errors='coerce')

# Ensure Order_Number is treated as string
staged_data['Order_Number'] = staged_data['Order_Number'].astype(int).astype(str)

# Handle missing values in non-critical columns
staged_data.fillna({
    'State_Code': 'Unknown',
    'Customer_Name': 'Unknown',
    'Status': 'Unknown',
    'Product': 'Unknown',
    'Category': 'Unknown',
    'Brand': 'Unknown',
    'Quantity': 0.0,
    'Total_Cost': 0.0,
    'Total_Sales': 0.0,
    'Assigned_Supervisor': 'Unknown'
}, inplace=True)

print("Data transformed successfully")
kwargs['ti'].xcom_push(key='transformed_data', value=staged_data.to_dict('records'))
```

Process:

- Check Order_Date and remove duplicates.
- Reformat data and adjust critical column types.
- Fill missing values in non-critical columns with defaults.
- Save transformed data to XCom.

STEP 5 Load Data into PostgreSQL

Load processed data into PostgreSQL for final storage

```python
def load_transformed_data_to_postgresql(**kwargs):
    """
    Load transformed data from pandas DataFrame into PostgreSQL.
    """
    # Mendapatkan data yang ditransformasi menggunakan XCom
    transformed_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='transform_data', key='transformed_data'))

    # Menggunakan PostgresHook untuk koneksi ke PostgreSQL
    postgres_hook = PostgresHook(postgres_conn_id="postgres_dibimbing")
    postgres_engine = postgres_hook.get_sqlalchemy_engine()

    upsert_sql = """
    INSERT INTO portofolio.online_order (
        Order_Number, State_Code, Customer_Name, Order_Date, Status, Product,
        Category, Brand, Cost, Sales, Quantity, Total_Cost, Total_Sales, Assigned_Supervisor
    )
    VALUES (
        :Order_Number, :State_Code, :Customer_Name, :Order_Date, :Status, :Product,
        :Category, :Brand, :Cost, :Sales, :Quantity, :Total_Cost, :Total_Sales, :Assigned_Supervisor
    )
    ON CONFLICT (Order_Number) DO NOTHING;
    """

    # Membuka koneksi dan melakukan eksekusi
    with postgres_engine.connect() as connection:
        # Iterasi melalui setiap baris data dan eksekusi upsert query
        for _, row in transformed_data.iterrows():
            connection.execute(text(upsert_sql), row.to_dict())

    print("Transformed data successfully loaded into PostgreSQL.")
```

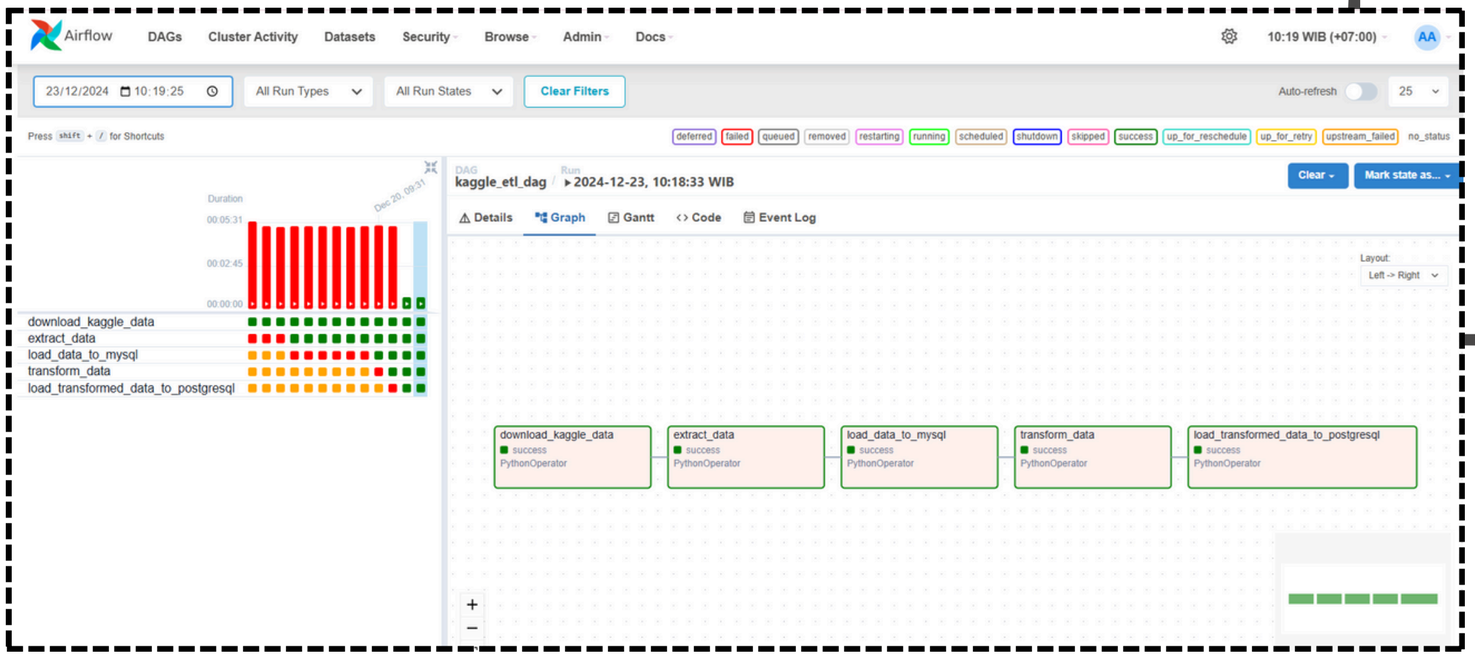
Process:

- Retrieve transformed data from XCom and load it into a DataFrame.
- Connect to PostgreSQL using PostgresHook.
- Perform an upsert operation to prevent duplicate entries.
- Ensure all data is successfully loaded into the final database.

WORKFLOW DIAGRAM WITH APACHE AIRFLOW

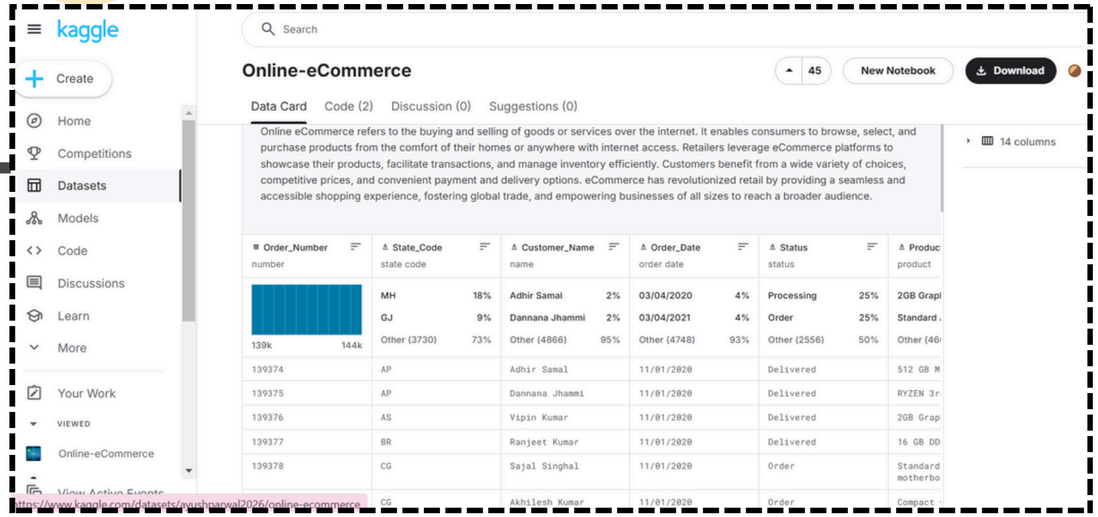
Demonstrates how these tasks are organized and executed using Apache Airflow for automation

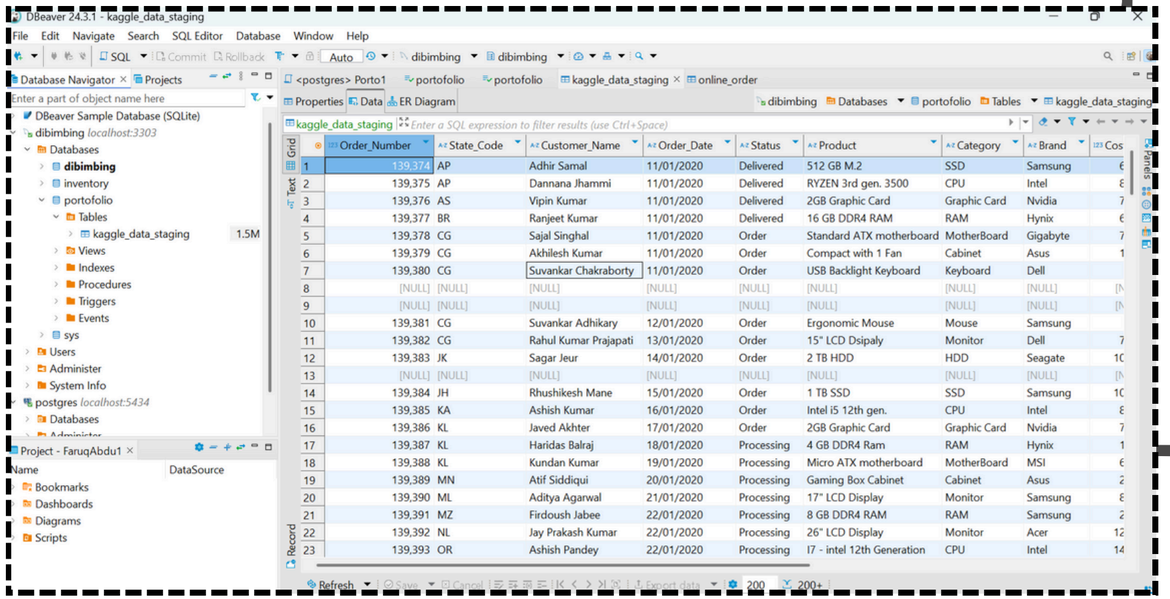
```python
# Define the DAG
with DAG(
    'kaggle_etl_dag',
    default_args={
        'owner': 'burqi',
        'retries': 1,
        'retry_delay': timedelta(minutes=5),
    },
    description='ETL process for Kaggle dataset',
    schedule_interval=None,
    start_date=datetime(2024, 12, 17),
    catchup=False,
) as dag:

    # Task 1: Download Data from Kaggle
    download_task = PythonOperator(
        task_id='download_kaggle_data',
        python_callable=download_kaggle_data,
    )

    # Task 2: Extract Data from CSV
    extract_task = PythonOperator(
        task_id='extract_data',
        python_callable=extract_data,
        provide_context=True,
    )

    # Task 3: Load Data to MySQL
    load_mysql_task = PythonOperator(
        task_id='load_data_to_mysql',
        python_callable=load_data_to_mysql,
        provide_context=True,
    )

    # Task 4: Transform Data
    transform_task = PythonOperator(
        task_id='transform_data',
        python_callable=transform_data,
        provide_context=True,
    )

    # Task 5: Load Transformed Data to PostgreSQL
    load_postgresql_task = PythonOperator(
        task_id='load_transformed_data_to_postgresql',
        python_callable=load_transformed_data_to_postgresql,
        provide_context=True,
    )

    # Set task dependencies
    download_task >> extract_task >> load_mysql_task >> transform_task >> load_postgresql_task
```

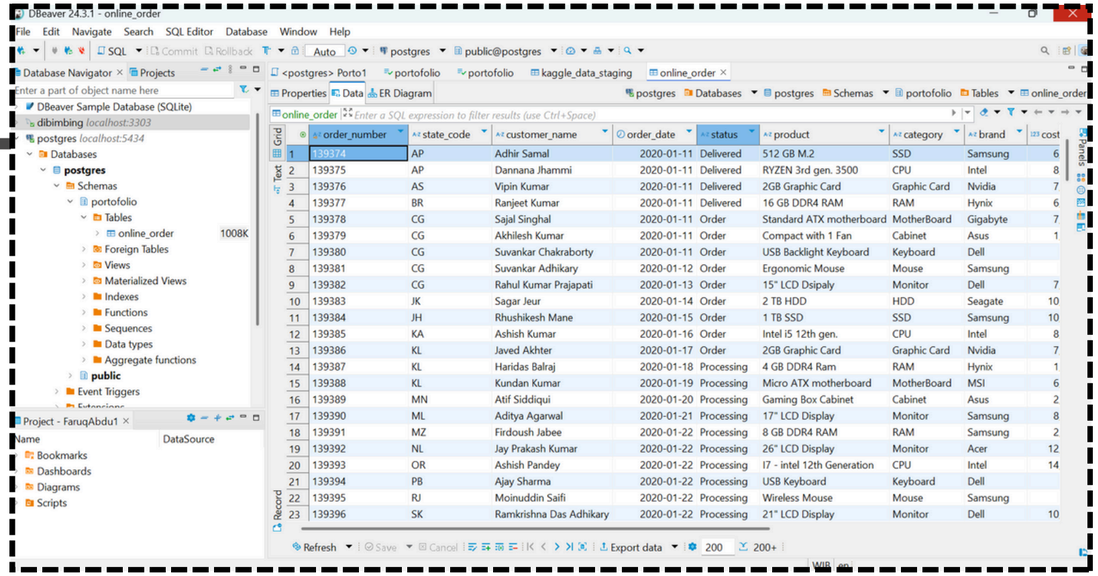


RESULTS ACHIEVED

Processed data successfully loaded into PostgreSQL







CONCLUSION & FUTURE WORK

An effective ETL pipeline for managing Kaggle data and loading it into MySQL and PostgreSQL

A workflow that can be automated using Apache Airflow

NEXT

Adding data analysis steps after transformation to generate reports

Optimizing the code to improve performance and reduce processing time

FEEDBACK & SUGGESTIONS

This project is far from perfect and efficient. Constructive feedback and suggestions are highly appreciated to improve the quality of this project.
Please provide any insights for further development.

FEEDBACK & SUGGESTIONS

This project is far from perfect and efficient. Constructive feedback and suggestions are highly appreciated to improve the quality of this project.
Please provide any insights for further development.

THANK YOU