

# **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

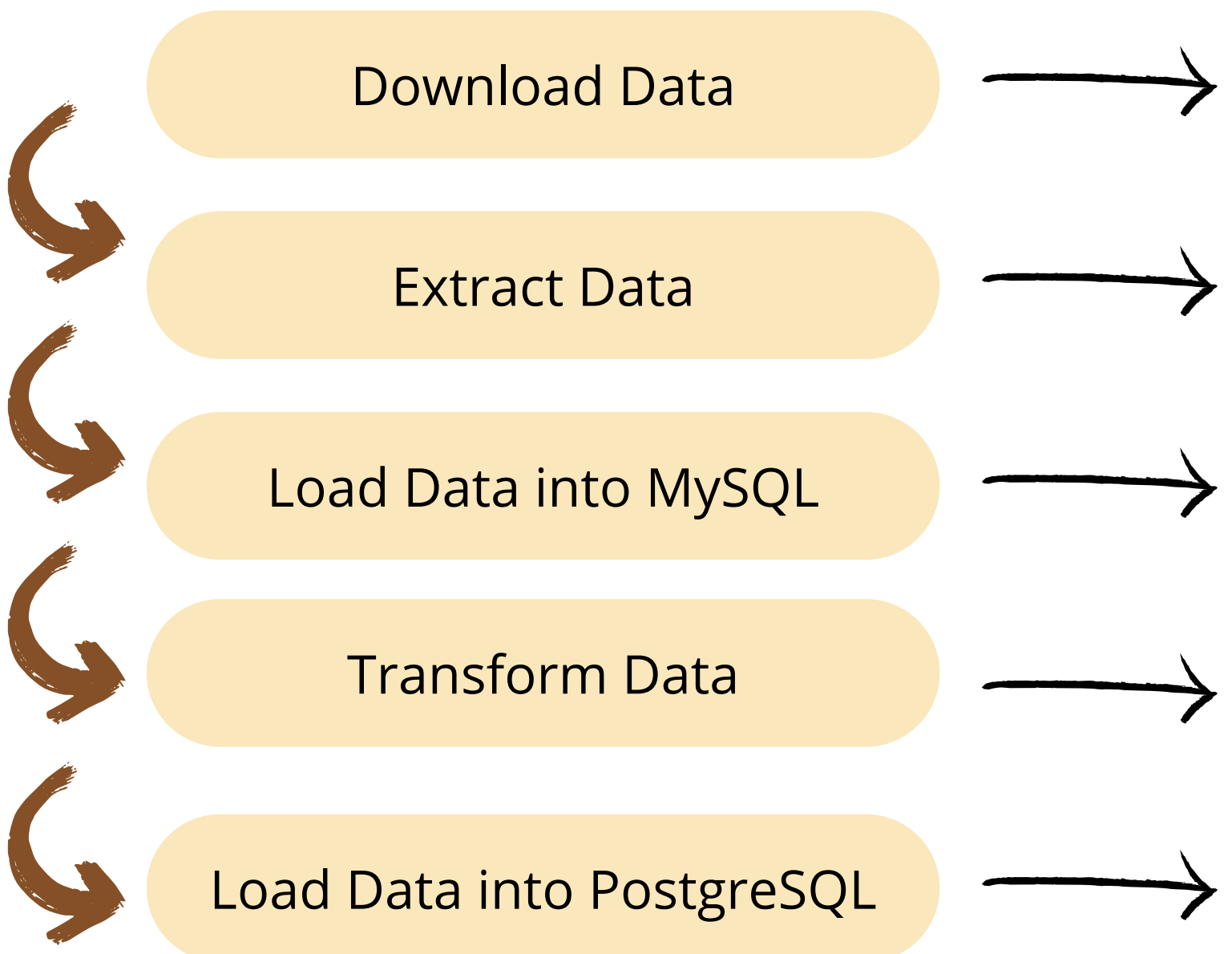
Automation of the ETL workflow

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



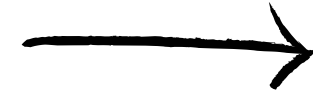
Store data in the staging area

PostgreSQL



Store transformed data

Pandas



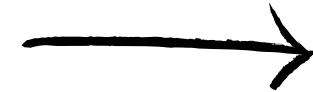
Data manipulation and transformation

Python



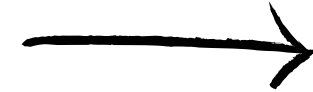
Programming for writing ETL scripts

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

```
1 def download_kaggle_data(**kwargs):
2     """
3     Download dataset from Kaggle using the Kaggle API and return the file path.
4     """
5     os.environ['KAGGLE_CONFIG_DIR'] = "/opt/airflow/.kaggle"
6
7     dataset_name = "ayushparwal2026/online-ecommerce"
8     download_path = "/opt/airflow/data/online-ecommerce"
9     os.makedirs(download_path, exist_ok=True)
10    os.system(f"kaggle datasets download -d {dataset_name} -p {download_path} --unzip")
11
12    file_path = os.path.join(download_path, "Online-eCommerce.csv")
13    kwargs['ti'].xcom_push(key='file_path', value=file_path)
14    print("Data downloaded to:", file_path)
15
```



## 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

```
1 def extract_data(**kwargs):
2     """
3     Load CSV data into a pandas DataFrame.
4     """
5     file_path = kwargs['ti'].xcom_pull(task_ids='download_kaggle_data', key='file_path')
6     print(f"Extracting data from file: {file_path}")
7     data = pd.read_csv(file_path)
8     print("Data extracted successfully")
9     kwargs['ti'].xcom_push(key='extracted_data', value=data.to_dict('records'))
10
```



## 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

```
1 def load_data_to_mysql(**kwargs):
2     """
3     Load new data into the MySQL staging area after checking for duplicates.
4     """
5     extracted_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='extract_data', key='extracted_data'))
6
7     if 'Order_Number' not in extracted_data.columns:
8         raise ValueError("Column 'Order_Number' not found in the provided data.")
9
10    # Use MySqlHook for MySQL connection
11    mysql_hook = MySqlHook(mysql_conn_id='mysql_dibimbing')
12    engine = mysql_hook.get_sqlalchemy_engine()
13
14    with engine.connect() as connection:
15        query = "SELECT Order_Number FROM kaggle_data_staging"
16        existing_data = pd.read_sql(query, connection)
17        new_data = extracted_data[~extracted_data['Order_Number'].isin(existing_data['Order_Number'])]
18
19    if not new_data.empty:
20        new_data.to_sql('kaggle_data_staging', con=engine, if_exists='append', index=False)
21        print(f"{len(new_data)} new rows added to MySQL staging area.")
22    else:
23        print("No new data to add to MySQL.")
24
25    kwargs['ti'].xcom_push(key='staged_data', value=new_data.to_dict('records'))
26
```



## 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

```
1 def transform_data(**kwargs):
2     """
3     Perform all necessary data transformations including formatting, renaming columns,
4     handling missing values, and ensuring data consistency.
5     """
6     staged_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='load_data_to_mysql', key='staged_data'))
7
8     # Check if 'Order_Date' exists in the data
9     if 'Order_Date' not in staged_data.columns:
10         print("Warning: 'Order_Date' column is missing!")
11         return # Skip the transformation if the column is missing
12
13     # Remove duplicates
14     staged_data.drop_duplicates(inplace=True)
15
16     # Format Order_Date to YYYY-MM-DD
17     staged_data['Order_Date'] = pd.to_datetime(staged_data['Order_Date'], format='%d/%m/%Y', errors='coerce').dt.strftime('%Y-%m-%d')
18
19     # Rename columns for PostgreSQL compatibility
20     staged_data.rename(columns={"Assigned Supervisor": "Assigned_Supervisor"}, inplace=True)
21
22     # Drop rows with missing critical fields
23     staged_data.dropna(subset=['Cost', 'Sales'], inplace=True)
```

```
1 # Convert Cost and Sales to numeric values
2 staged_data['Cost'] = pd.to_numeric(staged_data['Cost'], errors='coerce')
3 staged_data['Sales'] = pd.to_numeric(staged_data['Sales'], errors='coerce')
4
5 # Ensure Order_Number is treated as string
6 staged_data['Order_Number'] = staged_data['Order_Number'].astype(int).astype(str)
7
8 # Handle missing values in non-critical columns
9 staged_data.fillna({
10     'State_Code': 'Unknown',
11     'Customer_Name': 'Unknown',
12     'Status': 'Unknown',
13     'Product': 'Unknown',
14     'Category': 'Unknown',
15     'Brand': 'Unknown',
16     'Quantity': 0.0,
17     'Total_Cost': 0.0,
18     'Total_Sales': 0.0,
19     'Assigned_Supervisor': 'Unknown'
20 }, inplace=True)
21
22 print("Data transformed successfully")
23 kwargs['ti'].xcom_push(key='transformed_data', value=staged_data.to_dict('records'))
24
```

## 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

```
1 def load_transformed_data_to_postgresql(**kwargs):
2     """
3     Load transformed data from pandas DataFrame into PostgreSQL.
4     """
5     # Mendapatkan data yang ditransformasi menggunakan XCom
6     transformed_data = pd.DataFrame(kwargs['ti'].xcom_pull(task_ids='transform_data', key='transformed_data'))
7
8     # Menggunakan PostgresHook untuk koneksi ke PostgreSQL
9     postgres_hook = PostgresHook(postgres_conn_id="postgres_dibimbing")
10    postgres_engine = postgres_hook.get_sqlalchemy_engine()
11
12    upsert_sql = """
13    INSERT INTO portofolio.online_order (
14        Order_Number, State_Code, Customer_Name, Order_Date, Status, Product,
15        Category, Brand, Cost, Sales, Quantity, Total_Cost, Total_Sales, Assigned_Supervisor
16    )
17    VALUES (
18        :Order_Number, :State_Code, :Customer_Name, :Order_Date, :Status, :Product,
19        :Category, :Brand, :Cost, :Sales, :Quantity, :Total_Cost, :Total_Sales, :Assigned_Supervisor
20    )
21    ON CONFLICT (Order_Number) DO NOTHING;
22    """
23
24    # Membuka koneksi dan melakukan eksekusi
25    with postgres_engine.connect() as connection:
26        # Iterasi melalui setiap baris data dan eksekusi upsert query
27        for _, row in transformed_data.iterrows():
28            connection.execute(text(upsert_sql), row.to_dict())
29
30    print("Transformed data successfully loaded into PostgreSQL.")
```

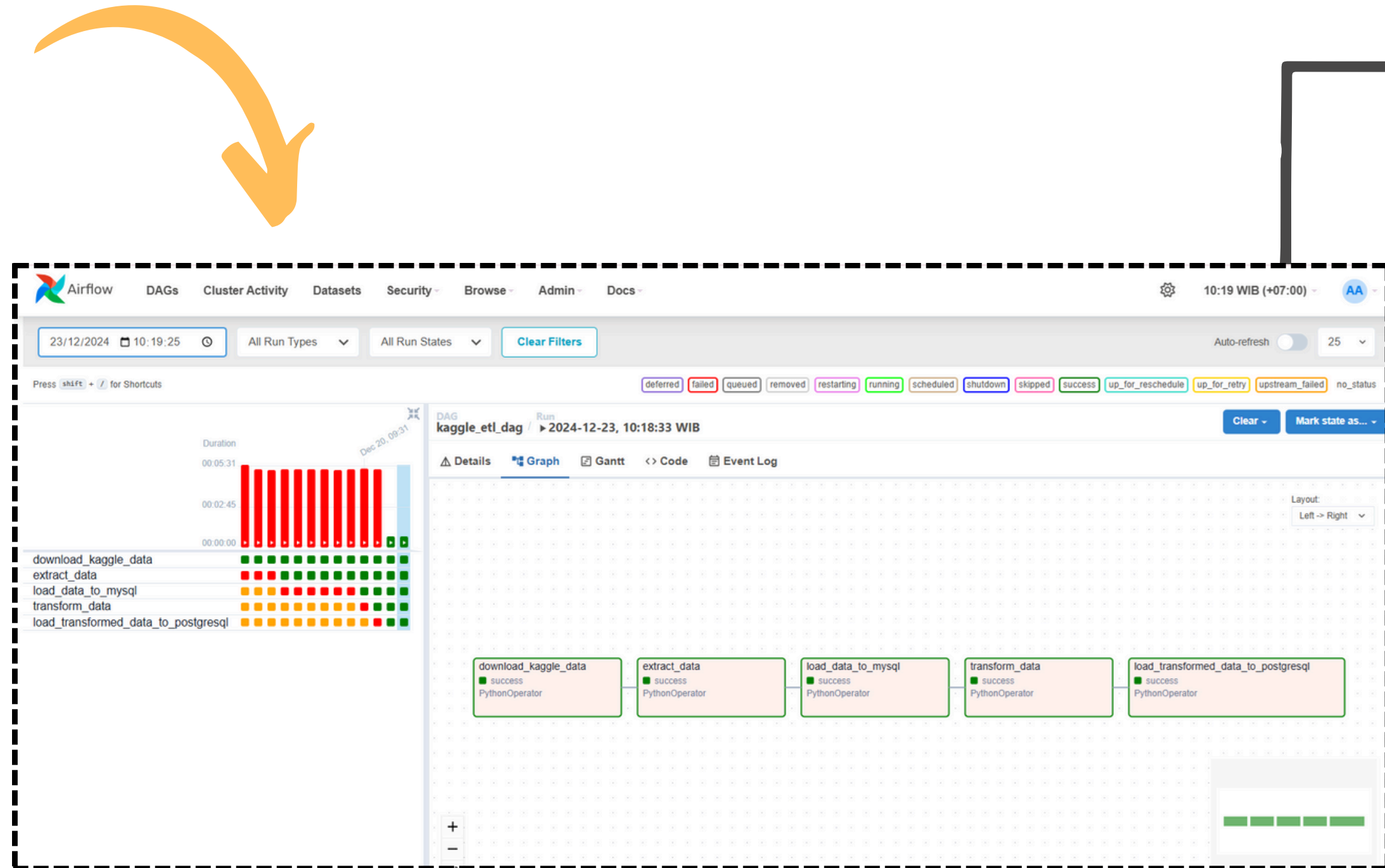
## Process:

- 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

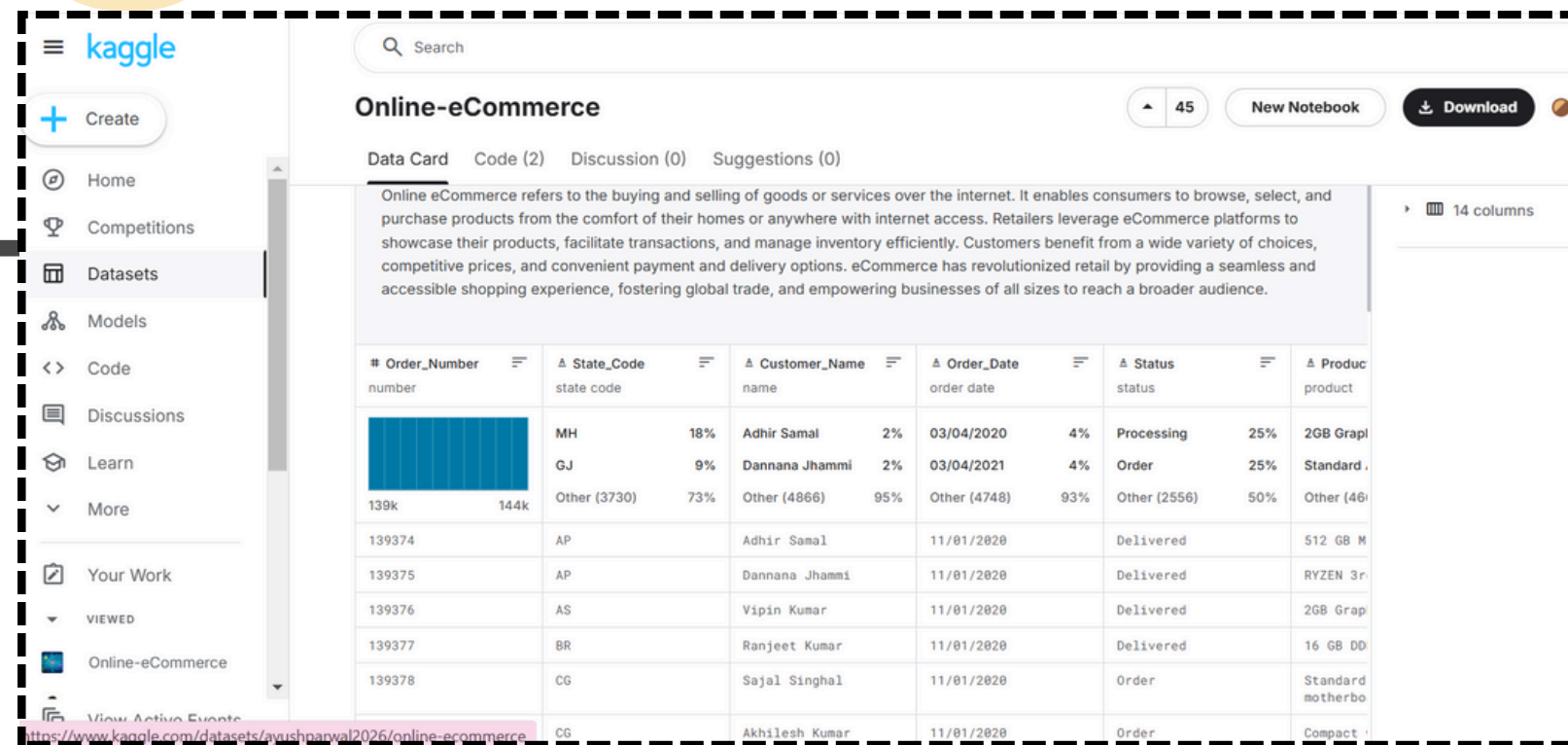
```
1 # Define the DAG
2 with DAG(
3     'kaggle_etl_dag',
4     default_args={
5         'owner': 'burqi',
6         'retries': 1,
7         'retry_delay': timedelta(minutes=5),
8     },
9     description='ETL process for Kaggle dataset',
10    schedule_interval=None,
11    start_date=datetime(2024, 12, 17),
12    catchup=False,
13 ) as dag:
14
15     # Task 1: Download Data from Kaggle
16     download_task = PythonOperator(
17         task_id='download_kaggle_data',
18         python_callable=download_kaggle_data,
19     )
20
21     # Task 2: Extract Data from CSV
22     extract_task = PythonOperator(
23         task_id='extract_data',
24         python_callable=extract_data,
25         provide_context=True,
26     )
27
28     # Task 3: Load Data to MySQL
29     load_mysql_task = PythonOperator(
30         task_id='load_data_to_mysql',
31         python_callable=load_data_to_mysql,
32         provide_context=True,
33     )
34
35     # Task 4: Transform Data
36     transform_task = PythonOperator(
37         task_id='transform_data',
38         python_callable=transform_data,
39         provide_context=True,
40     )
41
42     # Task 5: Load Transformed Data to PostgreSQL
43     load_postgresql_task = PythonOperator(
44         task_id='load_transformed_data_to_postgresql',
45         python_callable=load_transformed_data_to_postgresql,
46         provide_context=True,
47     )
48
49     # Set task dependencies
50     download_task >> extract_task >> load_mysql_task >> transform_task >> load_postgresql_task
```





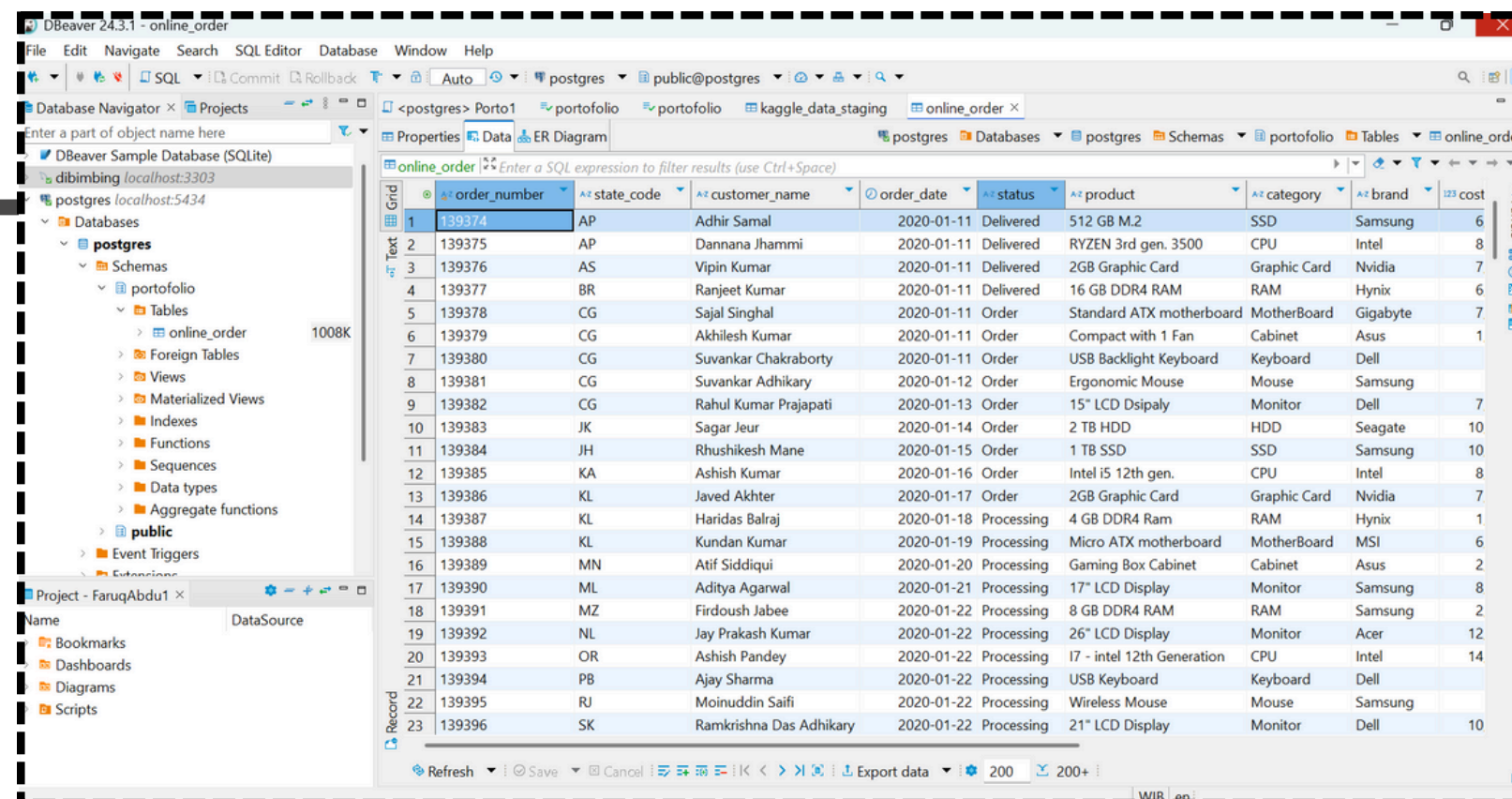
# RESULTS ACHIEVED

Processed data successfully loaded into PostgreSQL



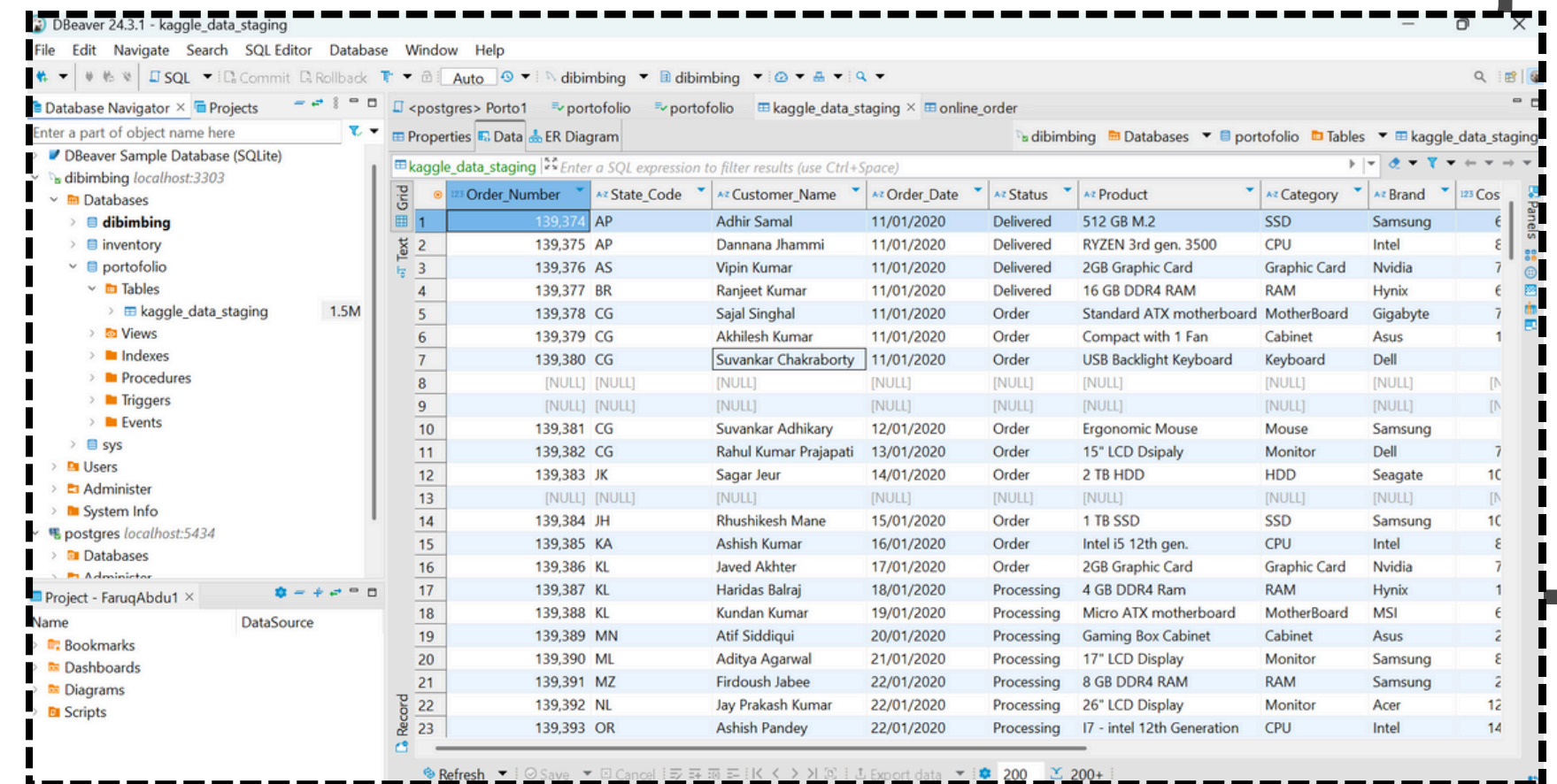
The image shows a preview of the 'Online-eCommerce' dataset on Kaggle. It includes a description of online eCommerce and a table with 14 columns: Order\_Number, State\_Code, Customer\_Name, Order\_Date, Status, Product, Category, Brand, and Cost. The table shows a summary of orders and a detailed list of individual orders.

#	Order_Number	State_Code	Customer_Name	Order_Date	Status	Product	Category	Brand	Cost
1	139374	AP	Adhir Samal	11/01/2020	Delivered	512 GB M.2	SSD	Samsung	6
2	139375	AP	Dannana Jhammi	11/01/2020	Delivered	RYZEN 3rd gen. 3500	CPU	Intel	8
3	139376	AS	Vipin Kumar	11/01/2020	Delivered	2GB Graphic Card	Graphic Card	Nvidia	7
4	139377	BR	Ranjeet Kumar	11/01/2020	Delivered	16 GB DDR4 RAM	RAM	Hynix	6
5	139378	CG	Sajal Singhal	11/01/2020	Order	Standard ATX motherboard	MotherBoard	Gigabyte	7
6	139379	CG	Akhilesh Kumar	11/01/2020	Order	Compact with 1 Fan	Cabinet	Asus	1
7	139380	CG	Suvankar Chakraborty	11/01/2020	Order	USB Backlight Keyboard	Keyboard	Dell	6
8	139381	CG	Suvankar Adhikary	2020-01-12	Order	Ergonomic Mouse	Mouse	Samsung	7
9	139382	CG	Rahul Kumar Prajapati	2020-01-13	Order	15" LCD Dsipaly	Monitor	Dell	7
10	139383	JK	Sagar Jeur	2020-01-14	Order	2 TB HDD	HDD	Seagate	10
11	139384	JH	Rhushikesh Mane	2020-01-15	Order	1 TB SSD	SSD	Samsung	10
12	139385	KA	Ashish Kumar	2020-01-16	Order	Intel i5 12th gen.	CPU	Intel	8
13	139386	KL	Javed Akhter	2020-01-17	Order	2GB Graphic Card	Graphic Card	Nvidia	7
14	139387	KL	Haridas Balraj	2020-01-18	Processing	4 GB DDR4 Ram	RAM	Hynix	1
15	139388	KL	Kundan Kumar	2020-01-19	Processing	Micro ATX motherboard	MotherBoard	MSI	6
16	139389	MN	Atif Siddiqui	2020-01-20	Processing	Gaming Box Cabinet	Cabinet	Asus	2
17	139390	ML	Aditya Agarwal	2020-01-21	Processing	17" LCD Display	Monitor	Samsung	8
18	139391	MZ	Firdoush Jabee	2020-01-22	Processing	8 GB DDR4 RAM	RAM	Samsung	2
19	139392	NL	Jay Prakash Kumar	2020-01-22	Processing	26" LCD Display	Monitor	Acer	12
20	139393	OR	Ashish Pandey	2020-01-22	Processing	I7 - intel 12th Generation	CPU	Intel	14
21	139394	PB	Ajay Sharma	2020-01-22	Processing	USB Keyboard	Keyboard	Dell	6
22	139395	RJ	Moinuddin Saifi	2020-01-22	Processing	Wireless Mouse	Mouse	Samsung	2
23	139396	SK	Ramkrishna Das Adhikary	2020-01-22	Processing	21" LCD Display	Monitor	Dell	10



The image shows a screenshot of DBeaver 24.3.1. The 'Database Navigator' on the left shows the 'postgres' database. The 'SQL Editor' on the right shows a table named 'online\_order' with 23 rows of data. The data is identical to the table shown in the Kaggle dataset preview.

#	order_number	state_code	customer_name	order_date	status	product	category	brand	cost
1	139374	AP	Adhir Samal	2020-01-11	Delivered	512 GB M.2	SSD	Samsung	6
2	139375	AP	Dannana Jhammi	2020-01-11	Delivered	RYZEN 3rd gen. 3500	CPU	Intel	8
3	139376	AS	Vipin Kumar	2020-01-11	Delivered	2GB Graphic Card	Graphic Card	Nvidia	7
4	139377	BR	Ranjeet Kumar	2020-01-11	Delivered	16 GB DDR4 RAM	RAM	Hynix	6
5	139378	CG	Sajal Singhal	2020-01-11	Order	Standard ATX motherboard	MotherBoard	Gigabyte	7
6	139379	CG	Akhilesh Kumar	2020-01-11	Order	Compact with 1 Fan	Cabinet	Asus	1
7	139380	CG	Suvankar Chakraborty	2020-01-11	Order	USB Backlight Keyboard	Keyboard	Dell	6
8	139381	CG	Suvankar Adhikary	2020-01-12	Order	Ergonomic Mouse	Mouse	Samsung	7
9	139382	CG	Rahul Kumar Prajapati	2020-01-13	Order	15" LCD Dsipaly	Monitor	Dell	7
10	139383	JK	Sagar Jeur	2020-01-14	Order	2 TB HDD	HDD	Seagate	10
11	139384	JH	Rhushikesh Mane	2020-01-15	Order	1 TB SSD	SSD	Samsung	10
12	139385	KA	Ashish Kumar	2020-01-16	Order	Intel i5 12th gen.	CPU	Intel	8
13	139386	KL	Javed Akhter	2020-01-17	Order	2GB Graphic Card	Graphic Card	Nvidia	7
14	139387	KL	Haridas Balraj	2020-01-18	Processing	4 GB DDR4 Ram	RAM	Hynix	1
15	139388	KL	Kundan Kumar	2020-01-19	Processing	Micro ATX motherboard	MotherBoard	MSI	6
16	139389	MN	Atif Siddiqui	2020-01-20	Processing	Gaming Box Cabinet	Cabinet	Asus	2
17	139390	ML	Aditya Agarwal	2020-01-21	Processing	17" LCD Display	Monitor	Samsung	8
18	139391	MZ	Firdoush Jabee	2020-01-22	Processing	8 GB DDR4 RAM	RAM	Samsung	2
19	139392	NL	Jay Prakash Kumar	2020-01-22	Processing	26" LCD Display	Monitor	Acer	12
20	139393	OR	Ashish Pandey	2020-01-22	Processing	I7 - intel 12th Generation	CPU	Intel	14
21	139394	PB	Ajay Sharma	2020-01-22	Processing	USB Keyboard	Keyboard	Dell	6
22	139395	RJ	Moinuddin Saifi	2020-01-22	Processing	Wireless Mouse	Mouse	Samsung	2
23	139396	SK	Ramkrishna Das Adhikary	2020-01-22	Processing	21" LCD Display	Monitor	Dell	10



The image shows a screenshot of DBeaver 24.3.1. The 'Database Navigator' on the left shows the 'postgres' database. The 'SQL Editor' on the right shows a table named 'online\_order' with 23 rows of data. The data is identical to the table shown in the Kaggle dataset preview.

#	Order_Number	State_Code	Customer_Name	Order_Date	Status	Product	Category	Brand	Cost
1	139374	AP	Adhir Samal	11/01/2020	Delivered	512 GB M.2	SSD	Samsung	6
2	139375	AP	Dannana Jhammi	11/01/2020	Delivered	RYZEN 3rd gen. 3500	CPU	Intel	8
3	139376	AS	Vipin Kumar	11/01/2020	Delivered	2GB Graphic Card	Graphic Card	Nvidia	7
4	139377	BR	Ranjeet Kumar	11/01/2020	Delivered	16 GB DDR4 RAM	RAM	Hynix	6
5	139378	CG	Sajal Singhal	11/01/2020	Order	Standard ATX motherboard	MotherBoard	Gigabyte	7
6	139379	CG	Akhilesh Kumar	11/01/2020	Order	Compact with 1 Fan	Cabinet	Asus	1
7	139380	CG	Suvankar Chakraborty	11/01/2020	Order	USB Backlight Keyboard	Keyboard	Dell	6
8	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
9	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
10	139381	CG	Suvankar Adhikary	12/01/2020	Order	Ergonomic Mouse	Mouse	Samsung	7
11	139382	CG	Rahul Kumar Prajapati	13/01/2020	Order	15" LCD Dsipaly	Monitor	Dell	7
12	139383	JK	Sagar Jeur	14/01/2020	Order	2 TB HDD	HDD	Seagate	10
13	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
14	139384	JH	Rhushikesh Mane	15/01/2020	Order	1 TB SSD	SSD	Samsung	10
15	139385	KA	Ashish Kumar	16/01/2020	Order	Intel i5 12th gen.	CPU	Intel	8
16	139386	KL	Javed Akhter	17/01/2020	Order	2GB Graphic Card	Graphic Card	Nvidia	7
17	139387	KL	Haridas Balraj	18/01/2020	Processing	4 GB DDR4 Ram	RAM	Hynix	1
18	139388	KL	Kundan Kumar	19/01/2020	Processing	Micro ATX motherboard	MotherBoard	MSI	6
19	139389	MN	Atif Siddiqui	20/01/2020	Processing	Gaming Box Cabinet	Cabinet	Asus	2
20	139390	ML	Aditya Agarwal	21/01/2020	Processing	17" LCD Display	Monitor	Samsung	8
21	139391	MZ	Firdoush Jabee	22/01/2020	Processing	8 GB DDR4 RAM	RAM	Samsung	2
22	139392	NL	Jay Prakash Kumar	22/01/2020	Processing	26" LCD Display	Monitor	Acer	12
23	139393	OR	Ashish Pandey	22/01/2020	Processing	I7 - intel 12th Generation	CPU	Intel	14

# 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**