**1. Data Warehouse Structure:**

The following directory structure is used for the data warehouse:

D:\Ali\REVEST\data\_warehouse

├── data

│ ├── sales.csv

├── scripts

│ ├── export\_to\_parquet.py

│ ├── ingest\_data.py

│ ├── queries.sql

│ ├── sales\_data.parquet

├── docker-compose.yml

└── Dockerfile

**2. Files and Their Purpose:**

1. **D:\Ali\REVEST\data\_warehouse\data\sales.csv**:
   * This is the source file containing sales data in CSV format. It includes various columns like order\_id, product\_id, quantity, sales, etc.
2. **Scripts Folder** (D:\Ali\REVEST\data\_warehouse\scripts):
   * **ingest\_data.py**:
     + This Python script ingests and preprocesses the sales data from sales.csv into a PostgreSQL database. It handles reading the CSV file, transforming the data if needed, and inserting it into the database.
   * **queries.sql**:
     + Contains SQL queries to extract the following insights from the sales data:
       1. The average total value of an order.
       2. The average total revenue for each month of the year.
   * **export\_to\_parquet.py**:
     + This Python script exports the data from the PostgreSQL database into a Parquet file for optimized analytics and storage.
   * **sales\_data.parquet**:
     + The output Parquet file that contains the transformed data for future analytical tasks.
3. **docker-compose.yml**:
   * Defines the services for running the database and the ingestion/processing tasks using Docker containers. It connects the PostgreSQL database and facilitates the execution of tasks within containers.
4. **Dockerfile**:
   * Specifies the necessary steps to containerize the environment, ensuring that all dependencies for data ingestion and processing are available for deployment in a Docker container.

**3. Task Accomplished:**

1. **Ingest and Pre-process Data**:
   * The ingest\_data.py script was created to load and clean the sales data into the database for further analysis.
2. **SQL Queries**:
   * The queries.sql file contains SQL queries that calculate the:
     1. **Average Total Order Value**: Using the sales data.
     2. **Average Monthly Revenue**: Extracting revenue information month-wise from the data.
3. **Export Data**:
   * The export\_to\_parquet.py script exports the data into a Parquet file for more efficient storage and analysis, which is saved as sales\_data.parquet.
4. **Docker Setup**:
   * Dockerfiles and docker-compose were created to facilitate running the environment in containers, making the solution portable and replicable.

**1. Model Deployment Structure:**

The following directory structure is used for the model deployment:

makefile

CopyEdit

D:\Ali\REVEST\model\_deployment

├── app.py

├── docker-compose.yml

├── Dockerfile

├── logging.db

└── requirements.txt

**2. Files and Their Purpose:**

1. **app.py**:
   * This Python file contains the Flask application that serves as the REST API for the recommendation service. It provides the following functionality:
     + Receives a product ID via a POST request.
     + Logs the incoming request and recommendation in a database.
     + Returns a static set of recommended products [1, 2, 3] for the given product ID.
2. **docker-compose.yml**:
   * Defines the services for running the application and the PostgreSQL logging database in Docker containers. It ensures that the application can connect to the logging database and facilitates communication between the services.
3. **Dockerfile**:
   * Specifies the steps to build the Docker container for the Flask application. It installs dependencies from requirements.txt, sets up the environment, and runs the Flask app.
4. **logging.db**:
   * This SQLite database stores the logs for incoming requests and responses generated by the Flask app. It logs details such as the timestamp, product ID, recommended product, and log type (INFO/ERROR).
5. **requirements.txt**:
   * Lists the Python dependencies required for running the Flask app. This includes libraries like Flask, psycopg2, SQLAlchemy, and others.

**3. Task Accomplished:**

1. **REST API Development**:
   * The Flask application (app.py) was created to expose a /recommend endpoint. This endpoint receives a product ID and returns a static list of recommended product IDs [1, 2, 3]. The app also logs requests and responses to a logging database.
2. **Logging Implementation**:
   * The logging functionality was integrated into the application using an SQLite database (logging.db). The logs capture incoming requests and the associated recommendations, including error logs in case of issues.
3. **Docker Setup**:
   * A Dockerfile and docker-compose.yml were created to containerize the Flask application and PostgreSQL logging service, ensuring the environment is easily deployable.
4. **SQLite Logging Database**:
   * The logging.db SQLite database was used to store logs of the product recommendations and request details. This helps in keeping track of system activity.