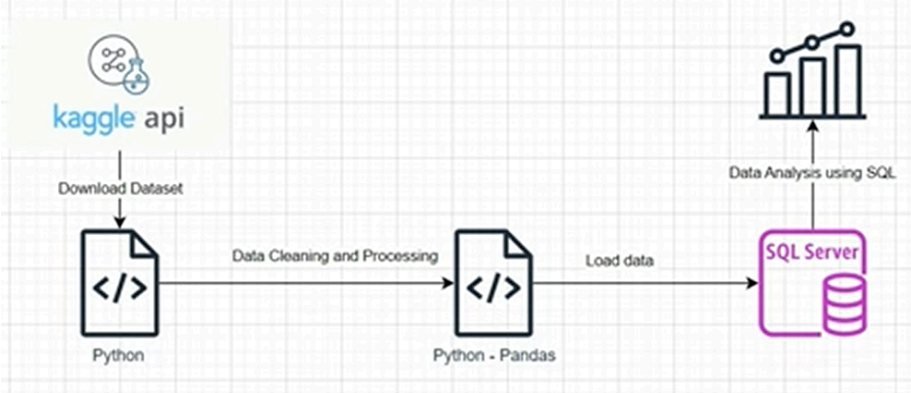
End to End Data Project (ETL)

TOPOLOGY



In this section, we'll build and execute an **ETL (Extract, Transform, Load)** workflow — a core data engineering process used to move data from its source to a destination where it can be analyzed and visualized.

- **Extract** – Retrieve raw data from various sources such as APIs, databases, or files.
- **Transform** – Clean, format, and structure the data to make it consistent and meaningful.
- **Load** – Store the processed data into a target system, such as a database or data warehouse.

Our **first step** will be to **extract the data** from the source. We'll connect to the dataset, verify access and structure, and ensure it's ready for transformation in the next phase.

Part 1: EXTRACT

Summary

- Initial environment setup in PowerShell
- Creating Kaggle API credentials
- Install tools and launch Jupyter lab from PowerShell
- Create Jupyter notebook and verify Kaggle CLI access and data set

---

Configure Environment in Powershell

1. Launch PowerShell & Navigate to the Project

- Open Windows PowerShell (Start Menu → type PowerShell → Enter).
- Choose or create a folder for your project, for example:

```
cd C:\Users\<YourUser>\Projects
mkdir end-to-end-retail-analytics
cd end-to-end-retail-analytics
```

2. Create and Activate a Virtual Environment

- Create a virtual environment:

```
python -m venv .venv
```

- Activate it:

```
.\.venv\Scripts\Activate
```

- Confirm the environment is active: your prompt should now start with `(.venv)`.

3. Install Required Python Packages

- Install the core tools needed for extraction and exploration:

```
pip install kaggle jupyterlab pandas
```

- You can also install any additional libraries later as needed.

Set up Kaggle API Credentials

Kaggle README Instructions

API credentials

To use the Kaggle API, sign up for a Kaggle account at https://www.kaggle.com. Then go to the 'Account' tab of your user profile (`https://www.kaggle.com/<username>/account`) and select 'Create API Token'. This will trigger the download of `kaggle.json`, a file containing your API credentials. Place this file in the location appropriate for your operating system:

- Linux: `$XDG_CONFIG_HOME/kaggle/kaggle.json` (defaults to `~/.config/kaggle/kaggle.json`). The path `~/.kaggle/kaggle.json` which was used by older versions of the tool is also still supported.
- Windows: `C:\Users\<Windows-username>\.kaggle\kaggle.json` - you can check the exact location, sans drive, with `echo %HOMEPATH%`.
- Other: `~/.kaggle/kaggle.json`

You can define a shell environment variable `KAGGLE_CONFIG_DIR` to change this location to `$KAGGLE_CONFIG_DIR/kaggle.json` (on Windows it will be `%KAGGLE_CONFIG_DIR%\kaggle.json`).

For your security, ensure that other users of your computer do not have read access to your credentials. On Unix-based systems you can do this with the following command:

```
chmod 600 ~/.config/kaggle/kaggle.json
```

You can also choose to export your Kaggle username and token to the environment:

```
export KAGGLE_USERNAME=datadinosaur
export KAGGLE_KEY=xxxxxxxxxxxxxx
```

In addition, you can export any other configuration value that normally would be in the `kaggle.json` in the format 'KAGGLE_' (note uppercase). For example, if the file had the variable "proxy" you would export `KAGGLE_PROXY` and it would be discovered by the client.

4. Configure Kaggle API (One-Time Setup)

4.1 Get Your Kaggle API Token

- Go to Kaggle in your browser and sign in.
- Click your profile icon → Settings.
- Scroll to API → click Create New API Token.
- A file named kaggle.json will download.

4.2 Place kaggle.json in the Correct Location (Windows)

- In File Explorer, go to:

  C:\Users\<YourUser>\

- If it doesn't exist, create a folder:

  .kaggle

- Move kaggle.json into:

  C:\Users\<YourUser>\.kaggle\

- (Optional but recommended) Right-click kaggle.json → Properties → check Hidden → OK
  This keeps the key out of casual view.

5. Verify Kaggle CLI Configuration

Back in your activated PowerShell session:

```
kaggle datasets list -s retail
```

- If you see a list of datasets, your API is configured correctly.
- If you get an error about credentials or kaggle not found:
  - Make sure kaggle.json is in C:\Users\<YourUser>\.kaggle\
  - Make sure you're in the .venv and kaggle is installed (pip install kaggle)

6. Download the Dataset with Kaggle API

Inside your project folder:

- Create a folder to store raw data:

```
mkdir data_raw
```

- Download and unzip the dataset (example uses a retail orders dataset):

```
kaggle datasets download -d ankitbansal06/retail-orders -p
data_raw --unzip
```

- `-d` specifies the dataset.
- `-p data_raw` saves files into the data_raw folder.
- `--unzip` extracts the zip automatically.

At this point, the Extract from Kaggle → local environment is complete.

7. Launch JupyterLab from the Project Environment

From the same folder, with `.venv` still active:

```
jupyter lab
```

- This will open JupyterLab in your browser.
- Make sure the URL points to your project directory.

If prompted to choose a kernel, select the one associated with `.venv`.

8. Connect and Inspect the Data in Jupyter

In JupyterLab:

- Create a new Python 3 notebook.
- Run:

```
import pandas as pd
```

```
file_path = "data_raw/Retail Orders.csv"  # update to the actual
filename
df = pd.read_csv(file_path)

df.head()
df.info()
```

This confirms:

- The Kaggle API connection worked
- The dataset is successfully extracted and readable
- The notebook is running inside the correct environment

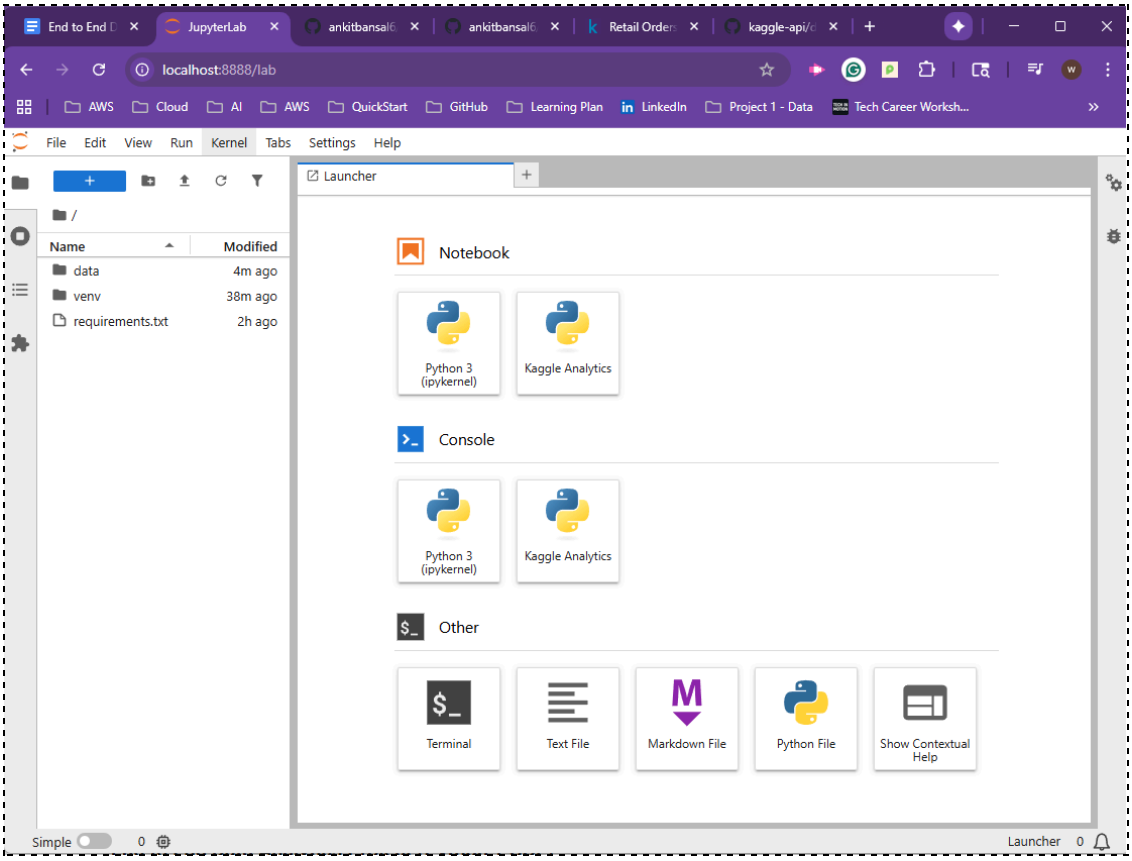
Part 2: Transform

Description:

Once your data is successfully loaded, the **next phase** is **Transform**, where you'll:

- Clean missing or inconsistent values,
- Format data types,
- Derive new columns or metrics, and
- Prepare the data for analysis or loading into a database.

Set Up Jupyter Environment



✅ **Step 1: Create your new notebook**

You're in the **Launcher** tab, which is perfect.
Under "**Notebook**", click on:

> **Kaggle Analytics**

That will open a new tab named something like `Untitled.ipynb`.

This ensures that your notebook runs inside the **virtual environment** (`venv`) where you installed your packages (pandas, matplotlib, kaggle, etc.).

✅ Step 2: Rename the notebook

In the **file browser** on the left, right-click the file `Untitled.ipynb` → click **Rename** → type:

Copy code

```
explore.ipynb
```

This keeps things organized and descriptive.

✅ Step 3: Verify the kernel

Look at the **top-right corner** of the notebook (beside the circle icon).

You should see:

> `Kaggle Analytics`

If it says anything else (like "Python 3 (ipykernel)"), click on it and **change it** to **Kaggle Analytics**.

That's your environment with all your installed libraries.

✅ Step 4: Run your first cell to confirm setup

In the first cell, paste and run this:

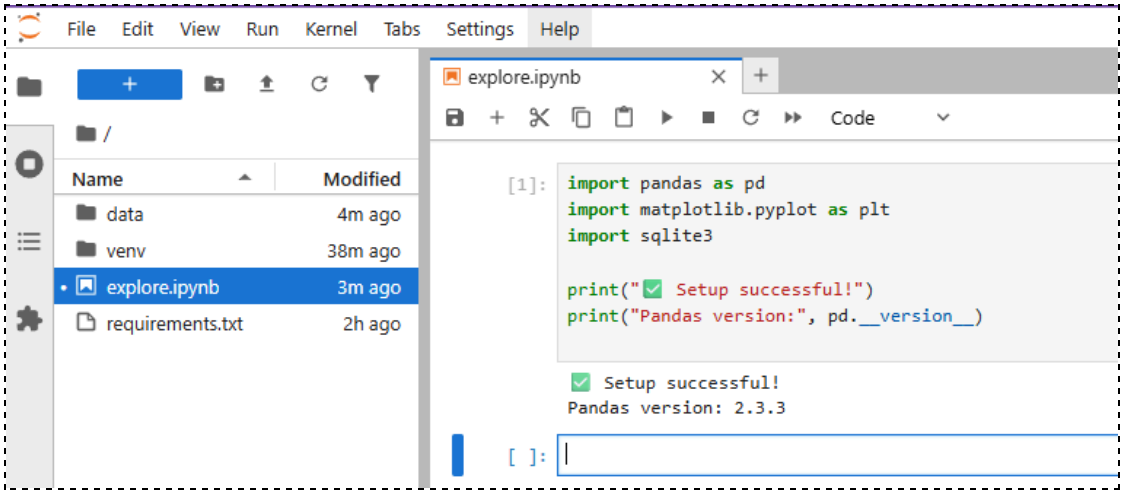
python Copy code

```python
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3

print("✅ Setup successful!")
print("Pandas version:", pd.__version__)
```

Then hit **Shift** + **Enter** to execute the cell.

If it prints the lines correctly with no errors, your environment is fully working and connected to the right Python.

Confirmation Example below.



✅ Step 5: (Optional Check) Confirm Kaggle CLI access

In a new cell, run:

```python
!kaggle --version
```

You should see something like:

```nginx
Kaggle API 1.6.x
```

That confirms your Kaggle token and CLI tool are installed correctly in this environment.

```
[2]: !kaggle --version

     Kaggle API 1.7.4.5
```

✅ Step 6: Load your dataset

Now that everything's ready, we'll start analysis.
You already downloaded your dataset earlier (the retail orders CSV).
Let's verify and load it.

```python
df = pd.read_csv("data/Retail Orders.csv")
df.head()
```

If you see a table appear, that means the dataset loaded perfectly and we can move on to exploration, cleaning, and SQL integration.

```
[5]: df = pd.read_csv("data/orders.csv")
     df.head()
```

[5]:

	Order Id	Order Date	Ship Mode	Segment	Country	City	State	Postal Code	Region	Category	Sub Category	Product Id	cost price	List Price	Quantity	Discount Percent
0	1	2023-03-01	Second Class	Consumer	United States	Henderson	Kentucky	42420	South	Furniture	Bookcases	FUR-BO-10001798	240	260	2	2
1	2	2023-08-15	Second Class	Consumer	United States	Henderson	Kentucky	42420	South	Furniture	Chairs	FUR-CH-10000454	600	730	3	3
2	3	2023-01-10	Second Class	Corporate	United States	Los Angeles	California	90036	West	Office Supplies	Labels	OFF-LA-10000240	10	10	2	5
3	4	2022-06-18	Standard Class	Consumer	United States	Fort Lauderdale	Florida	33311	South	Furniture	Tables	FUR-TA-10000577	780	960	5	2
4	5	2022-07-13	Standard Class	Consumer	United States	Fort Lauderdale	Florida	33311	South	Office Supplies	Storage	OFF-ST-10000760	20	20	2	5

Part 3: Clean

Description:

After extracting the dataset from Kaggle, the next step was to clean and prepare it for analysis. The raw data contained missing values, inconsistent formatting, and unnecessary columns. Using Pandas in JupyterLab, I performed a series of data cleaning tasks to ensure accuracy and consistency across all records.

Key Steps:

1. Loaded the dataset into a Pandas DataFrame for inspection using df.info() and df.describe().

2. Handled missing values by filling or removing null entries depending on the column's significance.
3. Standardized data types for date, numeric, and categorical fields to ensure compatibility for SQL import and calculations.
4. Removed duplicates and irrelevant columns that did not contribute to the analysis.
5. Renamed columns for readability and uniformity (e.g., converting spaces to underscores).
6. Created new derived columns such as total revenue (quantity * unit_price) and profit margin to support later analysis.
7. Validated the dataset by re-checking row counts, null values, and unique identifiers before loading it into SQL Server.

This process produced a clean, structured dataset ready for the Transform and Load phases of the pipeline.

Data Cleaning Process for this project data:

Handle Null Values

```
[17]: #Handle Null Values
      df = pd.read_csv("../data/orders.csv",na_values=['Not Available','unknown'])
      df['Ship Mode'].unique()
```

```
[17]: array(['Second Class', 'Standard Class', nan, 'First Class', 'Same Day'],
            dtype=object)
```

Format Columns

```
[21]: #rename columns, make them lower case, and replace space with underscore
      df.rename(columns={'Order Id':'order_id', 'City':'city'})
      df.columns=df.columns.str.lower()
      df.columns=df.columns.str.replace(' ','_')
      df.head(5)
```

Order Id	Order Date	Ship Mode	Segment	Country

to

order_id	order_date	ship_mode	segment	country

Derive new columns

```
[23]: #derive new columns discount, sale price, and profit
      df['discount']=df['list_price']*df['discount_percent']*.01
      df['sale_price']= df['list_price']-df['discount']
      df['profit']=df['sale_price']-df['cost_price']
      df
```

cost_price	list_price	quantity	discount_percent	discount	sale_price	profit
240	260	2	2	5.2	254.8	14.8
600	730	3	3	21.9	708.1	108.1
10	10	2	5	0.5	9.5	-0.5

Check Data Types

[24]:
```
#check data types
df.dtypes
```

[24]:
```
order_id              int64
order_date           object
ship_mode            object
segment              object
country              object
city                 object
state                object
postal_code           int64
region               object
category             object
sub_category         object
product_id           object
cost_price            int64
list_price            int64
quantity              int64
discount_percent      int64
discount            float64
sale_price          float64
profit              float64
dtype: object
```

Convert Order Date Data Type

[25]:
```
#convert order date from object data type to datetime
df['order_date']=pd.to_datetime(df['order_date'],format="%Y-%m-%d")
```

[26]:
```
df.dtypes
```

[26]:
```
order_id                     int64
order date          datetime64[ns]
```

Drop Unnecessary Columns

```
#drop cost price list price and discount percent columns
df.drop(columns=['list_price','cost_price','discount_percent'],inplace=True)
```

```
df.head(5)
```

quantity	discount	sale_price	profit
2	5.2	254.8	14.8
3	21.9	708.1	108.1

Part 3: Load

Description:

System Requirements:

- SQL Server Express
- SQL Server management Studio (SSMS)

Load Process for this project:

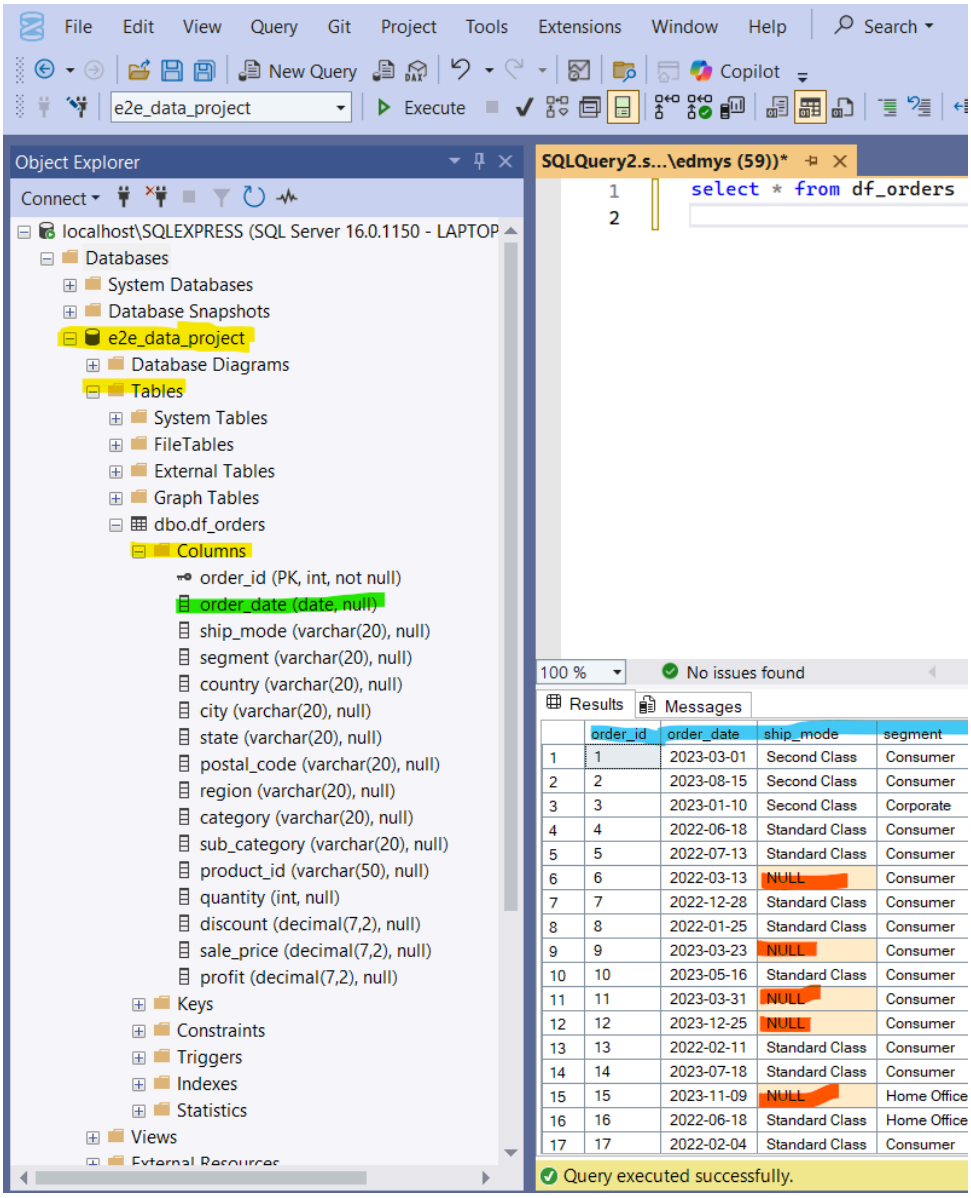
Set up column data types in SSMS to minimize storage volume.

Upload data into SSMS database using “append” to …

```
[1]:
import sqlalchemy as sal
from sqlalchemy import text

server    = r"localhost\SQLEXPRESS"
driver    = "ODBC Driver 17 for SQL Server"
target_db = "e2e_data_project"

# 1) Connect to master with AUTOCOMMIT so CREATE DATABASE is allowed
engine_master = sal.create_engine(
    f"mssql+pyodbc://@{server}/master?driver={driver.replace(' ', '+')}&trusted_connection=yes",
    isolation_level="AUTOCOMMIT",  # <-- key: no transaction
)

with engine_master.connect() as conn:
    # Check existence first (single statement)
    exists = conn.execute(text("SELECT DB_ID(:name)"), {"name": target_db}).scalar()
    if exists is None:
        # CREATE DATABASE must be its own statement with autocommit
        conn.exec_driver_sql(f"CREATE DATABASE [{target_db}]")

# 2) Connect to the target DB (normal transactional behavior)
engine = sal.create_engine(
    f"mssql+pyodbc://@{server}/{target_db}?driver={driver.replace(' ', '+')}&trusted_connection=yes"
)

# 3) Load your dataframe
with engine.begin() as conn:
    df.to_sql("df_orders", con=conn, index=False, if_exists="append")

print("✅ Database ensured and data uploaded.")
```

```
✅ Database ensured and data uploaded.
```

Verify data loaded and data constraints are correct.


File Edit View Query Git Project Tools Extensions Window Help Search
New Query Copilot
e2e_data_project Execute
Object Explorer
Connect
localhost\SQLEXPRESS (SQL Server 16.0.1150 - LAPTOP
Databases
System Databases
Database Snapshots
e2e_data_project
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.df_orders
Columns
order_id (PK, int, not null)
order_date (date, null)
ship_mode (varchar(20), null)
segment (varchar(20), null)
country (varchar(20), null)
city (varchar(20), null)
state (varchar(20), null)
postal_code (varchar(20), null)
region (varchar(20), null)
category (varchar(20), null)
sub_category (varchar(20), null)
product_id (varchar(50), null)
quantity (int, null)
discount (decimal(7,2), null)
sale_price (decimal(7,2), null)
profit (decimal(7,2), null)
Keys
Constraints
Triggers
Indexes
Statistics
Views
SQLQuery2.s...\edmys (59))*
1 select * from df_orders
2
100 %
No issues found
Results
Messages
order_id order_date ship_mode segment
1 1 2023-03-01 Second Class Consumer
2 2 2023-08-15 Second Class Consumer
3 3 2023-01-10 Second Class Corporate
4 4 2022-06-18 Standard Class Consumer
5 5 2022-07-13 Standard Class Consumer
6 6 2022-03-13 NULL Consumer
7 7 2022-12-28 Standard Class Consumer
8 8 2022-01-25 Standard Class Consumer
9 9 2023-03-23 NULL Consumer
10 10 2023-05-16 Standard Class Consumer
11 11 2023-03-31 NULL Consumer
12 12 2023-12-25 NULL Consumer
13 13 2022-02-11 Standard Class Consumer
14 14 2023-07-18 Standard Class Consumer
15 15 2023-11-09 NULL Home Office
16 16 2022-06-18 Standard Class Home Office
17 17 2022-02-04 Standard Class Consumer
Query executed successfully.