

Project Overview

Objective:

Create an end-to-end analytics pipeline for an ecommerce platform to track sales, customers, products, and performance metrics.

Scope:

- Generate synthetic data for orders, products, customers, payments, shipping, and categories (>1 million records).
- Store data in Amazon S3.
- Extract data into Snowflake using Airbyte.
- Transform and model data using dbt.
- Perform analytics in Snowflake.
- Connect Snowflake marts to Power BI for professional dashboards.

Tools & Technologies:

Layer	Tool/Technology
Data Generation	Python
Data Storage	Amazon S3
Data Integration	Airbyte
Data Warehouse	Snowflake
Data Modeling & Transform	dbt
BI & Visualization	Power BI
Programming / Scripting	Python / SQL / DAX

1- Data Generation

The screenshot shows a Jupyter Notebook interface with two code cells. The left cell contains Python code for initializing an S3 client and defining a function to upload a DataFrame to S3 as a Parquet file. The right cell contains Python code for generating synthetic datasets for regions, categories, products, customers, and orders.

```
17 # Initialize S3 client
18 s3 = boto3.client([
19     "s3",
20     aws_access_key_id=AWS_ACCESS_KEY_ID,
21     aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
22     region_name=AWS_REGION
23 ])
24
25
26 # -----
27 # DATASET SIZES
28 #
29 n_customers = 100_000
30 n_orders = 200_000
31 n_products = 5_000
32 n_regions = 20
33 n_categories = 50
34
35
36 def upload_df_to_s3(df, filename):
37     """
38         Convert DataFrame → Parquet → Upload to S3 (in-memory)
39     """
40     table = pa.Table.from_pandas(df)
41
42     buffer = BytesIO()
43     pq.write_table(table, buffer)
44     buffer.seek(0)
45
46     s3_path = f"{S3_PREFIX}{filename}.parquet"
47
48     s3.upload_fileobj(buffer, BUCKET_NAME, s3_path)
49
50     print(f"✓ Uploaded {filename}.parquet to s3://{BUCKET_NAME}/{s3_path}")
51
52
53
54 # GENERATE SYNTHETIC TABLES
55 #
56 regions = pd.DataFrame({
57     "region_id": range(1, n_regions + 1),
58     "region_name": [f"Region_{i}" for i in range(1, n_regions + 1)]
59 })
60
61 categories = pd.DataFrame({
62     "category_id": range(1, n_categories + 1),
63     "category_name": [f"Category_{i}" for i in range(1, n_categories + 1)]
64 })
65
66 products = pd.DataFrame({
67     "product_id": range(1, n_products + 1),
68     "product_name": [f"Product_{i}" for i in range(1, n_products + 1)],
69     "category_id": np.random.randint(1, n_categories + 1, n_products),
70     "price": np.round(np.random.uniform(5, 2000, n_products), 2)
71 })
72
73 customers = pd.DataFrame({
74     "customer_id": range(1, n_customers + 1),
75     "first_name": [f"Name{i}" for i in range(1, n_customers + 1)],
76     "last_name": [f"Last{i}" for i in range(1, n_customers + 1)],
77     "email": [f"user{i}@example.com" for i in range(1, n_customers + 1)],
78     "region_id": np.random.randint(1, n_regions + 1, n_customers),
79     "created_at": pd.to_datetime("2023-01-01") +
80                 pd.to_timedelta(np.random.randint(0, 400, n_customers), "D")
81 })
82
83 orders = pd.DataFrame({
84     "order_id": range(1, n_orders + 1),
85     "customer_id": np.random.randint(1, n_customers + 1, n_orders),
86     "order_date": pd.to_datetime("2023-01-01") +
87                 pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
88     "status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders),
89     "total_amount": np.round(np.random.uniform(10, 3000, n_orders), 2)
90 })
```

Activate Windows
Go to Settings to

```
Sales Snowflake > ⚡ datagen.py > ...
80     pd.to_timedelta(np.random.randint(0, 400, n_customers), "D")
81 }
82
83 orders = pd.DataFrame({
84     "order_id": range(1, n_orders + 1),
85     "customer_id": np.random.randint(1, n_customers + 1, n_orders),
86     "order_date": pd.to_datetime("2023-01-01") +
87         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
88     "status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders),
89     "total_amount": np.round(np.random.uniform(10, 3000, n_orders), 2)
90 })
91
92 order_items = pd.DataFrame({
93     "order_item_id": range(1, n_orders * 2 + 1),
94     "order_id": np.random.randint(1, n_orders + 1, n_orders * 2),
95     "product_id": np.random.randint(1, n_products + 1, n_orders * 2),
96     "quantity": np.random.randint(1, 5, n_orders * 2)
97 })
98
99 payments = pd.DataFrame({
100     "payment_id": range(1, n_orders + 1),
101     "order_id": range(1, n_orders + 1),
102     "payment_method": np.random.choice(["Card", "PayPal", "Cash"], n_orders),
103     "amount": np.round(np.random.uniform(10, 3000, n_orders), 2),
104     "payment_date": pd.to_datetime("2023-01-01") +
105         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D")
106 })
107
108 shipping = pd.DataFrame({
109     "shipping_id": range(1, n_orders + 1),
110     "order_id": range(1, n_orders + 1),
111     "shipped_date": pd.to_datetime("2023-01-01") +
112         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
113     "delivery_date": pd.to_datetime("2023-01-01") +
114         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
115     "shipping_status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders)
116 })
117
118 # UPLOAD TABLES TO S3
119 # -----
120
121 upload_df_to_s3(regions, "raw_regions")
122 upload_df_to_s3(categories, "raw_categories")
123 upload_df_to_s3(products, "raw_products")
124 upload_df_to_s3(customers, "raw_customers")
125 upload_df_to_s3(orders, "raw_orders")
126 upload_df_to_s3(order_items, "raw_order_items")
127 upload_df_to_s3(payments, "raw_payments")
128 upload_df_to_s3(shipping, "raw_shipping")
129
130 print("\n⚡ All Parquet files uploaded to S3 successfully!")
131
```

Tables Generated:

- **Orders** (order_id, customer_id, order_date, total_amount, status)
- **Order Items** (order_item_id, order_id, product_id, quantity, price)
- **Products** (product_id, product_name, category_id, price)
- **Customers** (customer_id, first_name, last_name, email, region_id)
- **Categories** (category_id, category_name)
- **Payments** (payment_id, order_id, amount, payment_date, payment_method)
- **Shipping** (shipping_id, order_id, shipped_date, delivery_date, shipping_status)
- **Regions** (region_id, region_name)

2- Data Storage

The screenshot shows the AWS IAM User Details page for a user named 'python_sends_s3'. The left sidebar shows the navigation path: IAM > Users > python_sends_s3. The main content area displays the user's summary information, including ARN, console access status, creation date, and two access keys. The 'Permissions' tab is selected, showing one attached policy named 'PythonGenS3'. The policy details are listed as Customer managed and attached directly.

Summary
ARN: arn:aws:iam::767163357664:user/python_sends_s3
Console access: Disabled
Created: December 09, 2025, 15:53 (UTC+02:00)
Last console sign-in: -
Access key 1: AKIA3FHT7RPAIXPNOJM3 - Active Used today. Created today.
Access key 2: Create access key

Permissions policies (1)
Policy name: PythonGenS3
Type: Customer managed
Attached via: Directly

Screenshot of the AWS S3 console showing the contents of the 'sales-bucket-v4' bucket. The left sidebar shows navigation options like 'Buckets', 'Access management and security', and 'Storage management and insights'. The main area displays 8 objects, all of which are folders:

Name	Type	Last modified	Size	Storage class
categorie/_	Folder	-	-	-
customers/_	Folder	-	-	-
order_items/_	Folder	-	-	-
orders/_	Folder	-	-	-
payments/_	Folder	-	-	-
products/_	Folder	-	-	-
regions/_	Folder	-	-	-
shipping/_	Folder	-	-	-

Screenshot of the AWS S3 console showing the contents of the 'raw_data/' folder. It contains a single object:

Name	Type	Last modified	Size	Storage class
raw_order_items.parquet	parquet	December 9, 2025, 16:21:34 (UTC+02:00)	4.1 MB	Standard

s3://ecommerce-data/

- └─ orders/
- └─ order_items/
- └─ customers/
- └─ products/
- └─ categories/
- └─ payments/
- └─ shipping/
- └─ regions/

Format: Parquet

3- Data Extraction & Loading

Tool: Airbyte

The screenshot displays two main sections of the AWS IAM console.

User Details: The top section shows the user "Airbyte-read-s3" with the ARN `arn:aws:iam::767163337664:user/Airbyte-read-s3`. It indicates "Console access Disabled" and "Last console sign-in -". Two access keys are listed: "Access key 1" (Active, Never used, Created today) and "Access key 2" (Create access key).

Permissions Policies: The bottom section shows the policy "AirbyteS3Read" attached to the user. The policy document is displayed as:

```
1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "s3:listBucket",
8                 "s3:getObject",
9                 "s3:GetBucketLocation"
10            ],
11            "Resource": [
12                "arn:aws:s3:::sales-bucket-v4",
13                "arn:aws:s3:::sales-bucket-v4/*"
14            ]
15        }
16    ]
17 }
```

Details for the policy include: Type Customer managed, Creation time December 09, 2025, 16:33 (UTC+02:00), Edited time December 09, 2025, 16:33 (UTC+02:00), and ARN `arn:aws:iam::767163337664:policy/AirbyteS3Read`.

 **S3-sales-parquet1**
S3 v4.15.2 Fork in Builder

[Settings](#) [Connections](#)

Source Settings

Source name ⓘ
S3-sales-parquet1

Bucket ⓘ

sales-bucket-v4

The list of streams to sync ⓘ

- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/categorie/raw_categories.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/customers/raw_customers.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/order_items/raw_order_items.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/orders/raw_orders.parquet"]... X

[Add](#)

The list of streams to sync ⓘ

- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/categorie/raw_categories.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/customers/raw_customers.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/order_items/raw_order_items.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/orders/raw_orders.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/payments/raw_payments.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/products/raw_products.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/regions/raw_regions.parquet"]... X
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/shipping/raw_shipping.parquet"]... X

[Add](#)

Activate Window
Go to Settings to activate

▼ Optional fields

Start Date ⓘ *Optional* YYYY-MM-DDTHH:mm:ss.SSSSSZ

Edit

AWS Access Key ID ⓘ *Optional*

*****Edit

AWS Secret Access Key ⓘ *Optional*

*****Edit

Endpoint ⓘ *Optional*

AWS Region ⓘ *Optional*

us-east-1

From s3 load to snowflake

Snowflake Snowflake 1 connection 1 hour ago 1

 **Snowflake**
Snowflake v4.0.31

Settings Connections

Destination Settings

Destination name ⓘ
Snowflake

Connection

Host ⓘ {account_name}.snowflakecomputing.com or {accountname}.{aws_location}.aws.snowflakecomputing.com
ccykbwk-yc47376.snowflakecomputing.com

Role ⓘ ACCOUNTADMIN

Warehouse ⓘ ecommerce_wh

Activate Windows
Go to Settings to activate

Warehouse ⓘ
ecommerce_wh

Database ⓘ
ecommerce_db

Default Schema ⓘ
raw

Username ⓘ
dbt_user

Authorization Method ⓘ Username and Password ▾

Password ⓘ ***** Edit

Active Streams

Status	Stream name	Latest sync in	Data fresh as of	⋮
Synced	categorie	50 loaded	27 minutes ago	⋮
Synced	Customers	100,000 loaded	27 minutes ago	⋮
Synced	order_items	400,000 loaded	27 minutes ago	⋮
Synced	orders	200,000 loaded	27 minutes ago	⋮
Synced	payment	200,000 loaded	27 minutes ago	⋮
Synced	products	5,000 loaded	27 minutes ago	⋮
Synced	regions	20 loaded	27 minutes ago	⋮

Activate Windows
27 minutes ago Go to Settings to activate Window

The screenshot shows the Snowflake web interface. On the left, the sidebar includes sections for Ingestion, Transformation, AI & ML, Monitoring, Marketplace, and Catalog. It also displays credit information (\$390 credits left) and a trial end date (6 days). A user profile (Nada Mahmoud...) is shown at the bottom.

The main workspace contains two tabs: "E-commerce data.sql" and "test.sql". The "E-commerce data.sql" tab shows a query for raw customers data with a limit of 200 rows. The results pane displays a bar chart with the following data:

	# ORDER_ID	# QUANTITY	# PRODUCT_ID	# ORDER_ITEM_ID
1	148	1	88	1
2	198630	4	4993	50
3	135344	1	237	1
4	15831	1	4777	2
5	36143	4	1330	3
6	174564	3	4427	4

4- Data Transformation & Modeling

Tool: dbt (Data Build Tool)

Purpose: Transform RAW data into **analytics-ready marts**.

```
PS F:\Sales Snowflake> dbt init ecommerce_project
22:45:38  Running with dbt=1.10.13
22:45:38
Your new dbt project "ecommerce_project" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

  https://docs.getdbt.com/docs/configure-your-profile

One more thing:
Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:
  Activate Windows
  Go to Settings to activate a new window
```

```
PS F:\Sales Snowflake> dbt init ecommerce_project
22:45:38  Setting up your profile.
The profile ecommerce_project already exists in C:\Users\DeLL\.dbt\profiles.yml. Continue and overwrite it? [y/N]: y
Which database would you like to use?
[1] postgres
[2] snowflake

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 2
account (https://<this\_value>.snowflakecomputing.com): ccykbwk-yc47376
[1] password
[2] keypair
[3] sso
  Activate Windows
  Go to Settings to activate a new window
```


Transform data

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_categories.sql
 1  with raw as (
 2    select
 3      category_id,
 4      category_name
 5    from {{ source('ecommerce_raw', 'RAWCATEGORIES') }}}
 6  )
 7  select * from raw
 8
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_customers.sql
 1  with raw as (
 2    select
 3      customer_id,
 4      first_name,
 5      last_name,
 6      email,
 7      region_id
 8    from {{ source('ecommerce_raw', 'RAWCUSTOMERS_REAL') }}}
 9  )
10  select * from raw
11
12
13
14
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_order_items.sql
 1  with raw as (
 2    select
 3      order_item_id,
 4      order_id,
 5      product_id,
 6      quantity::integer as quantity
 7    from {{ source('ecommerce_raw', 'ORDER_ITEMS') }}}
 8  )
 9  select * from raw
10
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_orders.sql
1  -- models/staging/stg_orders.sql
2  with raw as (
3      select
4          order_id,
5          customer_id,
6          total_amount::decimal as total_amount,
7          order_date::timestamp as order_date,
8          status
9      from {{ source('ecommerce_raw', 'ORDERS') }}
10 )
11 select * from raw
12
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_payments.sql
1  with raw as (
2      select
3          payment_id,
4          order_id,
5          amount::decimal as amount,
6          payment_date::timestamp as payment_date,
7          payment_method
8      from {{ source('ecommerce_raw', 'PAYMENT') }}
9  )
10 select * from raw
11
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_products.sql
1  with raw as (
2      select
3          product_id,
4          product_name,
5          category_id,
6          price::decimal as price
7      from {{ source('ecommerce_raw', 'PRODUCT_REAL') }}
8  )
9 select * from raw
10
```

```

Sales Snowflake > ecommerce_project > models > models > staging > stg_regions.sql
1   with raw as (
2     select
3       region_id,
4       region_name
5     from {{ source('ecommerce_raw', 'REGION_REAL') }}}
6   )
7   select * from raw
8

Sales Snowflake > ecommerce_project > models > models > staging > stg_shipping.sql
1   with raw as (
2     select
3       shipping_id,
4       order_id,
5       shipped_date::timestamp as shipped_date,
6       delivery_date::timestamp as delivery_date,
7       shipping_status
8     from {{ source('ecommerce_raw', 'SHIPPING') }}}
9   )
10  select * from raw
11

```

Creating marts facts and dimensions

The screenshot shows the Snowflake UI interface. On the left, there is a tree view of the workspace structure:

- UNTITLED (WORKSPACE)
- DES & BLOWFISH
- Sales Snowflake
 - ecommerce_project
 - analyses
 - .gitkeep
 - logs
 - macros
 - .gitkeep
 - models
 - marts
 - marts
 - dim_categories.sql
 - dim_customers.sql
 - dim_products.sql
 - dim_regions.sql
 - fact_order_items.sql
 - fact_orders.sql

On the right, a preview window displays the content of the `dim_categories.sql` file:

```

Sales Snowflake > ecommerce_project > models > models > marts > marts > dim_categories.sql
1   select
2     category_id,
3     category_name
4   from {{ ref('stg_categories') }}
5

```

The status bar at the bottom indicates "File Saved" and "9 mins".

The image shows three separate code editors side-by-side, all displaying SQL queries for dimension tables in a Snowflake workspace named "ecommerce_project".

Top Editor (dim_customers.sql):

```
1 select
2     customer_id,
3     email,
4     region_id,
5     first_name || ' ' || last_name as customer_name
6 from {{ ref('stg_customers') }}
```

Middle Editor (dim_products.sql):

```
1 select
2     product_id,
3     product_name,
4     category_id,
5     price
6 from {{ ref('stg_products') }}
```

Bottom Editor (dim_regions.sql):

```
1 select
2     region_id,
3     region_name
4 from {{ ref('stg_regions') }}
```

Each editor has a sidebar with project and schema navigation, and a timeline at the bottom indicating the file was saved 11 minutes ago.

The image shows two side-by-side screenshots of a Snowflake SQL editor interface. Both windows have a dark theme.

Top Window (fact_order_items.sql):

```
1 select
2     oi.order_item_id,
3     oi.order_id,
4     oi.product_id,
5     oi.quantity,
6     oi.quantity * pr.price as total_price
7     from {{ ref('stg_order_items') }} oi
8     left join {{ ref('stg_products') }} pr
9     on oi.product_id = pr.product_id
10
```

Bottom Window (fact_orders.sql):

```
1 select
2     o.order_id,
3     o.customer_id,
4     o.order_date,
5     o.total_amount,
6     o.status,
7     p.amount as payment_amount,
8     p.payment_method,
9     s.shipped_date,
10    s.delivery_date,
11    s.shipping_status
12    from {{ ref('stg_orders') }} o
13    left join {{ ref('stg_payment') }} p
14    on o.order_id = p.order_id
15    left join {{ ref('stg_shipping') }} s
16    on o.order_id = s.order_id
17
```

Both windows show the file outline on the left and a timeline at the bottom. The top window has a 9 mins save time, and the bottom window has a 10 mins save time.

The screenshot shows the Snowflake interface with the following components:

- My Workspace** sidebar (left):
 - Icon: snowflake
 - Home icon
 - Add new icon (+)
 - Work with data
 - Projects** (selected)
 - Ingestion
 - Transformation
 - AI & ML
 - Monitoring
 - Marketplace
 - Horizon Catalog
 - Catalog
- Database Explorer** (right):
 - Icon: Database
 - My Workspace dropdown
 - Search for files input field
 - Add new button
 - E-commerce data.sql (selected)
 - test.sql
 - Worksheets (sub-section)
 - Search worksheets input field
 - Owned
 - Shared with you
 - Database Explorer Closes X icon
 - Objects tab (selected)
 - Data Products tab
 - Search input field
 - Filter icon
 - DIM_PRODUCTS (selected)
 - DIM_CATEGORIES
 - DIM_CUSTOMERS
 - DIM_REGIONS
 - FACT_ORDERS
 - FACT_ORDER_ITEMS

5- Analytics in Snowflake

The screenshot shows the Snowflake UI interface. On the left, there's a sidebar with 'Worksheets' (containing 'E-commerce data.sql' and 'test.sql'), 'Database Explorer' (listing objects like FACT_ORDERS, FACT_ORDER_ITEMS, etc.), and a 'Filter' section. The main area has tabs for 'E-commerce data.sql' and 'test.sql'. A 'Results (just now)' panel is open under 'E-commerce data.sql', showing the execution of SQL code to create two views:

```
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219 CREATE OR REPLACE VIEW SALES_MONTHLY_ORDERS AS  
SELECT  
    DATE_TRUNC('month', order_date) AS Month,  
    COUNT(order_id) AS OrderCount  
FROM Orders  
GROUP BY DATE_TRUNC('month', order_date)  
ORDER BY Month;  
  
CREATE OR REPLACE VIEW SALES_MONTHLY_REVENUE AS  
SELECT  
    DATE_TRUNC('month', ORDER_DATE) AS MONTH,  
    SUM(TOTAL_AMOUNT) AS TOTAL_REVENUE  
FROM ECOMMERCE_DB.RAW.ORDERS  
GROUP BY 1  
ORDER BY 1;
```

The results show 'View SALES_MONTHLY_ORDERS successfully created.' and 'View SALES_MONTHLY_REVENUE successfully created.' respectively. The UI includes standard navigation buttons (Home, Back, Forward, Refresh), a 'Share' button, and a 'Feedback' link.

Home E-commerce data.sql test.sql +

My Workspace > E-commerce data.sql

CREATE OR REPLACE VIEW SALES_TOP_PRODUCTS AS
SELECT
 p.PRODUCT_NAME AS PRODUCT_NAME,
 SUM(oi.TOTAL_PRICE) AS REVENUE
FROM ECOMMERCE_DB.RAW.FACT_ORDER_ITEMS oi
JOIN ECOMMERCE_DB.RAW.DIM_PRODUCTS p
 ON oi.PRODUCT_ID = p.PRODUCT_ID
GROUP BY p.PRODUCT_NAME
ORDER BY REVENUE DESC
LIMIT 5;
| Ctrl+I to generate

Results (just now)

Table Chart

1 View SALES_TOP_PRODUCTS successfully created.

Activate Windows

Home E-commerce data.sql test.sql +

My Workspace > E-commerce data.sql

CREATE OR REPLACE VIEW SALES_REVENUE_BY_CATEGORY AS
SELECT
 c.CATEGORY_NAME,
 SUM(oi.QUANTITY * p.PRICE) AS REVENUE
FROM ECOMMERCE_DB.RAW.ORDER_ITEMS oi
JOIN ECOMMERCE_DB.RAW.PRODUCT_REAL p
 ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN ECOMMERCE_DB.RAW.RAWCATEGORIES c
 ON p.CATEGORY_ID = c.CATEGORY_ID
GROUP BY c.CATEGORY_NAME
ORDER BY REVENUE DESC;

Results (just now)

Table Chart

1 View SALES_REVENUE_BY_CATEGORY successfully created.

My Workspace > E-commerce data.sql

```

269 CREATE OR REPLACE VIEW SALES_REVENUE_BY_REGION AS
270   SELECT
271     r.REGION_NAME,
272     SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENU
273   FROM ECOMMERCE_DB.RAW.ORDERS o
274   JOIN ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
275     ON o.CUSTOMER_ID = c.CUSTOMER_ID
276   JOIN ECOMMERCE_DB.RAW.REGION_REAL r
277     ON c.REGION_ID = r.REGION_ID
278   GROUP BY r.REGION_NAME

```

Results (just now)

	status
1	View SALES_REVENUE_BY_REGION successfully created.

Activate Windows
Go to Settings to activate Window

Database Explorer

- Objects
- Data Products

Search

Filter

- DIM_CATEGORIES
- DIM_CUSTOMERS
- DIM_PRODUCTS**
- DIM_REGIONS
- FACT_ORDERS
- FACT_ORDER_ITEMS
- MY_SECOND_DBT_M...
- SALES_MONTHLY_O...

Home > E-commerce data.sql

```

278   GROUP BY r.REGION_NAME
279   ORDER BY TOTAL_REVENU DESC;
280
281 CREATE OR REPLACE VIEW CUSTOMER_NEW_RETURNING_OVER_TIME AS
282   WITH customer_orders AS (
283     SELECT
284       CUSTOMER_ID,
285       MIN(ORDER_DATE) AS FIRST_ORDER_DATE,
286       COUNT(ORDER_ID) AS TOTAL_ORDERS
287     FROM ECOMMERCE_DB.RAW.ORDERS
288     GROUP BY CUSTOMER_ID
289   )
290   SELECT
291     DATE_TRUNC('month', o.ORDER_DATE) AS MONTH,
292     SUM(CASE WHEN co.TOTAL_ORDERS = 1 THEN 1 ELSE 0 END) AS NEW_CUSTOMERS,
293     SUM(CASE WHEN co.TOTAL_ORDERS > 1 THEN 1 ELSE 0 END) AS RETURNING_CUSTOMERS
294   FROM ECOMMERCE_DB.RAW.ORDERS o
295   JOIN customer_orders co
296     ON o.CUSTOMER_ID = co.CUSTOMER_ID
297   GROUP BY 1
298   ORDER BY 1;
299

```

Results (just now)

	status
1	View CUSTOMER_NEW_RETURNING_OVER_TIME successfully created.

Activate Windows
Go to Settings to activate Window

My Workspace > E-commerce data.sql

```

270
299
300
301 CREATE OR REPLACE VIEW CUSTOMER_ORDERS_DISTRIBUTION AS
302 SELECT
303     COUNT(ORDER_ID) AS ORDERS_PER_CUSTOMER,
304     COUNT(CUSTOMER_ID) AS CUSTOMER_COUNT
305 FROM ECOMMERCE_DB.RAW.ORDERS
306 GROUP BY CUSTOMER_ID
307 ORDER BY ORDERS_PER_CUSTOMER;
308
309     Ctrl+I to generate
310
311
312
313

```

Results (just now)

	A status
1	View CUSTOMER_ORDERS_DISTRIBUTION successfully created.

Home E-commerce data.sql test.sql +

My Workspace > E-commerce data.sql

```

309
310 CREATE OR REPLACE VIEW TOP_CUSTOMERS_BY_REVENUE AS
311 SELECT
312     c.CUSTOMER_ID,
313     c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME,
314     SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE,
315     COUNT(o.ORDER_ID) AS TOTAL_ORDERS
316 FROM ECOMMERCE_DB.RAW.ORDERS o
317 JOIN ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
318     ON o.CUSTOMER_ID = c.CUSTOMER_ID
319 GROUP BY c.CUSTOMER_ID, CUSTOMER_NAME
320 ORDER BY TOTAL_REVENUE DESC
321 LIMIT 10;
322
323     | Ctrl+I to generate

```

Results (just now)

	A status
1	View TOP_CUSTOMERS_BY_REVENUE successfully created.

Home SQL E-commerce data.sql SQL test.sql +

My Workspace > E-commerce data.sql

▶ ▼ ✖ Share ...

```

321
322
323 CREATE OR REPLACE VIEW CUSTOMERS_BY_REGION AS
324     SELECT
325         r.REGION_NAME,
326         COUNT(DISTINCT c.CUSTOMER_ID) AS CUSTOMER_COUNT,
327         SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE
328     FROM ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
329     LEFT JOIN ECOMMERCE_DB.RAW.ORDERS o
330         ON c.CUSTOMER_ID = o.CUSTOMER_ID
331     LEFT JOIN ECOMMERCE_DB.RAW.REGION_REAL r
332         ON c.REGION_ID = r.REGION_ID
333     GROUP BY r.REGION_NAME
334     ORDER BY TOTAL_REVENUE DESC;
335

```

Results (just now)

Table	Chart

1 View CUSTOMERS_BY_REGION successfully created.


```

336
337 CREATE OR REPLACE VIEW CUSTOMER_SEGMENTS AS
338     WITH customer_revenue AS (
339         SELECT
340             c.CUSTOMER_ID,
341             c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME,
342             SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE
343         FROM ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
344         LEFT JOIN ECOMMERCE_DB.RAW.ORDERS o
345             ON c.CUSTOMER_ID = o.CUSTOMER_ID
346         GROUP BY c.CUSTOMER_ID, CUSTOMER_NAME
347     ),
348     quartiles AS (
349         SELECT
350             PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q1,
351             PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q2,
352             PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q3
353         FROM customer_revenue
354     )
355     SELECT

```

Results (just now)

Table	Chart

1 View CUSTOMER_SEGMENTS successfully created.

Activate Windows
Go to Settings to activate Window

```

349     SELECT
350         PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q1,
351         PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q2,
352         PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q3
353     FROM customer_revenue
354 )
355     SELECT
356         cr.CUSTOMER_ID,
357         cr.CUSTOMER_NAME,
358         cr.TOTAL_REVENUE,
359         CASE
360             WHEN cr.TOTAL_REVENUE >= q.Q3 THEN 'High'
361             WHEN cr.TOTAL_REVENUE >= q.Q2 THEN 'Medium'
362             ELSE 'Low'
363         END AS REVENUE_SEGMENT
364     FROM customer_revenue cr
365     CROSS JOIN quartiles q
366     ORDER BY cr.TOTAL_REVENUE DESC;
367
368

```

6- Connect to Power BI

The screenshot shows the Power BI Desktop interface. The top navigation bar includes File, Home, Insert, Modeling, View, Optimize, Help, and External tools. The Home tab is selected. On the left, the Navigator pane displays a tree view of the data model, including tables like airbyte_internal, INFORMATION_SCHEMA, PUBLIC, and RAW [27], along with dimensions such as DIM_CATEGORIES, DIM_CUSTOMERS, DIM_PRODUCTS, DIM_REGIONS, FACT_ORDER_ITEMS, and FACT_ORDERS. A preview grid for the FACT_ORDERS table is shown in the center, containing columns: ORDER_ID, CUSTOMER_ID, ORDER_DATE, TOTAL_AMOUNT, and STATUS. The grid shows 23 rows of data. To the right, there are various Power BI tools: Sensitivity, Publish, Prep data for Copilot, AI, Copilot, Build, Filters, Suggestions, and Data. At the bottom, there are buttons for Load, Transform Data, and Cancel.

ORDER_ID	CUSTOMER_ID	ORDER_DATE	TOTAL_AMOUNT	STATUS
1	55885	11/4/2023 12:00:00 AM	1031	Delivered
2	55330	12/1/2023 12:00:00 AM	478	Delivered
3	94523	11/7/2023 12:00:00 AM	779	Shipped
4	43229	9/30/2023 12:00:00 AM	639	Pending
5	26869	10/2/2023 12:00:00 AM	2243	Shipped
6	61057	11/21/2023 12:00:00 AM	1308	Shipped
7	99964	11/1/2023 12:00:00 AM	1447	Shipped
8	20344	8/8/2023 12:00:00 AM	2813	Shipped
9	12791	2/11/2023 12:00:00 AM	1765	Delivered
10	45412	11/23/2023 12:00:00 AM	2661	Delivered
11	13780	12/14/2023 12:00:00 AM	1167	Shipped
12	56460	5/26/2023 12:00:00 AM	871	Delivered
13	52400	11/21/2023 12:00:00 AM	649	Delivered
14	76053	8/12/2023 12:00:00 AM	1254	Pending
15	93064	3/7/2023 12:00:00 AM	222	Shipped
16	30596	3/30/2023 12:00:00 AM	2072	Delivered
17	64261	8/10/2023 12:00:00 AM	798	Delivered
18	95325	2/23/2023 12:00:00 AM	2946	Pending
19	24514	10/3/2023 12:00:00 AM	2076	Shipped
20	6441	12/1/2023 12:00:00 AM	441	Pending
21	82406	1/2/2023 12:00:00 AM	1620	Pending
22	59648	4/19/2023 12:00:00 AM	1750	Shipped
23	45333	3/28/2023 12:00:00 AM	2081	Delivered

- >  SALES_MONTHLY_OR...
- >  SALES_MONTHLY_... ...
- >  SALES_REVENUE_BY_C...
- >  SALES_REVENUE_BY_R...
- >  SALES_TOP_PRODUCTS

- >  CUSTOMER_NEW_RETURNING_OVER_TIME
- >  CUSTOMER_ORDERS_DISTRIBUTION
- >  CUSTOMER_SEGMENTS
- >  CUSTOMERS_BY_REGION