

Subscription-Based Financial Advisory ELT Pipeline and Product Analysis

Oamen Modupe
oamenmodupe@gmail.com

1. Project Overview

1.1 Introduction

This project analyzes data from a subscription-based digital product offering financial advisory services. Customers can choose between annual and digital subscriptions, and the product provides daytime customer support to address any product-related inquiries or issues, including subscription sign-ups and cancellations.

Data Source: [Source](#)

1.2 Objectives

The primary objectives of this project are:

1. **Subscription and Customer Care Analysis:** Analyze data related to customer subscriptions, sign-ups, cancellations, and customer care cases to gain insights into service usage and support demands.
2. **ELT Pipeline Implementation:** Develop an ELT pipeline that extracts data from Snowflake, transforms it using dbt, orchestrates daily operations with Apache Airflow, and visualizes the results on a dashboard.
3. **Customer Behavior Analysis:** Examine historical trends in customer sign-ups and cancellations to identify key factors influencing customer retention and churn, helping to inform strategies for improving customer loyalty.

1.3 Dataset Description

The dataset contains the following key information:

1. **Customer Sign-up and Cancellation Dates:** Records of sign-ups and cancellations at the product level, including timestamps for when a customer signs up or cancels their subscription.
2. **Call Center Activity:** Data on customer interactions with the support team, including inquiries and issues raised by customers.
3. **Customer Demographics:** Information about the customers' backgrounds, helping to segment and analyze customer behavior.
4. **Product Pricing Information:** Details on the pricing of the subscription products offered, providing insights into revenue generation.

1.4 Dataset Files

- **customer_product.csv:** This file captures product sign-ups and cancellations by customers, with the following columns:
 - **customer_id:** Unique identifier for each customer.

- **product**: Product ID linked to the product information table.
- **signup_date_time**: Timestamp of when the customer signed up.
- **cancel_date_time**: Timestamp recorded when a customer cancels their subscription.

2. Build ELT Pipeline with Snowflake, Apache Airflow, dbt, Git/Github

2.0 Install packages

Install dbt-core, dbt-snowflake adapter, astro to run apache airflow, astro-cosmos

```
pip install dbt-core dbt-snowflake astro astro-cosmos
```

2.1 Create Snowflake Account

If you don't already have an account, create a free trial account on [Snowflake](#)

2.1 Create data warehouse, database, and schema(s)

- Use the accountadmin role to create the data role, warehouse, database, and schema(s)
use role accountadmin;

```
-- create databases and schemas
create database customer_subscription; -- main database
create schema customer_subscription.raw_data; -- schema with main raw data
create schema customer_subscription.dbt_outputs; -- schema for materialized
dbt outputs
create role oamen_dbt_role;
```

- Grant permissions to the newly created role

```
grant usage on warehouse dbt_wh to role oamen_dbt_role;
grant usage on database customer_subscription to role oamen_dbt_role;
grant usage on schema customer_subscription.raw_data to role
oamen_dbt_role;
grant usage on schema customer_subscription.dbt_outputs to role
oamen_dbt_role;
GRANT SELECT ON ALL TABLES IN SCHEMA customer_subscription.raw_data TO ROLE
oamen_dbt_role;
GRANT SELECT ON ALL TABLES IN SCHEMA customer_subscription.dbt_outputs TO
ROLE oamen_dbt_role;
GRANT CREATE TABLE ON SCHEMA customer_subscription.dbt_outputs TO ROLE
oamen_dbt_role;
```

```
GRANT MODIFY ON SCHEMA customer_subscription.dbt_outputs TO ROLE  
oamen_dbt_role;
```

- c. Grant my user the role and use the role

```
grant role oamen_dbt_role to user Oamen;  
  
use role oamen_dbt_role
```

2.3 Load CSV files into tables in customer_subscription.raw_data (Optional)

2.4 GitHub Repository

- a. Create a new repository in GitHub
- b. Rename the default main branch to master to match the local environment
- c. Initialize project directory: git init
- d. Add remote repo: git remote add origin <git-url>
- e. Pull master branch to the local environment: git pull origin master
- f. Create a new branch: git checkout -b dev

2.5 Setup DBT Project

- a. Setup dbt project: dbt init <project_name>
- b. Choose adapter: snowflake
- c. Account name(from snowflake login link,
<https://app.snowflake.com/><account_locator>/<account_name>:<account_locator>-<account_name>
- d. User: username to login Snowflake
- e. Password: Snowflake login password
- f. Role: role created in snowflake e.g oamen_dbt_role
- g. Warehouse: snowflake warehouse dbt_wh
- h. Database: snowflake database e.g customer_subscription
- i. Schema: dbt_outputs
- j. Threads: 10
- k. cd <dbt project directory>
- l. Create profiles.yml under dbt project directory and run dbt debug to test connection

```
cus_subscription_pipeline:  
  target: dev  
  outputs:  
    dev:  
      type: snowflake  
      account: <account_locator>-<account_name>
```

```

user: <snowflake username>
password: <snowflake password>
role: oamen_dbt_role
warehouse: dbt_wh
database: customer_subscription
schema: dbt_outputs
threads: 10

```

2.6 Delete example directory under models/

2.7 Configure dbt_project.yml

Create project directories as needed(staging and transformed)

- Staging will hold models that are 1-1 with the source data.
- Transformed will hold transformations/models

Configure dbt_project.yml models to set materialization output for each model directory

```

models:
  cus_subscription_pipeline:
    # Config indicated by + and applies to all files under models/example/
    staging:
      +materialized: view
      snowflake_warehouse: dbt_wh

    transformed:
      +materialized: table
      snowflake_warehouse: dbt_wh

```

2.8 Configure Sources.yml for the Staging models

The sources.yml file defines the configuration for source tables used in the dbt project. It specifies the metadata and quality tests for various raw data files, ensuring data integrity and consistency before transformation and analysis.

```

version: 2

sources:
  - name: raw_CUSTOMER_CASES
    description: This file contains all cases received from customers from
    various channels and reasons.
    database: CUSTOMER_SUBSCRIPTION # Ensure the correct case is used

```

```
schema: RAW_DATA
tables:
  - name: CUSTOMER_CASES
    columns:
      - name: CASE_ID
        tests:
          - unique
          - not_null
      - name: CUSTOMER_ID
        tests:
          - unique
          - not_null

  - name: raw_customer_products
    description: This file captures product sign-ups and cancellations by
customers.
    database: CUSTOMER_SUBSCRIPTION # Ensure the correct case is used
    schema: RAW_DATA
    tables:
      - name: CUSTOMER_PRODUCTS
        columns:
          - name: PRODUCT
            tests:
              - accepted_values:
                  values: ['prd_1', 'prd_2']
              - not_null

  - name: raw_CUSTOMER_INFO
    description: This file contains customer demographic information as of
the signup date.
    database: CUSTOMER_SUBSCRIPTION # Ensure the correct case is used
    schema: RAW_DATA
    tables:
      - name: CUSTOMER_INFO
        columns:
          - name: CUSTOMER_ID
            tests:
              - unique
              - not_null
```

2.9 Seeds

Seeds are files that can be loaded into the data warehouse. They are typically rarely changing CSV files.

To define a seed, simply add it to the seeds/ directory then import it with dbt seed.

In this case, the seed is project_info.csv. It can be referenced in other models with Jinja {{ ref('product_info') }}. Jinja is a text-based templating language.

Seeds can be configured to define data types, they can also be tested.

```
version: 2

seeds:
  - name: product_info
    description: Data on product types
    config:
      column_type:
        NAME: varchar(10)
    columns:
      - name: BILLING_CYCLE
        test:
          - not_null
      - name: NAME
        test:
          - not_null
      - name: PRICE
        test:
          - not_null
      - name: PRODUCT_ID
        test:
          - not_null
```

2.10 Tests

In dbt, a test is an assertion/validation of several dbt objects like models, sources, seeds, and snapshots.

It is used to verify if data is as expected. It can be run with dbt test. There are 3 types of tests:

- a. Built-in: These are 4 predefined tests available in dbt
 - Unique(to check if the column is unique)
 - Not_null(to check if the column contains any nulls)
 - Relationships(to check the relationship between models/columns)
 - accepted_values(to check if the column has any value outside the defined values)
- b. Singular: a simple custom test. It is written as a SQL query and must return failing rows. They are defined as .sql files within the tests/ directory. For example, this checks if there are other values outside prd_1, and prd_2 in the name column

```
select * from
{{ ref('cancellation_per_product') }}
where name not in ('prd_1', 'prd_2')
```

- c. Generic: these are defined under tests/generic and can be added to the models_properties.yml file under models/. These are reusable tests

2.11 Macros

Macros allow you to reuse code across several models. For example, I created two macros under the macros/ directory.

- a. Convert_date.sql: To convert columns from string to timestamp

```
{% macro convert_str_to_timestamp(date, format = 'DD/MM/YYYY HH24:MI') %}

(TO_TIMESTAMP(CASE WHEN {{ date }} = 'NA' THEN null ELSE {{ date }} END,
'{{ format }}'))

{% endmacro %}
```

This can be reused in a model like this:

```
Select {{ convert_str_to_timestamp('DATE_TIME') }} as case_date from {{
ref('model_name') }}
```

- b. Trunc_date.sql: The TRUNC (date) function returns the date with the time portion of the day truncated to the unit specified by the format model fmt.

```
{% macro trunc_date(date, format = 'day') %}

(trunc('{{ date }}', '{{ format }}'))

{% endmacro %}
```

2.12 Create Models

Models are .sql files that contain transformations.

- models/staging/: this contains 1-1 models reading from the source data. For example, stg_customer_cases selects all rows from the source(defined in the sources.yml)

```
select *
FROM
{{ source('raw_CUSTOMER_CASES', 'CUSTOMER_CASES') }}
```


- `models/transformed/` : this contains models transforming data from the staging models. For example, This SQL query performs a left join between the `stg_CUSTOMER_CASES` and `stg_CUSTOMER_PRODUCTS` staging tables to combine case details with product information for each customer. It also includes conversions of date and time columns from string format to timestamp format.

```
-- Join cases and customers

SELECT cases.*,
       prod.PRODUCT,
       {{ convert_str_to_timestamp('prod.SIGNUP_DATE_TIME') }} AS
SIGNUP_DATE_TIME,
       {{ convert_str_to_timestamp('cases.DATE_TIME') }} AS CASE_DATE_TIME,
       {{ convert_str_to_timestamp('prod.CANCEL_DATE_TIME') }} AS
CANCEL_DATE_TIME

FROM {{ ref('stg_CUSTOMER_CASES') }} cases
LEFT JOIN {{ ref('stg_CUSTOMER_PRODUCTS') }} prod
ON cases.customer_id = prod.customer_id
```

2.13 Run Models

`dbt build` is a shortcut to perform all tasks:

It can:

- Run models in order(`dbt run`)
- Run test(`dbt test`)
- Create snapshots(`dbt snapshot`)
- Process seeds(`dbt seed`)

2.14 Snowflake Dashboard

After the `dbt_models` are created. We can visualize the transformations in a snowflake dashboard using an SQL/Python worksheet. Use SQL



2.15 Create ELT data pipeline with Apache Airflow

We use astronomer-cosmos, which allows us to run dbt projects using Airflow dags and task groups.

- a. Pip install astro astro-cosmos
- b. Make a directory under the dbt project directory and Initialize an astro project

```
mkdir dbt_dag / cd dbt_dag / astro dev init
```

- c. Open the DockerFile and add this under

```
RUN python -m venv dbt_venv && source dbt_venv/bin/activate && \
  pip install --no-cache-dir dbt-snowflake && deactivate
```

This is a shell script for setting up a Python virtual environment and installing dbt-snowflake within that environment.

- d. Update the requirements.txt file

```
astronomer-cosmos
apache-airflow-providers-snowflake
```

- e. Start docker engine(Open docker desktop)
- f. Run `astro dev start` in the terminal
- g. Open Airflow in localhost and input admin/admin as username/password
- h. Create a dag.py file under dags/

The script sets up a dbt DAG for Apache Airflow using the Cosmos library. It specifies the dbt project location, Snowflake connection details, execution environment, and scheduling. This configuration allows the automated execution of dbt tasks within the specified schedule.

```
import os
from datetime import datetime

from cosmos import DbtDag, ProjectConfig, ProfileConfig, ExecutionConfig
from cosmos.profiles import SnowflakeUserPasswordProfileMapping
```

Configures the profile for connecting to Snowflake.

`profile_name` is set to "default", and `target_name` is set to "dev".

Uses `SnowflakeUserPasswordProfileMapping` with connection ID "snowflake_conn" and specifies database and schema for dbt operations.

```
profile_config = ProfileConfig(
    profile_name="default",
    target_name="dev",
    profile_mapping=SnowflakeUserPasswordProfileMapping(
        conn_id="snowflake_conn",
        profile_args={"database": "customer_subscription", "schema":
"dbt_outputs"},
    )
)
```

project_config: Points to the location of the dbt project.

operator_args: Includes "install_deps": `True` to ensure dependencies are installed.

profile_config: Uses the previously defined Snowflake profile configuration.

execution_config: Sets the path to the dbt executable, using an environment variable `AIRFLOW_HOME` to locate the virtual environment.

schedule_interval: Specifies that the DAG should run daily (`@daily`).

start_date: Sets the start date for the DAG execution.

catchup: Disabled with `False`, meaning the DAG will not backfill missed runs.

dag_id: Identifies the DAG with the name "dbt_dag".

```
dbt_snowflake_dag = DbtDag(

project_config=ProjectConfig("/usr/local/airflow/dags/dbt/data_pipeline",),
    operator_args={"install_deps": True},
```

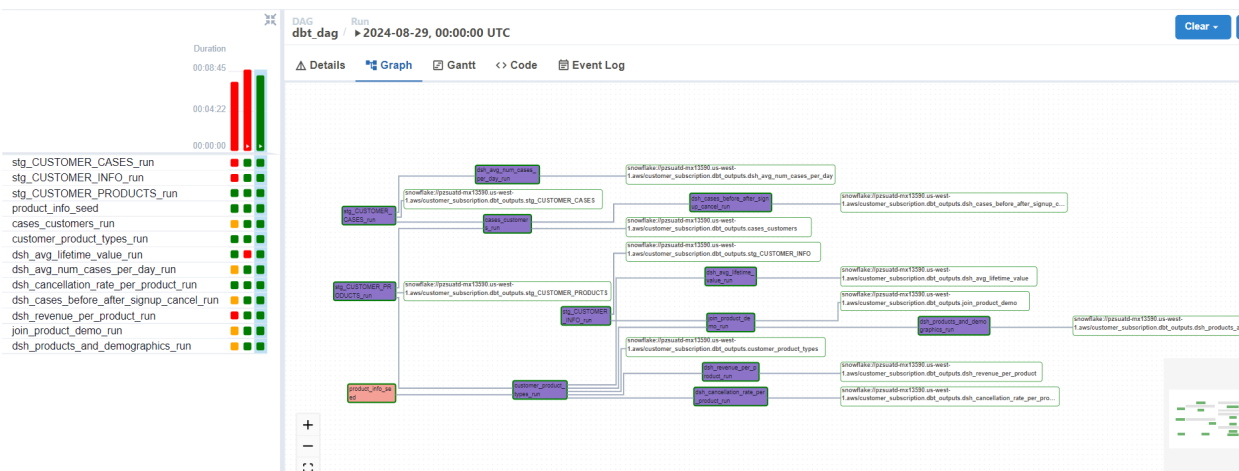
```

profile_config=profile_config,

execution_config=ExecutionConfig(dbt_executable_path=f"{os.environ['AIRFLOW_HOME']}/dbt_venv/bin/dbt",),
    schedule_interval="@daily",
    start_date=datetime(2023, 9, 10),
    catchup=False,
    dag_id="dbt_dag",
)

```

- i. Copy and paste the dbt project directory under dags/dbt/data_pipeline
- j. Create a new snowflake connection in Airflow under admins/connections. Match everything in the dbt profiles.yml file



2.16 Updates changes to GitHub


- a. Add to the staging area: git add .
- b. Commit changes: git commit -m "Commit message"
- c. git push origin dev
- d. Create pull request and merge

2.17 Generate dbt documentation

Dbt provides the option to add documentation to projects. We can descriptions under yml files.

Generate it with dbt doc generate

Access it in a web server with dbt docs serve



Search for models...

Overview

Project Database Group

Sources

- raw_CUSTOMER_CASES
- raw_CUSTOMER_INFO
- raw_customer_products

Projects

- cut_subscription_pipeline
 - macros
 - convert_str_to_timestamp
 - trunc_date
 - models
 - staging
 - stg_CUSTOMER_CASES
 - stg_CUSTOMER_INFO
 - stg_CUSTOMER_PRODUCTS
 - transformed
 - seeds

raw_CUSTOMER_CASES source

Details Description Sources

Details

CONTRACT	LOADER	OWNER	DATABASE	SCHEMA	TABLES
Not Enforced		ACCOUNTADMIN	CUSTOMER_SUBSCRIPTION	RAW_DATA	1

Description

This file contains all cases received from customers from various channels and reasons.

Source Tables

SOURCE	TABLE	DESCRIPTION	LINK	MORE?
raw_CUSTOMER_CASES	CUSTOMER_CASES		View docs	



REFERENCES

1. <https://www.youtube.com/watch?v=OLXkGB7krGo>
2. <https://app.datacamp.com/learn/courses/introduction-to-dbt>
- 3.