Feature Engineering Pipeline Using Snowflake

Abstract

This document describes how to use Snowflake to implement a Feature Engineering pipeline. Processing, cleaning, transforming, and producing useful features for machine learning workflows are the pipeline's main objectives. The document contains the project's SQL scripts, Python code snippets, configuration information, and step-by-step methodology.

Introduction

A crucial stage in creating machine learning models is feature engineering. It entails converting unprocessed data into useful variables (features) that enhance the performance and accuracy of the model. Snowflake offers a scalable and secure environment for storing, preprocessing, and transforming big datasets in light of the growing popularity of cloud data platforms.

This project shows how data can be ingested, cleaned, and transformed into a structured feature set using Snowflake. Additionally, Python is integrated into the workflow to carry out extra transformations where SQL might not be adequate. The optimised final processed data is prepared for model development and implementation.

Objectives

Building a clean data pipeline within Snowflake is one of the project's main goals.

- to carry out preprocessing and dataset validation.
- to use Python and SQL transformations to create engineered features.
- to save converted outputs for use and model training.
- to guarantee the process's scalability, maintainability, and repeatability.

Tools and Technologies Used

Component	Technology
Cloud Data Warehouse	Snowflake
Development Language	SQL, Python
Execution Platform	Snowflake Worksheets / Snowpark
Data Processing Type	Feature Engineering

Dataset Description

The structured tabular data used in this project was uploaded to Snowflake after being stored in CSV/Parquet format. It has several columns that show transactional or raw customer characteristics. Before being used for machine learning, some fields need to be cleaned, formatted, and have additional values derived.

The dataset typically contains the following kinds of variables:

- Numerical fields
- Categorical fields
- Date and timestamp fields
- Identifier columns
- Derived or calculated metrics

The dataset is used as the starting point for transformations and the creation of refined features.

Workflow Overview

Within Snowflake, the feature engineering process adheres to a sequential workflow. The process consists of:

1. Uploading the raw dataset to the Snowflake internal/external stage is known as "data ingestion."
2. Data loading involves transferring staged data into a Snowflake table using COPY INTO.
3. Preprocessing and data cleaning include addressing missing values, standardising formats, and eliminating inconsistencies.
4. Feature engineering is the process of creating new variables using transformations like date-based feature generation, encoding, normalisation, and aggregation.
5. Feature storage is the process of storing processed data in a final feature store table for use in machine learning.

This provides a clear and repeatable pipeline from initial raw data to final usable feature outputs.

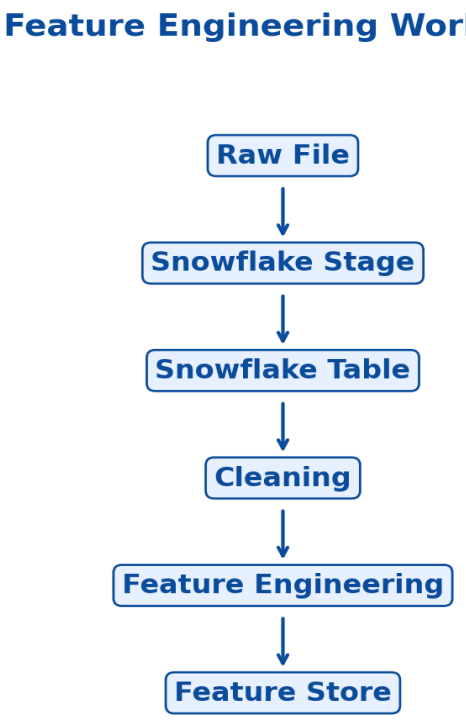

Figure:1. Working diagram

Project Setup in Snowflake

Before running the pipeline, the environment must be configured with a warehouse, database, and schema.

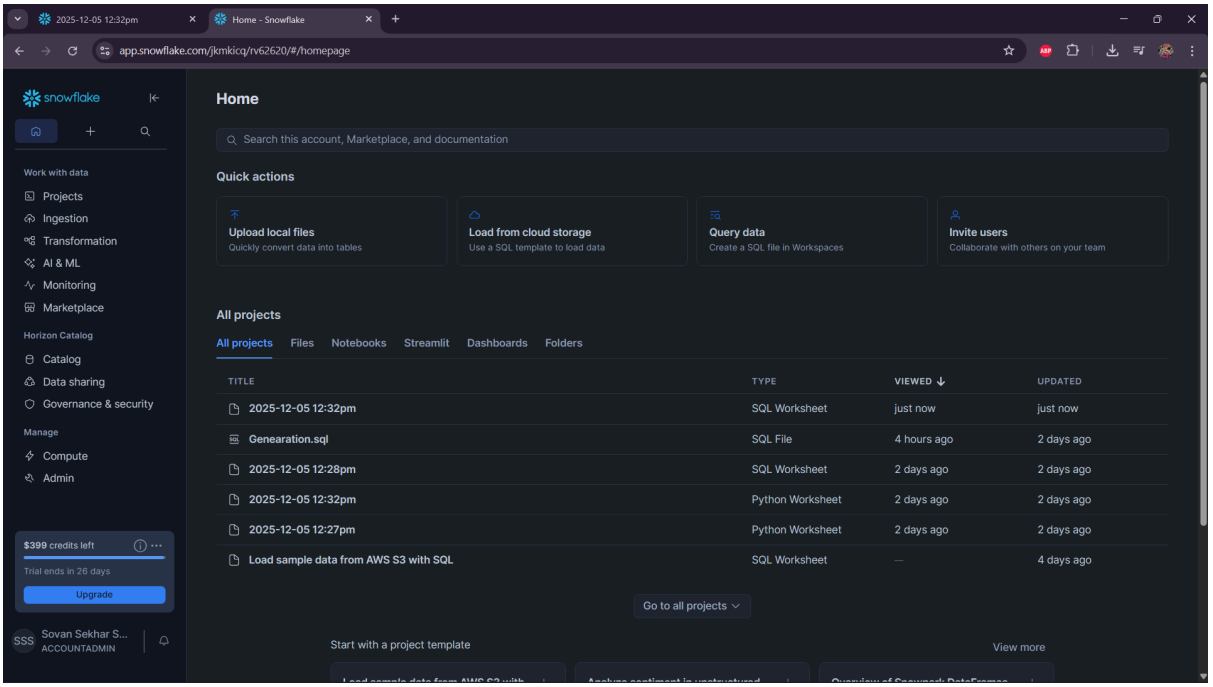

Figure 2: Snowflake Dashboard

SQL Script: Creating Warehouse, Database, and Schema

```
CREATE OR REPLACE WAREHOUSE FE_ENGINEERING_WH
WITH WAREHOUSE_SIZE = 'XSMALL'
AUTO_SUSPEND = 300
AUTO_RESUME = TRUE;

CREATE OR REPLACE DATABASE FEATURE_ENGINEERING_DB;

CREATE OR REPLACE SCHEMA FEATURE_ENGINEERING_DB.RAW_DATA;
```

This script ensures that Snowflake has an allocated compute environment and logical storage structure for handling data efficiently.

Creating Internal Stage for File Upload

```
CREATE OR REPLACE STAGE
FEATURE_ENGINEERING_DB.RAW_DATA.INPUT_STAGE;
```

This stage is used to store the raw input files before loading them into a Snowflake table.

Loading Raw Data into Snowflake Table

```
CREATE OR REPLACE TABLE RAW_CUSTOMER_DATA (
    CUSTOMER_ID STRING,
    NAME STRING,
    AGE INTEGER,
    PURCHASE_AMOUNT FLOAT,
    SIGNUP_DATE DATE,
    COUNTRY STRING
);

COPY INTO RAW_CUSTOMER_DATA
FROM @FEATURE_ENGINEERING_DB.RAW_DATA.INPUT_STAGE
FILE_FORMAT = (TYPE = 'CSV' FIELD_OPTIONALLY_ENCLOSED_BY='"'
SKIP_HEADER=1);
```

This script imports the staged dataset into a structured Snowflake table for further processing.

Data Cleaning and Validation

Once the raw data is loaded, the next step is to clean and validate records to ensure consistency and correctness. Data quality checks are performed to identify missing values, duplicate records, incorrect data formats, and outliers.

Basic Data Inspection

```
SELECT * FROM RAW_CUSTOMER_DATA LIMIT 10;
```

This query displays sample records to verify correct ingestion.

Checking for Missing Values

```
SELECT
```

```
    SUM(CASE WHEN CUSTOMER_ID IS NULL THEN 1 ELSE 0 END) AS
MISSING_CUSTOMER_ID,
    SUM(CASE WHEN AGE IS NULL THEN 1 ELSE 0 END) AS MISSING_AGE,
    SUM(CASE WHEN PURCHASE_AMOUNT IS NULL THEN 1 ELSE 0 END) AS
MISSING_PURCHASE_AMOUNT,
    SUM(CASE WHEN SIGNUP_DATE IS NULL THEN 1 ELSE 0 END) AS
MISSING_SIGNUP_DATE
FROM RAW_CUSTOMER_DATA;
```

This helps identify columns requiring imputation or correction.

Removing Duplicate Records

```
DELETE FROM RAW_CUSTOMER_DATA
WHERE CUSTOMER_ID IN (
    SELECT CUSTOMER_ID
    FROM RAW_CUSTOMER_DATA
    GROUP BY CUSTOMER_ID
    HAVING COUNT(*) > 1
);
```

This step ensures that only unique records exist, particularly on primary identifiers.

Standardizing Text Data

Categorical and name fields may contain inconsistent formatting. The following transformations standardize text to improve feature consistency.

```
UPDATE RAW_CUSTOMER_DATA
SET COUNTRY = UPPER(COUNTRY),
    NAME = INITCAP(NAME);
```

- `UPPER()` ensures consistency for categorical grouping
- `INITCAP()` standardizes name formatting

Handling Missing Numeric Values

Numeric missing values such as age or purchase amount can be filled using statistical imputation. In this case, mean imputation is used.

```
UPDATE RAW_CUSTOMER_DATA
SET AGE = (SELECT AVG(AGE) FROM RAW_CUSTOMER_DATA)
WHERE AGE IS NULL;
```

Similarly, for purchase amount:

```
UPDATE RAW_CUSTOMER_DATA
SET PURCHASE_AMOUNT = (SELECT AVG(PURCHASE_AMOUNT) FROM
RAW_CUSTOMER_DATA)
WHERE PURCHASE_AMOUNT IS NULL;
```

Validating Date Format and Filtering Invalid Entries

```
DELETE FROM RAW_CUSTOMER_DATA
WHERE TRY_TO_DATE(SIGNUP_DATE) IS NULL;
```

This removes inconsistent date records that cannot be parsed.

Creating a Clean Version of the Dataset

After preprocessing, the cleaned dataset is stored in a separate table to preserve traceability and versioning.

```
CREATE OR REPLACE TABLE CLEAN_CUSTOMER_DATA AS
SELECT * FROM RAW_CUSTOMER_DATA;
```

This table becomes the foundation for generating engineered features.

Feature Engineering

After cleaning the dataset, the next step is to generate new features that add more learning value for the model. Feature engineering improves model performance by transforming raw values into more meaningful representations.

This includes:

- Derived calculated fields
- Date-based feature extraction
- Encoding categorical values
- Scaling and normalization
- Binning and segmentation

Each section below includes the logic and implementation.

1. Derived Numerical Features

A common transformation is generating new metrics using existing values. Here, a discount-adjusted purchase metric is created for better revenue modeling.

```
ALTER TABLE CLEAN_CUSTOMER_DATA ADD COLUMN PURCHASE_CATEGORY STRING;

UPDATE CLEAN_CUSTOMER_DATA

SET PURCHASE_CATEGORY =

  CASE

    WHEN PURCHASE_AMOUNT > 500 THEN 'HIGH'

    WHEN PURCHASE_AMOUNT BETWEEN 200 AND 500 THEN 'MEDIUM'

    ELSE 'LOW'

  END;
```

This converts a continuous variable into a categorical grouped feature that may be more useful for segmentation.

2. Date-Based Features

Date columns are useful for extracting patterns like user activity, retention, or seasonal trends.

```
ALTER TABLE CLEAN_CUSTOMER_DATA

ADD COLUMN SIGNUP_YEAR INT,

ADD COLUMN SIGNUP_MONTH INT,

ADD COLUMN SIGNUP_DAY INT;

UPDATE CLEAN_CUSTOMER_DATA

SET SIGNUP_YEAR = YEAR(SIGNUP_DATE),

   SIGNUP_MONTH = MONTH(SIGNUP_DATE),

   SIGNUP_DAY = DAY(SIGNUP_DATE);
```

These new values can help machine learning models detect seasonality or cohort behavior.

3. Encoding Categorical Variables Using SQL

Snowflake supports efficient encoding through mapping tables.

```
ALTER TABLE CLEAN_CUSTOMER_DATA ADD COLUMN COUNTRY_CODE INT;

UPDATE CLEAN_CUSTOMER_DATA

SET COUNTRY_CODE =

   CASE COUNTRY

      WHEN 'INDIA' THEN 1

      WHEN 'USA' THEN 2

      WHEN 'CANADA' THEN 3

      ELSE 0
```

```
    END;
```

This converts text values into numerical encoding suitable for algorithms.

4. Normalizing Continuous Values (Using Python Snowpark)

Python can be used alongside Snowflake Snowpark to normalize numerical features such as purchase amount.

```
from snowflake.snowpark import Session

from snowflake.snowpark.functions import col

# Normalization using min-max scaling

df = session.table("CLEAN_CUSTOMER_DATA")

min_val = df.agg({"PURCHASE_AMOUNT" : "min"}).collect()[0][0]

max_val = df.agg({"PURCHASE_AMOUNT" : "max"}).collect()[0][0]

df_norm = df.with_column(

    "PURCHASE_AMOUNT_NORMALIZED",

    (col("PURCHASE_AMOUNT") - min_val) / (max_val - min_val)

)

df_norm.write.mode("overwrite").save_as_table("FEATURE_ENGINEERED_DATA")
```

This adds a normalized version of the purchase value suitable for machine learning training.

5. Final Feature Store Table

After all feature transformations, the final structured feature table is created.

```
CREATE OR REPLACE TABLE FINAL_FEATURE_STORE AS

SELECT
```

```
    CUSTOMER_ID,
    AGE,
    PURCHASE_AMOUNT,
    PURCHASE_CATEGORY,
    COUNTRY_CODE,
    SIGNUP_YEAR,
    SIGNUP_MONTH,
    SIGNUP_DAY,
    PURCHASE_AMOUNT_NORMALIZED
FROM FEATURE_ENGINEERED_DATA;
```

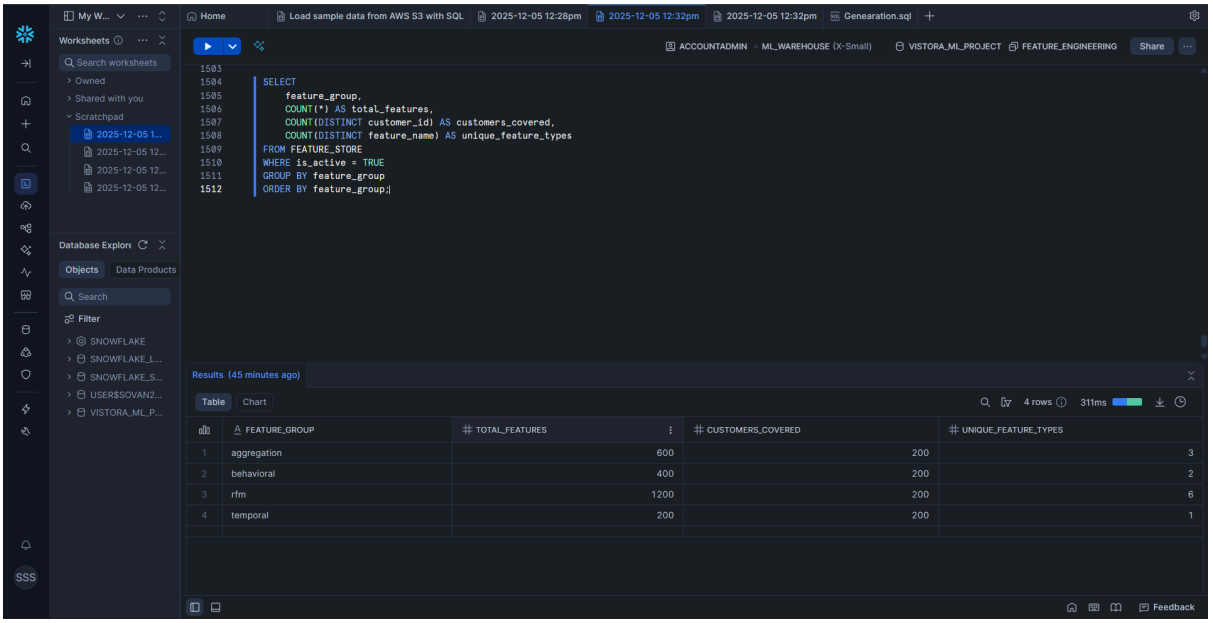

Figure 3: RFM_segmentation

This table serves as the finalized engineered feature set for downstream analytics or model training.

Model Readiness Validation

Before using the engineered dataset for machine learning, a final validation step ensures that:

- All required fields exist
- No critical missing values remain
- Data types are consistent and correct
- Encoded features are usable for model input

The validation is performed using the following checks:

```
DESC TABLE FINAL_FEATURE_STORE;
```

To verify numeric completeness:

```
SELECT
    COUNT(*) AS TOTAL_RECORDS,
    SUM(CASE WHEN AGE IS NULL THEN 1 ELSE 0 END) AS NULL_AGE,
    SUM(CASE WHEN PURCHASE_AMOUNT_NORMALIZED IS NULL THEN 1 ELSE 0 END) AS NULL_NORMALIZED_AMOUNT
FROM FINAL_FEATURE_STORE;
```

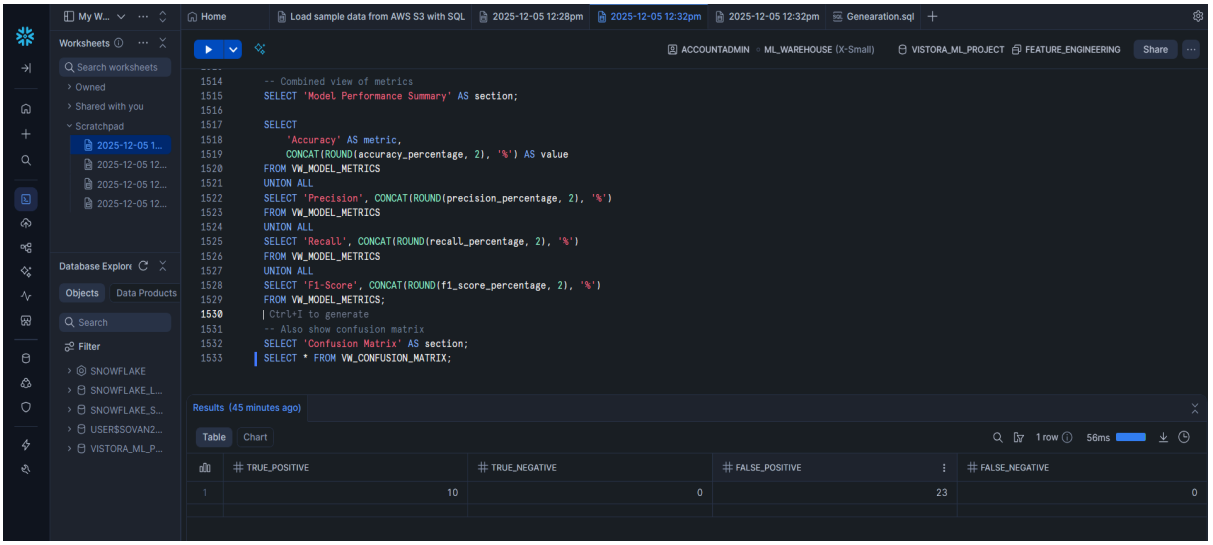

Figure 4: Model Performance

To verify categorical encoding consistency:

```
SELECT DISTINCT PURCHASE_CATEGORY, COUNTRY_CODE FROM
FINAL_FEATURE_STORE;
```

These checks confirm that the final table is clean, structured, and ready for AI/ML workflows.

Output Summary

The system produces a final feature dataset with the following attributes at the end of the pipeline:

- Clean and verified records
- Newly developed categorical and numerical features
- Normalised and encoded values
- Structured data that ML pipelines can use

The primary dataset for model deployment, training, and monitoring is the table FINAL_FEATURE_STORE.

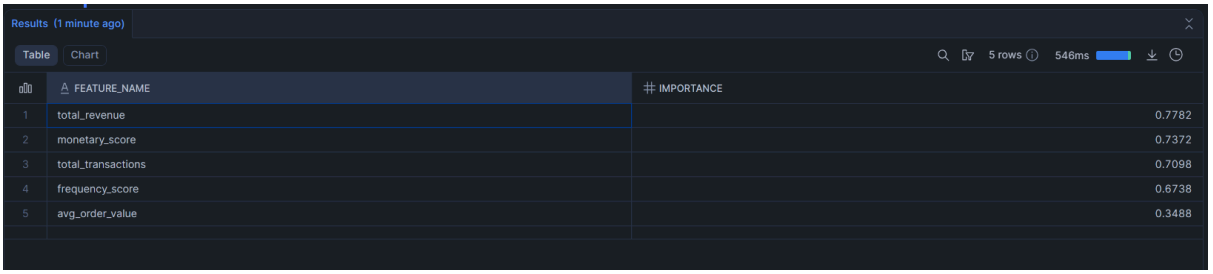

Figure 5: Result

Conclusion

This project effectively illustrates how SQL and Python Snowpark can be used to implement feature engineering within Snowflake. Through a methodical process of cleaning, validation, transformation, and encoding, the workflow turns unstructured data into a dataset ready for modelling.

Scalability, effectiveness, and simplicity of integration with external machine learning tools are guaranteed when Snowflake is used as the central processing layer. The end product is an optimised and repeatable feature engineering pipeline that can be used in practical data science and analytics applications

Future Enhancements

The following improvements can be added in later stages:

- Tracking feature lineage through integration with Snowflake Feature Store
- Snowflake Tasks for planned and incremental feature updates
- Automation of feature drift monitoring and retraining
- Connectivity to machine learning systems like Databricks, Azure ML, and AWS Sagemaker

The feature engineering pipeline will become more automated, maintainable, and production-ready as a result of these improvements.