Project Workflow – Feature Engineering Using Snowflake and Feature Stores

Introduction

This project was developed as part of the Vistora AI – AI/ML Assignment, focusing on real-world feature engineering using Snowflake for data handling and a feature store for preparing model-ready data. The primary objective was to extract raw transactional data, engineer meaningful features, and structure the output in a way that integrates seamlessly with a machine learning pipeline.

Dataset

We used the UCI Online Retail Dataset, which contains over 500,000 transaction records from a UK-based online retail store. Each record includes:

- Invoice information
- Product details
- Quantity and price
- Timestamp of purchase
- Customer ID and location

🔗 Dataset Source

Project Structure

SQL commands for extraction & loading

- SQL_Link: https://app.snowflake.com/trgayrn/ny30645/w43gxBMCQ8RB/query
- EDA.ipynb: Feature engineering logic in Jupyter all the plots and the feature engineering done in this notebook
- Retail_transection_table: Raw data without any feature engineering
- Retail_feature_store: Stored feature engineered data from the raw data
- README.md : Documentation & setup guide

Data Storage and Feature Store

- Snowflake was used to store both the raw data and the final engineered feature table.
- The feature table is organized by CustomerID and contains high-value attributes for modeling.
- Using Snowflake as a feature store ensures scalability, security, and seamless integration with tools like Snowpark, Python, and third-party ML platforms.

For SQL Logic

Click here for Snowflake SQL Commands

This query performs data extraction, transformation, and feature table creation directly in Snowflake.

Column	Description
InvoiceNo	Invoice ID, starts with ‘C’ if cancelled
StockCode	Product code
Description	Product description
Quantity	Number of items purchased
InvoiceDate	Date and time of transaction
UnitPrice	Price per item
CustomerID	Unique customer ID
Country	Country of the customer

\

Engineered Features

Feature Name	Description
CustomerID	Unique user ID
Country_OneHot	One-hot encoded country field
Total_Orders	Total number of invoices per customer
Total_Items	Sum of quantity purchased per customer
Total_Spent	TotalAmount = Quantity × UnitPrice
Avg_Items_PerOrder	Average items per order
Avg_Revenue_PerOrder	Average revenue per order
CLV_Per_Day	Customer lifetime value normalized over time
Recency_Days	Days since last purchase
Customer_Lifetime_Days	Days between first and last purchase
Return_Ratio	Ratio of cancelled invoices to total invoices
Last_Purchase_Date	Timestamp of last transaction

RETAIL_TRANSACTIONS

	INVOICENO	STOCKCODE	DESCRIPTION	QUANTITY	INVOICEDATE	UNITPRICE
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01T08:26:00.000+0000	2.55
2	536365	71053	WHITE METAL LANTERN	6	2010-12-01T08:26:00.000+0000	3.39
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01T08:26:00.000+0000	2.75
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01T08:26:00.000+0000	3.39
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01T08:26:00.000+0000	3.39
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01T08:26:00.000+0000	7.65

CUSTOMERID	COUNTRY
17850.0	United Kingdom
17850.0	United Kingdom
17850.0	United Kingdom
17850.0	United Kingdom
17850.0	United Kingdom
17850.0	United Kingdom

RETAIL_FEATURE_STORE

	CUSTOMERID	COUNTRY	IS_UK	IS_GERMANY	IS_FRANCE ···	IS_EIRE	IS_SPAIN	TOTAL_ORDERS	TOTAL_ITEMS
1	17850.0	United Kingdom	TRUE	FALSE	FALSE	FALSE	FALSE	35	1693
2	13047.0	United Kingdom	TRUE	FALSE	FALSE	FALSE	FALSE	18	1355
3	12583.0	France	FALSE	FALSE	TRUE	FALSE	FALSE	18	5009
4	13748.0	United Kingdom	TRUE	FALSE	FALSE	FALSE	FALSE	5	439
5	15100.0	United Kingdom	TRUE	FALSE	FALSE	FALSE	FALSE	6	58
6	15291.0	United Kingdom	TRUE	FALSE	FALSE	FALSE	FALSE	20	2074

AVG_ITEMS_PER_ORDER	AVG_REVENUE_PER_ORDER	CLV_PER_DAY	RECENCY_DAYS	CUSTOMER_LIFETIME_DAYS	RETURN_RATIO
5.426282	151.103714286	73.453194444	5204	71	0.048077
6.913265	171.061111111	8.97696793	4933	342	0.122449
19.956175	399.296666667	19.320806452	4904	371	0.015936
15.678571	189.65	3.39874552	4997	278	0
9.666667	105.85	14.434090909	5232	43	0.5
19.027523	229.8255	13.170515759	4927	348	0.055046

Why These Features Matter

These engineered features are much more informative and structured than the raw transactional data:

- Customer-level aggregation enables segmentation and personalization.
- Monetary features like Total_Spent and CLV_Per_Day provide value estimation.
- Recency and Lifetime metrics are critical for churn prediction and RFM modeling.
- Encoded country features make it ML-compatible while retaining geographic signals.
- Return ratio can indicate dissatisfaction or risk of churn.

These transformations convert low-signal raw logs into high-signal, ML-ready features.

Integration into a Machine Learning Pipeline

The engineered feature table can be integrated into any supervised ML pipeline using the following workflow:

1. Feature Table in Snowflake: Stored and refreshed periodically.
2. Model Input: Load features using Snowpark or Python connector.
3. Training: Train churn/CLV prediction models using the engineered dataset.
4. Inference: Query the feature store for real-time or batch prediction.
5. Model Monitoring: Continuously evaluate and update features based on feedback.

For More Information

Please read the README.md file in the GitHub repository for setup instructions, code walkthrough, and environment configuration.

SQL code snippet and Explnation:

```sql
CREATE OR REPLACE TABLE retail_feature_store AS
SELECT DISTINCT CustomerID, Country
FROM retail_transactions
WHERE CustomerID IS NOT NULL;
```

We use one-hot encoding to convert the Country column into boolean flags. This allows ML models to treat countries as independent binary variables.

```sql
-- One-Hot Encoded Country Features
ALTER TABLE retail_feature_store ADD COLUMN is_UK BOOLEAN;
ALTER TABLE retail_feature_store ADD COLUMN is_Germany BOOLEAN;
ALTER TABLE retail_feature_store ADD COLUMN is_France BOOLEAN;
ALTER TABLE retail_feature_store ADD COLUMN is_EIRE BOOLEAN;
ALTER TABLE retail_feature_store ADD COLUMN is_Spain BOOLEAN;

UPDATE retail_feature_store
SET
  is_UK = IFF(Country = 'United Kingdom', 1, 0),
  is_Germany = IFF(Country = 'Germany', 1, 0),
  is_France = IFF(Country = 'France', 1, 0),
  is_EIRE = IFF(Country = 'EIRE', 1, 0),
  is_Spain = IFF(Country = 'Spain', 1, 0);
```

We add a new column total_orders which counts the number of unique invoices per customer — this is a key frequency feature for customer behavior modeling.

```sql
--Total Orders per Customer
ALTER TABLE retail_feature_store ADD COLUMN total_orders INT;

UPDATE retail_feature_store AS t
SET total_orders = (
  SELECT COUNT(DISTINCT InvoiceNo)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

This computes the total number of items purchased by a customer. It reflects overall engagement and purchasing volume.

```sql
--Total Items Purchased
ALTER TABLE retail_feature_store ADD COLUMN total_items INT;

UPDATE retail_feature_store AS t
SET total_items = (
  SELECT SUM(Quantity)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

This computes the total number of items purchased by a customer. It reflects overall engagement and purchasing volume.

```sql
--Total Spend
ALTER TABLE retail_feature_store ADD COLUMN total_spent FLOAT;

UPDATE retail_feature_store AS t
SET total_spent = (
  SELECT SUM(Quantity * UnitPrice)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

We calculate the total revenue generated by each customer. This is the Monetary part of the RFM model and helps in identifying high-value customers.

```sql
--Avg Items per Order
ALTER TABLE retail_feature_store ADD COLUMN avg_items_per_order FLOAT;

UPDATE retail_feature_store AS t
SET avg_items_per_order = (
  SELECT AVG(Quantity)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

This metric tells us the average monetary value per order, helping assess customer value trends.

```sql
--CLV per Day
ALTER TABLE retail_feature_store ADD COLUMN clv_per_day FLOAT;
UPDATE retail_feature_store AS t
SET clv_per_day = (
  SELECT SUM(Quantity * UnitPrice) / NULLIF(DATEDIFF('day', MIN(InvoiceDate), DATEADD(day, 1, MAX(InvoiceDate))), 0)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

clv_per_day represents the average daily revenue generated by a customer over their active lifetime. It balances both value and time.

```sql
--First and Last Purchase Dates
ALTER TABLE retail_feature_store ADD COLUMN first_purchase_date DATE;
ALTER TABLE retail_feature_store ADD COLUMN last_purchase_date DATE;

UPDATE retail_feature_store AS t
SET first_purchase_date = (
  SELECT MIN(CAST(InvoiceDate AS DATE))
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
),
last_purchase_date = (
  SELECT MAX(CAST(InvoiceDate AS DATE))
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);
```

These two features track the **first and most recent transactions**. They're foundational for calculating recency and lifetime metrics.

```sql
ALTER TABLE retail_feature_store ADD COLUMN recency_days INT;
ALTER TABLE retail_feature_store ADD COLUMN customer_lifetime_days INT;

UPDATE retail_feature_store
SET recency_days = DATEDIFF('day', last_purchase_date, CURRENT_DATE),
    customer_lifetime_days = DATEDIFF('day', first_purchase_date, last_purchase_date);
```

recency_days: Days since the customer last purchased — key for churn prediction.

customer_lifetime_days: Number of active days between first and last purchase.

```sql
--Return Ratio
ALTER TABLE retail_feature_store ADD COLUMN return_ratio FLOAT;

UPDATE retail_feature_store AS t
SET return_ratio = (
  SELECT SUM(IFF(Quantity < 0, 1, 0)) / COUNT(*)
  FROM retail_transactions r
  WHERE r.CustomerID = t.CustomerID
);

select * from retail_feature_store
```

This measures the **proportion of cancelled or returned items**, indicating customer dissatisfaction or refund behavior.

Question asked in the Assignment:

1. **Define feature engineering and explain its importance in machine learning.**
   Feature engineering is the process of creating new useful data (called *features*) from the existing raw data. These features help machine learning models understand the data better and make more accurate predictions**.** Machine learning models don't work well with raw data. If we prepare the data properly using feature engineering, the model becomes smarter and gives better results.

2. **Describe different types of feature engineering techniques.**
   Common feature engineering techniques:
   - Normalization:
     This means scaling the data so all values fall within a certain range. It helps the model treat each feature fairly.
   - Encoding:
     Converting text or categories (like countries or product names) into numbers so the model can understand them.
   - Time-based aggregations:
     Creating features from time columns, like extracting the hour, day, or month from a timestamp, or calculating how recently someone made a purchase.
3. **How Snowflake is Used for Storing Structured and Semi-Structured Data.**
   Snowflake is a cloud-based data warehouse that supports the storage and processing of both structured and semi-structured data. Structured data refers to data that fits neatly into tables with rows and columns, such as customer records, transaction logs, or sales data. Snowflake allows users to create and manage such data using standard SQL commands, making it easy to query and analyze.
   In addition to structured data, Snowflake is also designed to handle semi-structured data formats like JSON, XML, Avro, and Parquet. This is useful when dealing with web data, logs, or complex nested information. Snowflake uses a special data type called `VARIANT` to store semi-structured data without requiring a predefined schema. This means you can load JSON files directly into a table and still be able to extract and filter individual elements using SQL queries. This combination of structured and semi-structured support makes Snowflake very flexible for modern data analytics and machine learning workflows.

4. **Example of SQL Query to Extract and Preprocess Data in Snowflake**

   Let's say we have a retail transactions table, and we want to prepare some customer-level features for machine learning. In Snowflake, you can use SQL to extract and preprocess data easily. For example:

```sql
SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS total_orders,
  SUM(Quantity) AS total_items,
  SUM(Quantity * UnitPrice) AS total_spent,
  AVG(Quantity) AS avg_items_per_order
FROM retail_transactions
WHERE CustomerID IS NOT NULL
GROUP BY CustomerID;
```

   This query does several things at once: it groups transactions by customer, counts how many orders each customer made, adds up how many items they purchased, calculates the total amount they spent, and finds the average number of items they buy per order. This is a great example of how Snowflake makes it easy to go from raw data to engineered features using just SQL.

5. **How Snowflake Integrates with Machine Learning Pipelines**

   Snowflake can be seamlessly integrated into machine learning pipelines in various ways. Once the data is cleaned and engineered using SQL in Snowflake, it can be pulled into ML models using tools like Snowpark for Python, Spark, or external Python scripts with Snowflake connectors. These tools allow data scientists to run their code either inside or outside Snowflake while still accessing Snowflake's high-performance computing power and secure storage.

   For example, you can write a Python script in Jupyter Notebook to read a feature table directly from Snowflake using a connector. You can then use that data to train a machine learning model, such as a churn prediction model or customer segmentation. This tight integration ensures that data engineers and data scientists can collaborate more effectively, using a single source of truth for both training and inference data.

6. **What is a Feature Store and Why is it Needed?**

   A Feature Store is a centralized system where all machine learning features are stored, managed, and shared. Instead of creating features separately for each model, a feature store allows teams to build a set of features once and reuse them across many models. This not only saves time but also ensures consistency between training and prediction phases.

   Feature stores are important because they solve common ML problems like "training-serving skew," which happens when the data used during model training is different from the data used during prediction. By using the same feature definitions from a feature store, we make sure the model always sees consistent and accurate data. Feature stores also improve collaboration, make experiments reproducible, and make deploying models faster and easier.

7. **Comparison of Different Feature Stores**

   There are several feature stores available today, each with its strengths. AWS SageMaker Feature Store is tightly integrated with the AWS ecosystem. It supports both real-time and batch features and works well with SageMaker for training and deploying models. Snowflake Feature Store is a lightweight and flexible option for teams already using Snowflake. It lets you create, store, and retrieve features using SQL, and integrates with ML tools like Snowpark and Python connectors. It's a great choice if your data is already in Snowflake.

   Databricks Feature Store is designed for big data environments and works well with Apache Spark. It's ideal when you're already using Databricks notebooks and Spark ML. It supports both offline (batch) and online (real-time) feature lookups.

   In summary, if you're using AWS services, SageMaker's feature store might be the best fit. If your data is already in Snowflake and you prefer SQL workflows, Snowflake Feature Store is convenient and simple. And if you're working with large-scale distributed data and Spark-based workflows, Databricks Feature Store is very powerful.