

VISTORA AI ASSIGNMENT

FEATURE ENGINEERING USING SNOWFLAKE AND FEATURE STORES

OBJECTIVE:

The goal of this assignment is to understand how feature engineering works in real-world scenarios, particularly how data is extracted from Snowflake, transformed, and stored in a Feature Store for use in machine learning models.

1. Introduction to Feature Engineering:

Feature Engineering is the process of transforming raw data into features that are suitable for machine learning. It involves selecting, extracting and transforming the relevant features from the available data to build more efficient and accurate machine learning models.

Importance of Feature Engineering:

- 1.) It transforms raw data into useful input because raw data is in a messy form and it cannot be fed directly into the model.
- 2.) It improves the model accuracy as good features capture the underlying structure of the data. Better features results in higher predictive performance.
- 3.) It reduces overfitting.
- 4.) There is no need to design complex models if better features are available thus making it easier for interpretation.
- 5.) Missing data or noisy data is handled.

Different types of Feature engineering techniques:

1.) One Hot Encoding: It is a technique used to convert categorical variables into numerical values. Each categorical value is transformed into a binary value(1 indicating it's presence and 0 indicating it's absence). For example, consider a categorical variable "Cricketer". It may have 3 categories Batsman, Bowler and Fielder. So, they can be one hot encoded using a three digit binary code.

2.) Normalization: It is a technique that adjusts the values of numeric features to a common scale without distorting differences in the range of values. It is required as many ML algorithms are sensitive to the scale of features. They cannot handle larger values and also it requires more computational power. There are various types of normalization but commonly used in practice are,

Min-Max Normalization: Let x be the value of the feature, then normalized feature value y is given by,

$$y = (x - x_{\min}) / (x_{\max} - x_{\min})$$

This restricts the feature value range from [0,1]. This technique is good for neural networks and algorithms that assume the bounded input.

Standardization: Let x be the value of the feature, then normalized feature value y is given by,

$$y = (x - m) / s$$

where m is the mean of the feature and s is the standard deviation of feature. It's basically used in case of Linear models, PCA and Logistic regression.

3.) Time based aggregations: They are operations where data is summarized and grouped based on time intervals such as by hour, day, week or month etc. These are especially useful for series data like sales, website traffic, sensor readings etc. This helps in smoothing the noise and identifying patterns over regular intervals.

2. Snowflake for data storage and processing:

For Semi-Structured data:

Semi-Structured data is stored using the following data types.

- 1.) Array
- 2.) Object(HashMap or a dictionary).
- 3.) Variant (Analogous to pointer).

(If imported data is split into multiple columns before it is stored, then some or all of those columns can be simple data types, such as FLOAT, VARCHAR, etc.)

The ARRAY, OBJECT, and VARIANT data types can be used individually, or nested to build a hierarchy.

If the data is imported in JSON, Avro, ORC, or Parquet format, then Snowflake can build the hierarchy for you and store it in a VARIANT. You can also create a hierarchy manually.

For Structured data:

- **Understanding Table Structures:**
 - Micro-partitions and data clustering for optimizing storage and query performance.

- Logical and physical structures of tables.
- Choosing appropriate clustering keys for large tables.
- **Data Types:**
 - Snowflake supports various data types, including structured types like ARRAY, OBJECT, and MAP.
 - Structured types contain elements or key-value pairs with specific Snowflake data types.
- **Data Loading:**
 - Data is loaded into Snowflake tables and then stored in Snowflake's optimized, compressed, columnar format.
 - Snowflake manages all aspects of data storage, including organization, file size, compression, and metadata.
- **Database and Schema:**
 - Databases and schemas are used to logically organize data in Snowflake.
 - Databases are logical groupings of schemas, while schemas group database objects like tables and views.
- **Storage Considerations:**
 - Snowflake recommends compressing large data files when loading.
 - Data files in internal stages are not subject to Time Travel and Fail-safe costs but do incur standard storage costs.

Example of SQL queries that extract and preprocess data:

Consider a raw table sales data as shown below:

sale_id	customer_name	sale_date	amount	region	status
101	Alice	2025-01-12	250.00	'North'	'PAID'
102	NULL	2025-01-15	-50.00	'Unknown'	'CANCEL'
103	Bob	2025-01-18	300.00	'East'	'PAID'
...

Suppose we want to extract valid sales, clean up bad data, and add a derived column for monthly grouping.

The SQL queries required to do the above are listed below:

1. For Extracting and Preprocessing the data:

```

WITH cleaned_sales AS (

    SELECT

        sale_id,

        COALESCE(customer_name, 'Unknown') AS customer_name,

        sale_date,

        amount,

        region,

        status,

        -- Extract month for grouping

        TO_CHAR(sale_date, 'YYYY-MM') AS sale_month

    FROM raw_sales_data

    WHERE

        amount > 0      -- Remove refunds or invalid sales

        AND status = 'PAID' -- Only keep paid orders

        AND region IS NOT NULL

        AND region != 'Unknown'

)

```

2. Store into a clean table:

```

CREATE OR REPLACE TABLE clean_sales_data AS

SELECT * FROM cleaned_sales;

```

3. Aggregate Query on clean data:

```

SELECT

    sale_month,

    region,

    COUNT(*) AS total_sales,

    SUM(amount) AS total_revenue

```

FROM clean_sales_data

GROUP BY sale_month, region

ORDER BY sale_month, region;

Integration of Snowflake with ML pipelines:

1. Data Ingestion

Snowflake supports loading large volumes of structured and semi-structured data:

- Batch via COPY INTO
- Streaming via Snowpipe
- APIs, Kafka, ETL tools like Fivetran, Talend, Airbyte

2. Feature Engineering and Preprocessing in SQL

We can use SQL to:

- Clean, normalize, and join data
- Aggregate features
- Generate time-based features

SQL-based preprocessing keeps data transformations close to the data source and reproducible.

3. Connect to ML Tools (Python, Spark, etc.)

Python: Snowpark + Pandas/Scikit-learn/XGBoost

- Use Snowpark for Python to write transformation logic in Python directly within Snowflake.
- Easily extract DataFrames into memory for training in Python.

Use the Snowflake Python Connector to pull training data into a notebook.

4. Model Deployment / Scoring

- Store model predictions back to Snowflake.
- Use UDFs in Snowflake to embed Python models and do inference inside SQL.

5. Integration with ML Platforms

- DataRobot, H2O.ai, AWS Sagemaker, Azure ML, Vertex AI can connect directly to Snowflake for feature and label data.
- Snowflake supports external functions to trigger remote APIs for scoring.

3. Feature Store Concepts:

A Feature Store is a centralized repository designed to store, manage, share, and serve features used in machine learning (ML) models. It provides a standardized way to:

- Ingest, clean, and transform data into features.
- Serve features in real-time or batch for training and inference.
- Track feature lineage, versioning, and consistency across environments.

It's needed for:

- Centralized feature definitions
- Unified offline & online serving.
- Reusable, documented transformations.
- Fast access to pre-computed features.
- Feature versioning & lineage.

Popular Feature Stores are:

1.) AWS Sage Maker Feature Store:

- It can work both online and offline.
- It is integrated with AWS ecosystem.
- It is useful real time and batch storage and it has the capability of low latency serving via dynamo DB.
- It can be made of better use for AWS Centric ML workflows.

2.) Snowflake Feature Store:

- It can work offline but limited online(via cache or API's).
- It is integrated with Python, SQL, Snowpark, MLops tools etc.
- It has strong data governance, SQL based features and easily scalable batch features.
- It is ideal for SQL heavy pipelines and regulated industries.

3.) Databricks Feature Store:

- It can work both offline and online.
- It is integrated with Spark and MLFlow.
- It includes feature lineage tracking, real time streaming support and auto logging to MLflow.
- It is of great usage for Spark/Delta Lake pipelines.

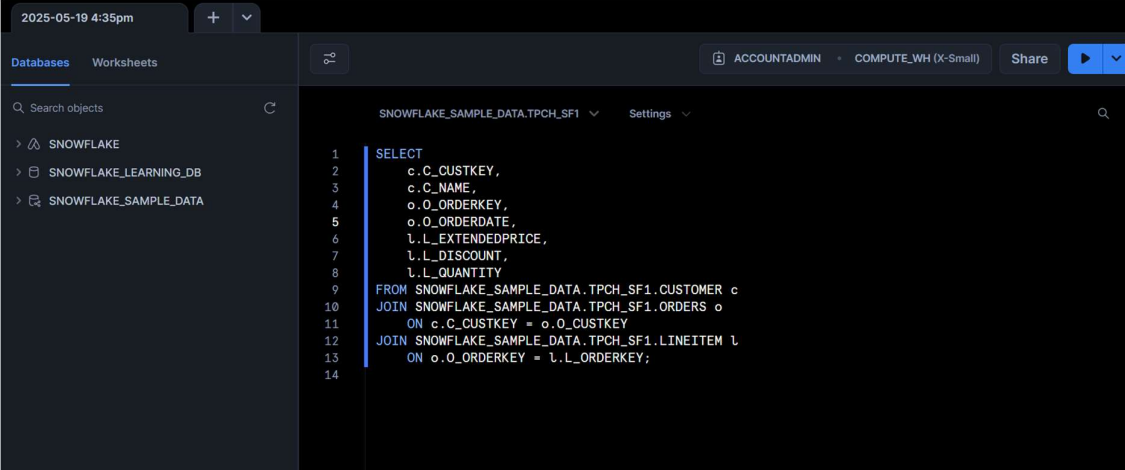
4.) Feast:

- It is an open source feature store available both online and offline.
- It is integrated with Cloud-Agnostic.
- It has simple interface and store agnostic backend system(Redis, BigQuery etc.)

- It is ideal for Cloud Neutral and customizable pipelines.

4. Implementing Feature Engineering with Snowflake & Feature Store

For selecting the features, I have used data available inbuilt in Snowflake from a table named SNOWFLAKE_SAMPLE_DATA.TPCH_SF1. It's basically a table based on TPC-H benchmark data. Following is the Code snippet,




```

1  SELECT
2    c.C_CUSTKEY,
3    c.C_NAME,
4    o.O_ORDERKEY,
5    o.O_ORDERDATE,
6    l.L_EXTENDEDPRICE,
7    l.L_DISCOUNT,
8    l.L_QUANTITY
9  FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER c
10 JOIN SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS o
11     ON c.C_CUSTKEY = o.O_CUSTKEY
12 JOIN SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.LINEITEM l
13     ON o.O_ORDERKEY = l.L_ORDERKEY;
14

```

The output is displayed as shown below,



	#_C_CUSTKEY	C_NAME	#_O_ORDERKEY	O_ORDERDATE
1	121361	Customer#000121361	1200001	1994-01-24
2	121361	Customer#000121361	1200001	1994-01-24
3	1775	Customer#000001775	1200002	1996-12-06
4	1775	Customer#000001775	1200002	1996-12-06
5	1775	Customer#000001775	1200002	1996-12-06
6	1775	Customer#000001775	1200002	1996-12-06
7	1775	Customer#000001775	1200002	1996-12-06

Now we can new generate customer level features such as,

- Total orders
- Average order value
- Total quantity ordered
- Average discount
- Recency(No of days from last order to present order)

This are aggregations. One-Hot Encoding can also be used if necessary where some of the column names are converted to binary for further processing.

Following is the code snippet based on above information.

```

1  SELECT
2      c.C_CUSTKEY,
3      c.C_NAME,
4      COUNT(DISTINCT o.O_ORDERKEY) AS num_orders,
5      SUM(l.L_EXTENDEDPRICE * (1 - l.L_DISCOUNT)) AS total_spent,
6      AVG(l.L_EXTENDEDPRICE * (1 - l.L_DISCOUNT)) AS avg_order_value,
7      SUM(l.L_QUANTITY) AS total_quantity,
8      AVG(l.L_DISCOUNT) AS avg_discount,
9      DATEDIFF('day', MAX(o.O_ORDERDATE), CURRENT_DATE) AS days_since_last_order
10 FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER c
11 JOIN SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS o
12     ON c.C_CUSTKEY = o.O_CUSTKEY
13 JOIN SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.LINEITEM l
14     ON o.O_ORDERKEY = l.L_ORDERKEY
15 GROUP BY c.C_CUSTKEY, c.C_NAME;
16
17

```

The output of the above is as shown below,

	NUM_ORDERS	TOTAL_SPENT	AVG_ORDER_VALUE	TOTAL_QUANTITY
1	8	1556350.0402	37055.9533380952	1070.00
2	18	3472384.1420	39458.9107045455	2330.00
3	19	3116672.4257	33156.0896351064	2219.00
4	23	3512341.0306	38177.6198978261	2450.00
5	21	3032989.9410	36542.0474819277	2130.00

Now these features can again be stored into the same table or new table in the same or different kinds of feature stores.

Here, we can use CREATE OR REPLACE Queries to update the table.

ML models can retrieve features in the store using Snowflake connector in python. Based on this, we can perform the required operations and update or modify the existing data.

5. Practical Task:

1. Connection of Snowflake to Google Colab(Python Editor):

It is done with the snowflake connector in python. The database it's connected to is SNOWFLAKE_SAMPLE_DATA and schema is TPCH_SF1. Below is the code snippet for this explanation.(We need to install Snow Flake library before making further operations).


```

import snowflake.connector
import pandas as pd

conn = snowflake.connector.connect(
    user='vaspon2005',
    password='Roarriserevolt2022',
    account='DXFYKFD-HJ95185',
    warehouse='COMPUTE_WH',
    database='SNOWFLAKE_SAMPLE_DATA',
    schema='TPCH_SF1'
)

```

2. Loading the Data from the database into a pandas data frame:

The data is loaded from the Snowflake using an SQL query and is converted to the pandas data frame. Below is the snippet of the code.

```

sql = """
SELECT
    C.C_CUSTKEY,
    C.C_NAME,
    C.C_MKTSEGMENT,
    COALESCE(COUNT(O.O_ORDERKEY), 0) AS NUM_ORDERS,
    COALESCE(AVG(O.O_TOTALPRICE), 0) AS AVG_ORDER_VALUE,
    MAX(O.O_ORDERDATE) AS LAST_ORDER_DATE,
    COALESCE(SUM(O.O_TOTALPRICE), 0) AS TOTAL_SPENT
FROM CUSTOMER C
LEFT JOIN ORDERS O ON C.C_CUSTKEY = O.O_CUSTKEY
GROUP BY C.C_CUSTKEY, C.C_NAME, C.C_MKTSEGMENT

"""

df = pd.read_sql(sql, conn)

```

3. Feature Engineering:

This is done by aggregation of certain columns such as Average of Order price, Max of Order date, sum of order price etc. and one hot encoding of the column MKTSEGMENT is done. Missing values such as NaN are replaced by 0 by the coalesce function in SQL and they are also replaced by Mean of the remaining data points. Below is the code snippet,

```

df_encoded = pd.get_dummies(df, columns=['C_MKTSEGMENT'], drop_first=True)

X = df_encoded.drop(columns=['C_CUSTKEY', 'C_NAME', 'LAST_ORDER_DATE', 'TOTAL_SPENT'])
y = df_encoded['TOTAL_SPENT']

```

4. Storing of features in Snowflake:

The newly created features are further stored in Snowflake by creating a new table. Following is the code snippet.

```
from snowflake.connector.pandas_tools import write_pandas

create_query = """
CREATE TABLE CUSTOMER_FEATURES (
  C_CUSTKEY INTEGER,
  C_NAME STRING,
  NUM_ORDERS INTEGER,
  TOTAL_SPENT FLOAT,
  AVG_ORDER_VALUE FLOAT,
  TOTAL_QUANTITY FLOAT,
  AVG_DISCOUNT FLOAT,
  LAST_ORDER_DATE DATE,
  DAYS_SINCE_LAST_ORDER INTEGER
);
"""
conn.cursor().execute(create_query)

# Uploading to Snowflake
success, nchunks, nrows, _ = write_pandas(conn, df_features, table_name='CUSTOMER_FEATURES')
print(f"Inserted {nrows} rows into Snowflake feature store.")
```

5. Retrieval of features:

These features are retrieved again from the Snowflake for feeding into the model. Below is the code snippet.

```
query_features = "SELECT * FROM CUSTOMER_FEATURES"
df_model = pd.read_sql(query_features, conn)

X = df_model.drop(columns=['C_CUSTKEY', 'C_NAME', 'LAST_ORDER_DATE'])

y = X['TOTAL_SPENT']

X = X.drop(columns=['TOTAL_SPENT'])
```

6. Feeding the features into the model:

These features can be fed into various models depending on our applications. Here for example, I have used a Multi Variable Linear Regression Model where TOTAL_SPENT is taken as the output variable and remaining non numeric columns are taken as the input variables. Below is the code snippet,

```

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression MSE:", round(mse, 2))
print("Linear Regression R2 Score:", round(r2, 2))

```

Linear Regression MSE: 66587586863.27
 Linear Regression R² Score: 0.96

CONCLUSIONS:

Therefore, feature extraction and its transformation is done successfully from Snow Flake and required feature engineering is done. The newly created features are stored in the Snow Flake therefore making it suitable for extraction and feeding them into the models whenever needed.