Feature Engineering with Snowflake and Feature Stores

# 1. Introduction to Feature Engineering

Feature engineering is the process of extracting useful features from raw data. It involves transforming raw data such that it can be used to train machine learning models and improve model performance.

Why is feature engineering important?

* Improves model performance- Well engineered features boost model performance.
* Reduces computational complexity- Computational complexity can be reduced by removing irrelevant features.
* Reduces noise- Feature engineering filters out noise and handles missing values.
* Improves model generalization (reduces overfitting).

Different types of feature engineering techniques include-

1. Normalization: This technique scales all the values to a [0,1] range. Normalization is used because different features often have different value ranges, which can bias the model's learning process. It scales the maximum value of a feature to 1 and minimum value to 0.

Formula = (x-min)/(max-min)

1. Encoding:

It is used to assign numerical values to categorical variables so that they can be used in model training.

Label Encoding- Converts categories to numbers. Ex. A->0, B->1, C->2.

One-hot encoding- Creates binary columns for each category.

Ex. A->[0 0 1], B->[0 1 0], C->[1 0 0]

1. Handling Missing Values:

Missing values can be replaced by mean/median(numeric) and mode (categorical).

1. Interaction Features:

Combines two or more features into one using multiplication, ratio or comparison.

1. Dimensionality Reduction:

Methods such as Principal Component Analysis (PCA) reduce the number of features while retaining variance. It improves visualization and reduces overfitting in higher dimensional data.

1. Time based Aggregation: Time-based aggregation is a feature engineering technique where we group and summarize data over specific time windows, such as daily, weekly, monthly, or hourly.

# 2. Using Snowflake for Data Storage & Processing

Snowflake is a cloud-based data platform that supports structured and semi-structured data.

* Structured Data: It is highly organized and fits into tables with defined columns and data types. It is stored in relational tables using SQL based schemas. (eg. INT, VARCHAR, DATE).

Example:

CREATE TABLE sales (

order\_id INT,

customer\_id STRING,

order\_amount FLOAT,

order\_date DATE

);

* Semi structured Data: Semi structured data doesn’t follow a strict tabular format. Snowflake provides support for semi structured data formats like json, xml, etc. It can be stored in the VARIANT data type.

Example:

CREATE TABLE user\_profiles (

id INTEGER,

profile\_data VARIANT

);

INSERT INTO user\_profiles

VALUES (1, PARSE\_JSON('{"name": "John", "age": 30, "address": {"city": "NYC"}}'));

Data can be extracted using-

SELECT

profile\_data:name::STRING AS user\_name,

profile\_data:age::INTEGER AS user\_age,

profile\_data:address.city::STRING AS city

FROM user\_profiles;

# Example of SQL queries:

Let there be a table named transactions-

| **customer\_id** | **transaction\_id** | **amount** | **category** | **transaction\_date** |
| --- | --- | --- | --- | --- |
| A123 | T001 | 100.00 | Grocery | 2024-05-01 |
| A123 | T002 | 250.00 | Grocery | 2024-05-03 |
| B456 | T003 | 500.00 | Apparel | 2024-05-01 |

**Step 1:** Extract data-

SELECT

customer\_id,

amount,

category,

transaction\_date

FROM transactions;

**Step 2:** Aggregation-

SELECT

customer\_id,

COUNT(\*) AS total\_transactions,

SUM(amount) AS total\_spent,

AVG(amount) AS avg\_transaction\_value

FROM transactions

GROUP BY customer\_id;

**Step 3:** Compute daily spent per customer

SELECT

customer\_id,

DATE(transaction\_date) AS transaction\_day,

SUM(amount) AS daily\_spend

FROM transactions

GROUP BY customer\_id, DATE(transaction\_date);

**Step 4:** Filter high value customers(those who have spent more than 500)-

SELECT

customer\_id,

SUM(amount) AS total\_spent

FROM transactions

GROUP BY customer\_id

HAVING SUM(amount) > 500;

Snowflake integration with ML pipelines-

* Python integration via connectors: We can connect snowflake to python using

snowflake-connector-python.

Example: Pulling data into pandas dataframe-

import snowflake.connector

import pandas as pd

conn = snowflake.connector.connect( ... )

df = pd.read\_sql("SELECT \* FROM features\_table", conn)

* SQL based feature engineering: Data cleaning, aggregation, encoding and time based transformations can be performed in Snowflake using SQL.
* Integration with ML libraries: Snowflake integrates with python libraries like scikit-learn, XGBoost, TensorFlow etc.
* Feature Stores: Features can be stored in Snowflake tables (acting as feature store).

# 3. Feature Store Concepts

A feature store is a centralized data management system built to store and manage features. It acts as a bridge between ML models ensuring the same features that are used during training are also used during inference.

Feature store is needed because-

* Consistency between training and inference: Without feature store, feature logic is duplicated at training and inference. This leads to mismatches. Thus, it ensures that the model sees the same input format during both training and inference.
* Feature reusability across models: Once defined, features can be used for other ML model s and projects as well.
* Speed and Scalability: Feature stores provide fast lookups for real time inference. Feature stores are optimized for large scale joins.

Difference between different feature stores-

1. Integration with ML Ecosystem

* AWS SageMaker: Deeply integrated into the SageMaker pipeline — supports ingestion, training, deployment, and monitoring within the same AWS ecosystem.
* Snowflake: Integrates well with external ML tools (e.g., scikit-learn, XGBoost) via Snowpark or Python connectors.
* Databricks: Strong integration with MLflow, Delta Lake, and notebooks, enabling experiment tracking, feature reuse, and model deployment in one environment.

2. Feature Engineering Approach

* AWS SageMaker: Feature engineering is typically done in Python notebooks or pipelines, outside the store itself; logic must be written separately for training and inference.
* Snowflake: Encourages SQL-based feature engineering directly on data warehouse tables, ensuring training and inference logic stays unified.
* Databricks: Feature engineering happens using PySpark or SQL, often as part of a DataFrame transformation pipeline; supports registering these as reusable features.

3. Online vs Offline Store

* AWS SageMaker: Provides separate online (DynamoDB) and offline (S3) stores; you must manage syncing between them.
* Snowflake: Uses a single source — the same Snowflake tables serve both training and inference (no duplication).
* Databricks: Primarily designed for batch/offline usage via Delta Lake; online serving is possible through integration but not native.

4. Data Type Support

* AWS SageMaker Feature Store:  
  Supports standard tabular types such as string, float, int, boolean, and timestamp.  
  It does not natively support complex types like arrays, maps, or nested structures — you’d need to flatten them manually or use preprocessing.
* Snowflake Feature Store:  
  Supports both structured (string, number, boolean, date/time) and semi-structured data like VARIANT, ARRAY, OBJECT — meaning you can store nested JSON and complex types without flattening.  
  This is a strong differentiator for mixed-format enterprise data.
* Databricks Feature Store:  
  Built on top of Delta Lake and Spark, it supports a wide range of types, including primitive types (int, float, string, etc.) and Complex types (array, map, struct).

5. Data Access and Querying

* AWS SageMaker: Access is primarily through Python SDK; not meant for direct querying via SQL.
* Snowflake: Full SQL support — all features are stored in warehouse tables and accessible via standard SQL.
* Databricks: Uses Spark SQL or PySpark for data access; tables are queried through the Delta Lake framework.

6. Ingestion

* AWS SageMaker: Requires explicit schema definition and ingestion via SDK or pipelines. Supports batch and streaming but setup is complex.
* Snowflake: Ingests easily via SQL or Python (write\_pandas()), directly from existing tables or external sources. No extra infra needed.
* Databricks: Ingests from Spark DataFrames or Delta tables, supports batch and streaming. Requires familiarity with Spark.

# 4. Implementing Feature Engineering with Snowflake & Feature Store

1. Extract:

Use Snowflake SQL or Python to fetch raw data from existing tables in a database or schema.

Ex. df = pd.read\_sql("SELECT \* FROM LOAN\_RAW\_DATA;", conn)

1. Transform:

Perform feature engineering using Python or SQL. This includes:

* Handling missing values
* Creating new features (e.g., Total\_Income, EMI)
* Encoding categorical variables
* Normalizing or binning features

This ensures the data is clean and model-ready.

1. Load into Feature Store:

Store the engineered features as a new table in Snowflake, which serves as the Feature Store.  
Example using Python:

from snowflake.connector.pandas\_tools import write\_pandas

write\_pandas(conn, df\_features, table\_name='LOAN\_FEATURE\_STORE', auto\_create\_table=True, overwrite=True)

This table acts as a centralized, reusable feature repository for training and inference.

1. Access for ML:

ML models retrieve features directly from the Feature Store using SQL or Python queries.  
Example:

df\_features = pd.read\_sql("SELECT \* FROM LOAN\_FEATURE\_STORE;", conn)

These features can be fed into your ML pipeline using libraries like scikit-learn, XGBoost, etc., ensuring consistency between training and inference.