Q&A Section (Assignment Questions)

Q1: Define feature engineering and explain its importance in machine learning. A: Feature engineering is the process of creating, transforming, and selecting input variables (features) to improve a model's predictive performance. It bridges raw data and machine learning algorithms by:

- Extracting meaningful signals (e.g., aggregating transaction data to compute spending patterns).
- Enhancing model accuracy and convergence speed.
- Reducing overfitting by eliminating noisy or irrelevant features.
- Enabling interpretable insights for business stakeholders.

Q2: Describe different types of feature engineering techniques (e.g., normalization, encoding, time-based aggregations). A: Common techniques include:

- Encoding: Converting categorical variables into numeric form, e.g., one-hot encoding for multi-class features or binary encoding for two-value features.
- Normalization/Standardization: Scaling numeric features, such as Min–Max normalization (to [0,1]) or Z-score standardization (mean=0, std=1).
- Aggregation: Summarizing data across time or groups, e.g., computing monthly averages, rolling windows, or cumulative sums.
- Interaction Features: Creating features that capture interactions between variables, e.g., ratio of tenure to total charges or contract×payment method flags.
- Temporal Features: Extracting date/time components or trend indicators, e.g., time since last purchase or churn velocity.

Q3: Explain how Snowflake is used for storing structured and semi-structured data.

A: Snowflake natively stores structured data in tables and handles semi-structured data (JSON, Avro, Parquet) using VARIANT columns. It automatically parses, optimizes, and enables querying via SQL functions (e.g., SELECT data:key FROM table). Snowflake's micro-partitioning and columnar storage ensure efficient storage and fast query performance across both data types.

Q4: Provide an example of SQL queries that extract and preprocess data in Snowflake.

```sql
SELECT
  CustomerID,
  Gender,
  CASE WHEN SeniorCitizen = 1 THEN 'Yes' ELSE 'No' END AS Senior,
  MonthlyCharges,
  TotalCharges,
  tenure,
  TotalCharges / NULLIF(tenure, 0) AS AvgMonthlyCharge
FROM telco_churn
WHERE TotalCharges IS NOT NULL;
```

Q5: Discuss how Snowflake integrates with ML pipelines. A: Snowflake integrates via:

- Snowflake Connector for Python: Query data directly into Pandas or Snowpark DataFrames.
- Snowpark (Java/Python): Run user-defined functions and dataframes inside Snowflake compute.
- External Functions & Streams: Trigger pipelines via Snowflake Tasks and Streams into orchestration tools (Airflow).
- Native integration with ML platforms (DataRobot, AWS Sagemaker) and BI tools.

Q6: What is a Feature Store, and why is it needed? A: A Feature Store centralizes feature definitions, storage, and serving for ML. It ensures:

- Consistency between offline (training) and online (serving) features.
- Reusability of features across multiple models and teams.
- Governance & lineage tracking of feature computations and versions.

Q7: Compare different feature stores (AWS SageMaker Feature Store, Snowflake Feature Store, Databricks Feature Store).

Feature Store	Storage Layer	Key Strengths
AWS SageMaker FS	DynamoDB (online), S3 (offline)	Tight integration with SageMaker; auto-scaling online store
Snowflake Feature Store	Snowflake tables	Leverages Snowflake's performance, SQL access, and unified storage
Databricks Feature Store	Delta Lake (offline), Photon (online)	Built into Databricks Lakehouse, supports feature lineage and RBAC

Snowflake FS excels for SQL-centric teams already on Snowflake; AWS FS is ideal if your entire pipeline runs in SageMaker; Databricks FS best suits Lakehouse architectures.

Work Explanation

1. Introduction

The objective of this assignment is to demonstrate an end-to-end feature engineering and machine learning pipeline using Snowflake. As ML systems mature, maintaining consistency between training and inference becomes challenging. Feature stores help by centralizing feature computation and providing a reliable source for models, reducing training/serving skew and improving governance. The pipeline includes:

- Extracting and preprocessing raw data using SQL in Snowflake.
- Performing feature engineering including encoding, normalization, and derived features.
- Loading processed features into a Feature Store for reuse.
- Training a machine learning model using Python with data from Snowflake.

---

2. Dataset Overview

2.1 Telco Customer Churn Dataset

- Source: Kaggle
- Rows: 7,043
- Columns: 21 (describe a subset)

2.2 Target Variable

- **Churn**: Yes/No (binary classification)

2.3 Key Features

- Demographics: gender, SeniorCitizen, Partner, Dependents
- Account: tenure, Contract, PaymentMethod
- Services: InternetService, StreamingTV, etc.
- Charges: MonthlyCharges, TotalCharges

---

3. Snowflake Setup

Snowflake is a cloud-native data warehouse designed for scalable analytics. It supports ANSI SQL and handles structured/semi-structured data efficiently. Key benefits include auto-scaling compute, data sharing across environments, and native support for integrations with machine learning pipelines via Snowpark and connectors.

3.1 Warehouse, Database, Schema

```
CREATE OR REPLACE WAREHOUSE compute_wh WITH WAREHOUSE_SIZE =
'XSMALL';
CREATE OR REPLACE DATABASE churn_db;
USE DATABASE churn_db;
CREATE OR REPLACE SCHEMA telco_schema;
USE SCHEMA telco_schema;
```

3.2 Table Creation

```
CREATE OR REPLACE TABLE telco_churn_raw (
    customerID STRING,
    gender STRING,
    SeniorCitizen INT,
    Partner STRING,
    Dependents STRING,
    tenure INT,
    PhoneService STRING,
    MultipleLines STRING,
    InternetService STRING,
    OnlineSecurity STRING,
    OnlineBackup STRING,
    DeviceProtection STRING,
    TechSupport STRING,
    StreamingTV STRING,
    StreamingMovies STRING,
    Contract STRING,
    PaperlessBilling STRING,
    PaymentMethod STRING,
    MonthlyCharges FLOAT,
    TotalCharges STRING,
    Churn STRING
);
```

---

4. Data Ingestion & Cleaning

Data cleaning ensures that inconsistencies and missing values do not negatively impact model accuracy. For instance, converting columns to correct data types, handling nulls, and standardizing formats are essential to prepare the data for reliable feature engineering and modeling.

4.1 Loading the CSV

- Uploaded WA_Fn-UseC_-Telco-Customer-Churn.csv via Snowflake UI / SnowSQL.

4.2 Type Conversion & Imputation

```
CREATE OR REPLACE TABLE telco_churn_features AS
SELECT
    *,
    TRY_CAST(TotalCharges AS FLOAT) AS TotalCharges_Cleaned
FROM telco_churn_raw;
```

```
--imputation
CREATE OR REPLACE TABLE telco_churn_cleaned AS
SELECT *
FROM telco_churn_features
WHERE TotalCharges IS NOT NULL;
```

Filtering out rows with missing TotalCharges (11 rows dropped).

---

5. Encoding & Normalization

Most ML models require numerical inputs. Encoding transforms categorical data into numeric form, allowing algorithms to process them. Normalization ensures all features contribute proportionally, preventing dominance by features with larger ranges. This is especially critical in algorithms sensitive to feature scale like KNN or SVM.

```
SELECT DISTINCT gender FROM telco_churn_cleaned;
SELECT DISTINCT Partner FROM telco_churn_cleaned;
SELECT DISTINCT Dependents FROM telco_churn_cleaned;
SELECT DISTINCT PhoneService FROM telco_churn_cleaned;
SELECT DISTINCT MultipleLines FROM telco_churn_cleaned;
SELECT DISTINCT InternetService FROM telco_churn_cleaned;
SELECT DISTINCT OnlineSecurity FROM telco_churn_cleaned;
SELECT DISTINCT OnlineBackup FROM telco_churn_cleaned;
SELECT DISTINCT DeviceProtection FROM telco_churn_cleaned;
SELECT DISTINCT TechSupport FROM telco_churn_cleaned;
SELECT DISTINCT StreamingTV FROM telco_churn_cleaned;
SELECT DISTINCT StreamingMovies FROM telco_churn_cleaned;
SELECT DISTINCT Contract FROM telco_churn_cleaned;
SELECT DISTINCT PaperlessBilling FROM telco_churn_cleaned;
SELECT DISTINCT PaymentMethod FROM telco_churn_cleaned;
SELECT DISTINCT Churn FROM telco_churn_cleaned;
```

For getting the no. of unique values/categories in columns

5.1 Categorical Encoding

- Binary encoding for 2-value columns:

```
CREATE OR REPLACE TABLE telco_churn_binary AS
SELECT
  *,
  /* gender: Male/Female */
  CASE WHEN gender = 'Male' THEN 1 ELSE 0 END          AS
gender_enc,
  /* Partner: Yes/No */
  CASE WHEN Partner = 'Yes' THEN 1 ELSE 0 END          AS
partner_enc,
  /* Dependents: Yes/No */
  CASE WHEN Dependents = 'Yes' THEN 1 ELSE 0 END       AS
dependents_enc,
  /* PhoneService: Yes/No */
  CASE WHEN PhoneService = 'Yes' THEN 1 ELSE 0 END     AS
phoneservice_enc,
  /* PaperlessBilling: Yes/No */
  CASE WHEN PaperlessBilling = 'Yes' THEN 1 ELSE 0 END  AS
paperless_enc,
  /* Churn (target): Yes/No */
  CASE WHEN Churn = 'Yes' THEN 1 ELSE 0 END            AS churn_enc
FROM telco_churn_cleaned;
```

- One-hot encoding for multi-class columns:

```
CREATE OR REPLACE TABLE telco_churn_encoded AS
SELECT
  *,
  /* MultipleLines: No phone service / No / Yes */
  CASE WHEN MultipleLines = 'No phone service' THEN 1 ELSE 0 END AS ml_no_phone,
  CASE WHEN MultipleLines = 'No'               THEN 1 ELSE 0 END AS ml_no,
  CASE WHEN MultipleLines = 'Yes'              THEN 1 ELSE 0 END AS ml_yes,

  /* InternetService: DSL / Fiber optic / No */
  CASE WHEN InternetService = 'DSL'          THEN 1 ELSE 0 END AS is_dsl,
  CASE WHEN InternetService = 'Fiber optic'  THEN 1 ELSE 0 END AS is_fiber,
  CASE WHEN InternetService = 'No'           THEN 1 ELSE 0 END AS is_none,

  /* OnlineSecurity: No internet service / No / Yes */
  CASE WHEN OnlineSecurity = 'No internet service' THEN 1 ELSE 0 END AS os_no_service,
  CASE WHEN OnlineSecurity = 'No'                  THEN 1 ELSE 0 END AS os_no,
  CASE WHEN OnlineSecurity = 'Yes'                 THEN 1 ELSE 0 END AS os_yes,

  /* OnlineBackup: No internet service / No / Yes */
  CASE WHEN OnlineBackup = 'No internet service' THEN 1 ELSE 0 END AS ob_no_service,
  CASE WHEN OnlineBackup = 'No'                  THEN 1 ELSE 0 END AS ob_no,
  CASE WHEN OnlineBackup = 'Yes'                 THEN 1 ELSE 0 END AS ob_yes,
```

```sql
/* DeviceProtection: No internet service / No / Yes */
CASE WHEN DeviceProtection = 'No internet service' THEN 1 ELSE 0 END AS dp_no_service,
CASE WHEN DeviceProtection = 'No'                  THEN 1 ELSE 0 END AS dp_no,
CASE WHEN DeviceProtection = 'Yes'                 THEN 1 ELSE 0 END AS dp_yes,

/* TechSupport: No internet service / No / Yes */
CASE WHEN TechSupport = 'No internet service' THEN 1 ELSE 0 END AS ts_no_service,
CASE WHEN TechSupport = 'No'                  THEN 1 ELSE 0 END AS ts_no,
CASE WHEN TechSupport = 'Yes'                 THEN 1 ELSE 0 END AS ts_yes,

/* StreamingTV: No internet service / No / Yes */
CASE WHEN StreamingTV = 'No internet service' THEN 1 ELSE 0 END AS stv_no_service,
CASE WHEN StreamingTV = 'No'                  THEN 1 ELSE 0 END AS stv_no,
CASE WHEN StreamingTV = 'Yes'                 THEN 1 ELSE 0 END AS stv_yes,

/* StreamingMovies: No internet service / No / Yes */
CASE WHEN StreamingMovies = 'No internet service' THEN 1 ELSE 0 END AS sm_no_service,
CASE WHEN StreamingMovies = 'No'                  THEN 1 ELSE 0 END AS sm_no,
CASE WHEN StreamingMovies = 'Yes'                 THEN 1 ELSE 0 END AS sm_yes,

/* Contract: Month-to-month / One year / Two year */
CASE WHEN Contract = 'Month-to-month' THEN 1 ELSE 0 END AS contract_month_to_month,
CASE WHEN Contract = 'One year'       THEN 1 ELSE 0 END AS contract_one_year,
CASE WHEN Contract = 'Two year'       THEN 1 ELSE 0 END AS contract_two_year,
```

```sql
 /* PaymentMethod: Electronic check / Mailed check / Bank transfer / Credit card */
 CASE WHEN PaymentMethod = 'Electronic check'          THEN 1 ELSE 0 END AS pm_echeck,
 CASE WHEN PaymentMethod = 'Mailed check'              THEN 1 ELSE 0 END AS pm_mailed,
 CASE WHEN PaymentMethod = 'Bank transfer (automatic)' THEN 1 ELSE 0 END AS pm_bank_transfer,
 CASE WHEN PaymentMethod = 'Credit card (automatic)'   THEN 1 ELSE 0 END AS pm_credit_card

FROM telco_churn_binary;

select * from telco_churn_encoded;
```

```sql
--Feature Selection
CREATE OR REPLACE TABLE telco_churn_features_selected AS
SELECT
  -- Binary Encoded Features
  gender_enc,
  partner_enc,
  dependents_enc,
  phoneservice_enc,
  paperless_enc,

  -- One-Hot Encoded Features (excluding one category per feature)
  ml_no,ml_yes,is_dsl,is_fiber,os_no,os_yes,ob_no,ob_yes,dp_no,dp_yes,ts_no,
  ts_yes,stv_no,stv_yes,sm_no,sm_yes,contract_one_year,contract_two_year,
  pm_mailed,pm_bank_transfer,pm_credit_card,

  -- Numerical Features
  tenure,
  MonthlyCharges,
  TotalCharges,

  -- Target Variable
  churn_enc
FROM telco_churn_encoded;
```

5.2 Z-Score Normalization

```
CREATE OR REPLACE TABLE telco_churn_zscore_normalized AS
SELECT
  *,
  -- Z-score normalization for 'tenure'
  (tenure - AVG(tenure) OVER ()) / NULLIF(STDDEV(tenure) OVER (), 0)
AS tenure_zscore,

  -- Z-score normalization for 'MonthlyCharges'
  (MonthlyCharges - AVG(MonthlyCharges) OVER ()) /
NULLIF(STDDEV(MonthlyCharges) OVER (), 0) AS MonthlyCharges_zscore,

  -- Z-score normalization for 'TotalCharges'
  (TotalCharges - AVG(TotalCharges) OVER ()) /
NULLIF(STDDEV(TotalCharges) OVER (), 0) AS TotalCharges_zscore
FROM telco_churn_features_selected;
```

---

6. Feature Construction

5 derived features:

Derived features help extract deeper insights from existing raw data by capturing patterns or segmenting users into meaningful groups. The following features were engineered to enhance model performance:

- Average Monthly Charge (TotalCharges / tenure): Normalizes the total spending over the customer's lifecycle, giving a consistent view of monthly spending patterns, which can indicate customer value or engagement.
- Customer Lifetime Value Estimate (tenure * MonthlyCharges): Projects how much revenue a customer contributes, useful in identifying high-value customers more likely to churn.
- Tenure Buckets: Groups customers into segments (e.g., new, mid-term, loyal), helping the model learn patterns based on customer lifecycle stages.
- High Monthly Charge Flag (MonthlyCharges > 80): A binary indicator for expensive service plans, which may correlate with higher expectations or churn risk.
- Long Tenure Flag (tenure > 60): Identifies loyal customers. These users may have lower churn risk or may respond differently to retention strategies.

```sql
CREATE OR REPLACE TABLE telco_churn_feature_constructed AS
SELECT
  *,

  -- Avg Monthly Charge (safe only if tenure != 0)
  CASE
    WHEN tenure > 0 THEN TotalCharges / tenure
    ELSE NULL
  END AS avg_monthly_charge,

  -- Customer Lifetime Value (approx)
  tenure * MonthlyCharges AS clv_estimate,

  -- Tenure buckets
  CASE
    WHEN tenure <= 12 THEN '0-12'
    WHEN tenure <= 24 THEN '13-24'
    WHEN tenure <= 48 THEN '25-48'
    WHEN tenure <= 60 THEN '49-60'
    ELSE '60+'
  END AS tenure_bucket,
```

```sql
  -- High Monthly Charge Flag
  CASE
    WHEN MonthlyCharges > 80 THEN 1 ELSE 0
  END AS high_monthly_charge,

  -- Long Tenure Flag
  CASE
    WHEN tenure > 60 THEN 1 ELSE 0
  END AS has_long_tenure

FROM telco_churn_zscore_normalized;

CREATE OR REPLACE TABLE telco_churn_feature_constructed_ohe AS
SELECT
  *,
  CASE WHEN tenure_bucket = '0-12' THEN 1 ELSE 0 END AS bucket_0_12,
  CASE WHEN tenure_bucket = '13-24' THEN 1 ELSE 0 END AS bucket_13_24,
  CASE WHEN tenure_bucket = '25-48' THEN 1 ELSE 0 END AS bucket_25_48,
  CASE WHEN tenure_bucket = '49-60' THEN 1 ELSE 0 END AS bucket_49_60
  -- NOTE: Drop '60+' to avoid dummy variable trap (or vice versa)
FROM telco_churn_feature_constructed;
```

```sql
CREATE OR REPLACE TABLE telco_churn_final_features AS
SELECT
  -- Binary Encoded Features
  gender_enc,partner_enc,dependents_enc,
  phoneservice_enc,paperless_enc,
  -- One-Hot Encoded Features (categoricals)
  ml_no,ml_yes,is_dsl,is_fiber,os_no,os_yes,ob_no,ob_yes,dp_no,dp_yes,
  ts_no,ts_yes,stv_no,stv_yes,sm_no,sm_yes,ontract_one_year,contract_two_year
  pm_mailed,pm_bank_transfer,pm_credit_card,
  -- Constructed One-Hot for Tenure Buckets
  bucket_0_12,bucket_13_24,bucket_25_48,bucket_49_60,
  -- Constructed Numeric Features
  avg_monthly_charge,clv_estimate,high_monthly_charge,has_long_tenure,
  -- Z-score Normalized Columns
  tenure_zscore,MonthlyCharges_zscore,TotalCharges_zscore,
  -- Target Variab
  churn_enc

FROM telco_churn_feature_constructed_ohe;
```

---

7. Feature Store Registration

The feature store registers metadata, computes features, and versions them. This allows data scientists to reuse and track features reliably. For example, in Snowflake, registering engineered features in a dedicated schema allows easy access through SQL, maintaining traceability and performance.

7.1 Create Feature Store Schema

```
CREATE OR REPLACE DATABASE feature_store_db;
USE DATABASE feature_store_db;
CREATE OR REPLACE SCHEMA feature_store;
```

7.2 Persist Final Features

```
CREATE OR REPLACE TABLE feature_store.telco_customer_churn_features
AS
  SELECT * FROM CHURN_DB.TELCO_SCHEMA.telco_churn_final_features;
```

---

8. Model Training (Python)

The trained model benefits from the structured, engineered features. Connecting to Snowflake allows dynamic fetching of fresh data, enabling near real-time or batch predictions. Feature stores ensure that the same logic is used both during training and in production inference, avoiding feature drift.

8.1 Connecting to Snowflake

```
import pandas as pd
import snowflake.connector
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Snowflake connection config
conn = snowflake.connector.connect(
account = "PWOMYGF-UI19644",
user = "JATIN1708",
warehouse = "COMPUTE_WH",
database  = "FEATURE_STORE_DB"  ,   # or "CHURN_DB" if you kept it there
schema    = "FEATURE_STORE"  ,      # or "TELCO_SCHEMA"
password="*"
)

# Query final table
query = "SELECT * FROM telco_customer_churn_features"
df = pd.read_sql(query, conn)
conn.close()

# Preview
print("Data shape:", df.shape)
print("Columns:", df.columns.tolist())
```

```
# Split features & target
X = df.drop(columns=['CHURN_ENC'])
y = df['CHURN_ENC']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

8.2 Random Forest

```
# Model
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(class_weight='balanced', random_state=42)
clf.fit(X_train, y_train)

# Predict & evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```