### ****Candidate Name****: Nachiketbhai Prajapati

**GMS Data Engineering Challenge**

### ****Introduction****

This document provides a detailed list of all data cleaning, transformation and preparation steps/techniques I performed. And briefly explained my reason and logic for each step. It includes details about the data preparation, cleaning, and processing steps performed in Azure Synapse Analytics and Azure Databricks, as well as how each objective was achieved.

## ****A Detailed List of All Data Cleaning, Transformation, and Preparation Steps****

### ****1. Understanding the Data****

* **Reason/Logic**: Before performing any cleaning, it was critical to understand the structure, quality, and issues in the data.
* **Steps Taken**:
  + Reviewed the schema of all datasets.
  + Identified issues such as inconsistent date formats, null values, and duplicate records.
  + Categorized the data issues for targeted cleaning and preparation.

### ****2. Data Preparation in Azure Synapse Analytics****

#### ****Merging Multiple Files****

* **Reason/Logic**: To handle fragmented data from multiple files, merging was required for seamless processing.
* **Steps Taken**:
  + Used **Copy Activity** to merge:
    - 200 claim details files into one CSV (claim\_details.csv).
    - 3 claim master files into one CSV (claim\_masters.csv).

#### ****Data Cleaning Using Dataflows (Pipeline)****

1. **Source Data Load**
   * **Reason/Logic**: To load and process raw data.
   * **Steps Taken**: Imported all four datasets (claim\_details, claim\_masters, claims\_payment, policy\_premium) into the dataflow.
2. **Assert Activity**
   * **Reason/Logic**: To ensure data quality by checking uniqueness and null values.
   * **Steps Taken**:
     + Checked uniqueness of CustomerID, ClaimDate, PolicyStartDate, and PolicyEndDate.
     + Validated that ClaimID, PolicyID, and CustomerID are non-null.
3. **Derived Columns**
   * **Reason/Logic**: To add flags for duplicate and null rows.
   * **Steps Taken**:
     + Added column DuplicateRows: expression builder = "iif(hasError('assert1'), 'Y', 'N')"
     + Added column NullRows: expression builder = "iif(isNull(ClaimID) || isNull(PolicyID), 'Y', 'N')"
4. **Conditional Split**
   * **Reason/Logic**: To segregate good and bad records.
   * **Steps Taken**:
     + Created streams for:
       - **Good Records**: "Rows that do not meet any condition will use this output stream".
       - **Bad Records**: expression builder = "DuplicateRows == 'Y' || NullRows == 'Y'".
5. **Window Activity**
   * **Reason/Logic**: To assign row numbers to duplicate records.
   * **Steps Taken**:
     + 1. Over:- take from conditional split’s badrecords’ column DuplicateRows,
       2. Sort:- By “CustomerID” in ascending order,
       3. Range by: unbounded
       4. Assigned RowNumbers using expression builder = “toString(rowNumber())”.
6. **Conditional Split**
   * **Reason/Logic**: To segregate correct and wrong records.
   * **Steps Taken**:
     + Created streams for:
       - **correctRecords**: " Duplicaterows=='Y' && RowNumbers==('Y')".
       - **wrongRecords**: expression builder = "Rows that do not meet any condition will use this output stream".
7. **Union** 
   * **Reason/Logic**: To combine good records
   * **Steps Taken**:
     + Merged from **Good Records** of step 4 and **correctRecords** from step 5 using **Union activity**.
8. **Sinks**
   * **Reason/Logic**: To export cleaned data.
   * **Steps Taken**:
     + Exported data to:
       - **SQL Database (SSMS)**.
       - **ADLS Gen2** (as CSV files).

### ****3. Advanced Data Cleaning in Azure Databricks****

#### ****Connecting to Data Lake****

* **Reason/Logic**: To access data from ADLS Gen2 for further cleaning.
* **Steps Taken**:
  + Set up Key Vault in Azure portal and created Scope in Databricks.
  + Connected Databricks to ADLS Gen2 storage using secrets from key-vault and Scope.

#### ****Standardizing Date Formats****

* **Reason/Logic**: To ensure consistency in date representation for accurate analysis.
* **Steps Taken**:
  + Consolidated multiple date formats (e.g., MM/dd/yyyy, dd/MM/yyyy, yyyy-MM-dd) into yyyy-MM-dd.) and converted into one yyyy-mm-dd format.

#### ****Standardizing PaymentStatus column****

* **Reason/Logic**: To unify inconsistent values in the PaymentStatus column.
* **Steps Taken**:
  + Used substring-based logic to consolidate:
    - Variants like re, rej, req into Rejected.
    - Variants like pe, pen into Pending.
    - Variants like pa, pad into Paid.

### ****4. Transformation and Preparation for Objectives****

#### ****Objective 1: Fraud Detection****

* + - * **Objective**:Detect claims with a high probability of being fraudulent by

evaluating claim patterns, payment status, and other attributes.

1. **Key Metrics and Conditions**:
   * **High Claim Amount**: Flags claims where ClaimAmount is more than 5 times the PremiumAmount.
   * **Rejected Payment**: Flags claims with a PaymentStatus of Rejected.
   * **Frequent Claims**: Identifies claims filed within 30 days of a previous claim by the same customer. Uses LAG function to compare ServiceDate of consecutive claims.
   * **Risky Region**: Flags claims originating from specified "High Risk" regions.
   * **Normal**: Default category for claims not meeting any fraudulent criteria.
2. **Joins Used**:
   * **LEFT JOIN** to combine data from:
   * ClaimsDetails (cd) for ServiceDate.
   * ClaimsPayment (cp) for PaymentStatus.
   * PolicyPremium (pp) for PremiumAmount.
3. **Window Function**:
   * **LAG**: Compares the ServiceDate of the current claim with the previous claim for the same customer to identify claims filed within a short interval (e.g., 30 days).
4. **Fraud Classification**:
   * Uses a CASE statement to assign a FraudReason based on the defined conditions:
   * High Claim Amount, Rejected Payment, Frequent Claims, Risky Region, or Normal.
5. **Columns in Output**:
   * **ClaimID**: Unique identifier for the claim.
   * **CustomerID**: Unique identifier for the customer.
   * **ClaimDate**: The date the claim was filed.
   * **ClaimAmount**: The monetary amount of the claim.
   * **PremiumAmount**: The premium associated with the policy.
   * **PaymentStatus**: Status of the payment.
   * **Region**: Geographic region of the claim.
   * **FraudReason**: Classification of the fraud reason.
6. **Reason**:
   * Identify high-risk claims for further review.
   * Provide insights into potential fraud patterns based on customer behavior, payment status, and regional risk.

#### ****Objective 2: Customer Retention Analysis****

* **Logic**:

1. **Objective**: Identify customers at risk of churning based on their claims and

payment history.

1. **Key Metrics and Columns**:
   * **LastClaimDate**: The most recent claim date for each customer, derived using MAX(cd.ServiceDate).
   * **PaymentStatus**: The latest payment status for the customer.
2. **Customer Retention Classification**:
   * **RetentionStatus**:
   * **At Risk**:
   * If the customer’s last claim date (LastClaimDate) is more than 180 days old (< DATE\_ADD(CURRENT\_DATE(), -180)).
   * If the PaymentStatus is 'Rejected'.
   * **Engaged**: If neither condition for "At Risk" applies.
3. **Joins Used**:
   * **LEFT JOIN**:
   * **PolicyPremium (pp)** to include all customers.
   * **ClaimsMaster (cm)** for claim-related data by CustomerID.
   * **ClaimsDetails (cd)** for the claim's ServiceDate.
   * **ClaimsPayment (cp)** for the claim's payment status.
4. **Grouping**:
   * Grouped by **CustomerID** and **PaymentStatus** to aggregate data and classify customers.
5. **Columns in Output**:
   * **CustomerID**: Unique identifier for the customer.
   * **LastClaimDate**: The date of the customer's most recent claim.
   * **PaymentStatus**: The current payment status for the customer.
   * **RetentionStatus**: Classification of the customer as At Risk or Engaged.
6. **Reason**:
   * Identify at-risk customers to enable targeted retention strategies.
   * Provide actionable insights to reduce customer churn by analyzing recent claim activity and payment behavior.

#### ****Objective 3: Operational Efficiency****

* **Logic**:
  + - 1. **Create a Temporary View**
  + The SQL command creates or replaces a temporary view named OperationalEfficiency to store the results of the query for further analysis.
  + Temporary views are not persisted in the database and are valid for the current session only.
    - 1. **Data Source and Joins**
  + **ClaimsMaster (cm)**: Contains general claim information.
  + **ClaimsDetails (cd)**: Provides details about the claim, including the ServiceDate.
  + **ClaimsPayment (cp)**: Stores payment-related data, including the PaymentDate.
  + **Joins Used**:
  + LEFT JOIN: Ensures that all records from ClaimsMaster are included, even if there are no matching records in ClaimsDetails or ClaimsPayment. This helps in keeping all claims for analysis.
  + Join conditions:
  + cm.ClaimID = cd.ClaimID
  + cm.ClaimID = cp.ClaimID
    - 1. **Handle Missing Dates**
  + **WHERE Clause**:
  + Filters out records where either ServiceDate or PaymentDate is NULL.
  + Ensures valid records are used for calculating ProcessingTime and efficiency.
    - 1. **Calculate ProcessingTime**
  + **Logic**:
  + **Handle Negative Values**: If the difference between PaymentDate and ServiceDate is negative (indicating invalid or incorrect dates), set ProcessingTime to 0.
  + **Valid Calculation**: For valid dates, calculate the time difference using DATEDIFF(cp.PaymentDate, cd.ServiceDate).
  + The efficiency is categorized into several levels based on this processing time, while also handling potential errors, such as negative processing times due to invalid data.
  + If ProcessingTime is <0 days ‘Invalid dates’,
  + If ProcessingTime is <=5 days ‘Highly Efficient’
  + If ProcessingTime is <=10 days ‘Moderately Efficient’
  + If ProcessingTime is <=20 days ‘Average Efficiency’
  + If ProcessingTime is >20 days ‘Inefficient’

#### ****Objective 4: Region-Wise Insights****

* **Logic**:
  + Grouped data by Region and ClaimType.
  + **Aggregation Metrics**:
  + TotalClaims: Count of claims (ClaimID) for each region and claim type.
  + TotalClaimAmount: Sum of claim amounts (ClaimAmount) for each region and claim type.
  + AvgClaimAmount: Average of claim amounts for each region and claim type.
  + **Grouping**:
  + Data is grouped by Region and ClaimType to generate aggregated insights at the intersection of these dimensions.
  + **Columns in Output**:
  + Region: Geographic location of claims.
  + ClaimType: Category/type of claim.
  + Aggregation results (TotalClaims, TotalClaimAmount, AvgClaimAmount).
* **Reason**:
  + Provide insights into the claim distribution and trends across different regions and claim types.
  + Help in resource allocation and premium adjustments based on regional and category-specific claim patterns.

#### ****Objective 5: Policy Optimization****

* **Logic**:
  + - * 1. **Aggregation Metrics**:
  + **TotalClaims**: Count of claims (ClaimID) associated with each policy.
  + **ClaimsRatio**: Calculated as the sum of claim amounts (ClaimAmount) divided by the premium amount (PremiumAmount).
  + **Handling Division by Zero**: Used NULLIF to avoid division by zero.
  + **Handling Null Values**: Used COALESCE to replace null claim sums with 0.
    - * 1. **Policy Performance Classification**:
  + Based on the ClaimsRatio, policies are categorized into performance tiers:
  + **Invalid Ratio**: When the ClaimsRatio is null.
  + **Highly Profitable**: Ratio < 0.5.
  + **Moderately Profitable**: Ratio between 0.5 and 0.7.
  + **Risky**: Ratio between 0.7 and 0.9.
  + **Underperforming**: Ratio > 0.9.
  + **Optimal**: When none of the above conditions apply.
    - * 1. **Grouping**:
  + Data is grouped by **PolicyID** and **PremiumAmount** to compute metrics at the policy level.
    - * 1. **Columns in Output**:
  + **PolicyID**: Unique identifier for the policy.
  + **PremiumAmount**: Premium charged for the policy.
  + Aggregated insights: TotalClaims, ClaimsRatio, and PolicyPerformance.
    - 1. **Reason**:
  + Identify underperforming policies that require adjustments.
  + Highlight highly profitable policies for continued focus or expansion.
  + Support better policy design and resource allocation decisions.

### ****Conclusion****

The cleaning, transformation, and preparation steps ensured high data quality and enabled accurate analysis for each objective. These steps laid the foundation for actionable business insights, helping to mitigate fraud, improve customer retention, streamline operations, and optimize policies.