**SageMaker Customer Churn Pipeline – Project Summary**

**Objective:**

The Object of this project was to build and deploy a machine learning pipeline in **Amazon SageMaker** that predicts customer churn using **XGBoost**, leveraging real world Telco customer data. The pipeline performs data preprocessing, model training, evaluation, conditional registration, and automation using SageMaker Pipelines.

**Preprocessing Strategy (Why & How)**

**Dataset Source:**

We used Telco\_customer\_churn.csv, a real-world dataset commonly used for churn modeling tasks. It includes categorical and numerical customer features such as service plans, tenure, billing, internet service, and labels indicating churn.

 (press on file to open)

Dataset Variables:

This dataset represents a fictional telco company in California during Q3. It includes 7,043 customer records with 33 variables describing customer demographics, account details, service usage, and churn behavior.

- CustomerID: Unique ID for each customer.

- Count: Used in reporting to count customers.

- Country / State / City / Zip Code: Location of the customer's primary residence.

- Lat Long / Latitude / Longitude: Geographical coordinates of residence.

- Gender: Male or Female.

- Senior Citizen: Yes if 65+, No otherwise.

- Partner / Dependents: Indicates family context (Yes/No).

- Tenure Months: Number of months the customer has been with the company.

- Phone Service / Multiple Lines: Indicates phone service usage.

- Internet Service: No, DSL, Fiber Optic, Cable.

- Online Security / Backup / Device Protection: Value-added Internet services (Yes/No).

- Tech Support: Access to enhanced technical support (Yes/No).

- Streaming TV / Movies: Use of internet for streaming (Yes/No).

- Contract: Month-to-Month, One Year, Two Year.

- Paperless Billing: Yes/No flag.

- Payment Method: Bank Withdrawal, Credit Card, Mailed Check.

- Monthly Charge / Total Charges: Billing details.

- Churn Label / Value: Whether customer left (Yes/No or 1/0).

- Churn Score: Predicted probability of churn (0-100).

- CLTV: Predicted customer lifetime value.

- Churn Reason: Why the customer left.

**Steps Taken:**

**1. Label Handling (Target Column)**

* Checked for presence of either Churn Label (Yes/No) or Churn Value (0/1).
* Cleaned string labels using .str.strip().title() to avoid case mismatch.
* Created a unified binary churn column: 1 = churned, 0 = retained.

*Why?*  
To ensure consistent and numerical label values, which are required by XGBoost.

**2. Missing Value Handling**

* Dropped rows with missing values in critical columns like churn or Total Charges.
* Converted Total Charges to numeric and coerced invalid strings to NaN.

*Why?*  
ML models require clean, numerical inputs. Corrupted entries would reduce model performance or cause crashes.

**3. Dropped Irrelevant Identifiers**

* Removed CustomerID, Churn Label, and Churn Value.

*Why?*  
These identifiers do not contribute predictive value and can introduce bias.

**4. One-Hot Encoding**

* Used pd.get\_dummies(drop\_first=True) to convert categorical variables into numerical binary flags.

*Why?*  
ML models like XGBoost require numeric input. One-hot encoding ensures categorical variables are treated properly.

**5. Leakage Prevention**

* Dropped known leaky columns such as Churn Score, CLTV, Churn Reason, and any columns starting with Churn Reason\_.

*Why?*  
These columns are derived from the churn label or post-churn events. Keeping them would artificially boost accuracy and lead to misleading results.

**6. Reordering Columns**

* Moved the label column churn to the first position in the DataFrame.

*Why?*  
For consistency across training and evaluation steps, and to avoid accidental misalignment.

**7. Train/Test/Validation Split**

* Split into:
  + 60% Train
  + 20% Validation
  + 20% Test
* Used stratified sampling to preserve churn distribution.

*Why?*  
Balanced representation is crucial when working with imbalanced classes like churn.

**8. Feature List Export**

* Saved a model\_features.json file containing the list of features used during training.

*Why?*  
To ensure alignment during model evaluation, especially in automated SageMaker pipelines.

**Pipeline Structure Overview**

**1. Processing Step**

* Runs the full preprocessing script.
* Outputs:
  + train.csv
  + val.csv
  + test.csv
  + model\_features.json

**2. Training Step**

* Trains XGBoost on the cleaned and encoded dataset.
* Hyperparameters:
  + Objective: binary:logistic
  + Eval Metric: logloss
  + Rounds: 100

**3. Evaluation Step**

* Loads model, applies to test set, calculates:
  + Accuracy
  + ROC AUC
  + Confusion matrix

**4. Conditional Step**

* Registers model as Approved **only if accuracy ≥ 75%**.
* Otherwise, registers as PendingManualApproval.

**What I Learned**

* How to build end-to-end SageMaker Pipelines using modular steps.
* Importance of clean preprocessing for real-world ML.
* How feature leakage can falsely inflate performance.
* How to serialize feature schema for consistent downstream use.
* How to use condition steps in SageMaker to enforce evaluation-based decisions.-->

-> Next Page

A screenshot of a computer

AI-generated content may be incorrect.