Customer Churn Analysis - Data Preprocessing Documentation

Prepared by: Sagar Chhetri, Data Engineer

Overview

This document outlines the complete data preprocessing workflow used in the Customer Churn Analysis project. Each step is accompanied by Python code and reasoning to ensure transparency and reproducibility.

1. Loading the Dataset and Initial Inspection

Code:

```
import pandas as pd

df = pd.read_csv("Dataset_ATS_v2.csv")

print(df.info())

print(df.isnull().sum())
```

Explanation:

- The dataset was loaded using pandas.
- The info() method provides column-wise data types and non-null counts.
- isnull().sum() identifies the presence of missing values in each column.

2. Analyzing Categorical Values

Code:

```
categorical_columns = ['gender', 'Dependents', 'PhoneService', 'MultipleLines',
'InternetService', 'Contract', 'Churn']

unique_values = {col: df[col].unique() for col in categorical_columns}

print(unique_values)
```

Explanation:

- Ensures consistency and checks for typos or anomalies in categorical columns.

- Helps identify values like 'Fiber optic' that need standardization.

3. Standardizing Values

Code:

```
df['InternetService'] = df['InternetService'].replace({'Fiber optic': 'Fiber Optic'})

print(df['InternetService'].unique())
```

Explanation:

- Changed 'Fiber optic' to 'Fiber Optic' for consistency.

- Uniform entries are essential for accurate encoding and model performance.

4. One-Hot Encoding

Code:

```
df_encoded = pd.get_dummies(df, columns=['gender', 'Dependents', 'PhoneService',

'MultipleLines', 'InternetService', 'Contract', 'Churn'], drop_first=True)
```

Explanation:

- One-hot encoding transforms categorical variables into binary columns.

- drop_first=True avoids the dummy variable trap.

5. Splitting Features and Target

Code:

```
from sklearn.model_selection import train_test_split

X = df_encoded.drop(columns=['Churn_1'])

y = df_encoded['Churn_1']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)
```

Explanation:

- Dataset split into 80% training and 20% testing sets.

- Stratification preserves the churn rate across splits.

6. Handling Class Imbalance with SMOTE

Code:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42, k_neighbors=5)

X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

Explanation:

- SMOTE creates synthetic samples of the minority class.

- Balances the dataset to prevent bias during training.

7. Normalization and Scaling

Code:

```
from sklearn.preprocessing import MinMaxScaler

numerical_cols = ['tenure', 'MonthlyCharges']

scaler = MinMaxScaler()

X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])

X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])
```

Explanation:

- MinMaxScaler transforms features into a range of [0, 1].

- Ensures all variables contribute equally to the model.

8. Model Evaluation Setup

Code:

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report


models = {

"Logistic Regression": LogisticRegression(max_iter=1000),

"Random Forest": RandomForestClassifier(),

"SVM": SVC()

}


for name, model in models.items():

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(f"{name} Accuracy: ", accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

Explanation:

- Trains three baseline models.

- Evaluates them using accuracy and classification metrics like precision, recall, and F1-score.

Conclusion

This preprocessing pipeline ensures a clean, consistent, and balanced dataset, optimized for building robust churn prediction models. Each step is modular, documented, and designed to support reproducibility and collaboration.