Customer Churn Analysis for Telecommunications Company

Sishir Pandey - Project Manager

Fahim Arman - Data Engineer

Jitesh Akaveeti - Data Analyst (Predictive Modelling)

Chen - Data Analyst (Clustering)

Preeti Khatri - Data Analyst (Predictive Modelling)

Bishesh Aryal - Business Analyst

Data Loading:

In python pandas library is used for data collection. The data is loaded from “preprocessed_ dataset.csv” and saved it in a variable named “churn”.

Loading Data

```
churn = pd.read_csv('preprocessed_dataset.csv')
```

This line loads a preprocessed CSV dataset into a Pandas DataFrame called churn so you can work with it in Python.

Data View:

In churn.head() we can view the full loaded data.

```
In [6]: churn.head()
```

Out[6]:

	gender	SeniorCitizen	Dependents	tenure	PhoneService	MultipleLines	InternetService	Contract	MonthlyCharges	Churn
0	Female	0	No	1	No	No	DSL	Month-to-month	25	Yes
1	Male	0	No	41	Yes	No	DSL	One year	25	No
2	Female	0	Yes	52	Yes	No	DSL	Month-to-month	19	No
3	Female	0	No	1	Yes	No	DSL	One year	76	Yes
4	Male	0	No	67	Yes	No	Fiber optic	Month-to-month	51	No

```
print("Loaded rows:", churn.shape[0], "cols:", churn.shape[1])
```

```
Loaded rows: 7043 cols: 10
```

In this data set there is there is 7043 rows and 10 columns.

```
: print(churn.columns)
```

```
Index(['gender', 'SeniorCitizen', 'Dependents', 'tenure', 'PhoneService',
       'MultipleLines', 'InternetService', 'Contract', 'MonthlyCharges',
       'Churn'],
      dtype='object')
```

The name of the columns are 'gender', 'SeniorCitizen', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'Contract', 'MonthlyCharges' and'Churn'.

```
churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   gender           7043 non-null   object
 1   SeniorCitizen    7043 non-null   int64 
 2   Dependents       7043 non-null   object
 3   tenure           7043 non-null   int64 
 4   PhoneService     7043 non-null   object
 5   MultipleLines    7043 non-null   object
 6   InternetService  7043 non-null   object
 7   Contract         7043 non-null   object
 8   MonthlyCharges   7043 non-null   int64 
 9   Churn            7043 non-null   object
dtypes: int64(3), object(7)
```

In this data set 'SeniorCitizen', 'tenure' and 'MonthlyCharges' are int64 and 'gender', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'Contract', and'Churn' are object.

```
churn["gender"].value_counts()
```

```
Male      3555
Female    3488
Name: gender, dtype: int64
```

```
churn["Dependents"].value_counts()
```

```
No     4933
Yes    2110
Name: Dependents, dtype: int64
```

In “gender” column there is 2 types of data Male and female. Where male is 3555 and Female is 3488. In the “Dependents” column there is 4933 “No” and 2110 “Yes”.

```
churn["PhoneService"].value_counts()
```

```
Yes    6361
No      682
Name: PhoneService, dtype: int64
```

```
churn["MultipleLines"].value_counts()
```

```
No     4072
Yes    2971
Name: MultipleLines, dtype: int64
```

In “PhoneService” there is 6361 “YES” and 682 “No”. In the “MultipleLines” column there is 4072 “No” and 2971 “Yes”.

```
churn["InternetService"].value_counts()
```

```
DSL            3947
Fiber optic    3096
Name: InternetService, dtype: int64
```

```
churn["Contract"].value_counts()
```

```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```

In “InternetService” column there is two values “DSL” and “Fiber optic”. In “Contract” column there is 3 types of data they are “Month-to-month”, Two year and One year.

```
churn["Churn"].value_counts()
```

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In the “Churn” column there is 5174 “No” and 1869 “Yes”.

```
missing_counts_per_column = churn.isnull().sum()
```

```
missing_counts_per_column
```

```
gender             0
SeniorCitizen      0
Dependents         0
tenure             0
PhoneService       0
MultipleLines      0
InternetService    0
Contract           0
MonthlyCharges     0
Churn              0
dtype: int64
```

The is no missing values in the data set.

Data Clean:

```
for c in ["tenure","MonthlyCharges","SeniorCitizen"]:
    if c in churn.columns:
        churn[c] = pd.to_numeric(churn[c], errors="coerce")
```

The parameter errors="coerce" forces non-convertible values (such as "N/A", "unknown", or other text) to be replaced with NaN, representing missing data. This ensures that all values in numeric columns are properly converted to numeric types, allowing accurate mathematical operations and data scaling in subsequent steps.

```
num_cols = churn.select_dtypes(include=["number"]).columns.tolist()
for c in num_cols:
    churn[c] = churn[c].fillna(churn[c].median())
```

This step selects all numeric columns in the DataFrame and replaces any missing values (NaN) in those columns with the median value of each respective column. Filling in missing values ensures that the dataset remains complete and prevents potential errors during the model training process. The median is chosen instead of the mean because it is less sensitive to outliers, providing a more accurate and robust representation of the data when extreme values are present.

```
cat_cols = churn.select_dtypes(include=["object","category"]).columns.tolist()
for c in cat_cols:
    churn[c] = churn[c].fillna("Missing")
```

This step identifies all columns containing categorical or text data (such as “InternetService”, “Contract”, or “Gender”) and replaces any missing values with the string “Missing”. This ensures there are no null values before encoding, as most machine learning algorithms cannot process missing text data, while also preserving information about which entries originally had missing values.

```
if "Churn" in churn.columns:
    churn["Churn_binary"] = churn["Churn"].map({"Yes":1,"No":0}).fillna(0).astype(int)
```

This step checks whether the “Churn” column exists and converts its text values “Yes” and “No” into numeric values 1 and 0, respectively. Any missing values are filled with 0, treating them as “No”, and the resulting column is ensured to be of integer type. This transformation is necessary because machine learning models require numeric target variables, and it creates a new column, Churn_binary, for use in modeling and analysis.

One-hot Encode:

One-hot encode

```
to_encode = [c for c in cat_cols if c != "Churn"]
```

```
to_encode
```

```
['gender',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'Contract']
```

```
churn_enc = pd.get_dummies(churn.drop(columns=["Churn"] if "Churn" in churn.columns else []),
                           columns=to_encode, drop_first=False)
```

```
print(churn_enc.columns)
```

```
Index(['SeniorCitizen', 'tenure', 'MonthlyCharges', 'Churn_binary',
       'gender_Female', 'gender_Male', 'Dependents_No', 'Dependents_Yes',
       'PhoneService_No', 'PhoneService_Yes', 'MultipleLines_No',
       'MultipleLines_Yes', 'InternetService_DSL',
       'InternetService_Fiber optic', 'Contract_Month-to-month',
       'Contract_One year', 'Contract_Two year'],
      dtype='object')
```

This step performs one-hot encoding on all categorical features in the dataset except the “Churn” column. It first identifies which columns are categorical and stores them in the variable to_encode. Then, using the pd.get_dummies() function, it converts each categorical column into multiple binary (0 or 1) columns, one for each unique category. This process allows machine learning models to interpret categorical data numerically without implying any ordinal relationship between categories. The “Churn” column is excluded from encoding to preserve it as the target variable for prediction. The output is shown below.

```
churn_enc
```

gender_Female	gender_Male	Dependents_No	Dependents_Yes	PhoneService_No	PhoneService_Yes	MultipleLines_No	MultipleLines_Yes	InternetService_DSL
1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	1	0	1
1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	1	0	0
...	...	...	...	...	...	...	...	...
0	1	1	0	0	1	0	1	1
1	0	0	1	0	1	0	1	0
0	1	0	1	0	1	1	0	1
0	1	1	0	0	1	0	1	0
0	1	1	0	0	1	1	0	0

Prepare Features and Target:

Prepare features and target

```
feature_cols = [c for c in churn_enc.columns if c != "Churn_binary"]
```

```
feature_cols
```

```
['SeniorCitizen',
 'tenure',
 'MonthlyCharges',
 'gender_Female',
 'gender_Male',
 'Dependents_No',
 'Dependents_Yes',
 'PhoneService_No',
 'PhoneService_Yes',
 'MultipleLines_No',
 'MultipleLines_Yes',
 'InternetService_DSL',
 'InternetService_Fiber optic',
 'Contract_Month-to-month',
 'Contract_One year',
 'Contract_Two year']
```

```
X = churn_enc[feature_cols]
y = churn_enc["Churn_binary"] if "Churn_binary" in churn_enc.columns else None
```

This step prepares the dataset for modeling by separating the features and the target variable. All columns except “Churn_binary” are selected as input features and stored in X, while the “Churn_binary” column is assigned to y as the target variable for prediction. If the “Churn_binary” column does not exist, y is set to None. This separation ensures that the machine learning model can clearly distinguish between input data and the outcome it needs to predict.

Train Test Split:

Train Test Split

```
if y is not None:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                        random_state=42, stratify=y)
else:
    X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
```

This step splits the dataset into training and testing sets to evaluate the model's performance. If the target variable y exists, both the features (X) and target (y) are split, with 20% of the data reserved for testing. The "stratify=y" parameter ensures that the proportion of classes in the target variable is maintained in both training and testing sets. If y does not exist, only the features X are split into training and testing sets. The random_state=42 ensures the split is reproducible.

Save Data :

Save Data

```
# Training data
```

```
X_train.to_csv(os.path.join("./Training_Data", "X_train.csv"), index=False)
```

```
y_train.to_csv(os.path.join("./Training_Data", "y_train.csv"), index=False)
```

```
# Testing Data
```

```
X_test.to_csv(os.path.join("./Testing_Data", "X_test.csv"), index=False)
```

```
y_test.to_csv(os.path.join("./Testing_Data", "y_test.csv"), index=False)
```

This step saves the training and testing datasets to separate CSV files for future use. The feature and target variables for the training set (X_train and y_train) are saved in the ./Training_Data folder, while the corresponding testing set (X_test and y_test) is saved in the ./Testing_Data folder. The index=False parameter ensures that row indices are not included in the saved files, keeping the data clean and ready for modeling or analysis.