Predicting Customer Churn in Telecom

1. Introduction

Customer churn is one of the most critical issues faced by telecom service providers. Churn refers to customers leaving the service. Predicting customer churn allows companies to take preventive actions, improve customer satisfaction, reduce revenue loss, and design better retention strategies.
In this project, we used machine learning to predict whether a customer will churn based on several demographic, service-related, and account-related features. The entire workflow includes data cleaning, preprocessing, visualization, model building, evaluation, and saving the final model.

---

2. Dataset Overview

The dataset contains various customer attributes with the goal of predicting the **Churn** column (Yes/No). The key columns include:

- **Demographics:** gender, SeniorCitizen, Partner, Dependents
- **Service usage:** PhoneService, MultipleLines, InternetService, StreamingTV, TechSupport
- **Account details:** tenure, Contract type, PaymentMethod, MonthlyCharges, TotalCharges
- **Target variable:** Churn

The dataset originally included customerID, which was removed because IDs do not help prediction and cannot be scaled or encoded properly.

---

3. Data Cleaning & Preparation

Several preprocessing tasks were performed to ensure model readiness.

a) Converting Column Names to snake_case

To maintain consistency and follow Python best practices, all column names were converted from mixed case (e.g., “TotalCharges”, “PaymentMethod”) to **snake_case** (e.g., “total_charges”, “payment_method”).

This improves readability and prevents errors during coding, especially while selecting columns or applying transformations.

b) Handling TotalCharges

The TotalCharges column contained blank spaces, causing conversion issues. Then filled with the median to maintain numeric consistency.

```
# Convert all column names to snake_case
df.columns = (
    df.columns
      .str.replace('([A-Z])', r'_\1', regex=True)
      .str.lower()
      .str.strip('_')
)
```

```
# rename selected columns
df.rename(columns={
    'customer_i_d': 'customer_id',
    'streaming_t_v': 'streaming_tv'
}, inplace=True)
```

```
# Convert TotalCharges to float
df['total_charges'] = pd.to_numeric(df['total_charges'], errors='coerce')
```

```
# Fill blank cells
df['total_charges'] = df['total_charges'].fillna(df['total_charges'].median())
```

c) Encoding Categorical Data

Machine learning models cannot understand text categories, so encoding was necessary.

Binary Categories → LabelEncoder

Columns like Partner, Dependents, PhoneService, PaperlessBilling, and Churn contained **Yes/No** values.
LabelEncoder converted them to:

- Yes → 1
- No → 0

This keeps the data simple and numeric.

Multi-Class Categories → OneHotEncoder

Columns like InternetService, Contract, PaymentMethod have multiple categories.
LabelEncoder is not suitable here because it assigns ranking, which is wrong.

OneHotEncoder created new binary columns for each category:

Example:
Contract → Month-to-month | One year | Two year
becomes:
Contract_OneYear, Contract_TwoYear

This prevents misinterpretation of category importance.

```
# Encode Binary Columns using LabelEncoder
le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])
```

```
# OneHotEncode Multi-Class Columns
ct = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first'), multi_class_cols)
    ],
    remainder='passthrough'
)
```

---

4. Feature Scaling

Some models like **Logistic Regression** require features to be on the same scale.

We used **StandardScaler**, which transforms numerical data to:

- Mean = 0
- Standard deviation = 1

This improves model performance and optimization speed.

```
# Define Target & Features
X = df.drop('churn', axis=1)
y = df['churn']
```

---

5. Train-Test Split

The dataset was split as follows:

- **80% training data**
- **20% testing data**

This helps evaluate the model on unseen data.

```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

---

6. Model Building

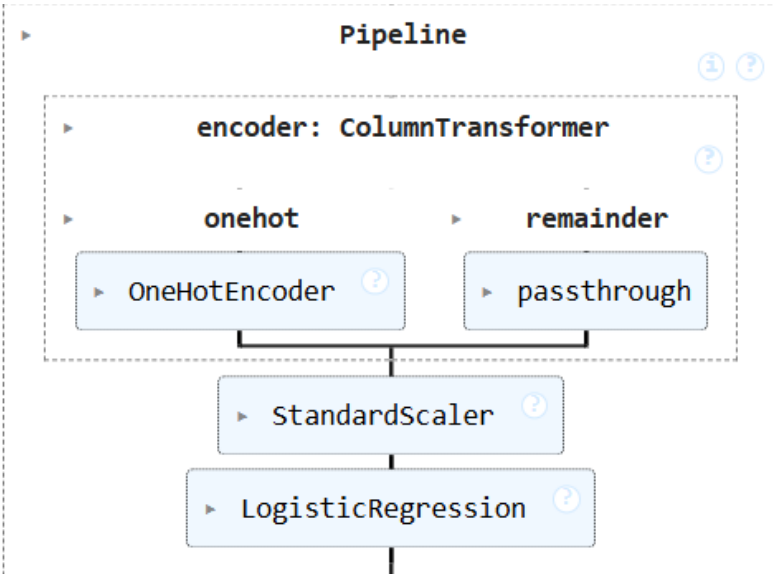
Two machine-learning models were built:

a) Logistic Regression

A pipeline was created:

1. Apply OneHotEncoder
2. Apply StandardScaler
3. Fit Logistic Regression model

Logistic Regression is good for binary classification, interpretable, and simple.



b) Random Forest Classifier

Random Forest does not need scaling and works well with mixed data types.
It builds several decision trees and combines their outputs for better accuracy.

---

7. Model Evaluation

a) Logistic Regression Results

We evaluated using:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Logistic Regression performed well in identifying churn customers but struggled slightly with rare classes.

b) Random Forest Results

Random Forest generally outperformed Logistic Regression due to its ability to capture complex relationships among features.
The classification report and confusion matrix showed improved predictions on churn and non-churn cases.

```
Random Forest Performance:
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1036
           1       0.65      0.47      0.55       373

    accuracy                           0.79      1409
   macro avg       0.74      0.69      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

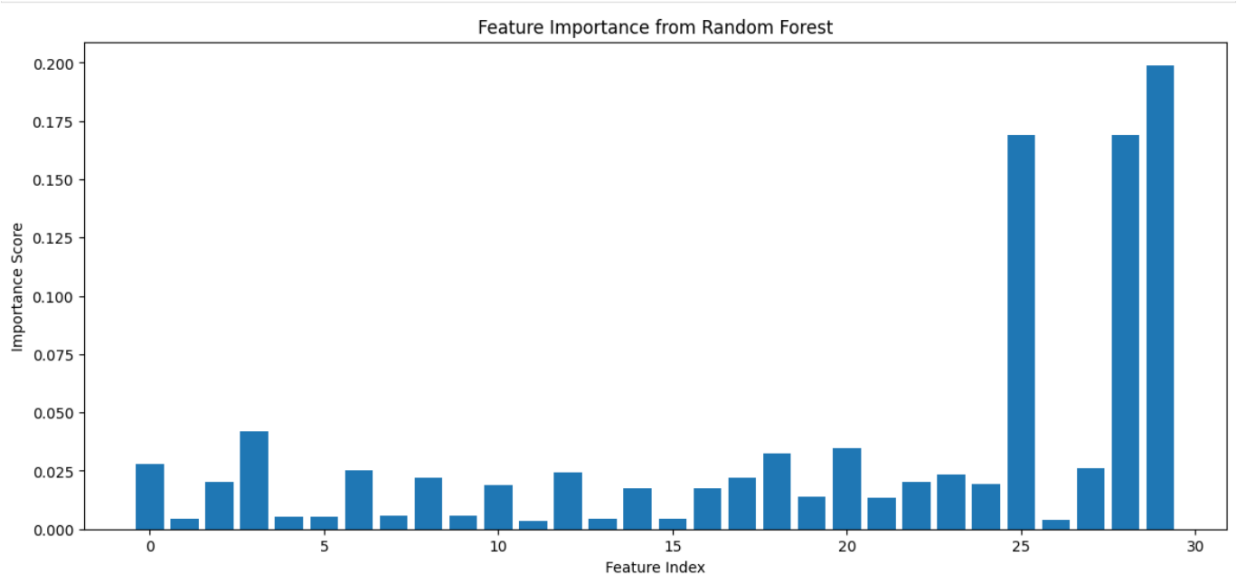
---

8. Feature Importance

Random Forest revealed the most influential features. Often, the top contributors include:

- Tenure
- MonthlyCharges
- Contract type
- InternetService
- TechSupport
- OnlineSecurity

These features strongly correlate with whether a customer leaves or stays.



---

9. Saving the Final Model

To reuse the model later without retraining, we used:

joblib.dump(model, "customer_churn_model.joblib")

This allows easy deployment in apps, dashboards, or APIs.

---

10. Conclusion

This churn prediction project demonstrates the complete lifecycle of a machine-learning model:

- Data cleaning
- Feature encoding
- Scaling
- Model building

- Evaluation
- Saving the final model

The Random Forest model performed best, and feature importance analysis provided valuable business insights. Telecom companies can use this model to identify at-risk customers and implement retention strategies such as personalized offers, improved service, and contract-based incentives.