**Predicting Customer Churn in a Telecommunications Company**

**Objective**

The primary objective of this project is to develop a predictive model that can identify customers at risk of churning, enabling the company to take proactive measures to retain them.

**Data Collection and Preprocessing**

**Data Collection**

The dataset used in this project is obtained from [Kaggle](https://www.kaggle.com/datasets/blastchar/telcocustomer-churn). It contains information about customers, their demographic details, services they have subscribed to, and whether they have churned or not.

**Data Preprocessing**

1. **Handling Missing Values**:
   * Checked for missing values and handled them appropriately. For example, missing values in numerical columns were replaced with the median values, and missing values in categorical columns were filled with the mode.
2. **Encoding Categorical Variables**:
   * Converted binary categorical variables (Yes/No) to numerical (1/0).
   * Created dummy variables for categorical features with more than two levels using one-hot encoding.
3. **Feature Scaling**:
   * Standardized numerical features such as tenure, MonthlyCharges, and TotalCharges using StandardScaler to ensure all features contribute equally to the model.

**Code Implementation**

python

Copy code

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

# Load datasets

churn\_data = pd.read\_csv('churn\_data.csv')

customer\_data = pd.read\_csv('customer\_data.csv')

internet\_data = pd.read\_csv('internet\_data.csv')

# Merge datasets

df\_1 = pd.merge(churn\_data, customer\_data, how='inner', on='customerID')

telecom = pd.merge(df\_1, internet\_data, how='inner', on='customerID')

# Handle missing values

telecom.fillna(telecom.median(), inplace=True)

# Encode categorical variables

varlist = ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']

def binary\_map(x):

return x.map({'Yes': 1, "No": 0})

telecom[varlist] = telecom[varlist].apply(binary\_map)

# Create dummy variables

dummy1 = pd.get\_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetService']], drop\_first=True)

telecom = pd.concat([telecom, dummy1], axis=1)

# Scale numerical features

scaler = StandardScaler()

telecom[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit\_transform(telecom[['tenure', 'MonthlyCharges', 'TotalCharges']])

**Exploratory Data Analysis (EDA)**

Exploratory Data Analysis was performed to understand customer behavior and factors influencing churn. Key findings were visualized using appropriate graphs and charts.

**Key Findings**

1. **Churn Distribution**:
   * A significant percentage of customers have churned, indicating potential issues with customer satisfaction or service quality.
2. **Correlation Analysis**:
   * Identified strong correlations between churn and variables such as contract type, payment method, and tenure.
3. **Visualizations**:
   * **Churn by Contract Type**: Customers with month-to-month contracts are more likely to churn compared to those with one or two-year contracts.
   * **Churn by Payment Method**: Electronic check payment method is associated with higher churn rates.
   * **Churn by Tenure**: Customers with shorter tenure are more likely to churn.

**Code Implementation**

python

Copy code

import matplotlib.pyplot as plt

import seaborn as sns

# Churn distribution

plt.figure(figsize=(6,4))

sns.countplot(x='Churn', data=telecom)

plt.title('Churn Distribution')

plt.show()

# Correlation analysis

plt.figure(figsize=(12,8))

sns.heatmap(telecom.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

# Churn by contract type

plt.figure(figsize=(6,4))

sns.countplot(x='Contract', hue='Churn', data=telecom)

plt.title('Churn by Contract Type')

plt.show()

# Churn by payment method

plt.figure(figsize=(6,4))

sns.countplot(x='PaymentMethod', hue='Churn', data=telecom)

plt.title('Churn by Payment Method')

plt.show()

# Churn by tenure

plt.figure(figsize=(6,4))

sns.histplot(data=telecom, x='tenure', hue='Churn', multiple='stack', kde=True)

plt.title('Churn by Tenure')

plt.show()

**Feature Engineering**

Relevant features were created to improve the predictive power of the model.

**Feature Creation**

1. **Tenure Group**:
   * Created a new feature TenureGroup by binning tenure into categorical groups.
2. **Interaction Features**:
   * Created interaction features such as MonthlyCharges \* Tenure to capture the combined effect of these variables.

**Code Implementation**

python

Copy code

# Create tenure group

def tenure\_group(tenure):

if tenure <= 12:

return '0-1 Year'

elif tenure <= 24:

return '1-2 Years'

elif tenure <= 36:

return '2-3 Years'

elif tenure <= 48:

return '3-4 Years'

elif tenure <= 60:

return '4-5 Years'

else:

return '5+ Years'

telecom['TenureGroup'] = telecom['tenure'].apply(tenure\_group)

dummy2 = pd.get\_dummies(telecom['TenureGroup'], drop\_first=True)

telecom = pd.concat([telecom, dummy2], axis=1)

# Create interaction features

telecom['MonthlyCharges\_Tenure'] = telecom['MonthlyCharges'] \* telecom['tenure']

**Building the Churn Prediction Model**

Various machine learning algorithms were implemented for churn prediction. The models considered include Logistic Regression, Random Forest, and Gradient Boosting.

**Model Implementation**

1. **Logistic Regression**:

python

Copy code

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

# Split data into training and test sets

X = telecom.drop(columns=['Churn', 'customerID'])

y = telecom['Churn']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Logistic Regression

logreg = LogisticRegression()

logreg.fit(X\_train, y\_train)

1. **Random Forest**:

python

Copy code

from sklearn.ensemble import RandomForestClassifier

# Random Forest

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

1. **Gradient Boosting**:

python

Copy code

from sklearn.ensemble import GradientBoostingClassifier

# Gradient Boosting

gb = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

gb.fit(X\_train, y\_train)

**Model Evaluation**

The performance of the churn prediction models was evaluated using metrics such as accuracy, precision, recall, and F1-score.

**Evaluation Metrics**

1. **Logistic Regression**:

python

Copy code

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Predictions

y\_pred\_logreg = logreg.predict(X\_test)

# Evaluation

print('Logistic Regression:')

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_logreg))

print('Precision:', precision\_score(y\_test, y\_pred\_logreg))

print('Recall:', recall\_score(y\_test, y\_pred\_logreg))

print('F1 Score:', f1\_score(y\_test, y\_pred\_logreg))

1. **Random Forest**:

python

Copy code

# Predictions

y\_pred\_rf = rf.predict(X\_test)

# Evaluation

print('Random Forest:')

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_rf))

print('Precision:', precision\_score(y\_test, y\_pred\_rf))

print('Recall:', recall\_score(y\_test, y\_pred\_rf))

print('F1 Score:', f1\_score(y\_test, y\_pred\_rf))

1. **Gradient Boosting**:

python

Copy code

# Predictions

y\_pred\_gb = gb.predict(X\_test)

# Evaluation

print('Gradient Boosting:')

print('Accuracy:', accuracy\_score(y\_test, y\_pred\_gb))

print('Precision:', precision\_score(y\_test, y\_pred\_gb))

print('Recall:', recall\_score(y\_test, y\_pred\_gb))

print('F1 Score:', f1\_score(y\_test, y\_pred\_gb))

**Conclusion**

Based on the analysis and model evaluation, we conclude the following:

* Logistic Regression provides a baseline model with decent performance.
* Random Forest and Gradient Boosting models offer better accuracy and precision, indicating they capture more complex patterns in the data.

**Next Steps**

* Further tuning of the Random Forest and Gradient Boosting models to optimize performance.
* Implementation of additional feature engineering techniques to improve predictive power.
* Deployment of the best model into a production environment for real-time churn prediction and customer retention strategies.