Customer Churn Prediction System

# 1. Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_curve, auc

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

from imblearn.over\_sampling import SMOTE

import shap

import warnings

warnings.filterwarnings("ignore")

# 2. Load Dataset

data = pd.read\_csv("customer\_churn.csv") # replace with your dataset path

# 3. Data Preprocessing

# Convert TotalCharges to numeric

data['TotalCharges'] = pd.to\_numeric(data['TotalCharges'], errors='coerce')

# Handle missing values

data.fillna(data.median(numeric\_only=True), inplace=True)

# Drop customerID (not predictive)

data.drop('customerID', axis=1, inplace=True)

# Encode categorical features

cat\_cols = data.select\_dtypes(include='object').columns

le = LabelEncoder()

for col in cat\_cols:

data[col] = le.fit\_transform(data[col])

# 4. Feature Engineering

data['MonthlyTenure'] = data['MonthlyCharges'] \* data['tenure']

data['ChargeRatio'] = data['MonthlyCharges'] / (data['TotalCharges'] + 1e-6)

# 5. Split Dataset

X = data.drop('Churn', axis=1)

y = data['Churn']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature Scaling

scaler = StandardScaler()

X\_train[X\_train.columns] = scaler.fit\_transform(X\_train)

X\_test[X\_test.columns] = scaler.transform(X\_test)

# 6. Handle Class Imbalance

smote = SMOTE(random\_state=42)

X\_train\_bal, y\_train\_bal = smote.fit\_resample(X\_train, y\_train)

# 7. Model Training and Tuning

# Logistic Regression

lr = LogisticRegression()

lr.fit(X\_train\_bal, y\_train\_bal)

# Random Forest

rf = RandomForestClassifier()

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

# Support Vector Machine

svm = SVC(probability=True)

svm.fit(X\_train\_bal, y\_train\_bal)

# XGBoost with Grid Search

grid\_params = {

'n\_estimators': [100],

'max\_depth': [4, 6],

'learning\_rate': [0.1],

'subsample': [1],

'colsample\_bytree': [1]

}

xgb = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

grid = GridSearchCV(xgb, grid\_params, cv=3, scoring='f1')

grid.fit(X\_train\_bal, y\_train\_bal)

best\_xgb = grid.best\_estimator\_

# 8. Voting Classifier

ensemble = VotingClassifier(estimators=[('lr', lr), ('rf', rf), ('xgb', best\_xgb)], voting='soft')

ensemble.fit(X\_train\_bal, y\_train\_bal)

# 9. Evaluation Function

def evaluate\_model(model, name):

y\_pred = model.predict(X\_test)

print(f"\n{name} Evaluation")

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

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

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

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

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d')

plt.title(f"{name} Confusion Matrix")

plt.show()

# Evaluate all models

evaluate\_model(lr, "Logistic Regression")

evaluate\_model(rf, "Random Forest")

evaluate\_model(svm, "SVM")

evaluate\_model(best\_xgb, "XGBoost")

evaluate\_model(ensemble, "Ensemble")

# 10. SHAP Explanations

explainer = shap.Explainer(best\_xgb)

shap\_values = explainer(X\_test)

shap.summary\_plot(shap\_values, X\_test, show=False)

plt.savefig("shap\_summary.png", dpi=300, bbox\_inches='tight')

plt.show()

# 11. Save Visualizations

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

sns.heatmap(data.corr(numeric\_only=True), annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap")

plt.savefig("correlation\_heatmap.png", dpi=300, bbox\_inches='tight')

plt.show()

sns.countplot(data['Churn'])

plt.title("Churn Distribution")

plt.savefig("churn\_distribution.png", dpi=300, bbox\_inches='tight')

plt.show()

sns.boxplot(x='Churn', y='MonthlyCharges', data=data)

plt.title("Monthly Charges by Churn")

plt.savefig("monthly\_charges\_by\_churn.png", dpi=300, bbox\_inches='tight')

plt.show()

# ROC Curve Plot

plt.figure(figsize=(8,6))

for model, name in zip([best\_xgb, ensemble], ["XGBoost", "Ensemble"]):

y\_score = model.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_score)

plt.plot(fpr, tpr, label=f"{name} (AUC = {auc(fpr, tpr):.2f})")

plt.plot([0, 1], [0, 1], 'k--')

plt.title("ROC Curve Comparison")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.savefig("roc\_curve\_comparison.png", dpi=300, bbox\_inches='tight')

plt.show()a