Step 1: Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from imblearn.combine import SMOTEENN # Handle class imbalance

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

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

confusion\_matrix, roc\_curve, auc, precision\_recall\_curve,

log\_loss, cohen\_kappa\_score, brier\_score\_loss, hamming\_loss)

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.ensemble import (StackingClassifier, VotingClassifier, ExtraTreesClassifier,

RandomForestClassifier, BaggingClassifier, AdaBoostClassifier)

import matplotlib.pyplot as plt

# Step 2: Load and preprocess the data

file\_path = "/content/WA\_Fn-UseC\_-Telco-Customer-Churn.csv"

data = pd.read\_csv(file\_path)

# Convert 'TotalCharges' to numeric and handle non-numeric values

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

data = data.dropna(subset=['TotalCharges']) # Drop rows with missing TotalCharges

# Encode target variable 'Churn' (Yes = 1, No = 0)

data['Churn'] = data['Churn'].map({'Yes': 1, 'No': 0})

# Drop 'customerID' (not useful for modeling)

data = data.drop(['customerID'], axis=1)

# One-hot encode categorical variables

data = pd.get\_dummies(data, drop\_first=True)

# Step 3: Separate features and target variable

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

y = data['Churn']

# Step 4: Handle class imbalance using SMOTEENN

smote\_enn = SMOTEENN(random\_state=42)

X\_resampled, y\_resampled = smote\_enn.fit\_resample(X, y)

# Step 5: Split the resampled data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_resampled, y\_resampled, test\_size=0.3, random\_state=42, stratify=y\_resampled

)

# Step 6: Standardize numerical features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 7: Define classifiers and ensemble models

svm = SVC(probability=True, kernel='rbf', C=1, gamma='scale', random\_state=42)

xgb = XGBClassifier(n\_estimators=300, learning\_rate=0.05, max\_depth=10, eval\_metric='logloss')

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

et = ExtraTreesClassifier(n\_estimators=100, random\_state=42)

# Stacking classifier

stacking\_model = StackingClassifier(

estimators=[('svm', svm), ('xgb', xgb), ('rf', rf), ('et', et)],

final\_estimator=xgb,

cv=5

)

# Voting classifier

voting\_model = VotingClassifier(

estimators=[('svm', svm), ('xgb', xgb), ('rf', rf), ('et', et)],

voting='soft'

)

# Bagging classifier with Random Forest

bagging\_model = BaggingClassifier(

estimator=rf,

n\_estimators=100,

random\_state=42

)

# Boosting classifier with AdaBoost

boosting\_model = AdaBoostClassifier(

n\_estimators=100,

random\_state=42

)

# Step 8: Evaluate Model Performance

def evaluate\_model(model, X\_test, y\_test):

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

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

# Calculate metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

log\_loss\_value = log\_loss(y\_test, y\_prob)

cohen\_kappa = cohen\_kappa\_score(y\_test, y\_pred)

brier\_score = brier\_score\_loss(y\_test, y\_prob)

hamming = hamming\_loss(y\_test, y\_pred)

entropy\_value = -np.sum(y\_prob \* np.log(y\_prob + 1e-9))

# Print metrics

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"Log Loss: {log\_loss\_value:.4f}")

print(f"Cohen's Kappa: {cohen\_kappa:.4f}")

print(f"Brier Score: {brier\_score:.4f}")

print(f"Hamming Loss: {hamming:.4f}")

print(f"Entropy: {entropy\_value:.4f}")

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

# Step 9: Plot ROC and Precision-Recall curves

def plot\_roc\_pr\_curves(model, X\_test, y\_test):

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

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

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_prob)

plt.figure(figsize=(12, 5))

# ROC Curve

plt.subplot(1, 2, 1)

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

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

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='lower right')

# Precision-Recall Curve

plt.subplot(1, 2, 2)

plt.plot(recall, precision, label='PR Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend(loc='lower left')

plt.show()

# Cross-validation and tuning for RandomForest

param\_grid\_rf = {'n\_estimators': [50, 100, 150], 'max\_depth': [10, 20, 30]}

grid\_search\_rf = GridSearchCV(estimator=rf, param\_grid=param\_grid\_rf, cv=5, scoring='accuracy')

grid\_search\_rf.fit(X\_train, y\_train)

best\_rf = grid\_search\_rf.best\_estimator\_

# Train and evaluate models

print("\nStacking Classifier Performance:")

stacking\_model.fit(X\_train, y\_train)

evaluate\_model(stacking\_model, X\_test, y\_test)

plot\_roc\_pr\_curves(stacking\_model, X\_test, y\_test)

print("\nVoting Classifier Performance:")

voting\_model.fit(X\_train, y\_train)

evaluate\_model(voting\_model, X\_test, y\_test)

plot\_roc\_pr\_curves(voting\_model, X\_test, y\_test)

print("\nBagging Classifier Performance:")

bagging\_model.fit(X\_train, y\_train)

evaluate\_model(bagging\_model, X\_test, y\_test)

plot\_roc\_pr\_curves(bagging\_model, X\_test, y\_test)

print("\nBoosting Classifier Performance:")

boosting\_model.fit(X\_train, y\_train)

evaluate\_model(boosting\_model, X\_test, y\_test)

plot\_roc\_pr\_curves(boosting\_model, X\_test, y\_test)

print("\nTuned Random Forest Performance:")

best\_rf.fit(X\_train, y\_train)

evaluate\_model(best\_rf, X\_test, y\_test)

plot\_roc\_pr\_curves(best\_rf, X\_test, y\_test).