import pandas as pd  
  
# Step 1: Load the data  
df = pd.read\_csv("WA\_Fn-UseC\_-Telco-Customer-Churn.csv")  
  
# Step 2: Preview the data  
print("First 5 rows:")  
print(df.head())  
  
# Step 3: Check dataset shape  
print("\nDataset shape:", df.shape)  
  
# Step 4: Check data types  
print("\nData types:")  
print(df.dtypes)  
  
# Step 5: Check for missing values  
print("\nMissing values per column:")  
print(df.isnull().sum())  
  
# Step 6: Check target variable distribution  
print("\nChurn value counts:")  
print(df['Churn'].value\_counts())

First 5 rows:  
 customerID gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 7590-VHVEG Female 0 Yes No 1 No   
1 5575-GNVDE Male 0 No No 34 Yes   
2 3668-QPYBK Male 0 No No 2 Yes   
3 7795-CFOCW Male 0 No No 45 No   
4 9237-HQITU Female 0 No No 2 Yes   
  
 MultipleLines InternetService OnlineSecurity ... DeviceProtection \  
0 No phone service DSL No ... No   
1 No DSL Yes ... Yes   
2 No DSL Yes ... No   
3 No phone service DSL Yes ... Yes   
4 No Fiber optic No ... No   
  
 TechSupport StreamingTV StreamingMovies Contract PaperlessBilling \  
0 No No No Month-to-month Yes   
1 No No No One year No   
2 No No No Month-to-month Yes   
3 Yes No No One year No   
4 No No No Month-to-month Yes   
  
 PaymentMethod MonthlyCharges TotalCharges Churn   
0 Electronic check 29.85 29.85 No   
1 Mailed check 56.95 1889.5 No   
2 Mailed check 53.85 108.15 Yes   
3 Bank transfer (automatic) 42.30 1840.75 No   
4 Electronic check 70.70 151.65 Yes   
  
[5 rows x 21 columns]  
  
Dataset shape: (7043, 21)  
  
Data types:  
customerID object  
gender object  
SeniorCitizen int64  
Partner object  
Dependents object  
tenure int64  
PhoneService object  
MultipleLines object  
InternetService object  
OnlineSecurity object  
OnlineBackup object  
DeviceProtection object  
TechSupport object  
StreamingTV object  
StreamingMovies object  
Contract object  
PaperlessBilling object  
PaymentMethod object  
MonthlyCharges float64  
TotalCharges object  
Churn object  
dtype: object  
  
Missing values per column:  
customerID 0  
gender 0  
SeniorCitizen 0  
Partner 0  
Dependents 0  
tenure 0  
PhoneService 0  
MultipleLines 0  
InternetService 0  
OnlineSecurity 0  
OnlineBackup 0  
DeviceProtection 0  
TechSupport 0  
StreamingTV 0  
StreamingMovies 0  
Contract 0  
PaperlessBilling 0  
PaymentMethod 0  
MonthlyCharges 0  
TotalCharges 0  
Churn 0  
dtype: int64  
  
Churn value counts:  
Churn  
No 5174  
Yes 1869  
Name: count, dtype: int64

import numpy as np  
  
# Step 1: Check why TotalCharges is object  
print("Unique problematic values in TotalCharges:")  
print(df.loc[df['TotalCharges'].str.strip() == '', 'TotalCharges'])  
  
# Step 2: Convert TotalCharges to numeric  
df['TotalCharges'] = pd.to\_numeric(df['TotalCharges'], errors='coerce')  
  
# Step 3: Check how many NaNs appeared after conversion  
print("\nMissing TotalCharges after conversion:", df['TotalCharges'].isnull().sum())  
  
# Step 4: Drop rows with missing TotalCharges  
df = df.dropna(subset=['TotalCharges'])  
print("Shape after dropping missing TotalCharges:", df.shape)  
  
# Step 5: Drop customerID — not useful for modeling  
df = df.drop(columns=['customerID'])  
  
# Step 6: Sanity check categorical values  
for col in df.select\_dtypes(include='object').columns:  
 print(f"Unique values in {col}: {df[col].unique()}")  
  
# Step 7: Strip whitespaces and standardize category names if needed  
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)  
  
# Final check  
print("\nData types after cleaning:")  
print(df.dtypes)

Unique problematic values in TotalCharges:  
488   
753   
936   
1082   
1340   
3331   
3826   
4380   
5218   
6670   
6754   
Name: TotalCharges, dtype: object  
  
Missing TotalCharges after conversion: 11  
Shape after dropping missing TotalCharges: (7032, 21)  
Unique values in gender: ['Female' 'Male']  
Unique values in Partner: ['Yes' 'No']  
Unique values in Dependents: ['No' 'Yes']  
Unique values in PhoneService: ['No' 'Yes']  
Unique values in MultipleLines: ['No phone service' 'No' 'Yes']  
Unique values in InternetService: ['DSL' 'Fiber optic' 'No']  
Unique values in OnlineSecurity: ['No' 'Yes' 'No internet service']  
Unique values in OnlineBackup: ['Yes' 'No' 'No internet service']  
Unique values in DeviceProtection: ['No' 'Yes' 'No internet service']  
Unique values in TechSupport: ['No' 'Yes' 'No internet service']  
Unique values in StreamingTV: ['No' 'Yes' 'No internet service']  
Unique values in StreamingMovies: ['No' 'Yes' 'No internet service']  
Unique values in Contract: ['Month-to-month' 'One year' 'Two year']  
Unique values in PaperlessBilling: ['Yes' 'No']  
Unique values in PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'  
 'Credit card (automatic)']  
Unique values in Churn: ['No' 'Yes']  
  
Data types after cleaning:  
gender object  
SeniorCitizen int64  
Partner object  
Dependents object  
tenure int64  
PhoneService object  
MultipleLines object  
InternetService object  
OnlineSecurity object  
OnlineBackup object  
DeviceProtection object  
TechSupport object  
StreamingTV object  
StreamingMovies object  
Contract object  
PaperlessBilling object  
PaymentMethod object  
MonthlyCharges float64  
TotalCharges float64  
Churn object  
dtype: object

C:\Users\danie\AppData\Local\Temp\ipykernel\_23152\3136820042.py:25: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.  
 df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)

# Step 1: Create Tenure Group  
def tenure\_group(tenure):  
 if tenure <= 12:  
 return '0-1 year'  
 elif tenure <= 24:  
 return '1-2 years'  
 elif tenure <= 48:  
 return '2-4 years'  
 elif tenure <= 60:  
 return '4-5 years'  
 else:  
 return '5+ years'  
  
df['TenureGroup'] = df['tenure'].apply(tenure\_group)  
  
# Step 2: Simplify categorical features  
# Convert 'No internet service' and 'No phone service' to 'No' for simplicity  
service\_cols = ['MultipleLines', 'OnlineSecurity', 'OnlineBackup',  
 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']  
  
for col in service\_cols:  
 df[col] = df[col].replace({'No internet service': 'No', 'No phone service': 'No'})  
  
# Step 3: Check updated category levels  
for col in service\_cols + ['TenureGroup']:  
 print(f"Unique values in {col}: {df[col].unique()}")  
  
# Step 4: Drop original 'tenure' since we now have TenureGroup  
df.drop(columns=['tenure'], inplace=True)

Unique values in MultipleLines: ['No' 'Yes']  
Unique values in OnlineSecurity: ['No' 'Yes']  
Unique values in OnlineBackup: ['Yes' 'No']  
Unique values in DeviceProtection: ['No' 'Yes']  
Unique values in TechSupport: ['No' 'Yes']  
Unique values in StreamingTV: ['No' 'Yes']  
Unique values in StreamingMovies: ['No' 'Yes']  
Unique values in TenureGroup: ['0-1 year' '2-4 years' '1-2 years' '5+ years' '4-5 years']

from sklearn.model\_selection import train\_test\_split  
  
# Step 1: Encode target variable  
df['Churn'] = df['Churn'].map({'No': 0, 'Yes': 1})  
  
# Step 2: One-hot encode categorical variables  
df\_encoded = pd.get\_dummies(df, drop\_first=True) # Drop first to avoid dummy trap  
  
print("Shape after encoding:", df\_encoded.shape)  
  
# Step 3: Define features (X) and target (y)  
X = df\_encoded.drop(columns=['Churn'])  
y = df\_encoded['Churn']  
  
# Step 4: Stratified train-test split (preserve churn proportion)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.2, random\_state=42, stratify=y)  
  
# Step 5: Sanity checks  
print("Training set shape:", X\_train.shape)  
print("Test set shape:", X\_test.shape)  
print("Churn rate in training set:", y\_train.mean())  
print("Churn rate in test set:", y\_test.mean())

Shape after encoding: (7032, 27)  
Training set shape: (5625, 26)  
Test set shape: (1407, 26)  
Churn rate in training set: 0.2657777777777778  
Churn rate in test set: 0.2658137882018479

from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix  
  
# Initialize and train Logistic Regression  
baseline\_lr = LogisticRegression(max\_iter=1000, random\_state=42, solver='liblinear')  
baseline\_lr.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred\_lr = baseline\_lr.predict(X\_test)  
y\_proba\_lr = baseline\_lr.predict\_proba(X\_test)[:, 1]  
  
# Evaluation  
print("===== Logistic Regression (Baseline) =====")  
print(classification\_report(y\_test, y\_pred\_lr))  
print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba\_lr))  
print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_lr))

===== Logistic Regression (Baseline) =====  
 precision recall f1-score support  
  
 0 0.84 0.90 0.87 1033  
 1 0.65 0.52 0.58 374  
  
 accuracy 0.80 1407  
 macro avg 0.74 0.71 0.72 1407  
weighted avg 0.79 0.80 0.79 1407  
  
ROC AUC: 0.83180316921277  
Confusion Matrix:  
 [[930 103]  
 [181 193]]

from sklearn.ensemble import RandomForestClassifier  
  
# Initialize and train Random Forest  
baseline\_rf = RandomForestClassifier(random\_state=42)  
baseline\_rf.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred\_rf = baseline\_rf.predict(X\_test)  
y\_proba\_rf = baseline\_rf.predict\_proba(X\_test)[:, 1]  
  
# Evaluation  
print("\n===== Random Forest (Baseline) =====")  
print(classification\_report(y\_test, y\_pred\_rf))  
print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba\_rf))  
print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_rf))

===== Random Forest (Baseline) =====  
 precision recall f1-score support  
  
 0 0.83 0.89 0.86 1033  
 1 0.61 0.49 0.55 374  
  
 accuracy 0.78 1407  
 macro avg 0.72 0.69 0.70 1407  
weighted avg 0.77 0.78 0.77 1407  
  
ROC AUC: 0.8133001330427445  
Confusion Matrix:  
 [[917 116]  
 [190 184]]

from imblearn.over\_sampling import ADASYN  
  
# Initialize ADASYN  
adasyn = ADASYN(random\_state=42)  
  
# Fit and resample  
X\_resampled\_adasyn, y\_resampled\_adasyn = adasyn.fit\_resample(X\_train, y\_train)  
  
# Check new class distribution  
print("ADASYN resampled class distribution:")  
print(pd.Series(y\_resampled\_adasyn).value\_counts())

ADASYN resampled class distribution:  
Churn  
1 4194  
0 4130  
Name: count, dtype: int64

from imblearn.under\_sampling import TomekLinks  
  
# Initialize Tomek Links  
tomek = TomekLinks()  
  
# Apply Tomek Links \*directly\* to the original training set  
X\_resampled\_tomek, y\_resampled\_tomek = tomek.fit\_resample(X\_train, y\_train)  
  
# Check new class distribution  
print("Tomek Links resampled class distribution:")  
print(pd.Series(y\_resampled\_tomek).value\_counts())

Tomek Links resampled class distribution:  
Churn  
0 3680  
1 1495  
Name: count, dtype: int64

# Step 1: Apply ADASYN first  
X\_resampled\_hybrid, y\_resampled\_hybrid = adasyn.fit\_resample(X\_train, y\_train)  
  
# Step 2: Apply Tomek Links \*to the ADASYN-resampled data\*  
X\_resampled\_hybrid\_final, y\_resampled\_hybrid\_final = tomek.fit\_resample(  
 X\_resampled\_hybrid, y\_resampled\_hybrid)  
  
# Check new class distribution  
print("ADASYN + Tomek Links hybrid resampled class distribution:")  
print(pd.Series(y\_resampled\_hybrid\_final).value\_counts())

ADASYN + Tomek Links hybrid resampled class distribution:  
Churn  
0 4130  
1 3765  
Name: count, dtype: int64

from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix  
  
def train\_and\_evaluate\_model(model, X\_train\_resampled, y\_train\_resampled, X\_test, y\_test, model\_name):  
 # Train  
 model.fit(X\_train\_resampled, y\_train\_resampled)  
  
 # Predict  
 y\_pred = model.predict(X\_test)  
 y\_proba = model.predict\_proba(X\_test)[:, 1]  
  
 # Evaluation  
 print(f"\n===== {model\_name} =====")  
 print(classification\_report(y\_test, y\_pred))  
 print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba))  
 print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Initialize Logistic Regression  
lr = LogisticRegression(max\_iter=1000, random\_state=42, solver='liblinear')  
  
# ADASYN  
train\_and\_evaluate\_model(lr, X\_resampled\_adasyn, y\_resampled\_adasyn, X\_test, y\_test, "Logistic Regression (ADASYN)")  
  
# Tomek Links  
train\_and\_evaluate\_model(lr, X\_resampled\_tomek, y\_resampled\_tomek, X\_test, y\_test, "Logistic Regression (Tomek Links)")  
  
# Hybrid  
train\_and\_evaluate\_model(lr, X\_resampled\_hybrid\_final, y\_resampled\_hybrid\_final, X\_test, y\_test, "Logistic Regression (ADASYN + Tomek)")

===== Logistic Regression (ADASYN) =====  
 precision recall f1-score support  
  
 0 0.85 0.80 0.82 1033  
 1 0.52 0.61 0.57 374  
  
 accuracy 0.75 1407  
 macro avg 0.69 0.71 0.69 1407  
weighted avg 0.76 0.75 0.75 1407  
  
ROC AUC: 0.7829914945825202  
Confusion Matrix:  
 [[823 210]  
 [144 230]]  
  
===== Logistic Regression (Tomek Links) =====  
 precision recall f1-score support  
  
 0 0.86 0.86 0.86 1033  
 1 0.61 0.61 0.61 374  
  
 accuracy 0.79 1407  
 macro avg 0.74 0.74 0.74 1407  
weighted avg 0.79 0.79 0.79 1407  
  
ROC AUC: 0.8304727417676566  
Confusion Matrix:  
 [[888 145]  
 [145 229]]  
  
===== Logistic Regression (ADASYN + Tomek) =====  
 precision recall f1-score support  
  
 0 0.85 0.83 0.84 1033  
 1 0.55 0.58 0.57 374  
  
 accuracy 0.76 1407  
 macro avg 0.70 0.70 0.70 1407  
weighted avg 0.77 0.76 0.76 1407  
  
ROC AUC: 0.7806826594053973  
Confusion Matrix:  
 [[854 179]  
 [156 218]]

# Initialize Random Forest  
rf = RandomForestClassifier(random\_state=42)  
  
# ADASYN  
train\_and\_evaluate\_model(rf, X\_resampled\_adasyn, y\_resampled\_adasyn, X\_test, y\_test, "Random Forest (ADASYN)")  
  
# Tomek Links  
train\_and\_evaluate\_model(rf, X\_resampled\_tomek, y\_resampled\_tomek, X\_test, y\_test, "Random Forest (Tomek Links)")  
  
# Hybrid  
train\_and\_evaluate\_model(rf, X\_resampled\_hybrid\_final, y\_resampled\_hybrid\_final, X\_test, y\_test, "Random Forest (ADASYN + Tomek)")

===== Random Forest (ADASYN) =====  
 precision recall f1-score support  
  
 0 0.85 0.83 0.84 1033  
 1 0.56 0.59 0.57 374  
  
 accuracy 0.77 1407  
 macro avg 0.70 0.71 0.71 1407  
weighted avg 0.77 0.77 0.77 1407  
  
ROC AUC: 0.8072950908780303  
Confusion Matrix:  
 [[856 177]  
 [153 221]]  
  
===== Random Forest (Tomek Links) =====  
 precision recall f1-score support  
  
 0 0.85 0.85 0.85 1033  
 1 0.59 0.60 0.60 374  
  
 accuracy 0.78 1407  
 macro avg 0.72 0.72 0.72 1407  
weighted avg 0.78 0.78 0.78 1407  
  
ROC AUC: 0.8200130972040317  
Confusion Matrix:  
 [[881 152]  
 [151 223]]  
  
===== Random Forest (ADASYN + Tomek) =====  
 precision recall f1-score support  
  
 0 0.84 0.83 0.84 1033  
 1 0.55 0.56 0.56 374  
  
 accuracy 0.76 1407  
 macro avg 0.70 0.70 0.70 1407  
weighted avg 0.76 0.76 0.76 1407  
  
ROC AUC: 0.8098990014029022  
Confusion Matrix:  
 [[861 172]  
 [163 211]]

from sklearn.feature\_selection import RFE  
from sklearn.linear\_model import LogisticRegression  
  
def perform\_rfe(X, y, n\_features):  
 # Create a Logistic Regression model (solver liblinear for small datasets)  
 model = LogisticRegression(max\_iter=1000, random\_state=42, solver='liblinear')  
  
 # Recursive Feature Elimination  
 rfe = RFE(model, n\_features\_to\_select=n\_features)  
 rfe.fit(X, y)  
  
 # Get selected feature names  
 selected\_features = X.columns[rfe.support\_].tolist()  
 return selected\_features

# Let's select 10 features to keep things simple and prevent overfitting  
n\_features = 10

# ADASYN  
features\_adasyn = perform\_rfe(X\_resampled\_adasyn, y\_resampled\_adasyn, n\_features)  
print("\nSelected features for ADASYN:", features\_adasyn)  
  
# Tomek Links  
features\_tomek = perform\_rfe(X\_resampled\_tomek, y\_resampled\_tomek, n\_features)  
print("\nSelected features for Tomek Links:", features\_tomek)  
  
# ADASYN + Tomek Hybrid  
features\_hybrid = perform\_rfe(X\_resampled\_hybrid\_final, y\_resampled\_hybrid\_final, n\_features)  
print("\nSelected features for ADASYN + Tomek Links hybrid:", features\_hybrid)

Selected features for ADASYN: ['gender\_Male', 'InternetService\_Fiber optic', 'StreamingMovies\_Yes', 'Contract\_One year', 'Contract\_Two year', 'PaperlessBilling\_Yes', 'PaymentMethod\_Credit card (automatic)', 'PaymentMethod\_Electronic check', 'PaymentMethod\_Mailed check', 'TenureGroup\_5+ years']  
  
Selected features for Tomek Links: ['InternetService\_Fiber optic', 'InternetService\_No', 'OnlineSecurity\_Yes', 'Contract\_One year', 'Contract\_Two year', 'PaymentMethod\_Electronic check', 'TenureGroup\_1-2 years', 'TenureGroup\_2-4 years', 'TenureGroup\_4-5 years', 'TenureGroup\_5+ years']  
  
Selected features for ADASYN + Tomek Links hybrid: ['gender\_Male', 'InternetService\_Fiber optic', 'StreamingMovies\_Yes', 'Contract\_One year', 'Contract\_Two year', 'PaperlessBilling\_Yes', 'PaymentMethod\_Credit card (automatic)', 'PaymentMethod\_Electronic check', 'PaymentMethod\_Mailed check', 'TenureGroup\_5+ years']

# Subset for ADASYN  
X\_adasyn\_rfe = X\_resampled\_adasyn[features\_adasyn]  
  
# Subset for Tomek Links  
X\_tomek\_rfe = X\_resampled\_tomek[features\_tomek]  
  
# Subset for ADASYN + Tomek Hybrid  
X\_hybrid\_rfe = X\_resampled\_hybrid\_final[features\_hybrid]  
  
# Also subset the test set for fair comparison  
X\_test\_adasyn = X\_test[features\_adasyn]  
X\_test\_tomek = X\_test[features\_tomek]  
X\_test\_hybrid = X\_test[features\_hybrid]

def train\_and\_evaluate\_model(model, X\_train\_resampled, y\_train\_resampled, X\_test, y\_test, model\_name):  
 model.fit(X\_train\_resampled, y\_train\_resampled)  
  
 y\_pred = model.predict(X\_test)  
 y\_proba = model.predict\_proba(X\_test)[:, 1]  
  
 print(f"\n===== {model\_name} =====")  
 print(classification\_report(y\_test, y\_pred))  
 print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba))  
 print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Logistic Regression model  
lr = LogisticRegression(max\_iter=1000, random\_state=42, solver='liblinear')  
  
# ADASYN  
train\_and\_evaluate\_model(lr, X\_adasyn\_rfe, y\_resampled\_adasyn, X\_test\_adasyn, y\_test, "Logistic Regression (ADASYN + RFE)")  
  
# Tomek  
train\_and\_evaluate\_model(lr, X\_tomek\_rfe, y\_resampled\_tomek, X\_test\_tomek, y\_test, "Logistic Regression (Tomek + RFE)")  
  
# Hybrid  
train\_and\_evaluate\_model(lr, X\_hybrid\_rfe, y\_resampled\_hybrid\_final, X\_test\_hybrid, y\_test, "Logistic Regression (Hybrid + RFE)")

===== Logistic Regression (ADASYN + RFE) =====  
 precision recall f1-score support  
  
 0 0.85 0.72 0.78 1033  
 1 0.46 0.66 0.54 374  
  
 accuracy 0.71 1407  
 macro avg 0.66 0.69 0.66 1407  
weighted avg 0.75 0.71 0.72 1407  
  
ROC AUC: 0.7665902231701446  
Confusion Matrix:  
 [[748 285]  
 [129 245]]  
  
===== Logistic Regression (Tomek + RFE) =====  
 precision recall f1-score support  
  
 0 0.85 0.85 0.85 1033  
 1 0.59 0.59 0.59 374  
  
 accuracy 0.78 1407  
 macro avg 0.72 0.72 0.72 1407  
weighted avg 0.78 0.78 0.78 1407  
  
ROC AUC: 0.8270094890019722  
Confusion Matrix:  
 [[882 151]  
 [153 221]]  
  
===== Logistic Regression (Hybrid + RFE) =====  
 precision recall f1-score support  
  
 0 0.84 0.75 0.79 1033  
 1 0.47 0.61 0.53 374  
  
 accuracy 0.71 1407  
 macro avg 0.65 0.68 0.66 1407  
weighted avg 0.74 0.71 0.72 1407  
  
ROC AUC: 0.7680293625854813  
Confusion Matrix:  
 [[773 260]  
 [147 227]]

# Random Forest model  
rf = RandomForestClassifier(random\_state=42)  
  
# ADASYN  
train\_and\_evaluate\_model(rf, X\_adasyn\_rfe, y\_resampled\_adasyn, X\_test\_adasyn, y\_test, "Random Forest (ADASYN + RFE)")  
  
# Tomek  
train\_and\_evaluate\_model(rf, X\_tomek\_rfe, y\_resampled\_tomek, X\_test\_tomek, y\_test, "Random Forest (Tomek + RFE)")  
  
# Hybrid  
train\_and\_evaluate\_model(rf, X\_hybrid\_rfe, y\_resampled\_hybrid\_final, X\_test\_hybrid, y\_test, "Random Forest (Hybrid + RFE)")

===== Random Forest (ADASYN + RFE) =====  
 precision recall f1-score support  
  
 0 0.86 0.74 0.79 1033  
 1 0.48 0.66 0.56 374  
  
 accuracy 0.72 1407  
 macro avg 0.67 0.70 0.68 1407  
weighted avg 0.76 0.72 0.73 1407  
  
ROC AUC: 0.7754308358915158  
Confusion Matrix:  
 [[765 268]  
 [127 247]]  
  
===== Random Forest (Tomek + RFE) =====  
 precision recall f1-score support  
  
 0 0.84 0.87 0.86 1033  
 1 0.61 0.54 0.57 374  
  
 accuracy 0.78 1407  
 macro avg 0.72 0.71 0.71 1407  
weighted avg 0.78 0.78 0.78 1407  
  
ROC AUC: 0.819914738754782  
Confusion Matrix:  
 [[903 130]  
 [173 201]]  
  
===== Random Forest (Hybrid + RFE) =====  
 precision recall f1-score support  
  
 0 0.86 0.75 0.80 1033  
 1 0.49 0.66 0.56 374  
  
 accuracy 0.72 1407  
 macro avg 0.67 0.70 0.68 1407  
weighted avg 0.76 0.72 0.74 1407  
  
ROC AUC: 0.7792512851307908  
Confusion Matrix:  
 [[774 259]  
 [129 245]]

from sklearn.model\_selection import GridSearchCV  
  
# Define parameter grid  
lr\_param\_grid = {  
 'penalty': ['l1', 'l2'],  
 'C': [0.01, 0.1, 1, 10, 100],  
 'solver': ['liblinear']  
}  
  
# Initialize model  
lr = LogisticRegression(max\_iter=1000, random\_state=42)  
  
# Tomek + RFE  
grid\_lr\_tomek = GridSearchCV(lr, lr\_param\_grid, scoring='f1', cv=5)  
grid\_lr\_tomek.fit(X\_tomek\_rfe, y\_resampled\_tomek)  
  
print("Best LR params (Tomek):", grid\_lr\_tomek.best\_params\_)  
  
# ADASYN + RFE  
grid\_lr\_adasyn = GridSearchCV(lr, lr\_param\_grid, scoring='f1', cv=5)  
grid\_lr\_adasyn.fit(X\_adasyn\_rfe, y\_resampled\_adasyn)  
  
print("Best LR params (ADASYN):", grid\_lr\_adasyn.best\_params\_)

Best LR params (Tomek): {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}  
Best LR params (ADASYN): {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

rf\_param\_grid = {  
 'n\_estimators': [100, 200],  
 'max\_depth': [None, 10, 20, 30],  
 'min\_samples\_split': [2, 5, 10],  
 'min\_samples\_leaf': [1, 2, 4],  
 'max\_features': ['sqrt', 'log2']  
}  
  
rf = RandomForestClassifier(random\_state=42)  
  
# Tomek + RFE  
grid\_rf\_tomek = GridSearchCV(rf, rf\_param\_grid, scoring='f1', cv=5)  
grid\_rf\_tomek.fit(X\_tomek\_rfe, y\_resampled\_tomek)  
  
print("Best RF params (Tomek):", grid\_rf\_tomek.best\_params\_)  
  
# ADASYN + RFE  
grid\_rf\_adasyn = GridSearchCV(rf, rf\_param\_grid, scoring='f1', cv=5)  
grid\_rf\_adasyn.fit(X\_adasyn\_rfe, y\_resampled\_adasyn)  
  
print("Best RF params (ADASYN):", grid\_rf\_adasyn.best\_params\_)

Best RF params (Tomek): {'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_leaf': 4, 'min\_samples\_split': 2, 'n\_estimators': 200}  
Best RF params (ADASYN): {'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_leaf': 1, 'min\_samples\_split': 5, 'n\_estimators': 100}

# Logistic Regression with Tomek  
lr\_tomek = LogisticRegression(C=0.1, penalty='l1', solver='liblinear', max\_iter=1000, random\_state=42)  
train\_and\_evaluate\_model(lr\_tomek, X\_tomek\_rfe, y\_resampled\_tomek, X\_test\_tomek, y\_test, "Logistic Regression (Tomek + RFE + Tuned)")  
  
# Logistic Regression with ADASYN  
lr\_adasyn = LogisticRegression(C=0.01, penalty='l2', solver='liblinear', max\_iter=1000, random\_state=42)  
train\_and\_evaluate\_model(lr\_adasyn, X\_adasyn\_rfe, y\_resampled\_adasyn, X\_test\_adasyn, y\_test, "Logistic Regression (ADASYN + RFE + Tuned)")

===== Logistic Regression (Tomek + RFE + Tuned) =====  
 precision recall f1-score support  
  
 0 0.85 0.86 0.85 1033  
 1 0.60 0.59 0.59 374  
  
 accuracy 0.79 1407  
 macro avg 0.72 0.72 0.72 1407  
weighted avg 0.78 0.79 0.78 1407  
  
ROC AUC: 0.825956018242904  
Confusion Matrix:  
 [[884 149]  
 [153 221]]  
  
===== Logistic Regression (ADASYN + RFE + Tuned) =====  
 precision recall f1-score support  
  
 0 0.87 0.69 0.77 1033  
 1 0.46 0.71 0.55 374  
  
 accuracy 0.70 1407  
 macro avg 0.66 0.70 0.66 1407  
weighted avg 0.76 0.70 0.71 1407  
  
ROC AUC: 0.7852291493029492  
Confusion Matrix:  
 [[717 316]  
 [110 264]]

# Random Forest with Tomek  
rf\_tomek = RandomForestClassifier(  
 n\_estimators=200,  
 max\_depth=None,  
 max\_features='sqrt',  
 min\_samples\_leaf=4,  
 min\_samples\_split=2,  
 random\_state=42  
)  
train\_and\_evaluate\_model(rf\_tomek, X\_tomek\_rfe, y\_resampled\_tomek, X\_test\_tomek, y\_test, "Random Forest (Tomek + RFE + Tuned)")  
  
# Random Forest with ADASYN  
rf\_adasyn = RandomForestClassifier(  
 n\_estimators=100,  
 max\_depth=None,  
 max\_features='sqrt',  
 min\_samples\_leaf=1,  
 min\_samples\_split=5,  
 random\_state=42  
)  
train\_and\_evaluate\_model(rf\_adasyn, X\_adasyn\_rfe, y\_resampled\_adasyn, X\_test\_adasyn, y\_test, "Random Forest (ADASYN + RFE + Tuned)")

===== Random Forest (Tomek + RFE + Tuned) =====  
 precision recall f1-score support  
  
 0 0.84 0.88 0.86 1033  
 1 0.61 0.53 0.57 374  
  
 accuracy 0.79 1407  
 macro avg 0.73 0.71 0.71 1407  
weighted avg 0.78 0.79 0.78 1407  
  
ROC AUC: 0.822491471287098  
Confusion Matrix:  
 [[907 126]  
 [174 200]]  
  
===== Random Forest (ADASYN + RFE + Tuned) =====  
 precision recall f1-score support  
  
 0 0.86 0.74 0.80 1033  
 1 0.48 0.66 0.56 374  
  
 accuracy 0.72 1407  
 macro avg 0.67 0.70 0.68 1407  
weighted avg 0.76 0.72 0.73 1407  
  
ROC AUC: 0.7781266339150286  
Confusion Matrix:  
 [[766 267]  
 [127 247]]

import numpy as np  
from sklearn.metrics import precision\_recall\_fscore\_support  
  
# Predict probabilities  
y\_proba\_lr\_tomek = lr\_tomek.predict\_proba(X\_test\_tomek)[:, 1]  
  
# Define thresholds to test  
thresholds = np.arange(0.3, 0.71, 0.01)  
  
print("Threshold | Precision | Recall | F1")  
for thresh in thresholds:  
 y\_pred\_thresh = (y\_proba\_lr\_tomek >= thresh).astype(int)  
 precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred\_thresh, average='binary')  
 print(f"{thresh:.2f} | {precision:.2f} | {recall:.2f} | {f1:.2f}")

Threshold | Precision | Recall | F1  
0.30 | 0.47 | 0.81 | 0.60  
0.31 | 0.48 | 0.80 | 0.60  
0.32 | 0.49 | 0.78 | 0.60  
0.33 | 0.52 | 0.77 | 0.62  
0.34 | 0.52 | 0.77 | 0.62  
0.35 | 0.52 | 0.77 | 0.62  
0.36 | 0.52 | 0.76 | 0.62  
0.37 | 0.52 | 0.76 | 0.62  
0.38 | 0.52 | 0.76 | 0.62  
0.39 | 0.52 | 0.76 | 0.62  
0.40 | 0.53 | 0.75 | 0.62  
0.41 | 0.53 | 0.74 | 0.62  
0.42 | 0.53 | 0.74 | 0.62  
0.43 | 0.53 | 0.72 | 0.61  
0.44 | 0.55 | 0.68 | 0.61  
0.45 | 0.55 | 0.68 | 0.61  
0.46 | 0.59 | 0.59 | 0.59  
0.47 | 0.59 | 0.59 | 0.59  
0.48 | 0.60 | 0.59 | 0.59  
0.49 | 0.60 | 0.59 | 0.59  
0.50 | 0.60 | 0.59 | 0.59  
0.51 | 0.60 | 0.59 | 0.59  
0.52 | 0.60 | 0.59 | 0.59  
0.53 | 0.60 | 0.58 | 0.59  
0.54 | 0.62 | 0.54 | 0.58  
0.55 | 0.62 | 0.54 | 0.58  
0.56 | 0.64 | 0.45 | 0.53  
0.57 | 0.66 | 0.40 | 0.50  
0.58 | 0.66 | 0.39 | 0.49  
0.59 | 0.66 | 0.39 | 0.49  
0.60 | 0.66 | 0.39 | 0.49  
0.61 | 0.66 | 0.39 | 0.49  
0.62 | 0.66 | 0.39 | 0.49  
0.63 | 0.66 | 0.39 | 0.49  
0.64 | 0.66 | 0.39 | 0.49  
0.65 | 0.66 | 0.39 | 0.49  
0.66 | 0.68 | 0.32 | 0.44  
0.67 | 0.68 | 0.32 | 0.44  
0.68 | 0.68 | 0.32 | 0.44  
0.69 | 0.67 | 0.31 | 0.42  
0.70 | 0.68 | 0.21 | 0.33

y\_proba\_rf\_adasyn = rf\_adasyn.predict\_proba(X\_test\_adasyn)[:, 1]  
  
print("\nThreshold | Precision | Recall | F1")  
for thresh in thresholds:  
 y\_pred\_thresh = (y\_proba\_rf\_adasyn >= thresh).astype(int)  
 precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred\_thresh, average='binary')  
 print(f"{thresh:.2f} | {precision:.2f} | {recall:.2f} | {f1:.2f}")

Threshold | Precision | Recall | F1  
0.30 | 0.40 | 0.86 | 0.55  
0.31 | 0.40 | 0.86 | 0.55  
0.32 | 0.40 | 0.86 | 0.55  
0.33 | 0.41 | 0.84 | 0.55  
0.34 | 0.43 | 0.82 | 0.56  
0.35 | 0.43 | 0.82 | 0.57  
0.36 | 0.43 | 0.82 | 0.57  
0.37 | 0.43 | 0.82 | 0.57  
0.38 | 0.44 | 0.82 | 0.57  
0.39 | 0.44 | 0.82 | 0.57  
0.40 | 0.44 | 0.81 | 0.57  
0.41 | 0.44 | 0.78 | 0.56  
0.42 | 0.44 | 0.78 | 0.56  
0.43 | 0.44 | 0.73 | 0.55  
0.44 | 0.44 | 0.73 | 0.55  
0.45 | 0.44 | 0.73 | 0.55  
0.46 | 0.45 | 0.70 | 0.55  
0.47 | 0.45 | 0.69 | 0.55  
0.48 | 0.47 | 0.67 | 0.55  
0.49 | 0.48 | 0.66 | 0.56  
0.50 | 0.48 | 0.66 | 0.56  
0.51 | 0.48 | 0.65 | 0.55  
0.52 | 0.48 | 0.65 | 0.55  
0.53 | 0.48 | 0.65 | 0.55  
0.54 | 0.49 | 0.65 | 0.56  
0.55 | 0.49 | 0.65 | 0.56  
0.56 | 0.49 | 0.62 | 0.55  
0.57 | 0.50 | 0.61 | 0.55  
0.58 | 0.50 | 0.58 | 0.54  
0.59 | 0.50 | 0.56 | 0.53  
0.60 | 0.50 | 0.54 | 0.52  
0.61 | 0.51 | 0.53 | 0.52  
0.62 | 0.55 | 0.48 | 0.51  
0.63 | 0.55 | 0.48 | 0.51  
0.64 | 0.57 | 0.48 | 0.52  
0.65 | 0.57 | 0.46 | 0.51  
0.66 | 0.57 | 0.44 | 0.50  
0.67 | 0.58 | 0.40 | 0.48  
0.68 | 0.58 | 0.40 | 0.47  
0.69 | 0.58 | 0.40 | 0.47  
0.70 | 0.58 | 0.39 | 0.47

# Logistic Regression - Tomek  
lr\_tomek = LogisticRegression(C=0.1, penalty='l1', solver='liblinear', max\_iter=1000, random\_state=42)  
lr\_tomek.fit(X\_tomek\_rfe, y\_resampled\_tomek)  
  
# Random Forest - ADASYN  
rf\_adasyn = RandomForestClassifier(  
 n\_estimators=100,  
 max\_depth=None,  
 max\_features='sqrt',  
 min\_samples\_leaf=1,  
 min\_samples\_split=5,  
 random\_state=42  
)  
rf\_adasyn.fit(X\_adasyn\_rfe, y\_resampled\_adasyn)

RandomForestClassifier(min\_samples\_split=5, random\_state=42)

from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix  
  
def evaluate\_with\_threshold\_full\_report(model, X\_test, y\_test, threshold, model\_name):  
 y\_proba = model.predict\_proba(X\_test)[:, 1]  
 y\_pred = (y\_proba >= threshold).astype(int)  
  
 print(f"\n===== {model\_name} =====")  
 print("Threshold:", threshold)  
 print(classification\_report(y\_test, y\_pred))  
 print("ROC AUC:", round(roc\_auc\_score(y\_test, y\_proba), 3))  
 print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Logistic Regression - Tomek (Threshold 0.34)  
evaluate\_with\_threshold\_full\_report(lr\_tomek, X\_test\_tomek, y\_test, 0.34, "Logistic Regression (Tomek + RFE + Tuned)")  
  
# Random Forest - ADASYN (Threshold 0.35)  
evaluate\_with\_threshold\_full\_report(rf\_adasyn, X\_test\_adasyn, y\_test, 0.35, "Random Forest (ADASYN + RFE + Tuned)")

===== Logistic Regression (Tomek + RFE + Tuned) =====  
Threshold: 0.34  
 precision recall f1-score support  
  
 0 0.90 0.74 0.81 1033  
 1 0.52 0.77 0.62 374  
  
 accuracy 0.75 1407  
 macro avg 0.71 0.76 0.72 1407  
weighted avg 0.80 0.75 0.76 1407  
  
ROC AUC: 0.826  
Confusion Matrix:  
 [[764 269]  
 [ 85 289]]  
  
===== Random Forest (ADASYN + RFE + Tuned) =====  
Threshold: 0.35  
 precision recall f1-score support  
  
 0 0.91 0.61 0.73 1033  
 1 0.43 0.82 0.57 374  
  
 accuracy 0.67 1407  
 macro avg 0.67 0.72 0.65 1407  
weighted avg 0.78 0.67 0.69 1407  
  
ROC AUC: 0.778  
Confusion Matrix:  
 [[629 404]  
 [ 66 308]]