```
# Basic imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# ML imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
import joblib
import warnings
warnings.filterwarnings("ignore")

RND = 42  # random state for reproducibility
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving support_data.csv to support_data.csv
Saving usage_data.csv to usage_data.csv
Saving customer_profile.csv to customer_profile.csv
Saving churn_label.csv to churn_label.csv
Saving billing_data.csv to billing_data.csv
```

```
# Load CSVs
profile = pd.read_csv('customer_profile.csv')
usage = pd.read_csv('usage_data.csv')
billing = pd.read_csv('billing_data.csv')
support = pd.read_csv('support_data.csv')
churn = pd.read_csv('churn_label.csv')
```

```
# Merge step-by-step
df = profile.merge(usage, on='customer_id', how='left') \
            .merge(billing, on='customer_id', how='left') \
            .merge(support, on='customer_id', how='left') \
            .merge(churn, on='customer_id', how='left')
```

```
# Keeping customer_id separately for final results
customer_ids = df['customer_id'].copy()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   customer_id               1000 non-null   object
 1   join_date                 1000 non-null   datetime64[ns]
 2   age                       1000 non-null   int64
 3   gender                    1000 non-null   object
 4   region                    1000 non-null   object
 5   plan_type                 1000 non-null   object
 6   device_type               1000 non-null   object
 7   avg_monthly_sessions      1000 non-null   int64
 8   last_30_day_sessions      1000 non-null   int64
 9   last_7_day_sessions       1000 non-null   int64
 10  feature_usage_score       1000 non-null   float64
 11  inactivity_days           1000 non-null   int64
 12  engagement_drop_pct       1000 non-null   float64
 13  tenure_months             1000 non-null   int64
 14  renewal_status            1000 non-null   object
 15  last_payment_date         1000 non-null   datetime64[ns]
 16  payment_failed_count      1000 non-null   int64
 17  plan_changes              1000 non-null   int64
 18  monthly_revenue           1000 non-null   int64
 19  complaint_count           1000 non-null   int64
 20  avg_resolution_time       1000 non-null   float64
 21  negative_sentiment_score  1000 non-null   float64
```

```
 22  last_complaint_category   717 non-null    object
 23  churn                     1000 non-null   int64
dtypes: datetime64[ns](2), float64(4), int64(11), object(7)
memory usage: 187.6+ KB
```

Cleaning and handling missing values

```
# Convert dates to datetime
df['join_date'] = pd.to_datetime(df['join_date'], errors='coerce')
df['last_payment_date'] = pd.to_datetime(df['last_payment_date'], errors='coerce')
```

```
# Fill numeric NaNs with median
num_cols = df.select_dtypes(include=['int64','float64']).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())
```

```
# Fill categorical NaNs with 'Unknown'
cat_cols = df.select_dtypes(include=['object']).columns.drop('customer_id')
df[cat_cols] = df[cat_cols].fillna('Unknown')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   customer_id              1000 non-null   object
 1   join_date                1000 non-null   datetime64[ns]
 2   age                      1000 non-null   int64
 3   gender                   1000 non-null   object
 4   region                   1000 non-null   object
 5   plan_type                1000 non-null   object
 6   device_type              1000 non-null   object
 7   avg_monthly_sessions     1000 non-null   int64
 8   last_30_day_sessions     1000 non-null   int64
 9   last_7_day_sessions      1000 non-null   int64
 10  feature_usage_score      1000 non-null   float64
 11  inactivity_days          1000 non-null   int64
 12  engagement_drop_pct      1000 non-null   float64
 13  tenure_months            1000 non-null   int64
 14  renewal_status           1000 non-null   object
 15  last_payment_date        1000 non-null   datetime64[ns]
 16  payment_failed_count     1000 non-null   int64
 17  plan_changes             1000 non-null   int64
 18  monthly_revenue          1000 non-null   int64
 19  complaint_count          1000 non-null   int64
 20  avg_resolution_time      1000 non-null   float64
 21  negative_sentiment_score 1000 non-null   float64
 22  last_complaint_category  1000 non-null   object
 23  churn                    1000 non-null   int64
dtypes: datetime64[ns](2), float64(4), int64(11), object(7)
memory usage: 187.6+ KB
```

```
#encode categorical values
categorical_cols = ['gender','region','plan_type','device_type','renewal_status','last_complaint_category']
df = pd.get_dummies(df, columns=[c for c in categorical_cols if c in df.columns], drop_first=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 31 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   customer_id                        1000 non-null   object
 1   join_date                          1000 non-null   datetime64[ns]
 2   age                                1000 non-null   int64
 3   avg_monthly_sessions               1000 non-null   int64
 4   last_30_day_sessions               1000 non-null   int64
 5   last_7_day_sessions                1000 non-null   int64
 6   feature_usage_score                1000 non-null   float64
 7   inactivity_days                    1000 non-null   int64
 8   engagement_drop_pct                1000 non-null   float64
 9   tenure_months                      1000 non-null   int64
 10  last_payment_date                  1000 non-null   datetime64[ns]
 11  payment_failed_count               1000 non-null   int64
 12  plan_changes                       1000 non-null   int64
```

```
 13  monthly_revenue                   1000 non-null   int64
 14  complaint_count                   1000 non-null   int64
 15  avg_resolution_time               1000 non-null   float64
 16  negative_sentiment_score          1000 non-null   float64
 17  churn                             1000 non-null   int64
 18  gender_Male                       1000 non-null   bool
 19  gender_Other                      1000 non-null   bool
 20  region_North                      1000 non-null   bool
 21  region_South                      1000 non-null   bool
 22  region_West                       1000 non-null   bool
 23  plan_type_Premium                 1000 non-null   bool
 24  plan_type_Standard                1000 non-null   bool
 25  device_type_Mobile                1000 non-null   bool
 26  device_type_Tablet                1000 non-null   bool
 27  renewal_status_Manual             1000 non-null   bool
 28  last_complaint_category_Service   1000 non-null   bool
 29  last_complaint_category_Technical 1000 non-null   bool
 30  last_complaint_category_Unknown   1000 non-null   bool
dtypes: bool(13), datetime64[ns](2), float64(4), int64(11), object(1)
memory usage: 153.4+ KB
```

Feature Engineering

```
today = pd.Timestamp.today()

# Days since join / last payment
df['days_since_join'] = (today - df['join_date']).dt.days
df['days_since_payment'] = (today - df['last_payment_date']).dt.days
```

```
# Engagement & session features
df['sessions_7_to_30_ratio'] = df['last_7_day_sessions'] / (df['last_30_day_sessions'] + 1)
df['session_change_pct'] = ((df['last_7_day_sessions'] - df['avg_monthly_sessions']) / (df['avg_monthly_sessions'] + 1)) * 100
df['inactivity_proportion'] = df['inactivity_days'] / 30
df['feature_use_per_session'] = df['feature_usage_score'] / (df['avg_monthly_sessions'] + 1)
```

```
# Billing / risk features
df['payment_risk'] = df['payment_failed_count'] / (df['tenure_months'] + 1)
df['revenue_engagement_gap'] = df['monthly_revenue'] / (df['avg_monthly_sessions'] + 1)
df['complaints_per_month'] = df['complaint_count'] / (df['tenure_months'] + 1)
df['sentiment_adjusted_complaints'] = df['complaint_count'] * df['negative_sentiment_score']
df['support_pain_score'] = df['avg_resolution_time'] * df['negative_sentiment_score']
df['tenure_risk'] = 1 / (df['tenure_months'] + 1)
df['price_sensitivity'] = df['plan_changes'] / (df['tenure_months'] + 1)
```

```
# Drop raw date columns
df.drop(['join_date','last_payment_date'], axis=1, inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   customer_id                       1000 non-null   object
 1   age                               1000 non-null   int64
 2   avg_monthly_sessions              1000 non-null   int64
 3   last_30_day_sessions              1000 non-null   int64
 4   last_7_day_sessions               1000 non-null   int64
 5   feature_usage_score               1000 non-null   float64
 6   inactivity_days                   1000 non-null   int64
 7   engagement_drop_pct               1000 non-null   float64
 8   tenure_months                     1000 non-null   int64
 9   payment_failed_count              1000 non-null   int64
 10  plan_changes                      1000 non-null   int64
 11  monthly_revenue                   1000 non-null   int64
 12  complaint_count                   1000 non-null   int64
 13  avg_resolution_time               1000 non-null   float64
 14  negative_sentiment_score          1000 non-null   float64
 15  churn                             1000 non-null   int64
 16  gender_Male                       1000 non-null   bool
 17  gender_Other                      1000 non-null   bool
 18  region_North                      1000 non-null   bool
 19  region_South                      1000 non-null   bool
 20  region_West                       1000 non-null   bool
 21  plan_type_Premium                 1000 non-null   bool
 22  plan_type_Standard                1000 non-null   bool
```

```
 23  device_type_Mobile                1000 non-null   bool
 24  device_type_Tablet                1000 non-null   bool
 25  renewal_status_Manual             1000 non-null   bool
 26  last_complaint_category_Service   1000 non-null   bool
 27  last_complaint_category_Technical 1000 non-null   bool
 28  last_complaint_category_Unknown   1000 non-null   bool
 29  days_since_join                   1000 non-null   int64
 30  days_since_payment                1000 non-null   int64
 31  sessions_7_to_30_ratio            1000 non-null   float64
 32  session_change_pct                1000 non-null   float64
 33  inactivity_proportion             1000 non-null   float64
 34  feature_use_per_session           1000 non-null   float64
 35  payment_risk                      1000 non-null   float64
 36  revenue_engagement_gap            1000 non-null   float64
 37  complaints_per_month              1000 non-null   float64
 38  sentiment_adjusted_complaints     1000 non-null   float64
 39  support_pain_score                1000 non-null   float64
 40  tenure_risk                       1000 non-null   float64
 41  price_sensitivity                 1000 non-null   float64
dtypes: bool(13), float64(15), int64(13), object(1)
memory usage: 239.4+ KB
```

```
# Features and target
X = df.drop(['churn','customer_id'] if 'customer_id' in df.columns else ['churn'], axis=1)
y = df['churn'].astype(int)
```

Train-Test Split

```
X_train, X_test, y_train, y_test, cid_train, cid_test = train_test_split(
    X, y, customer_ids, test_size=0.2, stratify=y, random_state=RND
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
#handling class imbalance with SMOTE
sm = SMOTE(random_state=RND)
X_train_bal, y_train_bal = sm.fit_resample(X_train_scaled, y_train)
```

Train Models

```
# Logistic Regression
log_reg = LogisticRegression(max_iter=2000, class_weight='balanced', random_state=RND)
log_reg.fit(X_train_bal, y_train_bal)
```

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=2000, random_state=42)
```

```
# Random Forest
rf = RandomForestClassifier(n_estimators=300, class_weight='balanced', random_state=RND)
rf.fit(X_train_bal, y_train_bal)
```

```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', n_estimators=300,
                       random_state=42)
```

```
# XGBoost
xgb_final = XGBClassifier(
    n_estimators=300, learning_rate=0.05, max_depth=5,
    subsample=0.8, colsample_bytree=0.8,
    objective='binary:logistic', eval_metric='logloss',
    use_label_encoder=False, random_state=RND
)
xgb_final.fit(X_train_bal, y_train_bal)
```

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=None,
              num_parallel_tree=None, ...)
```

```python
#evaluating the models
def eval_report(y_true, y_pred, y_prob, model_name):
    print(f"\n=== {model_name} ===")
    print("Accuracy:", round(accuracy_score(y_true, y_pred),4))
    print("Precision:", round(precision_score(y_true, y_pred),4))
    print("Recall:", round(recall_score(y_true, y_pred),4))
    print("F1 Score:", round(f1_score(y_true, y_pred),4))
    print("ROC-AUC:", round(roc_auc_score(y_true, y_prob),4))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
```

```python
# Predictions
#logistic regression
y_pred_lr = log_reg.predict(X_test_scaled)
y_prob_lr = log_reg.predict_proba(X_test_scaled)[:,1]
```

```python
#random forest
y_pred_rf = rf.predict(X_test_scaled)
y_prob_rf = rf.predict_proba(X_test_scaled)[:,1]
```

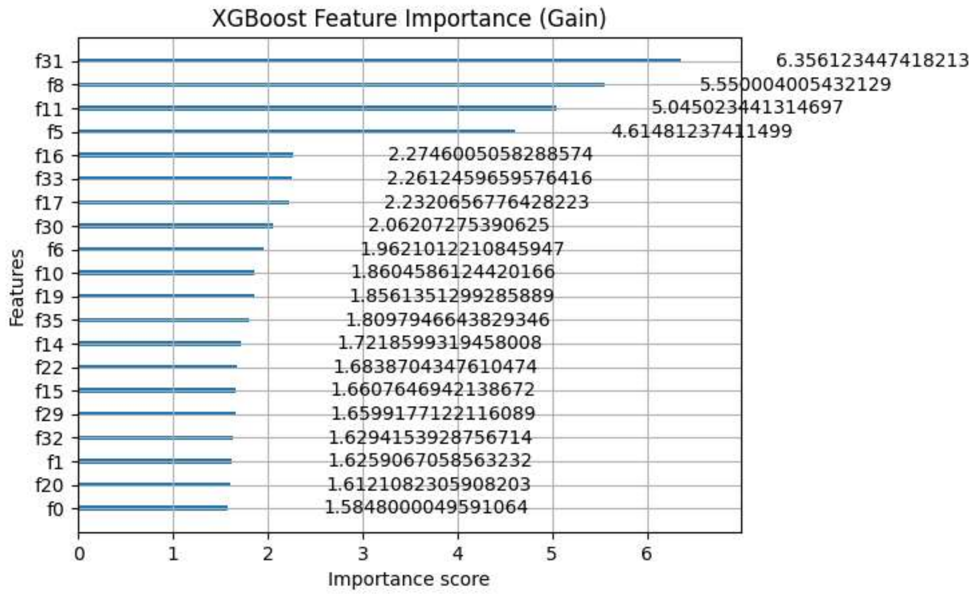
```python
#XGBoost
y_pred_xgb = xgb_final.predict(X_test_scaled)
y_prob_xgb = xgb_final.predict_proba(X_test_scaled)[:,1]
```

```python
#evaluate
eval_report(y_test, y_pred_lr, y_prob_lr, "Logistic Regression")
eval_report(y_test, y_pred_rf, y_prob_rf, "Random Forest")
eval_report(y_test, y_pred_xgb, y_prob_xgb, "XGBoost")
```

```
=== Logistic Regression ===
Accuracy: 0.735
Precision: 0.7946
Recall: 0.7479
F1 Score: 0.7706
ROC-AUC: 0.8079
Confusion Matrix:
 [[58 23]
 [30 89]]

=== Random Forest ===
Accuracy: 0.77
Precision: 0.8288
Recall: 0.7731
F1 Score: 0.8
ROC-AUC: 0.8459
Confusion Matrix:
 [[62 19]
 [27 92]]

=== XGBoost ===
Accuracy: 0.79
Precision: 0.8667
Recall: 0.7647
F1 Score: 0.8125
ROC-AUC: 0.858
Confusion Matrix:
 [[67 14]
 [28 91]]
```

As per the above evaluation, we can see that XGBoost has the most accuracy and precision. So now we'll do feature importance for XGBoost

```
import xgboost as xgb
xgb.plot_importance(xgb_final, importance_type='gain', max_num_features=20)
plt.title("XGBoost Feature Importance (Gain)")
plt.show()
```



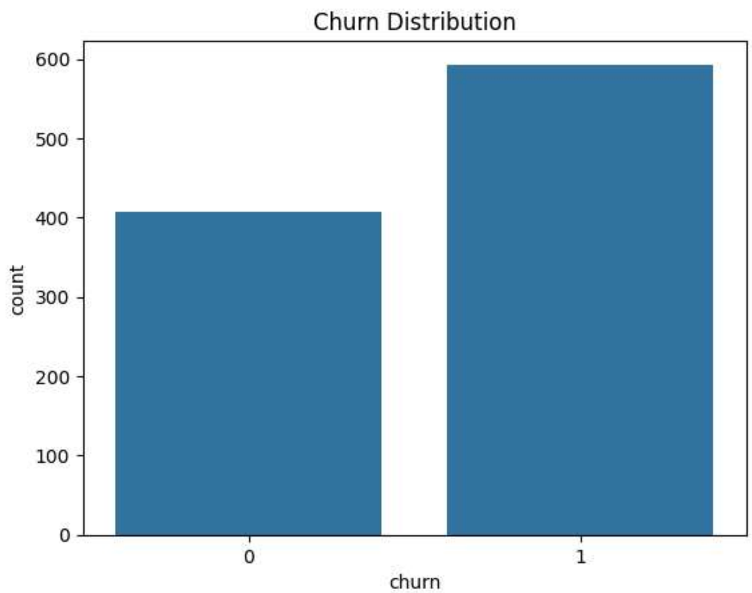

```
#prediction of top 20 customers who are most likely to churn
final_results = pd.DataFrame({
    'customer_id': cid_test.values,
    'actual_churn': y_test.values,
    'predicted_churn': y_pred_xgb,
    'churn_probability': y_prob_xgb
})
top_churners = final_results.sort_values(by='churn_probability', ascending=False).reset_index(drop=True)
top_churners.head(20)
```

	customer_id	actual_churn	predicted_churn	churn_probability
0	C00978	1	1	0.998749
1	C00831	1	1	0.998031
2	C00874	1	1	0.997370
3	C00695	1	1	0.997316
4	C00604	1	1	0.997300
5	C00223	1	1	0.997279
6	C00188	1	1	0.996610
7	C00205	1	1	0.995920
8	C00854	1	1	0.995815
9	C00379	1	1	0.995759
10	C00576	1	1	0.995724
11	C00813	1	1	0.995177
12	C00340	1	1	0.993665
13	C00464	1	1	0.993460
14	C00380	1	1	0.993040
15	C00227	0	1	0.992648
16	C00433	1	1	0.992552
17	C00302	1	1	0.992124
18	C00289	1	1	0.990299
19	C00210	1	1	0.988677

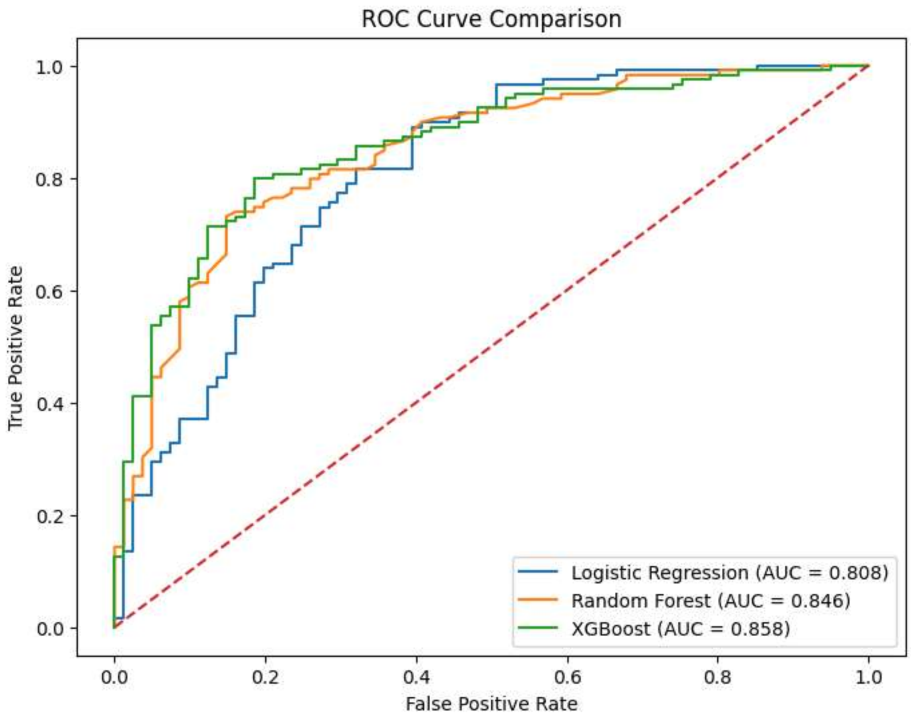
```
#saving models and results
joblib.dump(log_reg, "log_reg_model.joblib")
joblib.dump(rf, "rf_model.joblib")
joblib.dump(xgb_final, "xgb_model.joblib")
joblib.dump(scaler, "scaler.joblib")
final_results.to_csv("final_churn_predictions.csv", index=False)
```

```
#saving into the system
from google.colab import files

files.download("log_reg_model.joblib")
files.download("rf_model.joblib")
files.download("xgb_model.joblib")
files.download("scaler.joblib")
files.download("final_churn_predictions.csv")
```

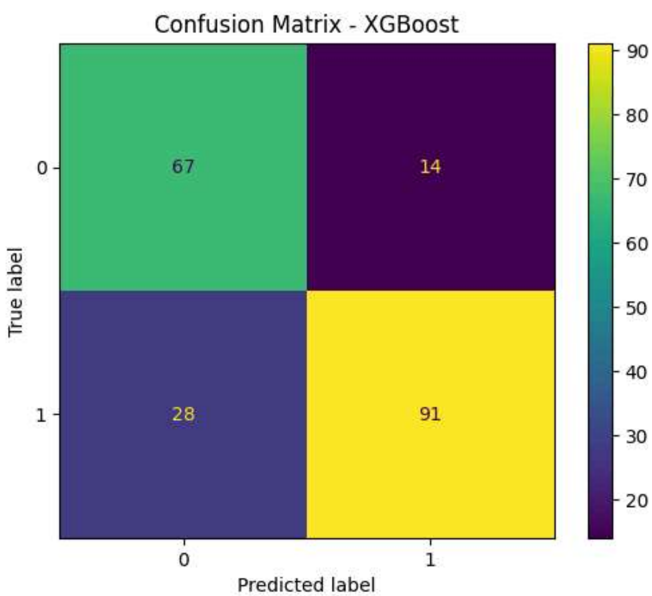
```
#churn distribution
sns.countplot(data=df, x='churn')
plt.title("Churn Distribution")
plt.show()
```



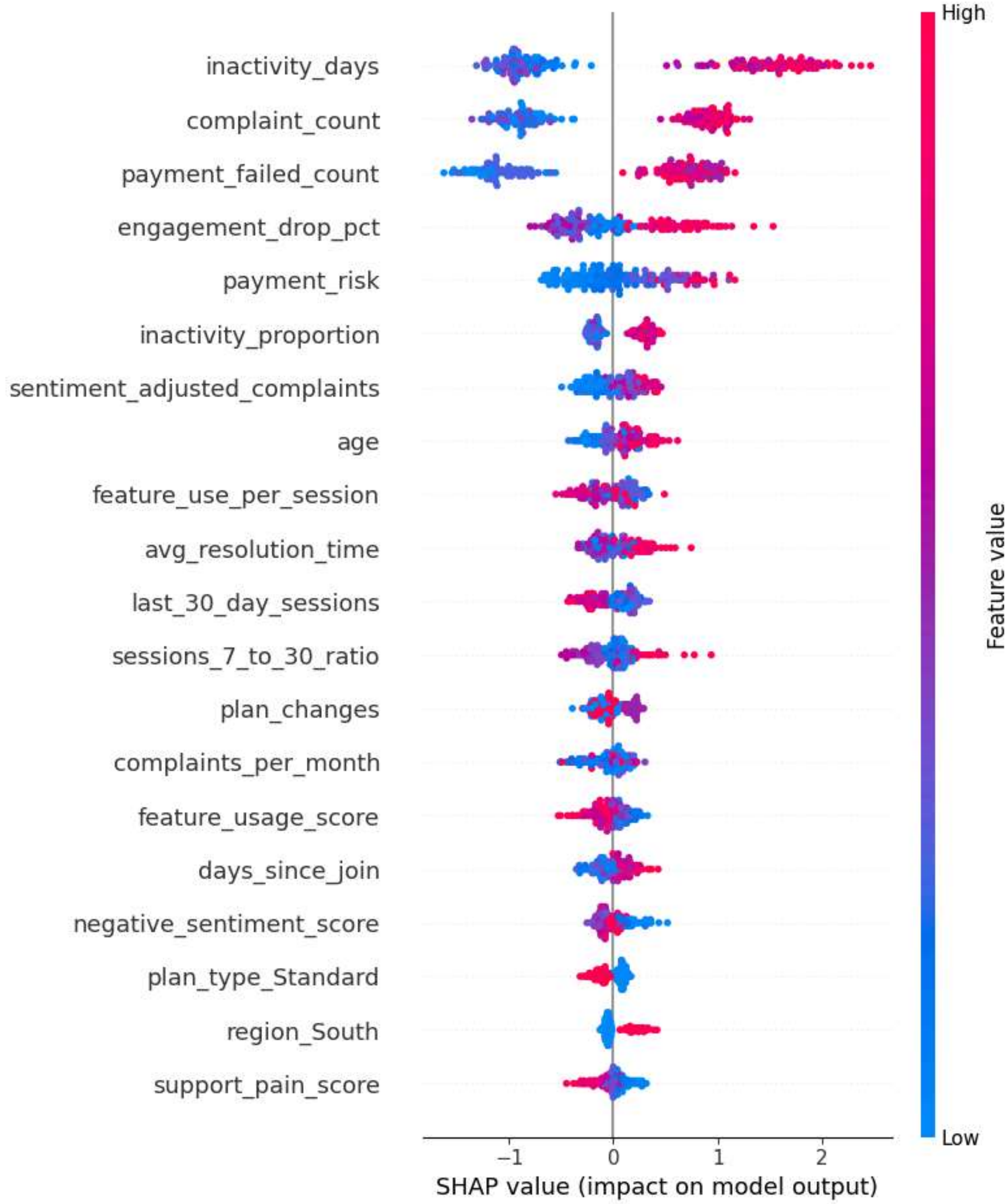

```
#ROC curve comparison
from sklearn.metrics import roc_curve, auc

models = {
    "Logistic Regression": (y_test, y_prob_lr),
    "Random Forest": (y_test, y_prob_rf),
    "XGBoost": (y_test, y_prob_xgb)
}

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

for model_name, (true, prob) in models.items():
    fpr, tpr, _ = roc_curve(true, prob)
    auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {auc_score:.3f})")

plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.show()
```



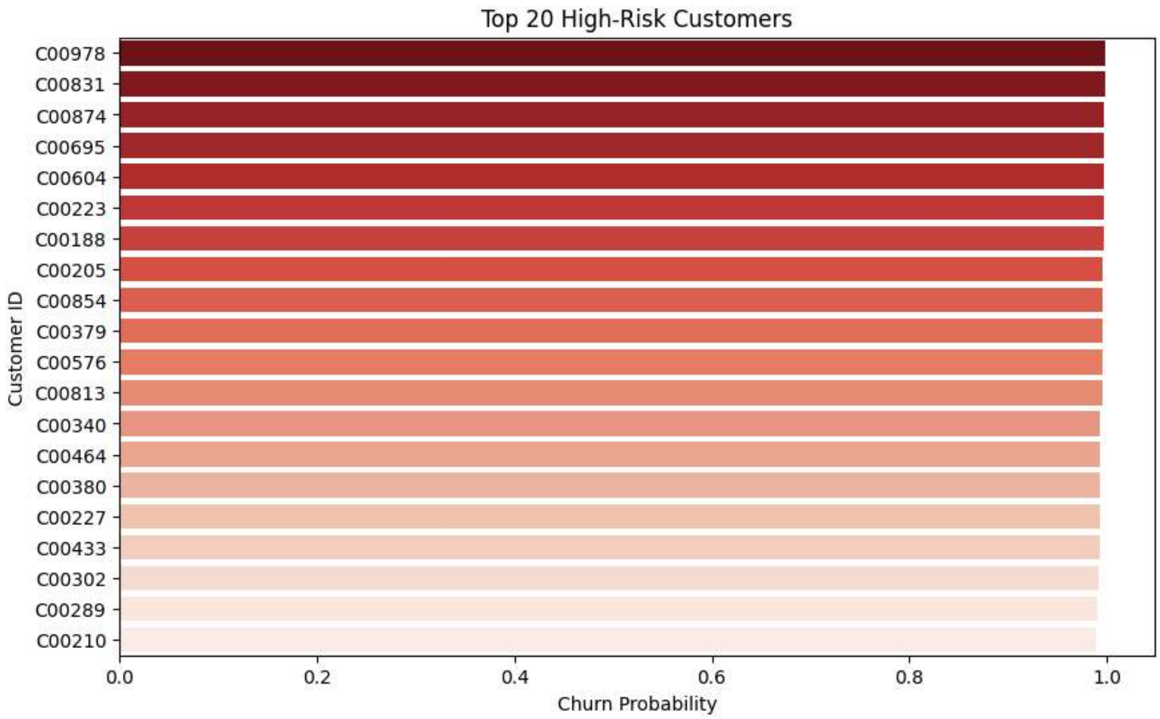

```
#XGBoost confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(xgb_final, X_test_scaled, y_test)
plt.title("Confusion Matrix - XGBoost")
plt.show()
```




```
#shap summary plot
import shap

explainer = shap.TreeExplainer(xgb_final)
shap_values = explainer.shap_values(X_test_scaled)

shap.summary_plot(shap_values, X_test, plot_type="dot")
```




```
#top churners
top20 = final_results.sort_values(by="churn_probability", ascending=False).head(20)

plt.figure(figsize=(10,6))
sns.barplot(x="churn_probability", y="customer_id", data=top20, palette="Reds_r")
plt.title("Top 20 High-Risk Customers")
plt.xlabel("Churn Probability")
plt.ylabel("Customer ID")
plt.show()
```




```
# Export the full cleaned + feature engineered dataset
df_with_id = df.copy()   # your final DF with customer_id preserved
df_with_id.to_csv("full_feature_dataset.csv", index=False)
```

```
final_results.to_csv("final_churn_predictions.csv", index=False)
```