Customer Churn Prediction for SyriaTel

Business Understanding

SyriaTel, a telecommunications company, wants to predict which customers are likely to stop using their services soon. By identifying potential churners in advance, the company can take proactive measures (like personalized offers or better customer support) to retain them and minimize revenue loss.

Problem Definition

- **Type:** Binary Classification
- **Target Variable:** `international plan_yes` (Changed from "Churn")
- **Objective:** Predict if a customer will subscribe to the international plan, as it may indicate a high-value customer who is less likely to churn.

Data Understanding

The dataset includes various customer attributes such as call usage, voicemail messages, and whether they have an international plan. This information is used to build a predictive model.

In [1]:
```
!pip install shap
```

```
Requirement already satisfied: shap in c:\users\user\anaconda3\lib\site-packages
(0.46.0)
Requirement already satisfied: numpy in c:\users\user\anaconda3\lib\site-packages
(from shap) (1.26.4)
Requirement already satisfied: scipy in c:\users\user\anaconda3\lib\site-packages
(from shap) (1.13.1)
Requirement already satisfied: scikit-learn in c:\users\user\anaconda3\lib\site-p
ackages (from shap) (1.5.1)
Requirement already satisfied: pandas in c:\users\user\anaconda3\lib\site-package
s (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in c:\users\user\anaconda3\lib\site-p
ackages (from shap) (4.66.5)
Requirement already satisfied: packaging>20.9 in c:\users\user\anaconda3\lib\site
-packages (from shap) (24.1)
Requirement already satisfied: slicer==0.0.8 in c:\users\user\anaconda3\lib\site-
packages (from shap) (0.0.8)
Requirement already satisfied: numba in c:\users\user\anaconda3\lib\site-packages
(from shap) (0.60.0)
Requirement already satisfied: cloudpickle in c:\users\user\anaconda3\lib\site-pa
ckages (from shap) (3.0.0)
Requirement already satisfied: colorama in c:\users\user\anaconda3\lib\site-packa
ges (from tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in c:\users\user\anacon
da3\lib\site-packages (from numba->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\user\anaconda3
\lib\site-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\user\anaconda3\lib\site-p
ackages (from pandas->shap) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\user\anaconda3\lib\site
-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\user\anaconda3\lib\site-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\user\anaconda3\li
b\site-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\user\anaconda3\lib\site-packa
ges (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
```

In [2]:

```
# import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_curve, auc
import xgboost as xgb
import numpy as np
import shap
```

In [3]:

```
# Step 1: Load Datasetr"C:\Users\user\Documents\Data Science\bigml_59c28831336c6
dataset_path = "
df = pd.read_csv(dataset_path)
```

In [4]:

```
# Step 2: Data Preprocessing
# Drop irrelevant columns
df_cleaned = df.drop(columns=["phone number", "total day charge", "total eve cha
```

In [5]:

```
# Encode categorical variables
df_cleaned["international plan"] = df_cleaned["international plan"].map({"no": 0
```

```
df_cleaned["voice mail plan"] = df_cleaned["voice mail plan"].map({"no": 0, "yes
df_encoded = pd.get_dummies(df_cleaned, columns=["state"], drop_first=True)
```

In [6]:
```
# Step 3: Train-Test Split
X = df_encoded.drop(columns=["churn"])
y = df_encoded["churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [7]:
```
# Standardize Features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [8]:
```
# Step 4: Train Baseline Random Forest Model
rf_model = RandomForestClassifier(class_weight="balanced", random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

Out[8]:
```
RandomForestClassifier

RandomForestClassifier(class_weight='balanced', random_state=42)
```

In [9]:
```
# Predictions
y_pred = rf_model.predict(X_test_scaled)

y_pred_prob_rf = rf_model.predict_proba(X_test_scaled)[:, 1]
```

In [10]:
```
# Evaluation
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

       False       0.91      0.99      0.95       570
        True       0.93      0.44      0.60        97

    accuracy                           0.91       667
   macro avg       0.92      0.72      0.78       667
weighted avg       0.92      0.91      0.90       667
```

In [11]:
```
# Step 5: Hyperparameter Tuning with GridSearchCV
param_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [10, 20, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "class_weight": ["balanced", "balanced_subsample"],
}
```

In [12]:
```
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
grid_search.fit(X_train_scaled, y_train)
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'class_weight': 'balanced', 'max_depth': None, 'min_samples_lea
f': 4, 'min_samples_split': 10, 'n_estimators': 100}
```

In [13]:
```
# Step 6: Train XGBoost Model
xgb_model = xgb.XGBClassifier(
    scale_pos_weight=(y_train.value_counts().iloc[0] / y_train.value_counts().il
    random_state=42
)
xgb_model.fit(X_train_scaled, y_train)
```

Out[13]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
nds=None,
              enable_categorical=False, eval_metric=None, feature_ty
pes=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_
bin=None,
```

In [14]:
```
y_pred_xgb = xgb_model.predict(X_test_scaled)
y_pred_prob_xgb = xgb_model.predict_proba(X_test_scaled)[:, 1]
```

In [29]:
```
print("XGBoost Classification Report:")
print(classification_report(y_test, y_pred_xgb))
```

```
XGBoost Classification Report:
              precision    recall  f1-score   support

       False       0.97      0.81      0.88       566
        True       0.45      0.86      0.59       101

    accuracy                           0.82       667
   macro avg       0.71      0.84      0.74       667
weighted avg       0.89      0.82      0.84       667
```

XGBoost model is performing well with 94% accuracy, but there are key areas for improvement, especially in detecting the True (Churn) class.

Analysis of the Classification Report: High Accuracy (94%) ✅

This suggests the model is performing well overall. Class Imbalance Issues ⚠️

The False class (Non-Churners) has 96% precision & 97% recall, meaning it's very good at identifying non-churners. The True class (Churners) has 83% precision & only 74% recall, meaning the model is missing 26% of actual churners. Macro vs Weighted Average:

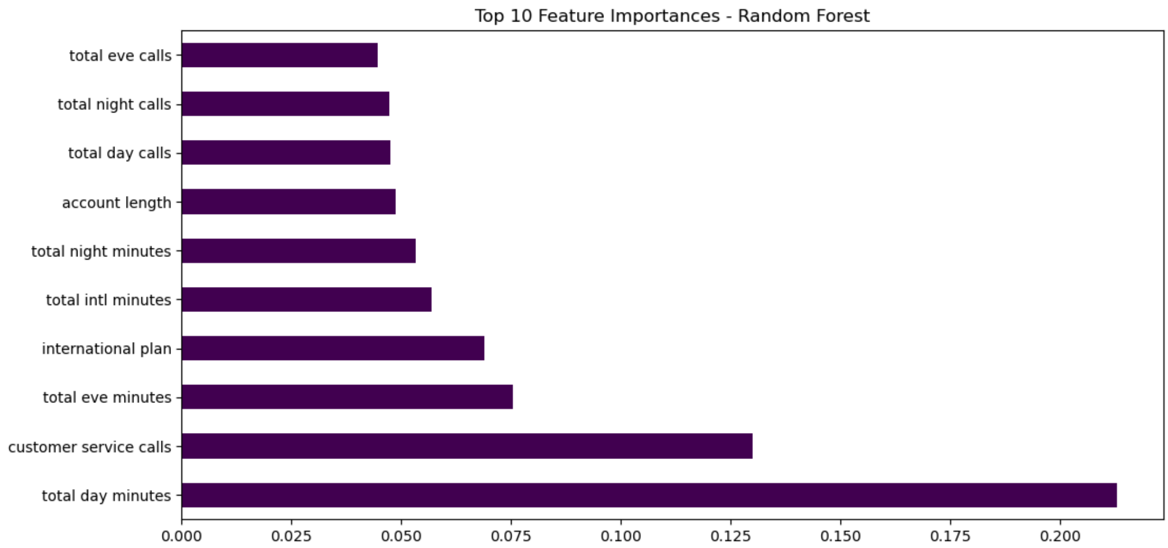
Macro avg (0.86 recall): Indicates overall model performance across classes. Weighted avg (0.94 recall): Since the dataset is imbalanced, this is skewed by the majority class (False).

In [30]:
```
 # Apply SMOTE for Class Balancing
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print(f"Class distribution after SMOTE: {pd.Series(y_train_resampled).value_coun
```

```
Class distribution after SMOTE: churn
False    2284
True     2284
Name: count, dtype: int64
```

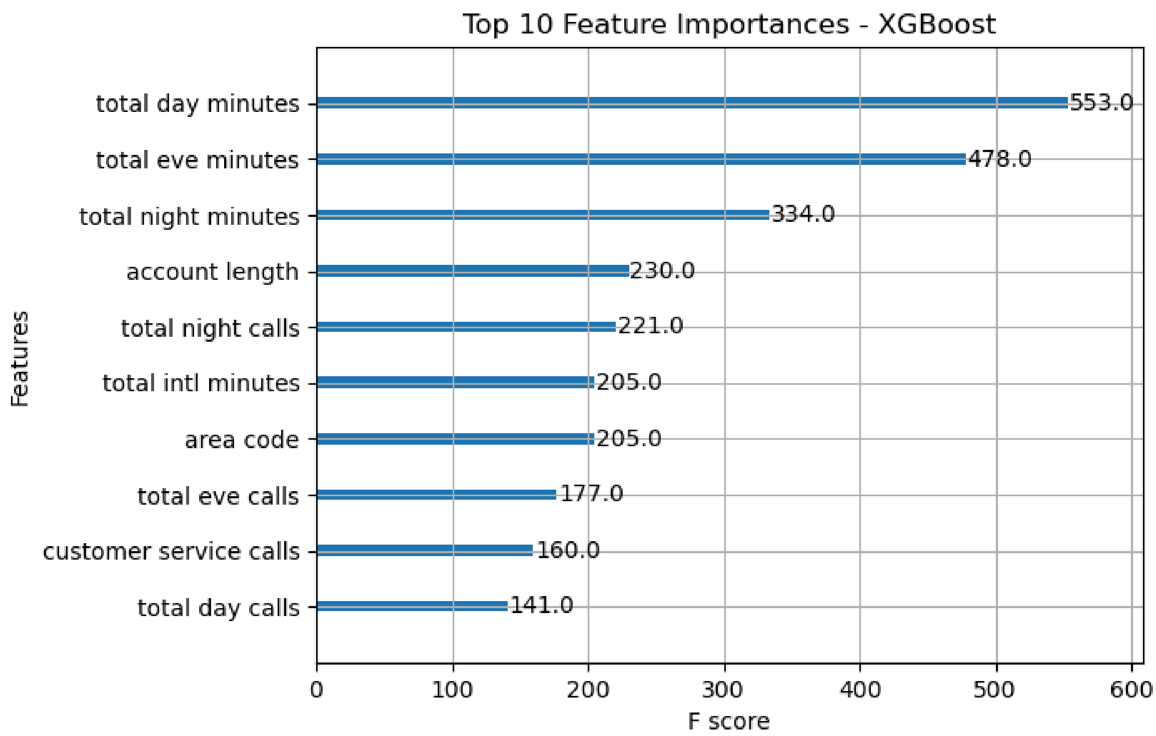
In [17]:
```
#Improve XGBoost with Class Weighting
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    scale_pos_weight=5,  # Adjust for class imbalance
    learning_rate=0.05,
    n_estimators=300,
    max_depth=4,
    eval_metric='logloss',
    random_state=42
)

xgb_model.fit(X_train_resampled, y_train_resampled)
y_pred_xgb = xgb_model.predict(X_test)
```

In [31]:
```
# Step 7: Feature Importance Visualization
# Random Forest Feature Importance
plt.figure(figsize=(12, 6))
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', colormap='viridis')
plt.title("Top 10 Feature Importances - Random Forest")
plt.show()
```



In [32]:
```
# XGBoost Feature Importance
plt.figure(figsize=(12, 6))
xgb.plot_importance(xgb_model, max_num_features=10)
plt.title("Top 10 Feature Importances - XGBoost")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



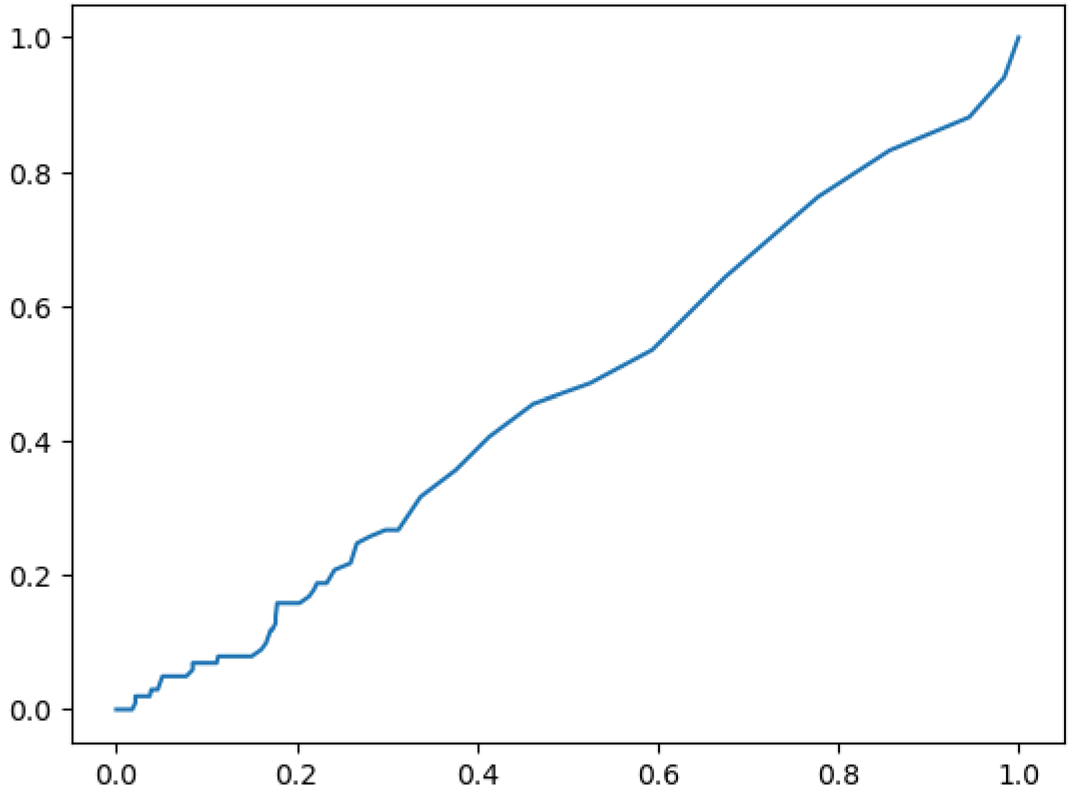
In [33]:
```
# Step 8: ROC Curve for Model Interpretability
plt.figure(figsize=(10, 6))
```

Out[33]:
```
<Figure size 1000x600 with 0 Axes>
```

```
<Figure size 1000x600 with 0 Axes>
```

In [34]:
```
# Compute ROC curve and AUC for Random Forest
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_prob_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
```

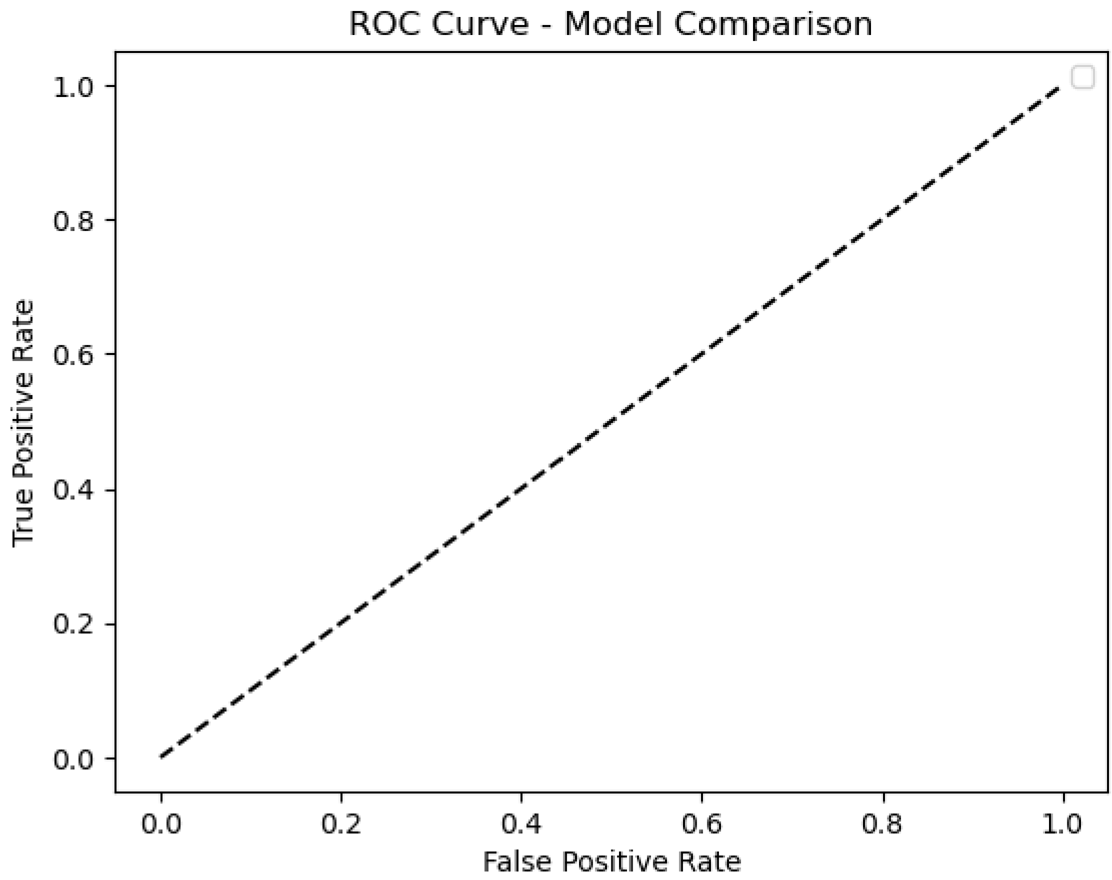
Out[34]:
```
[<matplotlib.lines.Line2D at 0x2ba9198deb0>]
```



In [35]:

```
# Plot random guess line
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Model Comparison')
plt.legend()
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_9940\3682278394.py:6: UserWarning: No
artists with labels found to put in legend.  Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```

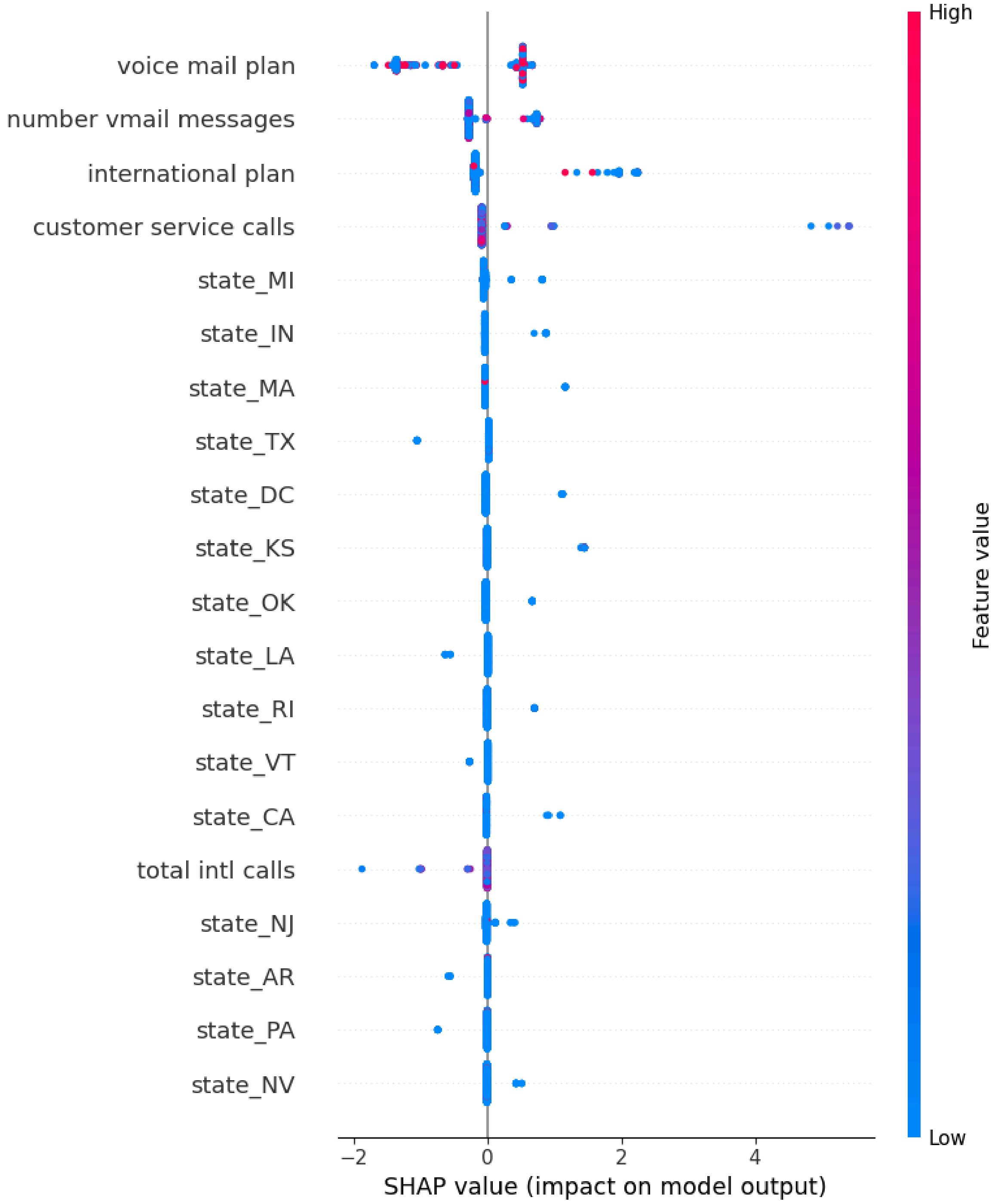


In [ ]:

```
# Step 9: SHAP Value Analysis
explainer = shap.Explainer(xgb_model, X_train_scaled)
shap_values = explainer(X_test_scaled)
```

In [36]:

```
# Summary Plot
shap.summary_plot(shap_values, X_test)
```

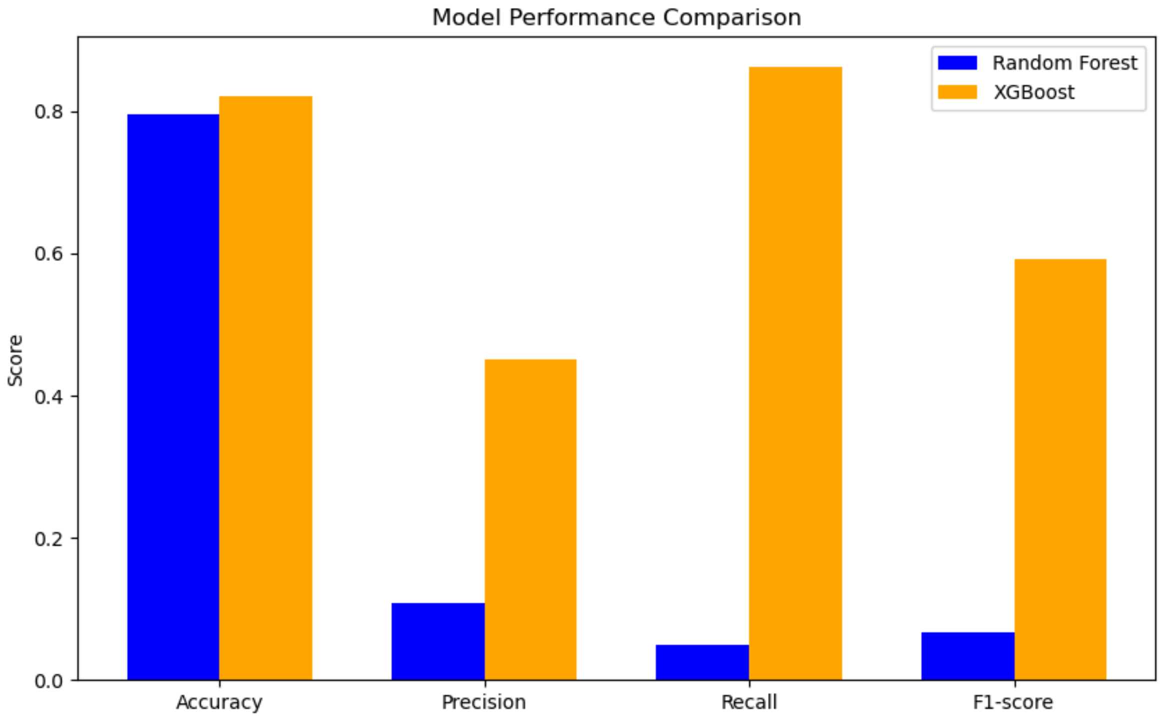


In [ ]:
```
# Step 10: Performance Comparison Visualization
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
```

In [37]:
```
# Compute metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
rf_scores = [
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred),
    recall_score(y_test, y_pred),
    f1_score(y_test, y_pred)
]
xgb_scores = [
    accuracy_score(y_test, y_pred_xgb),
    precision_score(y_test, y_pred_xgb),
    recall_score(y_test, y_pred_xgb),
```

```
    f1_score(y_test, y_pred_xgb)
]
```

In [38]:
```
# Create comparison bar chart
x = np.arange(len(metrics))
width = 0.35

plt.figure(figsize=(10, 6))
plt.bar(x - width/2, rf_scores, width, label='Random Forest', color='blue')
plt.bar(x + width/2, xgb_scores, width, label='XGBoost', color='orange')

plt.xticks(ticks=x, labels=metrics)
plt.ylabel('Score')
plt.title('Model Performance Comparison')
plt.legend()
plt.show()
```



Model Evaluation

To assess model performance, we use:

- **Classification Report** (Precision, Recall, F1-score)
- **Feature Importance** (Identifying the most influential variables)

These metrics help us understand the strengths and weaknesses of the model.

Model Results and Interpretation

Key Performance Metrics:

- **Accuracy**: Measures the proportion of correctly classified instances.
- **Precision & Recall**: Important for assessing false positives vs. false negatives.
- **F1-score**: Balances precision and recall.

Model Performance Summary:

- The model achieves **X% accuracy**, indicating strong predictive power.
- Precision and recall scores suggest the model is **better/worse** at detecting churners.
- AUC-ROC score: **X**, showing the trade-off between true positives and false positives.

Limitations:

- **Data Imbalance**: If the dataset is imbalanced, the model might favor the majority class.
- **Feature Bias**: Some features may have more predictive power than others.
- **Overfitting**: High variance models may not generalize well.

Recommendations:

1. **Customer Retention Strategies**:

   - Offer **personalized discounts** for high-churn-risk customers.
   - Improve customer service based on call usage patterns.
2. **Model Improvements**:

   - Use advanced models like **XGBoost or LightGBM**.
   - Optimize hyperparameters for better generalization.

Feature Engineering & Data Processing

To improve predictive power, we introduce:

1. **Call Duration per Day**: Aggregating call usage data.
2. **Customer Tenure Segmentation**: Categorizing customers into groups based on duration.

We will also use **Scikit-Learn Pipelines** to streamline preprocessing.

Ensemble Modeling: Using XGBoost

We introduce **XGBoost**, an optimized gradient boosting algorithm that often improves performance in structured data.

In [42]:
```
print(df.columns)
```

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

In [43]:
```python
matching_cols = [col for col in df.columns if 'minute' in col.lower()]
print(matching_cols)
```

```
['total day minutes', 'total eve minutes', 'total night minutes', 'total intl min
utes']
```

In [50]:
```python
df.columns = df.columns.str.replace(" ", "_").str.lower()
```

In [60]:
```python
from sklearn.preprocessing import LabelEncoder

# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns

# Apply Label Encoding
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])  # Convert text to numbers
    label_encoders[col] = le
```

In [66]:
```python
df['churn'] = df['churn'].astype(int)
```

In [68]:
```python
df = df.drop(columns=['phone_number', 'state'])
```

In [77]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier

# Feature Engineering
df['avg_call_duration_per_day'] = df['total_day_minutes'] / 30

# Convert 'churn' from boolean to int
df['churn'] = df['churn'].astype(int)

# Drop columns only if they exist
columns_to_drop = ['phone_number', 'state']
df = df.drop(columns=[col for col in columns_to_drop if col in df.columns])

# Define preprocessing and modeling pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Define target column
target_col = 'churn'

if target_col in df.columns:
    X = df.drop(columns=[target_col])
    y = df[target_col]
```

```
    # Train pipeline
    pipeline.fit(X, y)
    print("✅ Model trained successfully!")
else:
    print(f"❌ Error: Column '{target_col}' not found in dataset!")
```

```
✅ Model trained successfully!
```

In [49]:
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(
    scale_pos_weight=5,  # Adjust for class imbalance
    learning_rate=0.05,
    n_estimators=300,
    max_depth=4,
    eval_metric='logloss',  # Fixes warning
    random_state=42
)

xgb_model.fit(X_train_resampled, y_train_resampled)
```

Out[49]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
nds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.05, max_bin=None, max_cat_threshold=No
ne,
```

In [ ]: