Introduction

Customer churn is the phenomenon where customers stop using a company's products or servic
poses a significant challenge to businesses in highly competitive industries like telecommunicati
Accurate predictions of customers who are likely to churn enables companies like SyriaTel to take
proactive retention measures. This thereby reduces revenue loss and improves customer satisfact
project focuses on building a machine learning model to predict customer churn using a publicly
available dataset from SyriaTel.

We aim to develop a predictive model that not only achieves high accuracy, but also balances pr
and recall, especially for identifying the minority class (churners). Multiple models were evaluatec
including logistic regression and random forest with threshold tuning and advanced ensemble m
like XGBoost, to determine the most effective solution for churn detection.

Import Libraries

In [1]:
```
# Import necessary 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 StandardScaler
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_recall_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve,
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    roc_curve,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)
```

Load and Inspect Data

In [2]:
```
# Load data
df= pd.read_csv("bigml.csv")
```

In [3]:
```
# Check the first five rows of the data set
df.head()
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...

5 rows × 21 columns

In [4]:

```
# Check details of the data frame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [5]:

```
# Check the statistical summary of the data set(numerical columns)
df.describe()
```

Out[5]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tc r
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.

In [6]:

```
# Check for duplicates
df.duplicated().sum()
```

Out[6]: 0

In [7]:

```
# Check for missing values
df.isnull().sum()
```

Out[7]:

```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [8]:

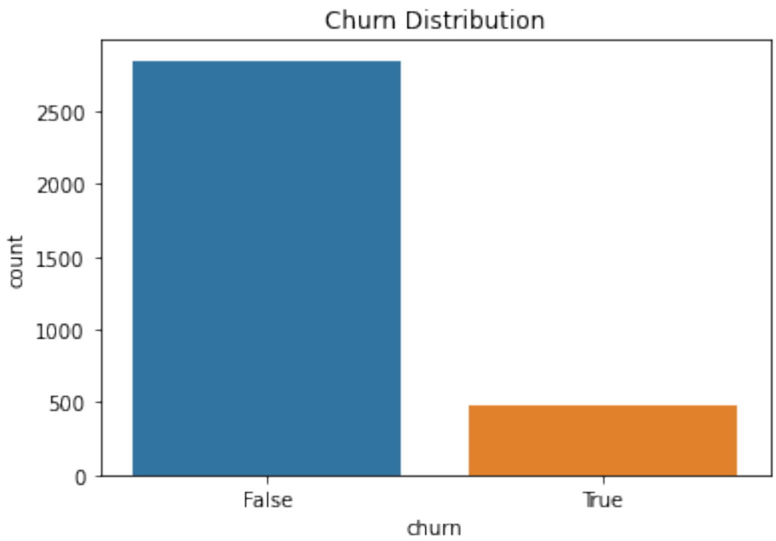
```
# Convert all to snake_case
df.columns= (
    df.columns
    .str.lower()
    .str.replace(' ', '_')
)
df.columns= df.columns.str.replace('total_', '')
# Replace abbreviations with complete names
df.columns= df.columns.str.replace('vmail', 'voice_mail').str.replace('intl', 'i
```

In [9]:
```
# Check column names
df.columns
```

Out[9]:
```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_voice_mail_messages',
       'day_minutes', 'day_calls', 'day_charge', 'eve_minutes', 'eve_calls',
       'eve_charge', 'night_minutes', 'night_calls', 'night_charge',
       'international_minutes', 'international_calls', 'international_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

Exploratory Data Analysis

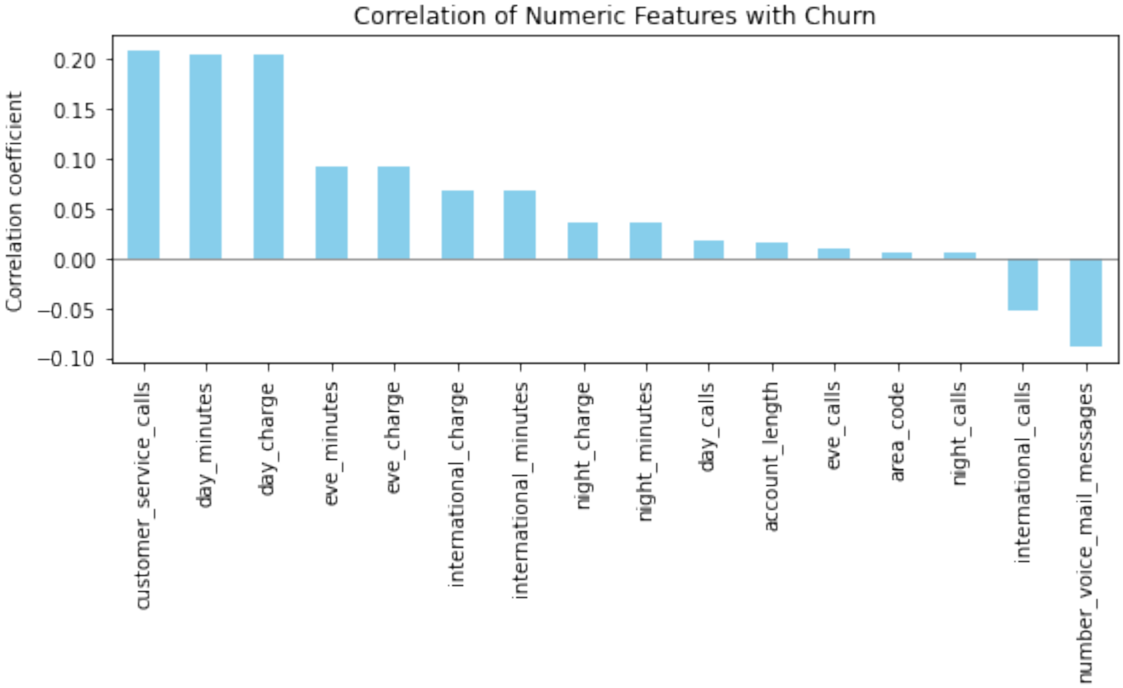
In [10]:
```
# Churn distribution
sns.countplot(data=df, x='churn')
plt.title("Churn Distribution")
plt.show()
```



In [11]:
```
# Check for correlation for numerical features
# Select numeric columns only
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Add churn column which is our target variable since it is not numeric
numeric_df['churn'] = df['churn']

# Compute correlation matrix
corr_matrix = numeric_df.corr()

# Sort by correlation with Churn
churn_corr = corr_matrix['churn'].drop('churn').sort_values(ascending=False)

# Display top correlated features
print("Correlation of numeric features with Churn:\n")
print(churn_corr)

# Plot bar chart of correlation with churn
plt.figure(figsize=(8, 5))
churn_corr.plot(kind='bar', color='skyblue')
plt.title("Correlation of Numeric Features with Churn")
plt.ylabel("Correlation coefficient")
plt.axhline(0, color='gray', linewidth=0.8)
```

```
plt.tight_layout()
plt.show()
```

```
<ipython-input-11-3b2647089454>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
user_guide/indexing.html#returning-a-view-versus-a-copy
  numeric_df['churn'] = df['churn']
Correlation of numeric features with Churn:

customer_service_calls       0.208750
day_minutes                  0.205151
day_charge                   0.205151
eve_minutes                  0.092796
eve_charge                   0.092786
international_charge         0.068259
international_minutes        0.068239
night_charge                 0.035496
night_minutes                0.035493
day_calls                    0.018459
account_length               0.016541
eve_calls                    0.009233
area_code                    0.006174
night_calls                  0.006141
international_calls         -0.052844
number_voice_mail_messages  -0.089728
Name: churn, dtype: float64
```



Data Preprocessing

In [12]:
```
# Binary encoding for churn
df['churn']=df['churn'].replace({False: 0, True: 1})
```

In [13]:
```
# Drop customer identifier columns
df.drop(columns=['phone_number'], inplace=True, errors= 'ignore')
```

In [14]:
```
# Convert binary (yes & no) to numerical (1 & 0) for the entire dataset
df.replace({'yes': 1, 'no': 0}, inplace= True)
```

In [15]:
```
# Check first five rows
df.head()
```

Out[15]:

	state	account_length	area_code	international_plan	voice_mail_plan	number_voice_mail_mess
0	KS	128	415	0	1	
1	OH	107	415	0	1	
2	NJ	137	415	0	0	
3	OH	84	408	1	0	
4	OK	75	415	1	0	

In [16]:
```
categorical_cols= df.select_dtypes(include=['object']).columns
categorical_cols
```

Out[16]:
```
Index(['state'], dtype='object')
```

In [17]:
```
encoded_df= pd.get_dummies(df, columns= categorical_cols, drop_first=True, dtype
encoded_df.head()
```

Out[17]:

	account_length	area_code	international_plan	voice_mail_plan	number_voice_mail_messages
0	128	415	0	1	25
1	107	415	0	1	26
2	137	415	0	0	0
3	84	408	1	0	0
4	75	415	1	0	0

5 rows × 69 columns

In [18]:
```
# Identify x and y variables
X=encoded_df.drop(columns=['churn'], axis=1)
y=encoded_df['churn']
```

In [19]:
```
# Feature Engineering
df['total_minutes'] = df['day_minutes'] + df['eve_minutes'] + df['night_minutes'
df['total_calls'] = df['day_calls'] + df['eve_calls'] + df['night_calls'] + df['
df['total_charge'] = df['day_charge'] + df['eve_charge'] + df['night_charge'] +
df['avg_call_duration'] = df['total_minutes'] / (df['total_calls'] + 1e-5)
df['high_service_calls'] = (df['customer_service_calls'] > 3).astype(int)
df['voicemail_use_ratio'] = df['number_voice_mail_messages'] / (df['voice_mail_p
df['charge_per_call'] = df['total_charge'] / (df['total_calls'] + 1e-5)
df['minutes_per_call'] = df['total_minutes'] / (df['total_calls'] + 1e-5)
```

In [20]:
```
# Split data into train and test
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2, random_s
```

In [21]:
```
# Standardize data using StandardScaler
scaler= StandardScaler()
```

```
scaled_x_train= scaler.fit_transform(X_train)
scaled_x_test= scaler.transform(X_test)
```

Model Building

1. Logistic Regression

Logistic Regression (Baseline)

In [22]:
```
logreg = LogisticRegression()
logreg.fit(scaled_x_train, y_train)
y_pred_lr = logreg.predict(scaled_x_test)

print("Logistic Regression Classification Report")
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Classification Report
              precision    recall  f1-score   support

           0       0.87      0.97      0.92       566
           1       0.58      0.21      0.31       101

    accuracy                           0.86       667
   macro avg       0.73      0.59      0.61       667
weighted avg       0.83      0.86      0.83       667
```

Improved Logistic Regression Model with Polynomial Features and Threshold Tuning

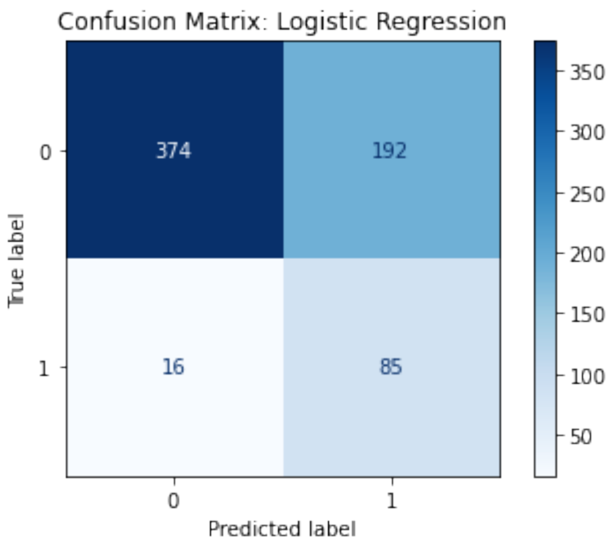
In [23]:
```
# Generate Polynomial Features
poly = PolynomialFeatures(degree=1, include_bias=False, interaction_only=False)
X_train_poly = poly.fit_transform(scaled_x_train)
X_test_poly = poly.transform(scaled_x_test)

# Train Logistic Regression
logreg = LogisticRegression(max_iter=1000, class_weight='balanced')
logreg.fit(X_train_poly, y_train)

# Predict Probabilities
y_probs = logreg.predict_proba(X_test_poly)[:, 1]

# Tune Threshold
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

target_precision = 0.80
target_recall = 0.90
best_threshold = 0.40
best_f1 = 0.80

for p, r, t in zip(precision, recall, thresholds):
    if r >= target_recall and p >= target_precision:
        f1 = 2 * (p * r) / (p + r)
        if f1 > best_f1:
            best_f1 = f1
            best_threshold = t

print(f"\n Best Threshold: {best_threshold:.2f} | F1: {best_f1:.2f}")

# Step 12: Final Predictions
```

```
y_pred = (y_probs >= best_threshold).astype(int)

# Step 13: Evaluation
print("\n Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Best Threshold: 0.40 | F1: 0.80

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.66      0.78       566
           1       0.31      0.84      0.45       101

    accuracy                           0.69       667
   macro avg       0.63      0.75      0.62       667
weighted avg       0.86      0.69      0.73       667
```

Confusion Matrix For Logistic Regression Model

In [24]:
```
# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix: Logistic Regression')
plt.show()
```



Interpretation

The baseline logistic regression model is a good model for predicting non churners and a poor n predicting churners. The baseline model does not align to our business goal which is capturing c customers. Recall for class 1(churners) is at 0.21, meaning that 79% of the actual churners have n identified. This is a risk to the business.

The improved version of the logistic regression model has a recall of 0.84, making it suitable for t business. Though it has a low precision of 0.31, it is still quite suitable for the business whose rete offers such as discounts are not costly and losing a customer is costly.

2. Random Forest

Random Forest (Baseline)

In [25]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Random Forest Classification Report")
print(classification_report(y_test, y_pred_rf))
```

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

           0       0.93      1.00      0.97       566
           1       1.00      0.60      0.75       101

    accuracy                           0.94       667
   macro avg       0.97      0.80      0.86       667
weighted avg       0.94      0.94      0.93       667
```

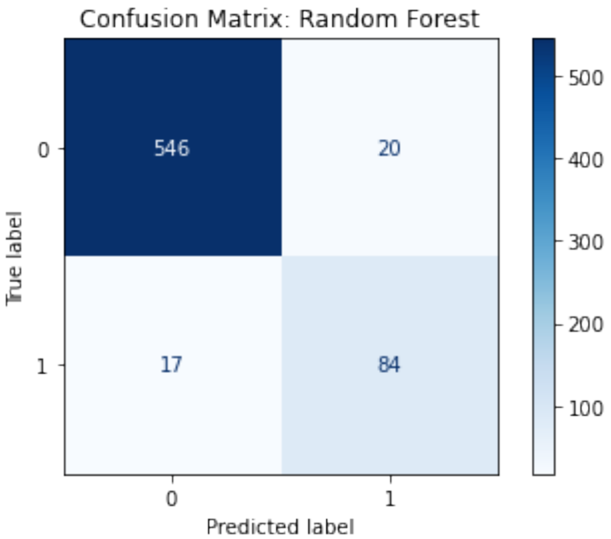
Random Forest Model with Class Weights and Thresholds

In [26]:
```
# Random Forest Model with Class Weight
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
rf.fit(X_train, y_train)

# Predict Probabilities
y_probs = rf.predict_proba(X_test)[:, 1]

# Precision-Recall Curve and Threshold Optimization
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Find the best threshold with a good recall and precision
target_precision = 0.80
target_recall = 0.92

best_threshold = 0.30
best_f1 = 0.85

for p, r, t in zip(precision, recall, thresholds):
    if r >= target_recall and p >= target_precision:
        f1 = 2 * (p * r) / (p + r)
        if f1 > best_f1:
            best_f1 = f1
            best_threshold = t

print(f"Best threshold: {best_threshold:.2f} with F1: {best_f1:.3f}")

# Make Predictions with Optimal Threshold
y_pred_opt = (y_probs >= best_threshold).astype(int)

# Final Evaluation
print("\nClassification Report at Optimized Threshold:")
print(classification_report(y_test, y_pred_opt))
```

```
Best threshold: 0.30 with F1: 0.850

Classification Report at Optimized Threshold:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       566
           1       0.81      0.83      0.82       101

    accuracy                           0.94       667
   macro avg       0.89      0.90      0.89       667
weighted avg       0.95      0.94      0.94       667
```

Confusion Matrix for Random Forest

In [27]:
```python
# Compute the confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_opt)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf)
disp_rf.plot(cmap='Blues')
plt.title("Confusion Matrix: Random Forest")
plt.show()
```



Interpretation

The baseline model is good at predicting non churners and bad at predicting churners. For the cl
has a recall 0f 0.60. This means that 40% of actual churners have not been accurately captured. T
the baseline model won't be a good fit for our business since it does not align with the business
which is capturing churners.

The improved version of our model has a good balance of precision and recall. It has a high prec
0.81, therefore, most of the loyal customers won't be mistaken for churners quite often. It has a h
of 0.83, therefore, out of all actual churners, 83% of them are accurately identified. The improved
of the random forest classifier, is therefore a good fit for our business.

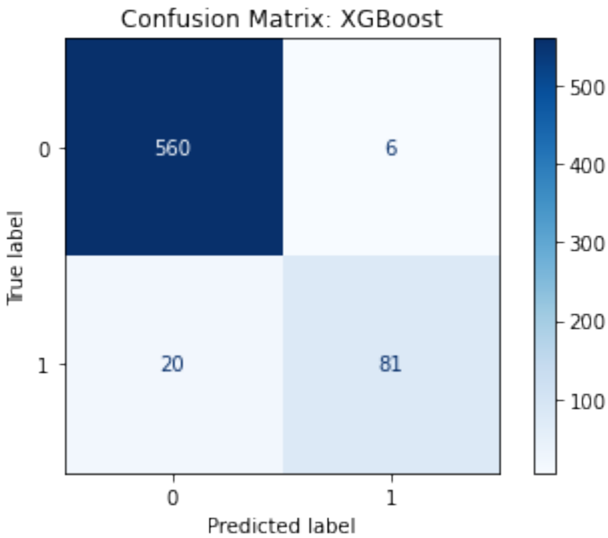
3. XGBoost

In [28]:
```python
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

```
print("XGBoost Classification Report")
print(classification_report(y_test, y_pred_xgb))
```

```
[16:26:00] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.0\src\learner.cc:516:
Parameters: { use_label_encoder } might not be used.

  This may not be accurate due to some parameters are only used in language bindings
  passed down to XGBoost core.  Or some parameters are not used but slip through thi:
  verification. Please open an issue if you find above cases.


XGBoost Classification Report
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       566
           1       0.93      0.80      0.86       101

    accuracy                           0.96       667
   macro avg       0.95      0.90      0.92       667
weighted avg       0.96      0.96      0.96       667
```

In [29]:
```
# Compute the confusion matrix
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_xgb)
disp_rf.plot(cmap='Blues')
plt.title("Confusion Matrix: XGBoost")
plt.show()
```



Interpretation

This model catches 80% of churners, making it a good model for our business. It has high precisi 93%. This means that it does not wrongly target loyal customers often. Having F1 score of 0.86 m there is harmonic balance between precision and recall. The model is balanced and fit for the bu:

Findings and Recommendation

Based on the comparison of the three models evaluated for customer churn prediction using the dataset, the XGBoost model emerges as the best performer. While earlier models optimized usin different thresholds (0.40 and 0.30) showed significant improvements in F1-score — with the 0.3(

threshold achieving an F1-score of 0.85 — the XGBoost model surpasses them by achieving an e higher F1-score of 0.86 for the minority class (churners), with precision at 0.93 and recall at 0.80. indicates that the model is not only good at correctly identifying customers who are likely to chu also does so with very few false positives, which is crucial in a business context to avoid unneces customer retention costs.

Furthermore, the XGBoost model also achieves overall accuracy of 96%, and its macro and weigh average F1-scores are the highest among the three evaluations. This suggests it performs well ac both classes, despite the class imbalance in the dataset. Importantly, the model provides a strong between precision and recall, which is essential for a churn prediction problem — where both fal negatives (missed churners) and false positives (non-churners flagged incorrectly) can have serio financial implications.

In conclusion, the XGBoost classifier is the final recommended model for deployment in SyriaTel's prediction system. It offers excellent performance in identifying churners while maintaining stron accuracy, and its robustness to imbalanced data, combined with superior handling of feature inte makes it a powerful choice for this business problem.