SyriaTel-Churn-Analysis / Untitled.ipynb

RoyceBett Completed Notebook daf41ff · 12 minutes ago

2402 lines (2402 loc) · 396 KB

Preview Code Blame Raw

SyriaTel Churn Analysis

1. Business Understanding

Business Problem

Phone companies require a steady base of customers. There are often new customers but a way to maximize the user base is to maintain existing customers i.e. keep them from churning. The aim of this exercise is to attempt to understand what causes customers to churn in order to optimize operations to keep as many customers as possible from churning.

Objectives

1. Establish probable reasons for churning.
2. Develop a prediction model for customers likely to churn.
3. Suggest ways to curb churning.

2. Data Understanding

In [504…

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMa
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
```

In [505…

```
# load and preview dataset
data = pd.read_csv('syriatel_data.csv')
data.head()
```

Out[505…

	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 [506…

```
# dataframe summary
data.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
```

3. Data Preparation

Data Preprocessing

In [507…

```
# check for duplicates
data.duplicated().sum()
```

Out[507…

```
0
```

In [508…

```
# check for missing values
data.isna().sum()
```

Out[508…

```
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
```

From the data understanding section we see that not all columns are made up of integers. As such, not every column can be parsed by a machine learning model. Some columns will need to be encoded in order to be usable.

In [509…

```
# Mapping dictionaries for conversion
intl_plan = {'yes': 1, 'no': 0}
vm_plan = {'yes': 1, 'no': 0}
churn_status = {True: 1, False: 0}

# Replacing values using map()
data['international plan'] = data['international plan'].map(intl_plan)
data['voice mail plan'] = data['voice mail plan'].map(vm_plan)
data['churn'] = data['churn'].map(churn_status)
```

In [510…

```
data.head()
```

Out[510…

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	0	1	25	265.1	110	45.07
1	OH	107	415	371-7191	0	1	26	161.6	123	27.47
2	NJ	137	415	358-1921	0	0	0	243.4	114	41.38
3	OH	84	408	375-9999	1	0	0	299.4	71	50.90
4	OK	75	415	330-6626	1	0	0	166.7	113	28.34

5 rows × 21 columns

Some columns of data show different facets of the same value, thus would be inefficient to work with separately. It would be ideal to combine them before proceeding.

In [511…

```
# Combining and summing columns
data['total_charges'] = data[['total day charge', 'total eve charge', 'total nig
data['total_calls'] = data[['total day calls', 'total eve calls', 'total night c
data['total_minutes'] = data[['total day minutes', 'total eve minutes', 'total n
```

In [512…

```
data.head()
```

Out[512…

	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	0	1	25	265.1	110	45.07
1	OH	107	415	371-7191	0	1	26	161.6	123	27.47
2	NJ	137	415	358-1921	0	0	0	243.4	114	41.38
3	OH	84	408	375-9999	1	0	0	299.4	71	50.90
4	OK	75	415	330-6626	1	0	0	166.7	113	28.34

5 rows × 24 columns

From here on, there are a number of columns that would be either unusable, redundant or would not contribute significantly to the analysis. We can drop those.

In [513…

```
data = data.drop(['phone number','total day minutes', 'total eve minutes', 'tota
data.head()
```

Out[513…

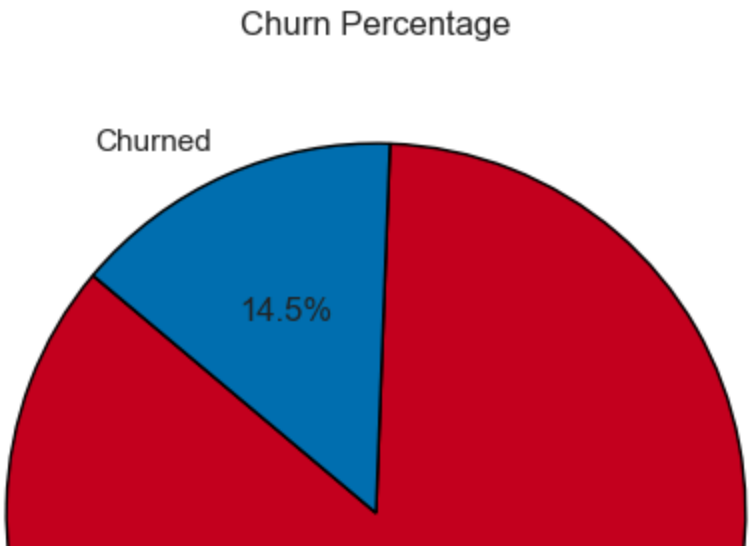
	state	account length	area code	international plan	voice mail plan	number vmail messages	customer service calls	churn	total_charges
0	KS	128	415	0	1	25	1	0	75.56
1	OH	107	415	0	1	26	1	0	59.24
2	NJ	137	415	0	0	0	0	0	62.29
3	OH	84	408	1	0	0	2	0	66.80
4	OK	75	415	1	0	0	3	0	52.09

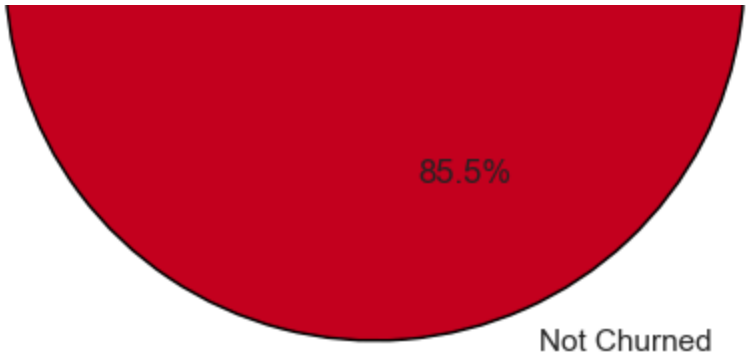
Exploratory Data Analysis

Univariate Analysis

In [514…

```
# Calculate the distribution of churn
churn_counts = data['churn'].value_counts()
churn_labels = churn_counts.index.map({0: 'Not Churned', 1: 'Churned'})
churn_percentages = churn_counts / churn_counts.sum() * 100

# Create a pie chart
plt.figure(figsize=(6, 6))
plt.pie(churn_counts, labels=churn_labels, autopct='%1.1f%%', colors=['#c3001e',
        wedgeprops={'edgecolor': 'black'})

# Set the title
plt.title("Churn Percentage")
plt.show()
```



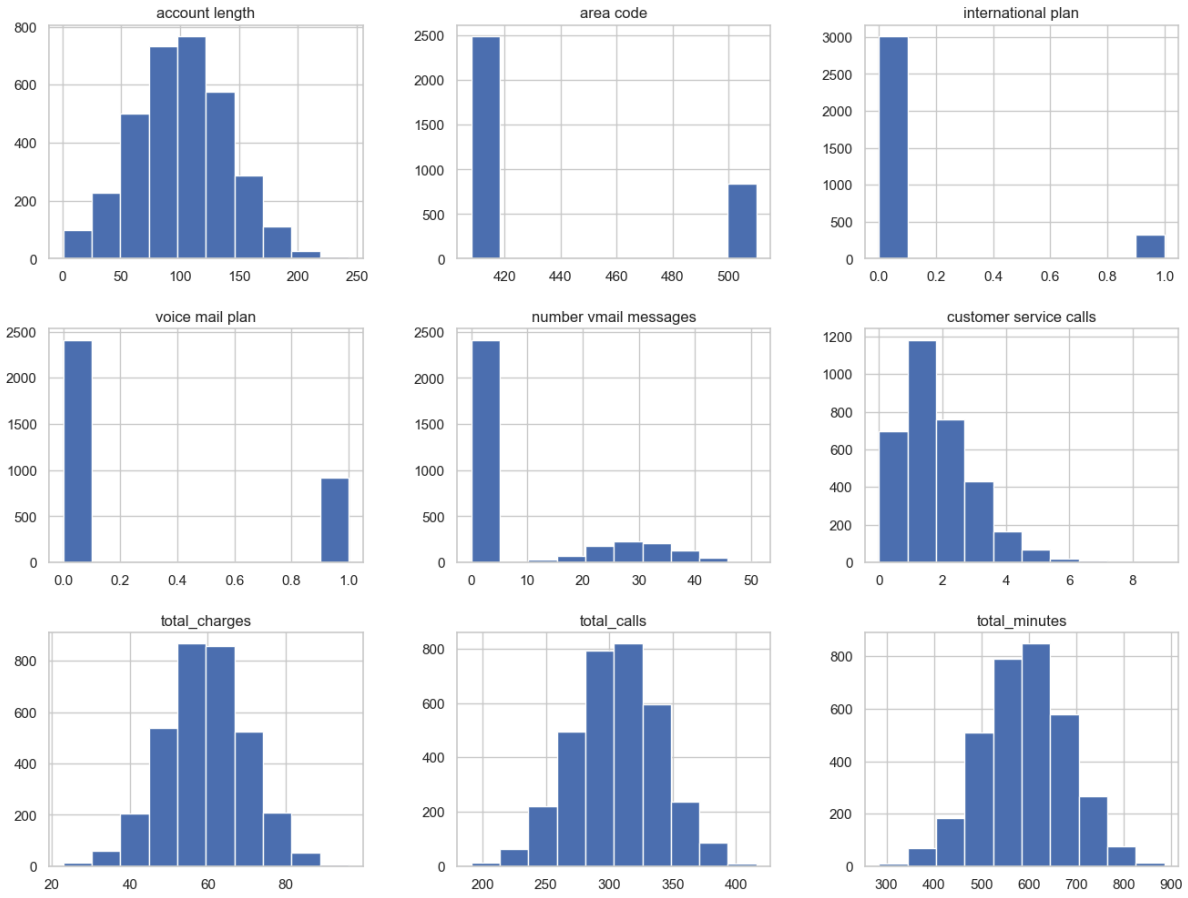


We observe that 14.5% of all the customers have churned as per the data available to us.

This indicates a class imbalance.

In [515…

```
# distribution of features
data.drop(columns='churn').hist(figsize=(16,12));
```

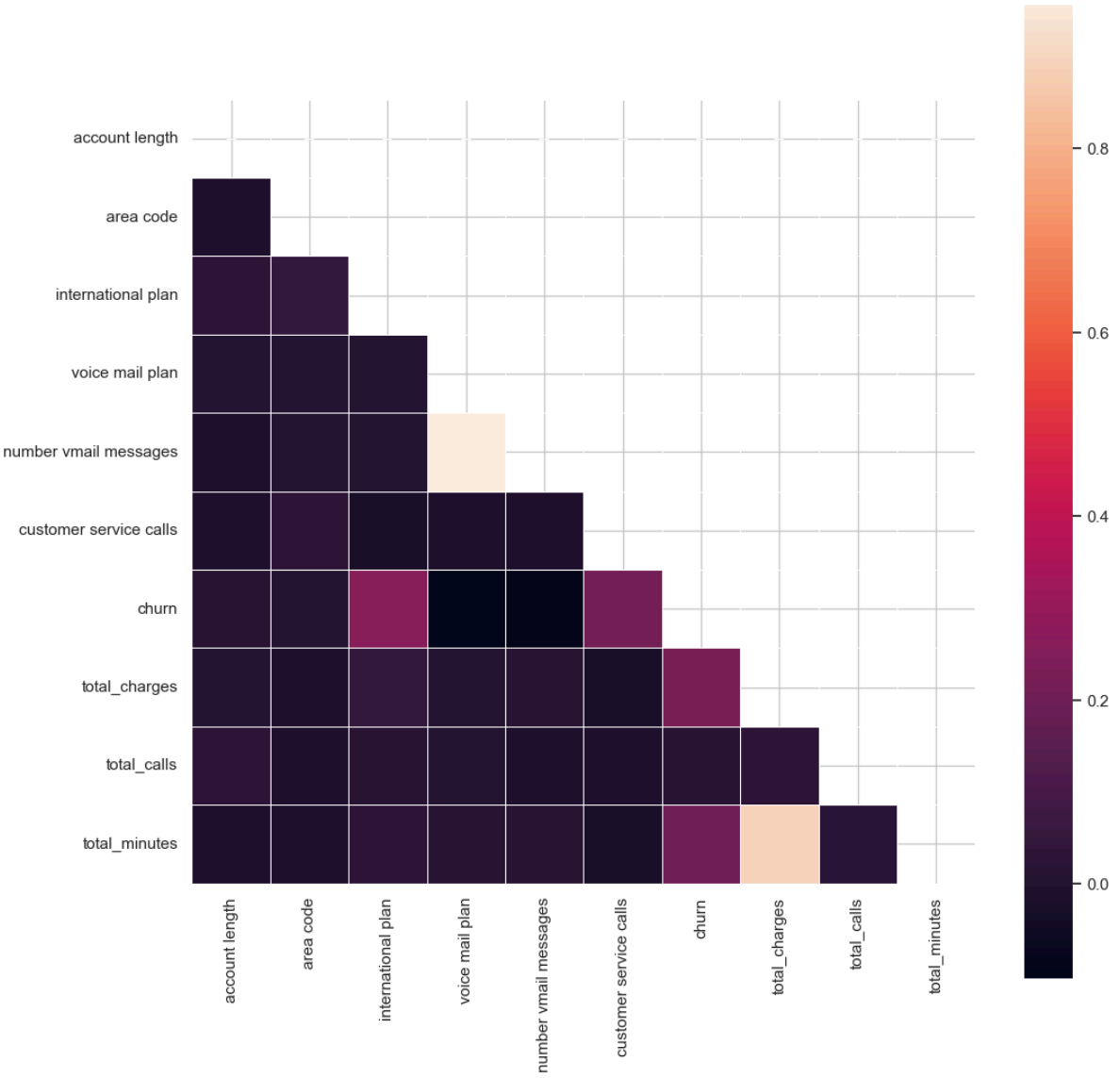


Multivariate Analysis

In [516…

```
# Step 1: Select only numeric columns
numeric_df = data.select_dtypes(include=['float64', 'int64'])

#using a heatmap to show correlation
```

```
fig, ax = plt.subplots(figsize=(12,12))
mask = np.triu(np.ones_like(numeric_df.corr(), dtype=bool))
sns.heatmap(numeric_df.corr(), linewidths=0.5, mask=mask, square=True, ax=ax, an
```



Key Takeaways from Correlation Matrix:

Total Calls: Strong positive correlations exist between call minutes, number of calls, and charges. This suggests customers who make more calls also spend more on those calls.

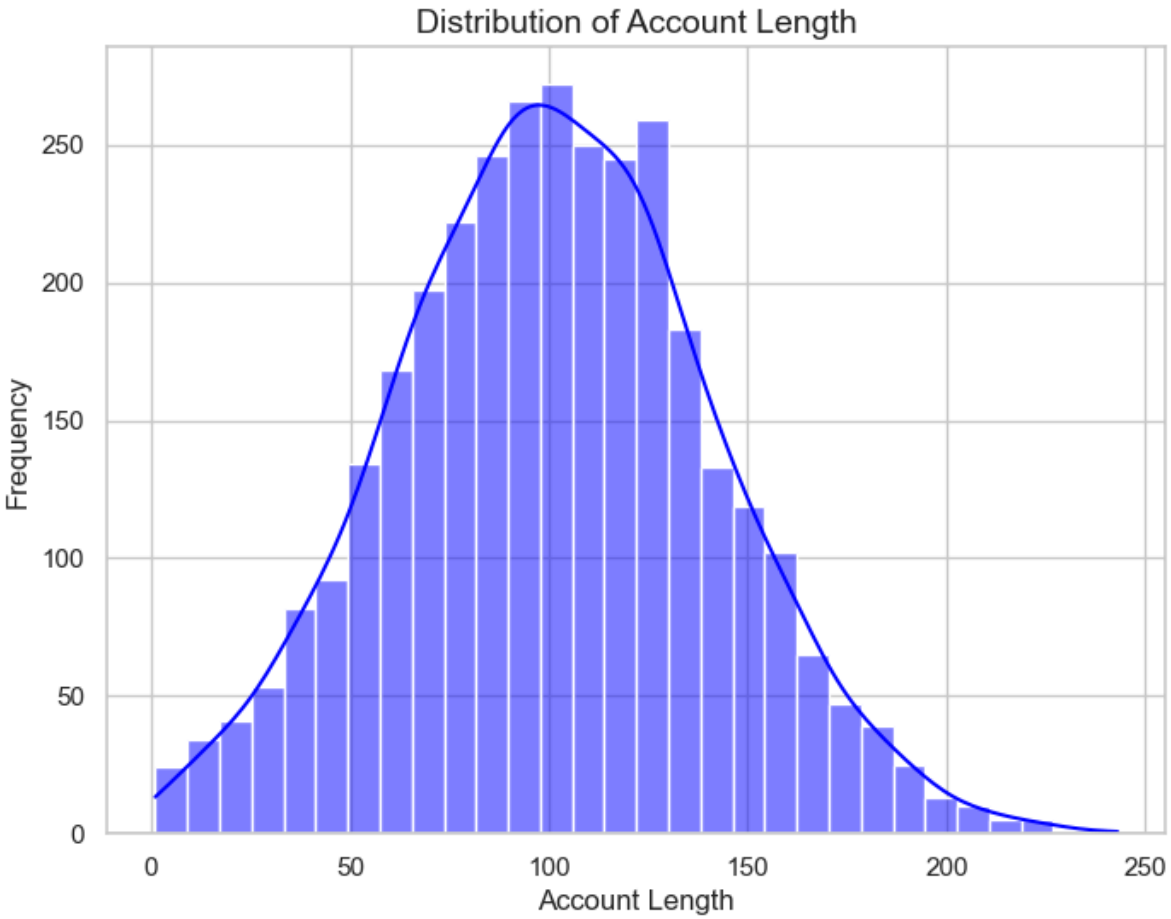
Account Length: A weak positive correlation exists between account length and total call minutes. This indicates that customers who have been with SyriaTel longer tend to make slightly more calls, but the effect is weak.

International Calls: A moderate positive correlation is seen between total international minutes, number of calls, and charges. Customers who make more international calls tend to spend more on them.

Customer Service Calls: This feature shows weak or no correlations with most others. The number of customer service calls seems unrelated to calling habits or account tenure.

In [517…

```
plt.figure(figsize=(8, 6))
sns.histplot(data['account length'], kde=True, color='blue', bins=30)
plt.title('Distribution of Account Length', fontsize=14)
plt.xlabel('Account Length', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(True)
plt.show()
```



4. Modelling

In [518…

```
# One-hot encode the 'state' column
data = pd.get_dummies(data, columns=['state', 'area code'], drop_first=True)

# Split data into features (X) and target (y)
X = data.drop('churn', axis=1)
y = data['churn']

data.head()
```

Out[518…

	account length	international plan	voice mail plan	number vmail messages	customer service calls	churn	total_charges	total_calls	t
0	128	0	1	25	1	0	75.56	303	
1	107	0	1	26	1	0	59.24	332	

2	137	0	0	0	0	0	62.29	333
3	84	1	0	0	2	0	66.80	255
4	75	1	0	0	3	0	52.09	359

5 rows × 61 columns

In [519…

```
# Initialize MinMaxScaler with the range [-1, 1]
scaler = MinMaxScaler(feature_range=(-1, 1))

# Select only the numerical columns
numerical_cols = data.select_dtypes(include=[np.number]).columns

# Apply scaling to the numerical columns
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

data.head()
```

Out[519…

	account length	international plan	voice mail plan	number vmail messages	customer service calls	churn	total_charges	total_calls
0	0.049587	-1.0	1.0	-0.019608	-0.777778	-1.0	0.437585	-0.004444
1	-0.123967	-1.0	1.0	0.019608	-0.777778	-1.0	-0.008194	0.253333
2	0.123967	-1.0	-1.0	-1.000000	-1.000000	-1.0	0.075116	0.262222
3	-0.314050	1.0	-1.0	-1.000000	-0.555556	-1.0	0.198306	-0.431111
4	-0.388430	1.0	-1.0	-1.000000	-0.333333	-1.0	-0.203496	0.493333

5 rows × 61 columns

Train-Test Split

In [520…

```
#select predictor and target variables
y = data['churn']
X = data.drop(columns=['churn'])
```

In [521…

```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

SMOTE

In [522…

```
# Initialize and apply SMOTE
```

```
smote = SMOTE(random_state=123)
resampled_X_train, resampled_y_train = smote.fit_resample(X_train, y_train)
```

Logistic Regression

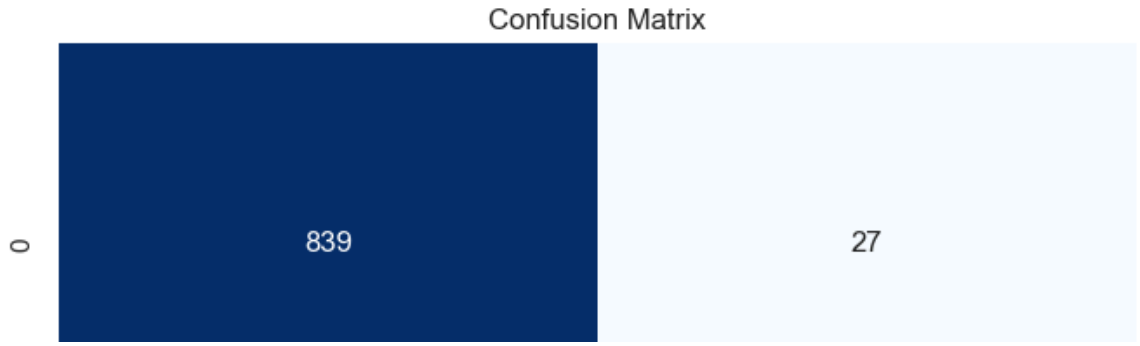
In [523…

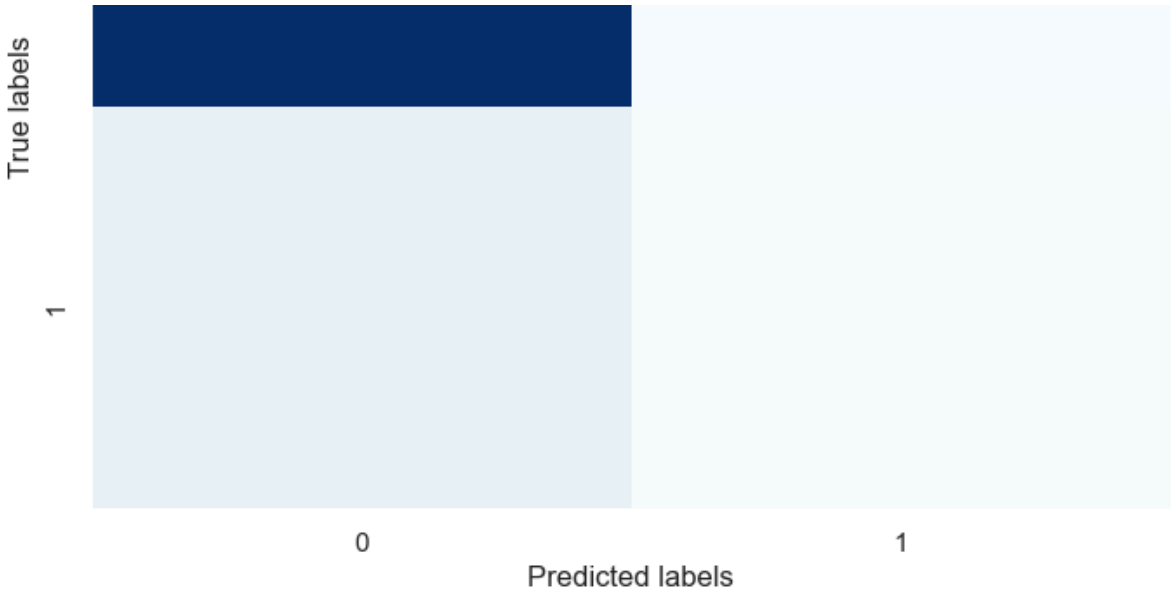
```
# Instantiate the Logistic Regression model
log_reg_model = LogisticRegression(random_state=42)

# Train the model
log_reg_model.fit(X_train, y_train)

# Predict on the test set
y_pred = log_reg_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.874

Classification Report:
              precision    recall  f1-score   support

        -1.0       0.89      0.97      0.93       866
         1.0       0.56      0.26      0.36       134

    accuracy                           0.87      1000
   macro avg       0.73      0.62      0.64      1000
weighted avg       0.85      0.87      0.85      1000
```

In [524…

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```





This model has an accuracy score of 0.87. The precision for predicting churn (1.0) is a relatively low 0.56. This shows that the model's ability to accurately identify churn cases is limited.

The recall for predicting churn is also low at 0.26, suggesting that the model misses a significant number of actual churn cases. These results show that the logistic regression model may not be the best choice for this task.

Decision Tree

In [525…

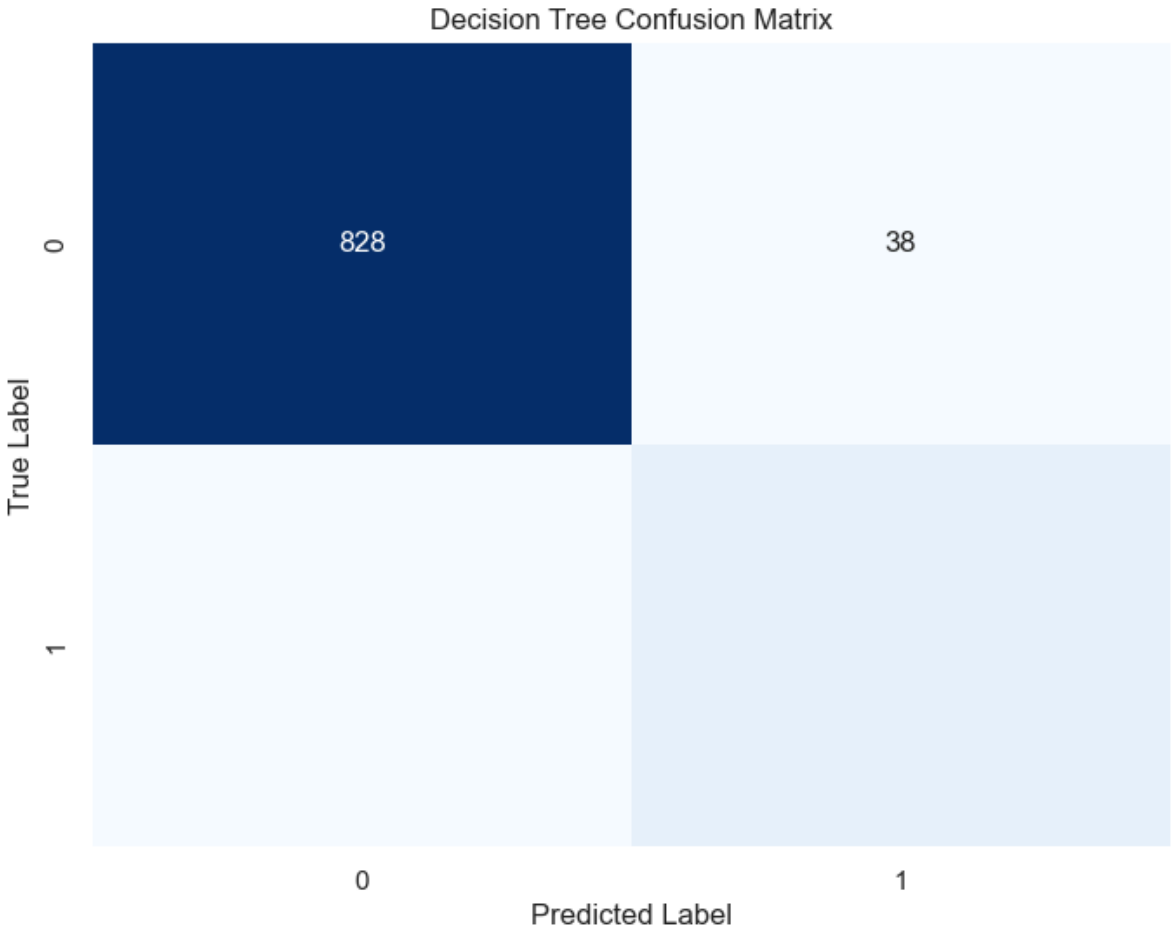
```
# Initialize the Decision Tree Classifier
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Train the model
decision_tree_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = decision_tree_model.predict(X_test)

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Accuracy:", accuracy_dt)

# Generate classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))
```

```
Accuracy: 0.919

Classification Report:
              precision    recall  f1-score   support

        -1.0       0.95      0.96      0.95       866
         1.0       0.71      0.68      0.69       134

    accuracy                           0.92      1000
```

```
           macro avg       0.83      0.82      0.82      1000
        weighted avg       0.92      0.92      0.92      1000
```

In [526…

```
# Generate confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



This model has an accuracy score of 0.92 and the precision for predicting churn (1.0) is 0.71. This shows that the model accurately identifies a significant portion of churn cases.

The recall for predicting churn is 0.68. This shows that the model captures a good chunk of actual churn cases. These results show that the decison tree model performs fairly well for this task.

Random Forest

In [527…

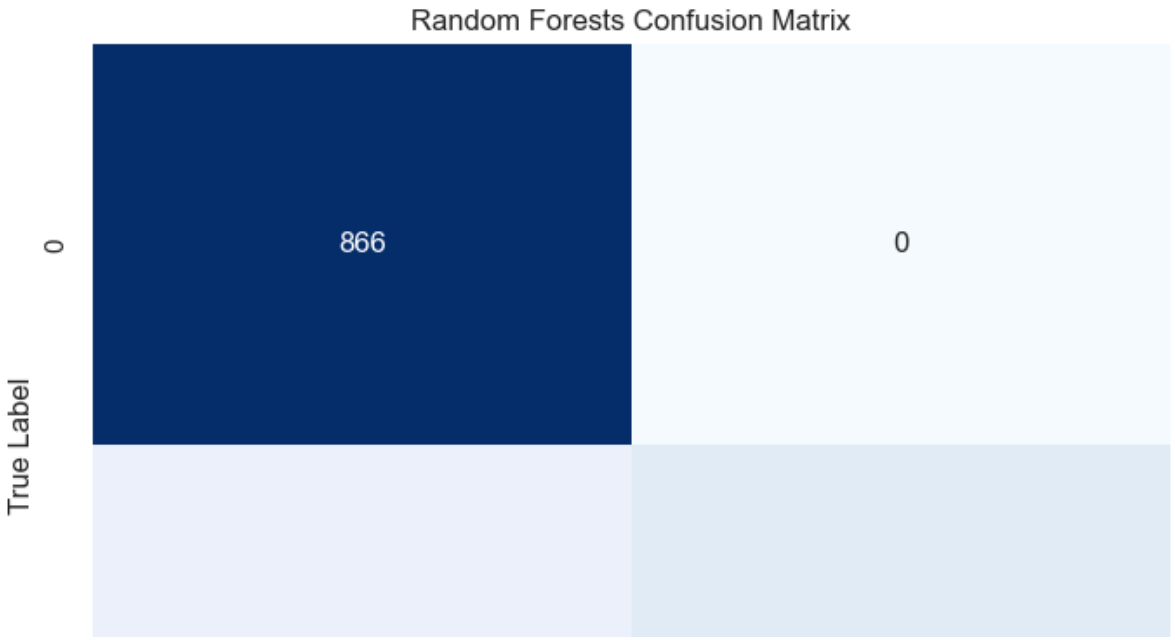
```
# Initialize Random Forest Classifier
```

```
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Calculate accuracy for Random Forests
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", accuracy_rf)

# Generate classification report for Random Forests
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.946

Classification Report:
              precision    recall  f1-score   support

        -1.0       0.94      1.00      0.97       866
         1.0       1.00      0.60      0.75       134

    accuracy                           0.95      1000
   macro avg       0.97      0.80      0.86      1000
weighted avg       0.95      0.95      0.94      1000
```

In [528…

```
# Generate confusion matrix for Random Forests
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Random Forests Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



This model has an accuracy score of 0.95 and the precision for predicting churn (1.0) is 1.0. This shows that the model accurately identifies all of the churn cases.

The recall for predicting churn is 0.60. This shows that the model captures a good chunk of actual churn cases. These results show that the decison tree model performs quite well for this task.

KNearest Neighbors

In [529…

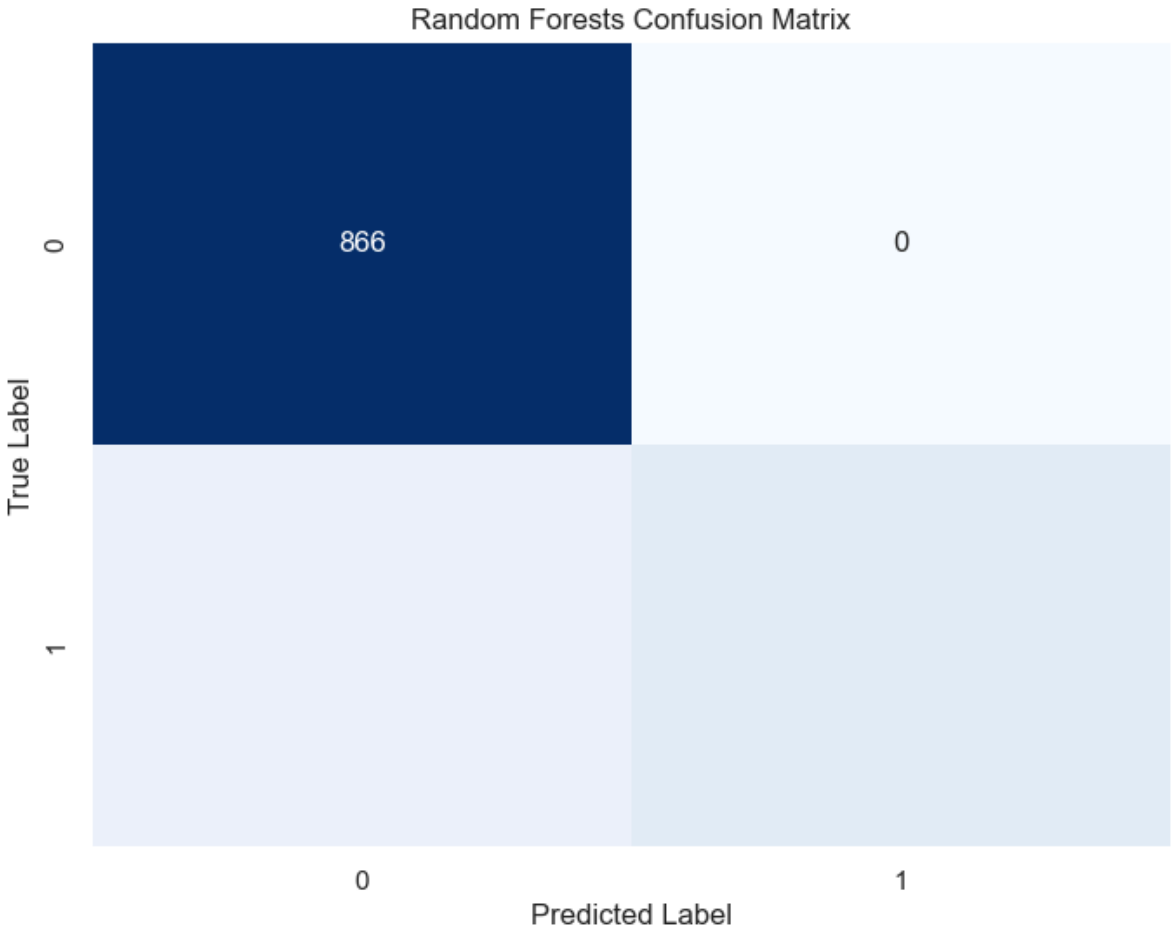
```
# Initialize KNN classifier
knn = KNeighborsClassifier()

# Train the model
knn.fit(X_train, y_train)

# Generate predictions
y_pred_knn = knn.predict(X_test)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Accuracy:", accuracy_knn)

# Generate classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_knn))
```

```
Accuracy: 0.862

Classification Report:
              precision    recall  f1-score   support

        -1.0       0.88      0.98      0.92       866
         1.0       0.45      0.13      0.20       134

    accuracy                           0.86      1000
   macro avg       0.66      0.55      0.56      1000
weighted avg       0.82      0.86      0.83      1000
```

In [530…

```
# Generate confusion matrix for Random Forests
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False)
```

```
plt.title("Random Forests Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



This model has an accuracy score of 0.86 and the precision for predicting churn (1.0) is 0.45. This shows that the model accurately identifies less than half of the churn cases.

The recall for predicting churn is 0.13. This shows that the model captures a very small proportion of actual churn cases. These results show that the decison tree model performs poorly for this task.

Evaluation

Hyperparameter Tuning

In [531…

```
# Define and train your models
logistic_regression_model = LogisticRegression()
decision_tree_model = DecisionTreeClassifier()
random_forest_model = RandomForestClassifier()
knn_model = KNeighborsClassifier()
```

In [532…

```
# Define the classifiers
```

```
classifiers = {
    "Logistic Regression": logistic_regression_model,
    "Decision Trees": decision_tree_model,
    "Random Forests": random_forest_model,
    "K-Nearest Neighbors": knn_model
}
```

In [533…

```
# Define a dictionary to store performance metrics
performance_metrics = {}
```

In [534…

```
# Evaluate each classifier using cross-validation
for clf_name, clf in classifiers.items():
    # Perform cross-validation
    cv_accuracy = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy
    cv_precision = cross_val_score(clf, X_train, y_train, cv=5, scoring='precisi
    cv_recall = cross_val_score(clf, X_train, y_train, cv=5, scoring='recall')
    cv_f1 = cross_val_score(clf, X_train, y_train, cv=5, scoring='f1')
```

In [535…

```
 # Compute average performance metrics
avg_accuracy = cv_accuracy.mean()
avg_precision = cv_precision.mean()
avg_recall = cv_recall.mean()
avg_f1 = cv_f1.mean()
```

In [536…

```
# Store the performance metrics in the dictionary
performance_metrics[clf_name] = {
    "Accuracy": avg_accuracy,
    "Precision": avg_precision,
    "Recall": avg_recall,
    "F1-score": avg_f1
}
```

In [537…

```
# Sort the models based on their average accuracy in descending order
sorted_models = sorted(performance_metrics.items(), key=lambda x: x[1]["Accuracy

# Select the top model
best_model = sorted_models[:1]

# Print the performance metrics for the top model
for clf_name, metrics in best_model:
    print(f"{clf_name}:")
    for metric, value in metrics.items():
        print(f"  {metric}: {value}")
    print()
```

```
K-Nearest Neighbors:
  Accuracy: 0.8508275817702255
  Precision: 0.508292749711514
  Recall: 0.14331262939958592
  F1-score: 0.22137349415426058
```

In [538…

```
# Define k-NN model
knn_model = KNeighborsClassifier()

# Define hyperparameters grid for k-NN
knn_param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]  # p=1 for Manhattan distance, p=2 for Euclidean distance
}

# Perform GridSearchCV for k-NN
knn_grid = GridSearchCV(estimator=knn_model, param_grid=knn_param_grid, scoring=
knn_grid.fit(X_train, y_train)

# Get best parameters and best score for k-NN
best_knn_params = knn_grid.best_params_
best_knn_score = knn_grid.best_score_

# Create new k-NN model with best parameters
best_knn_model = KNeighborsClassifier(**best_knn_params)
best_knn_model.fit(X_train, y_train)

# Evaluate best k-NN model
best_knn_model_score = best_knn_model.score(X_test, y_test)

print("Best k-NN parameters:", best_knn_params)
print("Best k-NN score (cross-validation):", best_knn_score)
print("Test score of best k-NN model:", best_knn_model_score)
```

```
Best k-NN parameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
Best k-NN score (cross-validation): 0.8551203462885187
Test score of best k-NN model: 0.871
```

Conclusion

According to the evaluation, the KNN model appears to be the best performing model overall.

With that said, there are still uncertainties with the predictions made using this model. With that in mind, I would recommend further developing this model and providing it with further data to continue to train it for a more robust and reliable prediction.

Recommendations

For SyriaTel to improve on customer retention they need to deploy the machine learning model to get realtime predicitions. Realtime continuous monitoring ensure the model is always learning and improving with time. Feature importance on can be applied to provide insight on how to enact service improvements and optimize retention efforts.

A few ways into which SyriaTel can reduce Churning rate is by:

- Focusing on retention programs.

- Improve on quality of customer service responses. Provide quick and effective customer support that makes customers feel heard and valued.
- Encourage and act upon customer feedback. Use surveys, reviews, and feedback forms to understand customer satisfaction and areas for improvement.
- Competitive Pricing on plans : Regularly review and adjust pricing strategies on different plans to remain competitive in the market.
- Consider offering flexible pricing plans that cater to different customer needs.

In [ ]: