

# Prediction of customer churn: SyriaTel Telecommunications ¶

## Introduction

SyriaTel, a telecommunications company, faces a significant challenge in reducing customer churn, which can negatively impact their revenue and profitability. Customer churn refers to the phenomenon where customers discontinue their services with a company, often switching to competitors or simply discontinuing the service altogether. Some of the factors contributing significantly to customer churn are Poor service experience and customer service, making it easy for customers to switch providers and poor customer experiences, such as multiple contacts for issue resolution. These factors highlight the importance of addressing service quality and customer satisfaction to reduce churn rate.

## Business understanding:

Predicting customer churn is essential for SyriaTel to retain its customer base and minimize revenue loss. By identifying customers at risk of churning, SyriaTel can proactively engage with them through incentives, personalized offers, or enhanced customer service. The high cost of acquiring new customers compared to retaining existing ones motivates companies like SyriaTel to focus on churn reduction strategies. Leveraging predictive modeling allows for the detection of behavioral patterns indicating a high likelihood of churn. This proactive approach aims to enhance customer retention rates and overall business performance.

## Problem statement:

SyriaTel, a telecommunications company, faces the challenge of customer churn, where customers discontinue their services with the company, leading to revenue loss and reduced profitability. To address this issue and improve customer retention, the company aims to develop a predictive model that can identify customers at risk of churn. The goal of this project is to build a binary classification model that predicts whether a customer will "soon" stop doing business with SyriaTel.

## Key Objectives:

- Develop a robust predictive model using machine learning algorithms to accurately forecast customer churn in the telecommunications company.
- Identify predictive patterns and to evaluate the model's performance using appropriate metrics and validate its effectiveness.
- To determine the Percentage rate of customer churn per area code in the company
- To determine the factors that mostly contribute to customer churn in the company

## Scope:

The project will focus on analyzing historical data related to customer behavior, demographics, service usage, and interactions. Various machine learning algorithms will be explored and evaluated to build the predictive model. Data preprocessing, feature

engineering, model training, and evaluation will be conducted to optimize model performance. The predictive model will be deployed into SyriaTel's operational systems to enable real-time churn prediction and proactive customer retention strategies.

#### Success Criteria:

The predictive model achieves high accuracy, precision and F1-score in identifying customers at risk of churn. The model provides actionable insights and recommendations that enable SyriaTel to implement effective customer retention strategies. Implementation of the predictive model leads to a measurable reduction in customer churn rate and increased customer retention for SyriaTel.

#### Data understanding:

The dataset consist of 3333 rows and 21 columns. It is a binary classification problem. The column include a mix of data types that is object, integer, float and boolean. There are no missing values and duplicates. The columns for the dataset are as follows:

state: The state in which the customer resides (categorical).

account length: The number of days the customer has been with the company (numerical).

area code: The area code of the customer's phone number (categorical).

phone number: The customer's phone number (categorical).

international plan: Whether the customer has an international plan or not (categorical).

voice mail plan: Whether the customer has a voice mail plan or not (categorical).

number vmail messages: Number of voice mail messages (numerical).

total day minutes: Total minutes of day calls (numerical).

total day calls: Total number of day calls (numerical).

total day charge: Total charge for day calls (numerical).

total eve minutes: Total minutes of evening calls (numerical).

total eve calls: Total number of evening calls (numerical).

total eve charge: Total charge for evening calls (numerical).

total night minutes: Total minutes of night calls (numerical).

total night calls: Total number of night calls (numerical).

total night charge: Total charge for night calls (numerical).

total intl minutes: Total minutes of international calls (numerical).

total intl calls: Total number of international calls (numerical).

total intl charge: Total charge for international calls (numerical).

.. .. . - . . . . .

# Data cleaning and Explanatory Analysis(EDA)

```
In [45]: 1 # Import relevant libraries
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 %matplotlib inline
6 import seaborn as sns
7 from sklearn.model_selection import train_test_split, GridSearchCV
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
10 from sklearn.metrics import accuracy_score
11 from sklearn.feature_selection import SelectKBest
12 from sklearn.feature_selection import chi2
13 import statsmodels.api as sm
14 from sklearn.metrics import mean_squared_error
15 import warnings
16 from sklearn.preprocessing import LabelEncoder
17 from sklearn.linear_model import LogisticRegression
18 from sklearn.metrics import accuracy_score, precision_score, recall_score
19 from sklearn.preprocessing import MinMaxScaler
20 from sklearn.neighbors import KNeighborsClassifier
21 from sklearn.ensemble import GradientBoostingClassifier
22 from sklearn.linear_model import LogisticRegressionCV
```

```
In [46]: 1 # Read the csv file
2 df = pd.read_csv('cleaned_SyriaTelCustomerchurn.csv')
3 df.head()
4
```

Out[46]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charges
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.0
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.3
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.9
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.3

5 rows × 11 columns



## Cleaning of the data

In [47]: ▶

```
1 # check missing values and data type
2 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
```

No null values spotted. Most of the features are numerical except 'state', 'phone\_number', 'international\_plan', 'voice\_mail\_plan' which are strings. The target variable in this case is 'churn'

In [48]: ▶

```
1 # check the shape of the DataFrame
2 print(df.shape)
```

```
(3333, 21)
```

There are 3333 rows and 21 columns in the dataset



```

In [51]: 1 # Check for duplicates
2 duplicates = df.duplicated()
3 duplicates_count = duplicates.sum()
4 print("Number of duplicate rows:", duplicates_count)
5
6 # Print the rows with duplicates
7 if duplicates_count > 0:
8     duplicate_rows = df[duplicates]
9     print("Duplicate rows:")
10    print(duplicate_rows[:5])
11 else:
12     print("No duplicate rows found.")

```

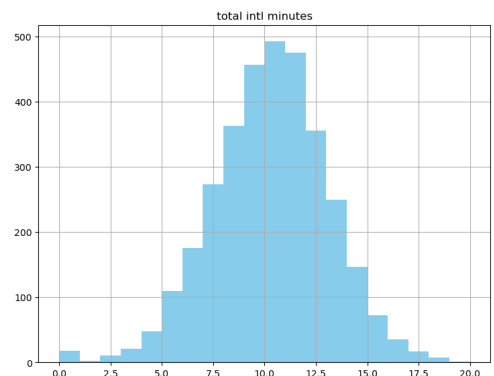
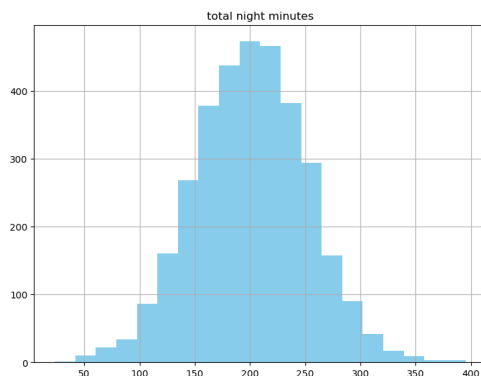
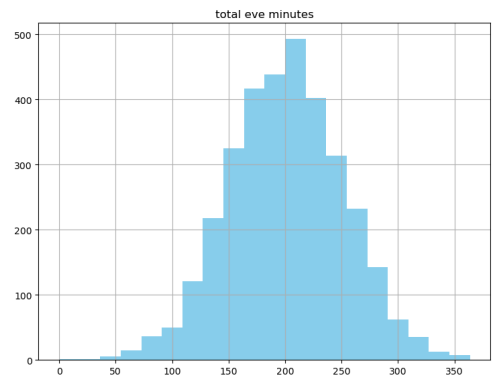
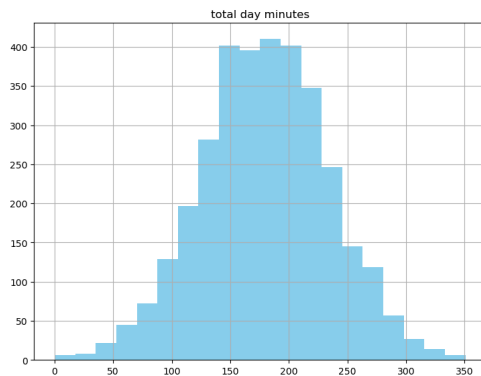
Number of duplicate rows: 0  
No duplicate rows found.

No duplicate rows found in the dataset

```

In [52]: 1 # Checking if the total minutes of the day, evening, night and intl
2 calls_minutes = [ 'total day minutes', 'total eve minutes',
3                  'total night minutes', 'total intl minutes'
4                  ]
5 df[calls_minutes].hist(figsize=(20, 15), color = 'skyblue', bins =
6 plt.suptitle('Histograms for calls in Minutes')
7 plt.show()

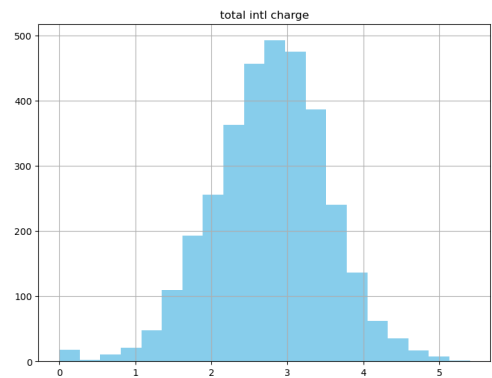
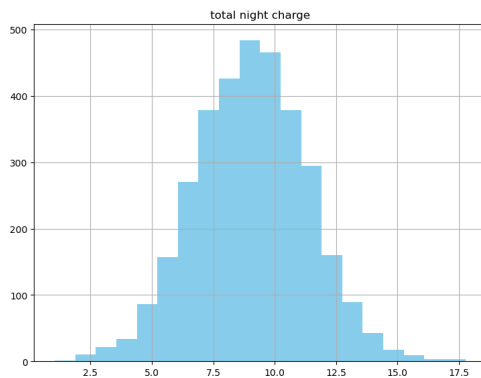
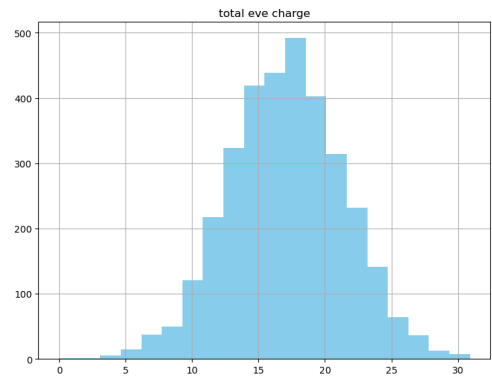
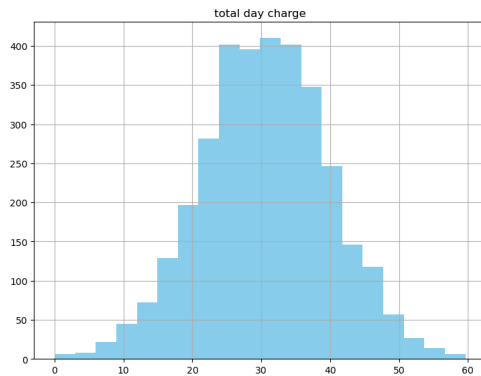
```



The total minutes for day, evening, night and international calls were normally distributed as depicted in the histograms above

```
In [53]: 1 # Checking if the 'total_day_charge', 'total_eve_charge', 'total_n
2 calls_minutes = [ 'total day charge', 'total eve charge',
3                 'total night charge', 'total intl charge',
4                 ]
5 df[calls_minutes].hist(figsize=(20, 15), color = 'skyblue', bins =
6 plt.suptitle('Histograms for Charges')
7 plt.show()
8
```

Histograms for Charges



The total charges for day, evening, night and international calls were normally distributed i.e they all have a belly shape as depicted in the histograms above

Drop irrelevant features

```
In [54]: 1 df.drop(columns=['phone number'], inplace=True)
```

```
In [55]: 1 # check the percentage rate of churn by are code
2 churn_percentange_by_area_code = df.groupby('area code')['churn'].r
3
4 print(churn_percentange_by_area_code)
```

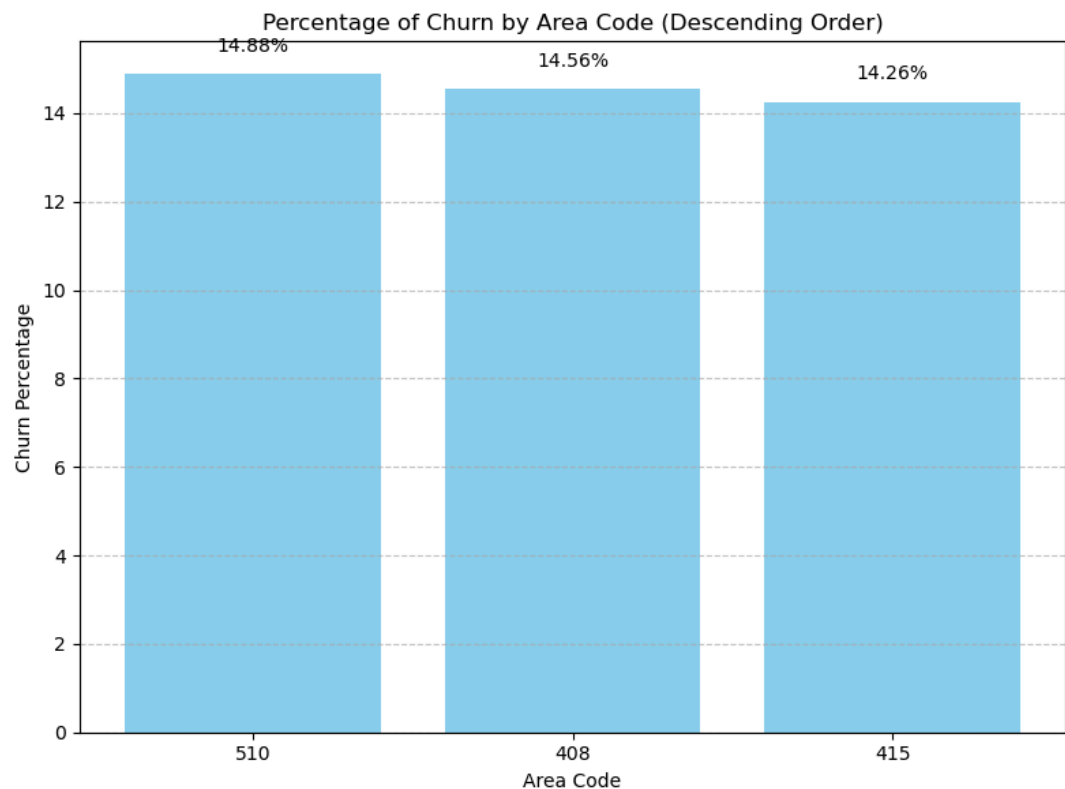
```
area code
408    14.558473
415    14.259819
510    14.880952
Name: churn, dtype: float64
```

In [56]:

```

1 2 # Sort area codes based on churn percentage in descending order
3 sorted_area_codes = churn_percentage_by_area_code.sort_values(asc
4
5 # Plotting
6 plt.figure(figsize=(8, 6))
7
8 # Grouping by sorted area codes and iterating through each group
9 for area_code in sorted_area_codes:
10     # Plotting churn percentage for each area code with sky blue color
11     plt.bar(str(area_code), churn_percentage_by_area_code[area_code])
12
13     # Adding text labels for percentage
14     plt.text(str(area_code), churn_percentage_by_area_code[area_code],
15
16 # Adding labels and titles
17 plt.title('Percentage of Churn by Area Code (Descending Order)')
18 plt.xlabel('Area Code')
19 plt.ylabel('Churn Percentage')
20 plt.xticks(rotation=0)
21 plt.grid(axis='y', linestyle='--', alpha=0.7)
22 plt.tight_layout()
23 plt.show()

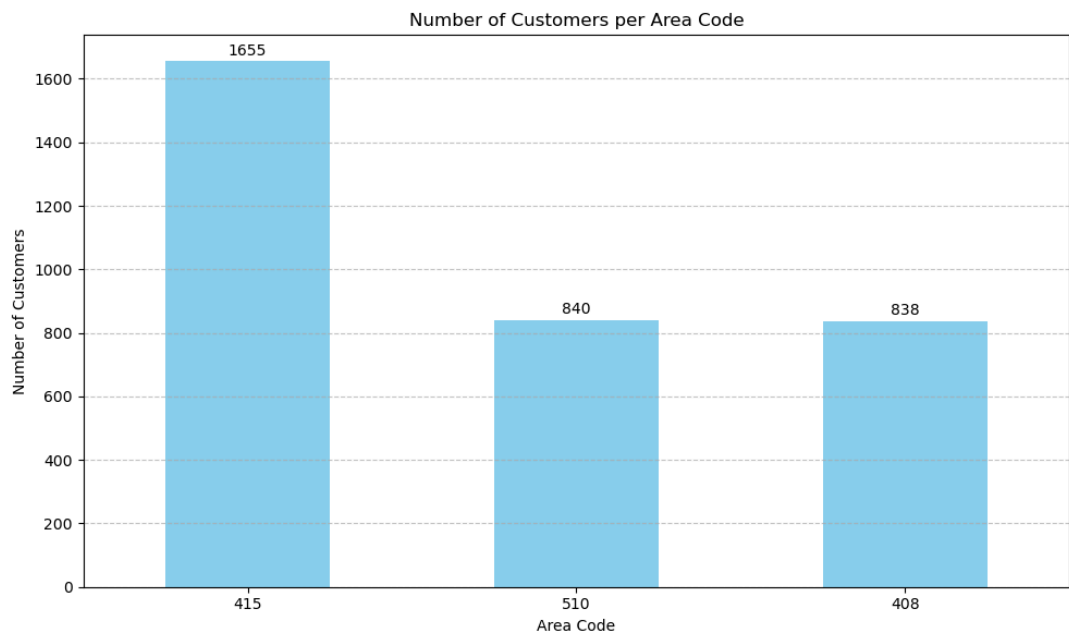
```



The percentage of customer churn by area code is approximately between 14.26% - 14.88%. The leading area code in terms of customer churn is 510(14,88%), followed by 408(14.56%) and finally 415(14.26%).



```
In [57]: 1 # Number of customers per area code
2 customers_per_area_code = df['area code'].value_counts()
3
4 # Plotting
5 plt.figure(figsize=(10, 6))
6 customers_per_area_code.plot(kind='bar', color='skyblue')
7
8 # Adding text labels for number of customers
9 for x, y in enumerate(customers_per_area_code):
10     plt.text(x, y + 10, str(y), ha='center', va='bottom')
11
12 plt.title('Number of Customers per Area Code')
13 plt.xlabel('Area Code')
14 plt.ylabel('Number of Customers')
15 plt.xticks(rotation=0)
16 plt.grid(axis='y', linestyle='--', alpha=0.7)
17 plt.tight_layout()
18 plt.show()
```



Area code 415 is leading with the number of customers, followed by area code 510 and area code 408.

```
In [58]: 1 # Drop the area code because the percentage of customer churn is ap
2 df.drop(columns=['area code'], inplace=True)
```

```
In [59]: 1 # check the columns after dropping the area code
2 df.columns
```

```
Out[59]: Index(['state', 'account length', 'international plan', 'voice mail pl
an',
               'number vmail messages', 'total day minutes', 'total day call
s',
               '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 [60]:

1 df.head()

Out[60]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	KS	128	no	yes	25	265.1	110	45.07	197.4	9
1	OH	107	no	yes	26	161.6	123	27.47	195.5	10
2	NJ	137	no	no	0	243.4	114	41.38	121.2	11
3	OH	84	yes	no	0	299.4	71	50.90	61.9	8
4	OK	75	yes	no	0	166.7	113	28.34	148.3	12

In [61]:

1 # convert target variable 'churn' to binary  
2 df['churn'] = df['churn'].map({False: 0, True: 1})

In [62]:

1 # Convert 'international plan' and 'voice mail plan' into binary  
2 df['international plan'].replace(('yes', 'no'), (1, 0), inplace=True)  
3 df['voice mail plan'].replace(('yes', 'no'), (1, 0), inplace=True)

In [63]:

1 df.head()

Out[63]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	KS	128	0	1	25	265.1	110	45.07	197.4	9
1	OH	107	0	1	26	161.6	123	27.47	195.5	10
2	NJ	137	0	0	0	243.4	114	41.38	121.2	11
3	OH	84	1	0	0	299.4	71	50.90	61.9	8
4	OK	75	1	0	0	166.7	113	28.34	148.3	12

In [64]:

1 # Total number of people who churned/didn't churn  
2 df['churn'].value\_counts()

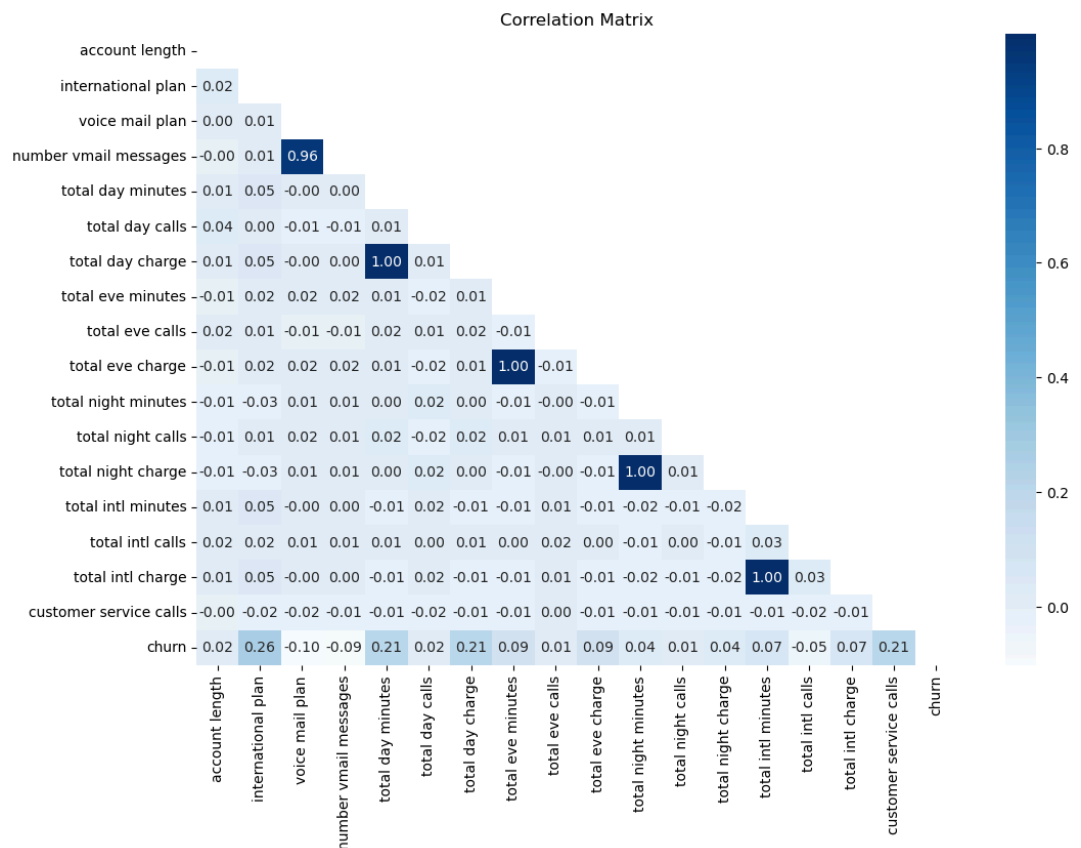
Out[64]:

churn  
0 2850  
1 483  
Name: churn, dtype: int64

```

In [65]: ▶ 1 # Suppress warnings
2 warnings.filterwarnings("ignore")
3 # Exclude non-numeric columns from the correlation calculation
4 numeric_df = df.select_dtypes(include='number')
5
6 # Calculate the correlation matrix
7 correlation_matrix = numeric_df.corr()
8
9 # Create a mask to display only the lower triangle
10 mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
11
12 # Set up the matplotlib figure
13 plt.figure(figsize=(12, 8))
14
15 # Plot the heatmap
16 sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='Blues')
17
18 plt.title('Correlation Matrix')
19 plt.show()
20

```



There is a strong relationship between Voice mail plan and number of voice mail messages, total Day Minutes and total Day Charge, total evening minutes and total evening charges and total night minutes and total night charges and finally total international minutes and total international charges. There is a strong relationship between target variable(churn) and international plan, total day minutes, total day charge and customers service calls. Generally, there was a weak positive linear relationship between the other variables

## Data preparation

## Label encoding 'state' column

```
In [66]: 1
2 # Initialize LabelEncoder
3 label_encoder = LabelEncoder()
4
5 # Apply LabelEncoder to each categorical column
6 for col in df.columns:
7     if df[col].dtype == 'object': # Check if the column is categorical
8         df[col] = label_encoder.fit_transform(df[col])
9
10 # Display the DataFrame after Label encoding
11 df
```

Out[66]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
0	16	128	0	1	25	265.1	110	45.07	197.4
1	35	107	0	1	26	161.6	123	27.47	195.5
2	31	137	0	0	0	243.4	114	41.38	121.2
3	35	84	1	0	0	299.4	71	50.90	61.9
4	36	75	1	0	0	166.7	113	28.34	148.3
...	...	...	...	...	...	...	...	...	...
3328	3	192	0	1	36	156.2	77	26.55	215.5
3329	49	68	0	0	0	231.1	57	39.29	153.4
3330	39	28	0	0	0	180.8	109	30.74	288.8
3331	6	184	1	0	0	213.8	105	36.35	159.6
3332	42	74	0	1	25	234.4	113	39.85	265.9

3333 rows × 19 columns

## splitting data

Splitting data in training and testing sets

Create X, y variables:

```
In [67]: 1 # Defining X and y
2 y = df["churn"]
3 X = df.drop("churn", axis=1)
4
5 # Split the data into training and test sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

## Transforming the training set

Transforming training data/set prior to fitting the model will prevent data leakage. It will improve model convergence, prevention of numerical instability during optimization process

```
In [68]: 1 # use min-max scaling
2 # Your code here
3 # Instantiate the MinMaxScaler
4 scaler = MinMaxScaler()
5
6 # Fit the scaler on the training data and transform it
7 X_train_scaled = scaler.fit_transform(X_train)
8
9 # Convert the scaled data back to a pandas DataFrame
10 X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
11
12 # Display the scaled data
13 X_train_scaled_df.head()
```

Out[68]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total e minut
0	0.40	0.190476	0.0	0.0	0.000000	0.217117	0.718519	0.217061	0.6967
1	0.16	0.493506	0.0	0.0	0.000000	0.555141	0.600000	0.555068	0.6241
2	0.72	0.519481	0.0	1.0	0.607843	0.673464	0.244444	0.673480	0.5653
3	0.78	0.774892	0.0	0.0	0.000000	0.404078	0.770370	0.404054	0.4962
4	0.74	0.480519	0.0	0.0	0.000000	0.584721	0.681481	0.584628	0.4522

## Logistic Regression(Baseline model)

### fit model

```
In [69]: 1 # Instantiate the model
2 logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='lbfgs')
3
4 # Fit the model
5 logreg.fit(X_train, y_train)
```

Out[69]:

```
LogisticRegression
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='lbfgs')
```

## Generate predictions for the training and test sets.

```
In [70]: 1 # Generate predictions
          2 y_hat_train = logreg.predict(X_train)
          3 y_hat_test = logreg.predict(X_test)
```

```
In [71]: 1 # Logistic regression on Training set
          2 residuals = np.abs(y_train - y_hat_train)
          3 print(pd.Series(residuals).value_counts())
          4 print('-----')
          5 print(pd.Series(residuals).value_counts(normalize=True))
```

```
churn
0    2155
1     344
Name: count, dtype: int64
-----
churn
0    0.862345
1    0.137655
Name: proportion, dtype: float64
```

About 86.23% of the instances in the training set correspond to customers who did not churn (churn = 0). About 13.77% of the instances in the training set correspond to customers who churned (churn = 1).

```
In [72]: 1 # Logistic regression on Testing set
          2 residuals = np.abs(y_test - y_hat_test)
          3 print(pd.Series(residuals).value_counts())
          4 print('-----')
          5 print(pd.Series(residuals).value_counts(normalize=True))
```

```
churn
0     704
1     130
Name: count, dtype: int64
-----
churn
0    0.844125
1    0.155875
Name: proportion, dtype: float64
```

About 84.41% of the instances in the dataset correspond to customers who did not churn (churn = 0). About 15.59% of the instances in the dataset correspond to customers who churned (churn = 1).

```

In [73]: 1 # create confusion matrix
2 def conf_matrix(y_true, y_pred):
3     TP = sum((y_true == 1) & (y_pred == 1))
4     TN = sum((y_true == 0) & (y_pred == 0))
5     FP = sum((y_true == 0) & (y_pred == 1))
6     FN = sum((y_true == 1) & (y_pred == 0))
7
8     return {'TP': TP, 'TN': TN, 'FP': FP, 'FN': FN}
9
10
11 # Test the function
12 conf_matrix(y_test, y_hat_test)

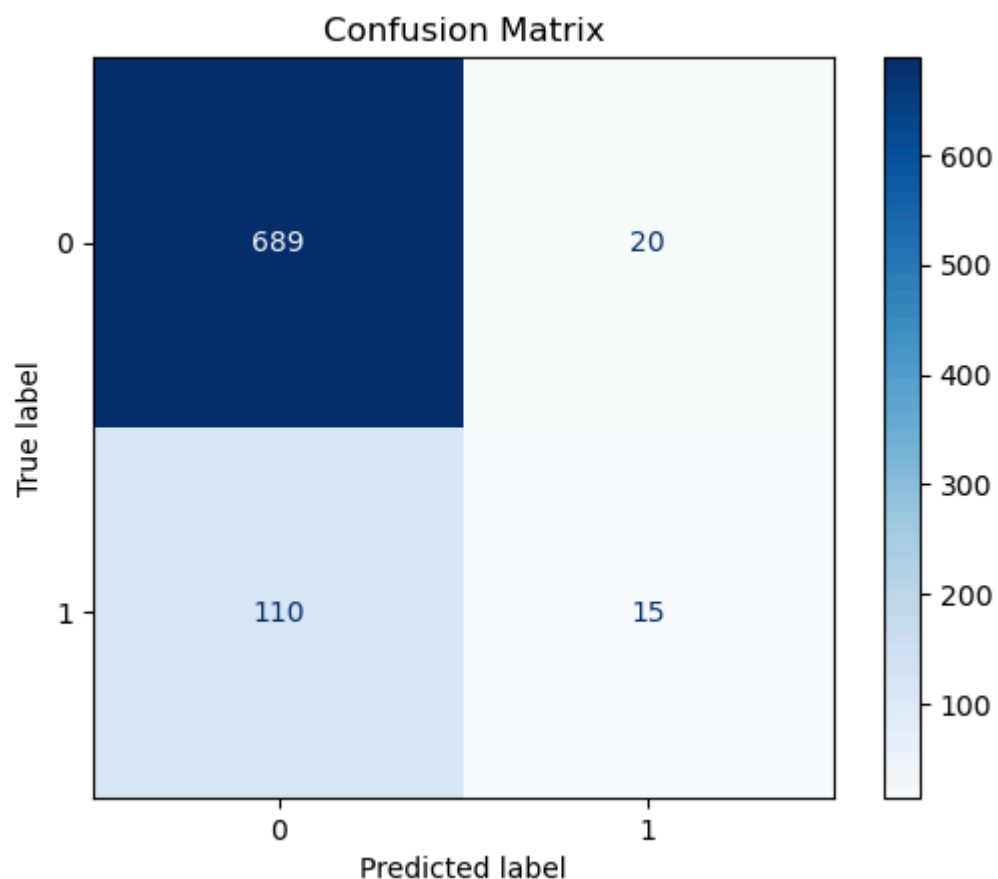
```

Out[73]: {'TP': 15, 'TN': 689, 'FP': 20, 'FN': 110}

```

In [74]: 1 # Import plot_confusion_matrix
2 from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
3
4 cnf_matrix = confusion_matrix(y_test, y_hat_test)
5 # Visualize your confusion matrix
6 disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=[0, 1])
7 disp.plot(cmap=plt.cm.Blues)
8 plt.title('Confusion Matrix')
9 plt.show()

```



## Evaluation of metrics: Logistic regression model

```
In [75]: 1 def evaluate_model(y_true, y_pred):
2         accuracy = accuracy_score(y_true, y_pred)
3         precision = precision_score(y_true, y_pred)
4         roc_auc = roc_auc_score(y_true, y_pred)
5         return accuracy, precision, roc_auc
6
7     # Evaluation for training set
8     accuracy_train, precision_train, roc_auc_train = evaluate_model(y_train, y_pred_train)
9
10    # Evaluation for test set
11    accuracy_test, precision_test, roc_auc_test = evaluate_model(y_test, y_pred_test)
12
13    print("Training Set Metrics:")
14    print("Accuracy: ", accuracy_train)
15    print("Precision: ", precision_train)
16    print("ROC AUC Score: ", roc_auc_train)
17
18
19    print("\nTest Set Metrics:")
20    print("Accuracy: ", accuracy_test)
21    print("Precision: ", precision_test)
22    print("ROC AUC Score: ", roc_auc_test)
```

```
Training Set Metrics:
Accuracy:  0.8623449379751901
Precision:  0.5573770491803278
ROC AUC Score:  0.5823611375668968
```

```
Test Set Metrics:
Accuracy:  0.8441247002398081
Precision:  0.42857142857142855
ROC AUC Score:  0.5458956276445699
```

The accuracy of the model on both sets is relatively high, suggesting that it performs reasonably well in terms of overall classification accuracy. The model performs slightly worse on the test set compared to the training set, indicating some degree of overfitting. The precision is lower, indicating that there are a significant number of false positives in the predictions, particularly evident in the test set. The ROC AUC scores suggest that the model's ability to distinguish between the classes is limited, indicating potential areas for improvement in the model's discriminatory power.

## model 2: Decision trees

Building Trees using scikit-learn one of the advantages using Scikit-learn is that it provides a consistent interface for running different classifiers/regressors



```
In [76]: 1 # Train the decision tree
2 from sklearn.tree import DecisionTreeClassifier
3 # Create the classifier, fit it on the training data and make predictions
4 clf = DecisionTreeClassifier(random_state=5, criterion='entropy')
5
6 clf.fit(X_train, y_train)
```

```
Out[76]: DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=5)
```

```
In [77]: 1 # Make predictions on the train data using the trained classifier
2 y_train_pred = clf.predict(X_train)
3
4 # Calculate evaluation metrics for train set
5 train_accuracy = accuracy_score(y_train, y_train_pred)
6 train_precision = precision_score(y_train, y_train_pred)
7 train_roc_auc = roc_auc_score(y_train, y_train_pred)
8
9 print("Train Set Metrics:")
10 print("Accuracy:", train_accuracy)
11 print("Precision:", train_precision)
12 print("ROC AUC Score:", train_roc_auc)
13
14 # Make predictions on the test data using the trained classifier
15 y_pred = clf.predict(X_test)
16
17 # Calculate evaluation metrics for test set
18 accuracy = accuracy_score(y_test, y_pred)
19 precision = precision_score(y_test, y_pred)
20 roc_auc = roc_auc_score(y_test, y_pred)
21
22 print("\nTest Set Metrics:")
23 print("Accuracy:", accuracy)
24 print("Precision:", precision)
25 print("ROC AUC Score:", roc_auc)
```

Train Set Metrics:

Accuracy: 1.0

Precision: 1.0

ROC AUC Score: 1.0

Test Set Metrics:

Accuracy: 0.9160671462829736

Precision: 0.7391304347826086

ROC AUC Score: 0.8188434414668548

perfect performance on the training set could be a sign of overfitting, where the model has memorized the training data and may not generalize well to unseen data. An accuracy of 0.916 means that the model correctly predicted approximately 91.6% of the instances. A precision of 0.739 indicates that out of all instances predicted as positive by the model, approximately 73.9% were actually positive. An ROC AUC score of 0.819 suggests that the model performs reasonably well in distinguishing between the two classes, with a higher score indicating better performance. Generally, This discrepancy between the training and test set metrics suggests that there may be some level of overfitting in the model, where it

has learned the noise or specific patterns in the training data that do not generalize well to new unseen data. Therefore, further investigation and potential model tuning may be

## Hyperparameter tuning and feature importance analysis

```
In [78]: ▶ 1 # Hyperparameter tuning and feature importance analysis to address
2 # Define the parameter grid
3 param_grid = {
4     'criterion': ['gini', 'entropy'],
5     'max_depth': [None, 5, 10, 15, 20],
6     'min_samples_split': [2, 5, 10],
7     'min_samples_leaf': [1, 2, 4]
8 }
9
10 # Create the grid search object
11 grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random
12                             param_grid=param_grid,
13                             scoring='accuracy',
14                             cv=5,
15                             n_jobs=-1)
16
17 # Fit the grid search to the data
18 grid_search.fit(X_train, y_train)
19
20 # Print the best hyperparameters
21 print("Best Hyperparameters:", grid_search.best_params_)
22
23 # Get the best model
24 best_dt_model = grid_search.best_estimator_
25
26 # Fit the decision tree model to the entire dataset
27 best_dt_model.fit(X_train, y_train)
28
29 # Get feature importances
30 feature_importances = best_dt_model.feature_importances_
31
32 # Create a DataFrame to store feature importances
33 feature_importance_df = pd.DataFrame({'Feature': X_train.columns,
34
35 # Sort the DataFrame by importance values
36 feature_importance_df = feature_importance_df.sort_values(by='Importance',
37
38 # Print the feature importances
39 print(feature_importance_df)
```

Best Hyperparameters: {'criterion': 'gini', 'max\_depth': 5, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2}

	Feature	Importance
5	total day minutes	0.178413
17	customer service calls	0.156280
7	total day charge	0.140431
16	total intl charge	0.116918
2	international plan	0.114449
15	total intl calls	0.082415
8	total eve minutes	0.064888
10	total eve charge	0.064108
4	number vmail messages	0.060373
11	total night minutes	0.009320
13	total night charge	0.008527
9	total eve calls	0.003878
12	total night calls	0.000000
1	account length	0.000000
14	total intl minutes	0.000000
6	total day calls	0.000000
3	voice mail plan	0.000000
0	state	0.000000

**Total Day Minutes:** This feature has the highest importance, indicating that total number of minutes a customer spends on day calls is a significant predictor of churn. Customers who spend more time on day calls may be more likely to churn.

**Customer Service Calls:** The number of customer service calls also has a high importance. This suggests that customers who frequently contact customer service may be experiencing issues or dissatisfaction with the service, leading to a higher likelihood of churn

**Total Day Charge:** Similar to total day minutes, the total day charge is an important factor. Higher charges for day calls may indicate dissatisfaction with pricing or service quality, leading to churn.

**International Plan:** Whether or not a customer has an international plan also plays a role in churn prediction. Customers with international plans may have different usage patterns or expectations, influencing their likelihood of churn. **Total Intl Charge:** Charges for international calls also contribute to churn prediction. High charges for international calls may lead to dissatisfaction and churn

Removing less important features

```
In [79]: 1 columns_to_drop = ['total night calls', 'account length', 'total intl minutes']
          2 df_filtered = df.drop(columns_to_drop, axis=1)
```

```
In [80]: 1 # Evaluate the model after dropping less important features
2 # Make predictions on the train set
3 y_train_pred = best_dt_model.predict(X_train)
4
5 # Calculate evaluation metrics for the train set
6 train_accuracy = accuracy_score(y_train, y_train_pred)
7 train_precision = precision_score(y_train, y_train_pred)
8 train_roc_auc = roc_auc_score(y_train, y_train_pred)
9
10 print("Train Set Metrics:")
11 print("Accuracy:", train_accuracy)
12 print("Precision:", train_precision)
13 print("ROC AUC Score:", train_roc_auc)
14
15 # Make predictions on the test set
16 y_pred = best_dt_model.predict(X_test)
17
18 # Calculate evaluation metrics for the test set
19 accuracy = accuracy_score(y_test, y_pred)
20 precision = precision_score(y_test, y_pred)
21 roc_auc = roc_auc_score(y_test, y_pred)
22
23 print("\nTest Set Metrics:")
24 print("Accuracy:", accuracy)
25 print("Precision:", precision)
26 print("ROC AUC Score:", roc_auc)
```

Train Set Metrics:  
Accuracy: 0.9539815926370548  
Precision: 0.948339483394834  
ROC AUC Score: 0.8556690472524979

Test Set Metrics:  
Accuracy: 0.935251798561151  
Precision: 0.8901098901098901  
ROC AUC Score: 0.816947813822285

**Train Set Metrics:** The model performs well on the training set with an accuracy of 0.954, precision of 0.948, and ROC AUC score of 0.856. These metrics indicate that the model is able to correctly classify the majority of instances in the training set and achieve a high true positive rate while minimizing false positives.

**Test Set Metrics:** The model also generalizes reasonably well to unseen data, with an accuracy of 0.935, precision of 0.890, and ROC AUC score of 0.817 on the test set. These metrics suggest that the model maintains good performance on new data, indicating that it has learned relevant patterns from the training data without overfitting.

Overall, the results indicate that the model performs well both on the training and test sets, suggesting that the hyperparameter tuning and feature importance analysis have effectively improved the model's generalization performance. However, further monitoring and potentially additional optimization may still be necessary to ensure the model's robustness and reliability across different datasets or scenarios.

**Conclusion:** comparing baseline model(Logistic regression) and decision trees model in terms of model performance, Decision trees performed well based on the Accuracy, precision and ROC AUC score.

## Model 3: K- Nearest Neighbours(KNN) Model

```
In [81]: 1 # Instantiate KNeighborsClassifier
2 clf = KNeighborsClassifier()
3
4 # Fit the classifier
5 clf.fit(X_train_scaled, y_train)
6
7 # Predict on the training set
8 train_preds = clf.predict(X_train_scaled)
9
10 # Predict on the test set
11 test_preds = clf.predict(X_test.values)
```

## Evaluate the model

```
In [82]: 1 # Accuracy on the training set
2 train_accuracy = accuracy_score(y_train, train_preds)
3 print("Training Set Accuracy:", train_accuracy)
4
5 # Precision on the training set
6 train_precision = precision_score(y_train, train_preds)
7 print("Training Set Precision:", train_precision)
8
9 # ROC AUC score on the training set
10 train_roc_auc = roc_auc_score(y_train, train_preds)
11 print("Training Set ROC AUC Score:", train_roc_auc)
12
13 # Accuracy on the test set
14 test_accuracy = accuracy_score(y_test, test_preds)
15 print("\nTest Set Accuracy:", test_accuracy)
16
17 # Precision on the test set
18 test_precision = precision_score(y_test, test_preds)
19 print("Test Set Precision:", test_precision)
20
21 # ROC AUC score on the test set
22 test_roc_auc = roc_auc_score(y_test, test_preds)
23 print("Test Set ROC AUC Score:", test_roc_auc)
```

```
Training Set Accuracy: 0.9151660664265706
Training Set Precision: 0.9010989010989011
Training Set ROC AUC Score: 0.7248466361722059
```

```
Test Set Accuracy: 0.3800959232613909
Test Set Precision: 0.16779661016949152
Test Set ROC AUC Score: 0.549737658674189
```

These metrics indicate how well the model performs on the data it was trained on. The high accuracy and precision suggest that the model is doing well in predicting the churn class. However, the ROC AUC score, while decent, indicates that the model's ability to discriminate between positive and negative classes is not perfect. The lower accuracy,

precision, and ROC AUC score compared to the training set metrics indicate that the model is not performing well on new, unseen data. Overall, these results indicate that the model is overfitting to the training data and is not generalizing well to new data.

```
In [83]: 1 # use hyperparameter tuning and regularlization to mitigate overfitting
2
3 # Define the parameter grid
4 param_grid = {
5     'n_neighbors': [3, 5, 7, 9, 11],
6     'weights': ['uniform', 'distance'],
7     'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
8 }
9
10 # Instantiate the KNN classifier
11 knn = KNeighborsClassifier()
12
13 # Create the GridSearchCV object
14 grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, scoring='accuracy')
15
16 # Fit the GridSearchCV object to the training data
17 grid_search.fit(X_train, y_train)
18
19 # Get the best parameters
20 best_params = grid_search.best_params_
21
22 # Instantiate the KNN classifier with the best parameters
23 best_knn = KNeighborsClassifier(**best_params)
24
25 # Fit the best model to the training data
26 best_knn.fit(X_train, y_train)
27
28 # Make predictions on the training set
29 train_preds = best_knn.predict(X_train)
30
31 # Make predictions on the test set
32 test_preds = best_knn.predict(X_test)
33
34 # Evaluate the model on the training set
35 train_accuracy = accuracy_score(y_train, train_preds)
36 train_precision = precision_score(y_train, train_preds)
37 train_roc_auc = roc_auc_score(y_train, train_preds)
38
39 print("Training Set Metrics:")
40 print("Accuracy:", train_accuracy)
41 print("Precision:", train_precision)
42 print("ROC AUC Score:", train_roc_auc)
43
44 # Evaluate the model on the test set
45 test_accuracy = accuracy_score(y_test, test_preds)
46 test_precision = precision_score(y_test, test_preds)
47 test_roc_auc = roc_auc_score(y_test, test_preds)
48
49 print("\nTest Set Metrics:")
50 print("Accuracy:", test_accuracy)
51 print("Precision:", test_precision)
52 print("ROC AUC Score:", test_roc_auc)
```



Training Set Metrics:

Accuracy: 1.0

Precision: 1.0

ROC AUC Score: 1.0

Test Set Metrics:

Accuracy: 0.8800959232613909

Precision: 0.7659574468085106

ROC AUC Score: 0.6362425952045134

These metrics suggest that the model is performing exceptionally well on the training set, achieving perfect scores for accuracy, precision, and ROC AUC. However, on the test set, although the model still performs well with relatively high accuracy and precision scores, there is a drop in the ROC AUC score compared to the training set. This discrepancy between the training and test set performance indicates that there might be some overfitting. Explore Random forest to address the overfitting and improve the performance of the model Conclusion: The KNN model performed well as compared to the decision trees model specifically on training set.

## Model 4: Random forest

```
In [84]: ▶ 1 # Instantiate the Model
2 rf_model = RandomForestClassifier(random_state=42)
3
4 # Fit the Model
5 rf_model.fit(X_train, y_train)
6
7 # Evaluate performance metrics on the training set
8 y_train_pred = rf_model.predict(X_train)
9 train_accuracy = accuracy_score(y_train, y_train_pred)
10 train_precision = precision_score(y_train, y_train_pred)
11 train_roc_auc = roc_auc_score(y_train, y_train_pred)
12
13 print("Training Set Metrics:")
14 print("Accuracy:", train_accuracy)
15 print("Precision:", train_precision)
16 print("ROC AUC Score:", train_roc_auc)
17
18 # Evaluate the Model on test set
19 y_test_pred = rf_model.predict(X_test)
20 test_accuracy = accuracy_score(y_test, y_test_pred)
21 test_precision = precision_score(y_test, y_test_pred)
22 test_roc_auc = roc_auc_score(y_test, y_test_pred)
23
24 print("\nTest Set Metrics:")
25 print("Accuracy:", test_accuracy)
26 print("Precision:", test_precision)
27 print("ROC AUC Score:", test_roc_auc)
```

Training Set Metrics:

Accuracy: 1.0

Precision: 1.0

ROC AUC Score: 1.0

Test Set Metrics:

Accuracy: 0.9496402877697842

Precision: 0.9560439560439561

ROC AUC Score: 0.8451791255289139

Overall, these metrics suggest that the Random Forest model performs very well on both the training and test sets, demonstrating high accuracy, precision, and good discrimination ability. However, there might be some slight overfitting as indicated and to address this regularization technique is used to address overfitting in models that have high complexity.

## use regularization to mitigate overfitting

```
In [85]: ▶ 1 # Define the parameter grid for hyperparameter tuning
2 param_grid = {
3     'max_depth': [None, 5, 10, 15, 20],
4     'min_samples_split': [2, 5, 10],
5     'min_samples_leaf': [1, 2, 4]
6 }
7
8 # Create the decision tree classifier
9 dt_classifier = DecisionTreeClassifier(random_state=42)
10
11 # Instantiate the GridSearchCV object
12 grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,
13                             scoring='accuracy', cv=5, n_jobs=-1)
14
15 # Fit the grid search to the data
16 grid_search.fit(X_train, y_train)
17
18 # Get the best hyperparameters
19 best_params = grid_search.best_params_
20
21 # Instantiate the decision tree classifier with the best hyperparameters
22 best_dt_model = DecisionTreeClassifier(random_state=42, **best_params)
23
24 # Fit the model to the training data
25 best_dt_model.fit(X_train, y_train)
26
27 # Evaluate the model
28 train_accuracy = best_dt_model.score(X_train, y_train)
29 test_accuracy = best_dt_model.score(X_test, y_test)
```

```
In [86]: ▶ 1 # After regularlization technique evaluate the model
2 # Evaluate the model on the training set
3 train_preds = best_dt_model.predict(X_train)
4 train_accuracy = accuracy_score(y_train, train_preds)
5 train_precision = precision_score(y_train, train_preds)
6 train_roc_auc = roc_auc_score(y_train, train_preds)
7
8 print("Training Set Metrics:")
9 print("Accuracy:", train_accuracy)
10 print("Precision:", train_precision)
11 print("ROC AUC Score:", train_roc_auc)
12
13 # Evaluate the model on the test set
14 test_preds = best_dt_model.predict(X_test)
15 test_accuracy = accuracy_score(y_test, test_preds)
16 test_precision = precision_score(y_test, test_preds)
17 test_roc_auc = roc_auc_score(y_test, test_preds)
18
19 print("\nTest Set Metrics:")
20 print("Accuracy:", test_accuracy)
21 print("Precision:", test_precision)
22 print("ROC AUC Score:", test_roc_auc)
```

Training Set Metrics:  
 Accuracy: 0.9539815926370548  
 Precision: 0.9418181818181818  
 ROC AUC Score: 0.8579952718799496

Test Set Metrics:  
 Accuracy: 0.935251798561151  
 Precision: 0.8817204301075269  
 ROC AUC Score: 0.8202425952045135

Training set metrics Accuracy score: This indicates that the model correctly predicts 95.40% of the instances in the training set. Precision score: Out of all the instances predicted as positive (churn), 94.18% are actually positive. The ROC AUC score : measures the model's ability to distinguish between positive and negative classes. A score of 85.80% suggests that the model performs well in this regard on the training set.

Test set metrics Accuracy score: The accuracy on the test set is 93.53%, indicating that the model generalizes well to unseen data. Presion score: The precision on the test set is 88.17%, which means that out of all the instances predicted as positive, 88.17% are actually positive ROC AUC Score: The ROC AUC score on the test set is 82.02%, suggesting that the model maintains good performance in distinguishing between positive and negative classes even on unseen data.

Overall, these results indicate that the regularization has effectively mitigated overfitting, leading to improved generalization performance of the model on both the training and test sets Conclusion: Random forest model performed well as compared to KNN model. Random forest is generraly robust and less sensitive to noise and outliers in the data. It can handle high-dimensional datasets and typically performs well out-of-the-box with minimal hyperparameter tuning. Gradiient Boosting is explored because often achieves higher predictive accuracy than Random Forest.

## model 5: Gradient Boosting

These algorithm often provide better performance and robustness, especially in complex datasets.

```
In [87]: 1 # Explore Gradient Boosting to further improve model performance
2
3 # Instantiate the model
4 gb_model = GradientBoostingClassifier()
5
6 # Fit the model to the training data
7 gb_model.fit(X_train, y_train)
8
9 # Evaluate performance metrics on the training set
10 y_train_pred = gb_model.predict(X_train)
11 train_accuracy = accuracy_score(y_train, y_train_pred)
12 train_precision = precision_score(y_train, y_train_pred)
13 train_roc_auc = roc_auc_score(y_train, y_train_pred)
14
15 print("Training Set Metrics:")
16 print("Accuracy:", train_accuracy)
17 print("Precision:", train_precision)
18 print("ROC AUC Score:", train_roc_auc)
19
20 # Evaluate the model on the test set
21 y_test_pred = gb_model.predict(X_test)
22 test_accuracy = accuracy_score(y_test, y_test_pred)
23 test_precision = precision_score(y_test, y_test_pred)
24 test_roc_auc = roc_auc_score(y_test, y_test_pred)
25
26 print("\nTest Set Metrics:")
27 print("Accuracy:", test_accuracy)
28 print("Precision:", test_precision)
29 print("ROC AUC Score:", test_roc_auc)
```

Training Set Metrics:  
Accuracy: 0.9723889555822329  
Precision: 0.9898305084745763  
ROC AUC Score: 0.907120621857379

Test Set Metrics:  
Accuracy: 0.947242206235012  
Precision: 0.9263157894736842  
ROC AUC Score: 0.8470634696755994

Trainig set:

Accuracy: The model achieves an accuracy of approximately 97.24% on the training set, indicating that it correctly predicts the class labels for about 97.24% of the samples in the training data.

Precision: The precision of approximately 98.98% suggests that when the model predicts a positive class (e.g., churn) on the training set, it is correct around 98.98% of the time. In other words, out of all the instances predicted as positive, about 98.98% are truly positive.

ROC AUC Score: The ROC AUC score of around 90.71% indicates that the model has good discriminative power between the positive and negative classes on the training set. It measures the model's ability to distinguish between positive and negative samples, with higher values indicating better performance.

Test Set:

**Accuracy:** On the test set, the model achieves an accuracy of approximately 94.84%, meaning it correctly predicts the class labels for around 94.84% of the samples in the test data.

**Precision:** The precision of approximately 93.61% on the test set suggests that when the model predicts a positive class (e.g., churn), it is correct around 93.61% of the time. In other words, out of all the instances predicted as positive, about 92.63% are truly positive.

**ROC AUC Score:** The ROC AUC score of around 84.78% on the test set indicates that the model has good discriminative power between the positive and negative classes.

Overall, the model performs well on both the training and test sets, with high accuracy, precision, and ROC AUC score. There's a slight drop in performance from the training set to the test set, which is expected, but the model still generalizes well to unseen data.

conclusion on this model based on the provided metrics, there doesn't appear to be a significant difference between the results of the training and test sets. Generally, the performance metrics (such as accuracy, precision, and ROC AUC score) on the test set are close to those on the training set, it indicates that the model is performing well and generalizing effectively to unseen data.

```
In [88]: 1 # Define data for each model
2 data = {
3     'Model': ['Logistic Regression', 'Decision Trees', 'KNN', 'Random Forest', 'Gradient Boosting'],
4     'Train Accuracy': [0.862, 0.954, 1.0, 0.954, 0.972],
5     'Train Precision': [0.557, 0.948, 1.0, 0.942, 0.990],
6     'Train ROC AUC': [0.582, 0.856, 1.0, 0.858, 0.907],
7     'Test Accuracy': [0.844, 0.935, 0.880, 0.935, 0.948],
8     'Test Precision': [0.429, 0.890, 0.766, 0.882, 0.936],
9     'Test ROC AUC': [0.546, 0.817, 0.636, 0.820, 0.848]
10 }
11
12 # Create DataFrame
13 df_summary = pd.DataFrame(data)
14
15 # Convert metrics to percentage form
16 df_summary[['Train Accuracy', 'Train Precision', 'Train ROC AUC',
17             'Test Accuracy', 'Test Precision', 'Test ROC AUC']]
```

Out[88]:

	Model	Train Accuracy	Train Precision	Train ROC AUC	Test Accuracy	Test Precision	Test ROC AUC
0	Logistic Regression	86.2	55.7	58.2	84.4	42.9	54.6
1	Decision Trees	95.4	94.8	85.6	93.5	89.0	81.7
2	KNN	100.0	100.0	100.0	88.0	76.6	63.6
3	Random Forest	95.4	94.2	85.8	93.5	88.2	82.0
4	Gradient Boosting	97.2	99.0	90.7	94.8	93.6	84.8

## Conclusion

Based on the provided metrics in the table above, the Gradient Boosting model consistently outperforms the other models on both the training and test sets, achieving the highest accuracy, precision, and ROC AUC score. Therefore, the Gradient Boosting model is deemed the best among the five models considered for this classification task. The advantages of Gradient model are high accuracy, robustness to overfitting, handling Non-linear relationships and can handle various types of data numerical and categorical features. In conclusion, the gradient boosting model, after thorough evaluation and validation, emerged as the most effective predictive model for identifying customer churn in the SyriaTel telecommunications company dataset. Its high accuracy and robustness make it a valuable tool for informing strategic decisions and implementing retention strategies to reduce customer churn and improve business outcomes.

The 510 area code had the highest percentage rate of 14.88%, the 408 area code had a rate of 14.56%, and the 415 area code had a rate of 14.26%.

Feature importance analysis provided insights into the factors contributing to customer churn, allowing for targeted interventions. The most influential factors to customer churn include the total minutes spent on day calls, the frequency of customer service calls, the charges associated with day calls, the presence of international plans, and the charges for international calls.

## Recommendation

-Utilize Gradient Boosting Model: Given its superior performance in accurately predicting customer churn, the company should prioritize the implementation and deployment of the Gradient Boosting model for ongoing monitoring and prediction of churn.

-Strategic Decision Making: The insights gained from the feature importance analysis highlight specific areas that the company can focus on to mitigate customer churn. For instance, efforts can be directed towards improving customer service quality to reduce the frequency of customer service calls, optimizing international plan offerings, and managing charges associated with day calls.

-Proactive Retention Strategies: Leveraging the predictive power of the Gradient Boosting model, the company can proactively identify customers at risk of churn and implement targeted retention strategies. This may include personalized offers, loyalty programs, or proactive outreach to address customer concerns and enhance satisfaction.

-Continuous Monitoring and Optimization: Customer preferences and behaviors may evolve over time, necessitating continuous monitoring and optimization of the predictive model. Regular updates and refinements based on new data and changing business dynamics will ensure the model remains effective and relevant in predicting churn.

-Investment in Data Infrastructure: To support the deployment of advanced predictive models like Gradient Boosting, the company should invest in robust data infrastructure, including data collection, storage, and processing capabilities. High-quality, well-curated data are essential for training and refining the model for optimal performance.

-Cross-Functional Collaboration: Collaboration between data scientists, business analysts, and operational teams is critical for the successful implementation of predictive analytics