Predicting customer churn: SyraTel 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 Percentange rate of customer churn per area code in the company

-To determine the factors tha mostly contribute to costomer 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, recall, 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:

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).

customer service calls: Number of customer service calls (numerical).

Data cleaning and Explanatory Analysis(EDA)

In [1]:

```
# Import relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClass
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import warnings
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recal
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegressionCV
```

In [2]:

```
# Read the csv file
df = pd.read_csv('cleaned_SyriaTelCustomerchurn.csv')
df.head()

```

Out[2]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tota da charg
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 × 21 columns

Cleaning of the data

In [3]:

```
# check missing values and data type
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 [4]:

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

```
(3333, 21)
```

There are 3333 rows and 21 columns in the dataset

In [5]:
```
# check unique values for each variable in the dataset
for column in df.columns:
    print(f"{column} values: {df[column].unique()} \n")

```

```
state values: ['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'LA' 'WV' 'IN'
'RI' 'IA' 'MT' 'NY'
 'ID' 'VT' 'VA' 'TX' 'FL' 'CO' 'AZ' 'SC' 'NE' 'WY' 'HI' 'IL' 'NH'
'GA'
 'AK' 'MD' 'AR' 'WI' 'OR' 'MI' 'DE' 'UT' 'CA' 'MN' 'SD' 'NC' 'WA'
'NM'
 'NV' 'DC' 'KY' 'ME' 'MS' 'TN' 'PA' 'CT' 'ND']

account length values: [128 107 137  84  75 118 121 147 117 141  6
5  74 168  95  62 161  85  93
  76  73  77 130 111 132 174  57  54  20  49 142 172  12  72  36
78 136
 149  98 135  34 160  64  59 119  97  52  60  10  96  87  81  68 1
25 116
  38  40  43 113 126 150 138 162  90  50  82 144  46  70  55 106
94 155
  80 104  99 120 108 122 157 103  63 112  41 193  61  92 131 163
91 127
 110 140  83 145  56 151 139   6 115 146 185 148  32  25 179  67
```

Generally, the values for all the features are normal. 'International_plan', 'voice_mail_plan', and 'churn' need to be changed to binary of 0 and 1. 'Phone_number' is of no significant and it will be dropped.

In [6]:
```
# check the outliers
df.describe()
```

Out[6]:

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

All values are normal, there are no outliers

In [7]:

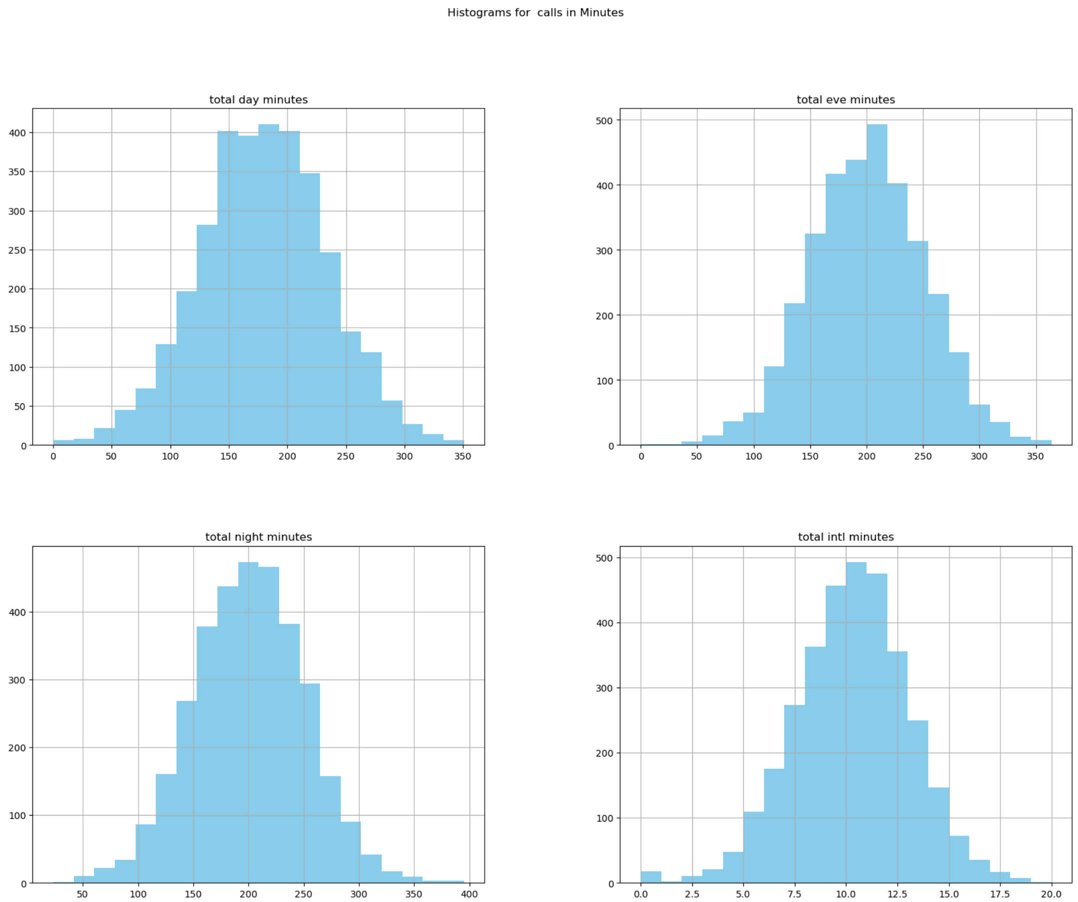
```
# Check for duplicates
duplicates = df.duplicated()
duplicates_count = duplicates.sum()
print("Number of duplicate rows:", duplicates_count)

# Print the rows with duplicates
if duplicates_count > 0:
    duplicate_rows = df[duplicates]
    print("Duplicate rows:")
    print(duplicate_rows[:5])
else:
    print("No duplicate rows found.")
```

```
Number of duplicate rows: 0
No duplicate rows found.
```

No duplicate rows found in the dataset

In [8]:

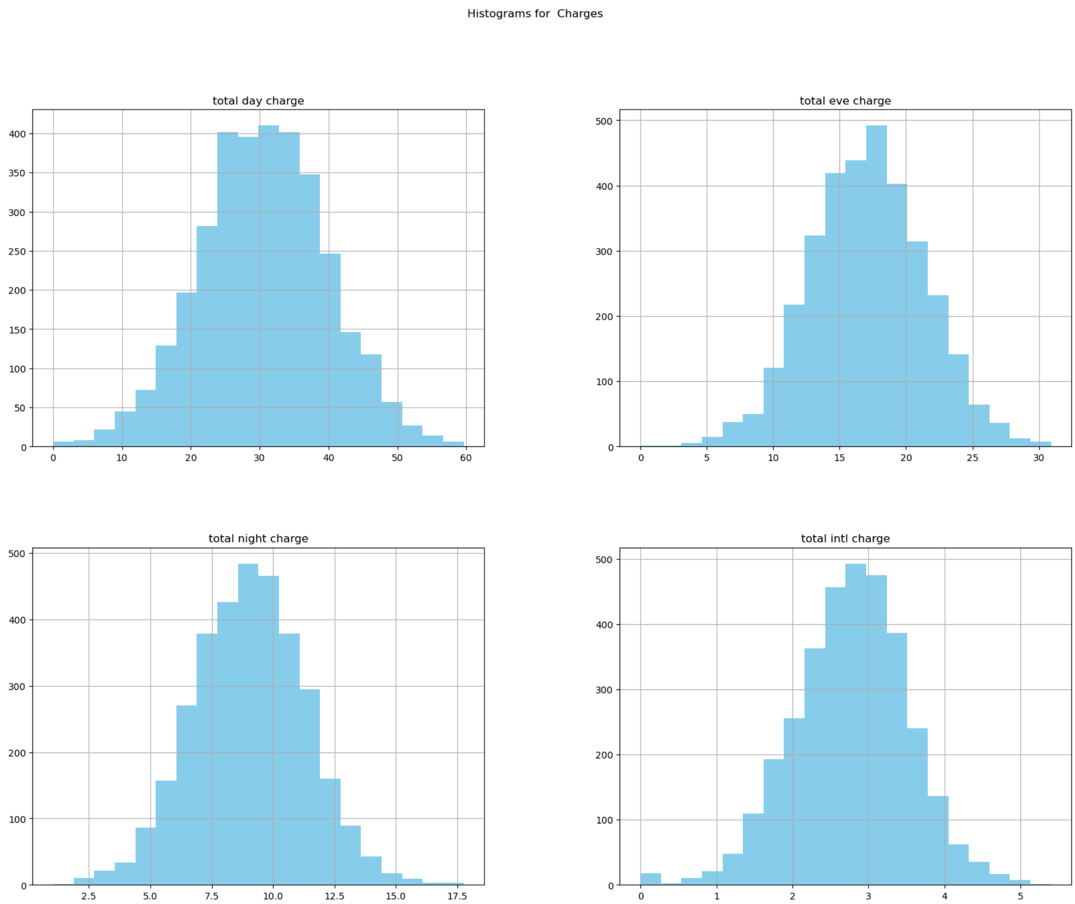
```
# Checking if the total minutes of the day, evening, night and inte
calls_minutes = [ 'total day minutes', 'total eve minutes',
                  'total night minutes',  'total intl minutes'
                  ]
df[calls_minutes].hist(figsize=(20, 15), color = 'skyblue', bins =
plt.suptitle('Histograms for  calls in Minutes')
plt.show()
```



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

In [9]:

```
# Checking if the 'total_day_charge', 'total_eve_charge', 'total_ni
calls_minutes = [ 'total day charge', 'total eve charge',
                  'total night charge',  'total intl charge',
                  ]
df[calls_minutes].hist(figsize=(20, 15), color = 'skyblue', bins =
plt.suptitle('Histograms for  Charges')
plt.show()
```



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 [10]:

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

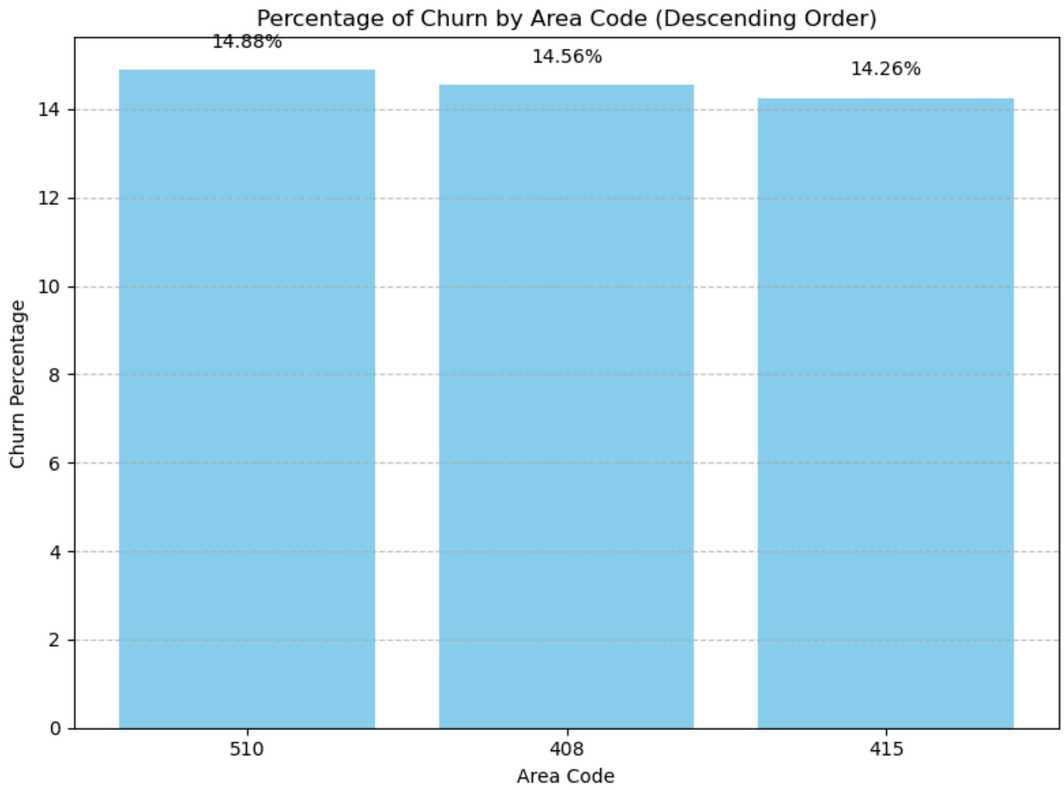
In [11]:

```
# check the percentage rate of churn by are code
churn_percentange_by_area_code = df.groupby('area code')['churn'].m

print(churn_percentange_by_area_code)
```

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

In [12]:

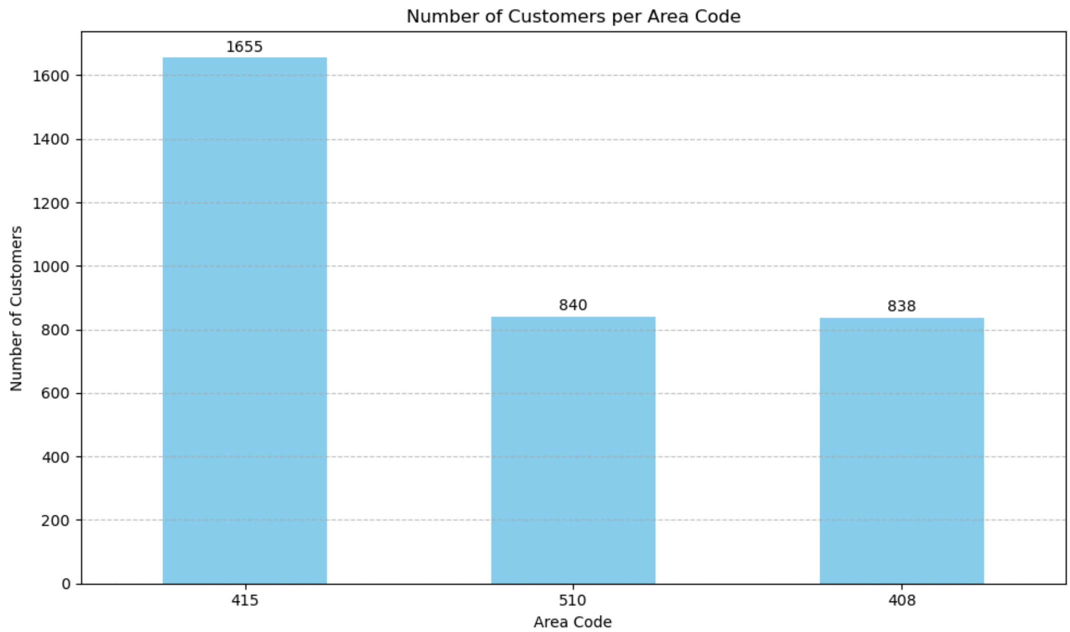
```

# Sort area codes based on churn percentage in descending order
sorted_area_codes = churn_percentange_by_area_code.sort_values(asc

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

# Grouping by sorted area codes and iterating through each group
for area_code in sorted_area_codes:
    # Plotting churn percentage for each area code with sky blue co
    plt.bar(str(area_code), churn_percentange_by_area_code[area_co

    # Adding text labels for percentage
    plt.text(str(area_code), churn_percentange_by_area_code[area_co

# Adding labels and titles
plt.title('Percentage of Churn by Area Code (Descending Order)')
plt.xlabel('Area Code')
plt.ylabel('Churn Percentage')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



The percentange 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 [13]:

```
# Number of customers per area code
customers_per_area_code = df['area code'].value_counts()

# Plotting
plt.figure(figsize=(10, 6))
customers_per_area_code.plot(kind='bar', color='skyblue')

# Adding text labels for number of customers
for x, y in enumerate(customers_per_area_code):
    plt.text(x, y + 10, str(y), ha='center', va='bottom')

plt.title('Number of Customers per Area Code')
plt.xlabel('Area Code')
plt.ylabel('Number of Customers')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



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

In [14]:

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

In [15]:

```
# check the columns after dropping the area code
df.columns
```

Out[15]:

```
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 [16]:
```
df.head()
```

Out[16]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	tot ev call
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 [17]:
```
# convert target variable 'churn' to binary
df['churn'] = df['churn'].map({False: 0, True: 1})
```

In [18]:
```
 # Convert 'international plan' and 'voice mail plan' into binary
df['international plan'].replace(('yes', 'no'), (1, 0), inplace=Tru
df['voice mail plan'].replace(('yes', 'no'), (1, 0), inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

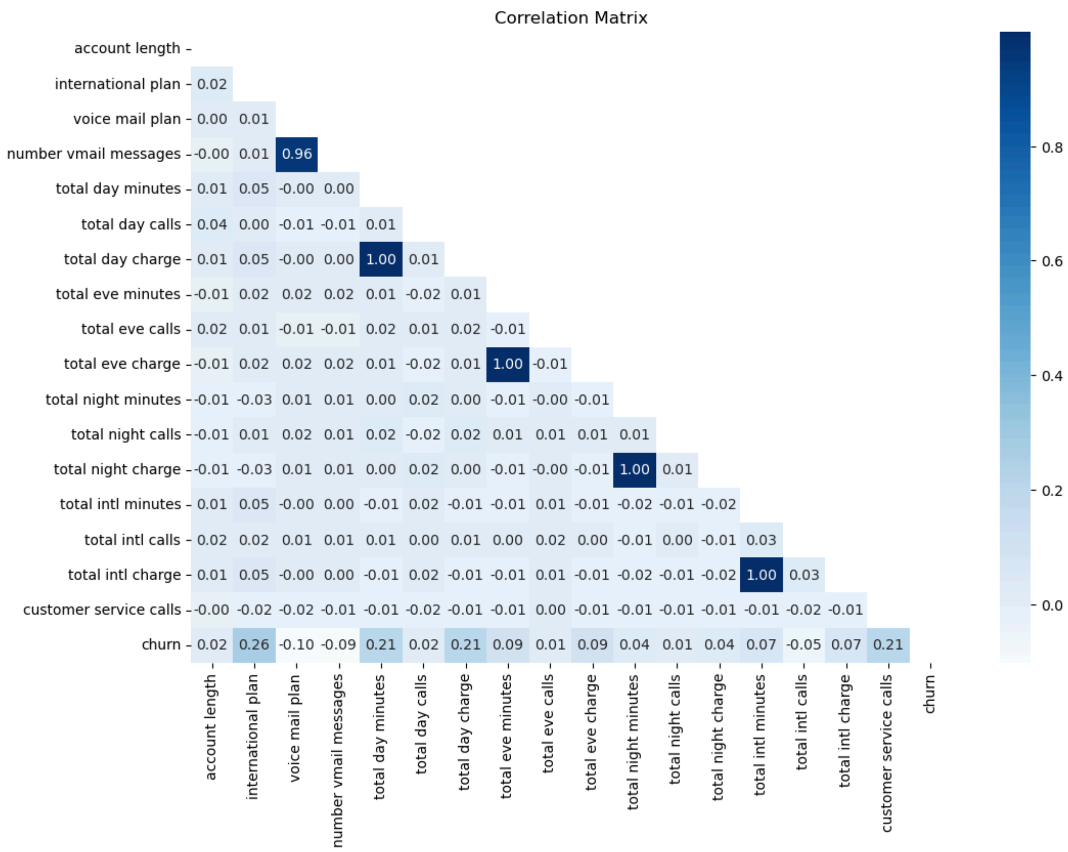
	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	tot ev call
0	KS	128	0	1	25	265.1	110	45.07	197.4	9
1	OH	107	0	1	26	161.6	123	27.47	195.5	10
2	NJ	137	0	0	0	243.4	114	41.38	121.2	11
3	OH	84	1	0	0	299.4	71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

In [20]:
```
# Total number of people who churned/didn't churn
df['churn'].value_counts()
```

Out[20]:
```
churn
0    2850
1     483
Name: count, dtype: int64
```

In [21]:

```
# Suppress warnings
warnings.filterwarnings("ignore")
# Exclude non-numeric columns from the correlation calculation
numeric_df = df.select_dtypes(include='number')

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Create a mask to display only the lower triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(12, 8))

# Plot the heatmap
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='Blues

plt.title('Correlation Matrix')
plt.show()

```



There is a strong relationship between Voice mail plan and number of voice mail messenges, 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 [22]:

```

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in df.columns:
    if df[col].dtype == 'object':  # Check if the column is categor
        df[col] = label_encoder.fit_transform(df[col])

# Display the DataFrame after label encoding
df
```

Out[22]:

	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

spliting data

Splitting data in training and testing sets

Create X, y variables:

In [23]:

```
# Defining X and y
y = df["churn"]
X = df.drop("churn", axis=1)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
```

Transforming the training set

Transforming training data/set prior to fitting the model will prevent data leakage. It will

In [24]:
```
# use min-max scaling
# Your code here
# Instantiate the MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Convert the scaled data back to a pandas DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.c

# Display the scaled data
X_train_scaled_df.head()
```

Out[24]:

	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

Baseline model :Logistic Regression in scikit-learn

fit model

In [25]:
```
# Instantiate the model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='l

# Fit the model
logreg.fit(X_train, y_train)
```

Out[25]:
```
LogisticRegression

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

Generate predictions for the training and test sets.

In [26]:

```
# Generate predictions
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```

In [27]:

```
# Logistic regression on Training set
residuals = np.abs(y_train - y_hat_train)
print(pd.Series(residuals).value_counts())
print('------------------------------------')
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 [28]:

```
# Logistic regression on Testing set
residuals = np.abs(y_test - y_hat_test)
print(pd.Series(residuals).value_counts())
print('------------------------------------')
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 [29]:

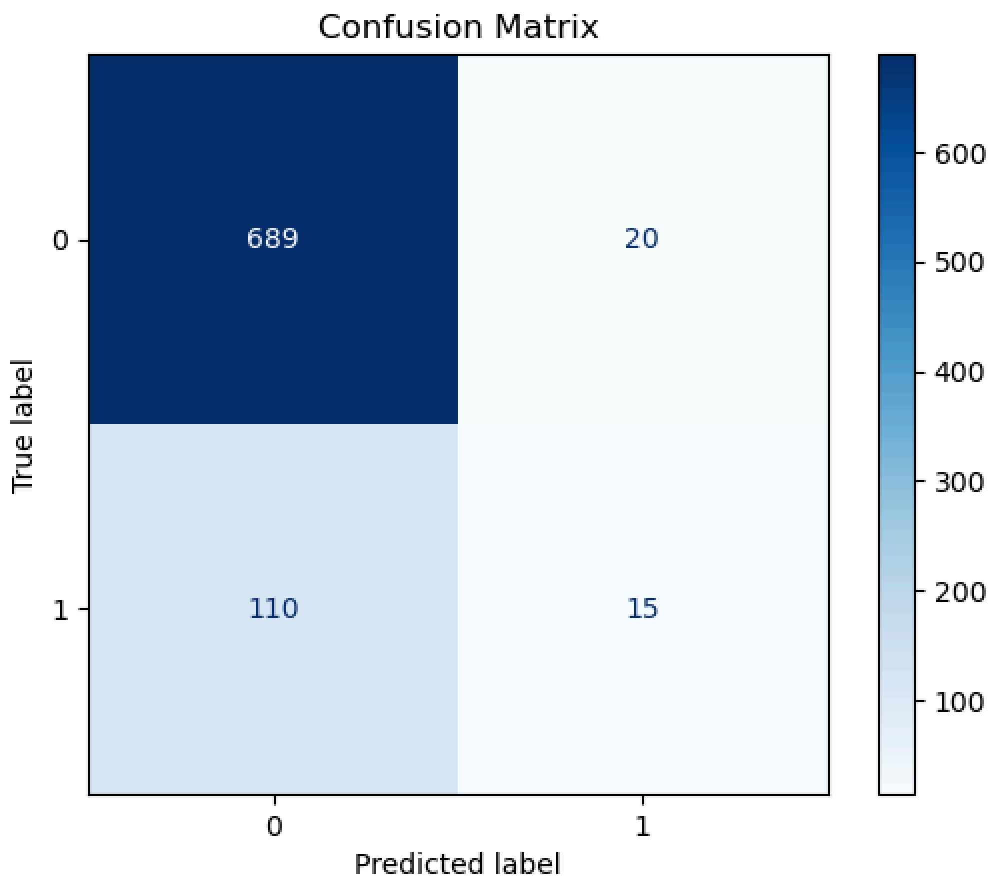
```
# create confusion matrix
def conf_matrix(y_true, y_pred):
    TP = sum((y_true == 1) & (y_pred == 1))
    TN = sum((y_true == 0) & (y_pred == 0))
    FP = sum((y_true == 0) & (y_pred == 1))
    FN = sum((y_true == 1) & (y_pred == 0))

    return {'TP': TP, 'TN': TN, 'FP': FP, 'FN': FN}


# Test the function
conf_matrix(y_test, y_hat_test)
```

Out[29]:

```
{'TP': 15, 'TN': 689, 'FP': 20, 'FN': 110}
```

In [30]:

```
# Import plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matr

cnf_matrix = confusion_matrix(y_test, y_hat_test)
# Visualize your confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



Evaluation of metrics: Logistic regression model

In [31]:

```
def evaluate_model(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    roc_auc = roc_auc_score(y_true, y_pred)
    return accuracy, precision, roc_auc

# Assuming y_train_true and y_train_pred are true and predicted lal
# Assuming y_test_true and y_test_pred are true and predicted label

# Evaluation for training set
accuracy_train, precision_train, roc_auc_train = evaluate_model(y_t

# Evaluation for test set
accuracy_test, precision_test, roc_auc_test = evaluate_model(y_tes

print("Training Set Metrics:")
print("Accuracy: ", accuracy_train)
print("Precision: ", precision_train)
print("ROC AUC Score: ", roc_auc_train)


print("\nTest Set Metrics:")
print("Accuracy: ", accuracy_test)
print("Precision: ", precision_test)
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, indicates 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 [32]:

```
# Train the decision tree
from sklearn.tree import DecisionTreeClassifier
# Create the classifier, fit it on the training data and make pred
clf = DecisionTreeClassifier(random_state=5, criterion='entropy')

clf.fit(X_train, y_train)
```

Out[32]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=5)
```

In [33]:

```
# Make predictions on the train data using the trained classifier
y_train_pred = clf.predict(X_train)

# Calculate evaluation metrics for train set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_roc_auc = roc_auc_score(y_train, y_train_pred)

print("Train Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Make predictions on the test data using the trained classifier
y_pred = clf.predict(X_test)

# Calculate evaluation metrics for test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("\nTest Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
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

Hyperparameter tuning and feature importance analysis

In [34]:

```
# Hyperparameter tuning and feature importance analysis to address
# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the grid search object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_dt_model = grid_search.best_estimator_

# Fit the decision tree model to the entire dataset
best_dt_model.fit(X_train, y_train)

# Get feature importances
feature_importances = best_dt_model.feature_importances_

# Create a DataFrame to store feature importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns,

# Sort the DataFrame by importance values
feature_importance_df = feature_importance_df.sort_values(by='Impo

# Print the feature importances
print(feature_importance_df)
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 5, 'min_sampl
es_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 [35]:

```
columns_to_drop = ['total night calls', 'account length', 'total in
df_filtered = df.drop(columns_to_drop, axis=1)
```

In [36]:

```python
# Evaluate the model after droping less important features
# Make predictions on the train set
y_train_pred = best_dt_model.predict(X_train)

# Calculate evaluation metrics for the train set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_roc_auc = roc_auc_score(y_train, y_train_pred)

print("Train Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Make predictions on the test set
y_pred = best_dt_model.predict(X_test)

# Calculate evaluation metrics for the test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("\nTest Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
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 perfomance, Decision trees performed well based on the Accuracy, precision and ROC AUC score.

Model 3: K- Nearest Neighbours(KNN) Model

In [37]:

```
# Instantiate KNeighborsClassifier
clf = KNeighborsClassifier()

# Fit the classifier
clf.fit(X_train_scaled, y_train)

# Predict on the training set
train_preds = clf.predict(X_train_scaled)

# Predict on the test set
test_preds = clf.predict(X_test.values)
```

Evaluate the model

In [38]:

```
# Accuracy on the training set
train_accuracy = accuracy_score(y_train, train_preds)
print("Training Set Accuracy:", train_accuracy)

# Precision on the training set
train_precision = precision_score(y_train, train_preds)
print("Training Set Precision:", train_precision)

# ROC AUC score on the training set
train_roc_auc = roc_auc_score(y_train, train_preds)
print("Training Set ROC AUC Score:", train_roc_auc)

# Accuracy on the test set
test_accuracy = accuracy_score(y_test, test_preds)
print("\nTest Set Accuracy:", test_accuracy)

# Precision on the test set
test_precision = precision_score(y_test, test_preds)
print("Test Set Precision:", test_precision)

# ROC AUC score on the test set
test_roc_auc = roc_auc_score(y_test, test_preds)
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

In [39]:

```
# use hyperparameter tuning and regularlization to mitigate overfi

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Instantiate the KNN classifier
knn = KNeighborsClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, s

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Instantiate the KNN classifier with the best parameters
best_knn = KNeighborsClassifier(**best_params)

# Fit the best model to the training data
best_knn.fit(X_train, y_train)

# Make predictions on the training set
train_preds = best_knn.predict(X_train)

# Make predictions on the test set
test_preds = best_knn.predict(X_test)

# Evaluate the model on the training set
train_accuracy = accuracy_score(y_train, train_preds)
train_precision = precision_score(y_train, train_preds)
train_roc_auc = roc_auc_score(y_train, train_preds)

print("Training Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_preds)
test_precision = precision_score(y_test, test_preds)
test_roc_auc = roc_auc_score(y_test, test_preds)

print("\nTest Set Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
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, despite regularization and hyperparameter tuning efforts. Explore Random forest to address the overfitting and improve the performance of the model Coclusion: The KNN model performed well as compared to the decision trees model specifically on training set.

Model 4: Random forest

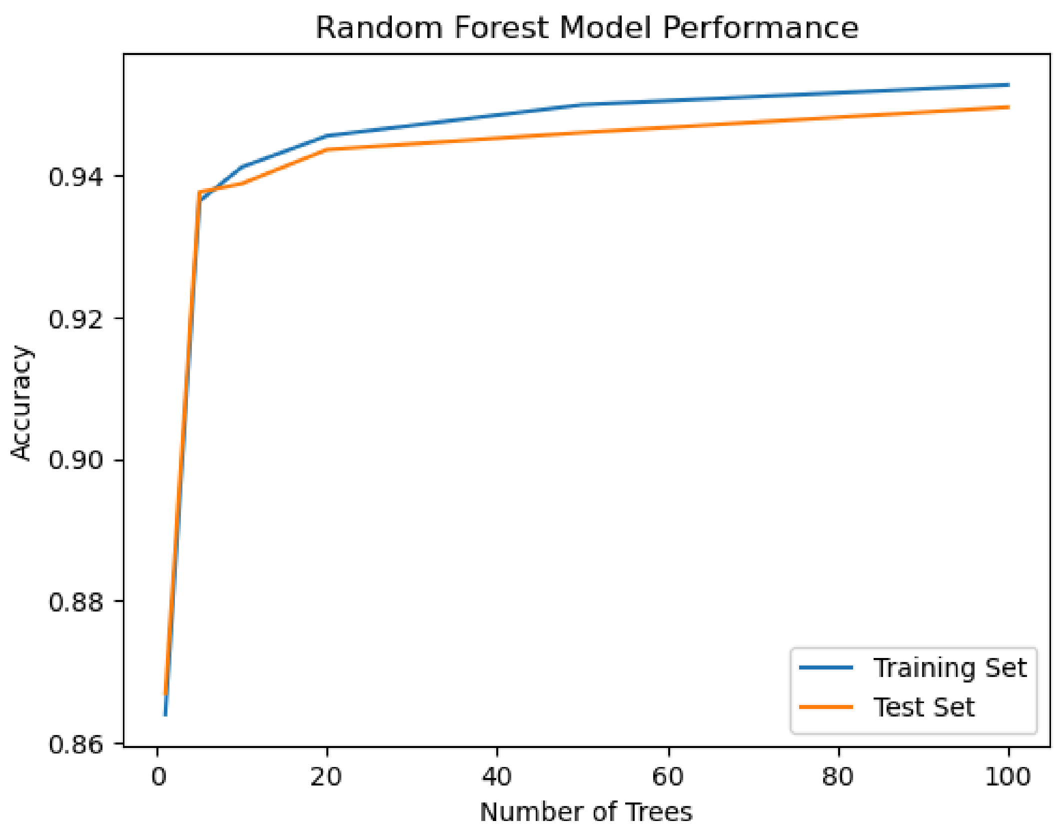
In [40]:

```
# Instantiate the Model
rf_model = RandomForestClassifier(random_state=42)

# Fit the Model
rf_model.fit(X_train, y_train)

# Evaluate performance metrics on the training set
y_train_pred = rf_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_roc_auc = roc_auc_score(y_train, y_train_pred)

print("Training Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Evaluate the Model on test set
y_test_pred = rf_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)

print("\nTest Set Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
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
```

In [41]:

```
# Define a range of values for a hyperparameter (e.g., number of tr
param_range = [1, 5, 10, 20, 50, 100]

# Initialize lists to store mean performance metrics for training a
train_scores_mean = []
test_scores_mean = []

# Iterate over the values of the hyperparameter
for param in param_range:
    # Initialize the model with the current value of the hyperparam
    rf_model = RandomForestClassifier(n_estimators=param, random_st
    
    # Calculate mean cross-validated performance metrics for the tr
    train_scores = cross_val_score(rf_model, X_train, y_train, cv=5
    train_scores_mean.append(train_scores.mean())
    
    # Fit the model on the entire training set
    rf_model.fit(X_train, y_train)
    
    # Calculate performance metrics on the test set
    test_scores_mean.append(accuracy_score(y_test, rf_model.predict

# Plot the performance metrics
plt.plot(param_range, train_scores_mean, label='Training Set')
plt.plot(param_range, test_scores_mean, label='Test Set')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Random Forest Model Performance')
plt.legend()
plt.show()
```



use regulalarlization to mitigate overfitting

In [42]:

```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Instantiate the GridSearchCV object
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=para
                           scoring='accuracy', cv=5, n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Instantiate the decision tree classifier with the best hyperparan
best_dt_model = DecisionTreeClassifier(random_state=42, **best_para

# Fit the model to the training data
best_dt_model.fit(X_train, y_train)

# Evaluate the model
train_accuracy = best_dt_model.score(X_train, y_train)
test_accuracy = best_dt_model.score(X_test, y_test)
```

In [43]:

```
# After regularlization technique evaluate the model
# Evaluate the model on the training set
train_preds = best_dt_model.predict(X_train)
train_accuracy = accuracy_score(y_train, train_preds)
train_precision = precision_score(y_train, train_preds)
train_roc_auc = roc_auc_score(y_train, train_preds)

print("Training Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Evaluate the model on the test set
test_preds = best_dt_model.predict(X_test)
test_accuracy = accuracy_score(y_test, test_preds)
test_precision = precision_score(y_test, test_preds)
test_roc_auc = roc_auc_score(y_test, test_preds)

print("\nTest Set Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
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

model 5: Gradient Boosting

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

In [44]:

```
# Explore Gradient Boosting  to further improve model performance

# Instantiate the model
gb_model = GradientBoostingClassifier()

# Fit the model to the training data
gb_model.fit(X_train, y_train)

# Evaluate performance metrics on the training set
y_train_pred = gb_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_roc_auc = roc_auc_score(y_train, y_train_pred)

print("Training Set Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
print("ROC AUC Score:", train_roc_auc)

# Evaluate the model on the test set
y_test_pred = gb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)

print("\nTest Set Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
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: 0.9484412470023981
Precision: 0.9361702127659575
ROC AUC Score: 0.8477686882933709
```

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.

Based on the performance metrics on both the training and test sets, the Gradient Boosting model generally outperforms the Random Forest model. The Gradient Boosting model exhibits higher accuracy, precision, and ROC AUC score on both the training and test sets

In [45]:

```
# Define data for each model
data = {
    'Model': ['Logistic Regression', 'Decision Trees', 'KNN', 'Ran
    'Train Accuracy': [0.862, 0.954, 1.0, 0.954, 0.972],
    'Train Precision': [0.557, 0.948, 1.0, 0.942, 0.990],
    'Train ROC AUC': [0.582, 0.856, 1.0, 0.858, 0.907],
    'Test Accuracy': [0.844, 0.935, 0.880, 0.935, 0.948],
    'Test Precision': [0.429, 0.890, 0.766, 0.882, 0.936],
    'Test ROC AUC': [0.546, 0.817, 0.636, 0.820, 0.848]
}

# Create DataFrame
df_summary = pd.DataFrame(data)

# Convert metrics to percentage form
df_summary[['Train Accuracy', 'Train Precision', 'Train ROC AUC',
df_summary
```

Out[45]:

	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

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 solutions. Close coordination ensures alignment between predictive insights and strategic business objectives, facilitating the effective execution of churn mitigation strategies.