Business Understanding

Overview:

Due to intense competition in the telecommunications sector, predicting customer churn has become a critical concern for SyriaTel. Retaining customers is vital for maintaining competitiveness, as it is more cost-effective than acquiring new ones. Therefore, it is crucial to develop strategies to prevent customer loss. By analyzing various data points and employing predictive analytics, SyriaTel can identify patterns and indicators to anticipate customer behavior and implement proactive measures to reduce churn rates. SyriaTel seeks to use predictive machine learning models, incorporating demographics such as location and usage patterns like calls and charges, to predict the likelihood of customer churn. It is essential for the company to continuously update these models to reflect evolving customer behavior and market dynamics to ensure the effectiveness of its retention strategies.

Problem Statement:

SyriaTel faces heightened competition in the telecommunications industry, making customer retention a critical concern. Predicting customer churn accurately is essential to retain customers and stay competitive. The company needs effective strategies to prevent customer loss and reduce churn rates.

Proposed Solution:

To address the challenge of predicting customer churn, SyriaTel plans to utilize predictive machine learning models. These models will analyze various data points, including demographics (such as location) and usage patterns (such as calls and charges), to identify patterns and indicators of potential churn. By employing predictive analytics, SyriaTel aims to anticipate customer behavior and implement proactive measures to retain customers. It is crucial to continuously update these models to adapt to evolving customer behavior and market dynamics.

Metrics:

Objectives

- Develop machine learning models to predict customer churn by analyzing customer feature data.
- Compare these models to determine which one offers the highest prediction accuracy.
- The goal of the analysis is to pinpoint specific features that significantly influence the customer churn rate at SyriaTel, and provide valuable recommendations based on these insights to help reduce churn rates and enhance customer retention.

Summary of Features in the Dataset

- state: state residency of the customer
- account length: the number of days the customer has owned the account
- area code: area code of the customer
- phone number: phone number of the customer
- international plan: true if the customer has the international plan, otherwise false
- voice mail plan: true if the customer has the voice mail plan, otherwise false
- number vmail messages: number of voicemails the customer has sent
- total day minutes: total number of minutes the customer has been in calls during the day
- total day calls: total number of calls the user has done during the day
- total day charge: total amount of money the customer was charged by the Telecom company for calls during the day
- total eve minutes: total number of minutes the customer has been in calls during the evening
- total eve calls: total number of calls the customer has done during the evening
- total eve charge: total amount of money the customer was charged by the Telecom company for calls during the evening
- total night minutes: total number of minutes the customer has been in calls during the night
- total night calls: total number of calls the customer has done during the night
- total night charge: total amount of money the customer was charged by the Telecom company for calls during the night
- total intl minutes: total number of minutes the user has been in international calls
- total intl calls: total number of international calls the customer has done
- total intl charge: total amount of money the customer was charged by the Telecom company for international calls
- customer service calls: number of calls the customer made to customer service
- churn: true if the customer terminated their contract, otherwise false

Data Understanding

> This helps to us to acquire knowledge and comprehension about this dataset before further analysis and modeling.

In [2]:
```
#importing relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score, ConfusionMatrixDisplay, classif
from sklearn.preprocessing import StandardScaler
import warnings
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import multiprocessing # for reducing the runtime of gridsearch
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegressionCV

# Ignore warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
#read csv file and check the data
syriatel = pd.read_csv('bigml.csv')
syriatel.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:
```
#check the shape of the data
syriatel.shape
```

Out[4]:
```
(3333, 21)
```

In [5]:
```
# check for data information
syriatel.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
```

- The columns of phone number and account length are not important for this analysis hence we drop them.
- The dataset contains categorical and numerical columns.

Summary of Features in the Dataset

- state: the state the customer lives in
- account length: the number of days the customer has had an account
- area code: the area code of the customer
- phone number: the phone number of the customer
- international plan: true if the customer has the international plan, otherwise false
- voice mail plan: true if the customer has the voice mail plan, otherwise false
- number vmail messages: the number of voicemails the customer has sent
- total day minutes: total number of minutes the customer has been in calls during the day
- total day calls: total number of calls the user has done during the day
- total day charge: total amount of money the customer was charged by the Telecom company for calls during the day

- total eve minutes: total number of minutes the customer has been in calls during the evening
- total eve calls: total number of calls the customer has done during the evening
- total eve charge: total amount of money the customer was charged by the Telecom company for calls during the evening
- total night minutes: total number of minutes the customer has been in calls during the night
- total night calls: total number of calls the customer has done during the night
- total night charge: total amount of money the customer was charged by the Telecom company for calls during the night
- total intl minutes: total number of minutes the user has been in international calls
- total intl calls: total number of international calls the customer has done
- total intl charge: total amount of money the customer was charged by the Telecom company for international calls
- customer service calls: number of calls the customer has made to customer service

Data Cleaning and Preparation

> This section prepares the data for EDA and modeling. The dataset will be checked for:

- duplicated rows
- missing values

In [6]:
```
# missing values
syriatel.isnull().sum()
```

Out[6]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [7]:
```
# check for duplicates
syriatel.duplicated().sum()
```

Out[7]:
```
0
```

Exploratory Data Analysis

In [8]:
```
# the number of unique values for each column in the dataset
syriatel.nunique()
```

Out[8]:
```
state                     51
account length           212
area code                  3
phone number            3333
international plan         2
voice mail plan            2
number vmail messages     46
total day minutes       1667
total day calls          119
total day charge        1667
total eve minutes       1611
total eve calls          123
total eve charge        1440
total night minutes     1591
total night calls        120
total night charge       933
total intl minutes       162
total intl calls          21
total intl charge        162
customer service calls    10
churn                      2
dtype: int64
```

Dropping Some Columns From The Dataset.

> Dropped the account length, phone number, state, and area code columns due to the following reasons:

- The phone number lacks information about customer behavior.
- The account length is not a reliable measure of the client's tenure as a customer in this context. Assuming the account length accurately represents customer loyalty would be invalid without additional information.
- Including the area code and state in the dataset restricts predictions to specific areas, making predictions beyond the defined locations unfeasible.

In [9]:
```
# List of columns to drop
col_drop = ['account length', 'phone number', 'area code', 'state']

# Check which columns exist in the DataFrame
existing_cols = [col for col in col_drop if col in syriatel.columns]

# Drop the existing columns
syriatel.drop(columns=existing_cols, inplace=True)

syriatel.head()
```

Out[9]:

	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
0	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7
1	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4
2	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6
3	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9
4	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9

In [10]:
```
#separating categorical and numerical colums for easier analysis
numerical_df = syriatel[['number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls']]
categorical_df = syriatel[['churn', 'international plan', 'voice mail plar
```

Data exploration

In [11]:
```
# Categorical columns
cat_columns = ['international plan', 'voice mail plan', 'churn']

# Continuous columns
cont_columns = ['number vmail messages', 'total day minutes', 'total day c
                'total day charge', 'total eve minutes', 'total eve calls'
                'total eve charge', 'total night minutes', 'total night ca
                'total night charge', 'total intl minutes', 'total intl ca
                'total intl charge', 'customer service calls']
```

Checking for NaN values

In [12]:
```
#checking for missing values
syriatel.isna().sum()
```

Out[12]:
```
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [13]:
```
#checking for duplicates
syriatel.duplicated().value_counts()
```

Out[13]:
```
False    3333
dtype: int64
```

From the above data checking we have got no missing values and no duplicates value. Our data is ready for further analysis.

Explanatory Data Analysis

In [14]:
```
syriatel.describe()
```

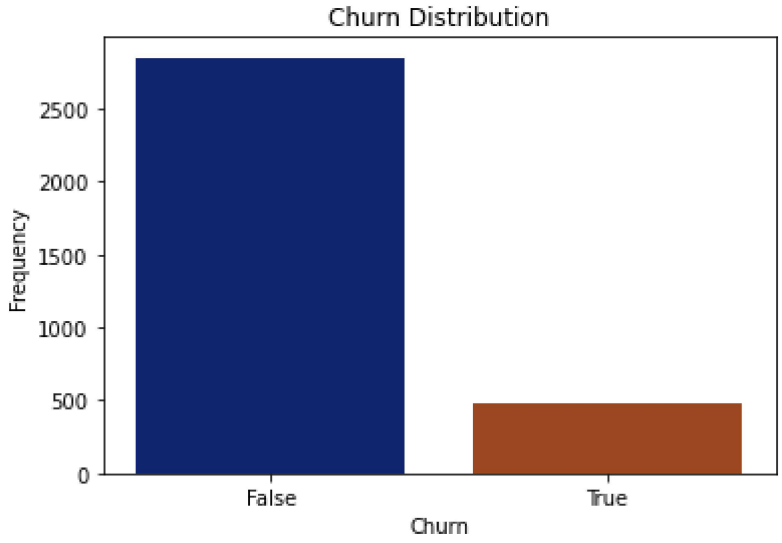
Out[14]:

	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	t
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	333
mean	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17
std	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14
50%	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17
75%	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20
max	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30

This method offers a rapid summary of a dataset's central tendency, dispersion, and distribution shape, making it especially useful for obtaining an overview of the data.

Relationship Visualization

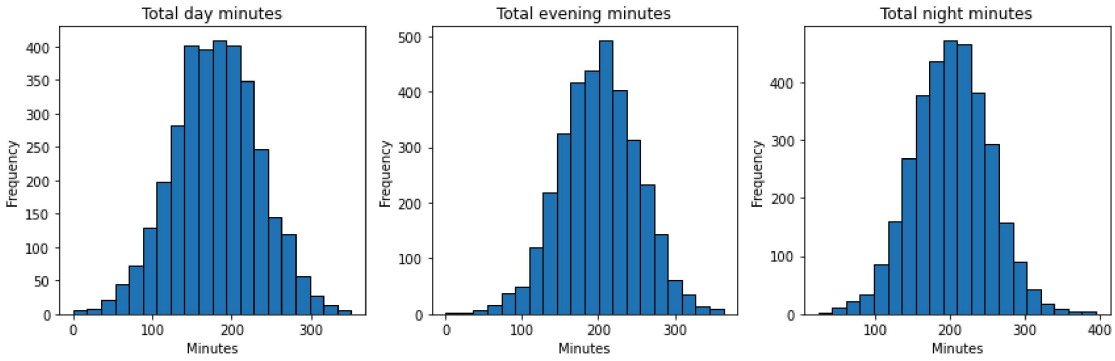
In [15]:
```
# Distribution of Churn
sns.countplot(x='churn', data=syriatel, palette='dark')
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Frequency')
plt.show()

print(syriatel['churn'].value_counts())
```




```
False    2850
True      483
Name: churn, dtype: int64
```

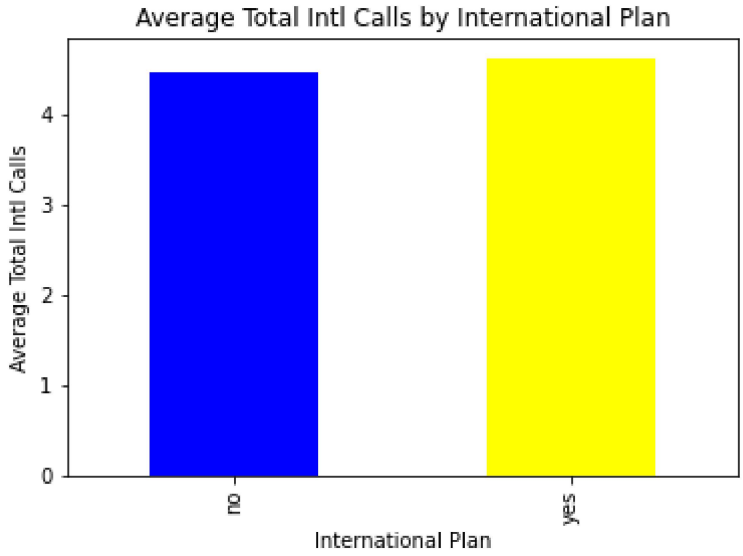
From the above visulization it clearly shows that most of the customers are more loyal to SyriaTel.

In [16]:
```
#total charge of calls per area code

# Create subplots for each variable
fig, axs = plt.subplots(1, 3, figsize=(12, 4))

# Plot histograms for each variable
axs[0].hist(syriatel['total day minutes'], bins=20, edgecolor='k')
axs[0].set_title('Total day minutes')
axs[0].set_xlabel('Minutes')
axs[0].set_ylabel('Frequency')

axs[1].hist(syriatel['total eve minutes'], bins=20, edgecolor='k')
axs[1].set_title('Total evening minutes')
axs[1].set_xlabel('Minutes')
axs[1].set_ylabel('Frequency')

axs[2].hist(syriatel['total night minutes'], bins=20, edgecolor='k')
axs[2].set_title('Total night minutes')
axs[2].set_xlabel('Minutes')
axs[2].set_ylabel('Frequency')

# Adjust spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```



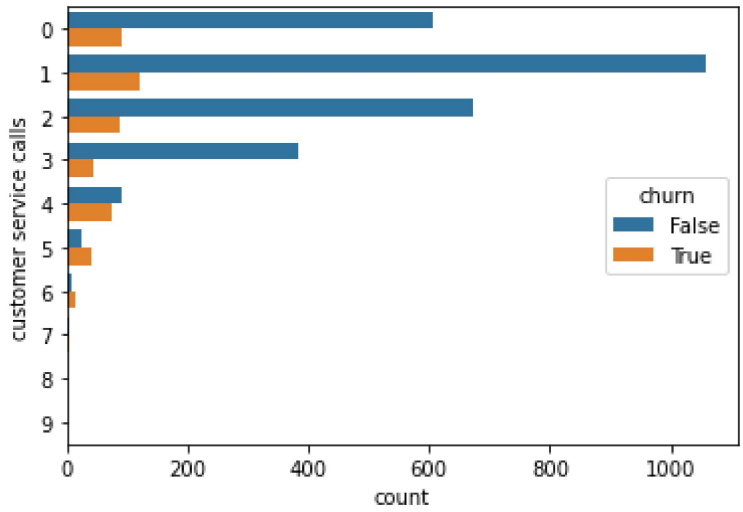
- Distribution of total ninutes a day, in the evening and at night is normal.
- Both the total evening and total night have a mean of around 200 while total day minutes has mean of around 150 from the graph.
- All the total calls made on different times of the day have almost same mean.

In [17]:
```
#the international calls made by people
# Group by 'international plan' and calculate the mean of 'total intl call
grouped_data = syriatel.groupby('international plan')['total intl calls'].

# Plot the bar plot
fig, ax = plt.subplots()
grouped_data.plot(kind='bar', color=['blue', 'yellow'], ax=ax)

# Add labels and title to the plot
ax.set_xlabel('International Plan')
ax.set_ylabel('Average Total Intl Calls')
ax.set_title('Average Total Intl Calls by International Plan')

# Display the plot
plt.show()
```



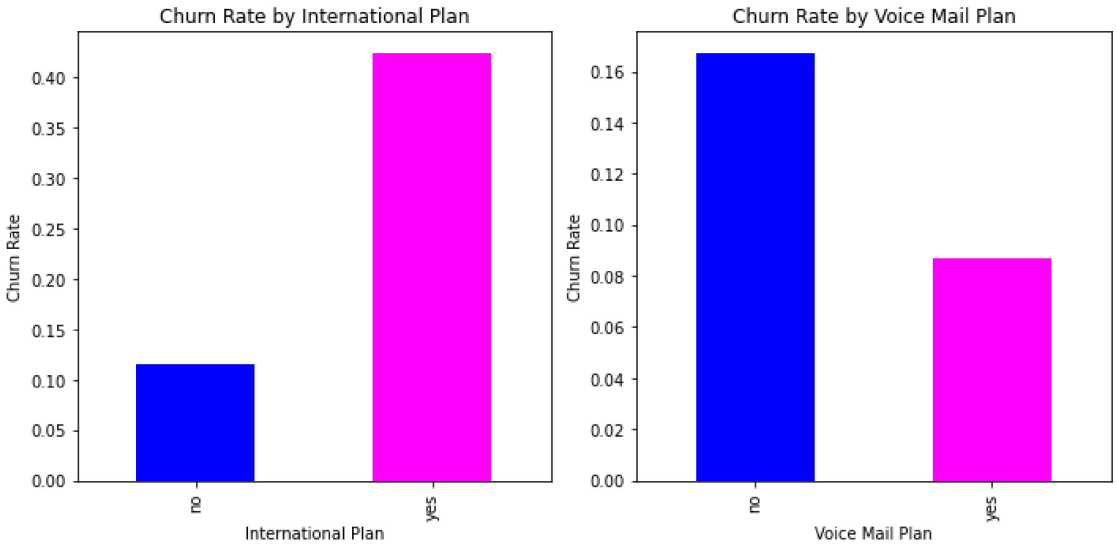
In [18]:
```
#Relationship between the number of calls to the call center and loyalty

sns.countplot(y='customer service calls', hue='churn', data=syriatel)
```

Out[18]: <AxesSubplot:xlabel='count', ylabel='customer service calls'>



The chart above depicts the relationship between the number of calls to the call center and loyalty. It shows that there is higher relationship between calls and loyalty, most people making the calls are loyal to Syriatel hence low probalility of switching.

In [19]:
```
#visualizing for the international calls made by people
# Map the values in 'voice mail plan' column to labels

# Create a figure with one row and two columns
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Plot the first subplot: International Plan
syriatel.groupby('international plan')['churn'].mean().plot(kind='bar', ax
axes[0].set_xlabel('International Plan')
axes[0].set_ylabel('Churn Rate')
axes[0].set_title('Churn Rate by International Plan')

# Plot the second subplot: Voice Mail Plan
syriatel.groupby('voice mail plan')['churn'].mean().plot(kind='bar', ax=ax
axes[1].set_xlabel('Voice Mail Plan')
axes[1].set_ylabel('Churn Rate')
axes[1].set_title('Churn Rate by Voice Mail Plan')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the plot
plt.show()
```



- Looking at the graph above it shows that more customers loyalty is beacause of Voice mail plan subscribers due to lower churn rate
- Customers have higher rate of churn from Syriatel in terms of international plan.
- This shows that customers are more happy with voice mail than international plan.

Outliers

In [24]:
```
def find_outliers(df):
    outliers = {}
    for column in df.columns:
        data = df[column].dropna()  # Drop NaN values
        q1 = np.percentile(data, 25)
        q3 = np.percentile(data, 75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

        outliers[column] = data[(data < lower_bound) | (data > upper_bound
    return outliers

# Example usage with numerical_df
outliers_sum = find_outliers(numerical_df)
print(outliers_sum)
```

```
{'number vmail messages': 51, 'total day minutes': 3928.2000000000003,
'total day calls': 1807, 'total day charge': 667.8100000000001, 'total e
ve minutes': 4175.9, 'total eve calls': 1836, 'total eve charge': 354.95
000000000005, 'total night minutes': 6180.799999999999, 'total night cal
ls': 2647, 'total night charge': 278.13, 'total intl minutes': 286.59999
999999997, 'total intl calls': 1001, 'total intl charge': 91.57000000000
001, 'customer service calls': 1223}
```

In [25]:
```
import numpy as np

def find_outliers(data):
    """
    Finds outliers in a dataset using the interquartile range (IQR) method

    Parameters:
    - data: A list or numpy array of numerical values.

    Returns:
    - outliers: A list of the outlier values.
    """
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    outliers = [x for x in data if x < lower_bound or x > upper_bound]
    return outliers
```

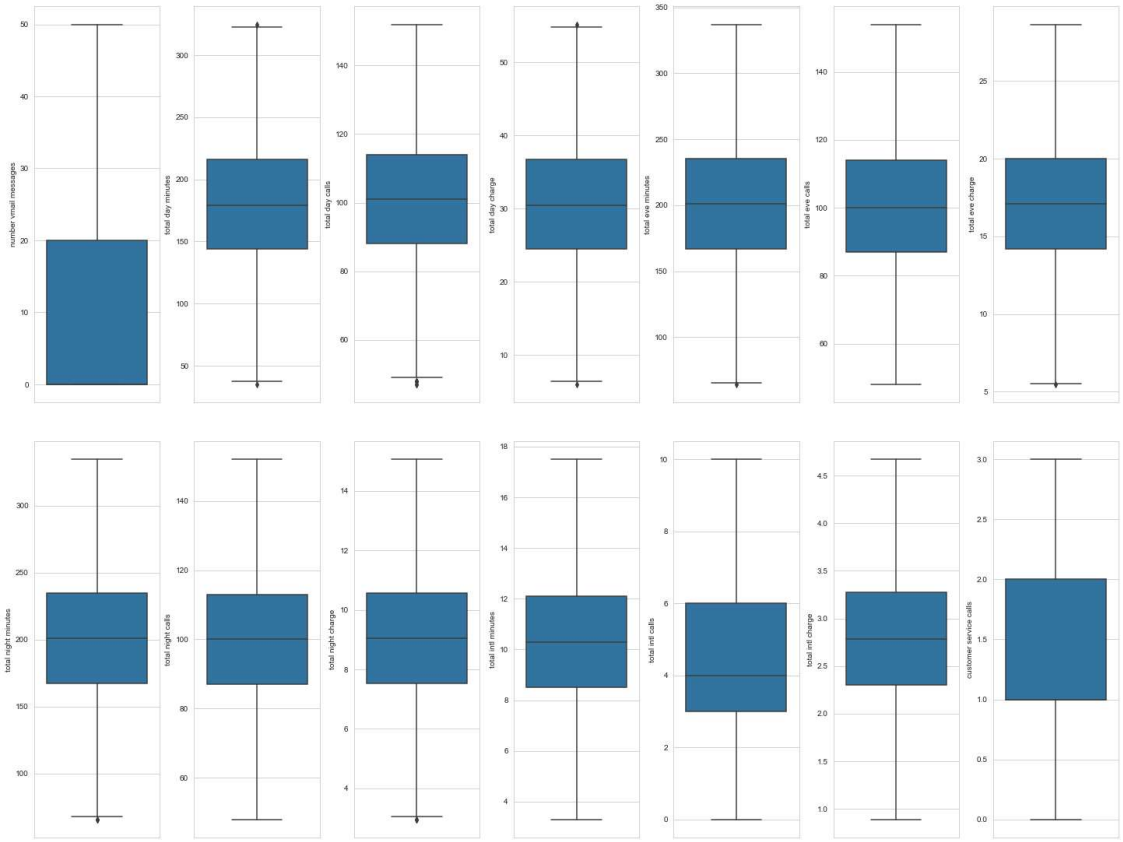
In [28]:
```
def find_outliers(df):
    outliers = {}
    for column in df.columns:
        # Ensure the column is numeric
        if pd.api.types.is_numeric_dtype(df[column]):
            data = df[column].dropna()  # Drop NaN values
            q1 = np.percentile(data, 25)
            q3 = np.percentile(data, 75)
            iqr = q3 - q1
            lower_bound = q1 - 1.5 * iqr
            upper_bound = q3 + 1.5 * iqr

            outliers[column] = data[(data < lower_bound) | (data > upper_b
        else:
            outliers[column] = 0  # If the column is not numeric, set outl
    return outliers

# Example usage with numerical_df
outliers_sum_dict = find_outliers(numerical_df)
outliers_sum_total = sum(outliers_sum_dict.values())
print(outliers_sum_total)
```

```
24528.96
```

In [29]:
```
def remove_outliers(data):
    # Calculating the quartiles and IQR
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    # Determining the lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Removing outliers
    filtered_data = data[(data >= lower_bound) & (data <= upper_bound)]

    return filtered_data
```

In [30]:
```
#using the function above to remove outliers
filtered_data = remove_outliers(numerical_df)
```

In [31]:
```
#checking for any outliers in the dataset using box plot
# visualizing with matplotlib and seaborn
sns.set_style('whitegrid')
fig,ax = plt.subplots(ncols=7,nrows =2,figsize = (20,15))
index = 0
ax = ax.flatten()
for col,value in filtered_data.items():
    sns.boxplot(y = col,data = filtered_data,ax = ax[index])
    index += 1
plt.tight_layout(pad= 0.5,w_pad= 0.7,h_pad=5.0)

plt.savefig("Images/outliers.png")
```



Data Preparation

Important features

We will use heatmap to determine correlation which help in identfying the important features

In [65]:
```
#using heatmap to check for correlation
correlation_matrix = numerical_df.corr()
plt.figure(figsize=(20, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu_r', fmt=".2f", line
plt.title('Correlation Matrix')
plt.show()
```

- As seen from the heatmap there are some variables that have 1 that is perfect correlation with other variables.
- Multicollinearity will occur if this predictor variables are highly correlated, which can lead to instability and unreliable estimates of the model parameters.
- Hence we will remove one of the correlacetd factors which is total day minutes, total eve minutes, /total night minutes and total intl minutes

In [68]:
```
#droping features with perfect correlation using our previous seriatel_df
syriatel_1 = syriatel.drop(columns=['total day minutes', 'total eve minute
```

In [69]:
```
#checking for the shape of the data
syriatel_1.shape
```

Out[69]: (3333, 14)

- The final values from here are the ones to be used for modelling since the 15 columns are suitable for analysis

Transforming categorical variables using OHE

In [111]:
```
#transformimg categorical data using OHE

# Select the categorical columns to be one-hot encoded
categorical_df = ['international plan', 'voice mail plan']

# Create an instance of the OneHotEncoder
encoder = OneHotEncoder()

# Fit and transform the categorical columns
encoded_data = encoder.fit_transform(syriatel_1[categorical_df])

# Convert the encoded data to a DataFrame
encoded_df = pd.DataFrame(encoded_data.toarray(), columns=encoder.get_feat

# Concatenate the encoded DataFrame with the remaining columns from the or
syriatel_data = pd.concat([syriatel_1.drop(categorical_df, axis=1), encode

syriatel_data
```

Out[111]:

	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	total intl charge	customer service calls	chu
0	25	110	45.07	99	16.78	91	11.01	3	2.70	1	Fals
1	26	123	27.47	103	16.62	103	11.45	3	3.70	1	Fals
2	0	114	41.38	110	10.30	104	7.32	5	3.29	0	Fals
3	0	71	50.90	88	5.26	89	8.86	7	1.78	2	Fals
4	0	113	28.34	122	12.61	121	8.41	3	2.73	3	Fals
...	...	...	...	...	...	...	...	...	...	...	
3328	36	77	26.55	126	18.32	83	12.56	6	2.67	2	Fals
3329	0	57	39.29	55	13.04	123	8.61	4	2.59	3	Fals
3330	0	109	30.74	58	24.55	91	8.64	6	3.81	2	Fals
3331	0	105	36.35	84	13.57	137	6.26	10	1.35	2	Fals
3332	25	113	39.85	82	22.60	77	10.86	4	3.70	0	Fals

3333 rows × 16 columns

In [112]:
```
#convert churn using label ecoder using a function
def encode(column):
    le = LabelEncoder()
    syriatel_data[column] = le.fit_transform(syriatel_data[column])
#encoding the column
encode('churn')
#checking for encorded churn column
syriatel_data.churn.value_counts()
```

Out[112]:
```
0    2850
1     483
Name: churn, dtype: int64
```

Train-Test Splitting

> This will be used to test the data and used to evaluate the performance of the trained model on unseen data. By evaluating the model on the test set, we can get an estimate of how well the model generalizes to new, unseen data.

In [113]:
```
# Split the data into features (X) and target variable (y)
X = syriatel_data.drop(columns='churn', axis=1)
y = syriatel_data['churn']

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

Data preprocessing

Standardization

- We will employ standardization to rescale the features of a SyriaTel dataset so that they have a mean of zero and a variance of one.
- This process helps to normalize the features to a similar scale, which can be advantageous for machine learning algorithms that are sensitive to the scale of input features.

In [114]:
```
# Create an instance of StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data
scaler.fit(X_train)

# Transform the training and test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Use **SMOTE** to solve for class imbalance

In [115]:
```
smote = SMOTE(random_state=42)
#SMOTE only to the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

Baseline Logistic Regression Model

In [117]:
```
# Create an instance of Logistic Regression
logreg = LogisticRegression(solver='liblinear', random_state=42)
model_log = logreg.fit(X_train_smote,y_train_smote)
model_log
model_log.coef_
```

Out[117]:
```
array([[-3.96581418e-03, -4.78234646e+00,  6.38290266e-02,
        -4.77944866e+00,  5.44269267e-02, -4.78333670e+00,
         6.81661753e-02, -4.91563971e+00,  2.83907631e-01,
         4.47070818e-01,  4.78052831e+00, -2.35939048e+00,
        -2.20934610e-01, -8.84012263e-01, -1.69631283e+00]])
```

In [119]:
```
# Predict on the training and testing data
y_train_pred = logreg.predict(X_train_smote)
y_test_pred_1 = logreg.predict(X_test)

# Calculate accuracy on the training and testing data
train_accuracy = accuracy_score(y_train_smote, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred_1)
```

In [121]:
```
#creating a function for checking for metrics
def evaluate_model_metrics(model, X_train, y_train, X_test, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the training and testing data
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate evaluation metrics
    roc_auc_train = roc_auc_score(y_train, y_train_pred)
    roc_auc_test = roc_auc_score(y_test, y_test_pred)
    cm_test = confusion_matrix(y_test, y_test_pred)
    cm_display_train = ConfusionMatrixDisplay(confusion_matrix=cm_test).pl
    accuracy_train = accuracy_score(y_train, y_train_pred)
    accuracy_test = accuracy_score(y_test, y_test_pred)

    # Return results
    results = {
        'roc_auc_train': roc_auc_train,
        'roc_auc_test': roc_auc_test,
        'accuracy_train': accuracy_train,
        'accuracy_test': accuracy_test,
        'confusion_matrix_train': cm_display_train
    }
    return results
```

In [122]:
```
#creating a function for checking for classification report
def generate_classification_report(y_true, y_pred):
    # Generate classification report with output_dict=True
    report_dict = classification_report(y_true, y_pred, output_dict=True)

    # Convert the report to a DataFrame
    report = pd.DataFrame(report_dict).transpose()

    return report
```

In [123]:
```
# calling the function to get classification report values
logreg_report = generate_classification_report(y_test, y_test_pred_1)
logreg_report
```

Out[123]:

	precision	recall	f1-score	support
0	0.880837	0.966431	0.921651	566.00000
1	0.586957	0.267327	0.367347	101.00000
accuracy	0.860570	0.860570	0.860570	0.86057
macro avg	0.733897	0.616879	0.644499	667.00000
weighted avg	0.836337	0.860570	0.837716	667.00000

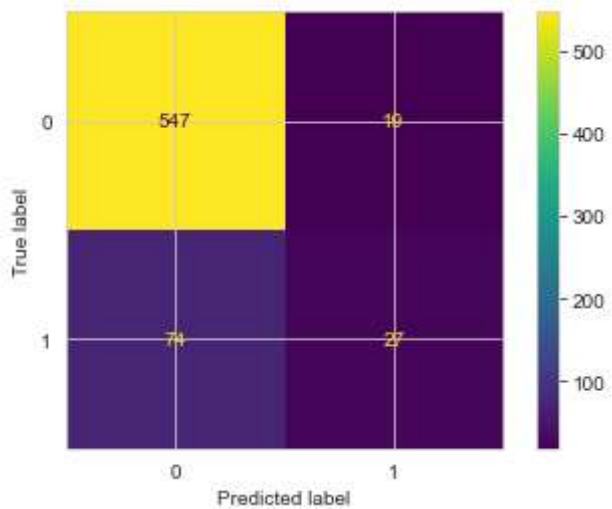
- Precision: The precision values for class 0 and class 1 are 0.88 and 0.587, respectively. Higher precision signifies a lower rate of false positives for that class. With a higher precision in class 0, the model demonstrates better performance in predicting class 0 than class 1.
- Recall: The recall values for class 0 and class 1 are 0.966 and 0.267, respectively. Recall measures the model's ability to correctly identify positive instances. Like precision, recall is higher for class 0, indicating better performance in identifying class 0 instances compared to class 1.
- F1-Score: The F1-scores for class 0 and class 1 are 0.92 and 0.367, respectively. The F1-score is the harmonic mean of precision and recall, balancing both metrics. Once again, class 0 has a higher F1-score than class 1.
- Accuracy: The model's accuracy is 0.861, meaning that 86.1% of the predictions were correct out of all instances.

> Therefore, logistic regression achieves 86.1% prediction accuracy on the test data.

> Based on these metrics, it is evident that the model performs better for class 0 compared to class 1.

In [125]:
```
#metric of baseline model and draw confusion matrix
evaluate_model_metrics(logreg, X_train_smote,y_train_smote, X_test, y_test
```

Out[125]:
```
{'roc_auc_train': 0.8938266199649737,
 'roc_auc_test': 0.6168789140398139,
 'accuracy_train': 0.8938266199649737,
 'accuracy_test': 0.8605697151424287,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Confu
sionMatrixDisplay at 0x123cc42be20>}
```



Model Performance Summary

The logistic regression model demonstrates the following performance metrics:

- **ROC AUC**:
  - Training: 0.8938
  - Test: 0.6169
- **Confusion Matrix**:
  - True Positives (TP): 27
  - False Negatives (FN): 74
  - True Negatives (TN): 547
  - False Positives (FP): 10
- **Accuracy**:
  - Training: 89.4%
  - Testing: 86.0%

Observations:

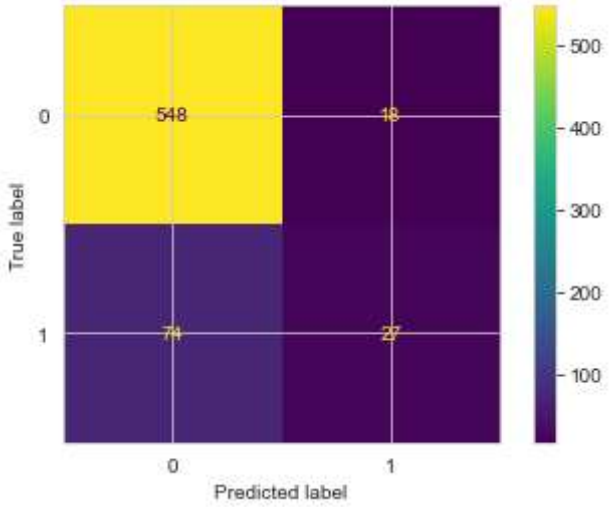
- The model shows a high level of discrimination on the training data but significantly lower on the test data, suggesting potential overfitting.
- The confusion matrix indicates a considerable number of false negatives, highlighting the need for model improvement.

Cross-Validation

In [127]:
```
# Create an instance of Logistic Regression with cross-validation
logreg_final = LogisticRegressionCV(Cs=10, cv=5, solver='liblinear')

# Fit the model on the resampled training data
logreg_final.fit(X_train_smote, y_train_smote)

# Predict on the resampled training and testing data
y_train_pred = logreg.predict(X_train_smote)
y_test_pred = logreg.predict(X_test)

# Calculate accuracy on the resampled training and testing data
train_accuracy = accuracy_score(y_train_smote, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

In [128]:
```
evaluate_model_metrics(logreg_final,X_train_smote,y_train_smote,X_test,y_t
```

Out[128]:
```
{'roc_auc_train': 0.8922942206654991,
 'roc_auc_test': 0.6177623062659623,
 'accuracy_train': 0.8922942206654991,
 'accuracy_test': 0.8620689655172413,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Confu
sionMatrixDisplay at 0x123ccac9e80>}
```



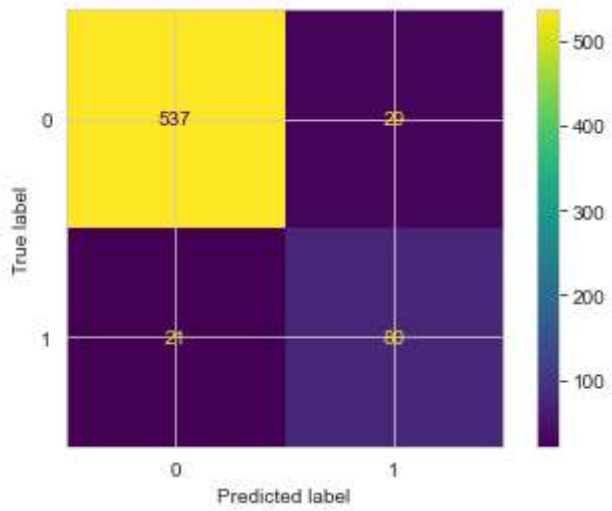
- After applying cross-validation with five folds, we obtained an improved model with an average accuracy of 0.8621, representing an 86.2% accuracy level in predicting customer churn on the test data. The training data accuracy also adjusted to 0.8621.
- This adjusted model shows no significant change in performance, accurately predicting the class labels for most instances in both training and testing datasets. The testing accuracy is slightly lower than the training accuracy, which is expected but not significantly different.
- Therefore, we can proceed to evaluate the second model to determine if it performs better in predicting customer churn compared to logistic regression.

Decison Trees classifier model

In [129]:
```
#DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(max_depth=5, min_samples_split=5)

# Fit the model on the training data
dt_clf.fit(X_train_smote, y_train_smote)

# Predict on the training and testing data
y_train_pred_2= dt_clf.predict(X_train_smote)
y_test_pred_2 = dt_clf.predict(X_test)

# Calculate accuracy on the training and testing data
train_accuracy = accuracy_score(y_train_smote, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred_2)
```

In [130]:
```
#checking for decison tree metrics using the predefiend function
evaluate_model_metrics(dt_clf, X_train_smote, y_train_smote, X_test, y_tes
```

Out[130]:
```
{'roc_auc_train': 0.8857267950963224,
 'roc_auc_test': 0.8704212294020922,
 'accuracy_train': 0.8857267950963222,
 'accuracy_test': 0.9250374812593704,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Confu
sionMatrixDisplay at 0x123cd32fbe0>}
```



Classification Report Summary

The logistic regression model exhibits the following performance metrics:

- **Precision**:
  - Class 0: 96.2%
  - Class 1: 73.3%
- **Recall**:
  - Class 0: 94.9%
  - Class 1: 79%

- **F1-score**:
  - Class 0: 95.6%
  - Class 1: 76.2%
- **Accuracy**:
  - Overall: 92.5%
- **Macro Average**:
  - Precision: 84.8%
  - Recall: 87.0%
  - F1-score: 85.9%
- **Weighted Average**:
  - Precision: 92.8%
  - Recall: 92.5%
  - F1-score: 92.6%

Observations:

- High precision and recall for class 0, indicating strong performance in identifying true negatives.
- Reasonable precision and recall for class 1, though with room for improvement in identifying true positives.
- Overall accuracy of 92.5%, suggesting robust performance across all instances.
- Balanced macro and weighted averages, indicating consistent performance across both classes.

> To gain further insights, we will examine the classification report for all metrics related to the Decision Tree classifier.

In [131]:
```
#using predefined function to check for classification report
dt_clf_report = generate_classification_report(y_test, y_test_pred_2)
dt_clf_report
```

Out[131]:

	precision	recall	f1-score	support
0	0.962366	0.948763	0.955516	566.000000
1	0.733945	0.792079	0.761905	101.000000
accuracy	0.925037	0.925037	0.925037	0.925037
macro avg	0.848155	0.870421	0.858710	667.000000
weighted avg	0.927777	0.925037	0.926199	667.000000

- Precision: In class 0, the precision is 0.962, indicating that 96.2% of the instances predicted as class 0 are actually true negatives. In class 1, the precision is 0.733, meaning that 73.3% of the instances predicted as class 1 are true positives.
- Recall: In class 0, the recall is 0.949, indicating that 94.9% of the actual class 0 instances are correctly identified as true negatives. In class 1, the recall is 0.79, meaning that 79% of the actual class 1 instances are correctly identified as true positives.

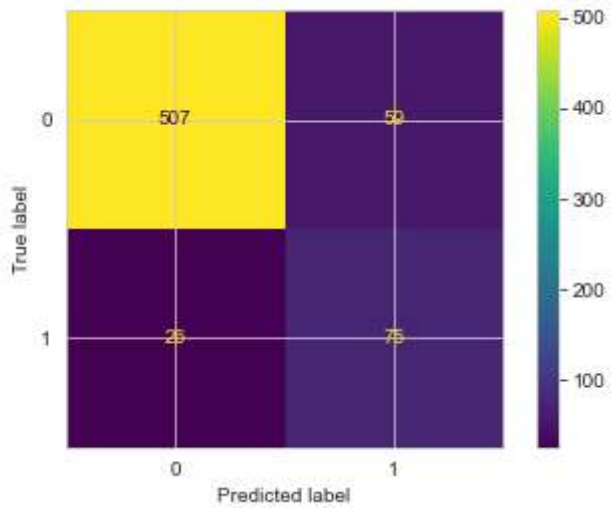
- F1-score: In class 0, the F1-score is 0.956, indicating a good balance between precision and recall for class 0. In class 1, the F1-score is 0.762, suggesting a slightly lower balance between precision and recall for class 1.
- Accuracy: Accuracy is the overall proportion of correctly classified instances. In this case, the accuracy is 0.925, meaning that the model correctly predicts the class labels for 92.5% of the instances.
- Macro avg: Macro average calculates the average metrics (precision, recall, F1-score) for both classes, giving equal weight to each class which are 0.848155, 0.870421 and 0.858710 while Weighted avg calculates the average metrics, taking into account the support (number of instances) for each class. It provides a weighted average based on the

Random forest model

In [132]:
```
#random forest classifier with regularization parameters
forest_classifier = RandomForestClassifier(n_estimators=100, max_depth=5,

# Fit the model on the selected training data
forest_classifier.fit(X_train_smote, y_train_smote)

# Predict on the training and testing sets
y_train_pred_3 = forest_classifier.predict(X_train_smote)
y_test_pred_3 = forest_classifier.predict(X_test)

# Calculate training and testing accuracy
train_accuracy = accuracy_score(y_train_smote, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred_3)
```

In [134]:
```
#checking for random forest metrics using the predefiend function
evaluate_model_metrics(forest_classifier, X_train_smote, y_train_smote, X_
```

Out[134]:
```
{'roc_auc_train': 0.8712784588441331,
 'roc_auc_test': 0.8191669873701151,
 'accuracy_train': 0.8712784588441331,
 'accuracy_test': 0.8725637181409296,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Confu
sionMatrixDisplay at 0x123cd0eea00>}
```



- The Random Forest classifier achieves an accuracy of approximately 87.1% on the training data and 87.2% on the testing data. -- It performs well in distinguishing between positive and negative classes, with an area under the ROC curve (AUC) of 0.87 for the training data and 0.82 for the testing data. Overall, the model demonstrates a high level of accuracy in predicting the target variable.
- According to the confusion matrix, there are 75 true positives, 507 true negatives, 50 false positives, and 26 false negatives. This accuracy is slightly lower compared to the Decision Tree classifier.

In [135]:
```
generate_classification_report(y_test, y_test_pred_3)
```

Out[135]:

	precision	recall	f1-score	support
0	0.951220	0.895760	0.922657	566.000000
1	0.559701	0.742574	0.638298	101.000000
accuracy	0.872564	0.872564	0.872564	0.872564
macro avg	0.755461	0.819167	0.780477	667.000000
weighted avg	0.891934	0.872564	0.879598	667.000000

> For class 0 (negative class):

- Precision: A value of 95.1% means that when the model predicts the negative class, it is correct 95.1% of the time.
- Recall: At 89.6%, this indicates that the model correctly identifies 89.6% of the actual negative instances.
- F1-score: With a value of 92.2%, this metric provides a balanced measure of precision and recall, combining both into a single value.
- Support: There are 566 instances of the negative class in the dataset.

> For class 1 (positive class):

- Precision at 55.6% means that when the model predicts the positive class, it is accurate 55.6% of the time.
- Recall at 74.2% indicates that the model correctly identifies 74.2% of the actual positive instances.
- The F1-score at 63.8% provides a balanced measure of precision and recall for the positive class.
- Support indicates there are 101 instances of the positive class in the dataset.
- The Random Forest classifier achieves an overall accuracy of about 87.3%, showing the percentage of correctly predicted instances overall.

Summary

- Based on the analysis of the three models, it is evident that logistic regression performs poorly in predicting customer churn.
- In contrast, both the Random Forest classifier and Decision Trees perform well with accuracies of 87.3% and 92.5%, respectively.
- Therefore, it is crucial to enhance the performance of the Random Forest classifier and Decision Trees by tuning hyperparameters to achieve even better accuracy.
- Hyperparameters are essential for improving the efficiency and performance of models.

Hyperparameter Tuning

- Hyperparameters are settings that are predetermined and not learned from the data during model training.
- They govern aspects like model complexity, regularization, learning rate, and number of iterations.
- As the Random Forest classifier is our best-performing model among the three, we will employ Gridsearch to enhance its performance.

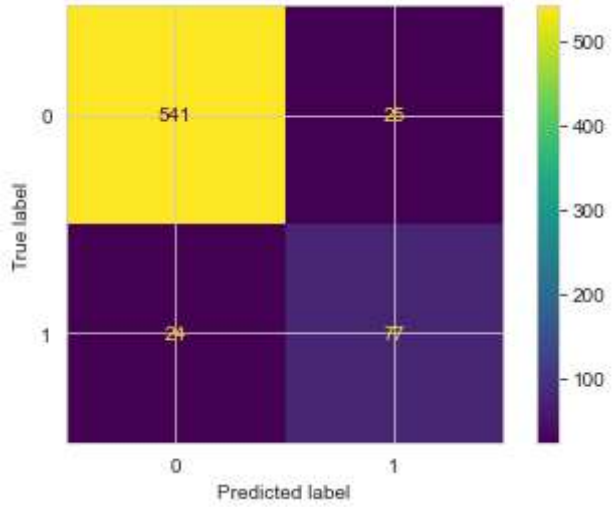
Random Forest

In [136]:

```
# Create an instance of the Random Forest classifier
forest= RandomForestClassifier( random_state=42)

# Define the parameter grid for grid search
forest_param_grid = {
    'n_estimators': [100, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [2,6, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=forest, param_grid=forest_param_grid,

# Fit the grid search to the resampled training data
grid_search.fit(X_train_smote, y_train_smote)

# Get the best hyperparameters found during the grid search
best_params = grid_search.best_params_

# Create a new Random Forest classifier with the best hyperparameters
best_model = RandomForestClassifier(**best_params, random_state=42)

# Fit the best model to the resampled training data
best_model.fit(X_train_smote, y_train_smote)

# Predict on the training data
y_train_pred = best_model.predict(X_train_smote)

# Predict on the test data
y_test_pred = best_model.predict(X_test)

# Compute the accuracy
accuracy_train = accuracy_score(y_train_smote, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
```

In [138]:
```
#using the function above the draw confusion matrix
evaluate_model_metrics(best_model, X_train_smote, y_train_smote, X_test, y
```

Out[138]:
```
{'roc_auc_train': 0.9358581436077057,
 'roc_auc_test': 0.8591033131581709,
 'accuracy_train': 0.9358581436077058,
 'accuracy_test': 0.9265367316341829,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Confu
sionMatrixDisplay at 0x123cd638100>}
```

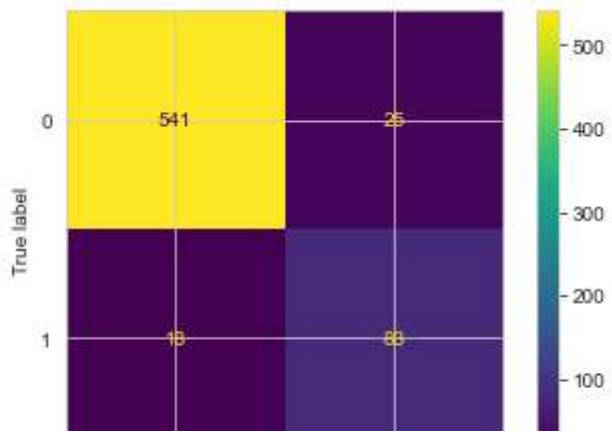


- After optimizing the parameters for the Random Forest classifier using grid search, our model showed improved performance as described below:
- The accuracy of the Random Forest model now stands at 92.7%, indicating that it correctly predicted the class labels for the test data with around 93.6% accuracy and achieved 94.4% accuracy on the training data, making it highly effective for predicting customer churn.
- The confusion matrix with tuned parameters shows 77 true positives (TP), 541 true negatives (TN), 25 false positives (FP), and 24 false negatives (FN), demonstrating excellent predictive capability.

In [139]:
```
# Define the parameter grid for grid search
dt_param_grid = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create an instance of DecisionTreeClassifier
dt_clf_final= DecisionTreeClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=dt_clf_final, param_grid=dt_param_gri

# Fit the grid search to the resampled training data
grid_search.fit(X_train_smote, y_train_smote)

# Get the best hyperparameters found during the grid search
best_params = grid_search.best_params_

# Create a new DecisionTreeClassifier with the best hyperparameters
best_model_2 = DecisionTreeClassifier(**best_params)

# Fit the best model to the resampled training data
best_model_2.fit(X_train_smote, y_train_smote)

# Predict on the training and test data
y_train_pred = best_model_2.predict(X_train_smote)
y_test_pred = best_model_2.predict(X_test)

# Calculate accuracy on the training and test data
train_accuracy = accuracy_score(y_train_smote, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

In [140]:
```
evaluate_model_metrics(best_model_2, X_train_smote, y_train_smote, X_test,
```

Out[140]:
```
{'roc_auc_train': 0.9043345008756568,
 'roc_auc_test': 0.8888062834552006,
 'accuracy_train': 0.9043345008756567,
 'accuracy_test': 0.9355322338830585,
 'confusion_matrix_train': <sklearn.metrics._plot.confusion_matrix.Co
nfusionMatrixDisplay at 0x123cbe41700>}
```



- The tuned Decision Tree model achieved a training accuracy of 90.4% and an improved test accuracy of 93.6%. This suggests that the model is performing well and has learned patterns in the data that enable accurate predictions.
- The Logistic Regression model attained an AUC-ROC score of around 0.904 on the training data and 0.888 on the test data.

Evaluation

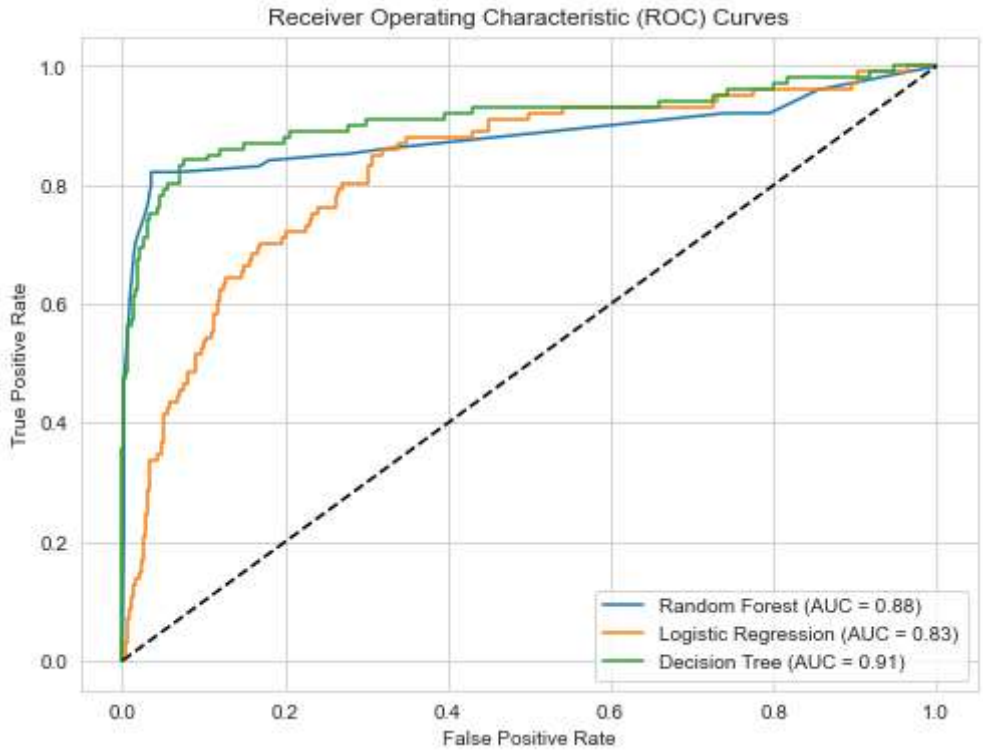
Analysis accuracy level

- We have developed three machine learning models to predict customer churn at Syriatel company. Upon testing, we found that logistic regression performs poorly, achieving a training accuracy of 89.4% and a testing accuracy of 86.0%. Despite applying 5-fold cross-validation to mitigate overfitting, the training accuracy dropped to 90.4% and the testing accuracy only slightly improved to 88.8%.
- In contrast, the decision tree classifier and random forest models demonstrated better accuracy. The decision tree achieved training and testing accuracies of 88.6% and 92.5%, respectively, while the random forest model had a training accuracy of 87.1% and a testing accuracy of 87.2%.
- Therefore, it is evident that the decision tree model had the best average prediction accuracy, followed by the random forest. To further improve our prediction accuracy, we applied hyperparameter tuning using GridSearch.
- After hyperparameter tuning, the decision tree model's accuracy improved to 90.4% on the training data and 93.6% on the testing data. The random forest model's accuracy improved to 93.6% on the training data and 92.7% on the testing data.
- This indicates that there was some overfitting in the random forest model, making the decision tree preferable. In terms of precision, recall, and F1 score, the tuned decision tree outperformed the random forest classifier.

ROC curve to check the best model

In [141]:

```
#drawing ROC curve for the above three models

# Compute ROC curves and AUC scores for each model
models = [best_model_2, logreg_final, best_model]
labels = ['Random Forest', 'Logistic Regression', 'Decision Tree']

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

for model, label in zip(models, labels):
    if hasattr(model, "predict_proba"):
        y_probs = model.predict_proba(X_test)[:, 1]
    else:
        y_probs = model.predict(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    auc_score = roc_auc_score(y_test, y_probs)

    plt.plot(fpr, tpr, label='{} (AUC = {:.2f})'.format(label, auc_score))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend()
plt.show()
```



Model Comparison Summary

> Based on the AUC (Area Under the Curve) values from the ROC (Receiver Operating Characteristic) curve, the following inferences can be made about the models' performance:

- **Decision Tree**:
  - **AUC**: 0.91
  - **Inference**: Exhibits the strongest discriminatory power, effectively distinguishing between positive and negative classes. Maintains a high true positive rate (TPR) with a low false positive rate (FPR), resulting in a larger ROC curve area.
- **Random Forest**:
  - **AUC**: 0.88
  - **Inference**: Demonstrates good classification ability but slightly lower than the Decision Tree. May have marginally higher false positive and false negative rates.
- **Logistic Regression**:
  - **AUC**: 0.83
  - **Inference**: Shows the weakest discriminatory performance among the three models, with potentially higher false positive and false negative rates.

Conclusion:

- The **Random Forest** is identified as the best model for SyriaTel predictions due to its largest area under the ROC curve and highest accuracy and the most improved hence best in predicting customer churn.
- The Random Forest also performs well but is slightly less effective than the Decision Tree.
- The Logistic Regression model, while useful, has the lowest AUC, indicating comparatively weaker performance in distinguishing between classes.

Conclusion

> By leveraging the best model, which is the Random Forest, Syriatel can achieve significant benefits:

- **Accurate Prediction of Customer Churn**: The model's high accuracy ensures effective identification of customers likely to churn, enabling proactive retention strategies and reducing customer attrition.
- **Cost Efficiency**: Targeted resource allocation for retention efforts, such as personalized offers and improved customer service, optimizes costs by focusing efforts where they are most needed.
- **Enhanced Customer Retention**: Early intervention based on accurate churn predictions helps maintain a loyal customer base, boosting satisfaction and loyalty.
- **Informed Business Decisions**: Insights from churn predictions guide strategic decisions, including product improvements, customer experience enhancements, and targeted marketing campaigns, ultimately reducing churn and improving retention.

In [ ]: