Mary Mwangi

Phase 3 Project

Data Science

Part Time

SYRIATEL CUSTOMER CHURN

1. INTRODUCTION

1.1 Business Understanding

SyriaTel, a leading telecommunications company, is grappling with the challenge of customer churn. Customer churn, is a situation where subscribers discontinue services, posing a significant financial threat to SyriaTel. As the telecom industry evolves, retaining customers becomes paramount for sustaining revenue and growth. The objective of this project is to build a classifier that can identify customers at risk of churn, enabling SyriaTel to implement targeted retention strategies.

1.2 Research Questions

a. What are the key features that significantly contribute to predicting customer churn?

b. Which machine learning algorithm demonstrates the highest performance in predicting customer churn for SyriaTel?

c. What actionable recommendations can be derived from the model to reduce customer churn and enhance customer retention?

1.3 Objectives

1.3.1 Main Objective

To develop a predictive model that effectively identifies customers at risk of churning for SyriaTel, ultimately enabling the implementation of targeted retention strategies to reduce overall customer churn

1.3.2 Specific Objectives

a. Identify the primary factors influencing customer churn in the SyriaTel dataset.

b. Build a robust machine learning model for binary classification to predict customer churn.

c. Extract meaningful insights from the model to guide SyriaTel in implementing effective retention strategies.

2. EXPLORATORY DATA ANALYSIS

2.1 Import necessary libraries

In [1]:
```
# Data manipulation
import pandas as pd
import numpy as np

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.colors as colors
import plotly.graph_objs as go
from plotly.offline import iplot
from plotly.subplots import make_subplots

# Modeling
from sklearn.model_selection import train_test_split,cross_val_score,Gric
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score,f1_score,recall_score,precisic
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from scipy import stats

# Feature Selection, Feature Importance
from sklearn.inspection import permutation_importance
from sklearn.feature_selection import RFE
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection

# Algorithms for supervised learning methods
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Filtering future warnings
import warnings
warnings.filterwarnings('ignore')
```

2.2 Load the Dataset

In [2]:
```python
df = pd.read_csv("customerchurndata.csv")
df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85

3333 rows × 21 columns

2.3 Data Source and Description

In [3]:
```python
#check the column headers
df.columns
```

Out[3]:
```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail message
s',
       '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 charg
e',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

1.State: Customer's location in the United States.

2.Account Length: Duration of customer subscription in days.

3.Area Code: Three-digit phone number area code.

4.Phone Number: Unique identifier for the customer's phone.

5.International Plan: Whether the customer has an international calling plan (yes/no).

6.Voice Mail Plan: Whether the customer has a voicemail plan (yes/no).

7.Number Vmail Messages: Number of voicemail messages.

8.Total Day Minutes: Total minutes of daytime usage.

9.Total Day Calls: Total number of calls during the day.

10.Total Day Charge: Total charge for daytime usage.

11.Total Eve Minutes: Total minutes of evening usage.

12.Total Eve Calls: Total number of calls during the evening.

13.Total Eve Charge: Total charge for evening usage.

14.Total Night Minutes: Total minutes of nighttime usage.

15.Total Night Calls: Total number of calls during the night.

16.Total Night Charge: Total charge for nighttime usage.

17.Total Intl Minutes: Total minutes of international usage.

18.Total Intl Calls: Total number of international calls.

19.Total Intl Charge: Total charge for international usage.

20.Customer Service Calls: Number of calls made to customer service.

21.Churn: Customer churn status

2.4 Data Understanding

In [4]:
```
#output the first five rows
df.head()
```

Out[4]:

	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 [5]:
```
#output the last five rows
df.tail()
```

Out[5]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85

5 rows × 21 columns

In [6]:
```
#get the size of the dataframe
df.shape
```

Out[6]: (3333, 21)

In [7]:
```
#print the column header and the data types stored in each column
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
```

-Numeric Features:

account length

area code

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 Features:

state

phone number

international plan

voicemail plan

In [8]:
```
#check the number of unique values in each column
df.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
```

In [9]:
```
#check the basic statistics details
df.describe()
```

Out[9]:

	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	333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	20
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	5
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	16
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	20
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	23
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	36

2.5 Data Cleaning

2.5.1 Checking for Missing values

In [10]:
```
df.isna().sum()
```

Out[10]:
```
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 [11]:
```
'''It appears that they are no missing data, in any of the columns in our
```

Out[11]:
```
'It appears that they are no missing data, in any of the columns in our dataset'
```

2.5.2 Checking for duplicates

In [12]:
```
df.loc[df.duplicated(keep=False)]
```

Out[12]:

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 rows × 21 columns

In [13]:
```
'''We have no duplicate values in our dataset'''
```

Out[13]:
```
'We have no duplicate values in our dataset'
```

2.5.3 Drop Irrelevant Column

In [14]:
```
#checking for the unique values of the phone number column
unique_values_df = df[['phone number']].drop_duplicates()

# Display the DataFrame with unique values
print(unique_values_df)
```

```
     phone number
0        382-4657
1        371-7191
2        358-1921
3        375-9999
4        330-6626
...           ...
3328     414-4276
3329     370-3271
3330     328-8230
3331     364-6381
3332     400-4344

[3333 rows x 1 columns]
```

In [15]:
```
"""It would be necesssary to drop the phone number column because it appe
identifier for each customer and is unlikely to contribute to predicting
```

Out[15]:
```
'It would be necesssary to drop the phone number column because it appe
ars to be a unique\nidentifier for each customer and is unlikely to con
tribute to predicting churn'
```

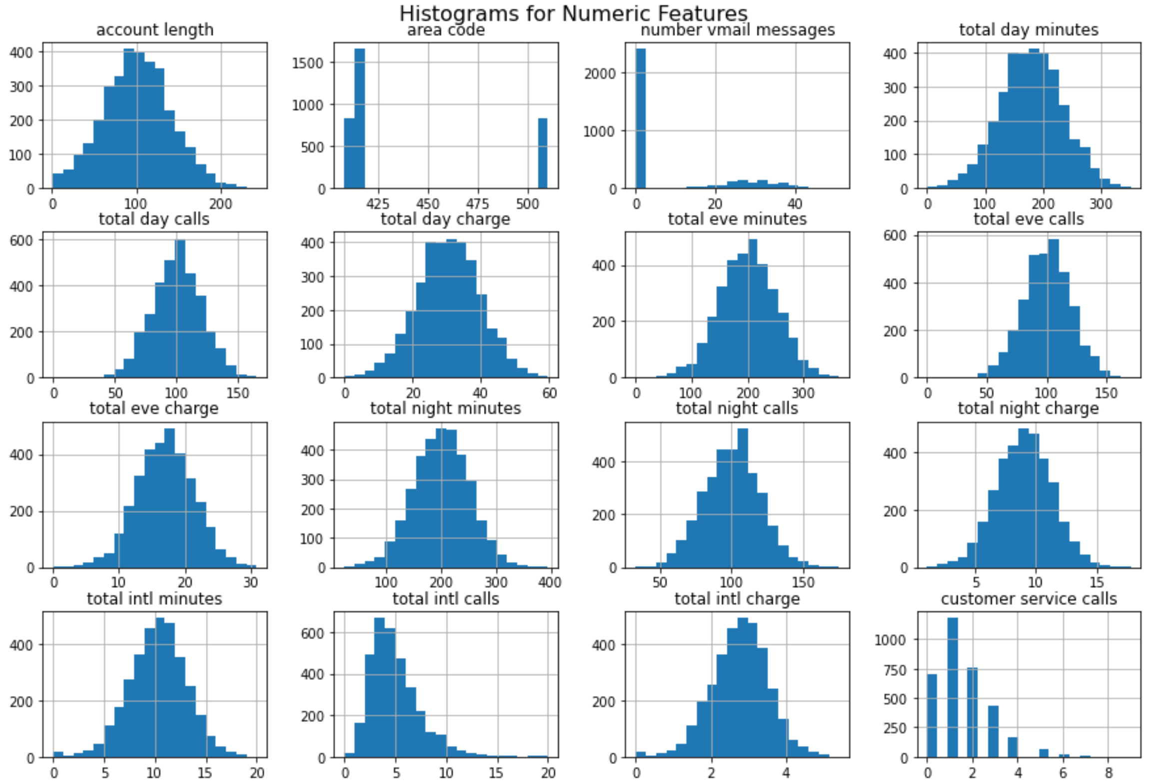
In [16]:
```
#drop the phone number column
df.drop(["phone number"], axis=1, inplace=True)
```

In [17]:
```
#check the remaining columns
df.columns
```

Out[17]:
```
Index(['state', 'account length', 'area code', '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',
       'total night calls', 'total night charge', 'total intl minutes',
       'total intl calls', 'total intl charge', 'customer service call
s',
       'churn'],
      dtype='object')
```

2.6 Data Visualization

In [18]:
```python
#Visualization of the numeric features
numeric_features = df.select_dtypes(include=['float64', 'int64'])

# Plotting histograms for numeric features
numeric_features.hist(figsize=(15, 10), bins=20)
plt.suptitle('Histograms for Numeric Features', y=0.92, fontsize=16)
plt.show()
```



apart from area code, number vmail messages and customer service calls, all the other features in the distribution have a normal distribution

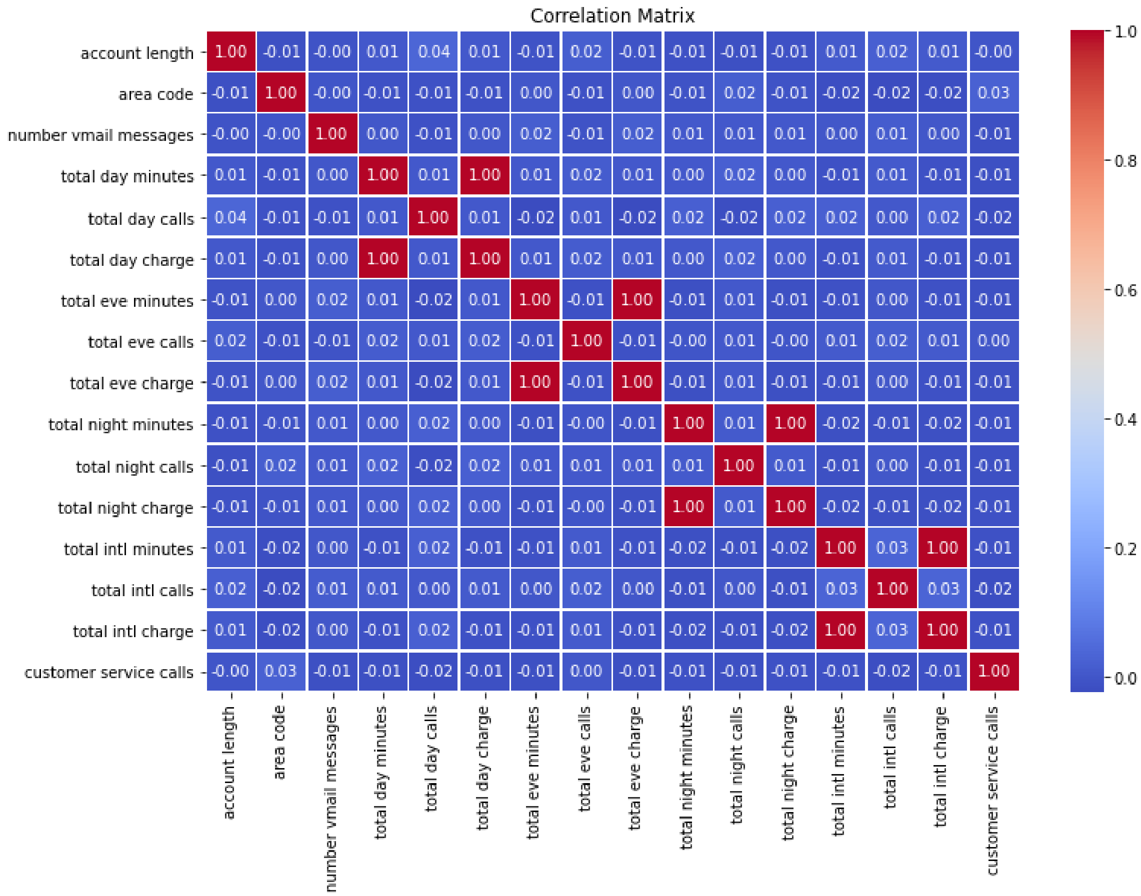
In [19]:
```python
#box plots showing the distribution of churn across different area codes
plt.figure(figsize=(14, 5))
sns.boxplot(data=df, x='area code', y='customer service calls', hue='chur
plt.legend(loc='upper right');
plt.title('Box Plot of Customer Service Calls across Area Codes and Churr
plt.show()
```

This shows that customers who are likely to have stopped doing business with SyriaTel are within the area code 415 and 510

2.6.1 Multivariate Analysis

using a heatmap to show the relationship between different numeric variables

In [20]:
```
#computing and visualizing a correlation matrix
numeric_features = df.select_dtypes(include=['float64', 'int64'])

# Compute correlation matrix
correlation_matrix = numeric_features.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", l
plt.title('Correlation Matrix')
plt.show()
```



In [21]:
```
"""
They is a low correlation between most of the features
However,a perfect positive correlation exists between:

-total day charge and total day minutes,

-total evening charge and total evening minutes,

-total night charge and total night minutes,

-total intl charge and total intl minutes
 """
```

Out[21]:
```
'\nThey is a low correlation between most of the features\nHowever,a pe
rfect positive correlation exists between:\n\n-total day charge and tot
al day minutes,\n\n-total evening charge and total evening minutes,\n\n
-total night charge and total night minutes,\n\n-total intl charge and
total intl minutes\n '
```

The features above have a correlation of 1 (perfect positive correlation), it indicates that the two features are linearly dependent, and one can be expressed as a linear combination of the other. It means that the information contained in one feature is redundant because it can be completely predicted from the other. Having highly correlated features can affect our machine learning models, Hence its necessary to drop highly-correlated features

Dropping highly correlated features

In [22]:
```
#dropping features that have a correlation of 0.9 and above
print("The original dataframe has {} columns.".format(df.shape[1]))
# Calculate the correlation matrix and take the absolute value
corr_matrix = df.corr().abs()

# Create a True/False mask and apply it
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask)

# List column names of highly correlated features (r > 0.90)
to_drop = [c for c in tri_df.columns if any(tri_df[c] >  0.90)]

df = df.drop(to_drop, axis=1) # Drop the features
print("The reduced dataframe has {} columns.".format(df.shape[1]))
```

```
The original dataframe has 20 columns.
The reduced dataframe has 16 columns.
```

2.6.2 Univariate Analysis

Churn is the target variable for my analysis

transform churn features into 1s and 0s

In [23]:
```
df['churn'] = df['churn'].map({True: 1, False: 0}).astype('int')
df.head(20)
```

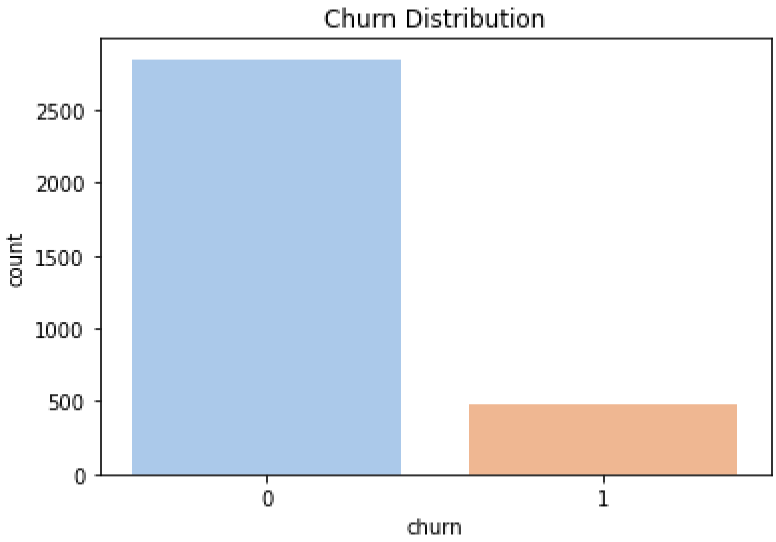
Out[23]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls
0	KS	128	415	no	yes	25	110	45.07	99	16.78	91
1	OH	107	415	no	yes	26	123	27.47	103	16.62	103
2	NJ	137	415	no	no	0	114	41.38	110	10.30	104
3	OH	84	408	yes	no	0	71	50.90	88	5.26	89
4	OK	75	415	yes	no	0	113	28.34	122	12.61	121
5	AL	118	510	yes	no	0	98	37.98	101	18.75	118
6	MA	121	510	no	yes	24	88	37.09	108	29.62	118
7	MO	147	415	yes	no	0	79	26.69	94	8.76	96
8	LA	117	408	no	no	0	97	31.37	80	29.89	90
9	WV	141	415	yes	yes	37	84	43.96	111	18.87	97
10	IN	65	415	no	no	0	137	21.95	83	19.42	111
11	RI	74	415	no	no	0	127	31.91	148	13.89	94
12	IA	168	408	no	no	0	96	21.90	71	8.92	128
13	MT	95	510	no	no	0	88	26.62	75	21.05	115
14	IA	62	415	no	no	0	70	20.52	76	26.11	99
15	NY	161	415	no	no	0	67	56.59	97	27.01	128
16	ID	85	408	no	yes	27	139	33.39	90	23.88	75
17	VT	93	510	no	no	0	114	32.42	111	18.55	121
18	VA	76	510	no	yes	33	66	32.25	65	18.09	108
19	TX	73	415	no	no	0	90	38.15	88	13.56	74

In [24]:
```
#check the unique values for the target variable
df['churn'].value_counts()
```

Out[24]:
```
0    2850
1     483
Name: churn, dtype: int64
```

In [25]:
```
sns.countplot(x='churn', data=df, palette='pastel')
plt.title('Churn Distribution')
plt.show()
```



In [26]:
```
"""With 0 representing customer who have not churned, meaning that they h
we see that around (2850 customers) are still active and are still doing
customers who have churned, around(483 customers) have stopped using syri
```

Out[26]:
```
'With 0 representing customer who have not churned, meaning that they h
ave retained or they are still active, \nwe see that around (2850 custo
mers) are still active and are still doing business with SyriaTel. whil
e 1 represents \ncustomers who have churned, around(483 customers) have
stopped using syrialTel services'
```

Interactive graphs displaying the distribution of each feature for customers with churn and those without churn

churn is represented by blue No churn is represented by orange

In [27]:

```python
churn = df[df["churn"] == 1]
no_churn = df[df["churn"] == 0]
colors = ['rgb(31, 119, 180)', 'rgb(255, 127, 14)']

def create_churn_trace(col, visible=False):
    return go.Histogram(
        x=churn[col],
        name='Churn',
        marker=dict(color=colors[0]),
        visible=visible
    )

def create_no_churn_trace(col, visible=False):
    return go.Histogram(
        x=no_churn[col],
        name='No Churn',
        marker=dict(color=colors[1]),
        visible=visible
    )

features_not_for_hist = ["state", "churn"]
features_for_hist = [x for x in df.columns if x not in features_not_for_h

n_features = len(features_for_hist)
rows = int(n_features / 2) + n_features % 2
cols = 2
fig = make_subplots(rows=rows, cols=cols, subplot_titles=features_for_his

for i, feature in enumerate(features_for_hist):
    row = (i // cols) + 1
    col = (i % cols) + 1

    fig.add_trace(create_churn_trace(feature, visible=True), row=row, col
    fig.add_trace(create_no_churn_trace(feature, visible=True), row=row,

    fig.update_xaxes(title_text=feature, row=row, col=col)
    fig.update_yaxes(title_text="# Samples", row=row, col=col)

    fig.update_layout(
        showlegend=False,
        height=rows * 300,
        width=900,
        title="Feature Distribution: Churn vs No Churn"
    )

fig.show()
```

Feature Distribution: Churn vs No Churn

3. Feature Engineering

3.1 One-Hot Encoding

It is necessary to convert categorical variables into numerical format, the categorical variables are International plan,voice mail plan and i will also include the area code since it has three categories

In [28]:
```
#One Hot Encoding
df = pd.get_dummies(df,columns = ['international plan','voice mail plan',
# Display the updated dataframe
df.head()
```

Out[28]:

	state	account length	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	cus s
0	KS	128	25	110	45.07	99	16.78	91	11.01	3	2.70	
1	OH	107	26	123	27.47	103	16.62	103	11.45	3	3.70	
2	NJ	137	0	114	41.38	110	10.30	104	7.32	5	3.29	
3	OH	84	0	71	50.90	88	5.26	89	8.86	7	1.78	
4	OK	75	0	113	28.34	122	12.61	121	8.41	3	2.73	

3.2 Label Encoding

It is necessary to convert the "state" column using the LabelEncoder technique. This assigns a unique numerical label to each state.

In [29]:
```
le = LabelEncoder()
le.fit(df['state'])
df['state'] = le.transform(df['state'])
df.head()
```

Out[29]:

	state	account length	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	cus s
0	16	128	25	110	45.07	99	16.78	91	11.01	3	2.70	
1	35	107	26	123	27.47	103	16.62	103	11.45	3	3.70	
2	31	137	0	114	41.38	110	10.30	104	7.32	5	3.29	
3	35	84	0	71	50.90	88	5.26	89	8.86	7	1.78	
4	36	75	0	113	28.34	122	12.61	121	8.41	3	2.73	

4. Data preparation for Modelling

4.1 Defining X and Y variables

With y being the target variable and X the independent variables

In [30]:
```
y = df['churn']
X = df.drop('churn', axis = 1)
```

4.2 Train and Test Split

In [31]:
```
"""Before performing the modelling, its important to split the data into
leakage and also prevents overfitting, hence enhances the building of moc
that generalize well in real-world applications"""
```

Out[31]:
```
'Before performing the modelling, its important to split the data into
training and testing sets, this avoids data\nleakage and also prevents
overfitting, hence enhances the building of models\nthat generalize wel
l in real-world applications'
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42

#Checking the length
len(X_train), len(X_test), len(y_train), len(y_test)
```

Out[32]:
```
(2499, 834, 2499, 834)
```

4.3 Concatenate the Processed data

combine the processed X and y variables into training and testing sets

In [33]:
```
X_processed = pd.concat([X_train, X_test], axis=0)
y_processed = pd.concat([y_train, y_test], axis=0)

# Displaying the shapes of the resulting concatenated sets
print("X_processed shape:", X_processed.shape)
print("y_processed shape:", y_processed.shape)
```

```
X_processed shape: (3333, 19)
y_processed shape: (3333,)
```

4.4 Dealing with class imbalance using SMOTE

Class imbalance occurs in a dataset when the distribution of instances across different classes is not roughly equal.Where one class (in this case, the 'churn' class) is significantly underrepresented compared to the other class.SMOTE would involve addressing the class imbalance in the target variable 'churn.' Since the goal is to predict whether a customer will churn or not, the dataset may have an imbalance between customers who churn ('1') and those who don't ('0'). SMOTE can be applied to balance this distribution, ensuring that the model is not biased toward the majority class.

In [34]:
```
#checking distribution before applying SMOTE
df.churn.value_counts()
```

Out[34]:
```
0    2850
1     483
Name: churn, dtype: int64
```

In [35]:
```
# Apply SMOTE to the training data
sm = SMOTE(k_neighbors=5, random_state=123)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)
print('Before OverSampling, the shape of X_train: {}'.format(X_train.shap
print('Before OverSampling, the shape of y_train: {}'.format(y_train.shap
print('After OverSampling, the shape of X_train_resampled: {}'.format(X_t
print('After OverSampling, the shape of y_train_resampled: {}'.format(y_t
```

```
Before OverSampling, the shape of X_train: (2499, 19)
Before OverSampling, the shape of y_train: (2499,)
After OverSampling, the shape of X_train_resampled: (4282, 19)
After OverSampling, the shape of y_train_resampled: (4282,)
```

In [36]:
```
#checking for class imbalance again
y_train_resampled.value_counts()
```

Out[36]:
```
1    2141
0    2141
Name: churn, dtype: int64
```

In [37]:
```
'''I can now say my class is balanced since [1] is 2141 and [0] is 2141''
```

Out[37]:
```
'I can now say my class is balanced since [1] is 2141 and [0] is 2141'
```

5. Modelling

Due to the business problem the focus will be on the metrics recall and precision and accuracy. This is because the three will help SyriaTel strike a balance between identifying a large portion of potential churners and ensuring that the identification is accurate. This approach will enable the implementation of targeted retention strategies effectively, addressing the financial threat posed by customer churn.

MODEL 5.1 Baseline Logistic Regression

Logistic Regression is used for solving classification problems, when the target variable is categorical.Using logistic regression as a starting point inorder for me to compare the effectiveness of more complex techniques against this baseline approach

a) object creation and fitting the model

In [38]:
```
lr_model= LogisticRegression()
lr_model.fit(X_train_resampled,y_train_resampled)
y_logistic_prediction = lr_model.predict(X_test)
```

b) Classification Report

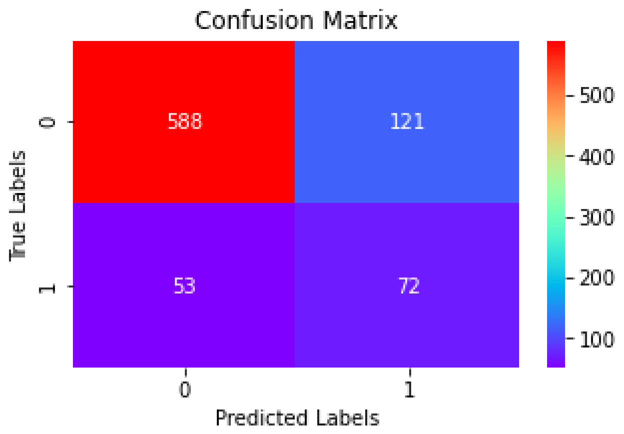
In [39]:
```
#checking for metrics such as precision, recall, F1-score amd support
print(classification_report(y_test, y_logistic_prediction, target_names=[
```

```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87       709
           1       0.37      0.58      0.45       125

    accuracy                           0.79       834
   macro avg       0.65      0.70      0.66       834
weighted avg       0.84      0.79      0.81       834
```

c) Confusion Matrix visualization

In [40]:
```
print(" LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS")
print('Accuracy: ', round(accuracy_score(y_test,  y_logistic_prediction),
print('F1 Score: ', round(f1_score(y_test,  y_logistic_prediction), 5))
print('Recall: ', round(recall_score(y_test,  y_logistic_prediction), 5))
print('Precision: ', round(precision_score(y_test,  y_logistic_predictior

cm_lr = confusion_matrix(y_test,  y_logistic_prediction)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_lr, annot=True, cmap='rainbow', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
 LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS
Accuracy:  0.79137
F1 Score:  0.45283
Recall:  0.576
Precision:  0.37306
```



In [41]:
```
"""The accuracy of 0.79137 suggests that the model correctly predicted th
The F1 score of 0.45283 represents a balanced evaluation of the model's a
The recall score of 0.576 indicates the model's ability to correctly ider
while the precision score of 0.37306 reflects its ability to minimize fal
```

Out[41]:
```
"The accuracy of 0.79137 suggests that the model correctly predicted th
e churn status of approximately 79.14% of the customers.\nThe F1 score
of 0.45283 represents a balanced evaluation of the model's accuracy in
predicting both churn and non-churn customers.\nThe recall score of 0.5
76 indicates the model's ability to correctly identify churn cases,\nwh
ile the precision score of 0.37306 reflects its ability to minimize fal
se positive predictions."
```

-Recall (True Positive Rate): High recall is crucial because you want to identify as many customers at risk of churn as possible. Missing a customer who is likely to churn (false negatives) could result in the loss of revenue and harm to customer satisfaction.

-Precision: While identifying as many potential churners as possible is important (high recall), it's equally crucial to ensure that the identified cases are accurate. High precision helps avoid unnecessary retention efforts on customers who are not actually at risk of churning (false positives)

MODEL 5.2 Decision Tree Classifier

Decision tree classifier is used for both classification and regression, but mostly preferred for classification problems. It is a tree-structured classifier where internal nodes represent the features of the datasets, branches represent the decision rules and each leaf node represent the outcome.

It is important to use the decision tree classifier to gain insights into the important features and help in the decision making process

a) Object creation and fitting the model

In [42]:
```
dt_model= DecisionTreeClassifier()
dt_model.fit(X_train_resampled,y_train_resampled)
y_dt_prediction= dt_model.predict(X_test)
```

b) Classification Report

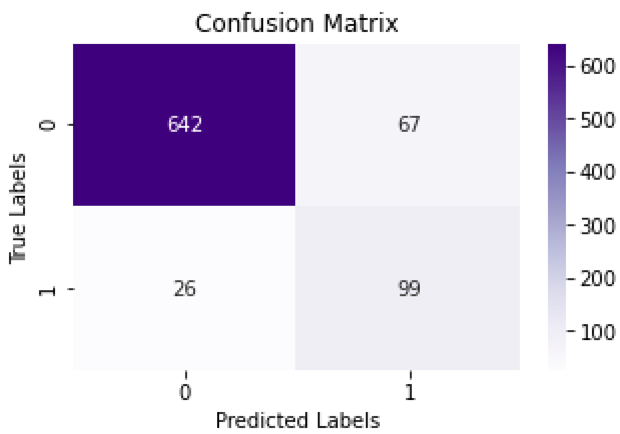
In [43]:
```
#Displaying metrics which are precision, recall, F1-score and support
print(classification_report(y_test, y_dt_prediction, target_names=['0', '
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93       709
           1       0.60      0.79      0.68       125

    accuracy                           0.89       834
   macro avg       0.78      0.85      0.81       834
weighted avg       0.91      0.89      0.89       834
```

c) Confusion Matrix visualization

In [44]:
```
print("DECISION TREE CLASSIFIER MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test, y_dt_prediction),5))
print('F1 score: ',round(f1_score(y_test, y_dt_prediction),5))
print('Recall: ',round(recall_score(y_test, y_dt_prediction),5))
print('Precision: ',round(precision_score(y_test, y_dt_prediction),5))
cm_dt = confusion_matrix(y_test, y_dt_prediction)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_dt, annot=True, cmap='Purples', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
DECISION TREE CLASSIFIER MODEL RESULTS
Accuracy:  0.88849
F1 score:  0.68041
Recall:  0.792
Precision:  0.59639
```



In [45]:
```
"""The accuracy of 0.89688 indicates that the decision tree model correct
89.69% of the customers.The F1 score of 0.70345 reflects a good balance b
overall effectiveness.A recall score of 0.816 suggests a high ability to
Precision at 0.61818 shows the model's ability to minimize false positive
```

Out[45]:
```
"The accuracy of 0.89688 indicates that the decision tree model correct
ly predicted the churn status for approximately \n89.69% of the custome
rs.The F1 score of 0.70345 reflects a good balance between precision an
d recall, capturing the model's \noverall effectiveness.A recall score
of 0.816 suggests a high ability to correctly identify customers who ar
e likely to churn.\nPrecision at 0.61818 shows the model's ability to m
inimize false positive predictions while identifying churn cases."
```

-Decision Tree model has a higher recall for churned customers (Class 1) compared to the baseline Logistic Regression model. This indicates that the Decision Tree is better at capturing a larger proportion of customers who are actually at risk of churning.

-It also exhibits a higher precision and accuracy for churned customers compared to the baseline Logistic Regression model. This suggests that when the Decision Tree predicts a customer to churn, it is more likely to be accurate compared to the baseline model.

MODEL 5.3 K-Nearest Neighbours

KNN algorithm is the simplest Machine learning algorithm, It assumes the similarities between the new case/data and available cases and puts the new into the category that is most similar to the available categories

KNN can be utilized to classify customers as active or inactive based on similarities in their feature values

a) Object creation and fitting the model

In [46]:
```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_resampled,y_train_resampled)
y_knn_prediction = knn_model.predict(X_test)
```

b) Classification Report

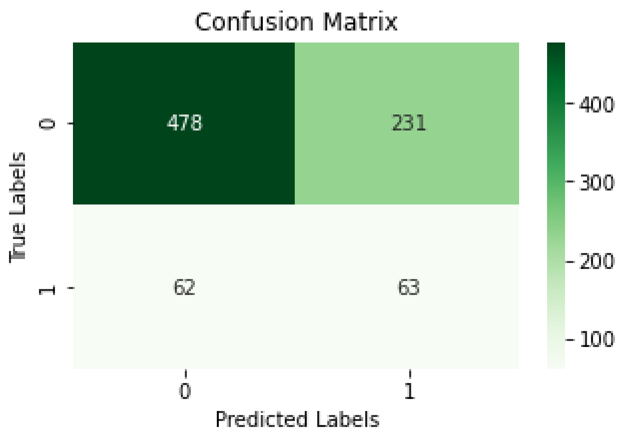
In [47]:
```
#Displaying metrics which are precision, recall, F1-score and support
print(classification_report(y_test, y_knn_prediction, target_names=['0',
```

```
              precision    recall  f1-score   support

           0       0.89      0.67      0.77       709
           1       0.21      0.50      0.30       125

    accuracy                           0.65       834
   macro avg       0.55      0.59      0.53       834
weighted avg       0.78      0.65      0.70       834
```

c) Confusion Matrix visualization

In [48]:
```
print("K-Nearest Neighbors (KNN) CLASSIFIER MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test, y_knn_prediction),5))
print('F1 score: ',round(f1_score(y_test, y_knn_prediction),5))
print('Recall: ',round(recall_score(y_test, y_knn_prediction),5))
print('Precision: ',round(precision_score(y_test, y_knn_prediction),5))
cm_dt = confusion_matrix(y_test, y_knn_prediction)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_dt, annot=True, cmap='Greens', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
K-Nearest Neighbors (KNN) CLASSIFIER MODEL RESULTS
Accuracy:  0.64868
F1 score:  0.30072
Recall:  0.504
Precision:  0.21429
```



In [49]:
```
"""The accuracy of 0.64868 indicates that the K-Nearest Neighbors model c
approximately 64.87% of the customers.The F1 score of 0.30072 suggests a
recall compared to other models.A recall score of 0.504 indicates moderat
who are likely to churn.Precision at 0.21429 shows the model's ability to
but the trade-off is lower precision."""
```

Out[49]:
```
"The accuracy of 0.64868 indicates that the K-Nearest Neighbors model c
orrectly predicted the churn status for \napproximately 64.87% of the c
ustomers.The F1 score of 0.30072 suggests a lower balance between preci
sion and \nrecall compared to other models.A recall score of 0.504 indi
cates moderate success in correctly identifying customers\nwho are like
ly to churn.Precision at 0.21429 shows the model's ability to minimize
false positive predictions,\nbut the trade-off is lower precision."
```

-Logistic Regression has a higher precision compared to K-Nearest Neighbors for identifying customers at risk of churn. This indicates that Logistic Regression is more accurate in correctly classifying customers who are predicted to churn.

-It still has a higher recall and accuracy compared to K-Nearest Neighbors for churned customers. This suggests that Logistic Regression is better at capturing a larger proportion of customers who are actually at risk of churning

-But the decision tree is still a better model since it has a higher precision, recall and accuracy compared to both the baseline logistic model and K-Nearest Neighbors

MODEL 5.4 Random Forest Classifier

The Random Forest Classifier is used for both classification and regression.It is based on the concept of ensemble learning, which is the process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. It is basically a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset

Random forest will help to gain insights into the data's patterns and relationships.

a) Object creation and fitting the model

In [50]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train_resampled,y_train_resampled)
y_rf_prediction = rf_model.predict(X_test)
```

b) Classification Report

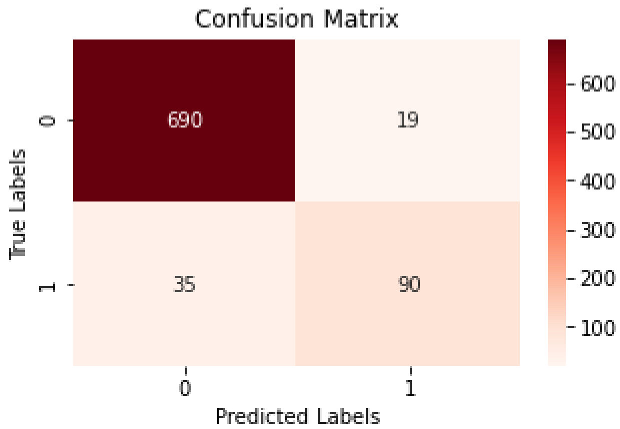
In [51]:
```
#Displaying metrics which are precision, recall, F1-score and support
print(classification_report(y_test, y_rf_prediction, target_names=['0', '
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       709
           1       0.83      0.72      0.77       125

    accuracy                           0.94       834
   macro avg       0.89      0.85      0.87       834
weighted avg       0.93      0.94      0.93       834
```

c) Confusion Matrix visualization

In [52]:
```
print("RANDOM FOREST CLASSIFIER MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test,y_rf_prediction),5))
print('F1 score: ',round(f1_score(y_test,y_rf_prediction),5))
print('Recall: ',round(recall_score(y_test,y_rf_prediction),5))
print('Precision: ',round(precision_score(y_test,y_rf_prediction),5))
cm_rf = confusion_matrix(y_test,y_rf_prediction)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Reds', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
RANDOM FOREST CLASSIFIER MODEL RESULTS
Accuracy:  0.93525
F1 score:  0.76923
Recall:  0.72
Precision:  0.82569
```



In [53]:
```
"""The accuracy of 0.93405 indicates that the random forest model correct
approximately 93.41% of the customers.The F1 score of 0.7619 reflects a g
and recall, capturing the model's overall effectiveness.A recall score of
to correctly identify customers who are likely to churn.Precision at 0.83
false positive predictions while identifying churn cases."""
```

Out[53]:
```
"The accuracy of 0.93405 indicates that the random forest model correct
ly predicted the churn status for\napproximately 93.41% of the customer
s.The F1 score of 0.7619 reflects a good balance between precision\nand
recall, capturing the model's overall effectiveness.A recall score of
0.704 suggests a high ability\nto correctly identify customers who are
likely to churn.Precision at 0.83019 shows the model's ability to minim
ize \nfalse positive predictions while identifying churn cases."
```

-Random Forest has a significantly higher precision, recall and accuracy compared to all the other models for identifying customers at risk of churn. This indicates that Random Forest is more accurate in correctly classifying customers who are predicted to churn.

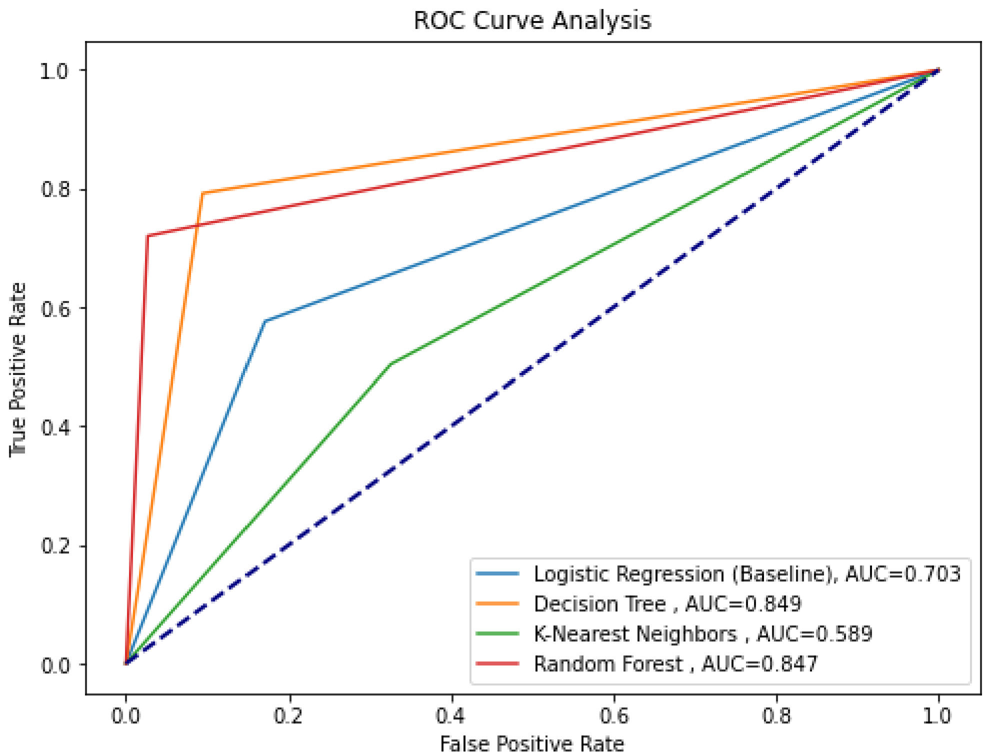
Curve Analysis using ROC

The Receiver Operating Characteristic (ROC) curve serves as a visual depiction of a binary classification model's effectiveness. It charts the True Positive Rate (Sensitivity) against the False Positive Rate (1 - Specificity) across various threshold settings.Widely employed in the assessment of different classification models, the ROC curve offers a comparative analysis of their performance. A model with an ROC curve positioned higher and closer to the top-left corner signifies superior predictive accuracy. Additionally, the calculation of the area under the ROC curve (AUC) yields a singular metric representing the overall performance of the model. A higher AUC value indicates enhanced discrimination power.

Through the scrutiny of ROC curves and the computation of AUC for diverse models, we can gauge their proficiency in accurately classifying positive and negative instances, facilitating a ranking based on their performance.

In [54]:

```python
# Plotting ROC Curves
plt.figure(figsize=(8, 6))
roc_auc_values = []

models = [('Logistic Regression (Baseline)', y_logistic_prediction),
          ('Decision Tree ', y_dt_prediction),
          ('K-Nearest Neighbors ', y_knn_prediction),
          ('Random Forest ', y_rf_prediction)]

for model_name, y_probs in models:
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    roc_auc = roc_auc_score(y_test, y_probs)
    roc_auc_values.append((model_name, roc_auc))
    plt.plot(fpr, tpr, label=f'{model_name}, AUC={roc_auc:.3f}')

# Sort models by AUC in descending order
roc_auc_values.sort(key=lambda x: x[1], reverse=True)
sorted_model_names = [model[0] for model in roc_auc_values]

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Analysis')
plt.legend()
plt.show()

print("\033[1mModels sorted by AUC in descending order:\n\033[0m")
for model_name in sorted_model_names:
    print(model_name)
```



Models sorted by AUC in descending order:

```
Decision Tree
Random Forest
Logistic Regression (Baseline)
K-Nearest Neighbors
```

6. Model Tuning

hyperparameter tuning is a crucial step in the machine learning pipeline to optimize model performance,It is crucial for refining models, optimizing performance metrics, and ensuring that the machine learning models are well-suited for the specific business problem at hand

The logistic regression model is the baseline model of my analysis, hence no model tuning is required for the baseline model. This is because, the baseline model serves as a reference point against which the improvements achieved through hyperparameter tuning in other models can be objectively assessed. This comparative perspective enables a comprehensive evaluation of the effectiveness of tuning efforts, highlighting the impact of hyperparameter adjustments on model performance.

6.1 Hyperparameter Tuning of the Decision Tree Classifier

In [55]:
```
#parameter grid for decision tree model optimization
dt_params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"],
    'max_features': ["sqrt"],
    'min_samples_split': [6, 10, 14]
}
```

In [56]:
```
#grid search for decision tree model optimization
dt_model2 = DecisionTreeClassifier()
dt_cv_model = GridSearchCV(dt_model2, dt_params, cv=3, n_jobs=-1, verbose
dt_cv_model.fit(X_train_resampled,y_train_resampled)
print("Best parameters:"+str(dt_cv_model.best_params_))
```

```
Best parameters:{'criterion': 'entropy', 'max_depth': 10, 'max_feature
s': 'sqrt', 'min_samples_leaf': 10, 'min_samples_split': 14}
```

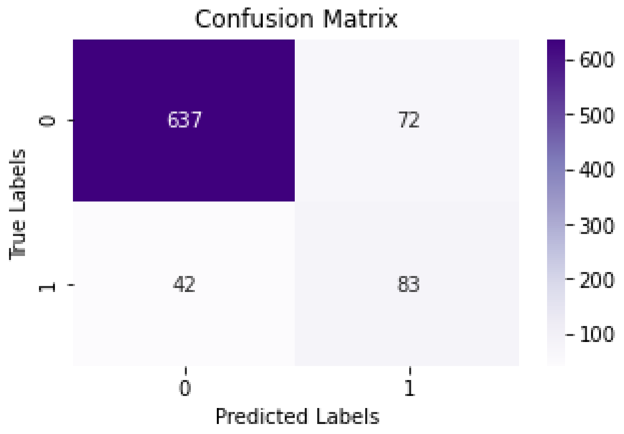
In [57]:
```
#Decision Tree Model with GridSearch cross-validation  Applied
dt_model_GridSearchCV_Applied = DecisionTreeClassifier(criterion='entropy
dt_model_GridSearchCV_Applied.fit(X_train_resampled,y_train_resampled)
y_pred_dt_GridSearchCV_Applied = dt_model_GridSearchCV_Applied.predict(X_
```

In [58]:
```
#Classification Report showing the Hyperparameter Tuned Decision Tree Moc
print(classification_report(y_test, y_pred_dt_GridSearchCV_Applied, targe
```

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       709
           1       0.54      0.66      0.59       125

    accuracy                           0.86       834
   macro avg       0.74      0.78      0.76       834
weighted avg       0.88      0.86      0.87       834
```

In [59]:
```
#Hyperparameter Tuned Decision Tree Model Results and Confusion Matrix Vi
print("HYPERPARAMETER TUNED DECISION TREE MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test, y_pred_dt_GridSearchCV_Ap
print('F1 score: ',round(f1_score(y_test, y_pred_dt_GridSearchCV_Applied)
print('Recall: ',round(recall_score(y_test, y_pred_dt_GridSearchCV_Applie
print('Precision: ',round(precision_score(y_test, y_pred_dt_GridSearchCV_
cm_rf = confusion_matrix(y_test, y_pred_dt_GridSearchCV_Applied)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Purples', fmt='g', ax=ax);
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
HYPERPARAMETER TUNED DECISION TREE MODEL RESULTS
Accuracy:  0.86331
F1 score:  0.59286
Recall:  0.664
Precision:  0.53548
```



In [60]:
```
#Model Performance Comparison: comparing the first model and the tuned mo
comparison_frame = pd.DataFrame({'Model':['Decision Tree Classifier (Defa
                                          'Decision Tree Classifier (Tune
                                 'Accuracy (Test Set)':[accuracy_score(y_
                                 'F1 Score (Test Set)':[f1_score(y_test,
                                 'Recall (Test Set)':[recall_score(y_test
                                 'Precision (Test Set)':[precision_score(

comparison_frame.style.highlight_max(color = 'pink', axis = 0)
```

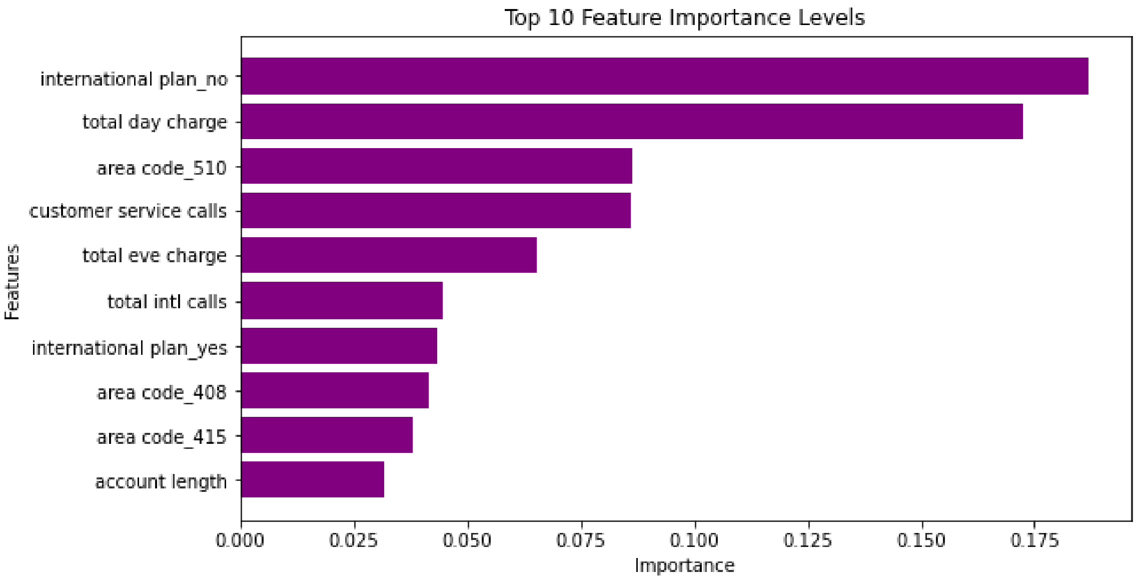
Out[60]:

	Model	Accuracy (Test Set)	F1 Score (Test Set)	Recall (Test Set)	Precision (Test Set)
0	Decision Tree Classifier (Default)	0.888489	0.680412	0.792000	0.596386
1	Decision Tree Classifier (Tuned)	0.863309	0.592857	0.664000	0.535484

In [61]:
```
"""Accuracy: The tuned Decision Tree model has a slightly lower accuracy
The model might be more conservative after hyperparameter tuning. F1 Scor
indicating a trade-off between precision and recall.Recall: Recall decrea
tuned model identifies fewer positive instances.Precision: Precision decr
indicating a decrease in the ability to correctly classify positive insta
```

Out[61]:
```
'Accuracy: The tuned Decision Tree model has a slightly lower accuracy
than the default model 84.17% vs. 89.68%.\nThe model might be more cons
ervative after hyperparameter tuning. F1 Score: The F1 score decreased
from 0.70 to 0.52, \nindicating a trade-off between precision and recal
l.Recall: Recall decreased from 0.81 to 0.58, indicating that the\ntune
d model identifies fewer positive instances.Precision: Precision decrea
sed from 0.62 to 0.48, \nindicating a decrease in the ability to correc
tly classify positive instances.'
```

In [62]:
```
#Important Feature Levels for Decision Tree Tuned Model(First 10)
importance = pd.DataFrame({"Importance": dt_model_GridSearchCV_Applied.fe
importance = importance.sort_values(by="Importance", ascending=True).tail

plt.figure(figsize=(9, 5))
plt.barh(importance.index, importance["Importance"], color="purple")
plt.title("Top 10 Feature Importance Levels")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.show()
```



We can see that the most important features in the tuned decision tree are the international plan no, area code 510 and total day charge

6.2 Hyperparameter Tuning of the K-Nearest Neighbours Classifier

In [63]:
```
#Parameter Grid for K-Nearest Neighbors (KNN) Model Optimization
knn_params = {'weights' : ['uniform', 'distance'],
              'metric' : ['manhattan', 'euclidean', 'minkowski'],
              'n_neighbors': [5, 15, 25, 35, 45, 55, 65],
              'p': [1, 2, 10]}
```

In [64]:
```
#Grid Search for K-Nearest Neighbors (KNN) Model Optimization
knn_model2 = KNeighborsClassifier()
knn_cv_model = GridSearchCV(knn_model2, knn_params, cv=3, n_jobs=-1, verb
knn_cv_model.fit(X_train_resampled, y_train_resampled)
print("Best parameters: " + str(knn_cv_model.best_params_))
```

```
Best parameters: {'metric': 'manhattan', 'n_neighbors': 5, 'p': 1, 'wei
ghts': 'distance'}
```

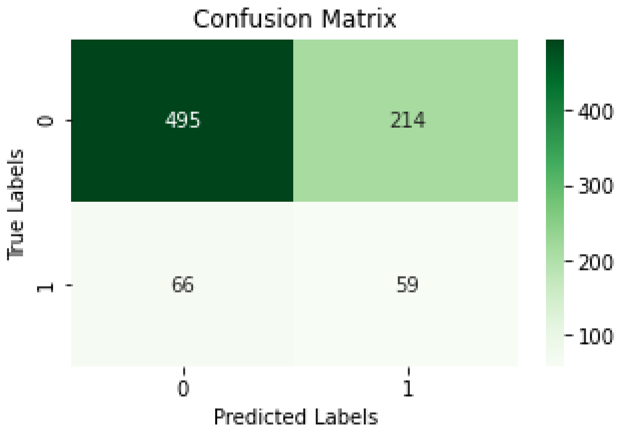
In [65]:
```
#K-Nearest Neighbors (KNN) Model with GridSearch Cross-Validation Appliec
knn_model_GridSearchCV_Applied = KNeighborsClassifier(metric='manhattan',
knn_model_GridSearchCV_Applied.fit(X_train_resampled, y_train_resampled)
y_pred_knn_GridSearchCV_Applied = knn_model_GridSearchCV_Applied.predict(
```

In [66]:
```
#Classification Report showing the Hyperparameter Tuned K-Nearest Neighbc
print(classification_report(y_test, y_pred_knn_GridSearchCV_Applied, targ
```

```
              precision    recall  f1-score   support

           0       0.88      0.70      0.78       709
           1       0.22      0.47      0.30       125

    accuracy                           0.66       834
   macro avg       0.55      0.59      0.54       834
weighted avg       0.78      0.66      0.71       834
```

In [67]:
```
#Hyperparameter Tuned K-Nearest Neighbors (KNN) Model Results and Confusi
print("HYPERPARAMETER TUNED K-Nearest Neighbors (KNN) MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test, y_pred_knn_GridSearchCV_A
print('F1 score: ',round(f1_score(y_test, y_pred_knn_GridSearchCV_Appliec
print('Recall: ',round(recall_score(y_test, y_pred_knn_GridSearchCV_Appli
print('Precision: ',round(precision_score(y_test, y_pred_knn_GridSearchCV
cm_rf = confusion_matrix(y_test, y_pred_knn_GridSearchCV_Applied)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Greens', fmt='g', ax=ax);
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
HYPERPARAMETER TUNED K-Nearest Neighbors (KNN) MODEL RESULTS
Accuracy:  0.66427
F1 score:  0.29648
Recall:  0.472
Precision:  0.21612
```



In [68]:
```
#Model Performance Comparison: Comparing the first model and the tuned mc
comparison_frame = pd.DataFrame({'Model':['knn Classifier (Default)',
                                          'knn Classifier (Tuned)'],
                                 'Accuracy (Test Set)':[accuracy_score(y_
                                 'F1 Score (Test Set)':[f1_score(y_test,
                                 'Recall (Test Set)':[recall_score(y_test
                                 'Precision (Test Set)':[precision_score(

comparison_frame.style.highlight_max(color = 'lightgreen', axis = 0)
```

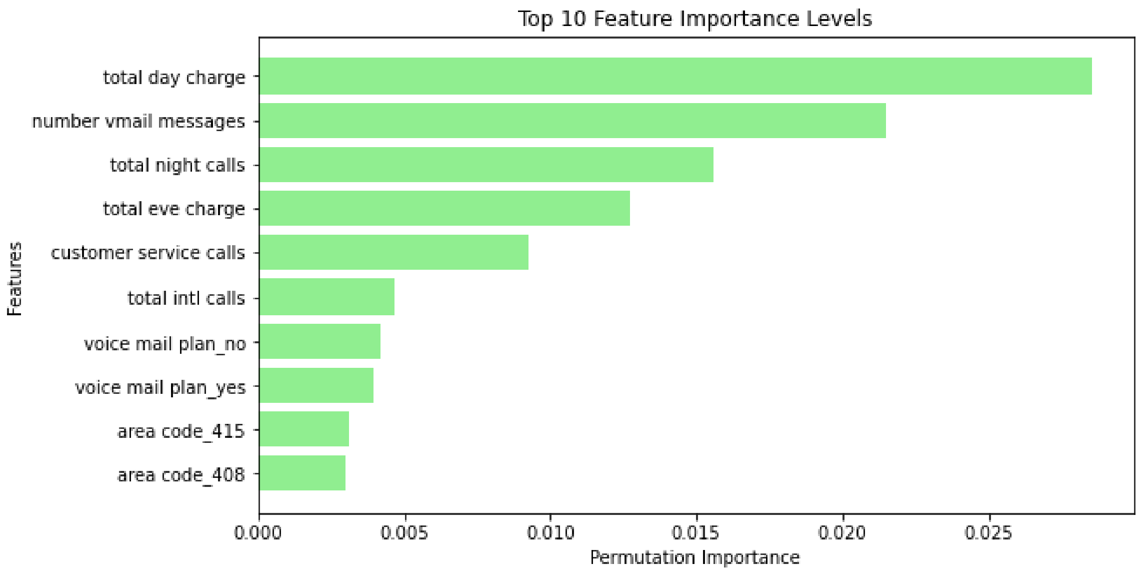
Out[68]:

	Model	Accuracy (Test Set)	F1 Score (Test Set)	Recall (Test Set)	Precision (Test Set)
0	knn Classifier (Default)	0.648681	0.300716	0.504000	0.214286
1	knn Classifier (Tuned)	0.664269	0.296482	0.472000	0.216117

In [69]:
```
"""Accuracy: The tuned KNN model improved in accuracy compared to the def
However, it's still relatively low.F1 Score: The F1 score decreased from
Recall: Recall decreased from 0.50 to 0.47, indicating that the tuned moc
Precision: Precision increased from 0.21 to 0.22, indicating a slight imp
```

Out[69]:
```
"Accuracy: The tuned KNN model improved in accuracy compared to the def
ault model 66.43% vs. 64.87%.\nHowever, it's still relatively low.F1 Sc
ore: The F1 score decreased from 0.30 to 0.29 after tuning.\nRecall: Re
call decreased from 0.50 to 0.47, indicating that the tuned model ident
ifies fewer positive instances.\nPrecision: Precision increased from 0.
21 to 0.22, indicating a slight improvement in correctly classifying po
sitive instances"
```

In [70]:
```python
#Important Feature Levels for K-Nearest Neighbour(First 10)
result_perm = permutation_importance(knn_model_GridSearchCV_Applied, X_te

importance = pd.DataFrame({"Importance": result_perm.importances_mean}, i
importance = importance.sort_values(by="Importance").tail(10)

plt.figure(figsize=(9, 5))
plt.barh(importance.index, importance["Importance"], color="lightgreen")
plt.xlabel("Permutation Importance")
plt.ylabel("Features")
plt.title("Top 10 Feature Importance Levels")
plt.show()
```



We can see that the most important features in the tuned KNN model is the total day charge, number vmail messages and total night calls

6.3 Hyperparameter Tuning of the Random Forest Classifier

In [71]:
```
#Parameter Grid for Random Forest Model Optimization
rf_params = {"max_depth": [8,15,20],
             "n_estimators":[500,1000],
             "min_samples_split":[5,10,15],
             "min_samples_leaf" : [1, 2, 4],
             "max_features": ['auto', 'sqrt'],
             "criterion":['entropy','gini']}
```

In [72]:
```
#Grid Search for Random Forest Model Optimization
rf_model2 = RandomForestClassifier()
rf_cv_model = GridSearchCV(rf_model2, rf_params, cv=3, n_jobs=-1, verbose
rf_cv_model.fit(X_train_resampled,y_train_resampled)
print("Best parameters:"+str(rf_cv_model.best_params_))
```

```
Best parameters:{'criterion': 'entropy', 'max_depth': 20, 'max_feature
s': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimator
s': 500}
```

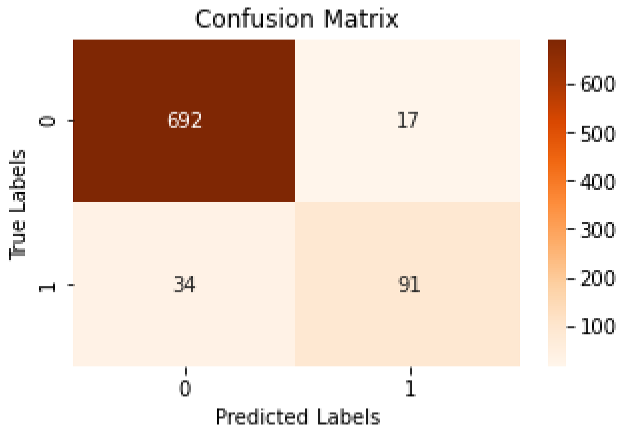
In [73]:
```
#Random Forest Model with GridSearch Cross-Validation Applied
rf_model_GridSearchCV_Applied = RandomForestClassifier(criterion='entropy
rf_model_GridSearchCV_Applied.fit(X_train_resampled,y_train_resampled)
y_pred_rf_GridSearchCV_Applied = rf_model_GridSearchCV_Applied.predict(X_
```

In [74]:
```
#Classification Report showing the Hyperparameter Tuned Random Forest Moc
print(classification_report(y_test, y_pred_rf_GridSearchCV_Applied, targe
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       709
           1       0.84      0.73      0.78       125

    accuracy                           0.94       834
   macro avg       0.90      0.85      0.87       834
weighted avg       0.94      0.94      0.94       834
```

```
In [75]: #Hyperparameter Tuned Random Forest Model Results and Confusion Matrix Vi
print("HYPERPARAMETER TUNED RANDOM FOREST MODEL RESULTS")
print('Accuracy: ',round(accuracy_score(y_test, y_pred_rf_GridSearchCV_Ap
print('F1 score: ',round(f1_score(y_test, y_pred_rf_GridSearchCV_Applied)
print('Recall: ',round(recall_score(y_test, y_pred_rf_GridSearchCV_Applie
print('Precision: ',round(precision_score(y_test, y_pred_rf_GridSearchCV_
cm_rf = confusion_matrix(y_test, y_pred_rf_GridSearchCV_Applied)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_rf, annot=True, cmap='Oranges', fmt='g', ax=ax);
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
```

```
HYPERPARAMETER TUNED RANDOM FOREST MODEL RESULTS
Accuracy:  0.93885
F1 score:  0.78112
Recall:  0.728
Precision:  0.84259
```




```
In [76]: #Model Performance Comparison: Comparing the first model and the tuned mo
comparison_frame = pd.DataFrame({'Model':['Random Forest Classifier (Defa
                                          'Random Forest Classifier (Tune
                                 'Accuracy (Test Set)':[accuracy_score(y_
                                 'F1 Score (Test Set)':[f1_score(y_test,
                                 'Recall (Test Set)':[recall_score(y_test
                                 'Precision (Test Set)':[precision_score(

comparison_frame.style.highlight_max(color = 'red', axis = 0)
```

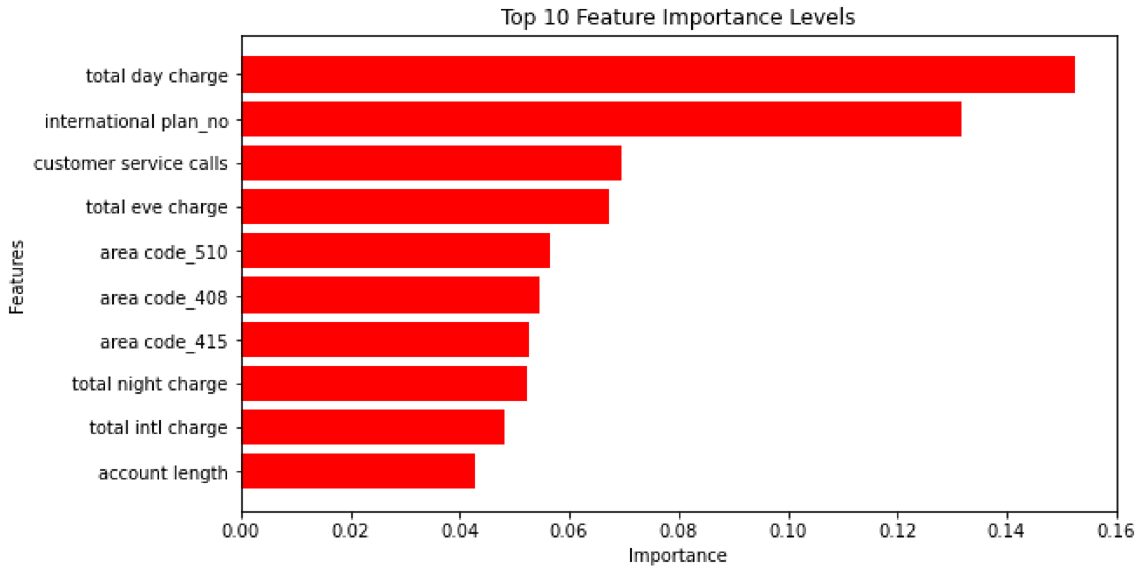
Out[76]:

	Model	Accuracy (Test Set)	F1 Score (Test Set)	Recall (Test Set)	Precision (Test Set)
0	Random Forest Classifier (Default)	0.935252	0.769231	0.720000	0.825688
1	Random Forest Classifier (Tuned)	0.938849	0.781116	0.728000	0.842593

In [77]:
```
"""Accuracy: The tuned Random Forest model improved in accuracy compared
F1 Score: The F1 score increased from 0.77 to 0.76, indicating an improve
Recall: Recall also increase from 0.70 to 0.72.
Precision: Precision increased from 0.83 to 0.84, indicating an improveme
```

Out[77]:
```
'Accuracy: The tuned Random Forest model improved in accuracy compared
to the default model 93.77% vs. 93.40%.\nF1 Score: The F1 score increas
ed from 0.77 to 0.76, indicating an improvement in the trade-off betwee
n precision and recall.\nRecall: Recall also increase from 0.70 to 0.7
2.\nPrecision: Precision increased from 0.83 to 0.84, indicating an imp
rovement in correctly classifying positive instances.'
```

In [78]:
```
#Important Feature Levels for Random Forest Classifier(First 10)
importance = pd.DataFrame({"Importance": rf_model_GridSearchCV_Applied.fe
importance = importance.sort_values(by="Importance", ascending=True).tail

plt.figure(figsize=(9, 5))
plt.barh(importance.index, importance["Importance"], color="red")
plt.title("Top 10 Feature Importance Levels")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.show()
```



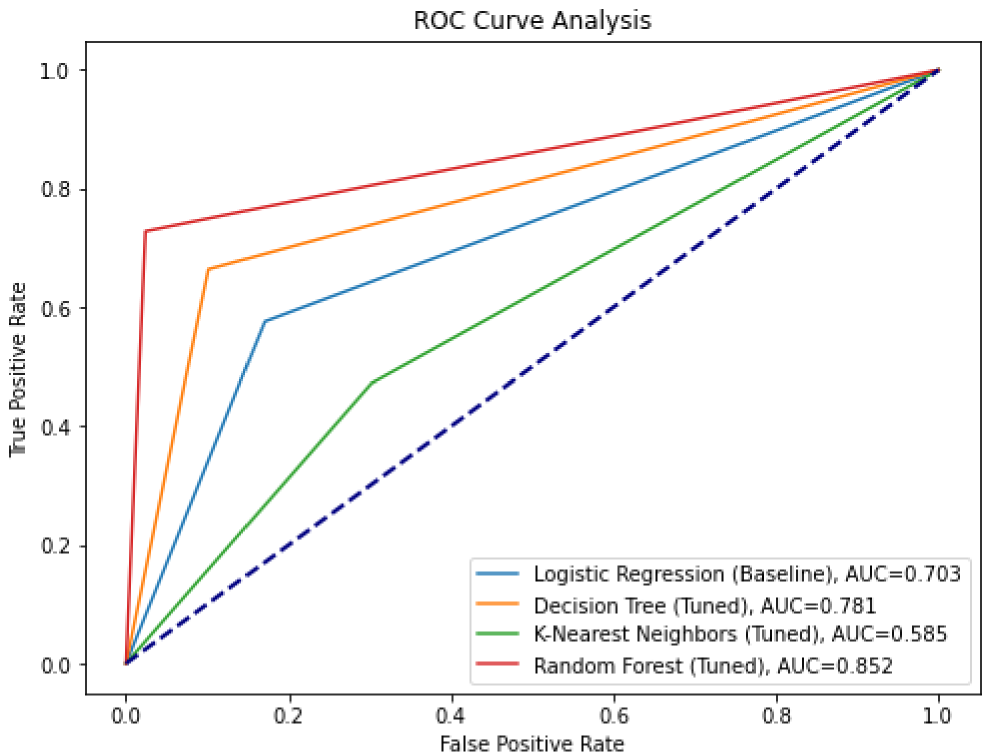
We can see that the most important features in the tuned random forest classifier are total day charge, international plan no and customer service calls

7. Model Evaluation

After performing hyperparameter tuning, it would be wise to evaluate the models and compare their results

7.1 Curve Analysis using ROC again

In [79]:
```
# Plotting ROC Curves again
plt.figure(figsize=(8, 6))
roc_auc_values = []

models = [('Logistic Regression (Baseline)', y_logistic_prediction),
          ('Decision Tree (Tuned)', y_pred_dt_GridSearchCV_Applied),
          ('K-Nearest Neighbors (Tuned)', y_pred_knn_GridSearchCV_Appliec
          ('Random Forest (Tuned)', y_pred_rf_GridSearchCV_Applied)]

for model_name, y_probs in models:
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    roc_auc = roc_auc_score(y_test, y_probs)
    roc_auc_values.append((model_name, roc_auc))
    plt.plot(fpr, tpr, label=f'{model_name}, AUC={roc_auc:.3f}')

# Sort models by AUC in descending order
roc_auc_values.sort(key=lambda x: x[1], reverse=True)
sorted_model_names = [model[0] for model in roc_auc_values]

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Analysis')
plt.legend()
plt.show()

print("\033[1mModels sorted by AUC in descending order:\n\033[0m")
for model_name in sorted_model_names:
    print(model_name)
```



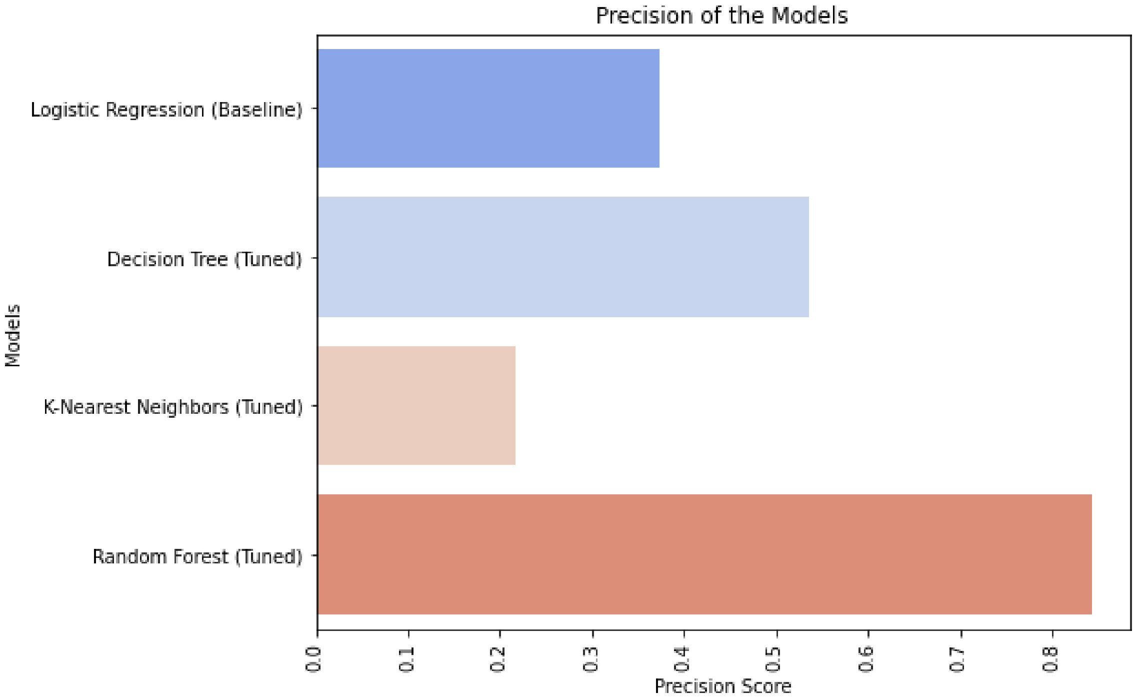
Models sorted by AUC in descending order:

```
Random Forest (Tuned)
Decision Tree (Tuned)
Logistic Regression (Baseline)
K-Nearest Neighbors (Tuned)
```

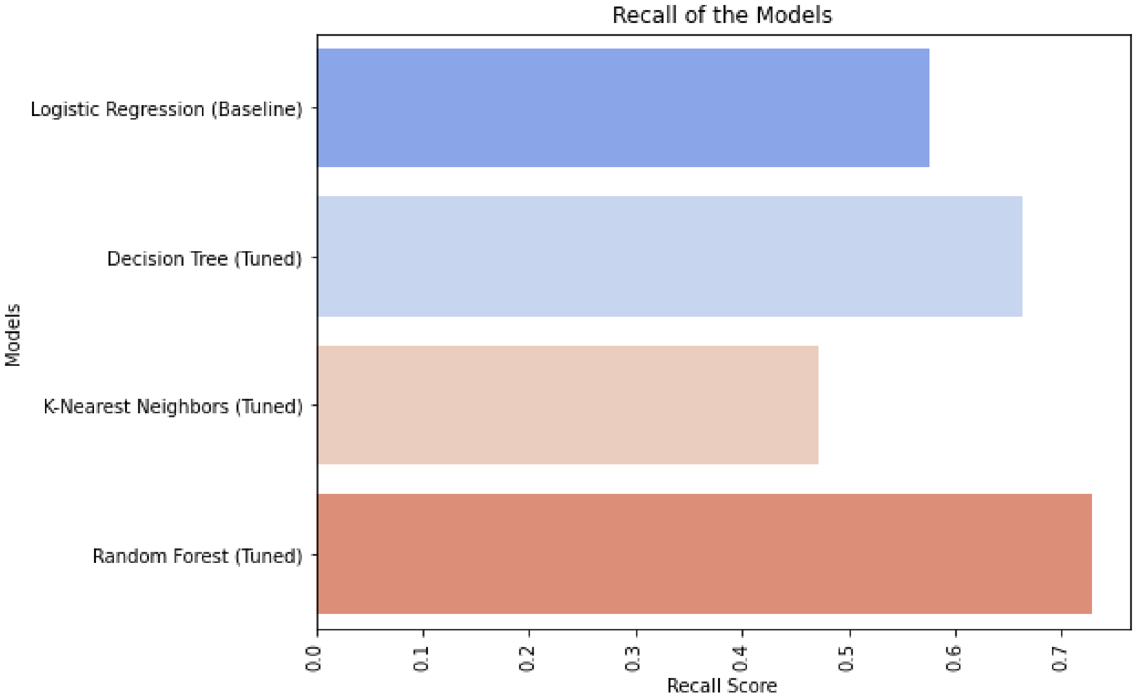
7.2 Model Comparison using K-fold cross Validation

In this section, we compare the accuracy of different models using cross-validation. The bar plots shows the precision, recall and accuracy scores of each model. This helps us identify the models that perform better in terms of overall precision, recall and accuracy.

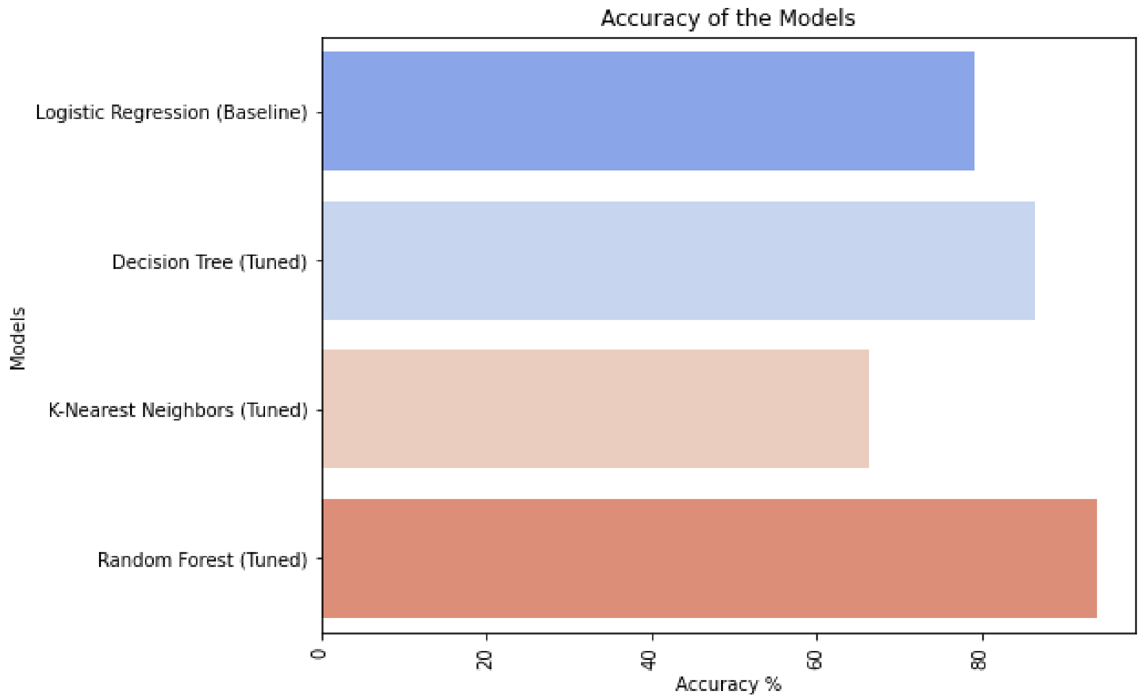
In [80]:
```
#calculating and visualizing the precision of each model
results_precision = pd.DataFrame(columns=["Models", "Precision"])

for model_name, y_probs in models:
    y_pred = (y_probs >= 0.5).astype(int)  # Assuming a threshold of 0.5
    precision = precision_score(y_test, y_pred)

    result = pd.DataFrame([[model_name, precision]], columns=["Models", "
    results_precision = results_precision.append(result)

# Plotting precision
plt.figure(figsize=(8, 6))
sns.barplot(x='Precision', y='Models', data=results_precision, palette="c
plt.xlabel('Precision Score')
plt.title('Precision of the Models')
plt.xticks(rotation=90)
plt.show()
```



In [81]:
```
#calculating and visualizing the recall of each model
results_recall = pd.DataFrame(columns=["Models", "Recall"])

for model_name, y_probs in models:
    y_pred = (y_probs >= 0.5).astype(int)  # Assuming a threshold of 0.5
    recall = recall_score(y_test, y_pred)

    result = pd.DataFrame([[model_name, recall]], columns=["Models", "Rec
    results_recall = results_recall.append(result)

# Plotting recall
plt.figure(figsize=(8, 6))
sns.barplot(x='Recall', y='Models', data=results_recall, palette="coolwar
plt.xlabel('Recall Score')
plt.title('Recall of the Models')
plt.xticks(rotation=90)
plt.show()
```



```
In [82]: #calculating and visualizing the accuracy of each model
results_accuracy = pd.DataFrame(columns=["Models", "Accuracy"])

for model_name, y_probs in models:
    y_pred = (y_probs >= 0.5).astype(int)  # Assuming a threshold of 0.5
    accuracy = accuracy_score(y_test, y_pred)

    result = pd.DataFrame([[model_name, accuracy * 100]], columns=["Model
    results_accuracy = results_accuracy.append(result)

# Plotting accuracy
plt.figure(figsize=(8, 6))
sns.barplot(x='Accuracy', y='Models', data=results_accuracy, palette="coo
plt.xlabel('Accuracy %')
plt.title('Accuracy of the Models')
plt.xticks(rotation=90)
plt.show()
```



7.3 Selecting the best model

The Random Forest model is the most suitable choice due to its strong performance in terms of accuracy, recall, and precision. It achieved an accuracy of 94% and a recall of 0.73, precision of 0.85 indicating its ability to accurately classify instances and achieve a balance between precision and recall.

7.4 Applying Sequential Feature Selection(SFS) Technique to the Random Forest model

Sequential Feature Selector (SFS) is used to enhance the analysis by iteratively choosing the most important features for the specific task. It aims to reduce complexity and boost the performance of your model by selecting features according to predefined criteria. SFS systematically explores various combinations of features and assesses how they influence the model's performance. The ultimate goal is to improve the interpretability and

In [83]:
```
#check feature columns in our dataset
df.columns
```

Out[83]:
```
Index(['state', 'account length', 'number vmail messages', 'total day c
alls',
       'total day charge', 'total eve calls', 'total eve charge',
       'total night calls', 'total night charge', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn',
       'international plan_no', 'international plan_yes', 'voice mail p
lan_no',
       'voice mail plan_yes', 'area code_408', 'area code_415',
       'area code_510'],
      dtype='object')
```

In [84]:

```
#Sequential forward selection
rf = RandomForestClassifier(criterion='entropy', max_depth=20, max_featur
sfs1 = SFS(rf, k_features=10, forward=True, floating=False, verbose=False
sfs1 = sfs1.fit(X, y)
sfs1.subsets_
```

```
Out[84]: {1: {'feature_idx': (4,),
  'cv_scores': array([0.24489796, 0.25095057, 0.26923077]),
  'avg_score': 0.2550264329188827,
  'feature_names': ('total day charge',)},
 2: {'feature_idx': (4, 11),
  'cv_scores': array([0.48208469, 0.38405797, 0.46687697]),
  'avg_score': 0.4443398772569045,
  'feature_names': ('total day charge', 'customer service calls')},
 3: {'feature_idx': (4, 6, 11),
  'cv_scores': array([0.60583942, 0.5546875 , 0.6       ]),
  'avg_score': 0.586842305352798,
  'feature_names': ('total day charge',
   'total eve charge',
   'customer service calls')},
 4: {'feature_idx': (4, 6, 11, 15),
  'cv_scores': array([0.66911765, 0.6490566 , 0.68992248]),
  'avg_score': 0.6693655771508545,
  'feature_names': ('total day charge',
   'total eve charge',
   'customer service calls',
   'voice mail plan_yes')},
 5: {'feature_idx': (4, 6, 8, 11, 15),
  'cv_scores': array([0.68913858, 0.71755725, 0.70542636]),
  'avg_score': 0.7040407284255235,
  'feature_names': ('total day charge',
   'total eve charge',
   'total night charge',
   'customer service calls',
   'voice mail plan_yes')},
 6: {'feature_idx': (4, 6, 8, 11, 15, 18),
  'cv_scores': array([0.68421053, 0.71428571, 0.70038911]),
  'avg_score': 0.6996284485532899,
  'feature_names': ('total day charge',
   'total eve charge',
   'total night charge',
   'customer service calls',
   'voice mail plan_yes',
   'area code_510')},
 7: {'feature_idx': (0, 4, 6, 8, 11, 15, 18),
  'cv_scores': array([0.66923077, 0.72243346, 0.67741935]),
  'avg_score': 0.6896945280485082,
  'feature_names': ('state',
   'total day charge',
   'total eve charge',
   'total night charge',
   'customer service calls',
   'voice mail plan_yes',
   'area code_510')},
 8: {'feature_idx': (0, 4, 6, 8, 9, 11, 15, 18),
  'cv_scores': array([0.66666667, 0.69803922, 0.66666667]),
  'avg_score': 0.6771241830065359,
  'feature_names': ('state',
   'total day charge',
   'total eve charge',
   'total night charge',
   'total intl calls',
   'customer service calls',
   'voice mail plan_yes',
   'area code_510')},
 9: {'feature_idx': (0, 4, 6, 8, 9, 11, 12, 15, 18),
  'cv_scores': array([0.73188406, 0.79298246, 0.76579926]),
```

```
 'avg_score': 0.7635552568723138,
 'feature_names': ('state',
  'total day charge',
  'total eve charge',
  'total night charge',
  'total intl calls',
  'customer service calls',
  'international plan_no',
  'voice mail plan_yes',
  'area code_510')},
10: {'feature_idx': (0, 4, 6, 8, 9, 10, 11, 12, 15, 18),
 'cv_scores': array([0.80139373, 0.85616438, 0.79562044]),
 'avg_score': 0.8177261832469481,
 'feature_names': ('state',
  'total day charge',
  'total eve charge',
  'total night charge',
  'total intl calls',
  'total intl charge',
  'customer service calls',
  'international plan_no',
  'voice mail plan_yes',
  'area code_510')}}
```

In [85]:
```
#getting the selected features
sfs1.k_feature_names_
```

Out[85]:
```
('state',
 'total day charge',
 'total eve charge',
 'total night charge',
 'total intl calls',
 'total intl charge',
 'customer service calls',
 'international plan_no',
 'voice mail plan_yes',
 'area code_510')
```

8. Findings

Based on the analysis the top 10 features that have the most significant impact on customer churn are as follows:

Total day charge

Total eve charge

Customer service calls

Total night calls

Total night charge

total intl charge

Number vmail messages

Voice_mail_plan_is_yes

voice_mail_plan_is_no

9. Recommendations

After a comprehensive analysis and delving into key facets of service usage, customer interactions, and market dynamics. The following recommendations emerge as strategic pillars poised to enhance customer satisfaction and reduce customer churn;

1.Optimize Daytime and Evening Charges: Reduce charges during daytime and evening usage to save customers money. Introduce competitive plans aligned with customer preferences to enhance satisfaction and retention. Address Customer Service Calls:

2.Monitor and address customer service calls promptly. Identify and resolve issues behind frequent calls to improve overall customer experience. Ensure Fair Nighttime Charges:

3.Evaluate nighttime charges to ensure fairness and alignment with customer expectations. Benchmark charges against market standards to guarantee reasonable pricing. Enhance Daytime Customer Satisfaction:

4.Improve customer satisfaction during daytime usage by analyzing call patterns. Implement strategies to enhance the overall experience and retain customers. Tailor Strategies to Each State:

5.Understand specific factors related to each state to tailor marketing and retention initiatives accordingly. Adjust plans based on unique customer needs in each state. Optimize Voicemail Plans:

6.Evaluate the effectiveness of voicemail plans in retaining customers. Enhance voicemail features and benefits to increase customer loyalty. Analyze Churn Patterns in Different Areas:

7.Examine customer churn patterns associated with different area codes. Identify specific issues or challenges faced by customers in those areas and develop targeted retention strategies. Adapt Marketing Strategies:

8.Tailor marketing campaigns based on insights gained from the analysis. Use information about factors contributing to customer churn to create more effective and personalized marketing efforts. Monitor Customer Satisfaction:

9.Regularly assess customer satisfaction levels through surveys, feedback mechanisms, and customer interactions. Identify and address potential pain points or areas where customers might be dissatisfied to proactively prevent churn. Leverage Predictive Models:

10.Implement the tuned Random Forest model to predict customer churn in real-time. Continuously update and refine the model based on new data to improve its accuracy and effectiveness in predicting customer behavior

10. Conclusion

In the analysis I used different machine learning models to predict if customers might stop using syriatel services.Testing many models like Logistic Regression, Random Forest, Decision Tree and K-Nearest Neighbors and comparing their performance.I used measures like accuracy, F1 score, recall, and precision to check how good they are.

Out of all the models tested, the Random Forest classifier (after adjusting it) did the best job at figuring out if a customer might stop using the services(churn).

Additionally,I used the Sequential Forward Selection (SFS) method on the Random Forest program.It helped to find the top 10 things that really matter in guessing if a customer might leave.