1.0 BUSINESS UNDERSTANDING

1.1 Background

SyriaTel is one of the leading telecommunication companies in Syria. It provides a wide range of telecommunications services, including mobile and fixed-line telephony, internet services and data services.

Syriatel has been a key player in the Syrian telecommunications market, serving millions of customers across the country. The company has played a significant role in expanding and modernizing telecommunications infrastructure, contributing to the country's connectivity and economic development.

The telecommunications company is interested in reducing how much money is lost because of customers who don't stick around very long.

1.2 Problem Statement

In this competitive world, business is becoming highly saturated. Especially, the field of telecommunication faces complex challenges due to a number of vibrant competitive service providers. Therefore, it has become very difficult for them to retain existing customers. Since the cost of acquiring new customers is much higher than the cost of retaining the existing customers, it is the time for the telecom industries to take necessary steps to retain the customers to stabilize their market value.

1.3 Objectives

1. To calculate the churn rate at SyriaTel, a telecommunications company.
2. To identify the factors that lead to churn and those that help in customer retention.

2.0 DATA UNDERSTANDING

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import re
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, 
StandardScaler, normalize
from sklearn.model_selection import train_test_split, cross_val_score, 
RandomizedSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,
```

```
GradientBoostingClassifier
from sklearn.svm import SVC
# %pip install xgboost
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
df=pd.read_csv(r"C:\Users\user\Desktop\Phase_3_project\
customer_churn_dataset.csv")
df.head(5)
```

```
  state  account length  area code phone number international plan  \
0    KS             128        415     382-4657                 no
1    OH             107        415     371-7191                 no
2    NJ             137        415     358-1921                 no
3    OH              84        408     375-9999                yes
4    OK              75        415     330-6626                yes

  voice mail plan  number vmail messages  total day minutes  total day
calls  \
0             yes                     25              265.1
110
1             yes                     26              161.6
123
2              no                      0              243.4
114
3              no                      0              299.4
71
4              no                      0              166.7
113

   total day charge  ...  total eve calls  total eve charge  \
0             45.07  ...               99             16.78
1             27.47  ...              103             16.62
2             41.38  ...              110             10.30
3             50.90  ...               88              5.26
4             28.34  ...              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
```

```
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]
```

```
# Better view of the dataframe since it has many columns
# Transposing does not transform or modify the original df. It only
# enhances the visibility of the many columns

df.head().T
```

```
                               0         1         2         3
4
state                         KS        OH        NJ        OH
OK
account length               128       107       137        84
75
area code                    415       415       415       408
415
phone number            382-4657  371-7191  358-1921  375-9999  330-
6626
international plan            no        no        no       yes
yes
voice mail plan              yes       yes        no        no
no
number vmail messages         25        26         0         0
0
total day minutes          265.1     161.6     243.4     299.4
166.7
total day calls              110       123       114        71
113
total day charge           45.07     27.47     41.38      50.9
28.34
total eve minutes          197.4     195.5     121.2      61.9
148.3
total eve calls               99       103       110        88
122
total eve charge           16.78     16.62      10.3      5.26
12.61
total night minutes        244.7     254.4     162.6     196.9
186.9
total night calls             91       103       104        89
121
```

```
total night charge          11.01      11.45       7.32       8.86
8.41
total intl minutes           10.0       13.7       12.2        6.6
10.1
total intl calls                3          3          5          7
3
total intl charge             2.7        3.7       3.29       1.78
2.73
customer service calls          1          1          0          2
3
churn                       False      False      False      False
False
```

```
# number of rows and columns in the dataframe
# Each row represents a record of a customer

print("The number of rows in the SyriaTel dataframe is", df.shape[0])
print("The number of columns in the SyriaTel dataframe is",
df.shape[1])
```

```
The number of rows in the SyriaTel dataframe is 3333
The number of columns in the SyriaTel dataframe is 21
```

```
# Getting column information such as the Datatype and number of non-
null values
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
```

```
#  Getting the names of the columns

df.columns
```

```
Index(['state', 'account length', 'area code', 'phone number',
       '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 calls', 'churn'],
      dtype='object')
```

```
# getting the descriptive statistics of the dataframe

df.describe()
```

```
       account length    area code  number vmail messages  total day
minutes  \
count     3333.000000  3333.000000            3333.000000
3333.000000
mean       101.064806   437.182418               8.099010
179.775098
std         39.822106    42.371290              13.688365
54.467389
min          1.000000   408.000000               0.000000
0.000000
25%         74.000000   408.000000               0.000000
143.700000
50%        101.000000   415.000000               0.000000
179.400000
75%        127.000000   510.000000              20.000000
216.400000
max        243.000000   510.000000              51.000000
350.800000

       total day calls  total day charge  total eve minutes  total eve
calls  \
count      3333.000000       3333.000000        3333.000000
3333.000000
mean        100.435644         30.562307         200.980348
100.114311
```

```
std          20.069084           9.259435          50.713844
19.922625
min           0.000000           0.000000           0.000000
0.000000
25%          87.000000          24.430000         166.600000
87.000000
50%         101.000000          30.500000         201.400000
100.000000
75%         114.000000          36.790000         235.300000
114.000000
max         165.000000          59.640000         363.700000
170.000000

       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
75%             3.270000                2.000000
max             5.400000                9.000000
```

Observations

The minimum number of voicemail messages, total day minutes, total day calls, total day charge, total evening minutes, total evening calls and total evening charge are all 0.

```python
# Understanding the current Churn Status

df["churn"].value_counts()
```

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

It is evident that there are 483 disloyal customers who have churned the services of SyriaTel

3.0 DATA PREPARATION.

3.1 Data Cleaning

```python
# checking for duplicates

df.duplicated().value_counts()
```

```
False    3333
dtype: int64
```

Observation: There are no duplicates in the 3333 records.

```python
# Checking for missing values

df.isna().sum()
```

```
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
```

Observation: No column has a missing value

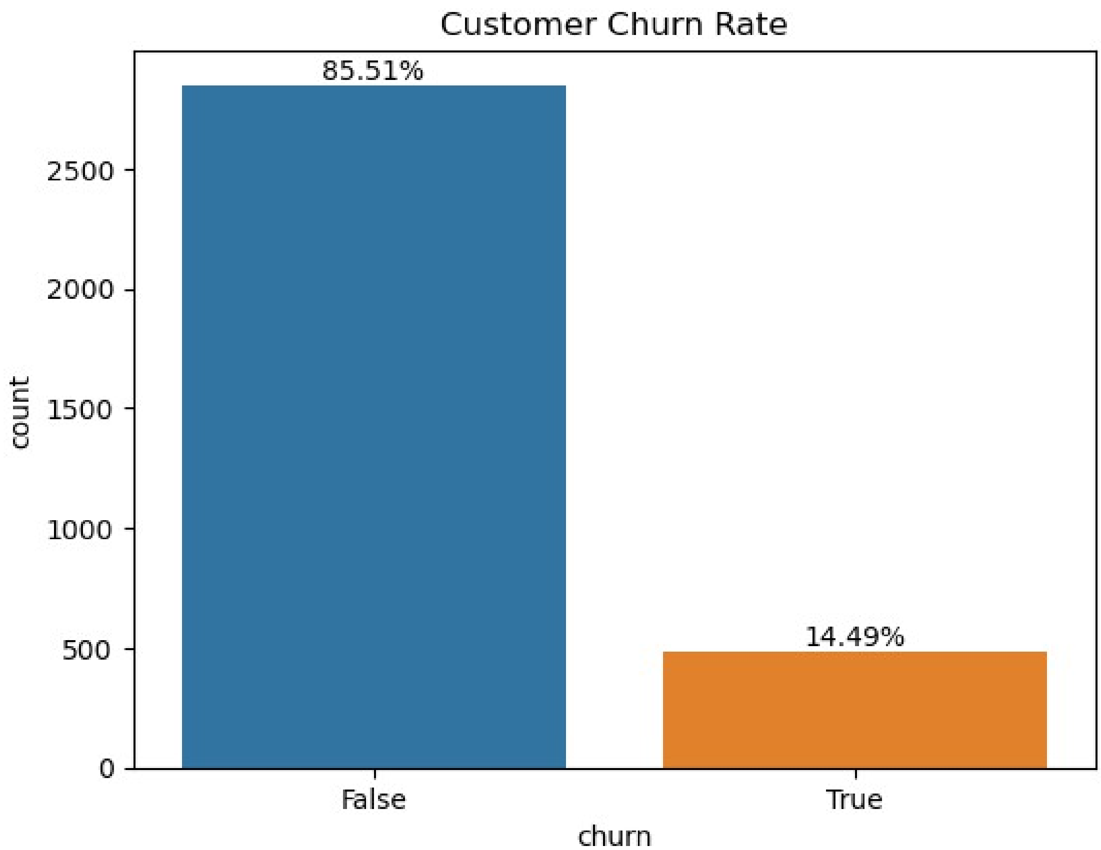
```
# Checking the unique values of the states

df.state.unique()

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

# checking for the number of unique values in state

no_of_unique=df.state.nunique()

print("Observation: There are {} unique states represented in this
Dataframe.".format(no_of_unique))

Observation: There are 51 unique states represented in this Dataframe.

# Removing phone number as it has no use here

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

# Confirming that we are now working with 20 columns


print("Number of columns are
now",df.columns.value_counts().sum(),"after removing phone number
column.")

Number of columns are now 20 after removing phone number column.
```

3.2 EXPLORATORY DATA ANALYSIS

3.2.1 Univariate Analysis

```
sns.countplot(x="churn", data=df)
plt.title ("Customer Churn Rate")

# Calculating churn percentages
total_count = df.shape[0]  #rows/total num of customers
churn_counts = df["churn"].value_counts()    # value count of each of
the two unique values
churn_percentages = churn_counts / total_count * 100

# Annotating the bars with churn percentages
```

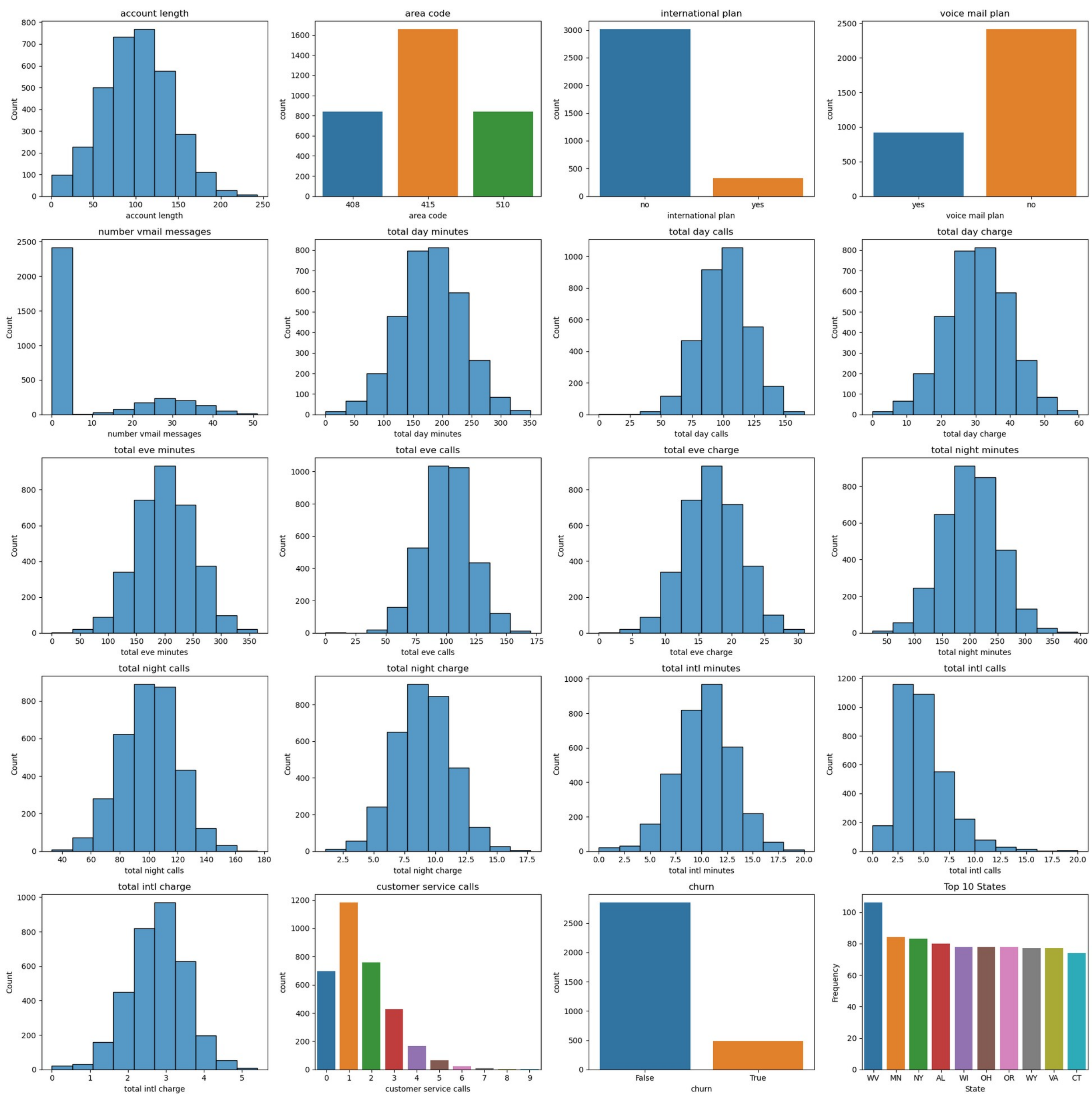
```
for i, percentage in enumerate(churn_percentages):
    plt.text(i, churn_counts[i], f'{percentage:.2f}%', ha="center",
va= "bottom")

plt.show()
```



Observation: The churn rate currently stands at 14.49%

```
# State column will be plotted separately from the countplot,
# despite being nominal. This will be done due to high number of
unique states(51) which cannot fit well in a countplot.

columns_to_plot = [col for col in df.columns if col != "state"]

# Setting up the figure and axes for subplots
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(20, 20))

# Flattening the axes for easier iteration
axes = axes.flatten()

# Iterating over each column and create countplot(nominal/object) or
histogram(discreet)
```

```python
for i, column in enumerate(columns_to_plot):
    if df[column].dtype == 'object' or df[column].nunique() < 20:
        sns.countplot(x=column, data=df, ax=axes[i])
    else:
        sns.histplot(df[column], bins= 10, kde=False, ax=axes[i])
    axes[i].set_title(column)  # title for each subplot



# now i will plot the top 10 states

state_counts = df['state'].value_counts().head(10)  # Top 10 states
sns.barplot(x=state_counts.index, y=state_counts.values)
plt.title('Top 10 States')
plt.xlabel('State')
plt.ylabel('Frequency')




# Adjusting layout to prevent overlap
plt.tight_layout()

plt.show()
```




```
skewness=df.skew()
skewness
```

```
account length            0.096606
area code                 1.126823
number vmail messages     1.264824
total day minutes        -0.029077
total day calls          -0.111787
total day charge         -0.029083
total eve minutes        -0.023877
total eve calls          -0.055563
total eve charge         -0.023858
```

```
total night minutes       0.008921
total night calls         0.032500
total night charge        0.008886
total intl minutes       -0.245136
total intl calls          1.321478
total intl charge        -0.245287
customer service calls    1.091359
churn                     2.018356
dtype: float64
```

Observation:

Based on the distribution and information above, The following columns were highly skewed (values > 1 or <-1):

1.Area code The distribution is highly skewed to the right. There are likely a few area codes that occur much more frequently than others.

2.Number of voicemail messages The distribution is highly skewed to the right. Most customers likely have few or no voicemail messages, with a few customers having many.

3.Total international calls The distribution is highly skewed to the right. Most customers likely make very few international calls, with a few customers making many.

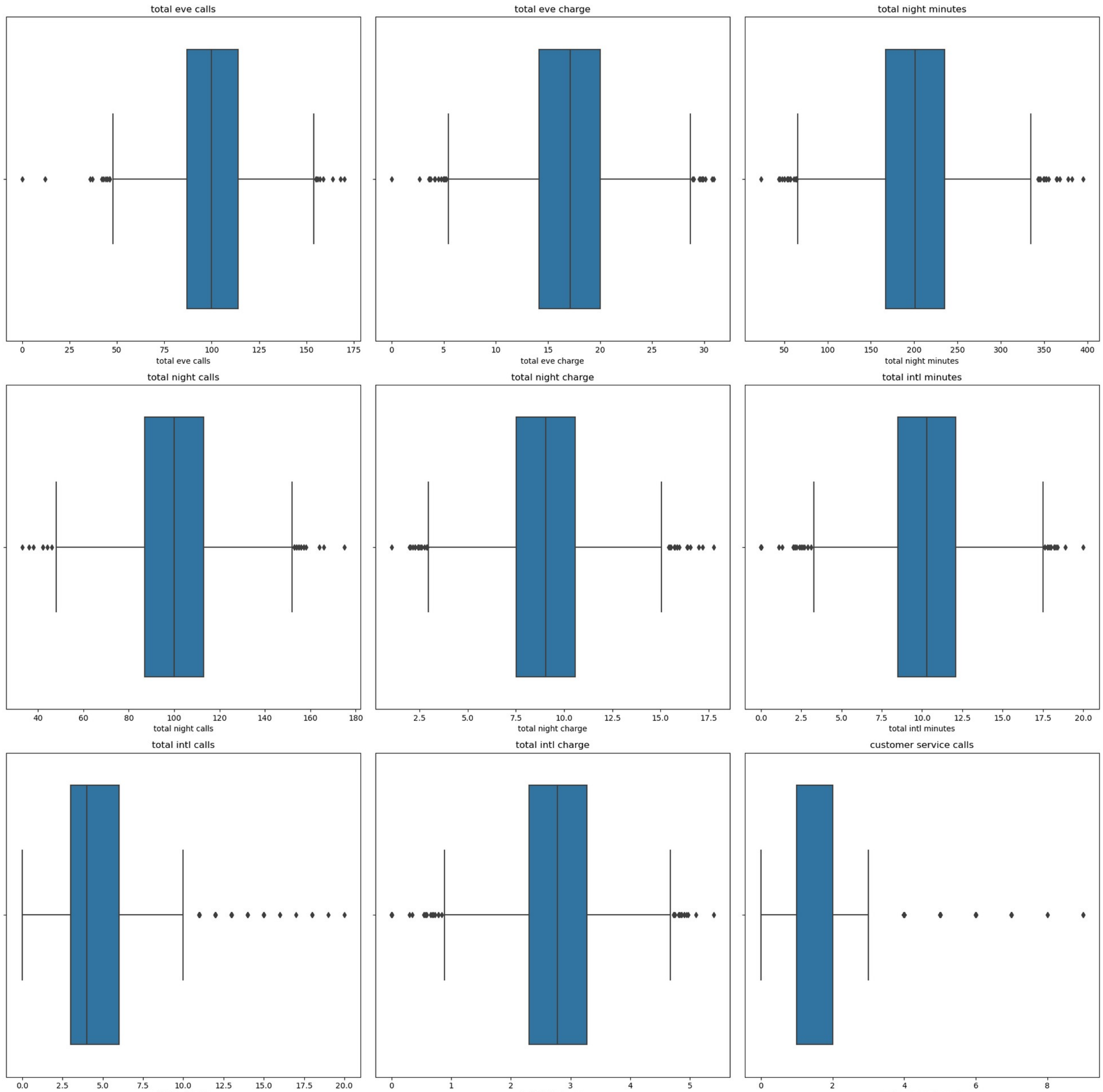
4.Customer service calls The distribution is highly skewed to the right. Most customers likely make very few customer service calls, with a few customers making many.

5.Churn The target variable was highly skewed to the right. This indicates that most customers did not churn, but a few did.

To correct this, various ensemble methods will be applied when modelling.

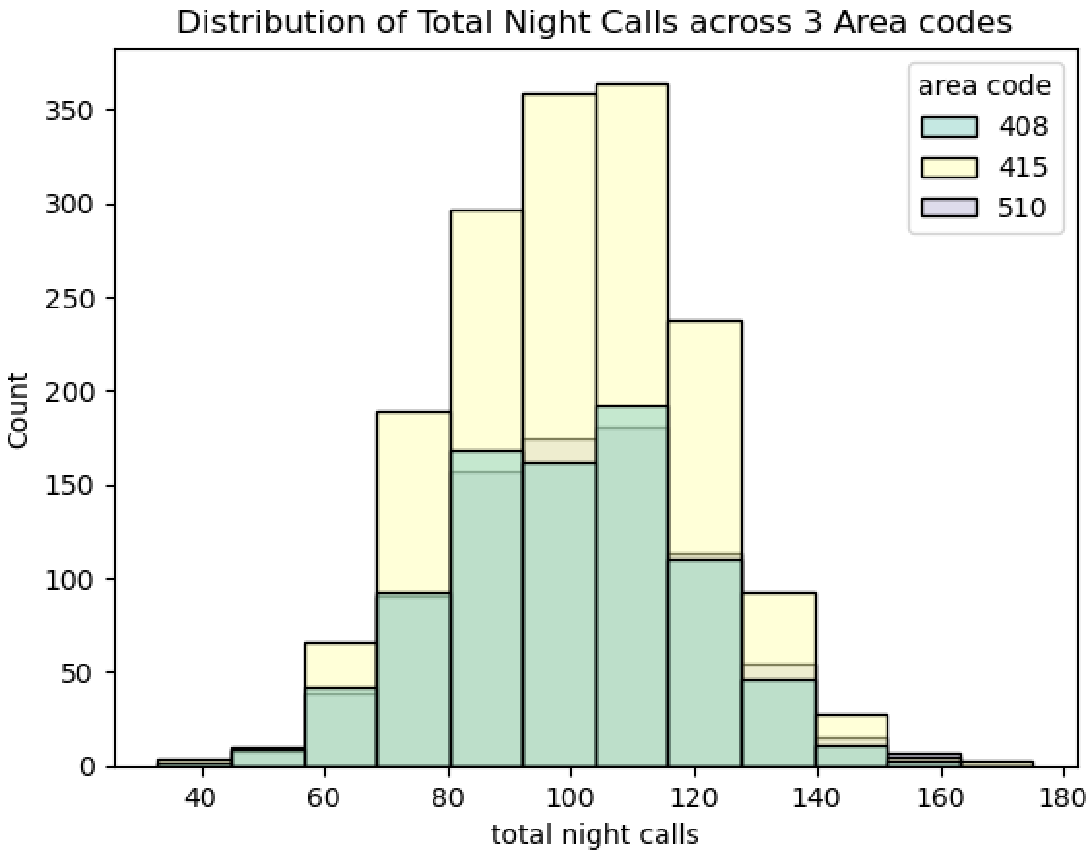
Visualizing the Outliers

```
cols_boxplot = df.columns[[10, 11, 12,13, 14,15, 16, 17, 18,]]

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(20, 20))

axes = axes.flatten()


for i, column in enumerate(cols_boxplot):
    sns.boxplot(x=df[column], ax=axes[i])
    axes[i].set_title(column)


plt.tight_layout()
plt.show()
```



3.2.2 Bivariate Analysis

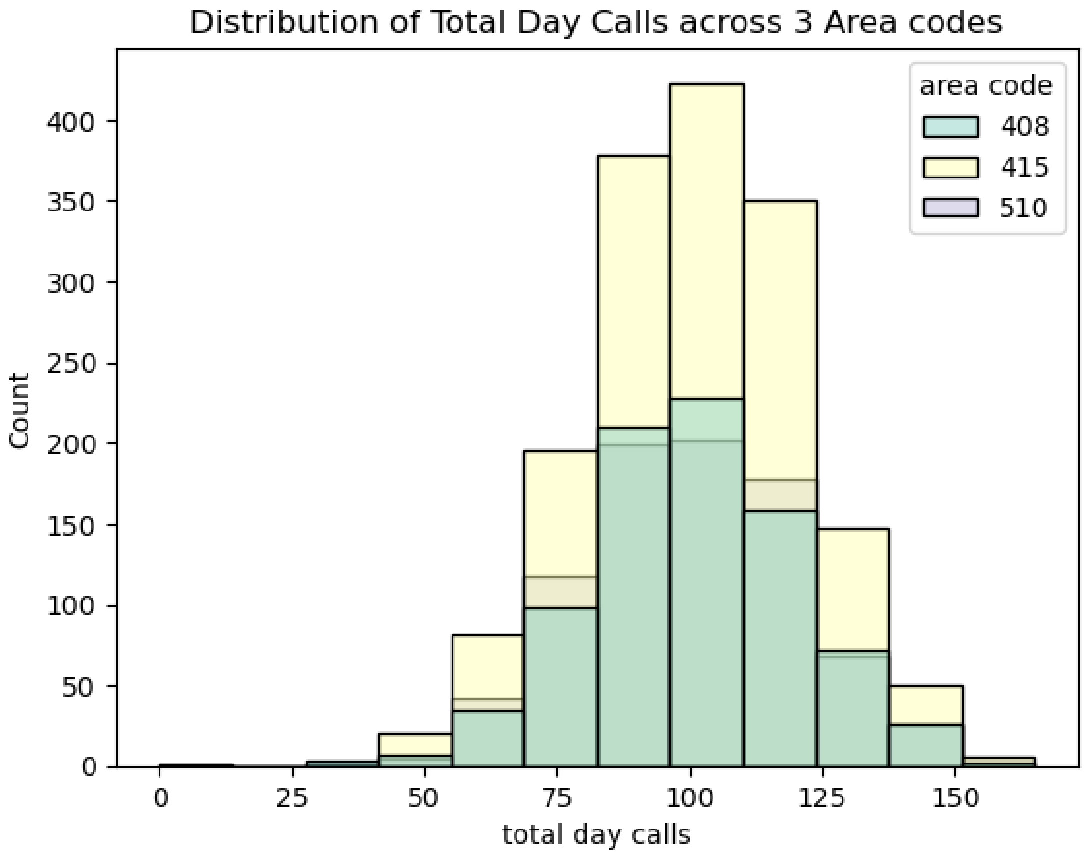
```
sns.histplot(x="total night calls", kde=False, bins=12, hue= "area
code", data=df, palette="Set3")
plt.title("Distribution of Total Night Calls across 3 Area codes")
plt.show

<function matplotlib.pyplot.show(close=None, block=None)>
```



Observation:

Area code 415 recorded the highest number of night calls made.

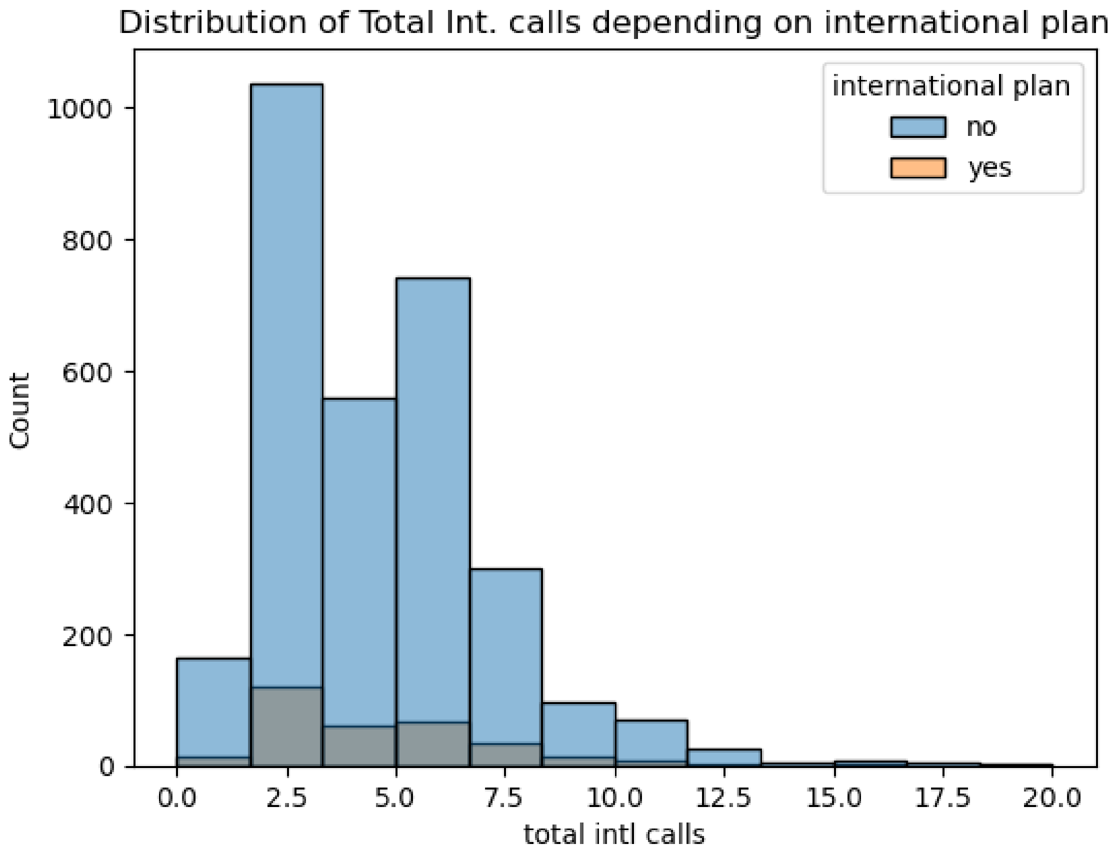
```
sns.histplot(x="total day calls",kde= False, bins=12, hue = "area
code", data=df, palette= "Set3")
plt.title("Distribution of Total Day Calls across 3 Area codes")
plt.show()
```



Observation:

Area code 415 also had the most day calls.

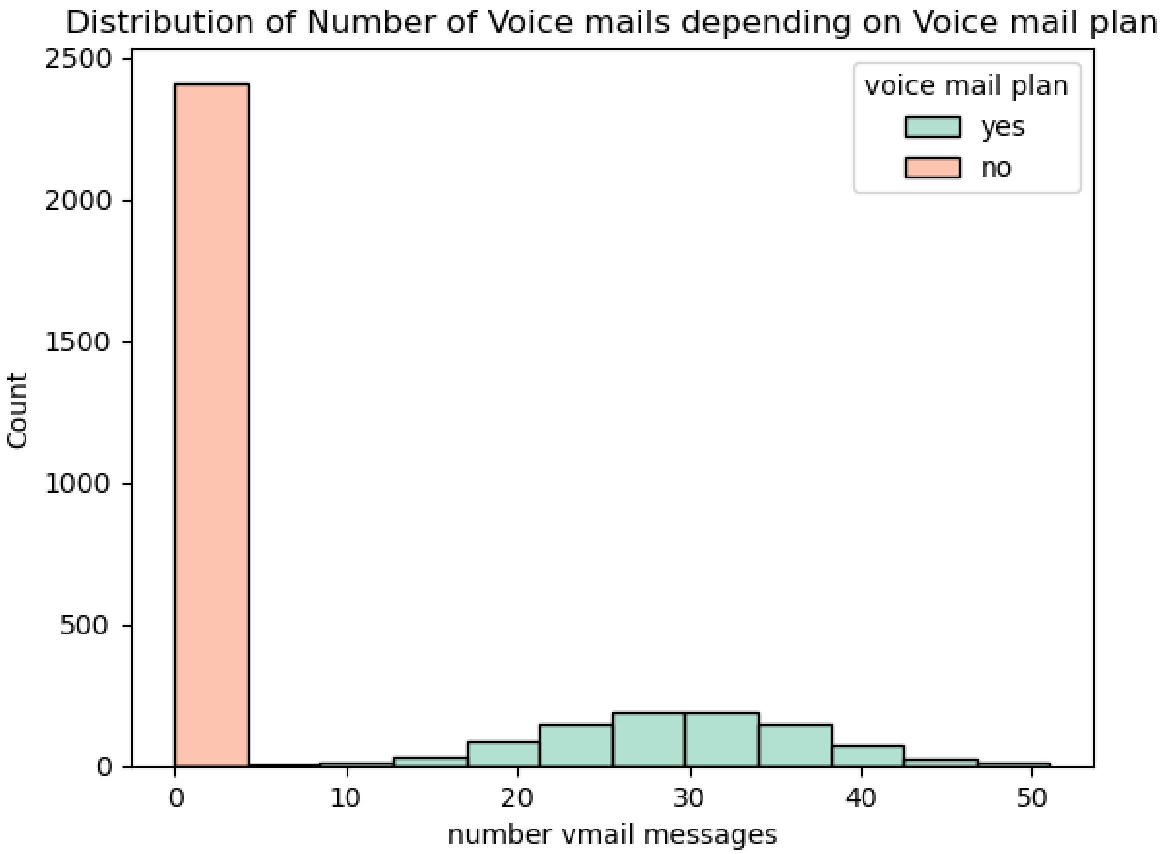
```
sns.histplot(x="total intl calls", data = df,bins=12,
hue="international plan", kde= False)
plt.title("Distribution of Total Int. calls depending on international
plan")
plt.show()
```



Observation:

Most customers that made international calls did not have an international plan.

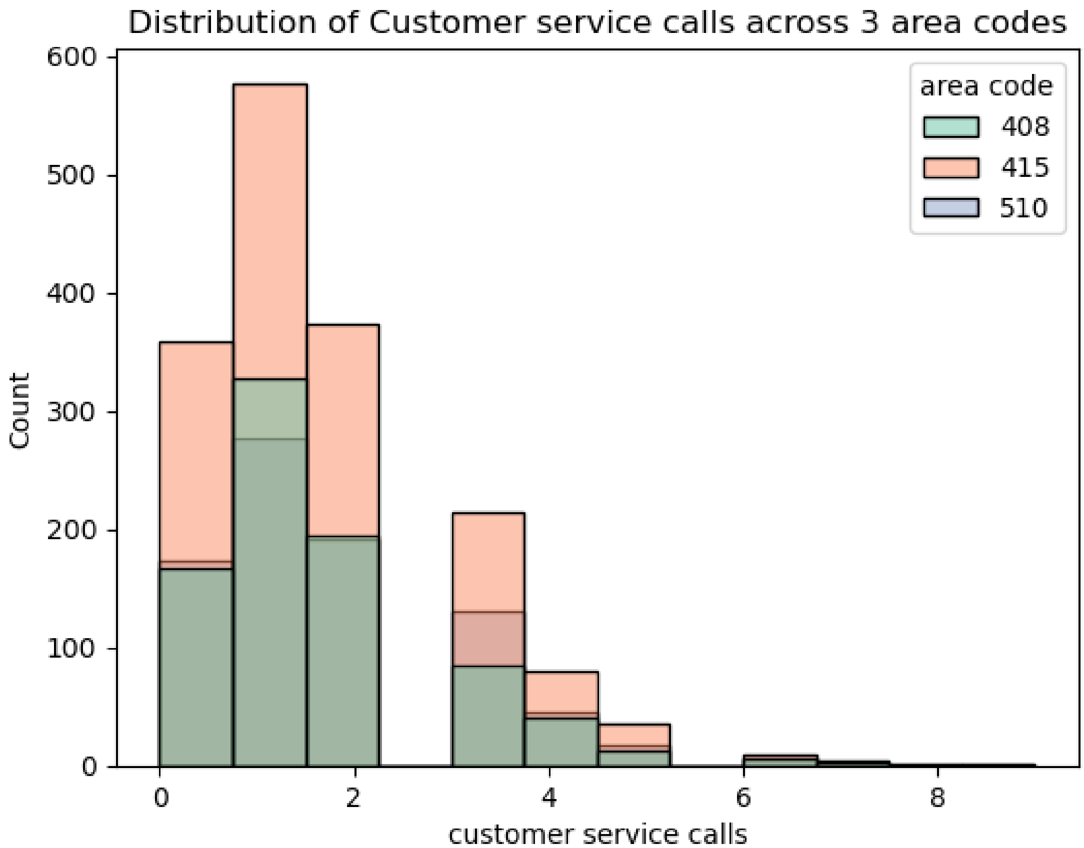
```
sns.histplot(x="number vmail messages", data = df, bins=12, hue="voice
mail plan", kde= False, palette= "Set2")
plt.title("Distribution of Number of Voice mails depending on Voice
mail plan")
plt.show()
```



Observation:

Many customers who sent voicemail messages had a voice mail plan.

```
sns.histplot(x="customer service calls", bins=12, kde= False,
hue="area code", data=df, palette="Set2")
plt.title("Distribution of Customer service calls across 3 area
codes")
plt.show()
```



Observation:

Area code 415 once again had the most customer service calls.

It can be therefore confirmed that Area Code 415 was the most active region in using SyriaTel Telecommunications services.

3.2.3 Multivariate Analysis

I will start with a heatmap of all the numerical values to see how the correlate with each other

```
cols_corr=[
       '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', 'churn']


plt.figure(figsize=(16,8))
sns.heatmap(data=df[cols_corr].corr(), annot=True)
```

```
plt.title("Correlation Heatmap")
plt.show()
```

4.0 MODELLING

```
# Assigning the target and predictor variables

X, y = df[df.columns.difference(["churn"])], df["churn"]

# Mapping categorical columns
df['churn'] = df['churn'].map({False: 0, True: 1})
df2 = pd.get_dummies(df, columns=['state', 'area code'])
df2['international plan'] = df2['international plan'].map({'no': 0,
'yes': 1})
df2['voice mail plan'] = df2['voice mail plan'].map({'no': 0, 'yes':
1})

# Splitting the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Identifying categorical and numerical columns

categorical_cols = ["state", "international plan", "voice mail plan"]
```

```
numerical_cols = X.select_dtypes(include=["int64",
"float64"]).columns.tolist()

# Removing categorical columns from numerical columns list if present
numerical_cols = [col for col in numerical_cols if col not in
categorical_cols]

# Defining the column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Creating the pipeline with preprocessing and baseline model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier',
LogisticRegression(random_state=42))])

# Fitting the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predicting on the test data
y_pred = pipeline.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

Accuracy: 0.85
```

- The first model, Logistic regression, which was the baseline model gave an accuracy of 85%
- The second model will be still Logistic Regression but with additional hyper parameters.

```
# The second model


pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier',
LogisticRegression(random_state=42,
                                                             C=1e3,

max_iter=500,

solver='lbfgs',

multi_class='auto'))])
```

```python
# Fitting the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predicting on the test data
y_pred = pipeline.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.86
```

- The second model improved slightly by 1%
- The third model will entail ensemble methods;

```python
# Defining hyperparameter grids for each model

param_grid_knn = {'n_neighbors': [3, 5, 7, 10]}

param_grid_svc = {'C': [0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1,
'scale', 'auto']}

param_grid_dt = {'max_depth': [None, 5, 10, 20], 'min_samples_split':
[2, 5, 10]}

param_grid_rf = {'n_estimators': [100, 200, 300], 'max_depth': [None,
5, 10, 20], 'min_samples_split': [2, 5, 10]}

param_grid_gb = {'n_estimators': [100, 200, 300], 'learning_rate':
[0.01, 0.1, 0.5], 'max_depth': [3, 5, 7]}

param_grid_xgb = {'n_estimators': [100, 200, 300], 'learning_rate':
[0.01, 0.1, 0.5], 'max_depth': [3, 5, 7]}



df['churn'] = df['churn'].map({False: 0, True: 1}) #remmaping again to
ensure all data is mapped

#  Defining categorical and numerical columns

categorical_cols = ['state', 'international plan', 'voice mail plan',
'area code']
numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()

# Creating the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
```

```
        ('cat', OneHotEncoder(drop='first'), categorical_cols)
    ])

# Defining the models and parameter grids
models = {
    'KNN': (KNeighborsClassifier(), {'classifier__n_neighbors': [3, 5,
7, 10]}),
    'SVM': (SVC(), {'classifier__C': [0.1, 1, 10, 100],
'classifier__gamma': [0.01, 0.1, 1, 'scale', 'auto']}),
    'Decision Tree': (DecisionTreeClassifier(),
{'classifier__max_depth': [None, 5, 10, 20],
'classifier__min_samples_split': [2, 5, 10]}),
    'Random Forest': (RandomForestClassifier(),
{'classifier__n_estimators': [100, 200, 300], 'classifier__max_depth':
[None, 5, 10, 20], 'classifier__min_samples_split': [2, 5, 10]}),
    'Gradient Boosting': (GradientBoostingClassifier(),
{'classifier__n_estimators': [100, 200, 300],
'classifier__learning_rate': [0.01, 0.1, 0.5],
'classifier__max_depth': [3, 5, 7]}),
    'XGBoost': (XGBClassifier(), {'classifier__n_estimators': [100,
200, 300], 'classifier__learning_rate': [0.01, 0.1, 0.5],
'classifier__max_depth': [3, 5, 7]})
}

# Performing RandomizedSearchCV for each model
for name, (model, param_grid) in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])
    clf = RandomizedSearchCV(pipeline, param_distributions=param_grid,
n_iter=10, cv=5, random_state=42)
    clf.fit(X_train, y_train)
    print(f"Best parameters for {name}: {clf.best_params_}")
    print(f"Training accuracy for {name}: {clf.best_score_}")
    print(f"Test accuracy for {name}: {clf.score(X_test, y_test)}")
    print()
```

```
Best parameters for KNN: {'classifier__n_neighbors': 7}
Training accuracy for KNN: 0.8807246101847361
Test accuracy for KNN: 0.881559220389805

Best parameters for SVM: {'classifier__gamma': 0.1, 'classifier__C':
10}
Training accuracy for SVM: 0.8998468143713417
Test accuracy for SVM: 0.904047976011994

Best parameters for Decision Tree: {'classifier__min_samples_split':
5, 'classifier__max_depth': 5}
Training accuracy for Decision Tree: 0.9377342580685962
```

```
Test accuracy for Decision Tree: 0.9400299850074962

Best parameters for Random Forest: {'classifier__n_estimators': 200, 'classifier__min_samples_split': 5, 'classifier__max_depth': 20}
Training accuracy for Random Forest: 0.9366092571902384
Test accuracy for Random Forest: 0.9400299850074962

Best parameters for Gradient Boosting: {'classifier__n_estimators': 200, 'classifier__max_depth': 5, 'classifier__learning_rate': 0.1}
Training accuracy for Gradient Boosting: 0.9542396582133497
Test accuracy for Gradient Boosting: 0.9505247376311844

Best parameters for XGBoost: {'classifier__n_estimators': 300, 'classifier__max_depth': 7, 'classifier__learning_rate': 0.1}
Training accuracy for XGBoost: 0.9534905945429376
Test accuracy for XGBoost: 0.9535232383808095
```

5.0 MODEL EVALUATION

- First I trained the first model which was a Logistic Regression as the baseline Model and the second model was the Logistic Regression, but with additional hyperparamaters.
- Based on the Ensemble methods metrics, Gradient Boosting and XGBoost not only performed well on training data but also generalized effectively to unseen test data, suggesting a good balance between underfitting and overfitting.
- Consequently, SyriaTel should prioritize deploying Gradient Boosting and XGBoost models for their churn prediction system.

Next Steps:

Based on the insights derived from the models and feature importance analysis, SyriaTel can implement the following customer retention strategies:

- Proactive Customer Support: This could involve regular check-ins or offering dedicated support channels.
- Personalized Offers and Discounts: Use predictive models to identify customers at high risk of churn and provide them with personalized offers, discounts, or loyalty programs to enhance their satisfaction and loyalty.
- Improved Service Plans: Evaluate and potentially revamp service plans based on the usage patterns identified as critical churn factors. For instance, if customers with low voice mail plan usage are more likely to churn, consider bundling voice mail services with other popular plans.

CONCLUSION

By leveraging the high-performing Gradient Boosting and XGBoost models, SyriaTel can gain valuable insights into customer behavior and implement targeted retention strategies to reduce churn. Continuous monitoring, model updates, and integration with CRM systems will further enhance the effectiveness of these efforts, ultimately contributing to increased customer retention and sustained business growth.