Business Understanding

SyriaTel, a leading telecommunications provider in Syria, is experiencing an increase in customer churn. Understanding the factors leading to service discontinuation is crucial for enhancing customer retention strategies. This analysis focuses on identifying predictive features that might indicate potential churn.

Problem Statement

Classify the key indicators or features that correlate with customer churn in order to formulate proactive retention strategies.

Objectives

1. **Identify key indicators of customer churn:** Determine which customer characteristics, behaviors, or interactions are most strongly associated with discontinuing service.
2. **Develop a predictive model for customer churn:** Create a model that can accurately predict which customers are likely to churn in the future.
3. **Formulate proactive customer retention strategies:** Use the insights gained from the analysis to design and implement strategies to reduce customer churn, such as targeted promotions, improved customer service, or personalized communication.

Data Understanding

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import plotly.express as px


warnings.filterwarnings('ignore')


df= pd.read_csv('/bigml_59c28831336c6604c800002a.csv')
df.head()
```

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

5 rows × 21 columns

```
df.describe()
```

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	to
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	30

```
df.info()
```

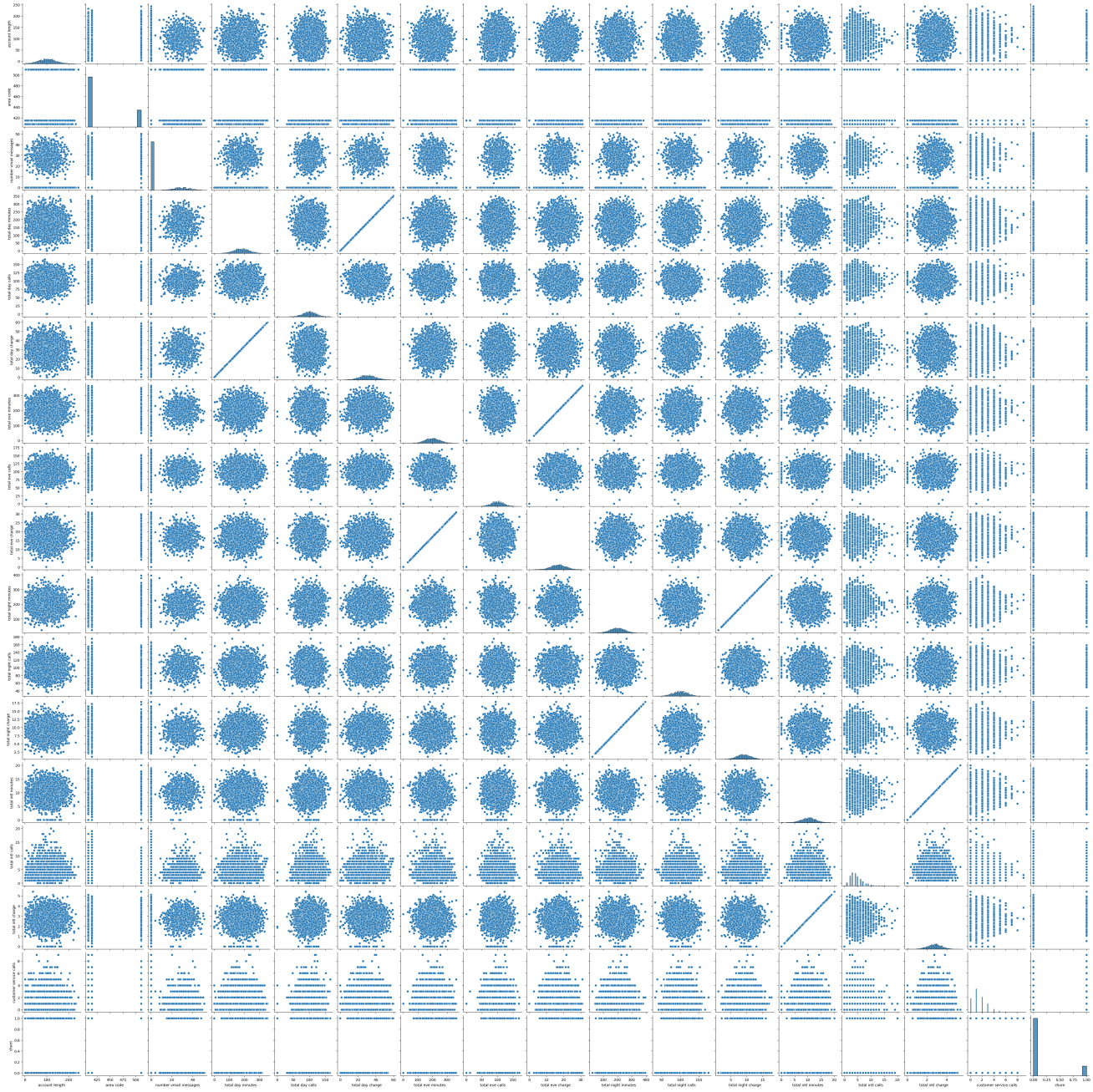
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   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
```

Data Cleaning

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7d1bd114a250>
```



Handling missings values

```
# Check for missing values
df.isnull().sum()
```

	0
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

```
# Remove customer number feature it is contact information on the client and adds

import plotly.express as px

if 'phone number' in df.columns:
    df = df.drop('phone number', axis=1)
    print("Phone number column dropped successfully.")
else:
    print("Phone number column not found in the DataFrame.")
```

```
Phone number column not found in the DataFrame.
```

Handling Duplicated values

```
# Check for duplicated row
df.duplicated().sum()
```

```
0
```

Handling NAs

```
# CHeck NAs
df.isna().mean()*100
```

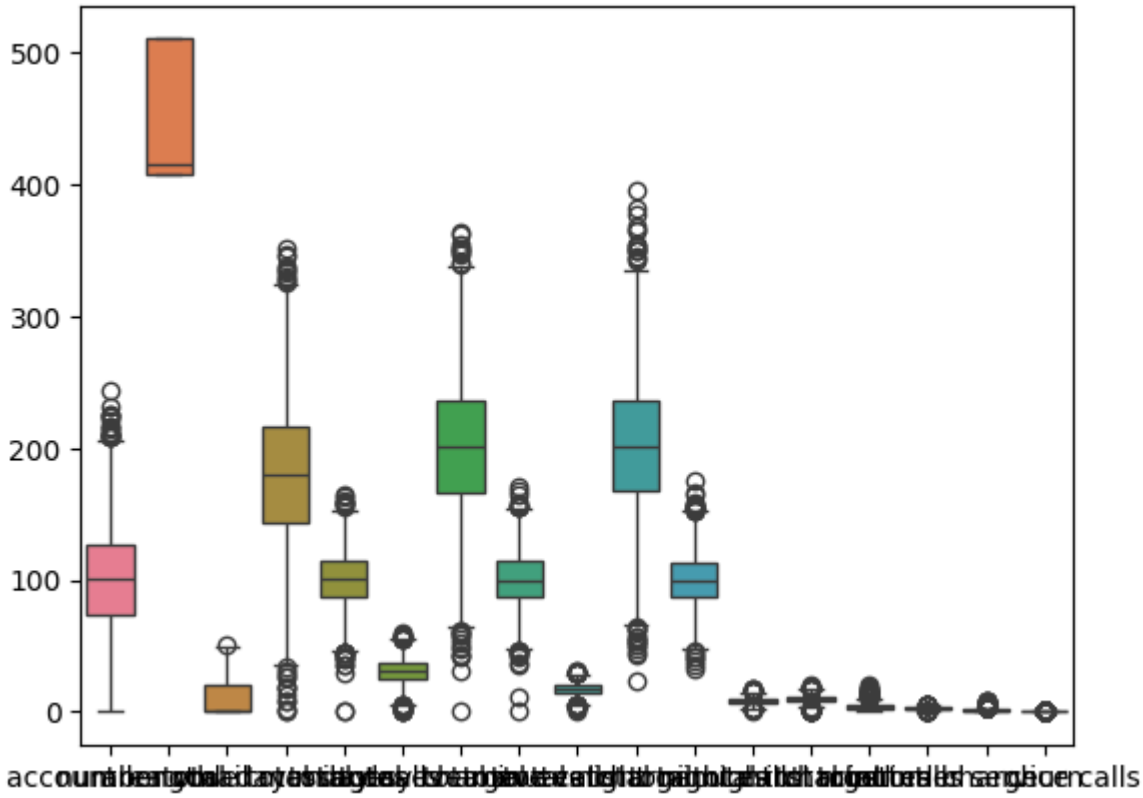
	0
state	0.0
account length	0.0
area code	0.0
international plan	0.0
voice mail plan	0.0
number vmail messages	0.0
total day minutes	0.0
total day calls	0.0
total day charge	0.0
total eve minutes	0.0
total eve calls	0.0
total eve charge	0.0
total night minutes	0.0
total night calls	0.0
total night charge	0.0
total intl minutes	0.0
total intl calls	0.0
total intl charge	0.0
customer service calls	0.0
churn	0.0

dtype: float64

Other cleaning steps

```
  # OUtliers

  # Feature engineering

sns.boxplot(df)
```

```
<Axes: >
```



EDA

```
# Check the number of unique values to determine feature types.

df.nunique()
```

	0
state	51
account length	212
area code	3
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

Feature Types

Continuous features are numeric values with an infinite number of possible values Categorical features are values that have a finite number of categories/groups

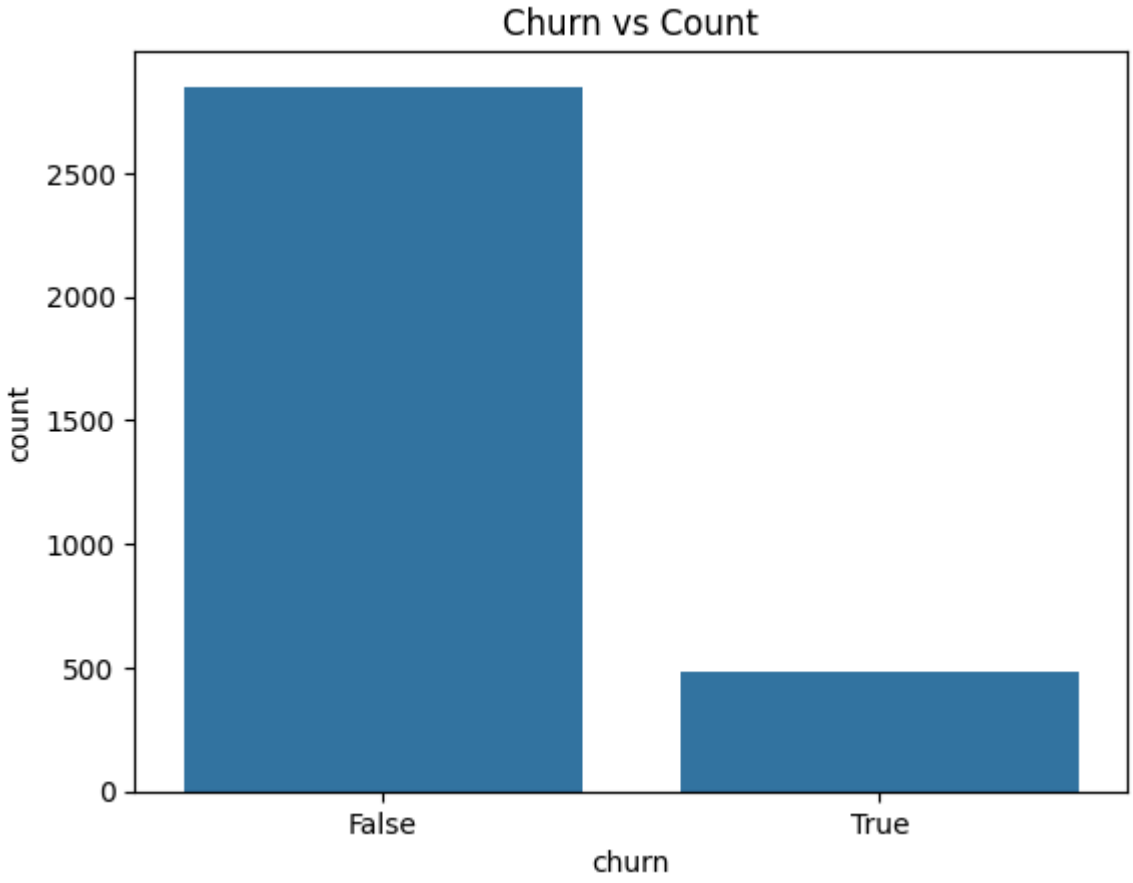
```
# Create numeric & categorical lists
numeric_cols = ['account length','number vmail messages','total day minutes','tot
                'total eve minutes','total eve calls','total eve charge','total n
                'total night charge','total intl minutes','total intl calls','tot
categoric_cols = ['state','area code','international plan','voice mail plan']
```

Analysis on Churn feature

Churn will be used as the dependent variable in this analysis. Churn indicates if a customer has terminated their contract with SyriaTel. True indicates they have terminated and false indicates they have not and have and have an existing account.

```
# Countplot of churn feature

sns.countplot(data=df, x='churn')
plt.title('Churn vs Count')
print(df.churn.value_counts())
```

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

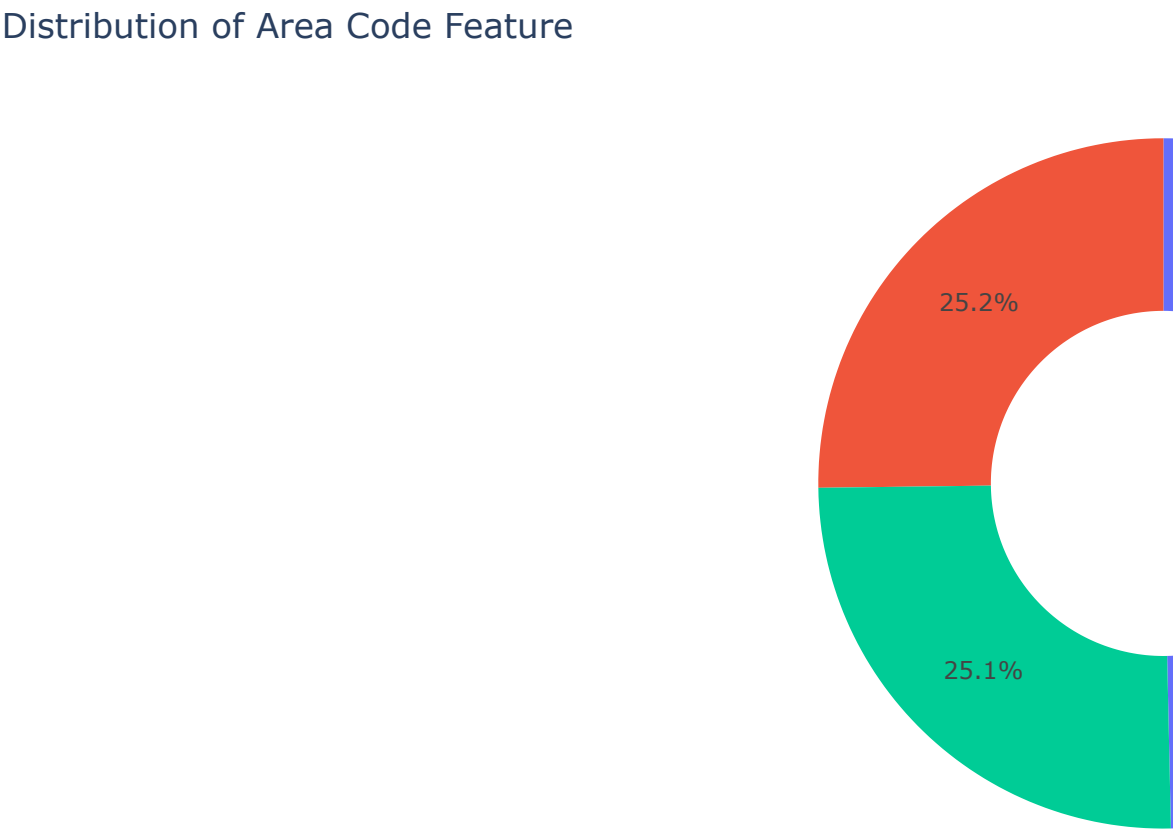


- Of the 3,333 customers in the dataset, 483 have terminated their contract with SyriaTel. That is 14.5% of customers lost.
- The distribution of the binary classes shows a data imbalance. This needs to be addressed before modeling as an unbalanced feature can cause the model to make false predictions.

Analysis on 'area code'

```
# Pie chart of area code feature
```

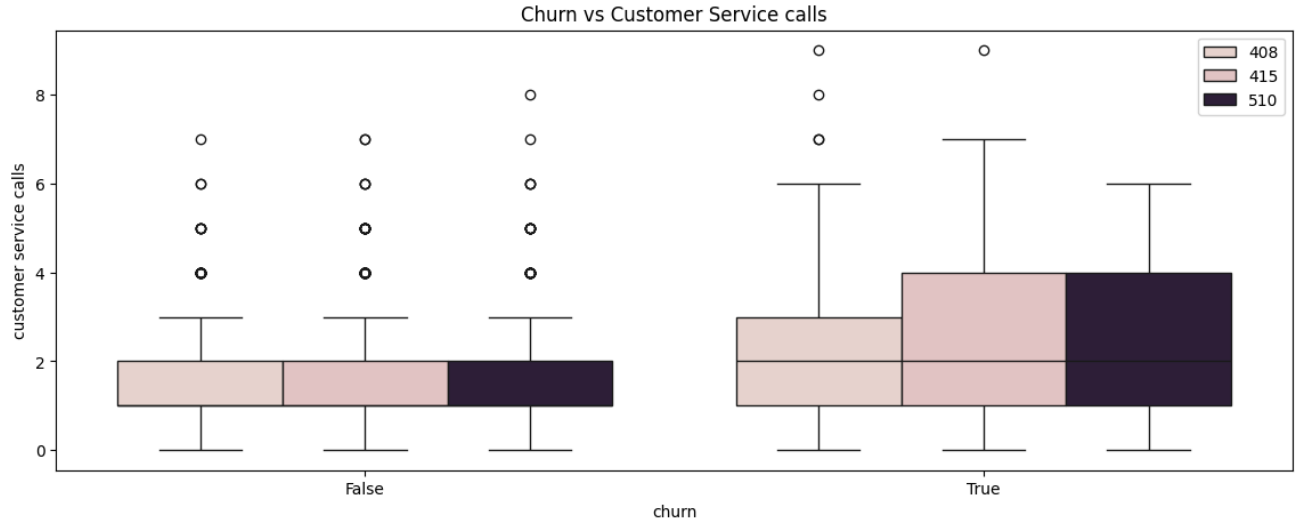
```
area = df['area code'].value_counts()
transuction = area.index
quantity = area.values

# draw pie circule with plotly
figure = px.pie(df,
                values = quantity,
                names = transuction,
                hole = .5,
                title = 'Distribution of Area Code Feature')
figure.show()
```



- Half of the customers have the area code 415.
- One fourth of customers have the area code 510
- Another fourth have the area code 408.

```
# Boxplot to see which area code has the highest churn
plt.figure(figsize=(14,5))
sns.boxplot(data=df, x='churn', y='customer service calls', hue='area code')
plt.title('Churn vs Customer Service calls')
plt.legend(loc='upper right')
```

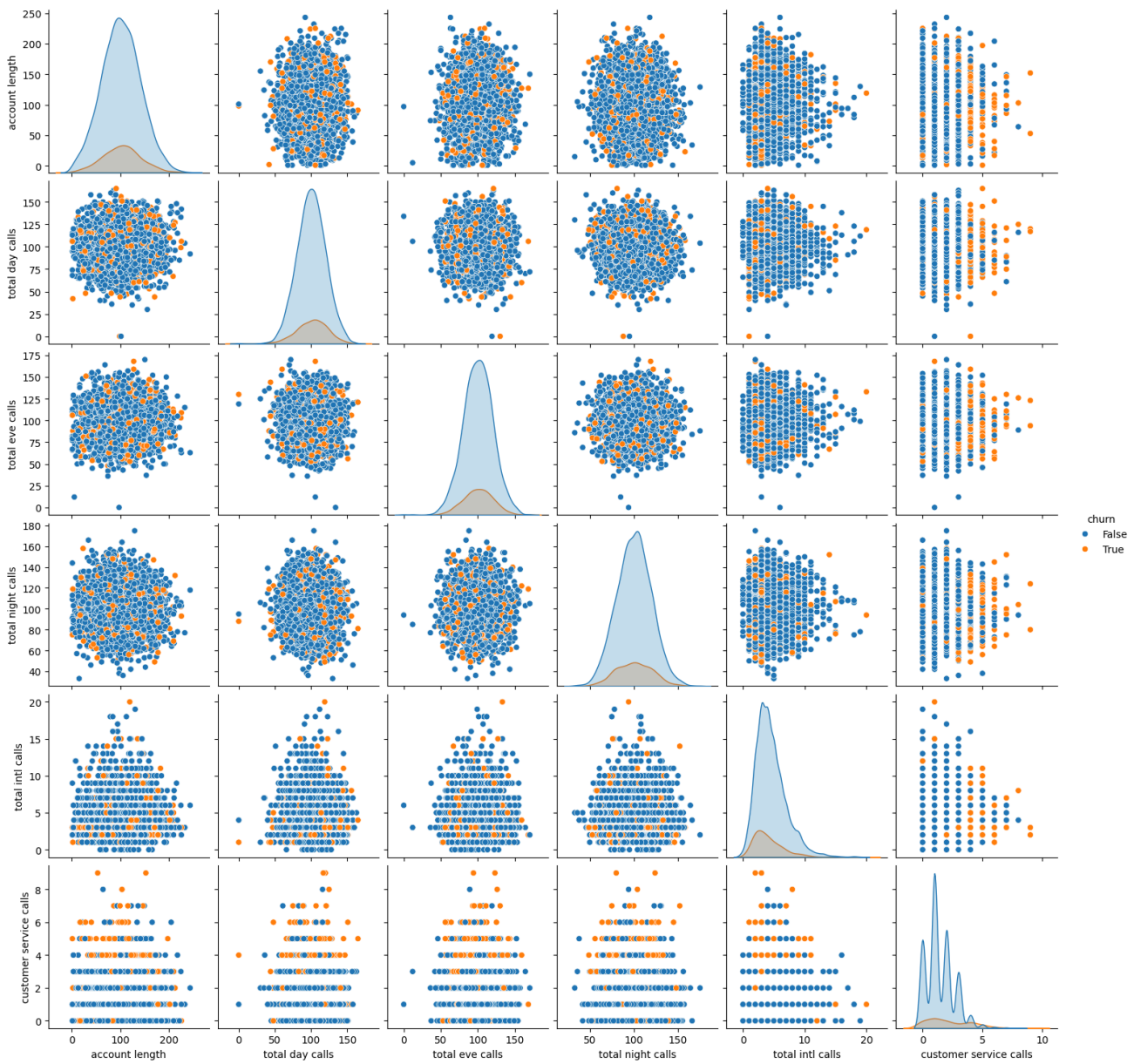
```
<matplotlib.legend.Legend at 0x7d1bc360dd90>
```



- There are outliers, in all area codes, amongst the customers who have not terminated their accounts.
- Of the customers who have terminated their account, they more likely have a 415 or a 510 area code

Pairplots for Numeric Features (Hue as "Churn")¶

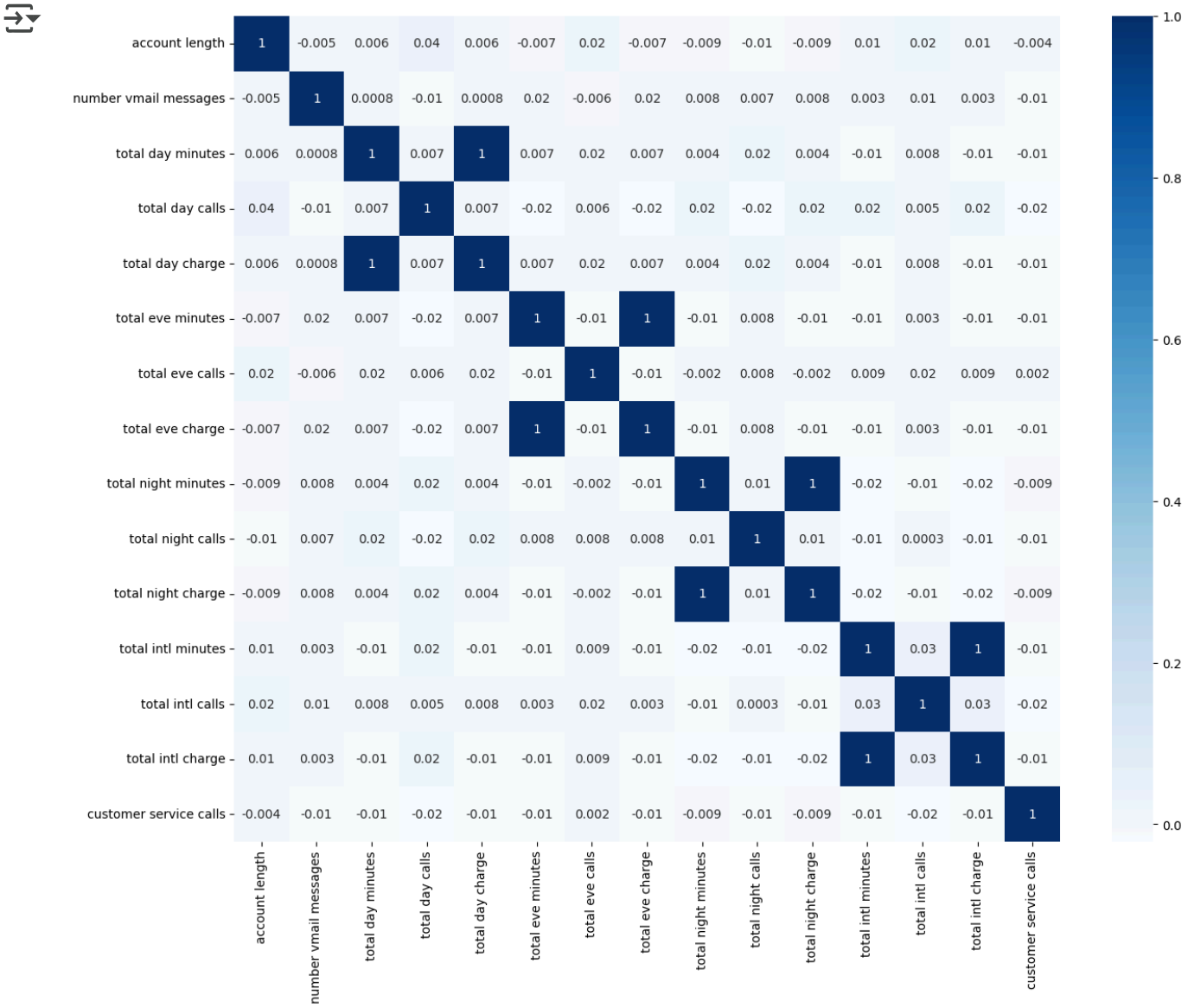
```
data_temp = df[["account length","total day calls","total eve calls","total night
sns.pairplot(data_temp, hue="churn",height=2.5);
plt.show()
```


account length
total day calls
total eve calls
total night calls
total intl calls
customer service calls
churn
False
True

- There seems to be a evident relationship between customer service calls and true churn values.
- After 4 calls, customers are a lot more likely to discontinue their service.

Correlation Heatmap for Numeric Features¶

```
corr_mat = df[numeric_cols].corr()
mask = np.triu(np.ones_like(corr_mat, dtype=bool))
plt.subplots(figsize=(15,12))
sns.heatmap(corr_mat, annot=True, cmap='Blues', square=True,fmt='.0g');
plt.xticks(rotation=90);
plt.yticks(rotation=0);
```


account length 1 -0.005 0.006 0.04 0.006 -0.007 0.02 -0.007 -0.009 -0.01 -0.009 0.01 0.02 0.01 -0.004
number vmail messages -0.005 1 0.0008 -0.01 0.0008 0.02 -0.006 0.02 0.008 0.007 0.008 0.003 0.01 0.003 -0.01
total day minutes 0.006 0.0008 1 0.007 1 0.007 0.02 0.007 0.004 0.02 0.004 -0.01 0.008 -0.01 -0.01
total day calls 0.04 -0.01 0.007 1 0.007 -0.02 0.006 -0.02 0.02 -0.02 0.02 0.02 0.005 0.02 -0.02
total day charge 0.006 0.0008 1 0.007 1 0.007 0.02 0.007 0.004 0.02 0.004 -0.01 0.008 -0.01 -0.01
total eve minutes -0.007 0.02 0.007 -0.02 0.007 1 -0.01 1 -0.01 0.008 -0.01 -0.01 0.003 -0.01 -0.01
total eve calls 0.02 -0.006 0.02 0.006 0.02 -0.01 1 -0.01 -0.002 0.008 -0.002 0.009 0.02 0.009 0.002
total eve charge -0.007 0.02 0.007 -0.02 0.007 1 -0.01 1 -0.01 0.008 -0.01 -0.01 0.003 -0.01 -0.01
total night minutes -0.009 0.008 0.004 0.02 0.004 -0.01 -0.002 -0.01 1 0.01 1 -0.02 -0.01 -0.02 -0.009
total night calls -0.01 0.007 0.02 -0.02 0.02 0.008 0.008 0.008 0.01 1 0.01 -0.01 0.0003 -0.01 -0.01
total night charge -0.009 0.008 0.004 0.02 0.004 -0.01 -0.002 -0.01 1 0.01 1 -0.02 -0.01 -0.02 -0.009
total intl minutes 0.01 0.003 -0.01 0.02 -0.01 -0.01 0.009 -0.01 -0.02 -0.01 -0.02 1 0.03 1 -0.01
total intl calls 0.02 0.01 0.008 0.005 0.008 0.003 0.02 0.003 -0.01 0.0003 -0.01 0.03 1 0.03 -0.02
total intl charge 0.01 0.003 -0.01 0.02 -0.01 -0.01 0.009 -0.01 -0.02 -0.01 -0.02 1 0.03 1 -0.01
customer service calls -0.004 -0.01 -0.01 -0.02 -0.01 -0.01 0.002 -0.01 -0.009 -0.01 -0.009 -0.01 -0.02 -0.01 1
account length
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
1.0
0.8
0.6
0.4
0.2
0.0

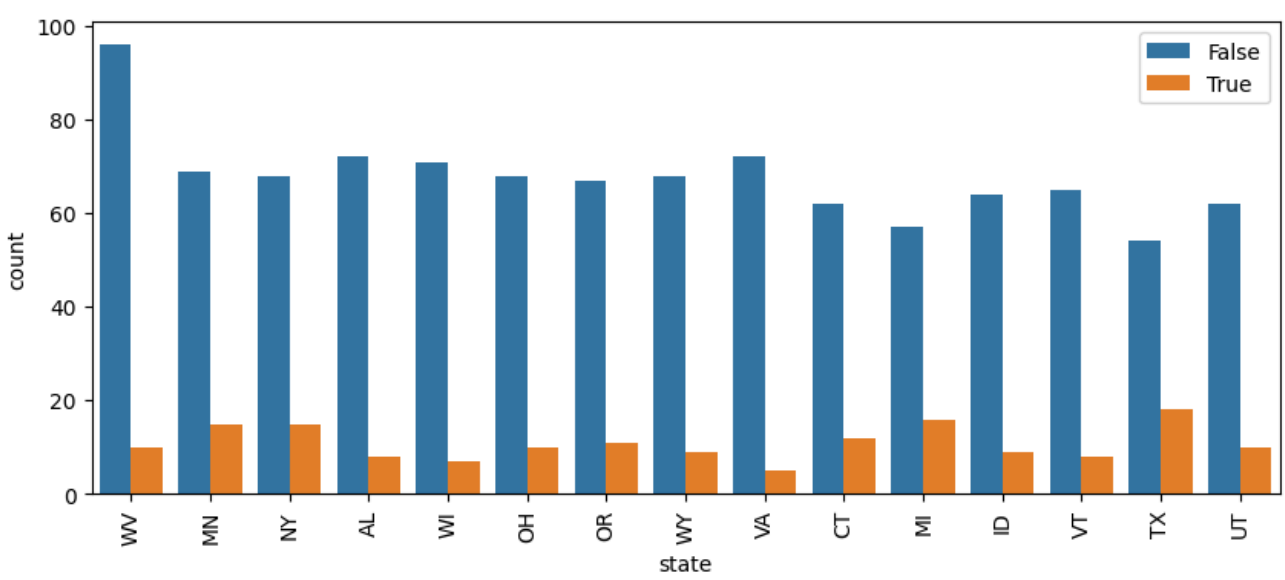
Most of the features are not correlated however some do share a perfect correlation.

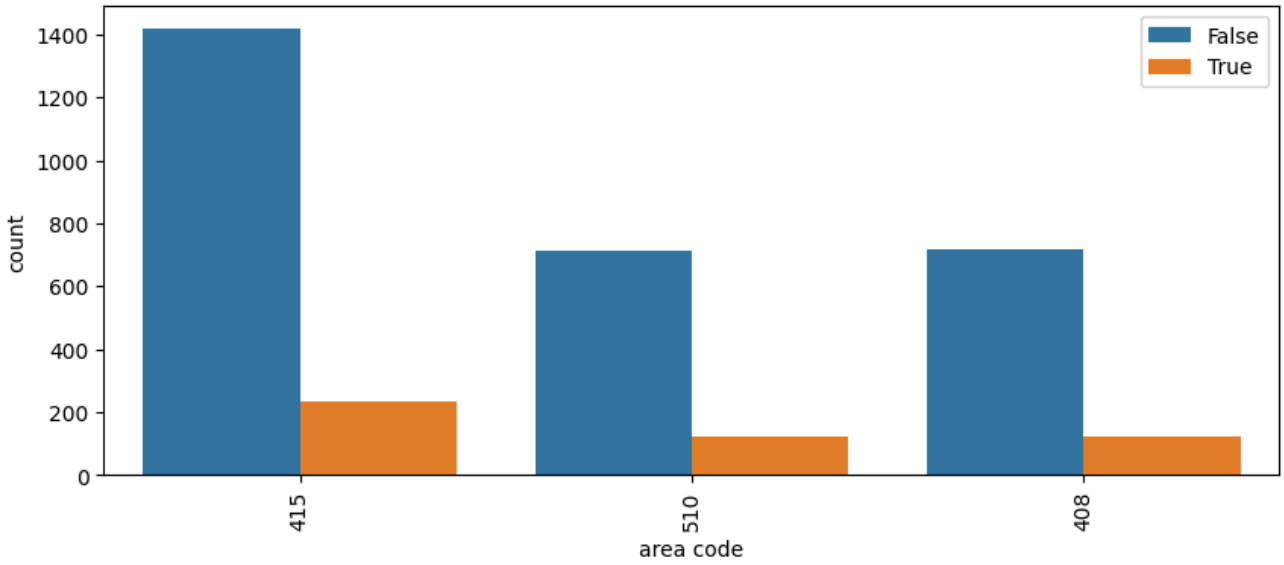
- Total day charge and total day minutes features are fully positively correlated.
- Total eve charge and total eve minutes features are fully positively correlated.
- Total night charge and total night minutes features are fully positively correlated.
- Total int charge and total int minutes features are fully positively correlated.

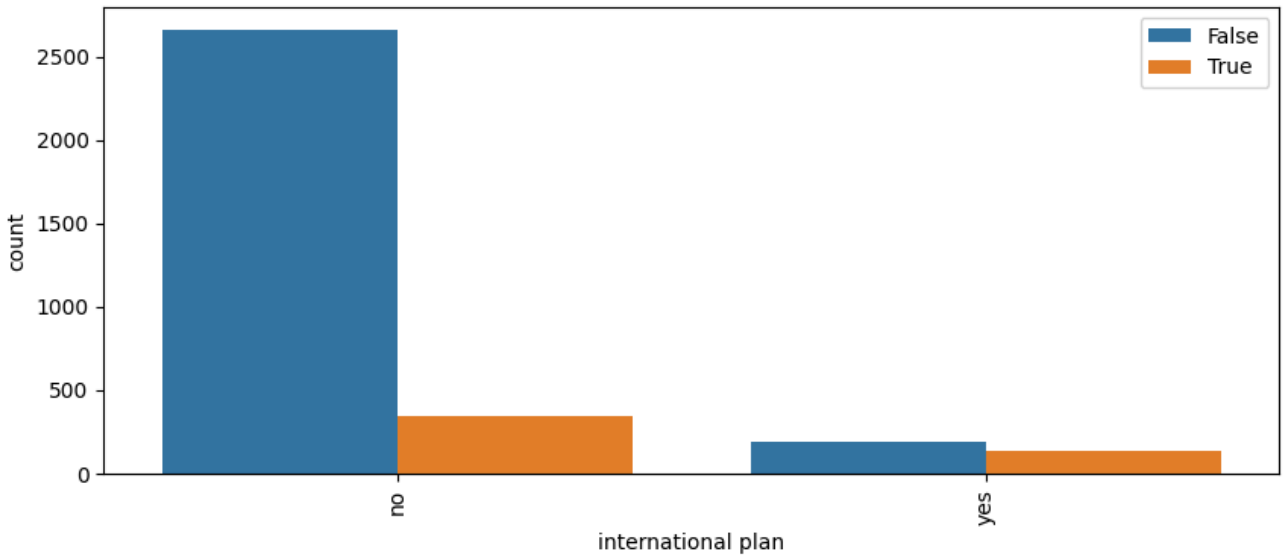
It makes sense for these features to be perfectly correlated because the charge is a direct result of the minutes used. The perfect correlation of 1 indicates the presence of perfect multicollinearity. It does not have the same impact on nonlinear models as it does on linear models. Some nonlinear models are impacted by perfect multicollinearity whereas others are not.

Categorical Feature Analysis

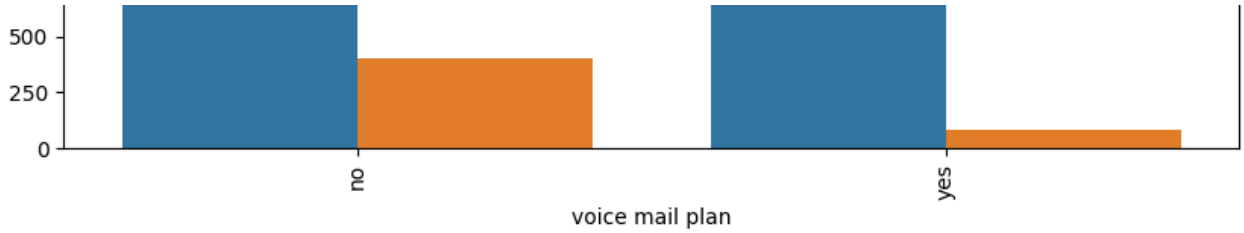
```
for i in categoric_cols:
    plt.figure(figsize=(10,4))
    sns.countplot(x=i, hue="churn", data=df,order= df[i].value_counts().iloc[0:15
    plt.xticks(rotation=90)
    plt.legend(loc="upper right")
    plt.show()
```


False
True
count
state
WV
MN
NY
AL
WI
OH
OR
WY
VA
CT
MI
ID
VT
TX
UT


False
True
count
area code
415
510
408


False
True
count
international plan
no
yes


False
True
count



Outlier Detection and Treatment

- Dropping outliers past 3 standard deviations.

```
from scipy import stats

print("Before dropping numerical outliers, length of the dataframe is: ",len(df))
def drop_numerical_outliers(df, z_thresh=3):
    constrains = df.select_dtypes(include=[np.number]).apply(lambda x: np.abs(sta
        .all(axis=1)
    df.drop(df.index[~constrains], inplace=True)

drop_numerical_outliers(df)
print("After dropping numerical outliers, length of the dataframe is: ",len(df))
```

```
Before dropping numerical outliers, length of the dataframe is:  3333
After dropping numerical outliers, length of the dataframe is:  3169
```

Dropping Highly-Correlated Features

- Dropping features that have a correlation of 0.9 or above.

```
print("The original dataframe has {} columns.".format(df.shape[1]))

# Identify highly correlated features
corr_matrix = df[numeric_cols].corr().abs()
```

```
upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astyp
to_drop = [column for column in upper_triangle.columns if any(upper_triangle[colu

# Create a new DataFrame without the highly correlated features
reduced_df = df.drop(to_drop, axis=1)
print("The reduced dataframe has {} columns.".format(reduced_df.shape[1]))
```

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

Transforming "Churn" Feature's Rows into 0s and 1s¶

```
reduced_df['churn'].value_counts()
```

churn	count
False	2727
True	442

dtype: int64

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

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes
0	KS	128	415	no	yes	25	265.1	110	197.4
1	OH	107	415	no	yes	26	161.6	123	195.5
2	NJ	137	415	no	no	0	243.4	114	121.2
3	OH	84	408	yes	no	0	299.4	71	61.9

Next steps: Generate code with reduced_df | View recommended plots | New interactive sheet

Preprocessing

Encoding

One-Hot Encoding

- Transforming categorical features into dummy variables as 0 and 1 to be able to use them in classification models.

```
dummy_df_state = pd.get_dummies(reduced_df["state"],dtype=np.int64,prefix="state_
dummy_df_area_code = pd.get_dummies(reduced_df["area code"],dtype=np.int64,prefix
dummy_df_international_plan = pd.get_dummies(reduced_df["international plan"],dty
dummy_df_voice_mail_plan = pd.get_dummies(reduced_df["voice mail plan"],dtype=np.
```

```
reduced_df = pd.concat([reduced_df,dummy_df_state,dummy_df_area_code,dummy_df_int
reduced_df = reduced_df.loc[:,~reduced_df.columns.duplicated()]
reduced_df = reduced_df.drop(['state','area code','international plan','voice mai

reduced_df.head()
```

	account length	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	t c
0	128	25	265.1	110	197.4	99	244.7	91	10.0	
1	107	26	161.6	123	195.5	103	254.4	103	13.7	
2	137	0	243.4	114	121.2	110	162.6	104	12.2	
3	84	0	299.4	71	61.9	88	196.9	89	6.6	
4	75	0	166.7	113	148.3	122	186.9	121	10.1	

5 rows × 68 columns

Scaling

Scaling Numerical Features

- Scaling is the process of transforming values of several variables into a similar range. Typical normalizations include scaling the variable so the variable average is 0, scaling the variable so the variable variance is 1, or scaling the variable so the variable values range from 0 to 1.
- In our example, Min-Max Normalization method is applied. MinMaxScaler is used to reduce the effects of outliers in the dataset. By applying the following method, standard deviation issues will be solved.
- MinMaxScaler is applied on the columns which is defined in "columns_to_be_scaled" variable below.

```
transformer = MinMaxScaler()

def scaling(columns):
    return transformer.fit_transform(reduced_df[columns].values.reshape(-1,1))
```

```
for i in reduced_df.select_dtypes(include=[np.number]).columns:
    reduced_df[i] = scaling(i)
reduced_df.head()
```

	account length	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	to i minu
0	0.587963	0.510204	0.773921	0.576271	0.490079	0.487179	0.643519	0.422414	0.487
1	0.490741	0.530612	0.450281	0.686441	0.483796	0.521368	0.675595	0.525862	0.713
2	0.629630	0.000000	0.706066	0.610169	0.238095	0.581197	0.372024	0.534483	0.621
3	0.384259	0.000000	0.881176	0.245763	0.041997	0.393162	0.485450	0.405172	0.280
4	0.342593	0.000000	0.466229	0.601695	0.327712	0.683761	0.452381	0.681034	0.493

5 rows × 68 columns

Classification

Logistic Regression

Logistic regression is a classification algorithm, used when the value of the target variable is categorical in nature. It is most commonly used when the data in question has binary output, so when it belongs to one class or another, or is either a 0 or 1. This method will be used to create a baseline mode

```
# 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


X=reduced_df.drop(['churn'],axis=1)
y=reduced_df['churn']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=



reduced_df.churn.value_counts()
```

churn	count
0.0	2727
1.0	442

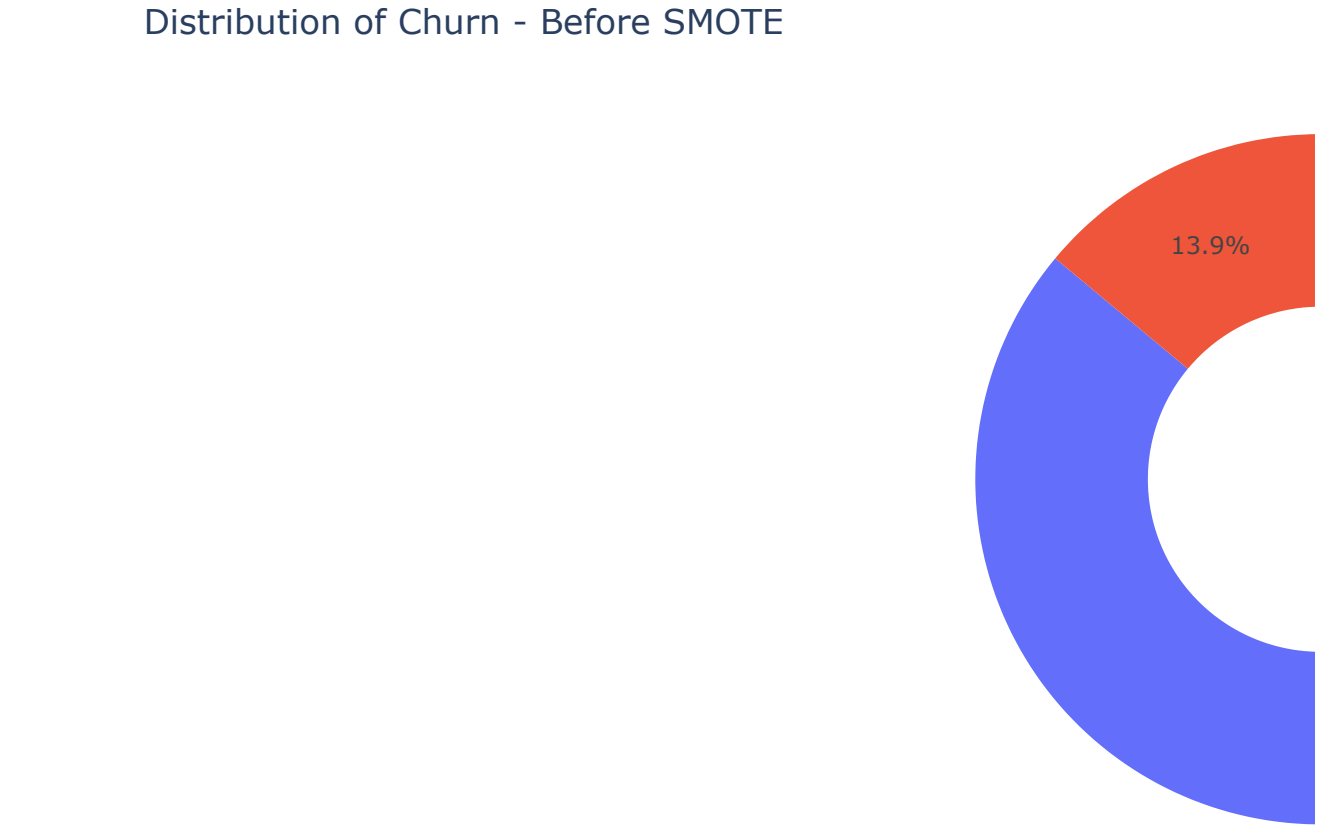
dtype: int64

```
y_train_over.value_counts()
```

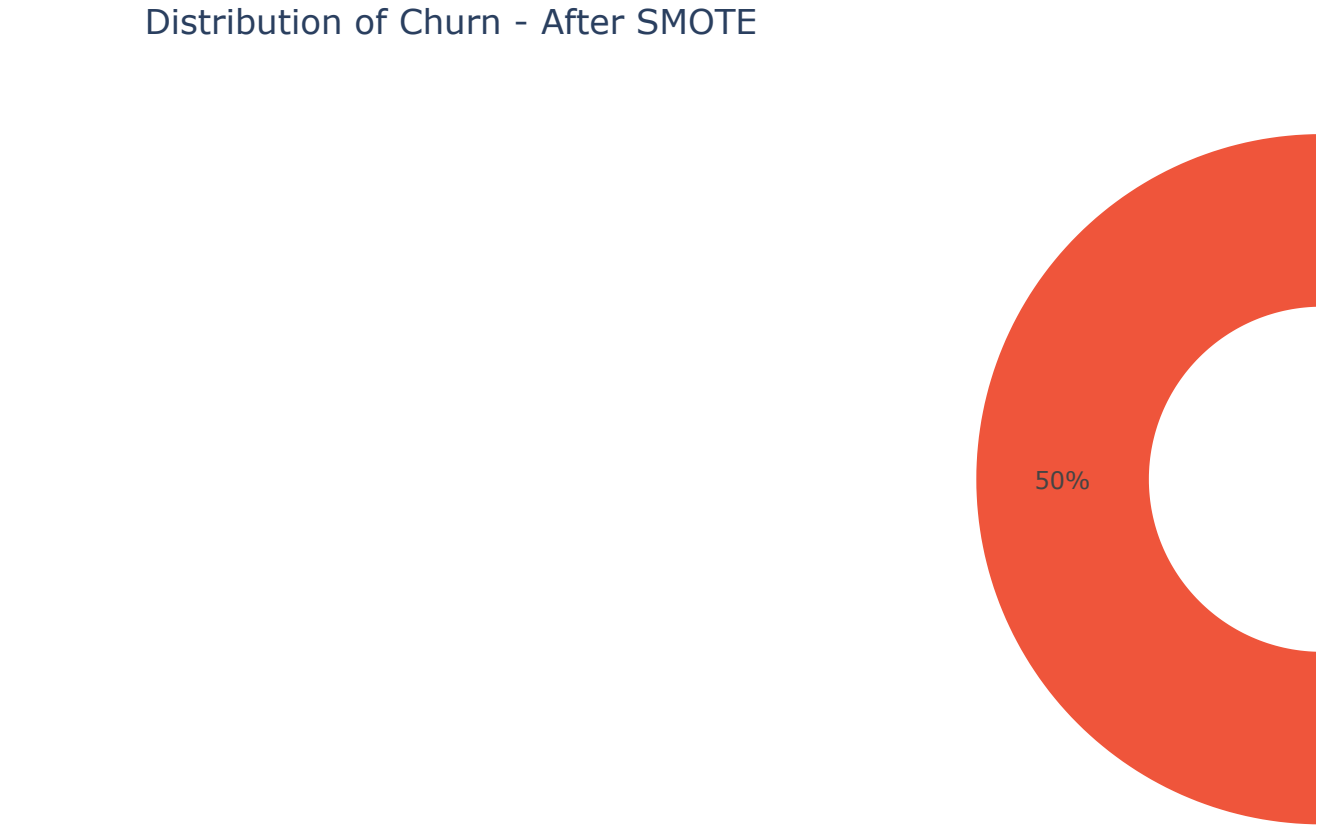
churn	count
0.0	2063
1.0	2063

dtype: int64

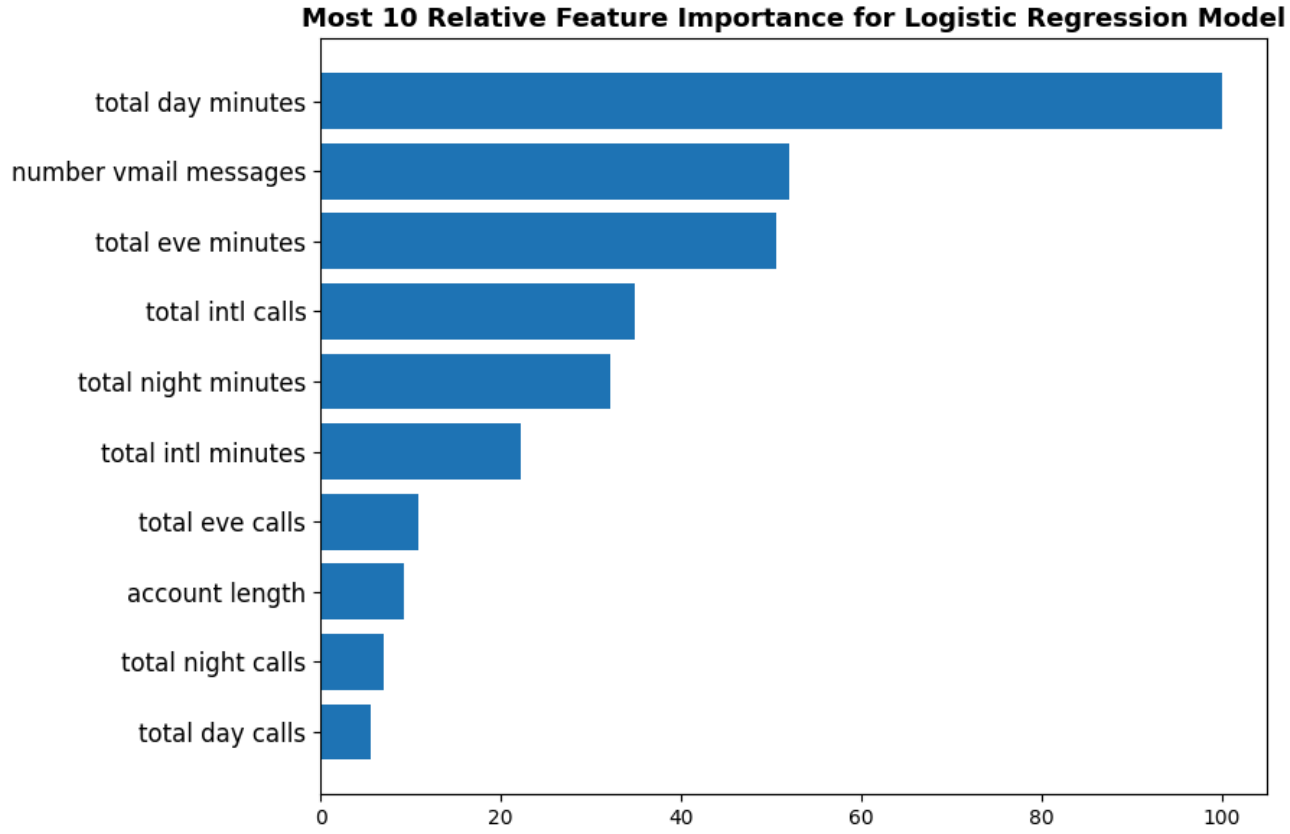
```
churn = reduced_df['churn'].value_counts()
transuction = churn.index
quantity = churn.values

# draw pie circule with plotly
figure = px.pie(y_train_over,
                values = quantity,
                names = transuction,
                hole = .5,
                title = 'Distribution of Churn - Before SMOTE')
figure.show()
```




```
y_train_over_df = y_train_over.to_frame()
churn = y_train_over_df['churn'].value_counts()
transuction = churn.index
quantity = churn.values

# draw pie circule with plotly
figure = px.pie(y_train_over_df,
                values = quantity,
                names = transuction,
                hole = .5,
                title = 'Distribution of Churn - After SMOTE')
figure.show()
```

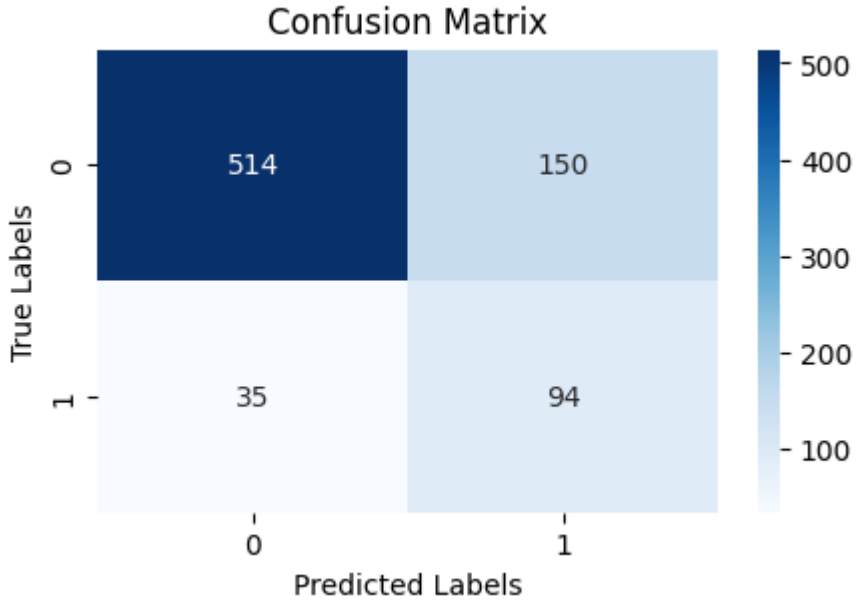



```
# Object creation, fitting the data & getting predictions
lr= LogisticRegression()
lr.fit(X_train_over,y_train_over)
y_pred_lr = lr.predict(X_test)


# Feature Importances
feature_importance = abs(lr.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())[0:10
sorted_idx = np.argsort(feature_importance)[0:10]
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure(figsize=(9, 6))
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
plt.title('Most 10 Relative Feature Importance for Logistic Regression Model', fo
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=12)

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




```
print("LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS")
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_lr),5
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_lr),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_lr),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_lr)
cm_lr = confusion_matrix(y_test, y_pred_lr)
f, ax= plt.subplots(1,1,figsize=(5,3))
sns.heatmap(cm_lr, annot=True, cmap='Blues', fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels'); ax.set_ylabel('True Labels') ; ax.set_title('C
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
print(classification_report(y_test, y_pred_lr, target_names=['0', '1']))
```

```
LOGISTIC REGRESSION CLASSIFIER MODEL RESULTS
Accuracy score for testing set:  0.76671
F1 score for testing set:  0.50402
Recall score for testing set:  0.72868
Precision score for testing set:  0.38525
```



	precision	recall	f1-score	support
0	0.94	0.77	0.85	664
1	0.39	0.73	0.50	129
accuracy			0.77	793
macro avg	0.66	0.75	0.68	793
weighted avg	0.85	0.77	0.79	793

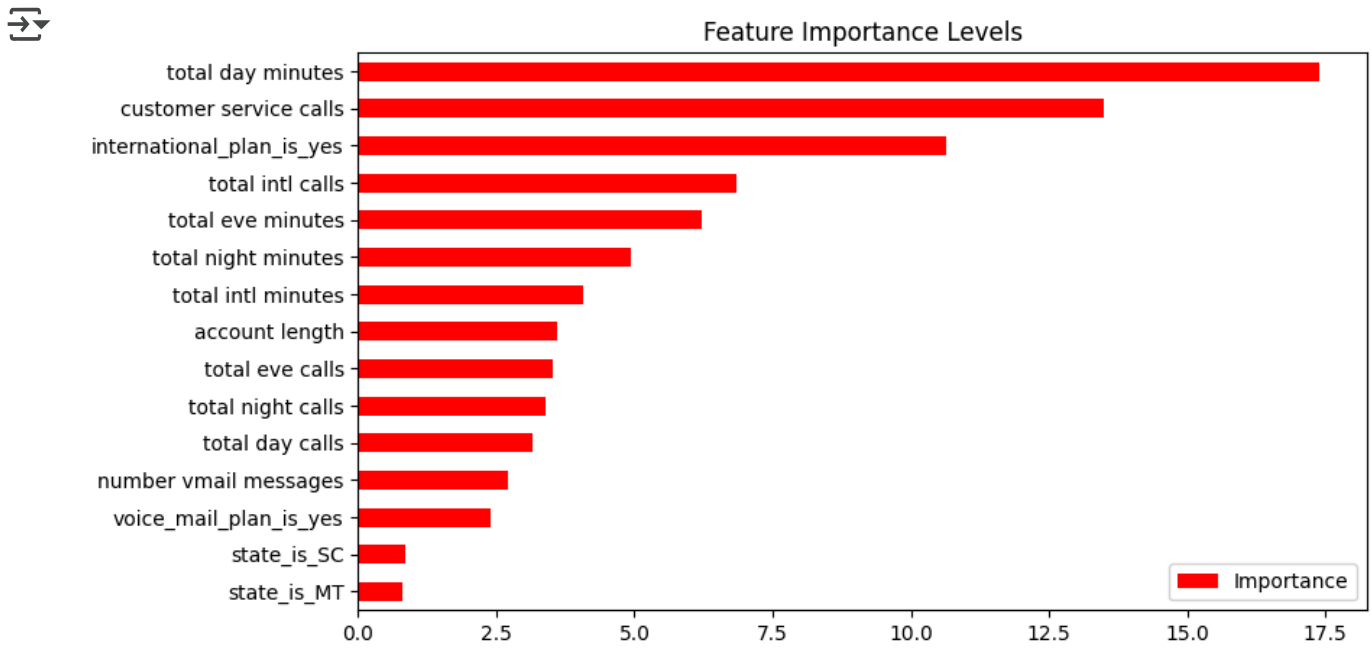
- According to the logistic regression classifier model, total day charge, number of voicemail messages and total evening charge are the top three important features.
- Model accuracy is 80%, which isn't bad. F1 score is only 50.% which means the test will only be accurate half the times it is ran.

Random Forest

- Random forest is an ensemble machine learning algorithm.
- Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

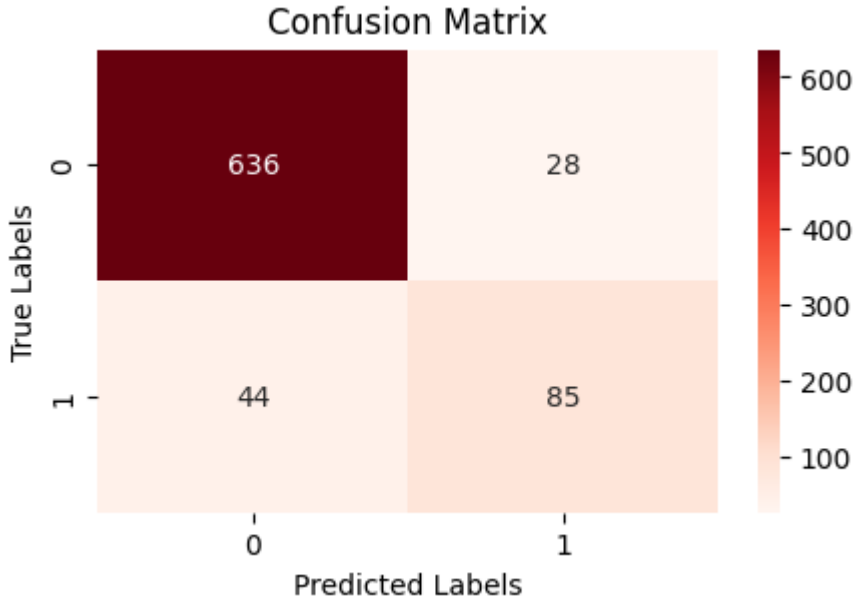
```
# Object creation, fitting the data & getting predictions
rf_model_final = RandomForestClassifier()
rf_model_final.fit(X_train_over,y_train_over)
y_pred_rf = rf_model_final.predict(X_test)
```

```
Importance =pd.DataFrame({"Importance": rf_model_final.feature_importances_*100},
Importance.sort_values(by = "Importance", axis = 0, ascending = True).tail(15).pl
plt.title("Feature Importance Levels");
plt.show()
```




```
print("RANDOM FOREST MODEL RESULTS")
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_rf),5
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_rf),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_rf),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_rf)
cm_rf = confusion_matrix(y_test, y_pred_rf)
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_title('C
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
plt.show();
print(classification_report(y_test, y_pred_rf, target_names=['0', '1']))
```

```
RANDOM FOREST MODEL RESULTS
Accuracy score for testing set:  0.90921
F1 score for testing set:  0.70248
Recall score for testing set:  0.65891
Precision score for testing set:  0.75221
```



	precision	recall	f1-score	support
0	0.94	0.96	0.95	664
1	0.75	0.66	0.70	129
accuracy			0.91	793
macro avg	0.84	0.81	0.82	793
weighted avg	0.91	0.91	0.91	793

- According to the random forest classifier, total day charge, customer service calles and "international plan is yes" features have the highest impact on the model.
- Accuracy and F1 score are much higher for this model, which is good news.

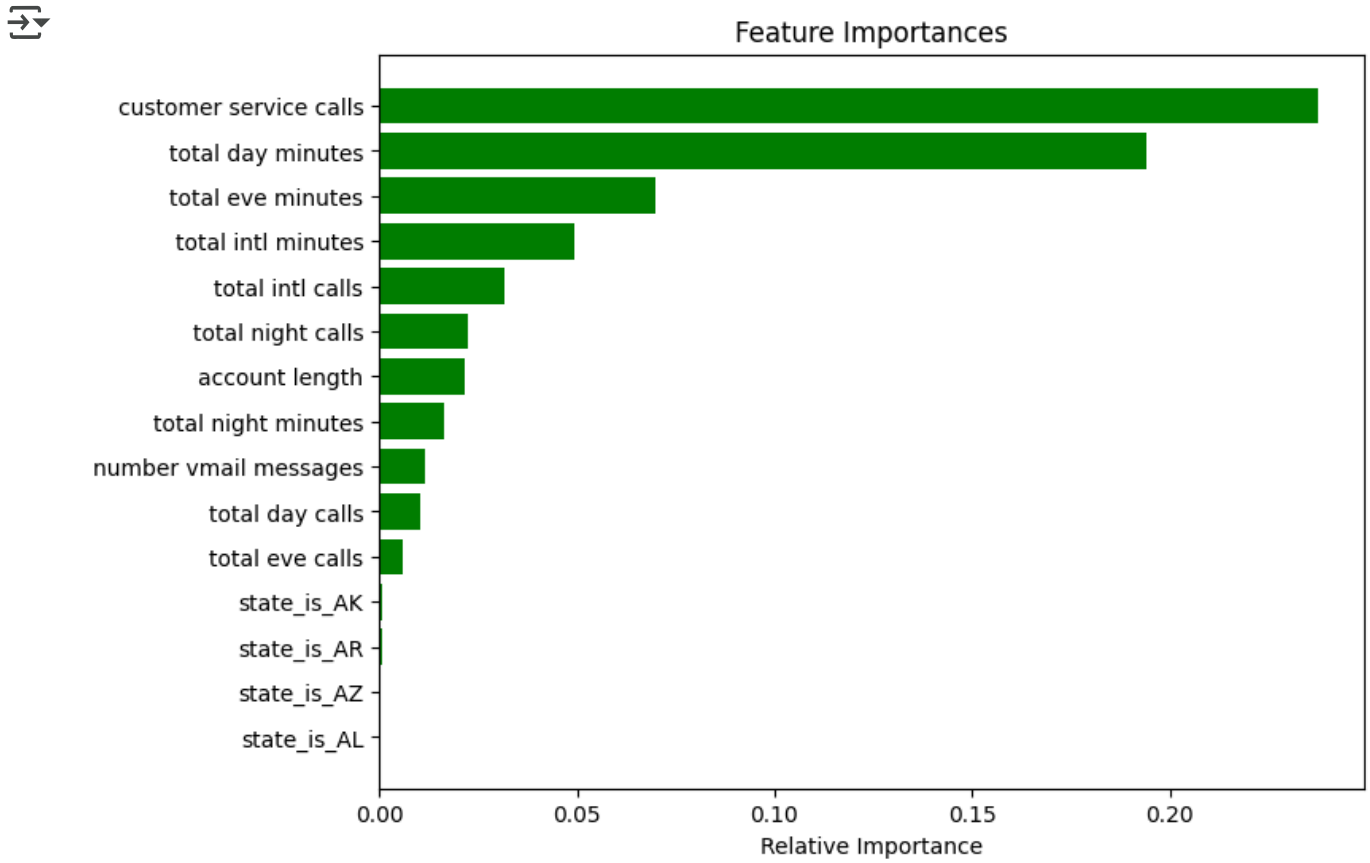
Decision Tree

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

```
# Object creation, fitting the data & getting predictions
decision_tree = DecisionTreeClassifier()
```

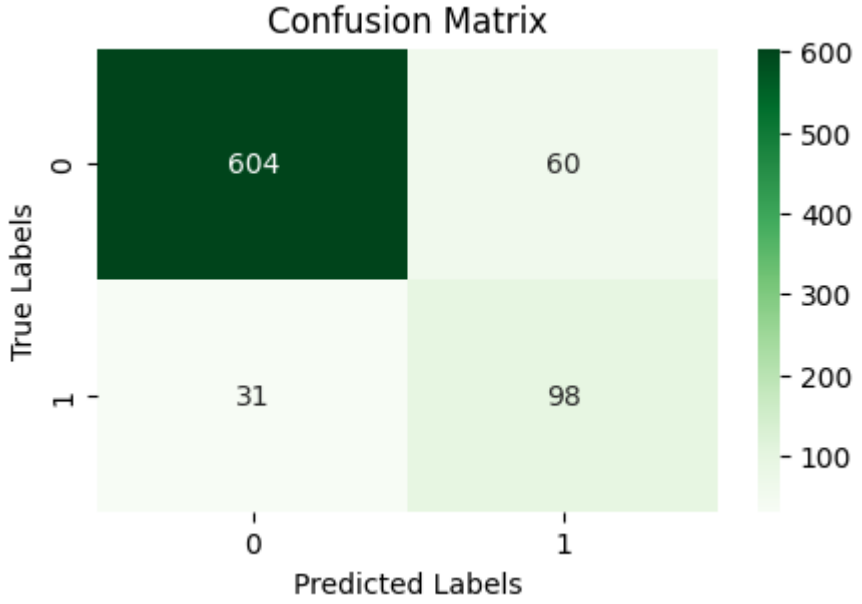
```
decision_tree.fit(X_train_over,y_train_over)
y_pred_dt = decision_tree.predict(X_test)


feature_names = list(X_train_over.columns)
importances = decision_tree.feature_importances_[0:15]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='green', align='center'
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```




```
print("DECISION TREE CLASSIFIER MODEL RESULTS")
print('Accuracy score for testing set: ',round(accuracy_score(y_test,y_pred_dt),5
print('F1 score for testing set: ',round(f1_score(y_test,y_pred_dt),5))
print('Recall score for testing set: ',round(recall_score(y_test,y_pred_dt),5))
print('Precision score for testing set: ',round(precision_score(y_test,y_pred_dt)
cm_dt = confusion_matrix(y_test, y_pred_dt)
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_title('C
ax.xaxis.set_ticklabels(['0', '1']) ; ax.yaxis.set_ticklabels(['0', '1'])
```

```
plt.show();
print(classification_report(y_test, y_pred_dt, target_names=['0', '1']))
```

```
DECISION TREE CLASSIFIER MODEL RESULTS
Accuracy score for testing set:  0.88525
F1 score for testing set:  0.68293
Recall score for testing set:  0.75969
Precision score for testing set:  0.62025
```



	precision	recall	f1-score	support
0	0.95	0.91	0.93	664
1	0.62	0.76	0.68	129
accuracy			0.89	793
macro avg	0.79	0.83	0.81	793
weighted avg	0.90	0.89	0.89	793

According to the decision tree classifier, customer service calls total day charge and total evening charge are the three most important for the model. The accuracy and F1 score for this model is not as great as model 2.

The best model is Random Forest because it had accuracy = 98% & recall of 93%. Better than all the other models

Hyperparameter tuning

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
```

```
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize RandomizedSearchCV or GridSearchCV
# rf_random = RandomizedSearchCV(estimator=rf_model, param_distributions=param_g
rf_grid = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, verbose=
```