Customer Churn Project

Project Overview

This project focuses on predicting customer churn in the telecom industry using a real-world dataset. By applying machine learning techniques such as logistic regression and decision trees, the aim is to identify patterns that lead to customer loss. The analysis follows the CRISP-DM methodology and includes data cleaning, exploratory analysis, model building, evaluation, and business recommendations. The final deliverables include a tableau dashboard and a stakeholder-friendly presentation.

CRISP-DM Methodology

1. Business Understanding

1.1 Business Problem

Customer churn is a key issue in the telecommunications industry, directly affecting profitability. Losing customers often costs more than acquiring new ones, so being able to predict which customers are likely to churn allows businesses to proactively intervene with retention strategies.

1.2 Key Questions

a. What proportion of customers churned vs stayed?

b. Which services are most associated with customer

c. How does customer service call frequency relate to Churn?

1.3 Objectives of ths project

Predict whether a custmer wll churn or not.

Build and evaluate two models: a baseline model and an improved one.

Recommend the best model based on performance.

Provide actionable business insights to reduce churn.

2. Data Understanding

2.1 Load data

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
```

```
# load the dataset
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
# check the first few rows of the dataset
df.head()
```

	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

2.2 Describe data

```
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
```

```
# shape of the dataset
df.shape
```

```
(3333, 21)
```

The above results shows that the data set has 3333 rows and 21 clumns

```
# descriptive statistics of the dataset
df.describe()
```

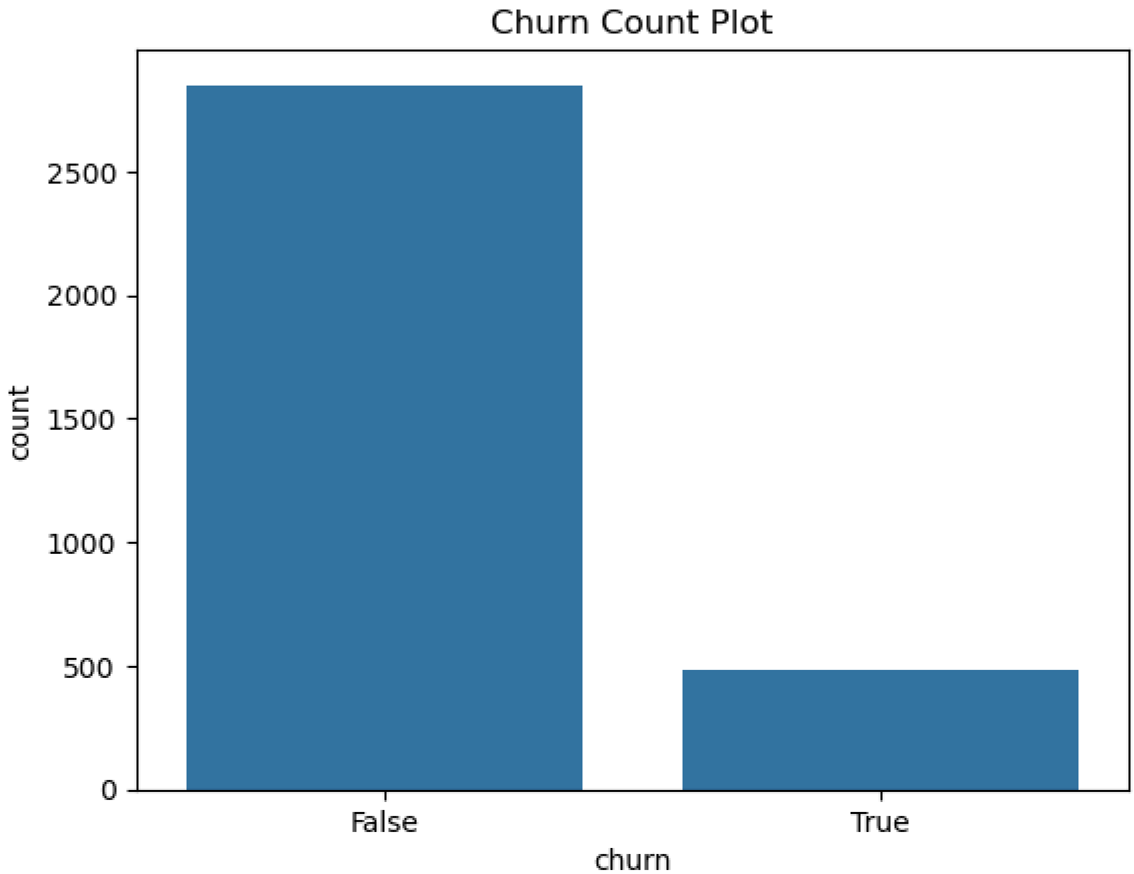
	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tota mi
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.9
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.7
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7

```
# column names of the dataset
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')
```

2.3 Data Understanding Visualizations

```
# Churn Count Plot
sns.countplot(x='churn', data=df)
plt.title('Churn Count Plot')
plt.show()
```



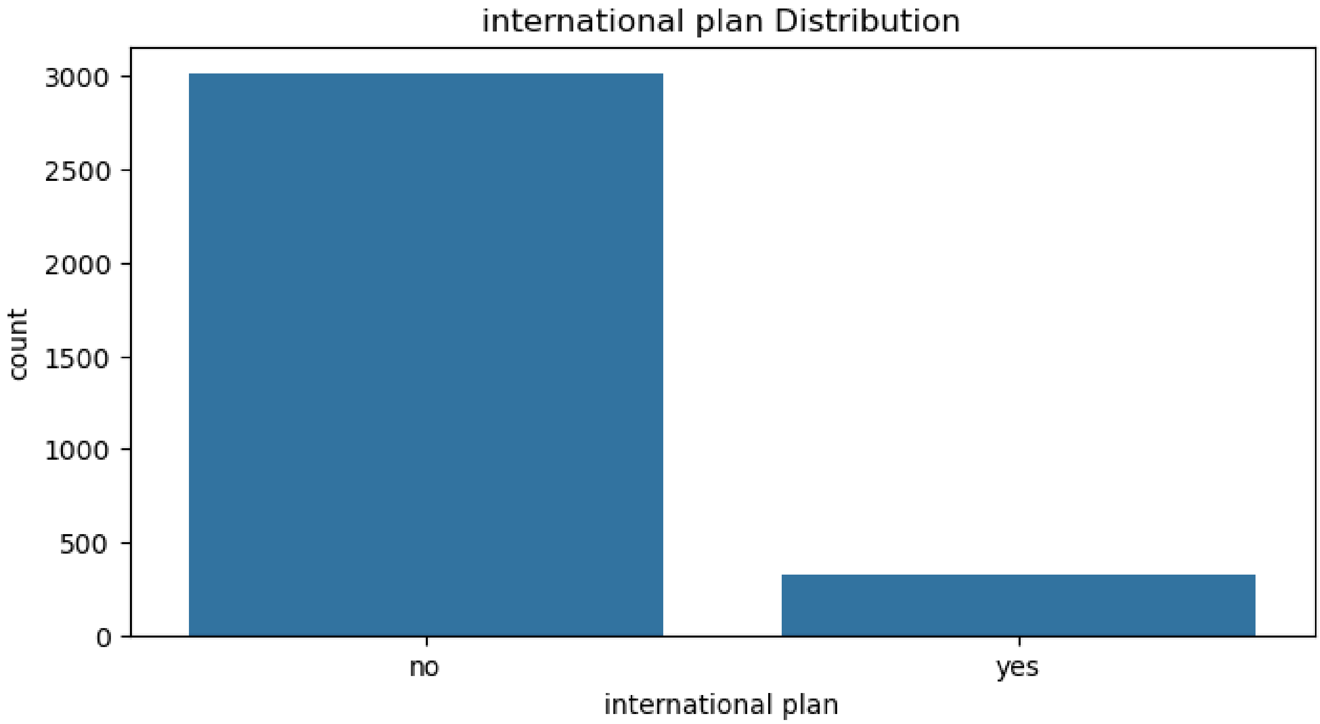
What does the visualization above mean

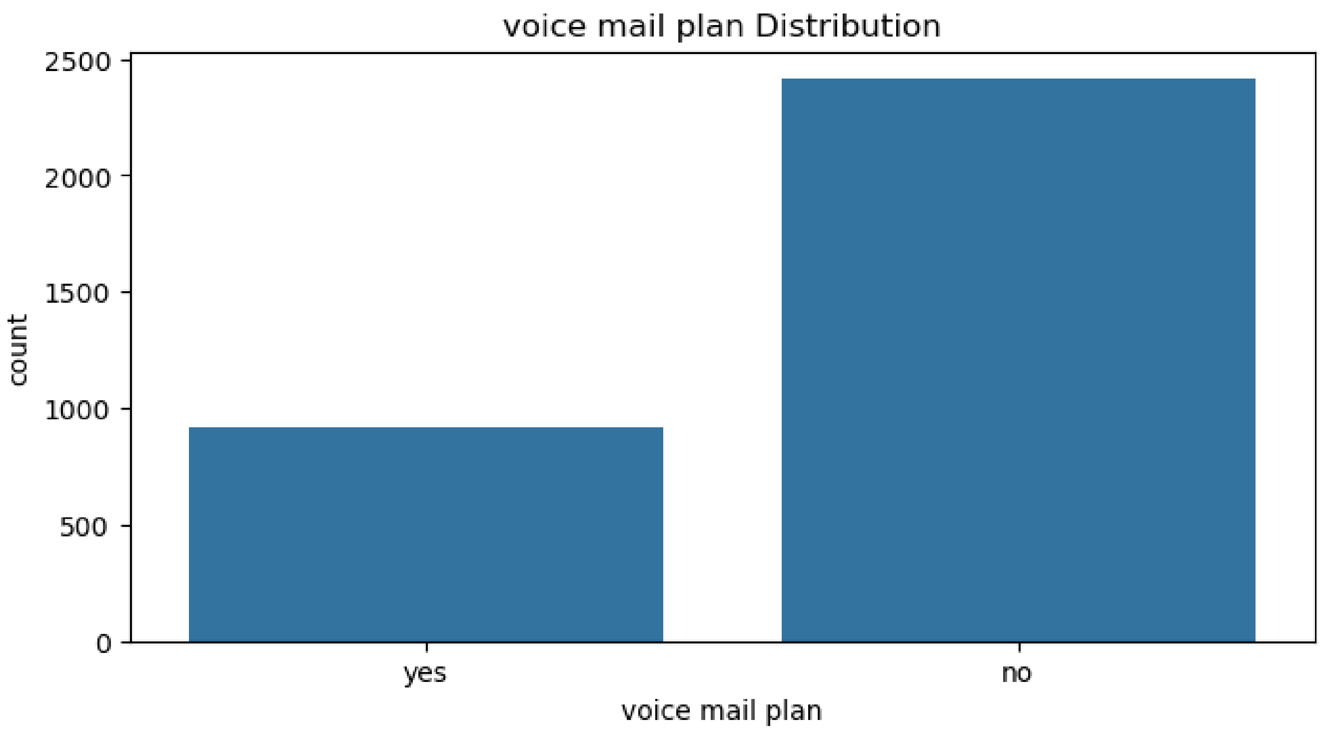
The company retains most customers

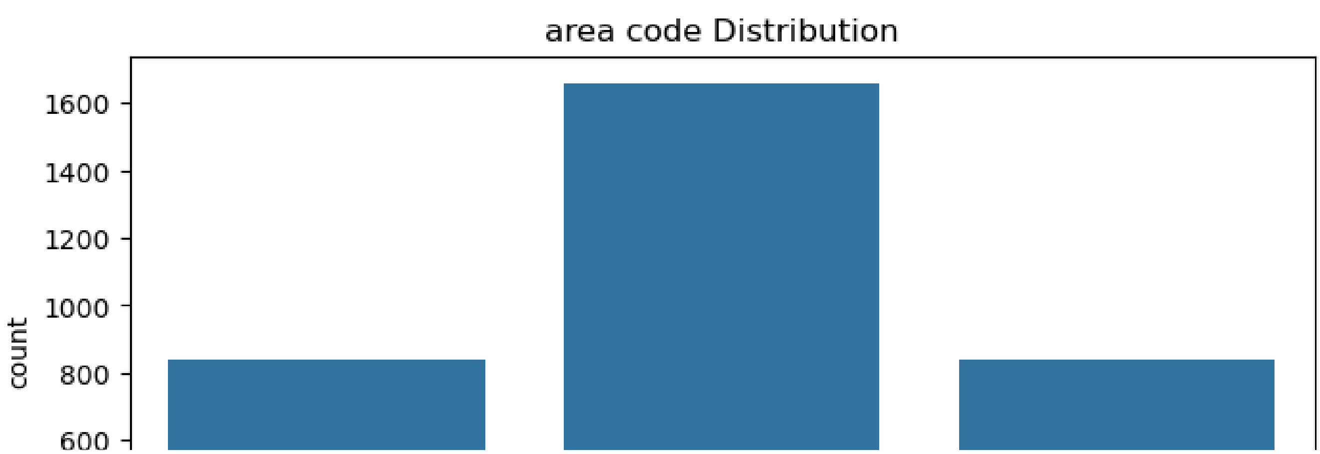
Churned customers(Churn = True) are a minority class.

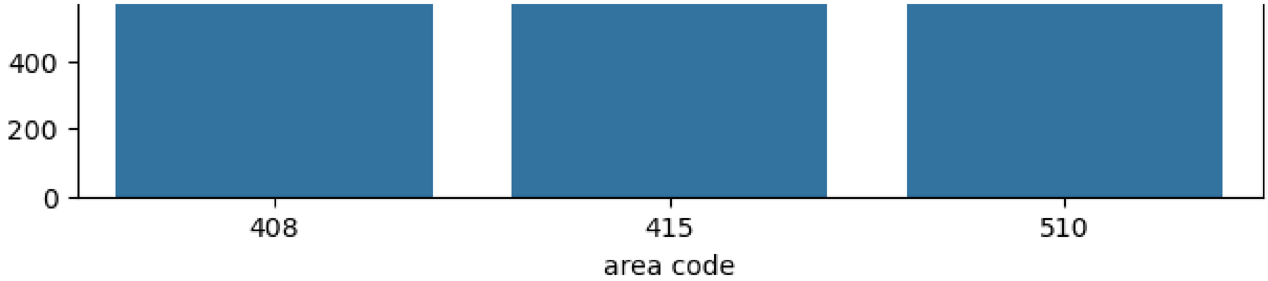
The dataset s mbalanced, and the models you train may be biased towards predicting the majority class(False) unless you handle this imbalance,.

```
# bar plot of categorical feature distribution
categorical_features = ['international plan', 'voice mail plan', 'area code']
for col in categorical_features:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=col, data=df)
    plt.title(f'{col} Distribution')
    plt.show()
```

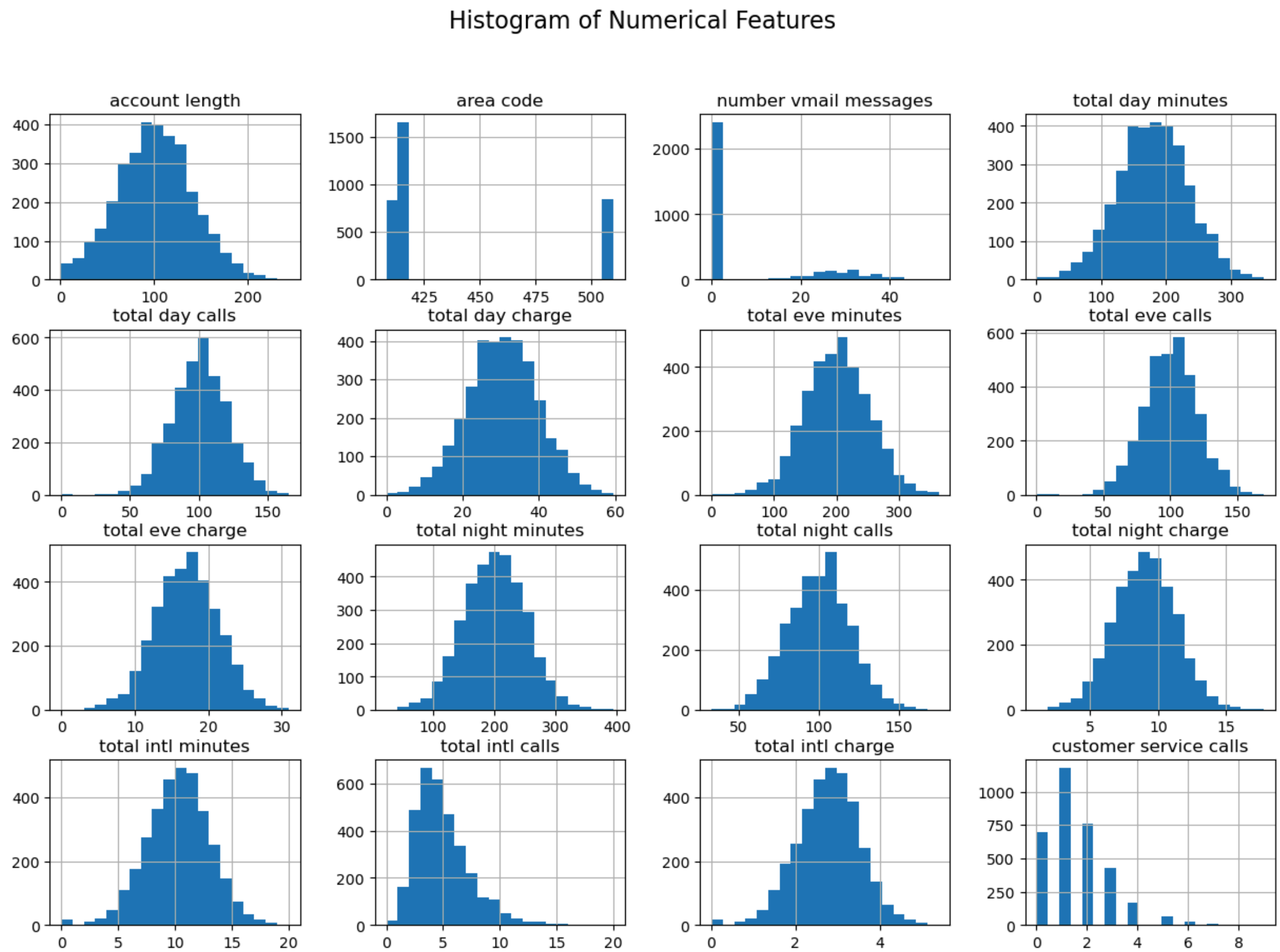

international plan Distribution
3000
2500
2000
1500
1000
500
0
count
no
yes
international plan


voice mail plan Distribution
2500
2000
1500
1000
500
0
count
yes
no
voice mail plan


area code Distribution
1600
1400
1200
1000
800
600
count




```
# histogram of numerical features
df.hist(figsize=(15, 10), bins=20)
plt.suptitle('Histogram of Numerical Features', fontsize=16)
plt.show()
```



3. Data Preparation

3.1 Check for missing values

```
df.isnull().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
```

As shown above there are no missing values.

3.2 Check for Duplicates

```
df.duplicated().sum()
df = df.drop_duplicates()
```

All duplicates have been dropped so now we have a dataset with no missing values and no duplicates.

3.3 Fix Data Types

```
df.dtypes
```

```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
```

```
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

```
df.dtypes
```

```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

3.4 Drop Irrelevant Columns

```
# dropping unnecessary columns
df = df.drop(columns=['phone number'])
df.columns
```

```
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 calls',
       'churn'],
      dtype='object')
```

3.5 Feature Engineering

```
# handling categorical variables with one-hot encoding
df = pd.get_dummies(df, columns=['area code', 'state'], drop_first=True)
```

```
# creating a new feature 'total minutes'
df['total minutes'] = df['total day minutes'] + df['total eve minutes'] + df['total night mi
```

```
# customer support intensity
df['high customer support'] = df['customer service calls'].apply(lambda x: 1 if x > 3 else 0
df['high customer support'].value_counts()
```

```
high customer support
0    3066
1     267
Name: count, dtype: int64
```

The customer support intensity is used mainly to flag for customers who called customer support more than 3 times. Therefore the above results show 267 customers called customer support more than 3 times.

3.6 Recheck for Data Integrity

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 72 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   account length          3333 non-null   int64
 1   international plan      3333 non-null   object
 2   voice mail plan         3333 non-null   object
 3   number vmail messages   3333 non-null   int64
 4   total day minutes       3333 non-null   float64
 5   total day calls         3333 non-null   int64
 6   total day charge        3333 non-null   float64
 7   total eve minutes       3333 non-null   float64
 8   total eve calls         3333 non-null   int64
 9   total eve charge        3333 non-null   float64
```

```
 10  total night minutes     3333 non-null   float64
 11  total night calls       3333 non-null   int64
 12  total night charge      3333 non-null   float64
 13  total intl minutes      3333 non-null   float64
 14  total intl calls        3333 non-null   int64
 15  total intl charge       3333 non-null   float64
 16  customer service calls  3333 non-null   int64
 17  churn                   3333 non-null   bool
 18  area code_415           3333 non-null   bool
 19  area code_510           3333 non-null   bool
 20  state_AL                3333 non-null   bool
 21  state_AR                3333 non-null   bool
 22  state_AZ                3333 non-null   bool
 23  state_CA                3333 non-null   bool
 24  state_CO                3333 non-null   bool
 25  state_CT                3333 non-null   bool
 26  state_DC                3333 non-null   bool
 27  state_DE                3333 non-null   bool
 28  state_FL                3333 non-null   bool
 29  state_GA                3333 non-null   bool
 30  state_HI                3333 non-null   bool
 31  state_IA                3333 non-null   bool
 32  state_ID                3333 non-null   bool
 33  state_IL                3333 non-null   bool
 34  state_IN                3333 non-null   bool
 35  state_KS                3333 non-null   bool
 36  state_KY                3333 non-null   bool
 37  state_LA                3333 non-null   bool
 38  state_MA                3333 non-null   bool
 39  state_MD                3333 non-null   bool
 40  state_ME                3333 non-null   bool
 41  state_MI                3333 non-null   bool
 42  state_MN                3333 non-null   bool
 43  state_MO                3333 non-null   bool
 44  state_MS                3333 non-null   bool
 45  state_MT                3333 non-null   bool
 46  state_NC                3333 non-null   bool
 47  state_ND                3333 non-null   bool
 48  state_NE                3333 non-null   bool
 49  state_NH                3333 non-null   bool
 50  state_NJ                3333 non-null   bool
 51  state_NM                3333 non-null   bool
```

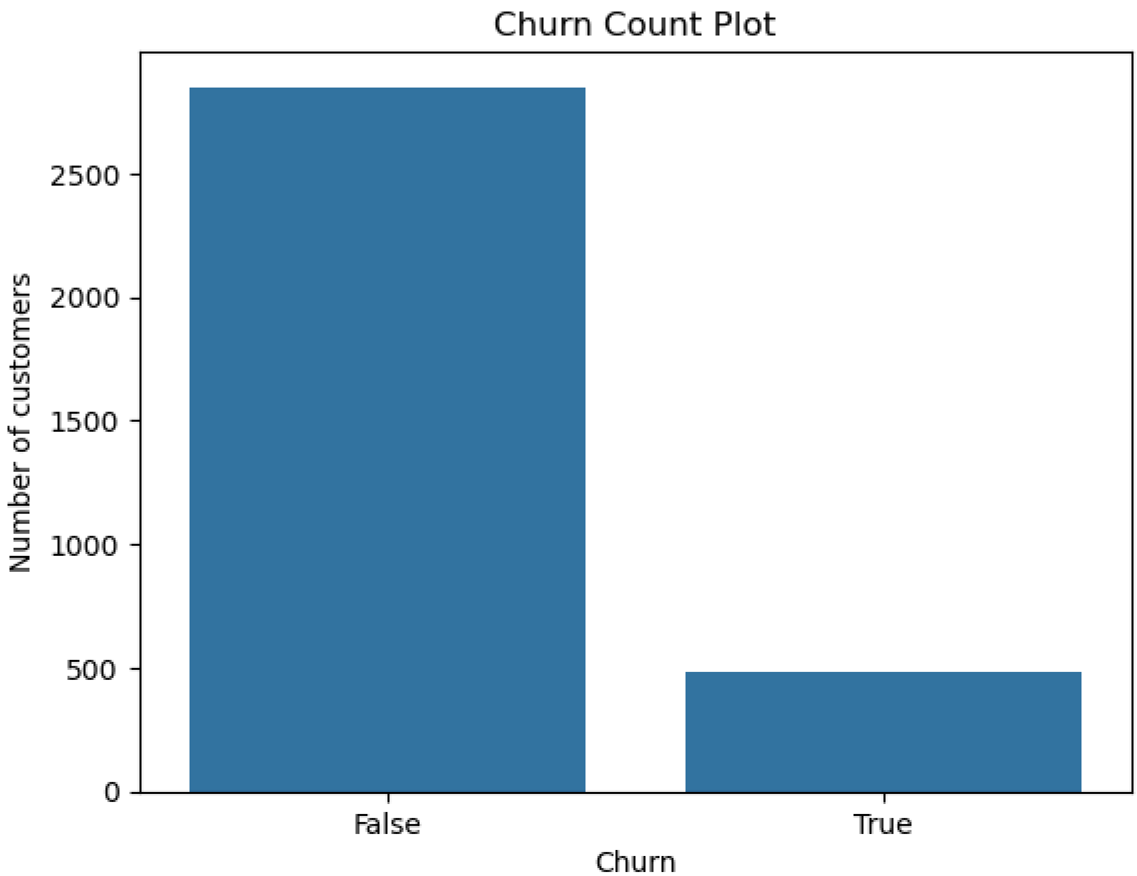
4. Exploratory Data Analysis(EDA)

Key Queston 1 What proportion of customers churned vs stayed?

This gives an overview of the class balance

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

```
plt.xlabel('Churn')
plt.ylabel('Number of customers')
plt.show()
```




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

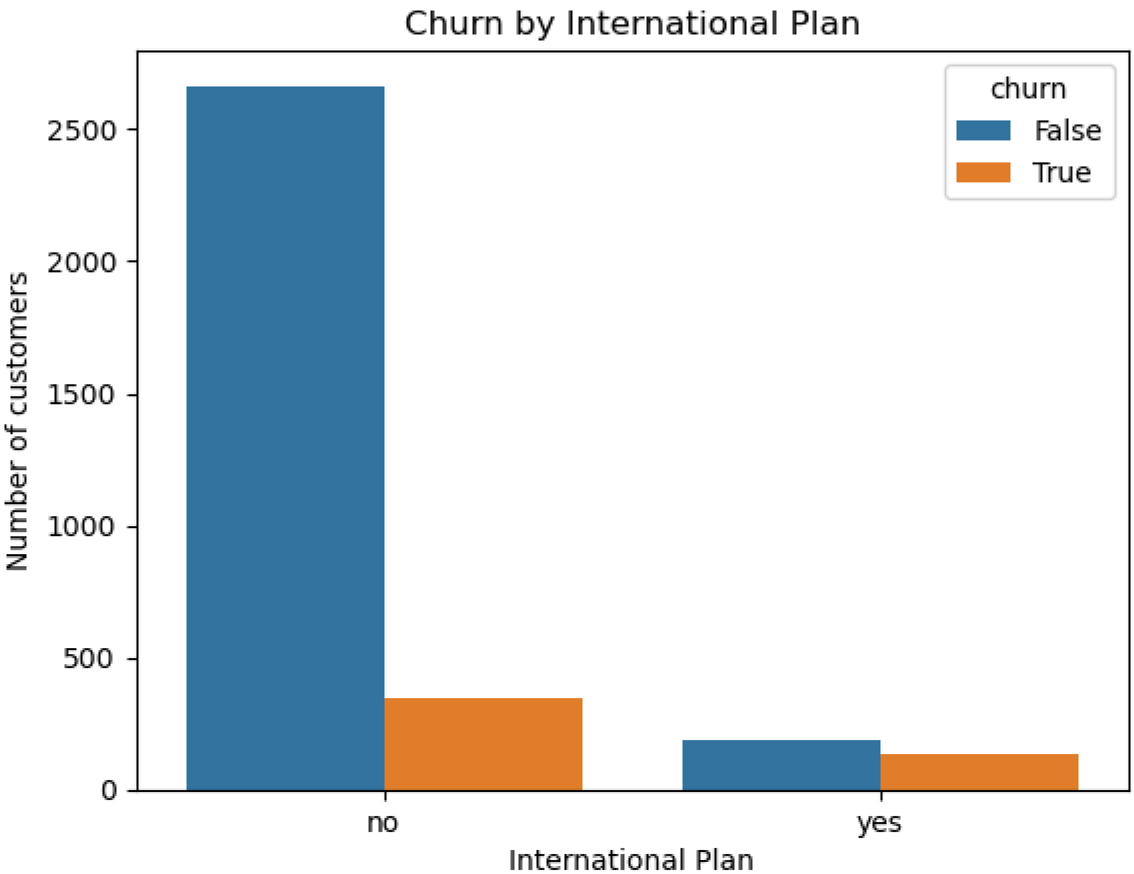
```
churn
False    2850
True      483
Name: count, dtype: int64
```

According to the visualization above there is a class imbalance where most of the customers stayed a small number of custmers churned

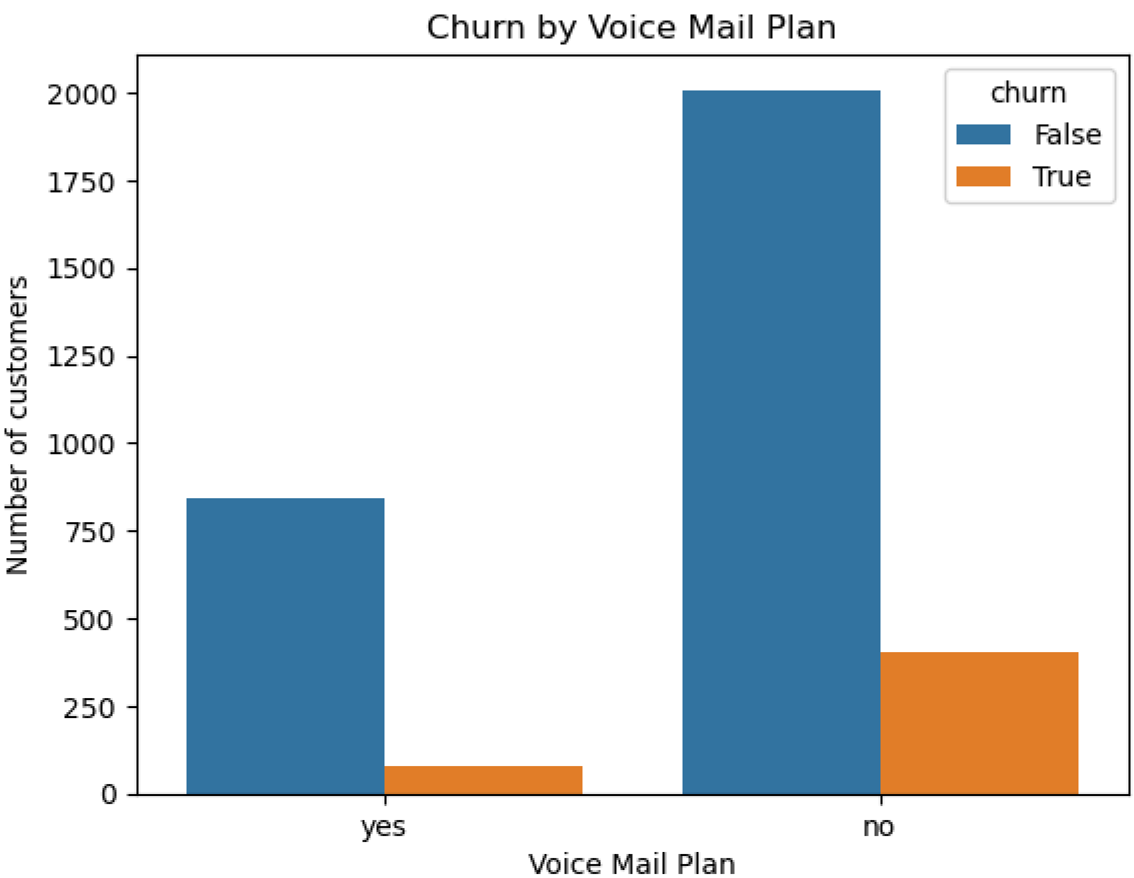
Key Question 2 Which services are most associated with customer

This helps identify service types that correlate with churn.

```
sns.countplot(x='international plan', hue='churn', data=df)
plt.title('Churn by International Plan')
plt.xlabel('International Plan')
plt.ylabel('Number of customers')
plt.show()
```



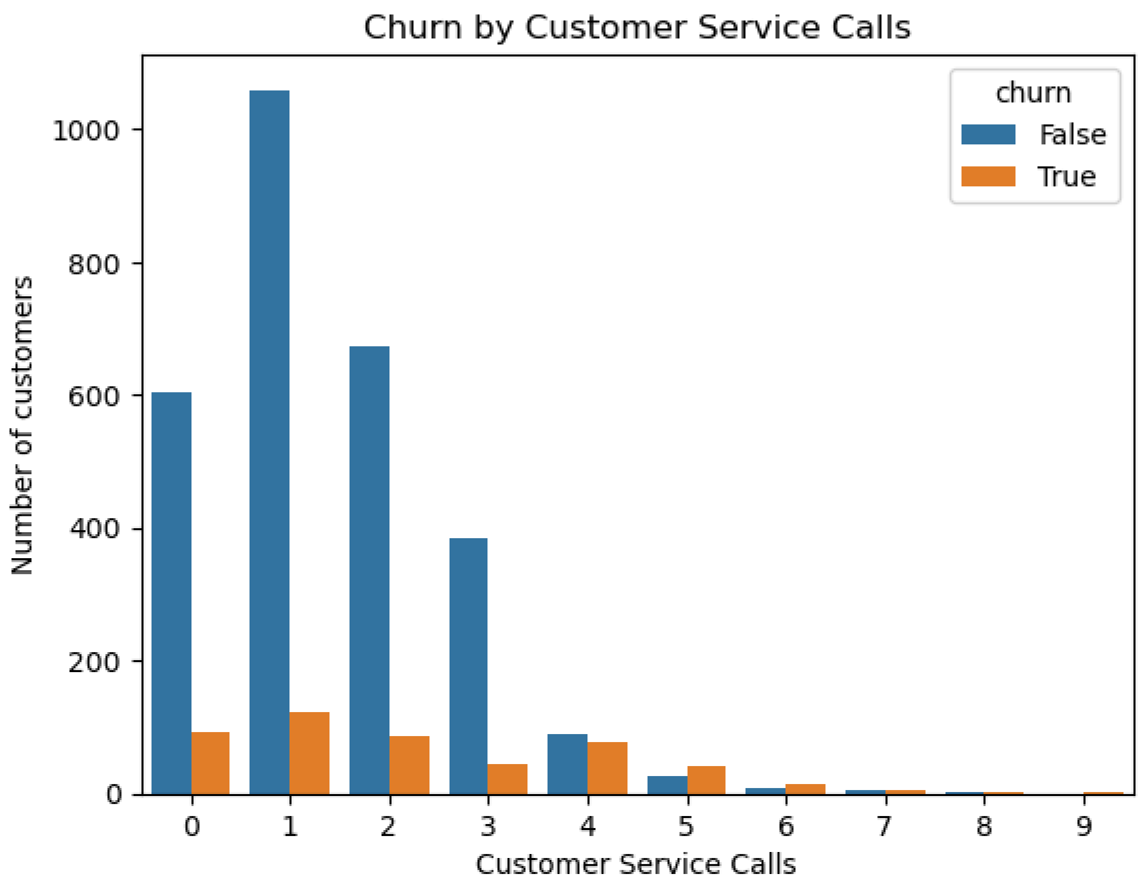

```
#countplot of voice mail plan vs churn
sns.countplot(x='voice mail plan', hue='churn', data=df)
plt.title('Churn by Voice Mail Plan')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Number of customers')
plt.show()
```



Key Question 3 How does customer service call frequency relate to Churn?

This is for the purpose of knowing if users with high service issues are more likely to leave.

```
# customer service calls vs churn
sns.countplot(x='customer service calls', hue='churn', data=df)
plt.title('Churn by Customer Service Calls')
plt.xlabel('Customer Service Calls')
plt.ylabel('Number of customers')
plt.show()
```



5. Modelling

5.1 Train/Test Split

```
# changing the data type of 'churn' to int
df['churn'] = df['churn'].astype('int')

# binarizing categorical variables
df['international plan'] = df['international plan'].map({'yes': 1, 'no': 0})
df['voice mail plan'] = df['voice mail plan'].map({'yes': 1, 'no': 0})

# training and testing data split
from sklearn.model_selection import train_test_split
X = df.drop(columns=['churn'])
y = df['churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5.2 Base Model-Logistic Regression

```
# logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
logreg = LogisticRegression(class_weight= 'balanced', max_iter=10000)
logreg.fit(X_train, y_train)
y_pred_log = logreg.predict(X_test)
y_proba_log = logreg.predict_proba(X_test)[:, 1]
```

5.3 Evaluating the base model

```
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
# evaluating the model
print('-----{"Logistic Regression Model"}-----')
print('Accuracy:', round(accuracy_score(y_test, y_pred_log), 4))
print('Precision:', round(precision_score(y_test, y_pred_log), 4))
print('Recall:', round(recall_score(y_test, y_pred_log), 4))
print('F1 Score:', round(f1_score(y_test, y_pred_log), 4))
print('ROC AUC:', round(roc_auc_score(y_test, y_proba_log), 4))
print('\nClassification Report:\n', classification_report(y_test, y_pred_log))
```

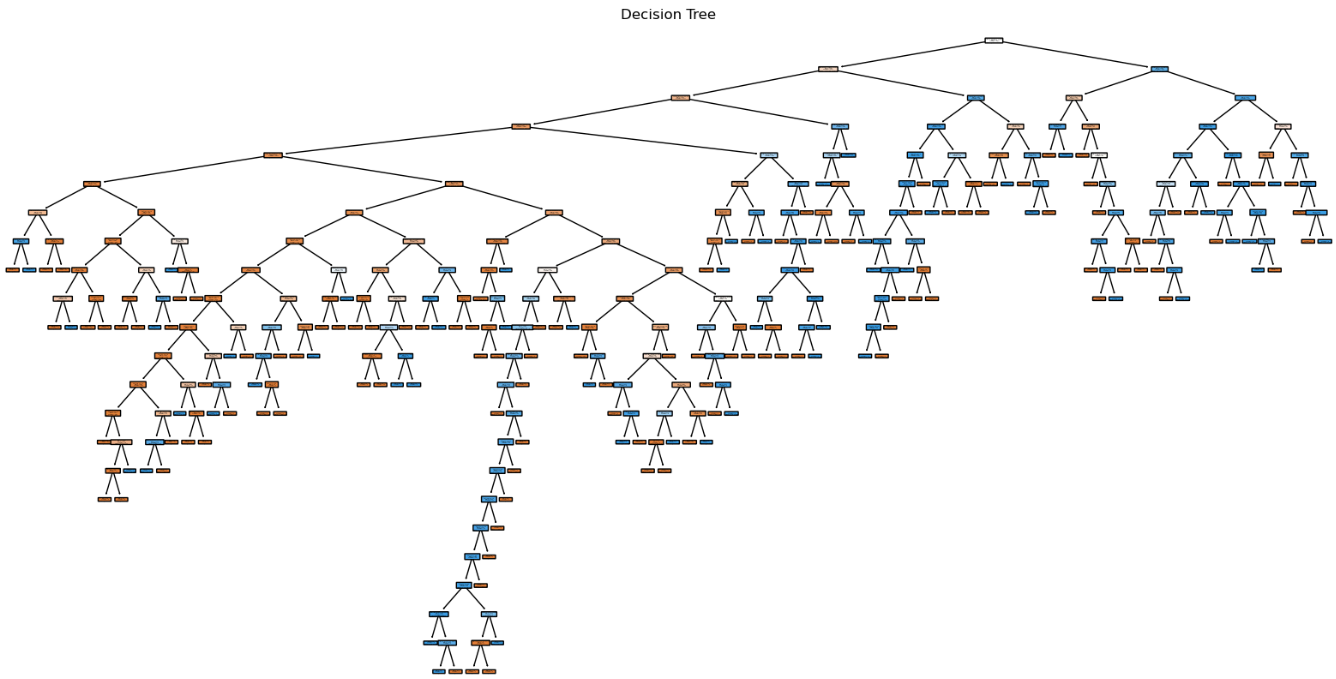
```
-----{"Logistic Regression Model"}-----
Accuracy: 0.8381
Precision: 0.478
Recall: 0.7525
F1 Score: 0.5846
ROC AUC: 0.8584

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.85      0.90       566
           1       0.48      0.75      0.58       101

    accuracy                           0.84       667
   macro avg       0.71      0.80      0.74       667
weighted avg       0.88      0.84      0.85       667
```

5.4 Advanced Model - Decision Tree

Predict

```
# decision tree model
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(class_weight= 'balanced', random_state=42)
```

```
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)
y_proba_tree = tree_model.predict_proba(X_test)[:, 1]


# visualizing the decision tree
from sklearn.tree import plot_tree
plt.figure(figsize=(20, 10))
plot_tree(tree_model, feature_names=X.columns, filled=True, rounded=True)
plt.title('Decision Tree')
plt.show()
```



5.5 Evaluating the advanced model

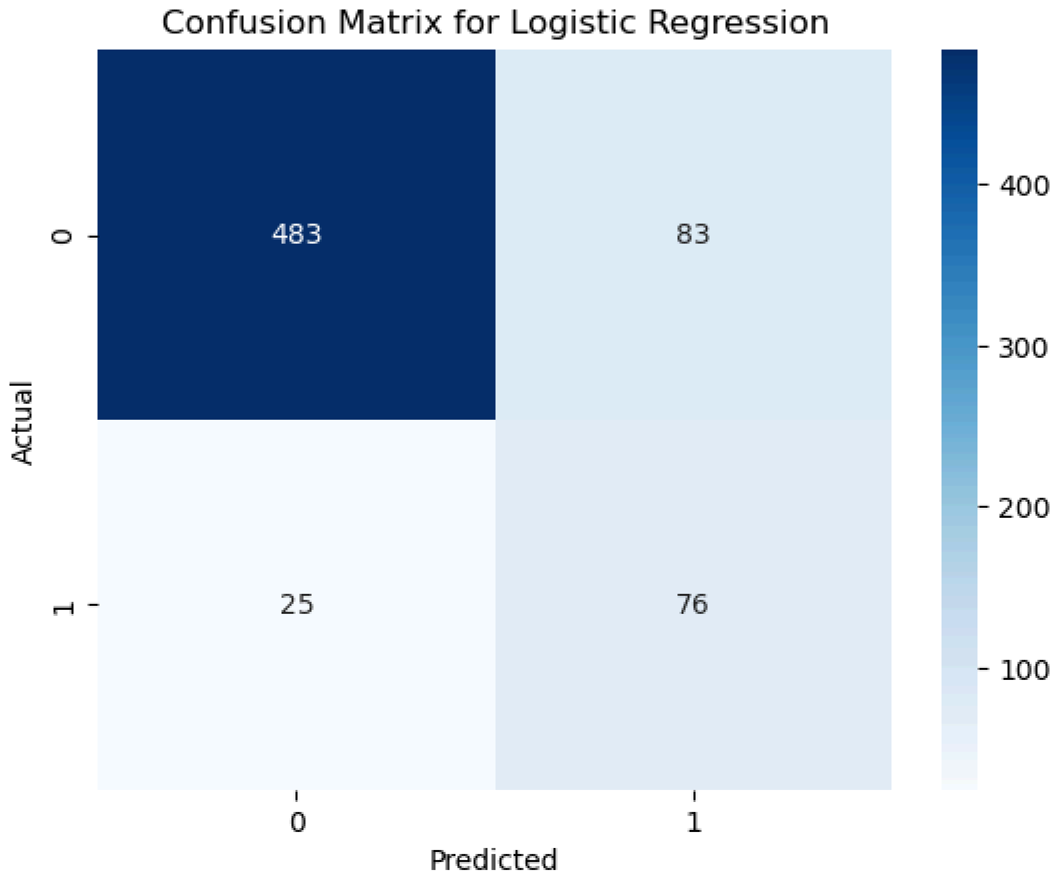
```
# evaluating the decision tree model
print('-----{"Decision Tree Model"}-----')
print('Accuracy:', round(accuracy_score(y_test, y_pred_tree), 4))
```

```
print('Precision:', round(precision_score(y_test, y_pred_tree), 4))
print('Recall:', round(recall_score(y_test, y_pred_tree), 4))
print('F1 Score:', round(f1_score(y_test, y_pred_tree), 4))
print('ROC AUC:', round(roc_auc_score(y_test, y_proba_tree), 4))
print('\nClassification Report:\n', classification_report(y_test, y_pred_tree))
```

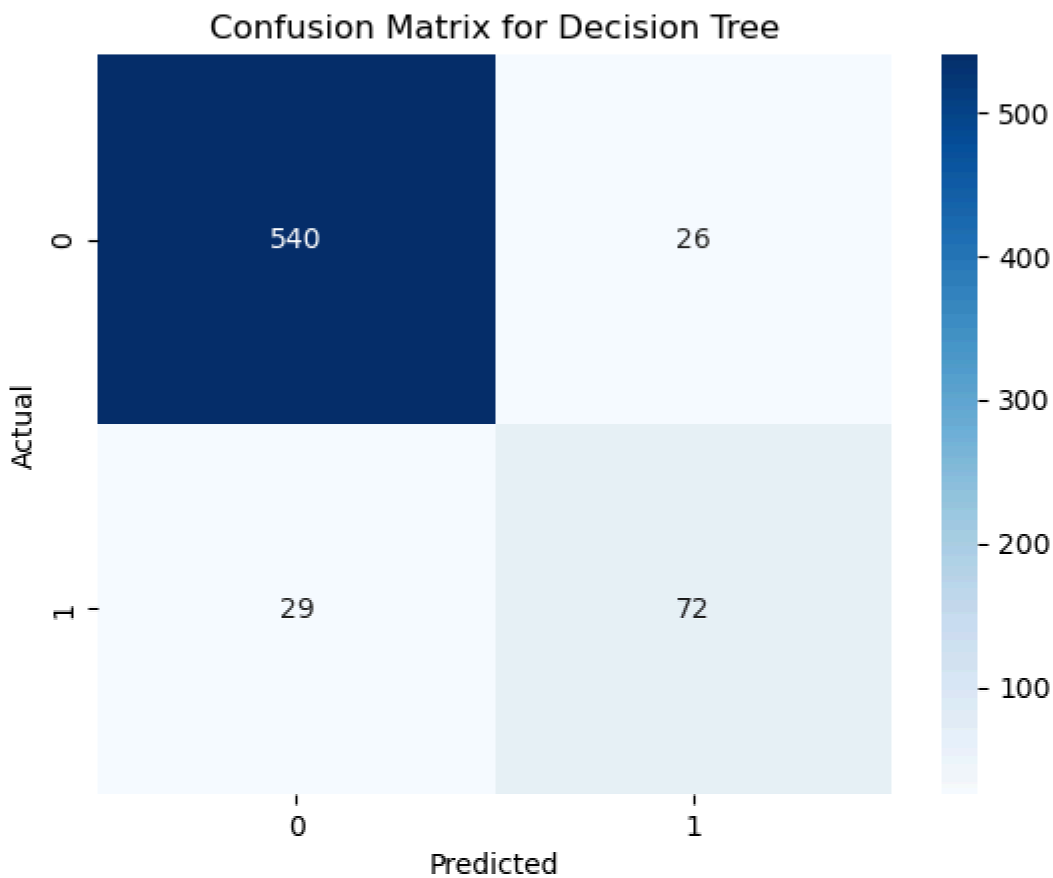
```
-----{"Decision Tree Model"}-----
Accuracy: 0.9175
Precision: 0.7347
Recall: 0.7129
F1 Score: 0.7236
ROC AUC: 0.8335

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.95      0.95       566
           1       0.73      0.71      0.72       101

    accuracy                           0.92       667
   macro avg       0.84      0.83      0.84       667
weighted avg       0.92      0.92      0.92       667
```

5.6 Confusion Matrix

```
# confusion matrix for logistic regression
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
cm_log = confusion_matrix(y_test, y_pred_log)
sns.heatmap(cm_log, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



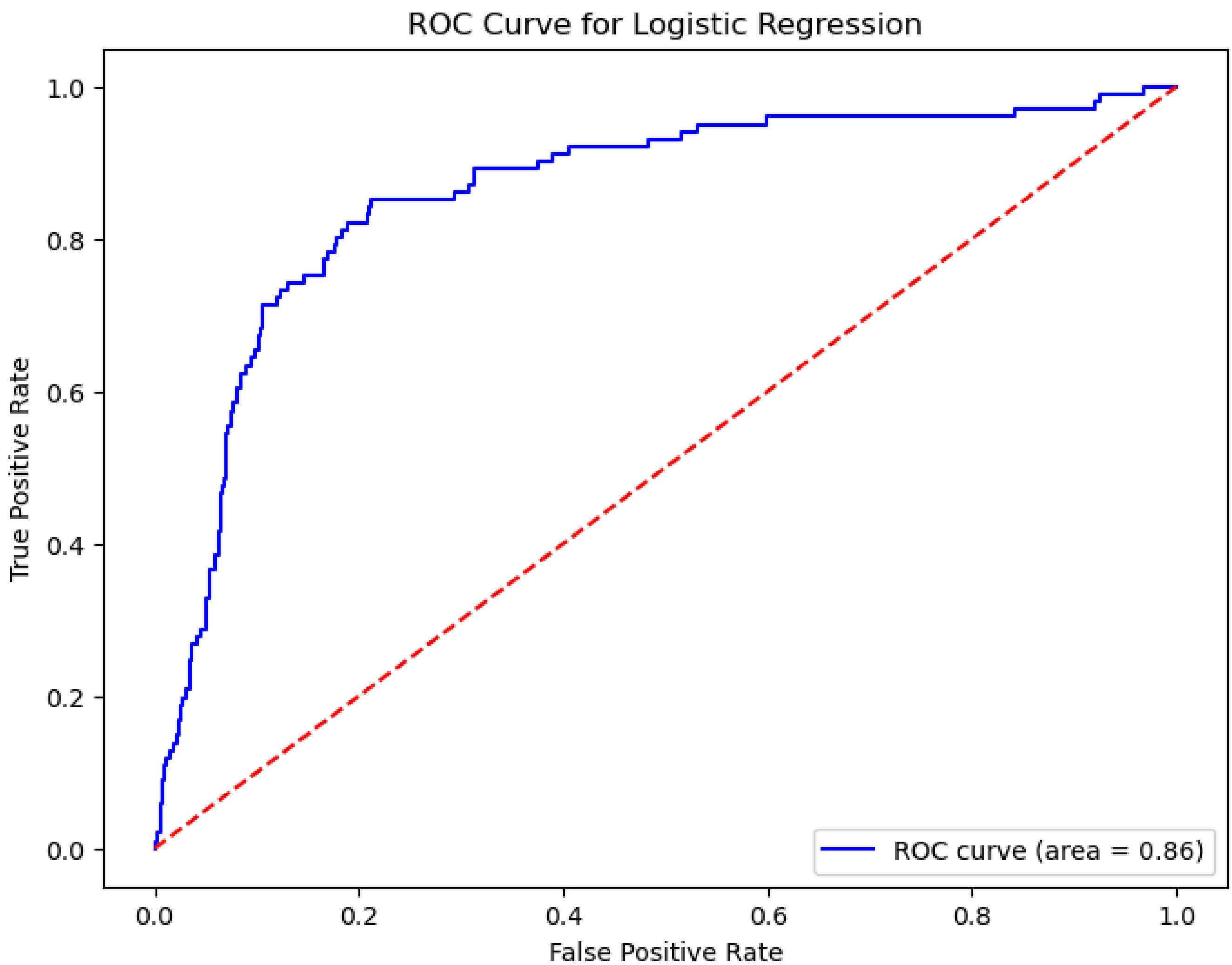

```
# confusion matrix for decision tree
cm_tree = confusion_matrix(y_test, y_pred_tree)
sns.heatmap(cm_tree, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Decision Tree')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



5.7 Model Comparison

5.7.1 ROC Curve

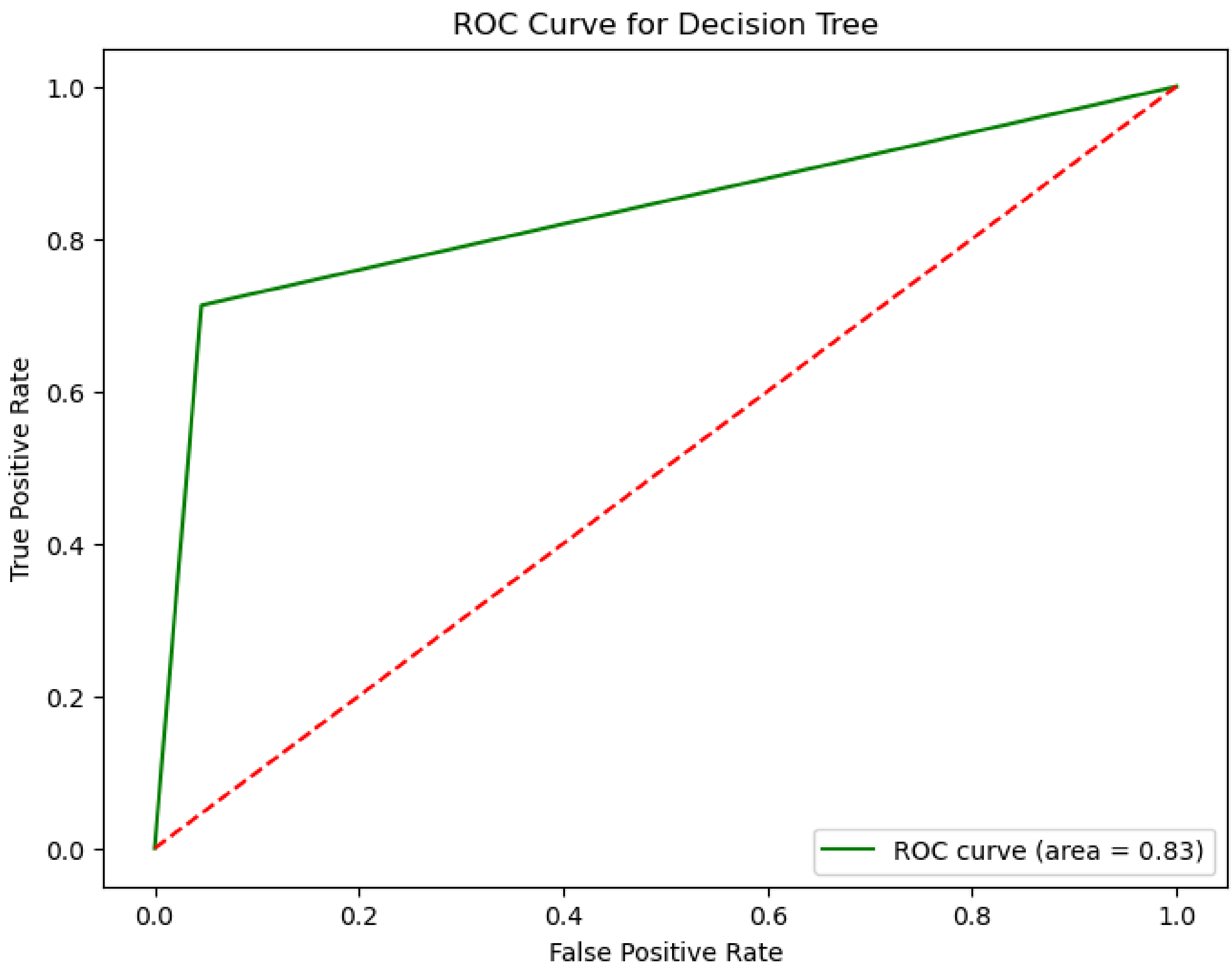
```
#ROC curve for logistic regression
from sklearn.metrics import roc_curve, auc
fpr_log, tpr_log, thresholds_log = roc_curve(y_test, y_proba_log)
roc_auc_log = auc(fpr_log, tpr_log)
plt.figure(figsize=(8, 6))
plt.plot(fpr_log, tpr_log, color='blue', label='ROC curve (area = %0.2f)' % roc_auc_log)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('ROC Curve for Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



The ROC curve (blue line)is generally positioned further away from the random classifier line(red dotted line), especially in the middle range of FPR.

The Area Under the Curve is 0.86. This indicates a good ability of the Logistic Regression model to discriminate between the positive and negative classes.

```
# ROC curve for decision tree
fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test, y_proba_tree)
roc_auc_tree = auc(fpr_tree, tpr_tree)
plt.figure(figsize=(8, 6))
plt.plot(fpr_tree, tpr_tree, color='green', label='ROC curve (area = %0.2f)' % roc_auc_tree)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('ROC Curve for Decision Tree')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



The ROC curve(green line) is also above the random classifier line, indicating better than random performance.

The Area Under the Curve is 0.83. This suggests that the model also performs well in distinguishing between the classes, but slightly less so than the Logistic Regression model based on the AUC value.

5.7.2 Evaluation

```
print('-----{"Logistic Regression Model"}-----')
print('Accuracy:', round(accuracy_score(y_test, y_pred_log), 4))
print('Precision:', round(precision_score(y_test, y_pred_log), 4))
print('Recall:', round(recall_score(y_test, y_pred_log), 4))
print('F1 Score:', round(f1_score(y_test, y_pred_log), 4))
print('ROC AUC:', round(roc_auc_score(y_test, y_proba_log), 4))


print('-----{"Decision Tree Model"}-----')
```