Business Understanding

- SyriaTel, a telecommunications company, is facing challenges with customer retention. A significant number of customers are churning, leading to revenue loss and increased costs associated with acquiring new customers. Understanding and predicting customer churn is critical for SyriaTel to implement targeted retention strategies, improve customer satisfaction, and ultimately reduce financial losses. By leveraging data analytics and machine learning, SyriaTel can identify patterns and predictors of churn, enabling proactive measures to retain at-risk customers.

Problem Statement

- Customer churn is a major issue for SyriaTel, resulting in lost revenue and increased operational costs. Without actionable insights, SyriaTel cannot effectively implement retention strategies, leading to continued customer attrition. The goal is:
    - Predict numerical factors that influence churn, such as service usage, customer complaints, or charges.(Regression task)
    - Predict whether a customer will churn (Classification task)

Objectives

Primary Objective:

- Develop a binary classification model to predict whether a customer will churn ("soon") based on historical customer data with high accuracy and reliability

Secondary Objectives:

- Identify key factors (features) that contribute to customer churn.
- Provide actionable insights to SyriaTel to design targeted retention strategies.
- Optimize model performance through feature engineering and hyperparameter tuning then select the best performing classifier either using F1-score or ROC-AUC.

Research Questions

1. What are the most significant factors influencing customer churn?
2. Does frequent customer service interaction indicate a higher risk of churn?
3. Do customers with an international plan have a higher churn rate?

4. Can a machine learning model accurately predict churn using available features?

Data understanding

Exploratory Data Analysis

```
# import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from scipy.stats import chi2_contingency
from scipy.stats import zscore
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusi
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import plot_tree


# load churn.csv dataset
df_churn = pd.read_csv('churn.csv')
df_churn.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

```
df_churn.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
```

```
# statistical overview of the data such as column names, no. of columns and data types
df_churn.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

```
# get the shape to show total number of columns and rows
df_churn.shape
```

```
(3333, 21)
```

```
# check unique values especiallly in categorical data columns
for col in df_churn.select_dtypes(include=['object']).columns:
    print(f"{col}: {df_churn[col].unique()}")
    print(df_churn[col].unique()[:10])
    print("-"* 40)
```

```
state: ['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']
['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'LA' 'WV' 'IN']
----------------------------------------
phone number: ['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']
['382-4657' '371-7191' '358-1921' '375-9999' '330-6626' '391-8027'
 '355-9993' '329-9001' '335-4719' '330-8173']
----------------------------------------
international plan: ['no' 'yes']
['no' 'yes']
----------------------------------------
voice mail plan: ['yes' 'no']
['yes' 'no']
----------------------------------------
```

Data Cleaning and preprocessing

```
# correct format to check data types
print(df_churn.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
```

```
 #First create contingency tables
state_contingency = pd.crosstab(df_churn['state'], df_churn['churn'])
area_code_contingency = pd.crosstab(df_churn['area code'], df_churn['churn'])

state_chi2, state_p, state_dof, state_expected = chi2_contingency(state_contingency)
area_code_chi2, area_code_p, area_code_dof, area_code_expected = chi2_contingency(area_code_
print("State Chi-Square Test:")
print(f"State Chi-Square Statistic: {state_chi2}, p-value: {state_p}")
print(f"Area Code Chi-Square Statistic: {area_code_chi2}, p-value: {area_code_p}")
```

```
State Chi-Square Test:
State Chi-Square Statistic: 83.04379191019663, p-value: 0.002296221552011188
Area Code Chi-Square Statistic: 0.17754069117425395, p-value: 0.9150556960243712
```

- State vs churn chi-square statistic is 83.04 and p-value of 0.0023 thus < 0.05 showing a significant impact on churn meaning we retain the column
- Area code vs churn chi-square statistic is 0.18, p-value of 0.9151 thus > than 0.05 showing no significant impact meaning we drop the column

```
# check for missing values
missing_values = df_churn.isnull().sum()


# check for duplicates
duplicate_rows = df_churn.duplicated().sum()


# unique values of churn data
unique_values = df_churn.nunique()
print("Missing Values:\n", missing_values)
print("\nDuplicate Rows:", duplicate_rows)
print("\nUnique Values per Column:\n", unique_values)
```

```
Missing Values:
 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

Duplicate Rows: 0

Unique Values per Column:
 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
```

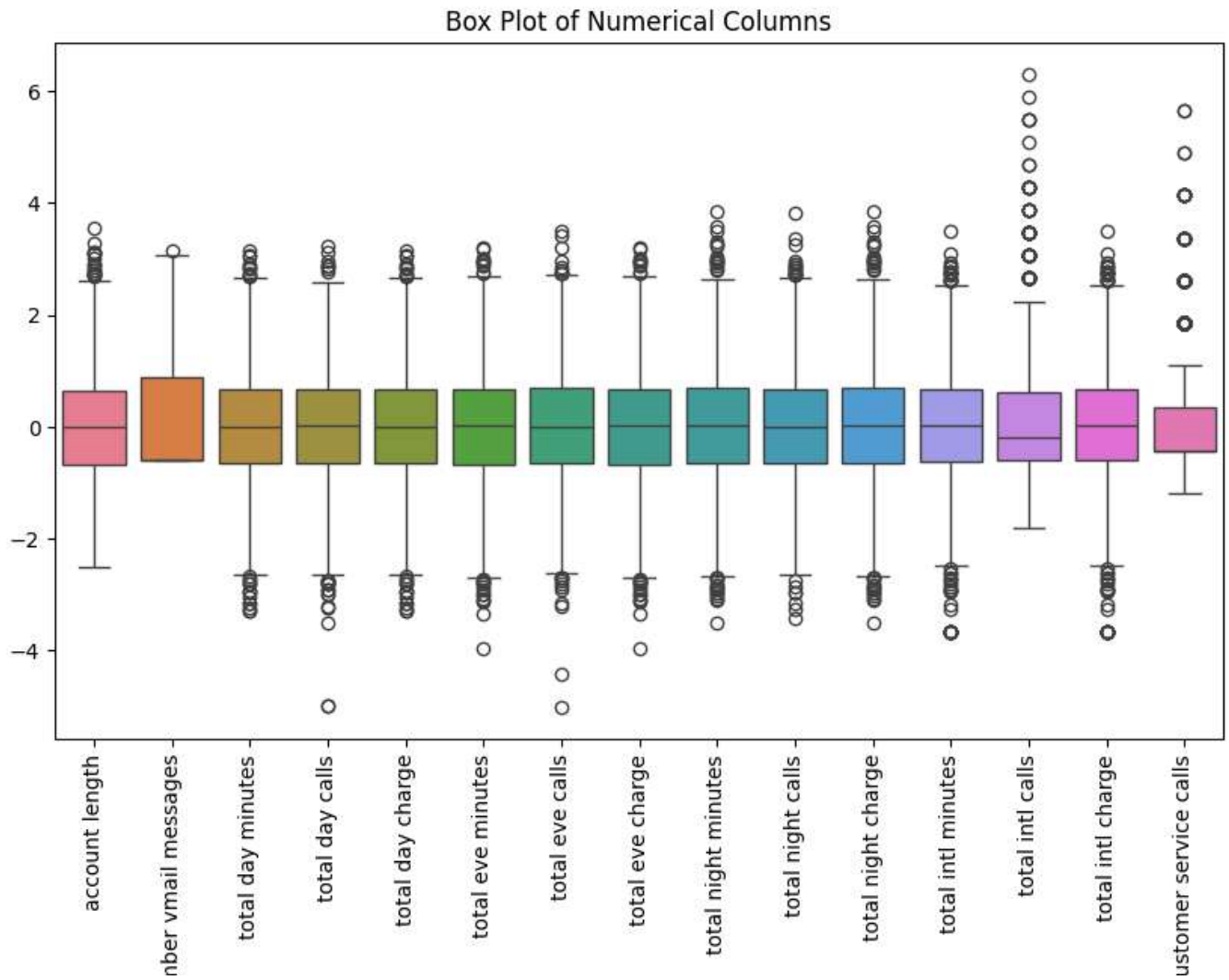
```
total night calls          120
total night charge         933
total intl minutes         162
total intl calls            21
total intl charge          162
customer service calls      10
churn                        2
dtype: int64
```

```
# Drop irrelevant columns
df_churn = df_churn.drop(columns=['state', 'area code', 'phone number'], errors='ignore')

# Standardize numerical features
scaler = StandardScaler()
numerical_features = df_churn.select_dtypes(include=['int64', 'float64']).columns.tolist()
if 'churn' in numerical_features:
 numerical_features.remove('churn')
df_churn[numerical_features] = scaler.fit_transform(df_churn[numerical_features])


# Show cleaned data
df_churn.head()
```

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	
0	0.676489	no	yes	1.234883	1.566767	0.476643	1.567036	-0.070610	-0.
1	0.149065	no	yes	1.307948	-0.333738	1.124503	-0.334013	-0.108080	0.
2	0.902529	no	no	-0.591760	1.168304	0.675985	1.168464	-1.573383	0.
3	-0.428590	yes	no	-0.591760	2.196596	-1.466936	2.196759	-2.742865	-0.
4	-0.654629	yes	no	-0.591760	-0.240090	0.626149	-0.240041	-1.038932	1.

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

```
#Check outliers by plotting a box plot for all numerical columns
plt.figure(figsize=(10, 6))

# Select only numerical columns for the box plot
numerical_cols = df_churn.select_dtypes(include=np.number).columns
sns.boxplot(data=df_churn[numerical_cols])
plt.title('Box Plot of Numerical Columns')
plt.xticks(rotation=90)# rotate for better visibility
plt.show()
```



- Multiple outliers in several features are detected.
- Total day minutes, total eve minutes, total night minutes show extreme outliers
- Total intl calls has fewer outliers but still needs to be worked on.
- Area code is categorical so should not be analyzed

```
#we use Z-score to count outliers on numerical columns
z_scores = np.abs(zscore(df_churn.select_dtypes(include=[np.number])))
threshold = 3
outlier_count = (z_scores > threshold).sum()
print(outlier_count)
```

```
account length            7
number vmail messages     3
total day minutes         9
total day calls           9
total day charge          9
total eve minutes         9
total eve calls           7
total eve charge          9
total night minutes      11
```

```
total night calls          6
total night charge        11
total intl minutes        22
total intl calls          50
total intl charge         22
customer service calls    35
dtype: int64
```

- The columns with extreme outliers > 10 need careful handling We can either;
  - Remove the outliers using z-score filtering
  - Apply log transformation- but check if data is skewed first

```
# Remove rows where any numeric column has a z-score above 3 or below -3
numeric_columns = df_churn.select_dtypes(include=[np.number]).columns
df_churn = df_churn[(np.abs(zscore(df_churn[numeric_columns])) < 3).all(axis=1)]
```

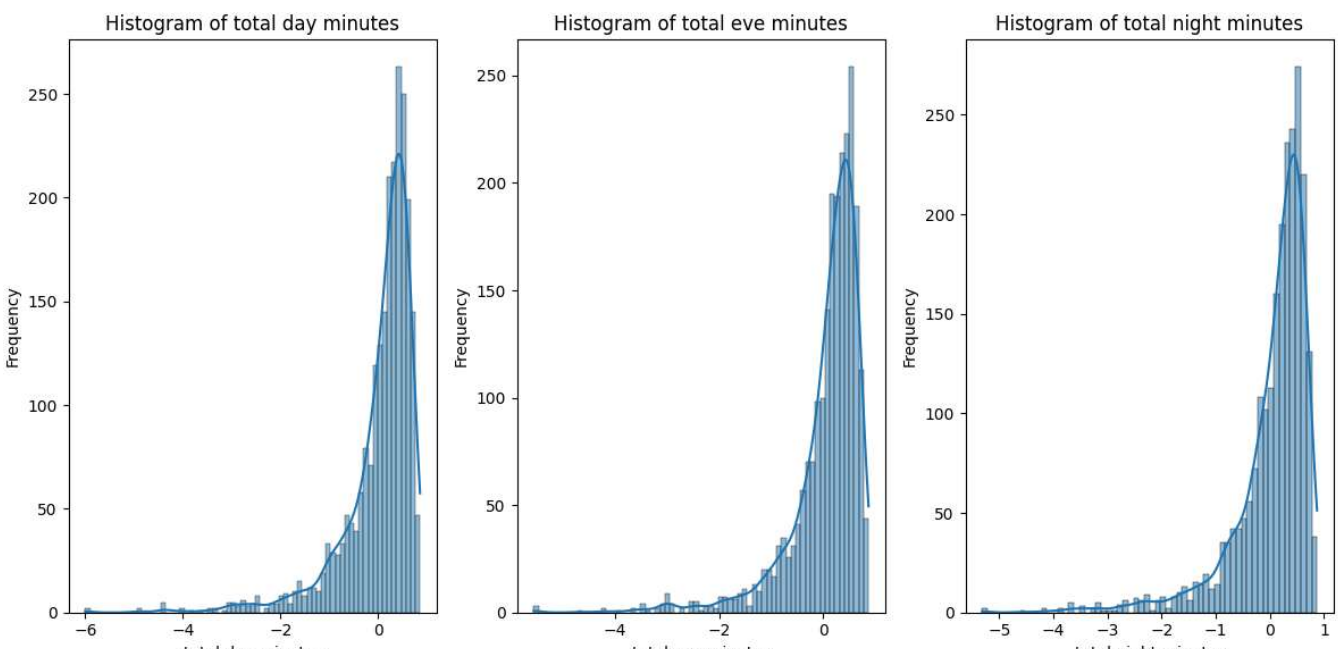
```
#check skewness
# Include only numeric columns for skewness calculation
numeric_df = df_churn.select_dtypes(include=np.number)
df_skew = numeric_df.skew()
print(df_skew)
```

```
account length            0.061234
number vmail messages     1.281257
total day minutes        -0.006560
total day calls          -0.018582
total day charge         -0.006562
total eve minutes         0.011379
total eve calls          -0.012807
total eve charge          0.011404
total night minutes      -0.024578
total night calls         0.010321
total night charge       -0.024631
total intl minutes       -0.039414
total intl calls          0.766630
total intl charge        -0.039296
customer service calls    0.723538
dtype: float64
```

```
#Apply log transformation to skewed columns
skewed_columns = ['total day minutes', 'total eve minutes', 'total night minutes']
for col in skewed_columns:
    df_churn[col] = np.log1p(df_churn[col])
```

```
#Now we plot of the histogram
plt.figure(figsize=(12, 6))
transformed_columns = ['total day minutes', 'total eve minutes', 'total night minutes']
for i, col in enumerate(transformed_columns, 1):
```

```
    plt.subplot(1, len(transformed_columns), i)
    sns.histplot(df_churn[col], kde=True)
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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



- There is presence of outliers we can perform z-score again to see if the count reduced

```
#zscore after log transformation
z_scores = np.abs(zscore(df_churn.select_dtypes(include=[np.number])))
threshold = 3
outlier_count = (z_scores > threshold).sum()
print(outlier_count)
```

```
account length            0
number vmail messages     1
total day minutes         0
total day calls           1
total day charge          1
total eve minutes         0
total eve calls           0
total eve charge          2
total night minutes       0
total night calls         0
total night charge        4
total intl minutes        6
total intl calls         28
total intl charge         6
customer service calls    0
dtype: int64
```

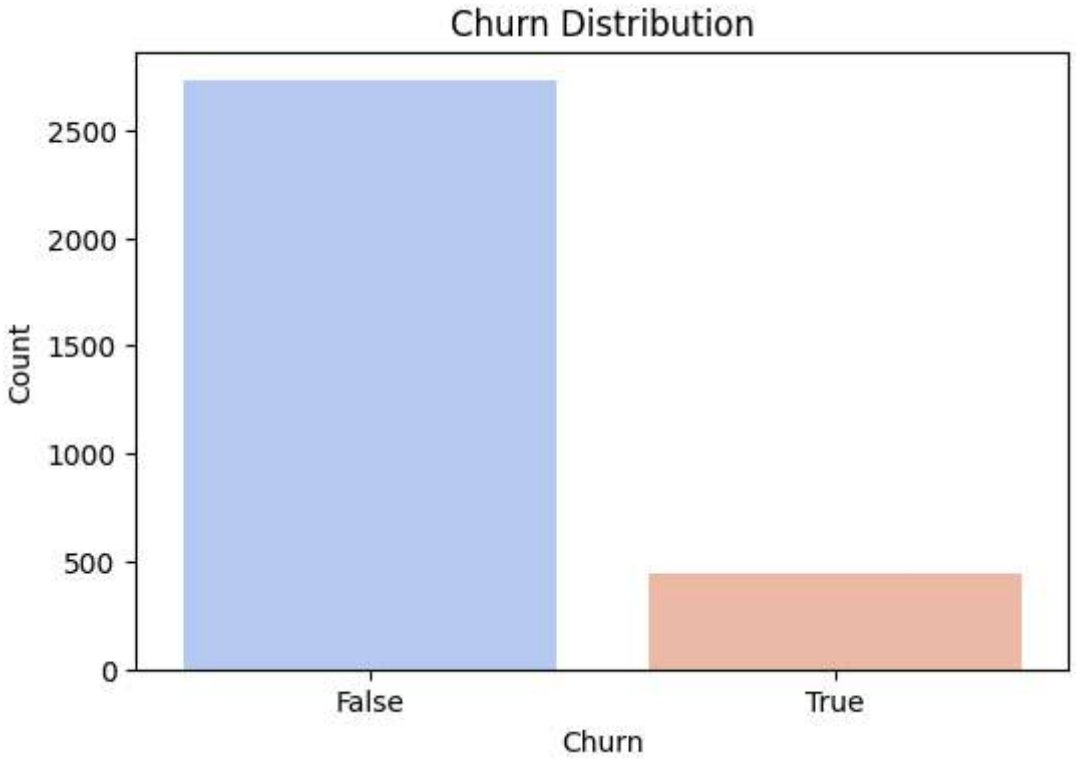
- There is a significant reduce of outlier count which indicates log transformation helped

Exploratory data analysis

Univariate Analysis (Single Variable Analysis)

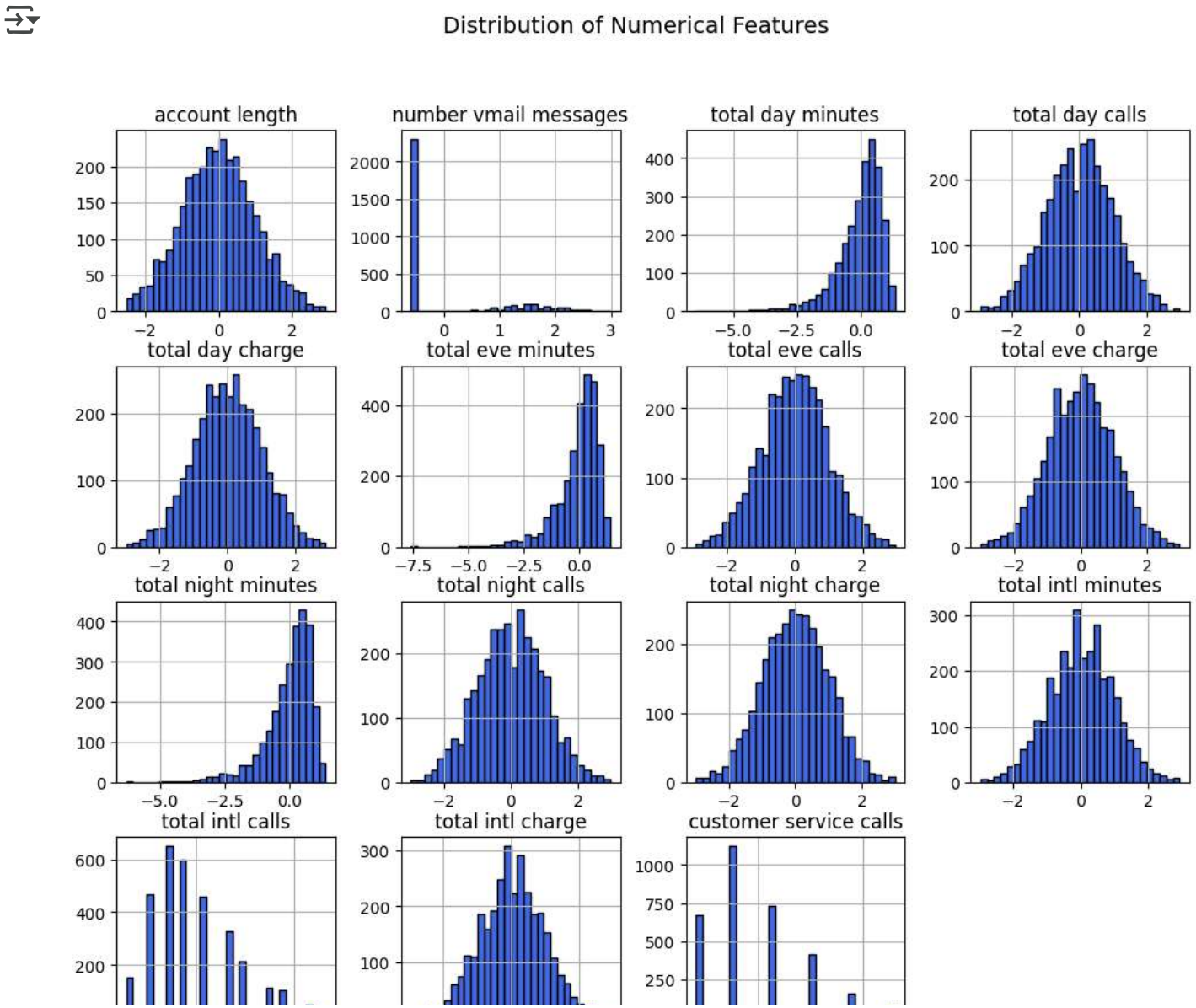
- The goal is to Understand the distribution of individual variables (one at a time).
- To analyze categorical variables (like churn, international plan) and numerical variables (like total day minutes, customer service calls)

```
# churn distribution bar chat
plt.figure(figsize=(6,4))
sns.countplot(x=df_churn['churn'], palette='coolwarm')
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



- The bar graph shows that the dataset is imbalanced, with fewer churned customers (1) than non-churned customers (0). This shows that churn is lower in number, the business might not yet have a severe churn issue

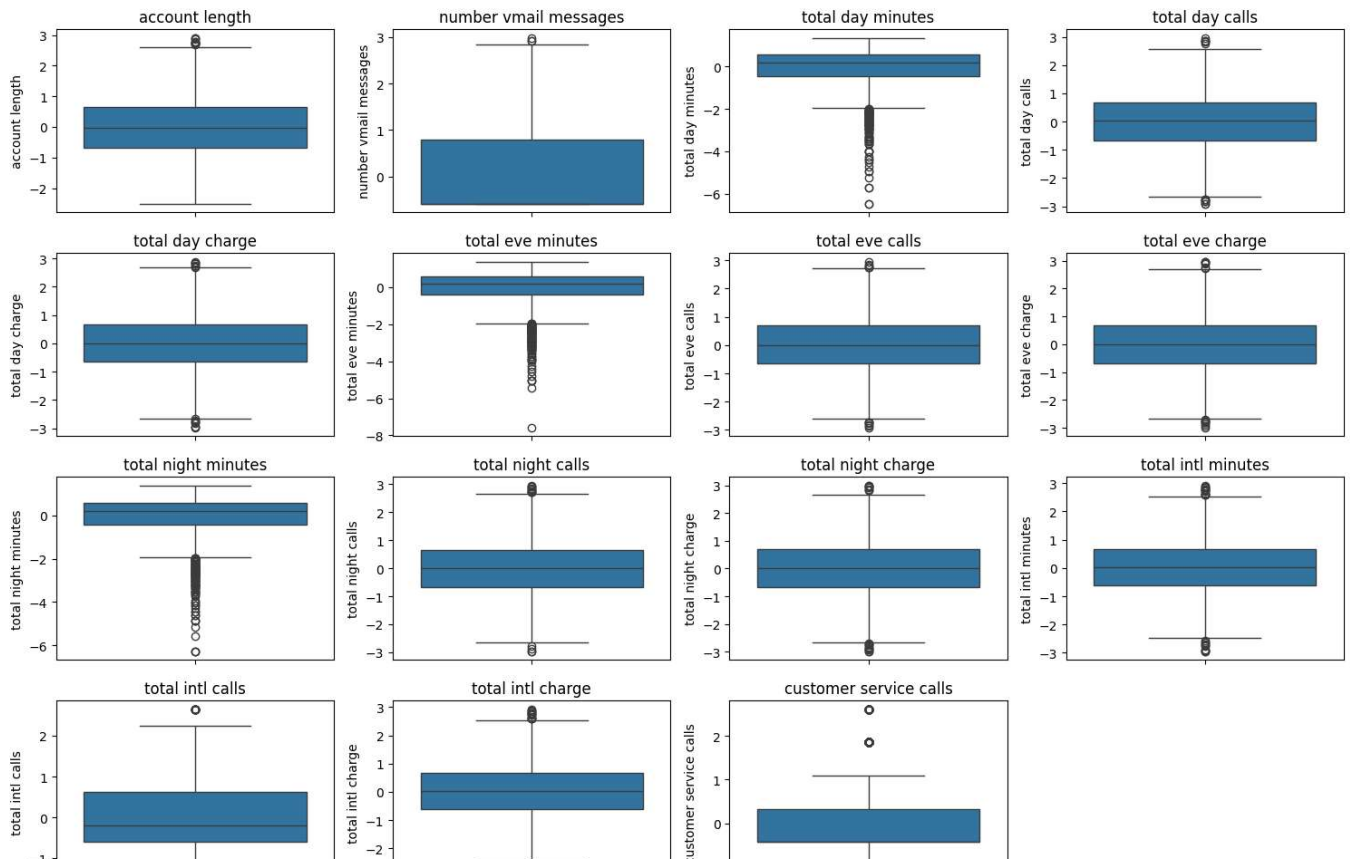
```
# Plot histograms for numerical columns
df_churn.hist(figsize=(12, 10), bins=30, color='royalblue', edgecolor='black')
plt.suptitle('Distribution of Numerical Features', fontsize=14)
plt.show()
```



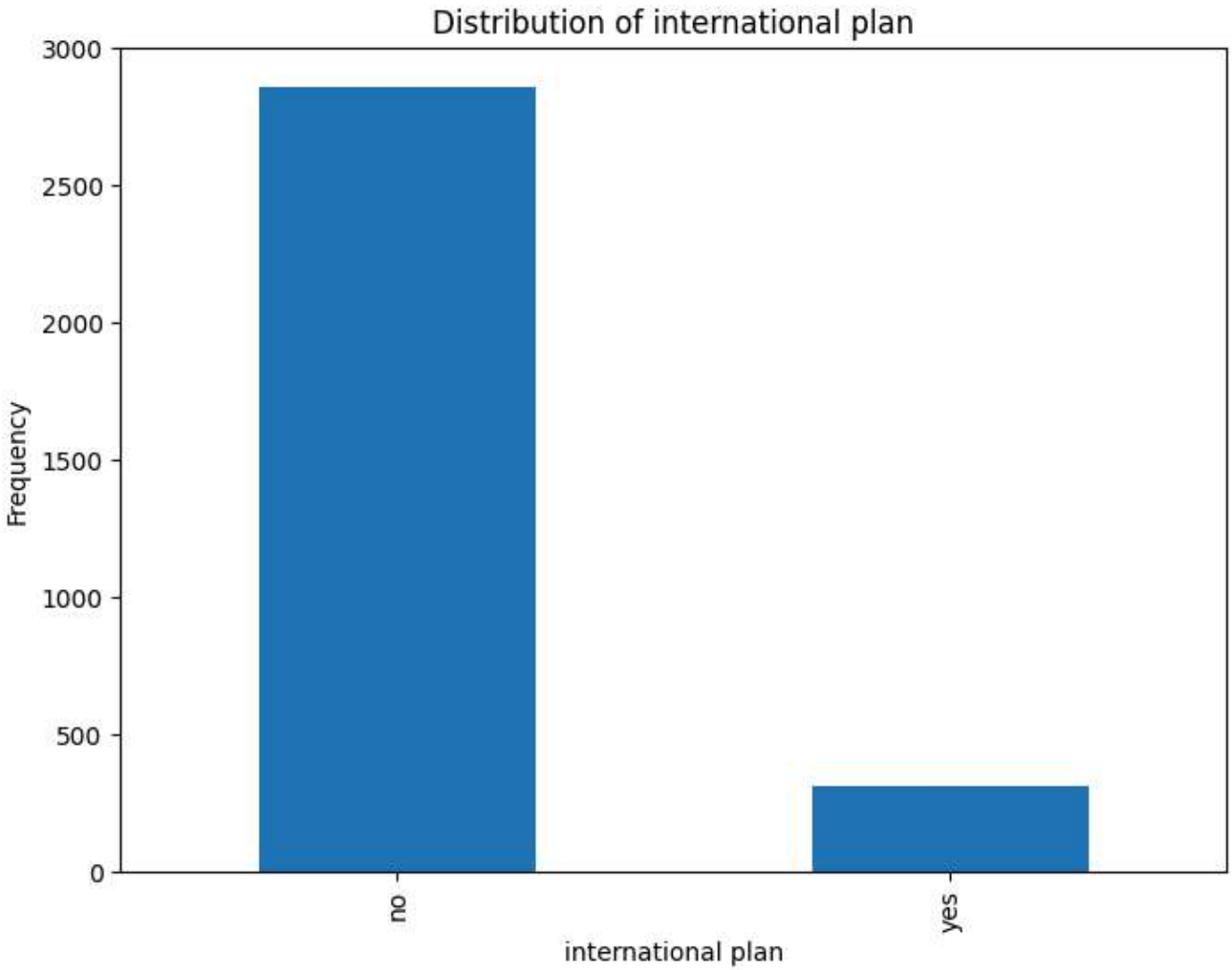
- Features like total day minutes, total eve minutes, and total night minutes are normally distributed.
- Customer service calls are right-skewed indicating that most customers make fewer calls.

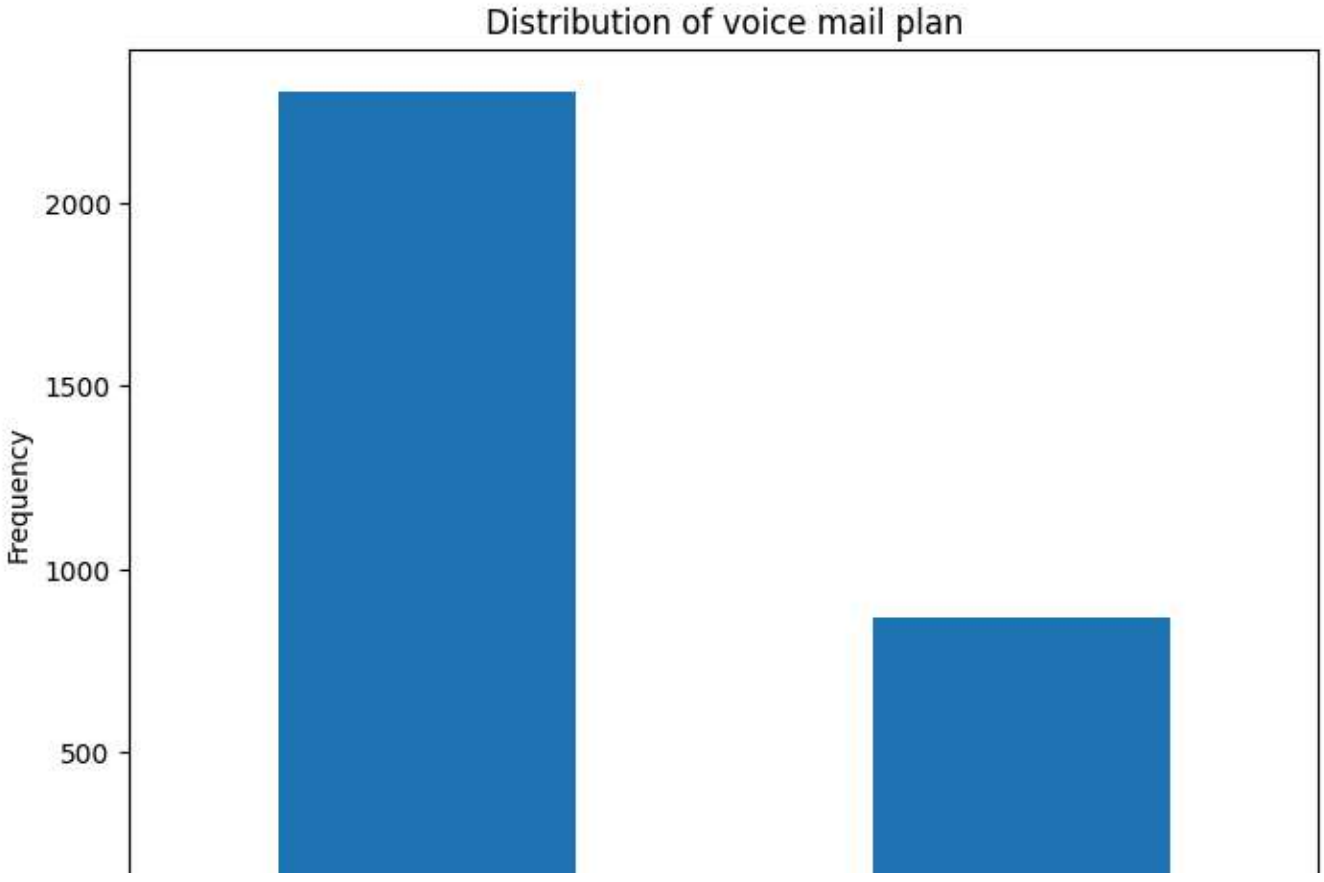
```
# Boxplots for numerical features
plt.figure(figsize=(15, 10))
num_rows = int(np.ceil(len(df_churn.select_dtypes(include=np.number).columns) / 4))
for i, col in enumerate(df_churn.select_dtypes(include=np.number).columns):
    plt.subplot(num_rows, 4, i + 1)
    sns.boxplot(y=df_churn[col])
    plt.title(col)
```

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




```
# Bar plots for categorical features
for col in df_churn.select_dtypes(include='object').columns:
    plt.figure(figsize=(8, 6))
    df_churn[col].value_counts().plot(kind='bar')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```


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


Distribution of voice mail plan
Frequency
2000
1500
1000
500

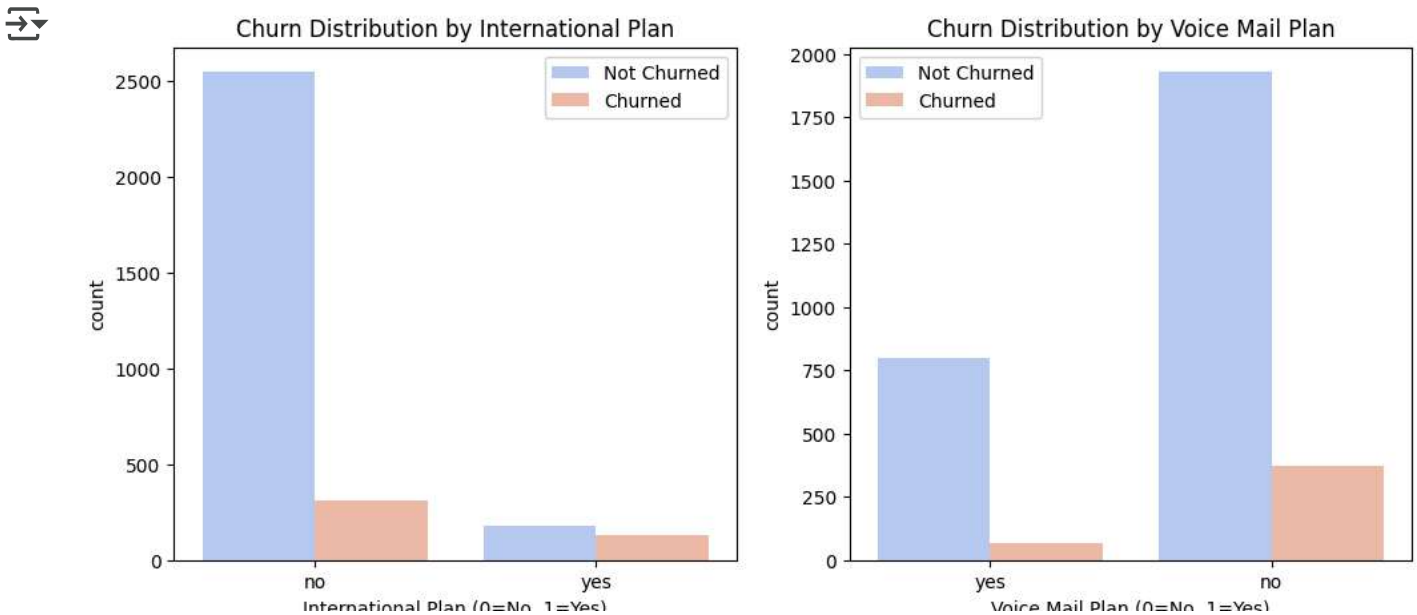
Bivariate Analysis (Two Variables)

- The goal is to Understand the relationship between two variables, especially how features relate to churn.

Churn vs. Categorical Features (International Plan & Voice Mail Plan)

- Checking how churn is distributed across categorical features

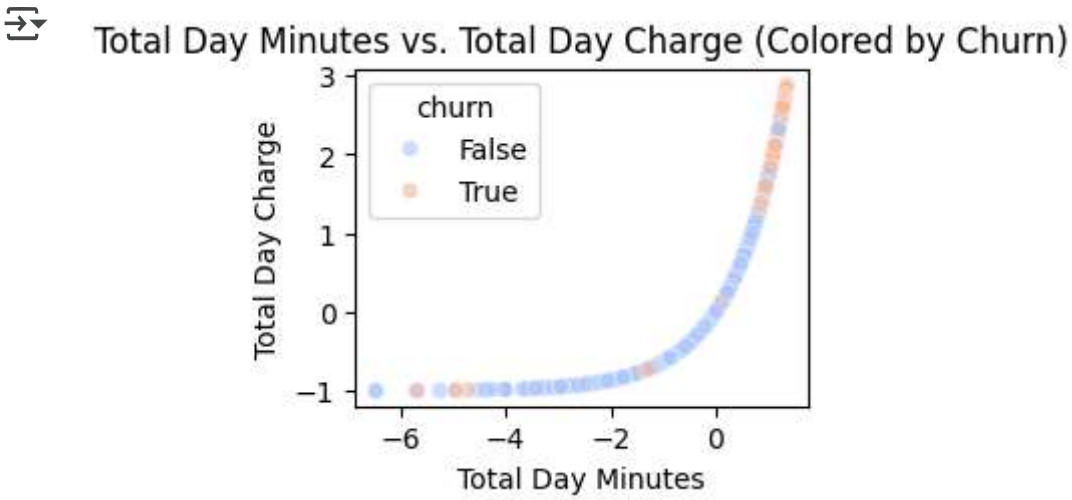
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.countplot(x='international plan', hue='churn', data=df_churn, palette='coolwarm', ax=axe
axes[0].set_title('Churn Distribution by International Plan')
axes[0].set_xlabel('International Plan (0=No, 1=Yes)')
axes[0].legend(['Not Churned', 'Churned'])

sns.countplot(x='voice mail plan', hue='churn', data=df_churn, palette='coolwarm', ax=axes[1
axes[1].set_title('Churn Distribution by Voice Mail Plan')
axes[1].set_xlabel('Voice Mail Plan (0=No, 1=Yes)')
axes[1].legend(['Not Churned', 'Churned'])

plt.show()
```



- Customers with an international plan churn more than those without it.
- Voice mail plan does not significantly affect churn..

```
# bivariate analysis visualization churn vs total day minutes
plt.subplot(2, 2, 1)
```

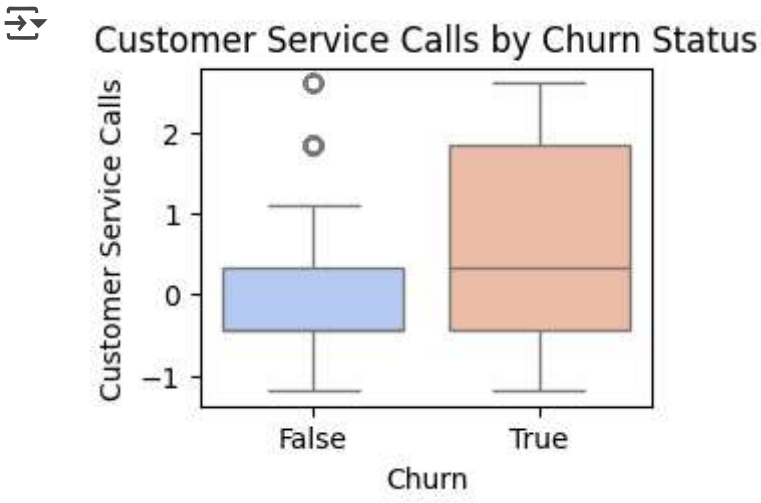
```
sns.scatterplot(x=df_churn["total day minutes"], y=df_churn["total day charge"], hue=df_chur
plt.title("Total Day Minutes vs. Total Day Charge (Colored by Churn)")
plt.xlabel("Total Day Minutes")
plt.ylabel("Total Day Charge")
plt.show()
```



Churn vs. Customer Service Calls

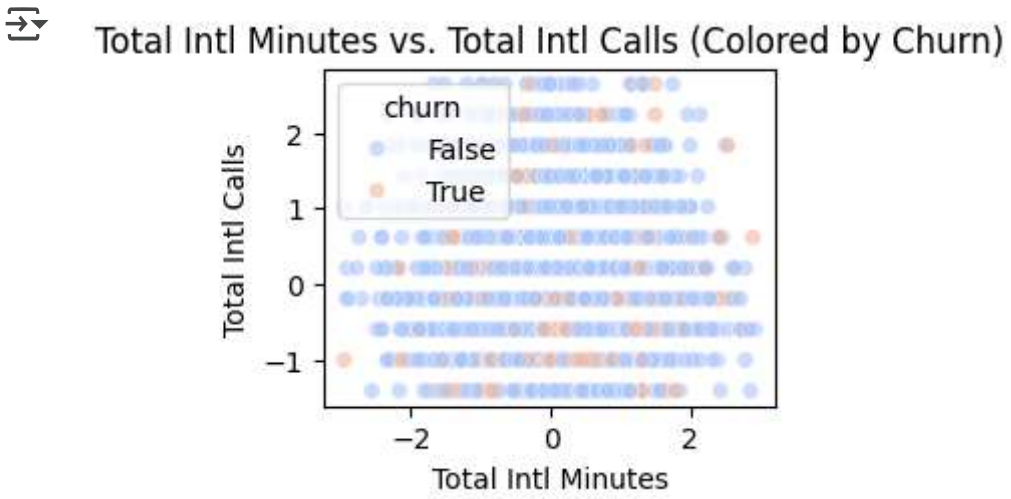
- Do customers who make more service calls churn more?

```
#Churn vs Customer Service Calls
plt.subplot(2, 2, 2)
sns.boxplot(x="churn", y="customer service calls", data=df_churn, palette="coolwarm")
plt.title("Customer Service Calls by Churn Status")
plt.xlabel("Churn")
plt.ylabel("Customer Service Calls")
plt.show()
```



- Customers who make more customer service calls tend to churn at a higher rate.
- Frequent service calls may indicate dissatisfaction with the service.

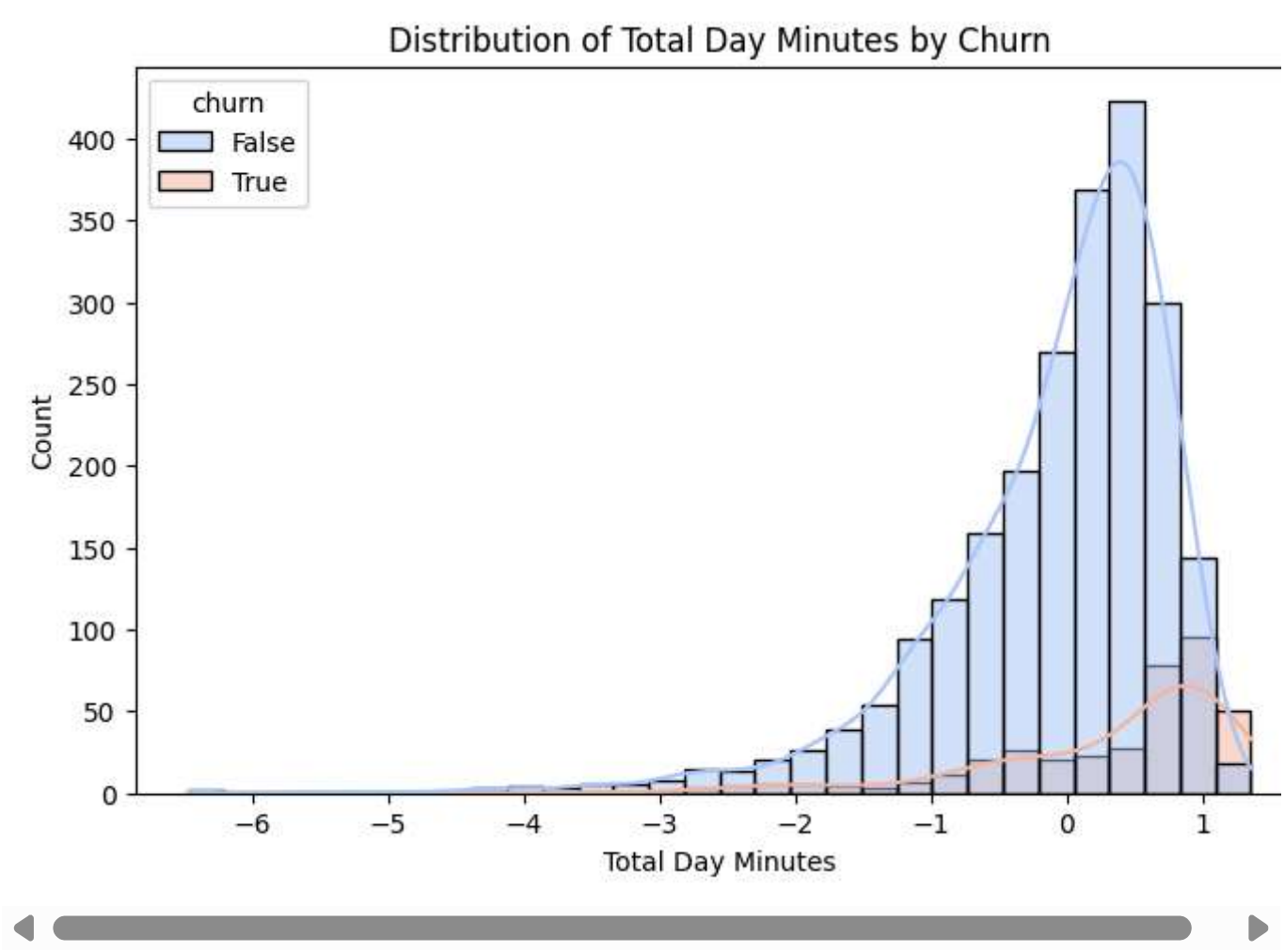
```
#  Churn vs Total Intl Minutes & Total Intl Calls
plt.subplot(2, 2, 3)
sns.scatterplot(x=df_churn["total intl minutes"], y=df_churn["total intl calls"], hue=df_chu
plt.title("Total Intl Minutes vs. Total Intl Calls (Colored by Churn)")
plt.xlabel("Total Intl Minutes")
plt.ylabel("Total Intl Calls")
plt.show()
```



Churn vs. Total Day Minutes

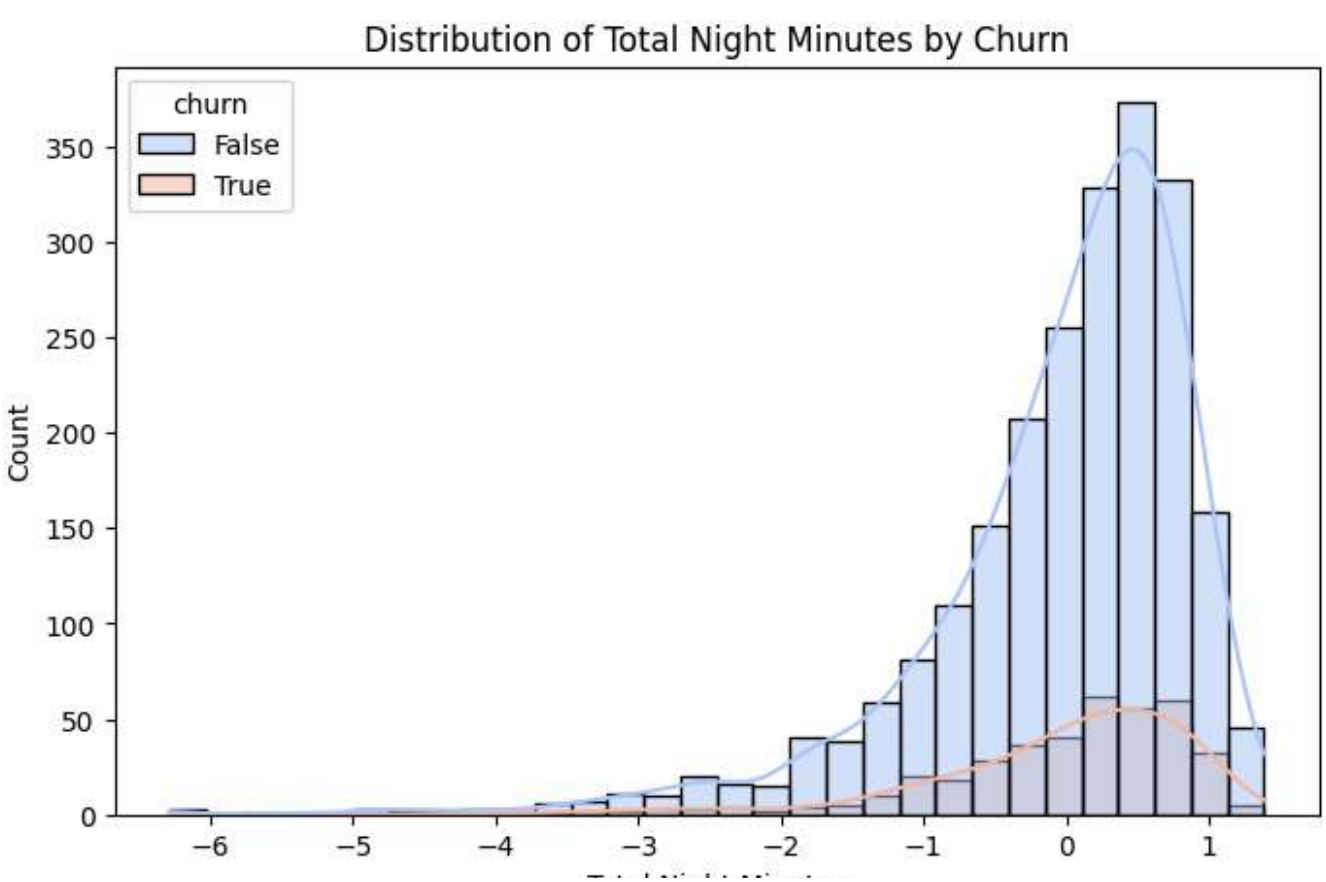
- Do customers who talk more during the day churn more?

```
plt.figure(figsize=(8,5))
sns.histplot(data=df_churn, x='total day minutes', hue='churn', kde=True, bins=30, palette='
plt.title('Distribution of Total Day Minutes by Churn')
plt.xlabel('Total Day Minutes')
plt.ylabel('Count')
plt.show()
```



- The histogram shows a higher density of churned customers in the upper range of total day minutes. Customers who spend more time on calls during the day tend to churn more. This means that high usage customers may be more sensitive to service quality or pricing issues.syriaTel should find ways to customer retention.

```
# A histogram of relationship between total night minutes and churn
plt.figure(figsize=(8,5))
sns.histplot(data=df_churn, x='total night minutes', hue='churn', kde=True, bins=30, palette
plt.title('Distribution of Total Night Minutes by Churn')
plt.xlabel('Total Night Minutes')
plt.ylabel('Count')
plt.show()
```

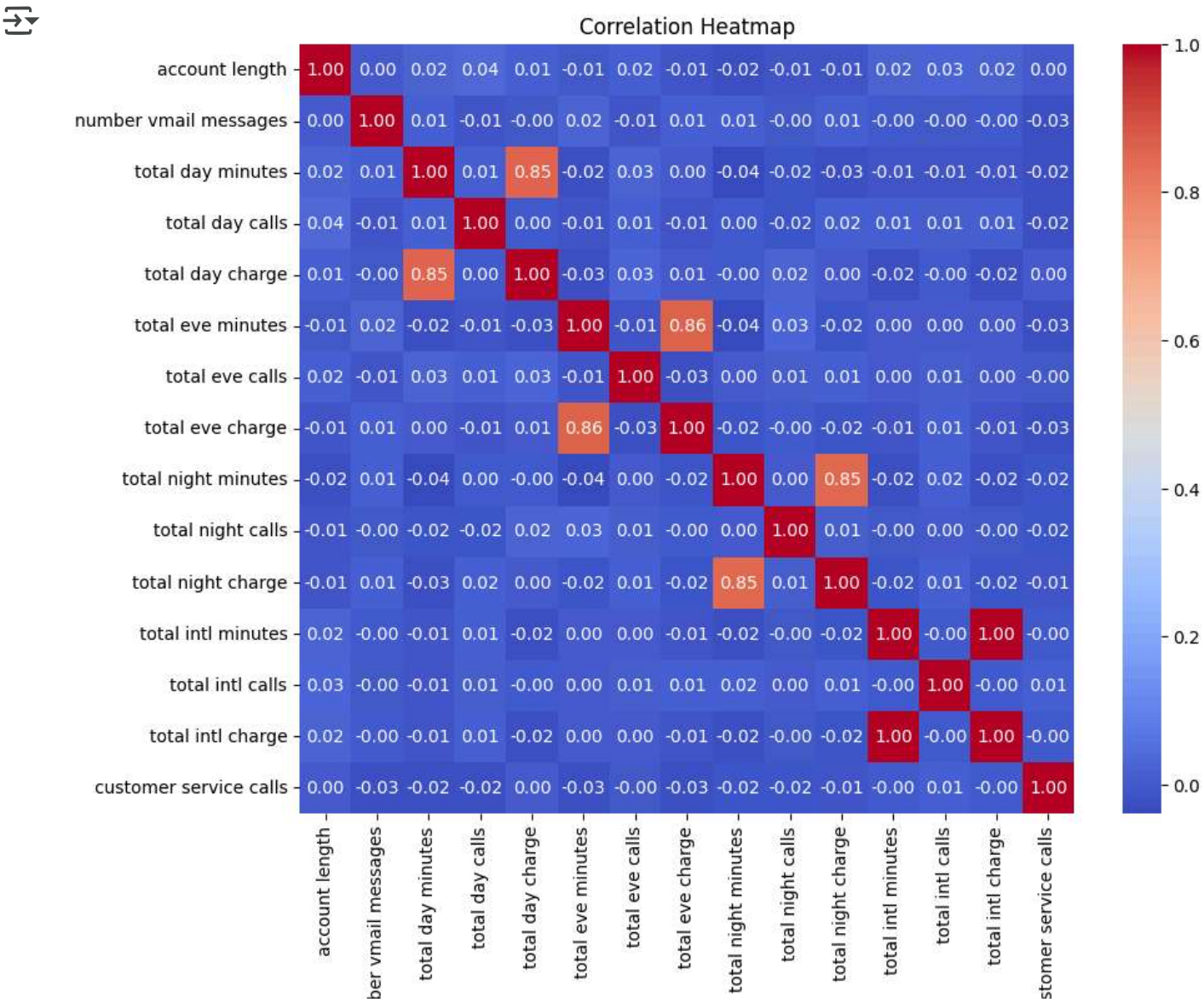


- The distribution of churned and non-churned customers is very similar, meaning night-time call usage is not a strong predictor of churn

Multivariate Analysis (Multiple Variables)

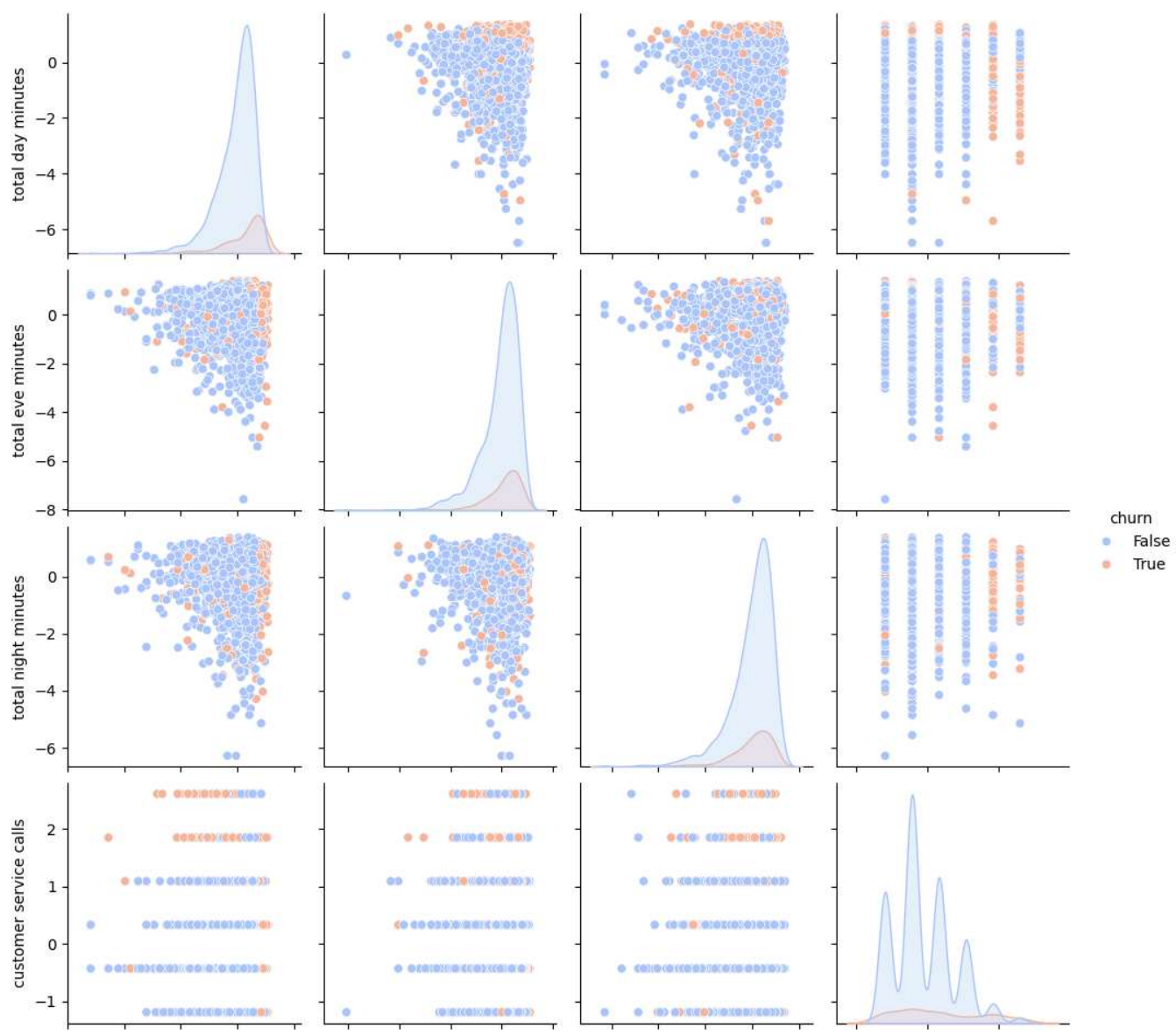
- To understand the relationship between multiple variables to see how they interact together.

```
# correlation heatmap plot
plt.figure(figsize=(10,8))

# Exclude non-numeric columns from correlation calculation
numerical_df = df_churn.select_dtypes(include=['number'])
sns.heatmap(numerical_df.corr(), annot=True, cmap='coolwarm', fmt =".2f")
plt.title('Correlation Heatmap')
plt.show()
```



- The correlation heatmap shows total day charge and total day minutes are highly correlated (almost 1.0), indicating they provide similar information. One of these features can be dropped to avoid multicollinearity in the model.
- Other features like customer service calls and churn have a positive correlation, meaning higher calls indicate higher churn.

```
# A pairplot of important features
selected_features = ['total day minutes', 'total eve minutes', 'total night minutes', 'custc
sns.pairplot(df_churn[selected_features], hue='churn', palette='coolwarm')
plt.show()
```



- There is a visible separation in total day minutes and customer service calls for churned and non-churned customers. Customers who make many service calls and use many day minutes tend to churn. when designing retention strategies these key features should be focused on

EDA Summary

- Churn Distribution- Churned customers are fewer, so we have class imbalance.
- International Plan- Customers with an international plan churn more.
- Customer Service Calls- More calls to customer service → Higher churn (dissatisfaction).
- Total Day Minutes- High usage during the day → Higher churn.
- Feature Correlation- Some features are redundant (total day minutes & total day charge).

Encoding

```
# Here we get to use either one hot encoding or label encoding since we are converting categ
# label encoding
df_churn['international plan']= df_churn['international plan'].map({'yes': 1, 'no': 0})
df_churn['voice mail plan'] = df_churn['voice mail plan'].map({'yes': 1, 'no': 0})

# Check if 'state' column exists before applying get_dummies
if 'state' in df_churn.columns:
    df_churn = pd.get_dummies(df_churn, columns=['state'], drop_first=True)
else:
    print("Column 'state' not found in the DataFrame. Skipping one-hot encoding.")

#create new features to improve predictive power
df_churn['day_call_ratio'] = df_churn['total day calls'] / df_churn['total day minutes']
df_churn['eve_call_ratio'] = df_churn['total eve calls'] / df_churn['total eve minutes']
df_churn['night_call_ratio'] = df_churn['total night calls'] / df_churn['total night minutes
df_churn['total_calls'] = df_churn['total day calls'] + df_churn['total eve calls'] + df_chu
df_churn['total minutes'] = df_churn['total day minutes'] + df_churn['total eve minutes'] +

df_churn['high_service_calls'] = df_churn['customer service calls'].apply(lambda x: 1 if x >
df_churn['high_service_calls'] = (df_churn['customer service calls'] > 3).astype(int)
```

```
Column 'state' not found in the DataFrame. Skipping one-hot encoding.
```

Feature selection

- Feature selection improves model efficiency through abandoning of multicollinearity

```
# identify the column with problematic value '382-4657'
problematic_column = df_churn.apply(lambda x: x.astype(str).str.contains('382-4657').any()).
print(f"Problematic column: {problematic_column}")

# Check if the problematic column exists before dropping
if problematic_column in df_churn.columns:
    # drop the problematic column
    df_churn = df_churn.drop(columns=[problematic_column])
else:
    print(f"Column '{problematic_column}' not found in DataFrame, skipping drop.")
```

```
Problematic column: <bound method Series.idxmax of account length           False
international plan        False
voice mail plan           False
number vmail messages     False
total day minutes         False
total day calls           False
total day charge          False
total eve minutes         False
total eve calls           False
total eve charge          False
```

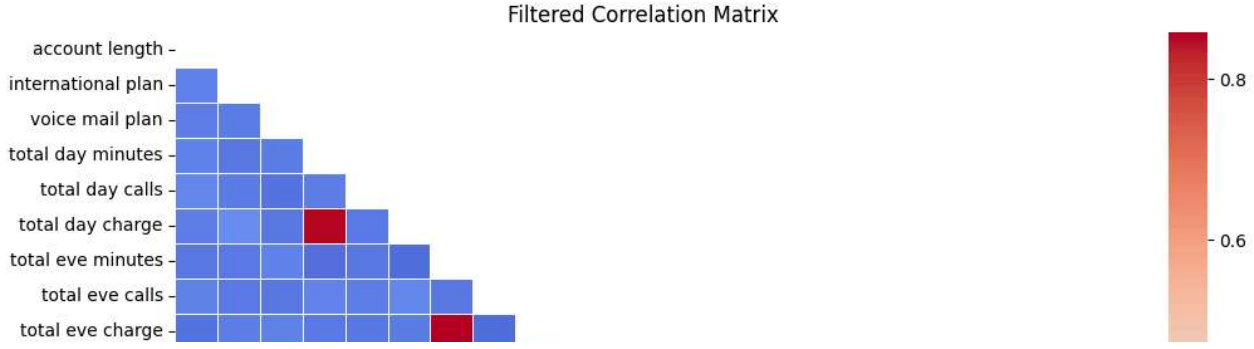
```
total night minutes       False
total night calls         False
total night charge        False
total intl minutes        False
total intl calls          False
total intl charge         False
customer service calls    False
churn                     False
day_call_ratio            False
eve_call_ratio            False
night_call_ratio          False
total_calls               False
total minutes             False
high_service_calls        False
dtype: bool>
Column '<bound method Series.idxmax of account length            False
international plan        False
voice mail plan           False
number vmail messages     False
total day minutes         False
total day calls           False
total day charge          False
total eve minutes         False
total eve calls           False
total eve charge          False
total night minutes       False
total night calls         False
total night charge        False
total intl minutes        False
total intl calls          False
total intl charge         False
customer service calls    False
churn                     False
day_call_ratio            False
eve_call_ratio            False
night_call_ratio          False
total_calls               False
total minutes             False
high_service_calls        False
dtype: bool>' not found in DataFrame, skipping drop.
```

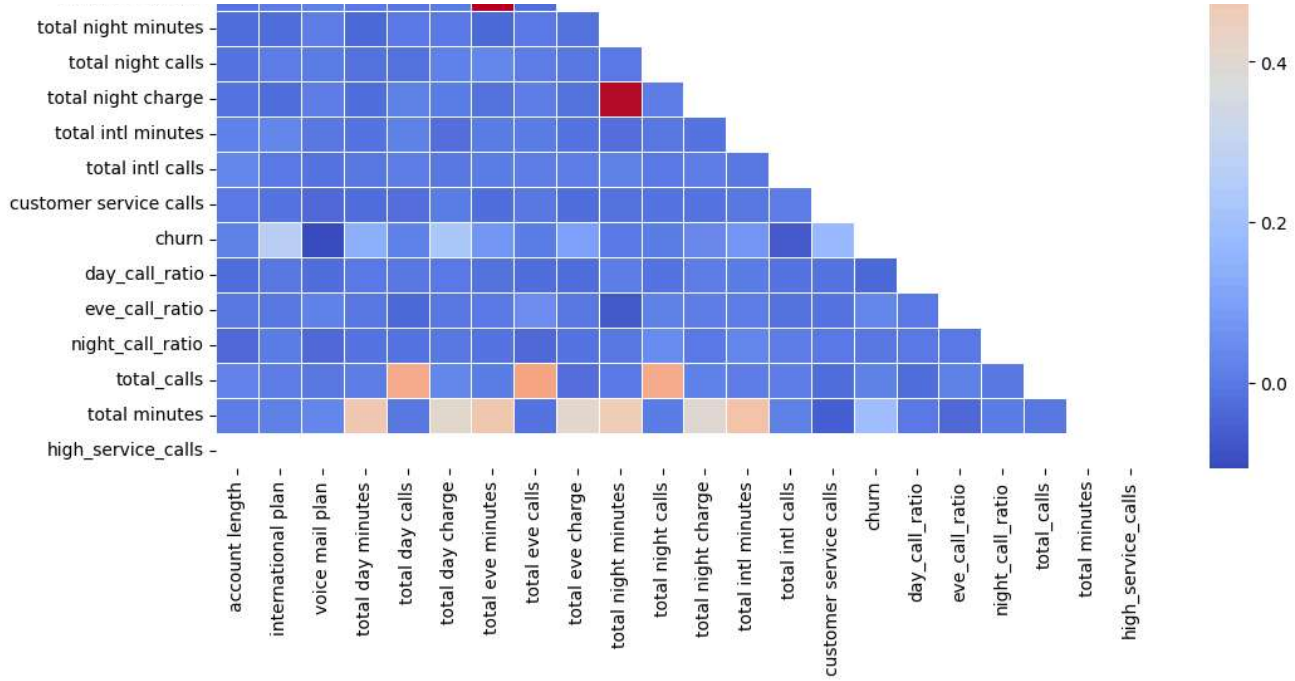
```
# correlation matrix, excluding non-numerical column
corr_matrix = df_churn.select_dtypes(include=np.number).corr().abs()

 # Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

 # Find features with correlation greater than 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
print("Dropping these features due to high correlation:", to_drop)

 # Drop highly redundant features
df_churn = df_churn.drop(columns=to_drop)
```

```
Dropping these features due to high correlation: ['number vmail messages', 'total intl c
```

```
# identify the column with problematic value '382-4657'
problematic_column = df_churn.apply(lambda x: x.astype(str).str.contains('382-4657').any()).
print(f"Problematic column: {problematic_column}")

# Cornfirmed if the problematic column exists before dropping
if problematic_column in df_churn.columns:

    # drop the problematic column
    df_churn = df_churn.drop(columns=[problematic_column])
else:
    print(f"Column '{problematic_column}' not found in DataFrame, skipping drop.")

# computed correlation matrix, excluding non-numerical columns
corr = df_churn.corr()

# upper triangle was masked for better visualization
mask = np.triu(np.ones_like(corr, dtype=bool))

plt.figure(figsize=(12, 8))
# plot the heatmap
sns.heatmap(corr, mask=mask, cmap="coolwarm", annot=False, fmt=".2f", linewidths=0.5)
plt.title("Filtered Correlation Matrix")
plt.show()
```

```
Problematic column: <bound method Series.idxmax of account length           False
international plan        False
voice mail plan           False
total day minutes         False
total day calls           False
total day charge          False
total eve minutes         False
total eve calls           False
total eve charge          False
total night minutes       False
total night calls         False
total night charge        False
total intl minutes        False
total intl calls          False
customer service calls    False
churn                     False
day_call_ratio            False
eve_call_ratio            False
night_call_ratio          False
total_calls               False
total minutes             False
high_service_calls        False
dtype: bool>
Column '<bound method Series.idxmax of account length           False
international plan        False
voice mail plan           False
total day minutes         False
total day calls           False
total day charge          False
total eve minutes         False
total eve calls           False
total eve charge          False
total night minutes       False
total night calls         False
total night charge        False
total intl minutes        False
total intl calls          False
customer service calls    False
churn                     False
day_call_ratio            False
eve_call_ratio            False
night_call_ratio          False
total_calls               False
total minutes             False
high_service_calls        False
dtype: bool>' not found in DataFrame, skipping drop.
```






```
# Create a feature flag for high customer service calls (Threshold = 4)
df_churn['high_service_calls'] = (df_churn['customer service calls'] > 4).astype(int)

# Create a total usage feature
df_churn['total usage minutes'] = df_churn['total day minutes'] + df_churn['total eve minute

# Verify
df_churn[['customer service calls', 'high_service_calls', 'total usage minutes']].head()
```

	customer service calls	high_service_calls	total usage minutes
0	-0.427932	0	1.493616
1	-0.427932	0	0.201561
2	-1.188218	0	NaN
3	0.332354	0	NaN
4	1.092641	0	NaN

Creating New Features

- Customers with high customer service calls churn more → Create a feature to flag high complaints.
- Customers with high usage (day/eve/night minutes) churn more → Create a total minutes used feature.

Action is to:

- Flag High Customer Service Callers (Threshold = 4 calls) Total Minutes Feature (total usage minutes)

Feature scaling

To normalize KNN model standard Scaler values between 0-1

```
# Select numerical features for scaling
num_features = ['account length', 'total day minutes', 'total eve minutes',
                'total night minutes', 'customer service calls']

# Apply Standard Scaling
scaler = StandardScaler()
df_churn[num_features] = scaler.fit_transform(df_churn[num_features])

# Verify
df_churn.head()
```

	account length	international plan	voice mail plan	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	
0	0.687718	0	1	1.059830	0.476643	1.567036	-0.024790	-0.055940	-0.
1	0.155649	0	1	-0.407515	1.124503	-0.334013	-0.068776	0.144867	-0.
2	0.915748	0	0	0.876290	0.675985	1.168464	NaN	0.496279	-1.
3	-0.427092	1	0	1.298570	-1.466936	2.196759	NaN	-0.608159	-2.
4	-0.655122	1	0	-0.264430	0.626149	-0.240041	NaN	1.098699	-1.

5 rows × 23 columns

```
print("Final Dataset Shape:", df_churn.shape)
df_churn.head()
```

```
Final Dataset Shape: (3169, 23)
```

	account length	international plan	voice mail plan	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	
0	0.687718	0	1	1.059830	0.476643	1.567036	-0.024790	-0.055940	-0.
1	0.155649	0	1	-0.407515	1.124503	-0.334013	-0.068776	0.144867	-0.
2	0.915748	0	0	0.876290	0.675985	1.168464	NaN	0.496279	-1.
3	-0.427092	1	0	1.298570	-1.466936	2.196759	NaN	-0.608159	-2.
4	-0.655122	1	0	-0.264430	0.626149	-0.240041	NaN	1.098699	-1.

5 rows × 23 columns

Modeling

Regression

```
 # Reset index to avoid issues after removing rows for outliers
df_churn = df_churn.reset_index(drop=True)

  # Define X and y
 X = df_churn.drop(columns=['churn'])
 y = df_churn['churn']

 # Identify object columns
 object_columns = X.select_dtypes(include=['object']).columns

 # One-hot encode object columns
if len(object_columns) > 0:
    X = pd.get_dummies(X, columns=object_columns, drop_first=True)

 # Add a constant to the independent variables
X = sm.add_constant(X)

# Convert boolean columns to integers
X = X.astype({col: 'int' for col in X.select_dtypes(include=['bool']).columns})

 # Drop columns containing phone number
X = X.drop(columns=[col for col in X.columns if 'phone number' in col], errors='ignore')

# One-hot encoding for remaining categorical columns
# Check and convert columns with 'object' dtype to numeric before applying get_dummies
for col in X.select_dtypes(include=['object']).columns:
```

```
    X[col] = pd.to_numeric(X[col], errors='coerce')
X = pd.get_dummies(X, drop_first=True, dummy_na=False)

 # Convert to numeric, replacing inf and NaN with a large and small number
X = X.apply(pd.to_numeric, errors='coerce').replace([np.inf, -np.inf], np.nan)
X.fillna(X.mean(), inplace=True)
print(X.isnull().sum().sum())
print(X.isnull().sum())
```

```
0
const                     0
account length            0
international plan        0
voice mail plan           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
customer service calls    0
day_call_ratio            0
eve_call_ratio            0
night_call_ratio          0
total_calls               0
total minutes             0
high_service_calls        0
total usage minutes       0
dtype: int64
```

```
X = X.apply(pd.to_numeric, errors='coerce')
X.fillna(X.mean(), inplace=True)
print("Remaining NaNs:", X.isnull().sum().sum())
print("Remaining Infs:", np.isinf(X).sum().sum())
print("Data Types:", X.dtypes.unique())
```

```
Remaining NaNs: 0
Remaining Infs: 0
Data Types: [dtype('float64') dtype('int64')]
```

```
#Fit the linear regression model
model = sm.OLS(y, X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  churn   R-squared:                       0.187
Model:                            OLS   Adj. R-squared:                  0.182
```

```
Method:                 Least Squares   F-statistic:                     36.22
Date:                Wed, 12 Mar 2025   Prob (F-statistic):          1.29e-125
Time:                        17:15:53   Log-Likelihood:                -809.20
No. Observations:                3169   AIC:                             1660.
Df Residuals:                    3148   BIC:                             1788.
Df Model:                          20
Covariance Type:            nonrobust
================================================================================
                           coef    std err          t      P>|t|      [0.025      0.97
--------------------------------------------------------------------------------
const                    0.1359      0.007     19.767      0.000       0.122       0.1
account length           0.0032      0.006      0.566      0.571      -0.008       0.0
international plan       0.2957      0.019     15.681      0.000       0.259       0.3
voice mail plan         -0.0823      0.013     -6.577      0.000      -0.107      -0.0
total day minutes       -0.0260      0.010     -2.652      0.008      -0.045      -0.0
total day calls          0.0057      0.005      1.210      0.226      -0.004       0.0
total day charge         0.0702      0.007      9.656      0.000       0.056       0.0
total eve minutes       -0.0212      0.010     -2.187      0.029      -0.040      -0.0
total eve calls         -0.0015      0.005     -0.307      0.759      -0.011       0.0
total eve charge         0.0304      0.007      4.164      0.000       0.016       0.0
total night minutes     -0.0396      0.010     -4.107      0.000      -0.059      -0.0
total night calls       -0.0019      0.005     -0.413      0.680      -0.011       0.0
total night charge       0.0229      0.007      3.172      0.002       0.009       0.0
total intl minutes       0.0172      0.009      1.845      0.065      -0.001       0.0
total intl calls        -0.0285      0.006     -4.481      0.000      -0.041      -0.0
customer service calls   0.0631      0.006     11.305      0.000       0.052       0.0
day_call_ratio          -0.0002   9.12e-05     -2.234      0.026      -0.000   -2.49e-
eve_call_ratio           0.0001   4.73e-05      2.480      0.013    2.46e-05       0.0
night_call_ratio    -1.581e-05    8.9e-05     -0.178      0.859      -0.000       0.0
total_calls              0.0023      0.002      0.940      0.347      -0.003       0.0
total minutes            0.0097      0.012      0.814      0.416      -0.014       0.0
high_service_calls            0          0        nan        nan           0
total usage minutes      0.0372      0.015      2.527      0.012       0.008       0.0
==============================================================================
Omnibus:                      820.425   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1685.475
Skew:                           1.539   Prob(JB):                         0.00
Kurtosis:                       4.813   Cond. No.                     1.71e+35
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
[2] The smallest eigenvalue is 1.53e-63. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
#calculate y_pred
y_pred = model.predict(X)

#Calculate the metrics
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y, y_pred)
r2 = r2_score(y, y_pred)
```

```
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R²: {r2}")
```

```
MSE: 0.09757053221722801
RMSE: 0.3123628214388326
MAE: 0.21414807540348993
R²: 0.18706514410613362
```

- The RMSE output means the model is making prediction errors of 31% on average
- An R² of 0.1870 that is 18.70% can be explained by the models features however they may not be strong predictors of churn.
- MAE suggests that the model on average is 21% off from the true values of churn
    - The above shows that churn is a binary classification problem not continuous hence the model is not capturing enough variance

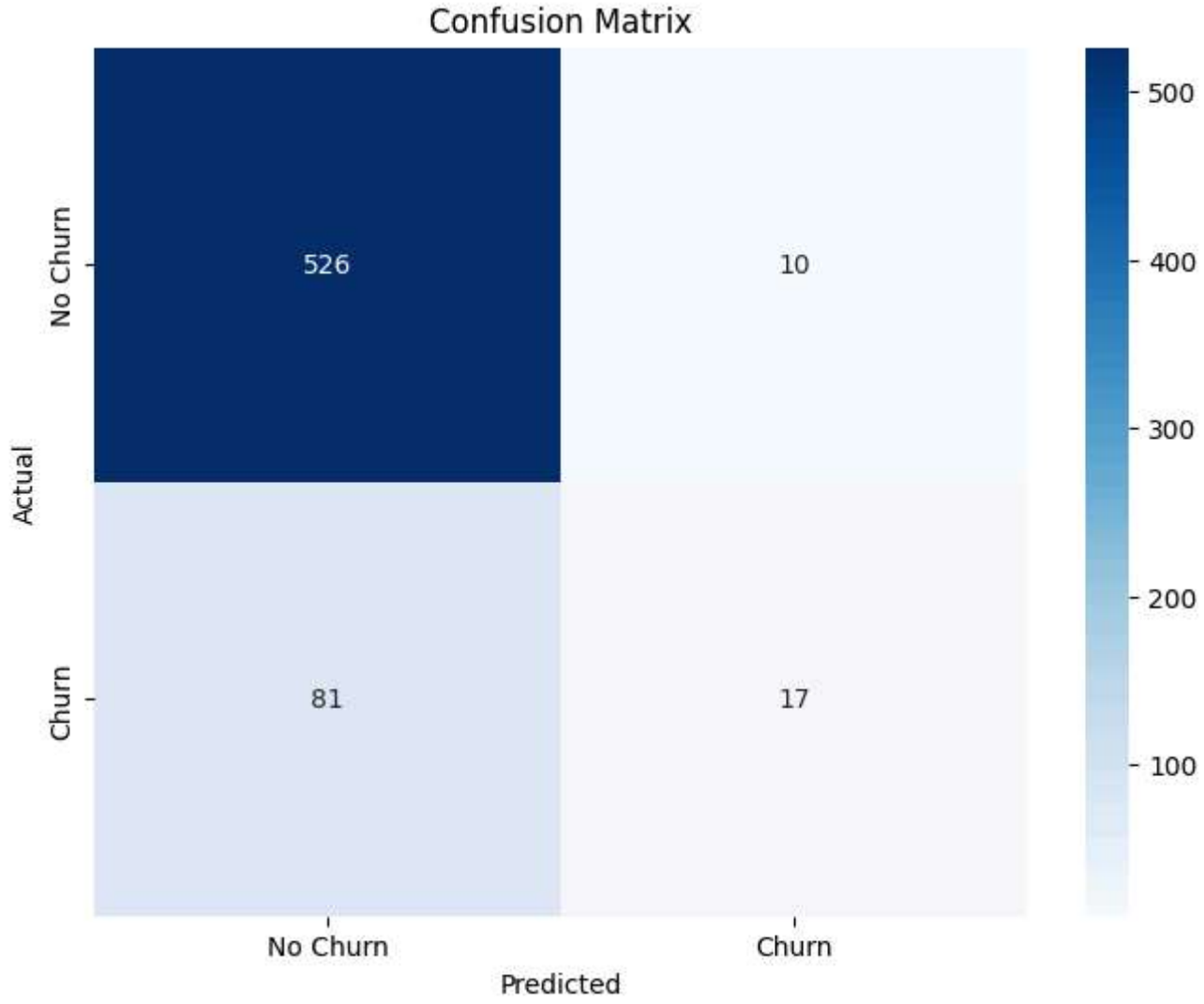
Classification

Logistic Regression

```
 #Define features X and target y
X = df_churn.drop(columns=['churn'])
y = df_churn['churn']

 #Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
 # Impute missing values using SimpleImputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')  # Replace NaNs with the mean of the column

 # Fit the imputer on the training data and transform both training and testing data
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

 #Initialize and train the logistic regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)

 #Make predictions on the test set
y_pred = log_reg_model.predict(X_test)

 #Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

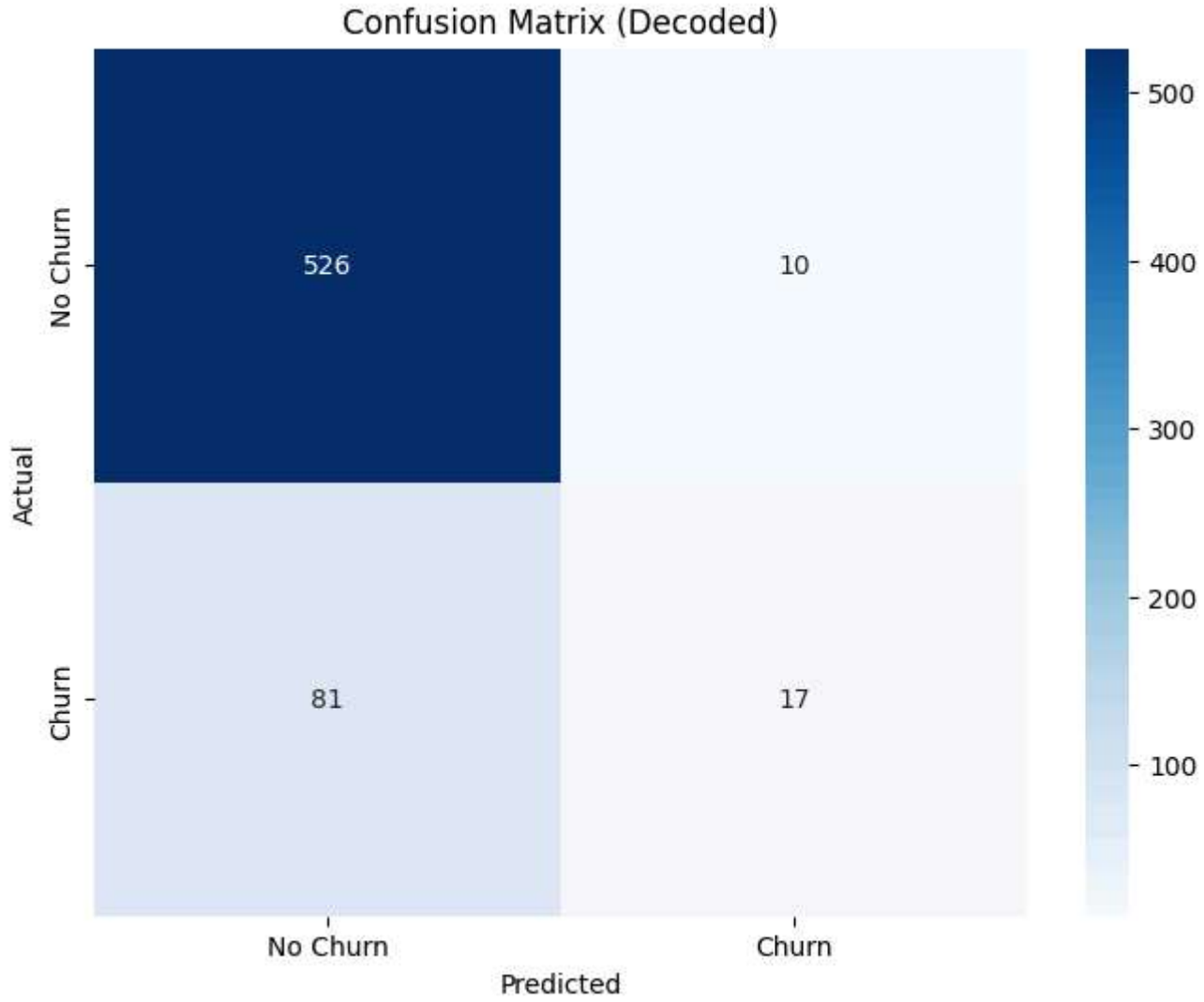
```
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

 #Classification report
print(classification_report(y_test, y_pred))

#Plot a confusion matrix
 cm = confusion_matrix(y_test, y_pred)
 plt.figure(figsize=(8, 6))
 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick
 plt.title('Confusion Matrix')
 plt.xlabel('Predicted')
 plt.ylabel('Actual')
 plt.show()
```

```
Accuracy: 0.8564668769716088
Recall: 0.17346938775510204
Precision: 0.6296296296296297
F1 Score: 0.272
              precision    recall  f1-score   support

       False       0.87      0.98      0.92       536
        True       0.63      0.17      0.27        98

    accuracy                           0.86       634
   macro avg       0.75      0.58      0.60       634
weighted avg       0.83      0.86      0.82       634
```




```
#Decoded the predictions and true labels
y_pred_decoded = np.where(y_pred == 1, 'Churn', 'No Churn')
y_test_decoded = np.where(y_test == 1, 'Churn', 'No Churn')

#plot the confusion matrix with decoded labels
cm= confusion_matrix(y_test_decoded, y_pred_decoded, labels=['No Churn', 'Churn'])
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick
plt.title('Confusion Matrix (Decoded)')
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



- From logistic regression model results a good accuracy of 86%
- Recall of 17% which means theres a low amount of people who churn
- A low F1 score showing poor balance
- Precision of 63% which is moderate predicted customers predicted actually churned

```
#Tune the logistic model

#Define the parameter grid to search
param_grid = {
'C': [0.001, 0.01, 0.1, 1, 10, 100],'penalty': ['l1', 'l2'], 'solver': ['liblinear', 'saga'
}
#Initialize GridSearchCV
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='accuracy')

#Fit the grid search to the data
grid_search.fit(X_train, y_train)
```

```
#Get the best hyperparameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_score}")

#Train a new model with the best hyperparameters
best_model = LogisticRegression(**best_params)
best_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = best_model.predict(X_test)

#Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

#classification report
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'C': 0.001, 'penalty': 'l2', 'solver': 'liblinear'}
Best Accuracy: 0.876923076923077
              precision    recall  f1-score   support

       False       0.86      1.00      0.92       536
        True       0.80      0.08      0.15        98

    accuracy                           0.85       634
   macro avg       0.83      0.54      0.53       634
weighted avg       0.85      0.85      0.80       634
```

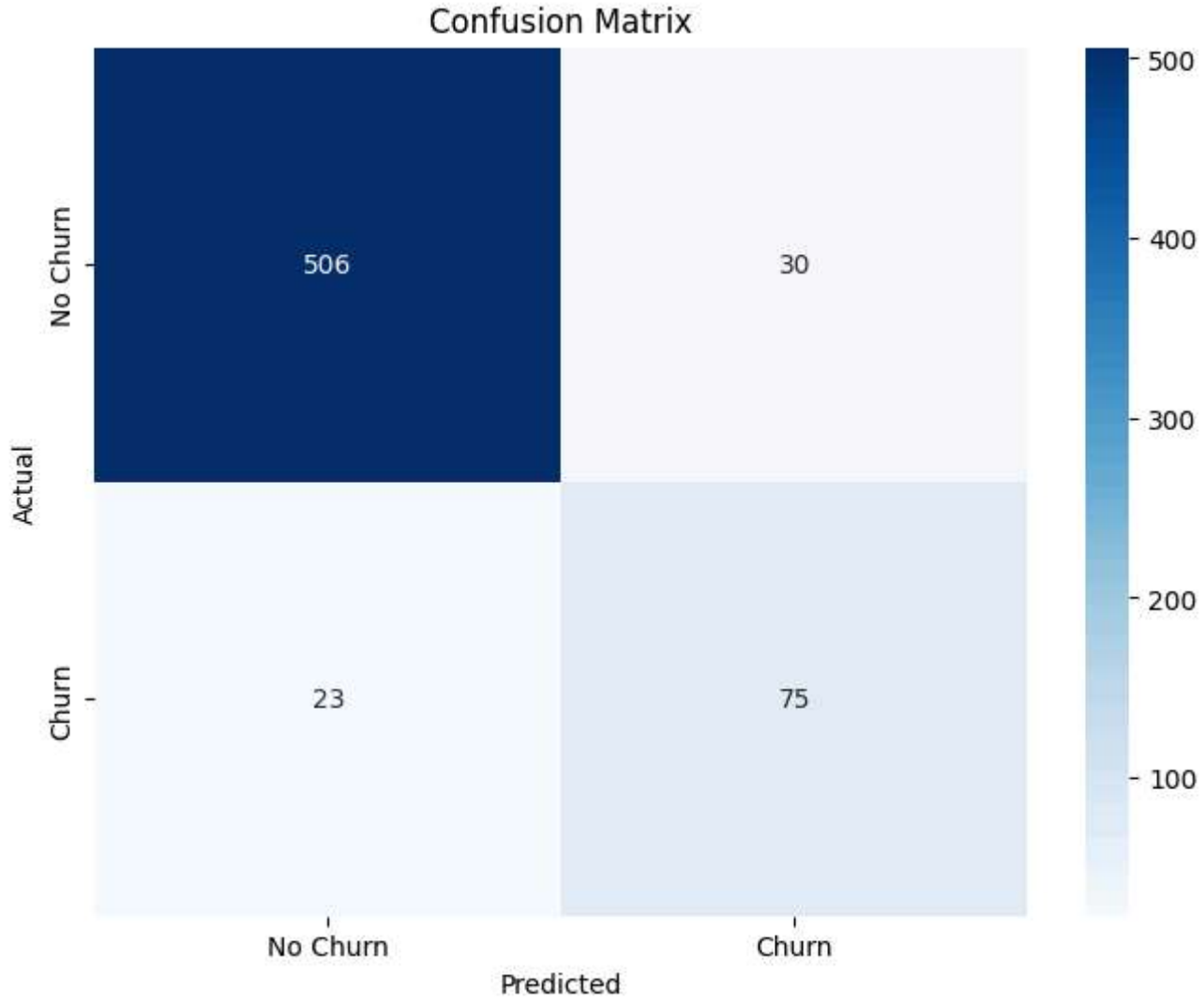
- Accuracy improved to 88% which is remarkable
  - Penalty L2 which is Ridge regression helps prevent overfitting
  - C= 0.001 controls regularization and smaller values give stronger regularization

Decision Tree

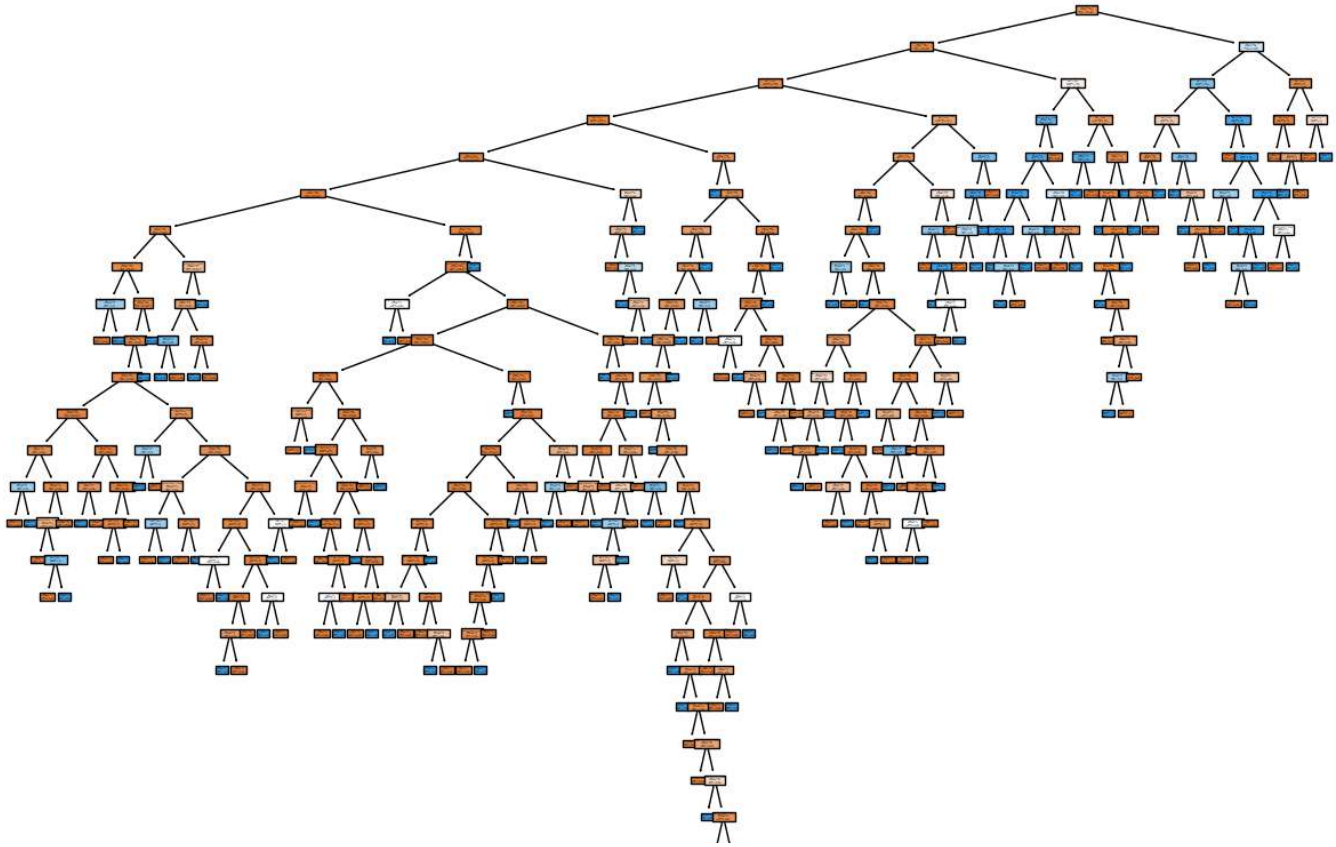
```
#Initialize and train the decision tree classsifier
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)

#Make predictions on the test set
```

```
y_pred = dt_model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

#Classification report
print(classification_report(y_test, y_pred))

#Plot a confusion matrix

cm= confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.916403785488959
Recall: 0.7653061224489796
Precision: 0.7142857142857143
F1 Score: 0.7389162561576355
              precision    recall  f1-score   support

       False       0.96      0.94      0.95       536
        True       0.71      0.77      0.74        98

    accuracy                           0.92       634
   macro avg       0.84      0.85      0.84       634
weighted avg       0.92      0.92      0.92       634
```




```
#Plot decision tree
#Assuming dt_model is your trained DecisionTreeClassifier
plt.figure(figsize=(15, 10))
plot_tree(dt_model, filled=True, feature_names=X.columns, class_names=['No Churn', 'Churn']
plt.show()
```



- Accuracy is at 92% which means the model is performing well
- Recall is at 77% which means the model is getting 77% of actual churn hence missing fewer churners
- Precision is at 71% as well which means when the model predicts churn it is correct 71% of the time
- F1-score is at 74% as well showing a well optimized model for predicting churn

```
#Tune the decision tree model
#Define the parameter grid
param_grid = {
'max_depth': [None, 10, 20, 30],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4]
}
#Initialize GridSearchCV
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring='accuracy')

#Fit the grid search to the data
grid_search.fit(X_train, y_train)

#Get the best hyperparameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Hyperparameters: {best_params}")
```

```
print(f"Best Accuracy: {best_score}")

#Train a new model with the best hyperparameters
best_model = DecisionTreeClassifier(**best_params)
best_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = best_model.predict(X_test)

#Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

#Print classification report
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10}
Best Accuracy: 0.944378698224852
              precision    recall  f1-score   support

       False       0.96      0.97      0.97       536
        True       0.83      0.79      0.81        98

    accuracy                           0.94       634
   macro avg       0.89      0.88      0.89       634
weighted avg       0.94      0.94      0.94       634
```

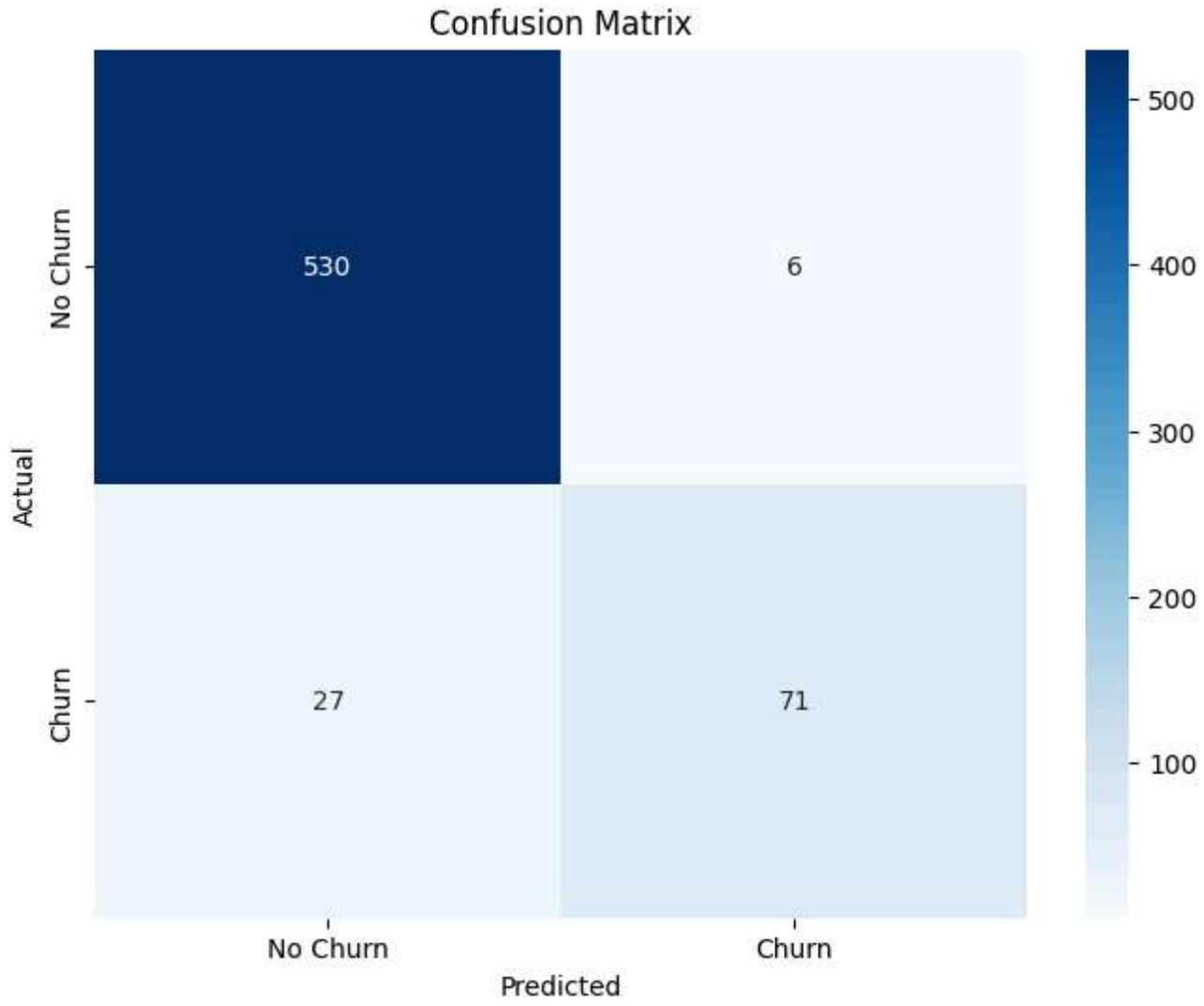
- After tuning the model improved its accuracy to 94%
- Precision did improve as well up to 83%
- Recall moved to 79%
- F1-score moved up to 81% showing a balance of precision and recall

Random Forest

```
#Initialize and train the random forest classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = rf_model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
```

```
    f1 = f1_score(y_test, y_pred)

    print(f"Accuracy: {accuracy}")
    print(f"Recall: {recall}")
    print(f"Precision: {precision}")
    print(f"F1 Score: {f1}")

    #Classificattion report
    print(classification_report(y_test, y_pred))

    #Plot a confusion matrix
cm= confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick]
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.9479495268138801
Recall: 0.7244897959183674
Precision: 0.922077922077922
F1 Score: 0.8114285714285714
              precision    recall  f1-score   support

       False       0.95      0.99      0.97       536
        True       0.92      0.72      0.81        98

    accuracy                           0.95       634
   macro avg       0.94      0.86      0.89       634
weighted avg       0.95      0.95      0.95       634
```



- Accuracyhas 95% giving more correct predictions meaning the model is performing well
- A higher precision 92% meaning when it predicts churn it is highly correct. Means retention efforts are targeted at actual churners.
- A lower recall of 72% however showing the model is missing more churn cases leading to potentiial revenue loss.
- F1-score of 81% shows there is balance. Means the model is optimized for both precision and recall

```
#Tune the random forest model
param_grid = {
'n_estimators': [100, 200, 300],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4]
}
#Initialize a GridSearchCV
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='accuracy')
#Fit the grid search to the data
grid_search.fit(X_train, y_train)

 #Get the best hyperparameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_score}")

#Train a new model with the best hyperparameters
best_model = RandomForestClassifier(**best_params)
best_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = best_model.predict(X_test)

#Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

#Classification Report
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
Best Accuracy: 0.9526627218934911
              precision    recall  f1-score   support

       False       0.95      0.99      0.97       536
        True       0.91      0.72      0.81        98

    accuracy                           0.95       634
   macro avg       0.93      0.86      0.89       634
weighted avg       0.95      0.95      0.94       634
```

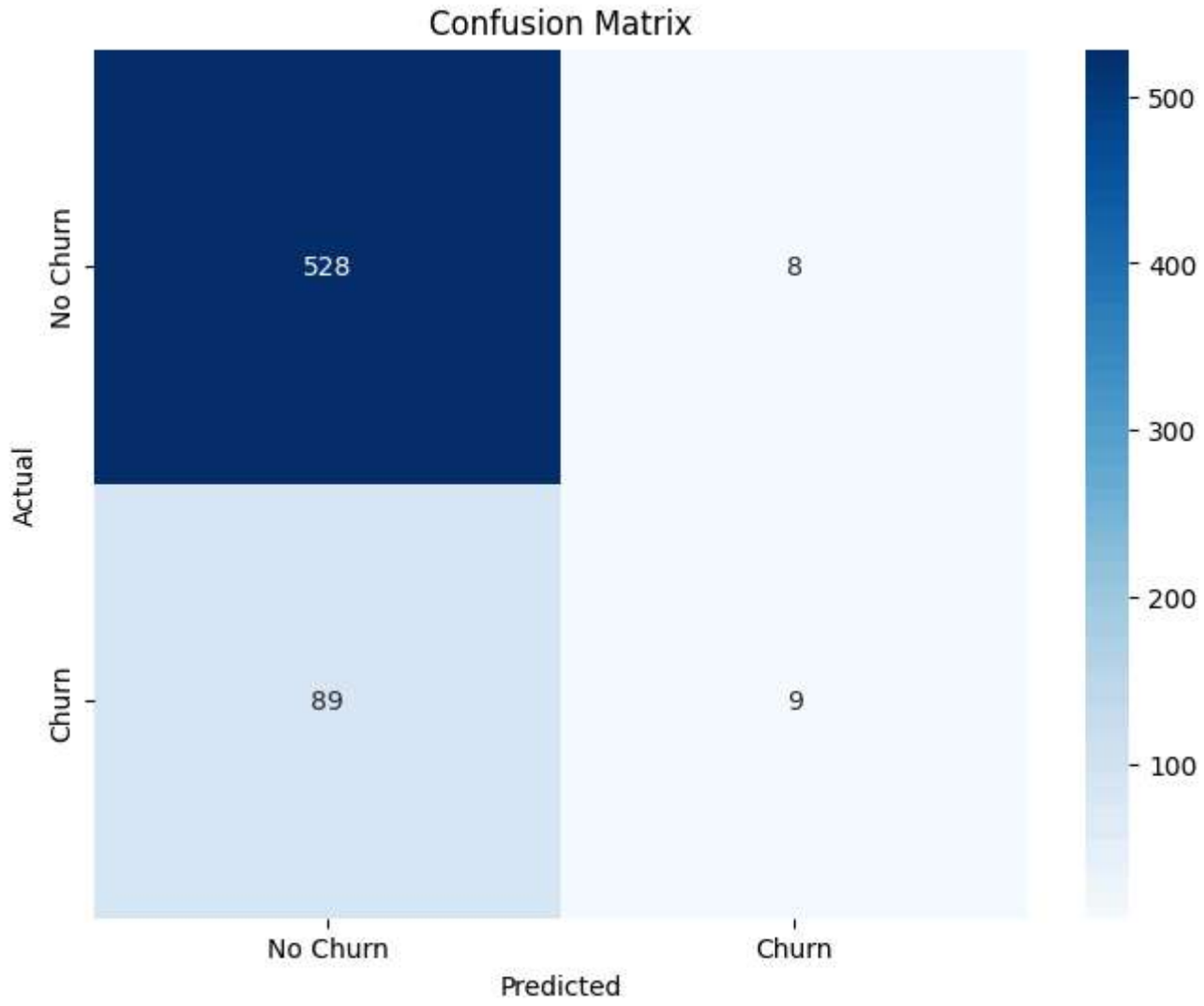
- After tuning the random forest the accuracy is at 95%

- Recall is at 72% which is lower thus the model misses 28% of the churn cases
- F1-score shows random forest is well balanced than the other models
- High precision of 91% may indicate that the model igores false alarms while predicting churn

K-NN model

```
#Initialize and train the K-NN classifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = knn_model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

#Classification report
print(classification_report(y_test, y_pred))

#Plot confusion matrix

cm= confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.8470031545741324
Recall: 0.09183673469387756
Precision: 0.5294117647058824
F1 Score: 0.1565217391304348
              precision    recall  f1-score   support

       False       0.86      0.99      0.92       536
        True       0.53      0.09      0.16        98

    accuracy                           0.85       634
   macro avg       0.69      0.54      0.54       634
weighted avg       0.81      0.85      0.80       634
```



- The accuracy looks good at 85% but might be misleading
- Precision is at 53% which means when predicting churn it is only correct 53% of the time
- 9% recall shows the model detects low churn cases
- F1-score 0f 16% shows very poor balance between precision and recall

```
#Tune the K-NN model
param_grid = {
```

```
'n_neighbors': [3, 5, 7, 9],
'weights': ['uniform', 'distance'],
'p': [1, 2]
}
#Initialize GridSearchCV
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')

#Fit the grid search to the data
grid_search.fit(X_train, y_train)

#Get the best hyperparameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_score}")

#Train a new model with the best hyperparameters
best_model = KNeighborsClassifier(**best_params)
best_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = best_model.predict(X_test)

#Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

#Classification report
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
Best Accuracy: 0.8757396449704142
Accuracy: 0.8564668769716088
Recall: 0.08163265306122448
Precision: 0.8888888888888888
F1 Score: 0.14953271028037382
              precision    recall  f1-score   support

       False       0.86      1.00      0.92       536
        True       0.89      0.08      0.15        98

    accuracy                           0.86       634
   macro avg       0.87      0.54      0.54       634
weighted avg       0.86      0.86      0.80       634
```

- Accuracy of 88% is good
- Precision of 86% is a better which shows predicts 82% correctly
- recall of 8% means the model has a lower detection for churn
- f1 score 15% there is a poor balance between precision and recall

```
#Tune the K-NN model
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

#Initialize GridSearchCV
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')

#Fit the grid search to the data
grid_search.fit(X_train, y_train)

#Get the best hyperparameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Hyperparameters: {best_params}")
print(f"Best Accuracy: {best_score}")

#Train a new model with the best hyperparameters
best_model = KNeighborsClassifier(**best_params)
best_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = best_model.predict(X_test)

#Evaluate the tuned model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

#Classification report
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
Best Accuracy: 0.8757396449704142
Accuracy: 0.8564668769716088
Recall: 0.08163265306122448
Precision: 0.8888888888888888
F1 Score: 0.14953271028037382
              precision    recall  f1-score   support

       False       0.86      1.00      0.92       536
        True       0.89      0.08      0.15        98

    accuracy                           0.86       634
   macro avg       0.87      0.54      0.54       634
weighted avg       0.86      0.86      0.80       634
```

- There is no improvement after tuning on accuracy, recall, precision and f1 score.

SVM - Support Vector Machine

```
# Initialize and train te SVC model
svm_model = SVC()
svm_model.fit(X_train, y_train)

#Make predictions on the test set
y_pred = svm_model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")

#Classification report
print(classification_report(y_test, y_pred))
```

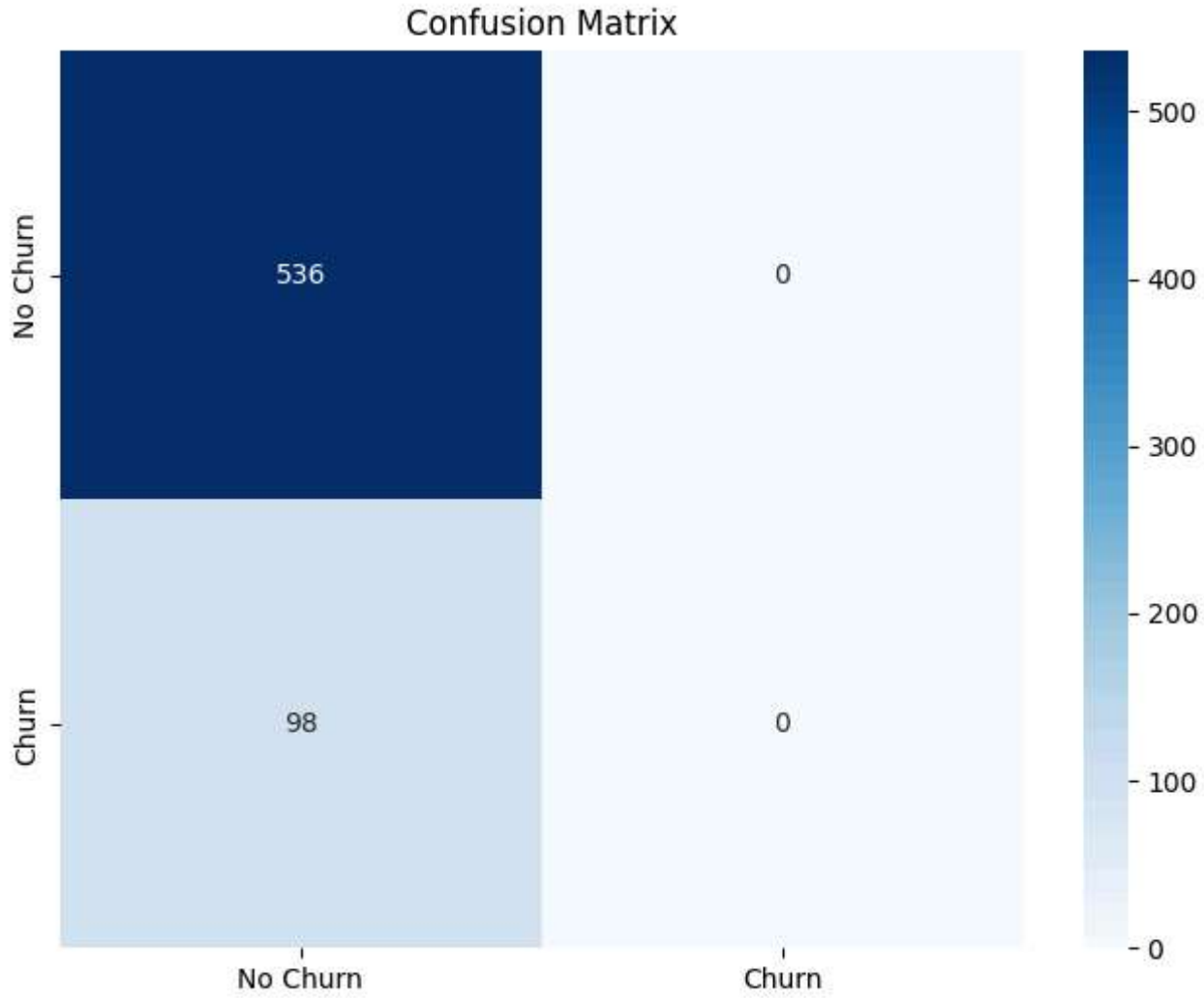
```
Accuracy: 0.8454258675078864
Recall: 0.0
Precision: 0.0
F1 Score: 0.0
              precision    recall  f1-score   support

       False       0.85      1.00      0.92       536
        True       0.00      0.00      0.00        98

    accuracy                           0.85       634
```

```
           macro avg       0.42      0.50      0.46       634
        weighted avg       0.71      0.85      0.77       634
```

- SVM has an accuracy of 84.5% which is a good percentage but not the best compared to random forest.
- With 0 recall, 0 precision and f1 score of 0 makes it not the option model to be used.

```
#Plot the confusion matrix

cm= confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Churn', 'Churn'], ytick
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```



Hyperparameter Tuning

```
#So far the best model is the Random Forest
```