BUSINESS UNDERSTANDING

In the telecommunications industry,customer churn is a major challenge. Retaining existing customers is more cost effective than acquiring new ones, making it essential to understand why customers leave. By analyzing customer behavior, usage patterns and interactions with service plans,we can identify the key factors driving churn. These insights will help the company develop proactive strategies to improve retention thus enhance customer satisfaction and reduce revenue loss.

PROBLEM STATEMENT

This project aims to build a predictive model to classify customers as either likely to churn or remain. Specifically, we will:

1.Churn Prediction Task: Develop a machine learning model to classify whether a customer will churn based on their usage and service history

2.Key Drivers Analysis: Identify the most influential factors contributing to churn, such as high call charges, frequent customer service interactions, or lack of service plans.

OBJECTIVES

1.To develop an accurate and reliable customer churn prediction model with an accuracy of 85%

2.To identify key factors contributing to customer churn

3.To formulate targeted customer retention strategies

DATA UNDERSTANDING

```
#importing the neccesary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import math
%matplotlib inline
warnings.filterwarnings('ignore')

# Libraries for building models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Libraries for Preprocessing
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Libraries for Model Evaluation
from sklearn.model_selection import cross_val_score
from sklearn.metrics import recall_score, accuracy_score, precision_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
```

```
#loading thee data set
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df
```

	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 calls	total eve charge	total night minutes	total night calls	total night charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.0
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.3
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.8
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83	12.5
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123	8.6
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91	8.6
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137	6.2
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77	10.8

3333 rows × 21 columns

```
#checking the nature of the 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
```

The data information shows that there are no missing values.

```
df.describe()
```

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000

```
#checking unique values
df.nunique()
```

```
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
```

```
#checking the unique values for the target column 'churn'
df['churn'].unique()
```

```
array([False,  True])
```

```
#counting the number of occurrences of each unique value in the 'churn' column
df['churn'].value_counts()
```

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

```
#checking the data types for all columns
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
```

```
#Checking the unique values in categorical columns
categorical_columns = ['state','international plan','voice mail plan']
df[categorical_columns].apply(pd.unique)
```

```
state                  [KS, OH, NJ, OK, AL, MA, MO, LA, WV, IN, RI, I...
international plan                                            [no, yes]
voice mail plan                                               [yes, no]
dtype: object
```

The 'international plan' and 'voice mail plan' columns have unique values of 'yes' and 'no'.This indicates the presence or absence of the respective plans.

```
# Displaying all unique values in the 'state'column and the value count
print("Unique values in 'state':")
print(df['state'].unique())
print(f"Total unique values: {df['state'].nunique()}")
```

```
Unique values in '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']
Total unique values: 51
```

The 'state' column has unique values representing 51 states.

DATA CLEANING

```
# Checking for duplicates
df.duplicated().sum()
```

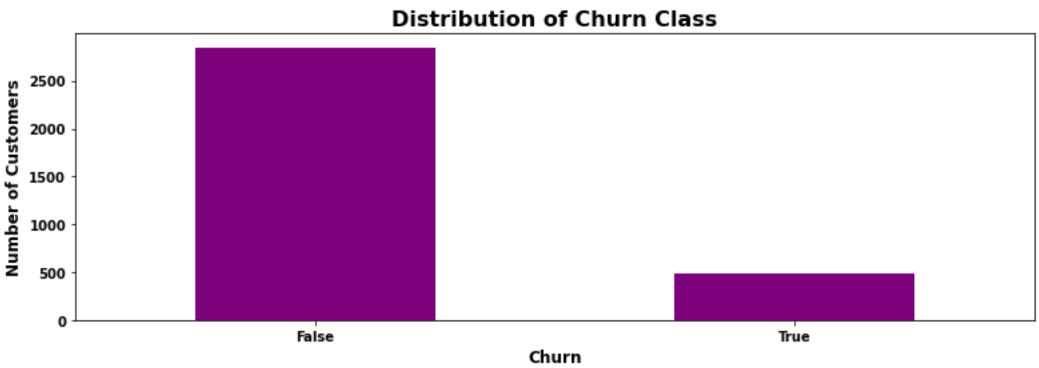
```
0
```

There are no duplicated entries

```
# Dropping irrelevant columns for our modelling and analysis.
df.drop(columns=['phone number'], inplace=True)
```

. EXPLORATORY DATA ANALYSIS

```
#Visualizing the class distribution in the 'churn' column
plt.figure(figsize=(11, 4))
df['churn'].value_counts().plot(kind='bar',color='purple')
plt.title("Distribution of Churn Class", fontsize=16, fontweight='bold')
plt.xlabel("Churn", fontsize=12, fontweight='bold')
plt.ylabel("Number of Customers", fontsize=12, fontweight='bold')
plt.xticks(rotation=0, fontsize=10, fontweight='bold')
plt.yticks(fontsize=10, fontweight='bold')
plt.tight_layout()
plt.show()
```



The majority of customers didnt churn.

```
df.dtypes
```

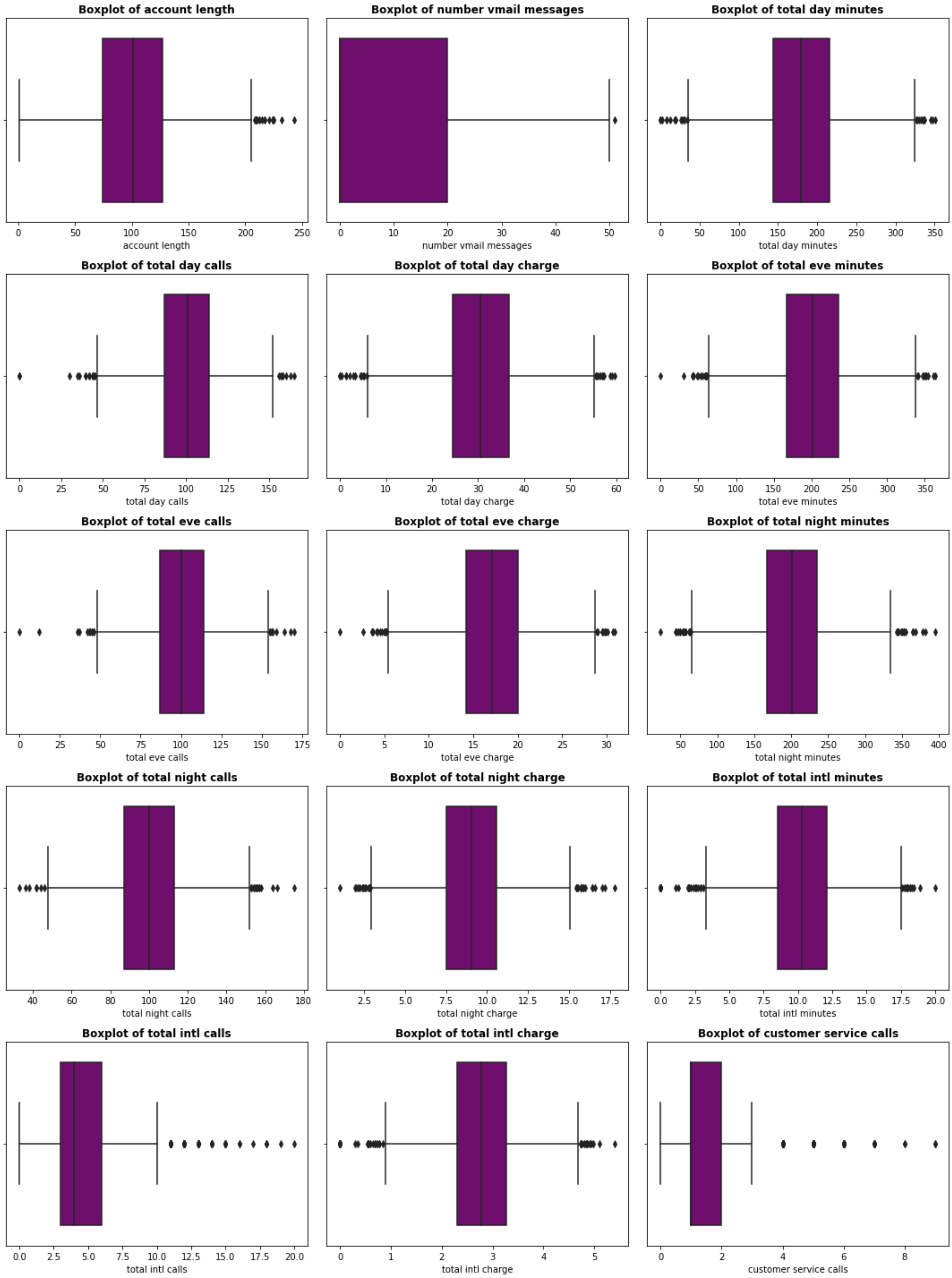
```
state                  object
account length          int64
area code               int64
international plan     object
```

```
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.head()
```

	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	to i minu
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	1
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	1
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	1
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	1

```
# identifying  the relationships between variables, patterns and outliers.
# Creating a df with only the numeric columns
numeric_columns = ['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']
numeric_df = df[numeric_columns]


# Defining the number of rows and columns for the grid layout
num_cols = len(numeric_columns)
num_rows = math.ceil(num_cols / 3)

# Seting the figure size
fig, axes = plt.subplots(num_rows, 3, figsize=(15, num_rows * 4))
axes = axes.flatten()
# Plotting each numerical column in a separate subplot
for i, col in enumerate(numeric_columns):
    sns.boxplot(x=df[col], ax=axes[i], color='purple')
    axes[i].set_title(f"Boxplot of {col}", fontsize=12, fontweight='bold')
plt.tight_layout()
plt.show()
```

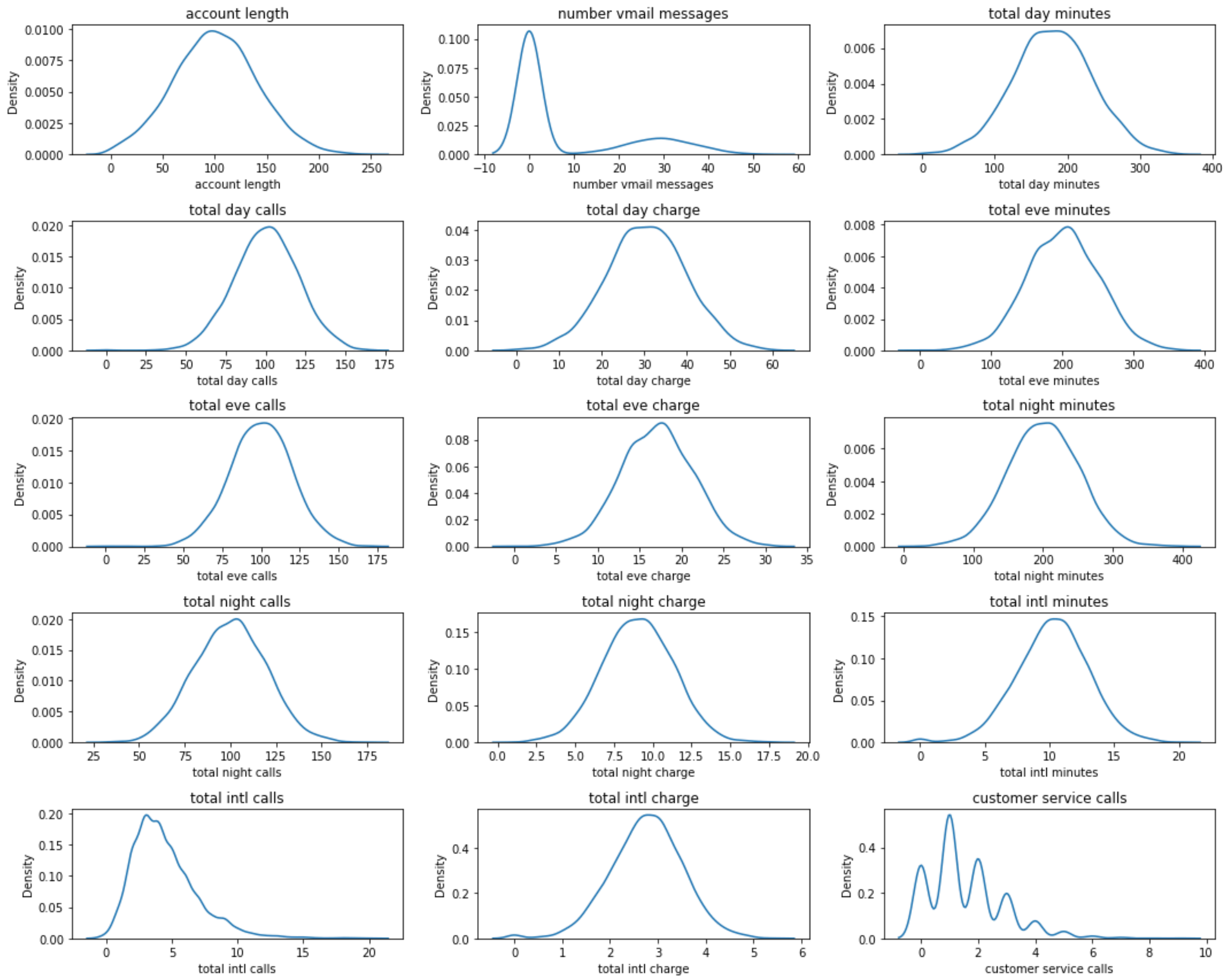


all numerical features have outliers.Some very significant like in the 'total_int_calls' column and some not very significant like 'total_night' column.

Presence of outliers in the dataset may have been attributed to extreme customer behavior.

```
EXPLORING THE DISTRIBUTION OF THE NUMERIC FEATURES
```

```
# Setting the figure size
plt.figure(figsize=(15, 12))

# Looping through numerical columns and create KDE plots
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(math.ceil(len(numeric_columns) / 3), 3, i)
    sns.kdeplot(df[col])
    plt.title(col)

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



-Voicemail Messages and Customer Service Calls are right-skewed, with many customers having zero messages or service calls.

-International Minutes and Charges are right-skewed, meaning most customers make fewer international calls.

-Call counts :Total Day, Evening, and Night Calls, Total Intl Calls) are roughly symmetric, peaking around 100 calls (except international calls, which peak around 3-4).

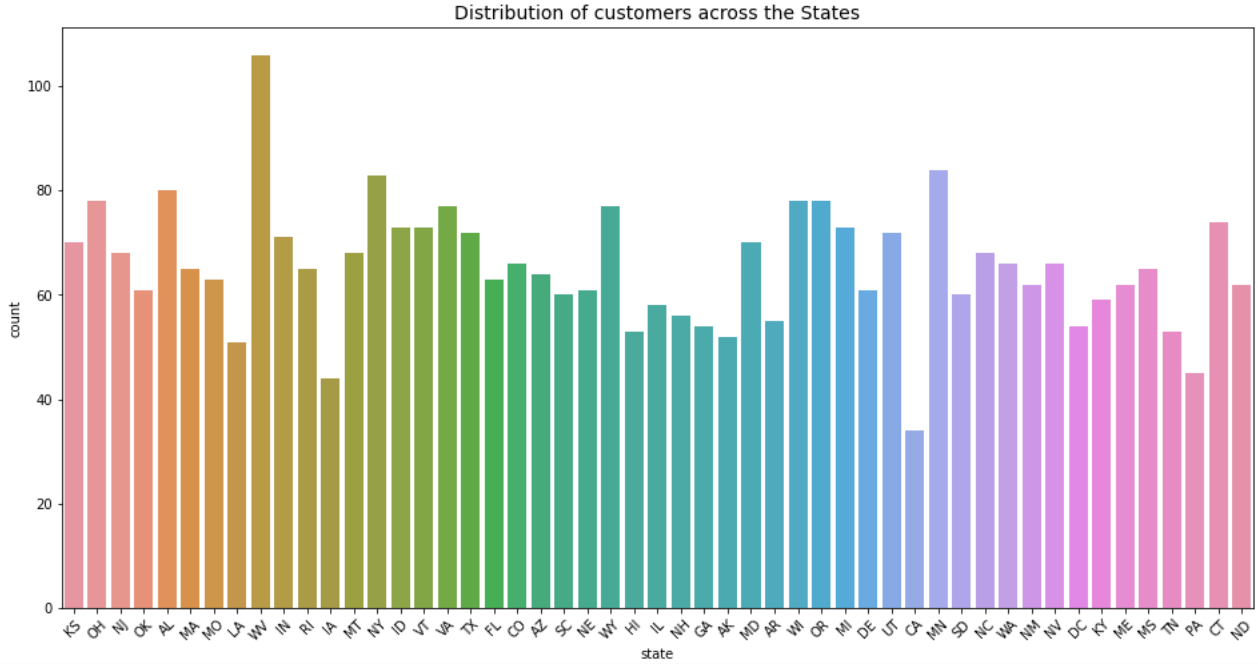
-Most call-related metrics (Total Day, Eve, and Night Minutes & Charges) follow an approximately normal distribution with a slight right skew.

-This indicates that while general call usage is normally distributed, customer service interactions and international usage are more varied, with some outliers

```
# customer distribution across the 51 states
plt.figure(figsize=(16,8))
plt.xticks(rotation=45)
plt.title('Distribution of customers across the States', fontsize=14)
xlabel='States'
```

```
sns.countplot(x='state', data=df)
```

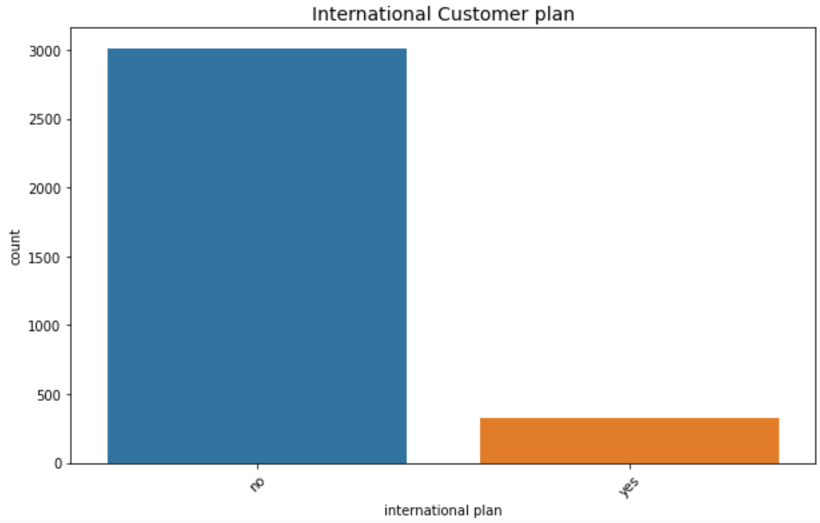
```
<Axes: title={'center': 'Distribution of customers across the States'}, xlabel='state', ylabel='count'>
```



the top 5 states with high customer count are: West Virginia (WV), Minnesota(MN), New York (NY), Alabama, Customer (AL), Ohio (OH).

```
#Plot showing international customers plan.
plt.figure(figsize=(10,6))
plt.xticks(rotation=45)
plt.title('International Customer plan', fontsize=14)
xlabel='International plan'
sns.countplot(x='international plan', data=df)
```

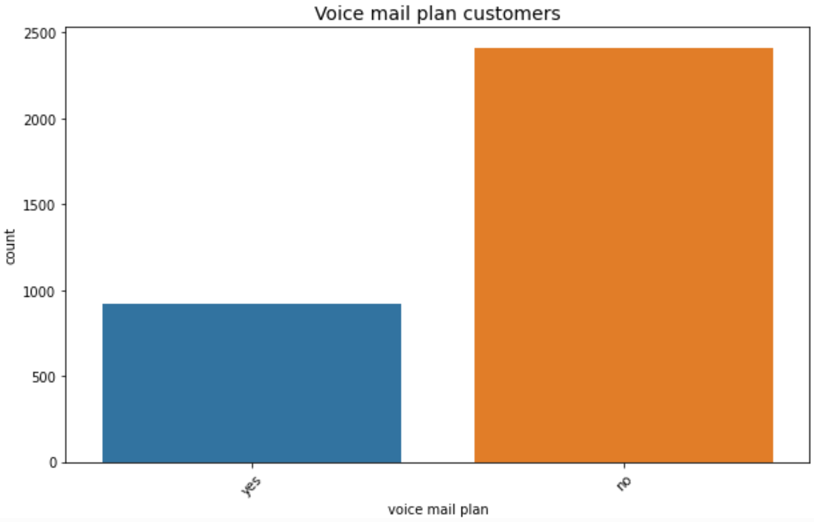
```
<Axes: title={'center': 'International Customer plan'}, xlabel='international plan', ylabel='count'>
```



In the above plot most customers have international plan.less than 500 dont have the international plan.

```
# plot showing voice mail plans for customers
plt.figure(figsize=(10,6))
plt.xticks(rotation=45)
plt.title('Voice mail plan customers', fontsize=14)
xlabel='voice mail plan'
sns.countplot(x='voice mail plan', data=df)
```

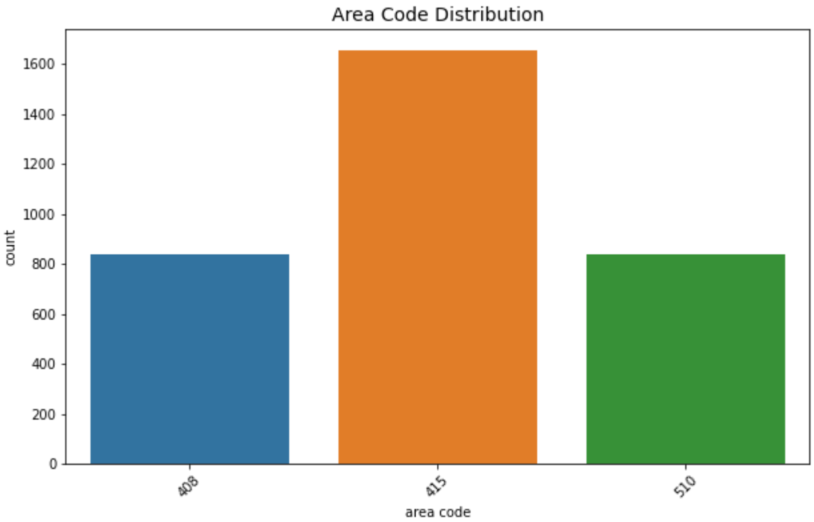
```
<Axes: title={'center': 'Voice mail plan customers'}, xlabel='voice mail plan', ylabel='count'>
```



most customers had voice mail plans

```
#Plot showing  area code distribution
plt.figure(figsize=(10,6))
plt.xticks(rotation=45)
plt.title('Area Code Distribution', fontsize=14)
xlabel='area code'
sns.countplot(x='area code', data=df)
```

```
<Axes: title={'center': 'Area Code Distribution'}, xlabel='area code', ylabel='count'>
```

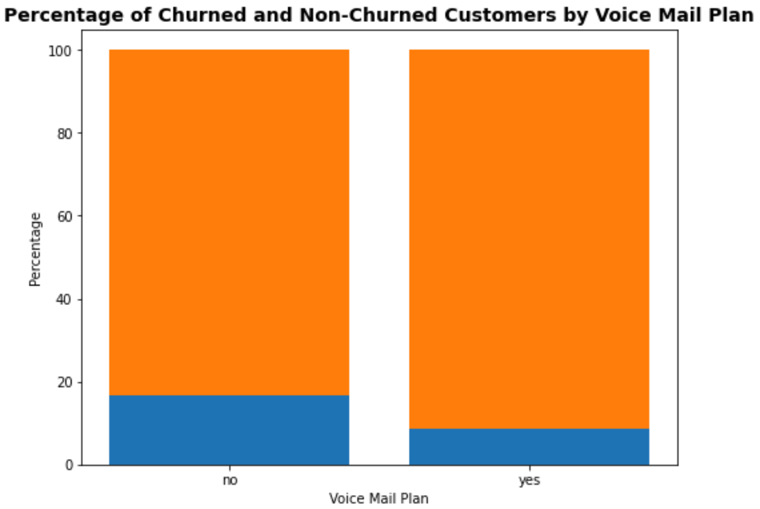


most customers resided in area code 415

Bivariate Analysis

Voice mail plan

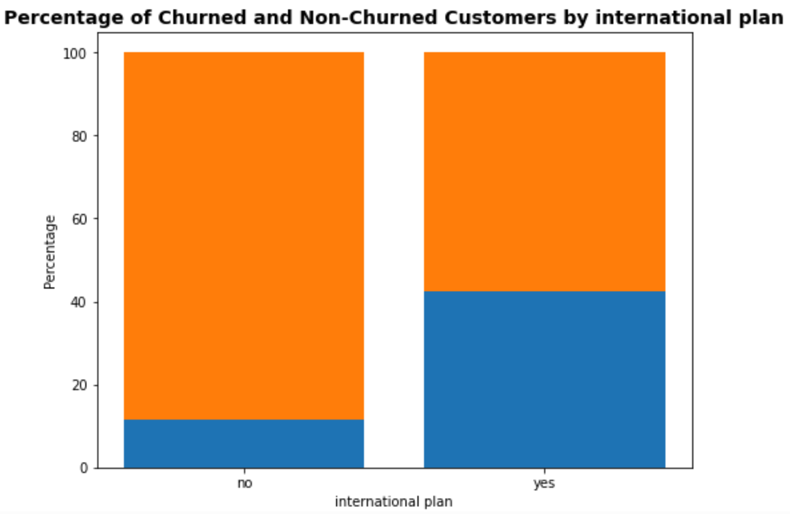
```
# Calculating the percentage of churned and non-churned customers by voice mail plan
churn_data = df.groupby(['voice mail plan', 'churn']).size().unstack(fill_value=0)
churn_percent = churn_data.div(churn_data.sum(axis=1), axis=0) * 100

# Plotting the stacked bar chart
plt.figure(figsize=(8, 6))
plt.bar(churn_percent.index, churn_percent[True], label='Churned')
plt.bar(churn_percent.index, churn_percent[False], bottom=churn_percent[True], label='Non-Churned')
# Adding labels and title
plt.xlabel('Voice Mail Plan')
plt.ylabel('Percentage')
plt.title('Percentage of Churned and Non-Churned Customers by Voice Mail Plan', fontsize=14, fontweight='bold')
plt.show()
```



There's a slight percentage of customers who have churned without a voice mail plan as opposed to those who do.

International plan

```
churn_data = df.groupby(['international plan', 'churn']).size().unstack(fill_value=0)
churn_percent = churn_data.div(churn_data.sum(axis=1), axis=0) * 100

# Plotting the stacked bar chart
plt.figure(figsize=(8, 6))
plt.bar(churn_percent.index, churn_percent[True], label='Churned')
plt.bar(churn_percent.index, churn_percent[False], bottom=churn_percent[True], label='Non-Churned')

# Adding labels and title
plt.xlabel('international plan')
plt.ylabel('Percentage')
plt.title('Percentage of Churned and Non-Churned Customers by international plan', fontsize=14, fontweight='bold')
plt.show()
```



A high percentage of customers with an international plan are more likely to churn.

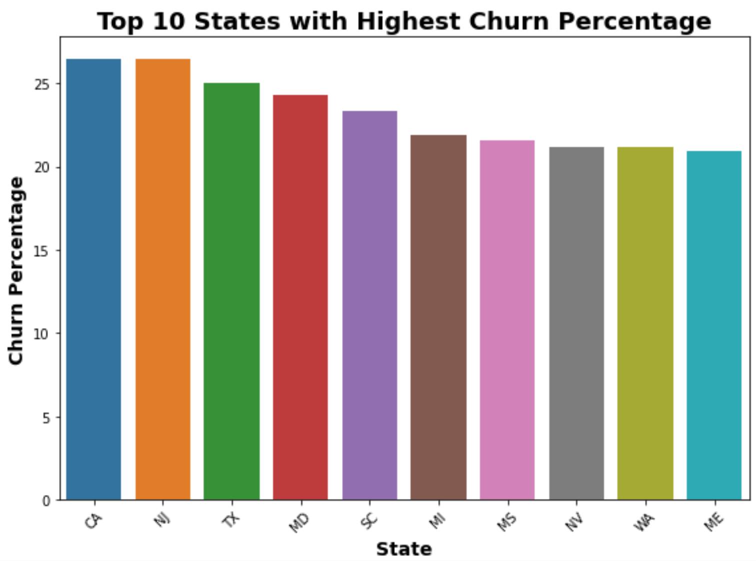
State

```
# Calculating churn percentage by state
state_churn_percentage = df.groupby('state')['churn'].mean() * 100

# Sorting the states by churn percentage in descending order and select the top 10 states
top_states = state_churn_percentage.sort_values(ascending=False).head(10)
top_states
```

```
state
CA    26.470588
NJ    26.470588
TX    25.000000
```

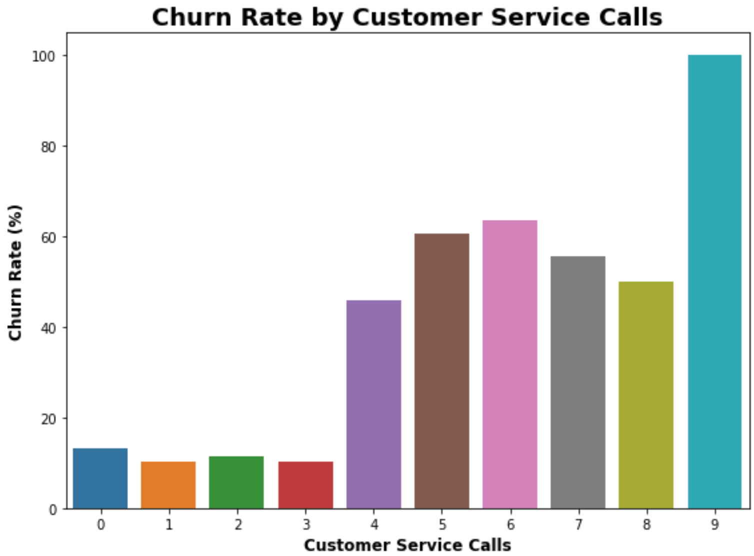
```
MD    24.285714
SC    23.333333
MI    21.917808
MS    21.538462
NV    21.212121
WA    21.212121
ME    20.967742
Name: churn, dtype: float64
```

```
# Calculating churn percentage by state
state_churn_percentage = df.groupby('state')['churn'].mean() * 100

# Sorting the states by churn percentage in descending order and select the top 10 states
top_states = state_churn_percentage.sort_values(ascending=False).head(10)

# Plotting the top 10 states
plt.figure(figsize=(8, 6))
sns.barplot(x=top_states.index, y=top_states.values)
plt.xlabel('State', fontsize=14, fontweight='bold')
plt.ylabel('Churn Percentage', fontsize=14, fontweight='bold')
plt.title('Top 10 States with Highest Churn Percentage', fontsize=18, fontweight='bold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Number of customer service calls

```
# Calculating the churn rate for each customer service call category
churn_rate_by_calls = df.groupby('customer service calls')['churn'].mean() * 100
plt.figure(figsize=(8, 6))

# Adding labels and title
sns.barplot(x=churn_rate_by_calls.index, y=churn_rate_by_calls.values)
plt.xlabel('Customer Service Calls', fontsize=12, fontweight='bold')
plt.ylabel('Churn Rate (%)', fontsize=12, fontweight='bold')
plt.title('Churn Rate by Customer Service Calls', fontsize=18, fontweight='bold')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



Customers who have a higher number of service calls, particularly 4 or more, are more likely to churn compared to those with fewer service calls.

This insight shows how important it is to deal with customer problems and concerns.this will reduce churn and improve customer satisfaction.

Feature Engineering

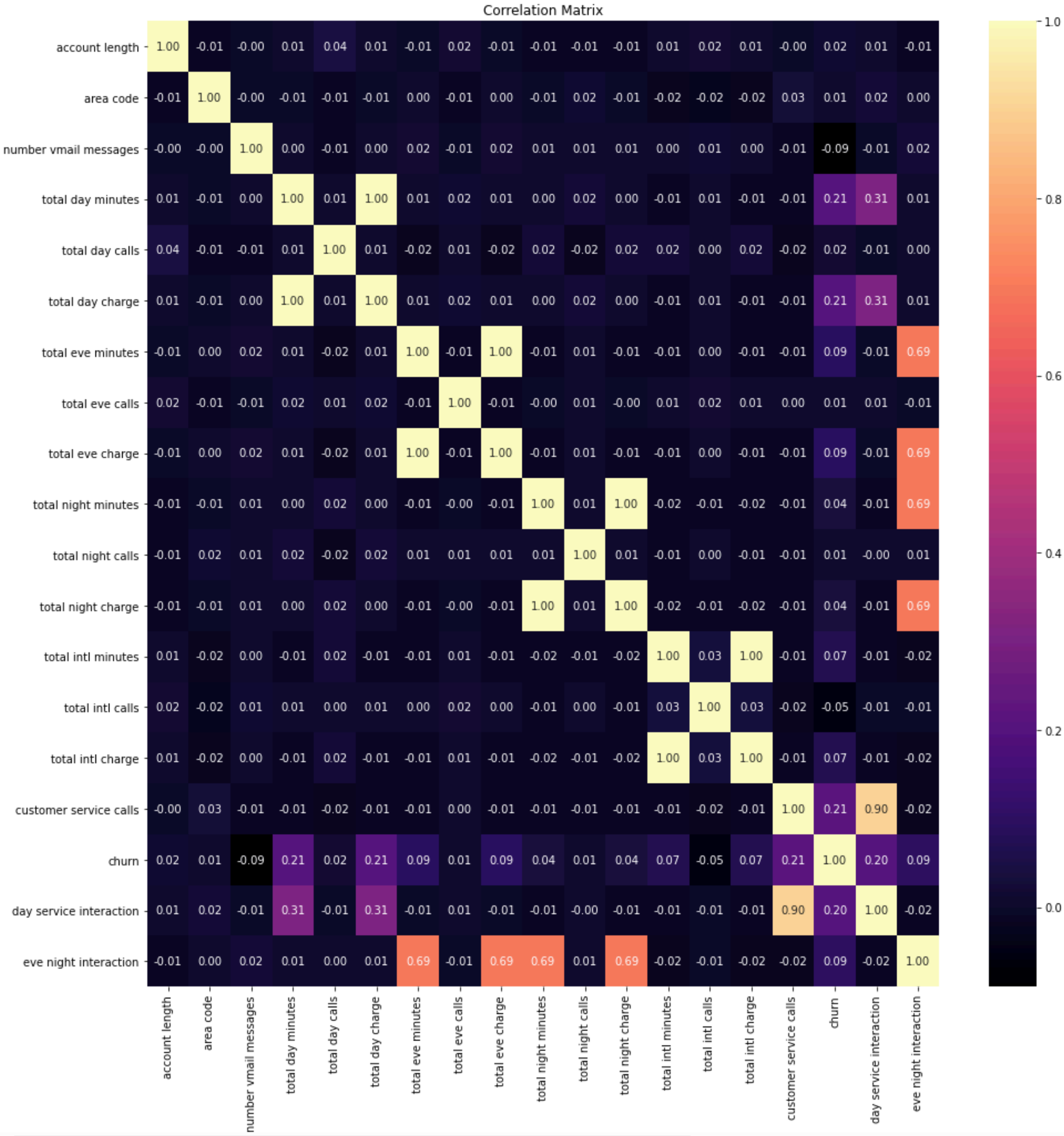
```
df['day service interaction'] = df['total day minutes'] * df['customer service calls']
df['eve night interaction'] = df['total eve minutes'] * df['total night minutes']

df.head()
```

	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 night minutes	total night calls	total night charge	total intl minutes	total intl calls
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	...	244.7	91	11.01	10.0	3
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	...	254.4	103	11.45	13.7	3
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	...	162.6	104	7.32	12.2	5
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	...	196.9	89	8.86	6.6	7
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	...	186.9	121	8.41	10.1	3

5 rows × 22 columns

Multivariate analysis

```
#plotting the correlation matrix
correlation_matrix = df.corr()
# Visualizing the correlation using heatmap
plt.figure(figsize=(16,16))
sns.heatmap(correlation_matrix, annot=True, cmap='magma', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



Churn has a positive correlation with total_day_minutes, total_day_charge, total_eve_charge, total_night_minutes, and customer_service_calls.

Higher values of day_service_interaction and eve_night_interaction indicate a higher likelihood of churn.

More customer_service_calls are associated with a higher likelihood of churn. There is a weak positive correlation between international calls/charges and churn. The number of voicemail messages has a weak negative correlation with churn.

Preprocessing data for modelling

```
# Encoding binary 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})

# Creating an instance of LabelEncoder
label_encoder = LabelEncoder()

# Encoding the "churn" column
df['churn'] = label_encoder.fit_transform(df['churn'])


# Splitting the dataset into features (X) and target variable (y)
X = df.drop(columns=['churn', 'state'], axis=1)
y = df['churn']
```

```
# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Applying SMOTE to handle class imbalance on the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Applying StandardScaler for feature scaling on the training set
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)

# Applying feature scaling and constant term to the test set
X_test_scaled = scaler.transform(X_test)
```

MODELING

The following steps were involved:

1 Selecting Modeling Techniques:

2 Generate Test Design

3 Build Model

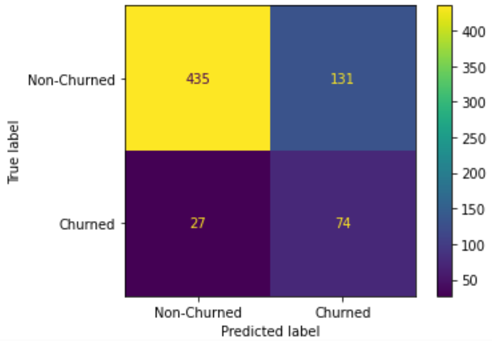
4 Feature Selection

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression

def perform_cross_validation(X, y, model_type='decision_tree', n_estimators=100, max_depth=None, cv=5):

    if model_type == 'decision_tree':
        clf = DecisionTreeClassifier(max_depth=max_depth, random_state=42)
    elif model_type == 'random_forest':
        clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=42)
    elif model_type == 'logistic_regression':
        clf = LogisticRegression(solver='liblinear', random_state=42)
    elif model_type == 'gradient_boosting':
        clf = GradientBoostingClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=42)
    else:
        raise ValueError("Invalid model_type. Supported types: 'decision_tree', 'random_forest', 'logistic_regression', 'gradient_boost:

    # Performing cross-validation and calculate mean accuracy
    scores = cross_val_score(clf, X, y, cv=cv, scoring='accuracy')
    return scores.mean()



def evaluate_model(y_true, y_pred):
    # Calculating accuracy
    accuracy = accuracy_score(y_true, y_pred)
    print("Accuracy:", accuracy)

    # Calculating precision
    precision = precision_score(y_true, y_pred)
    print("Precision:", precision)

    # Calculating recall
    recall = recall_score(y_true, y_pred)
    print("Recall:", recall)

    # Calculating F1-score
    f1 = f1_score(y_true, y_pred)
    print("F1-score:", f1)

    # Creating a confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    class_names = ['Non-Churned', 'Churned']
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
    disp.plot()
```

Baseline model

```
from imblearn.pipeline import Pipeline

# Defining the pipeline steps
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('scaler', StandardScaler()),
    ('model', LogisticRegression(solver='liblinear', random_state=42))
])

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

# Making predictions on the test data
y_pred_1 = pipeline.predict(X_test)

# Evaluating the model
accuracy = pipeline.score(X_test, y_test)


baseline_cv = perform_cross_validation(X_train_scaled, y_train_resampled, model_type='logistic_regression')
print("Baseline Cross Validation Score", baseline_cv)
evaluate_model(y_test, y_pred_1)
```

```
Baseline Cross Validation Score 0.7583228877315508
Accuracy: 0.7631184407796102
Precision: 0.36097560975609755
Recall: 0.7326732673267327
F1-score: 0.48366013071895425
```



Feature Selection

```
from sklearn.feature_selection import RFE


# Creating an instance of the logistic regression model
logreg = LogisticRegression(max_iter=1000)

# Creating an instance of the RFE selector
rfe = RFE(estimator=logreg, n_features_to_select=10)

# Fitting the RFE selector on the training data
rfe.fit(X_train_scaled, y_train_resampled)

# Getting the selected feature indices
selected_indices = rfe.get_support(indices=True)

# Subseting the training and testing the data based on the selected features
X_train_selected = X_train.iloc[:, selected_indices]
X_test_selected = X_test.iloc[:, selected_indices]

# Training the model
logreg.fit(X_train_selected, y_train)

# Making predictions on the test data
y_pred = logreg.predict(X_test_selected)

# Evaluating the performance of your model
accuracy = accuracy_score(y_test, y_pred)


# Printing the results
print(f"Selected Features: {X_train_selected.columns.tolist()}")
print(f"Accuracy: {accuracy}")
```

```
Selected Features: ['international plan', 'voice mail plan', 'number vmail messages', 'total day minutes', 'total day charge', 'tota
Accuracy: 0.848575712143928
```

from the above report the Recursive Feature Elimination (RFE) selected the above features.The logistic regression model trained on these selected features achieved an accuracy of approximately 84.86% on the test data.

Decision Tree

```
# Instantiate and fit a DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train_selected, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
# Creating a Random Forest classifier
rf_model = RandomForestClassifier(random_state=42)

# Fittong the model on the training data
rf_model.fit(X_train_selected, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Gradient Boosting Classifier

```
# Creating a Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(random_state=42)

# Fitting the model on the training data
gb_model.fit(X_train_selected, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

Predictions

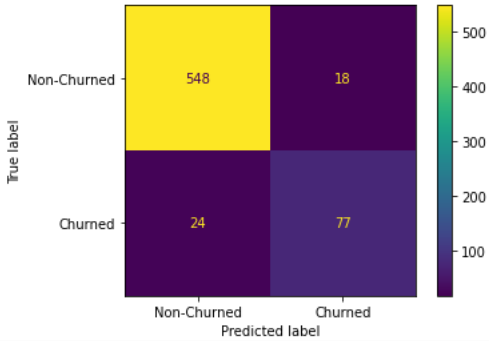
```
# Making predictions on the test data for the Decision Tree model
y_pred_2 = tree_clf.predict(X_test_selected)

# Making predictions on the test data for the Random Forest model
y_pred_3 = rf_model.predict(X_test_selected)

# Making predictions on the test data for the Gradient Boosting Classifier
y_pred_4 = gb_model.predict(X_test_selected)
```

Evaluation Metrics

The evaluation metrics used to assess the models include:Accuracy,Precision,Recall,F1-score and Cross Validation Score

```
cv_dt = perform_cross_validation(X_train_selected, y_train, model_type='decision_tree')
print("Decision Tree Model Cross Validation Score:", cv_dt)

evaluate_model(y_test, y_pred_2)
```

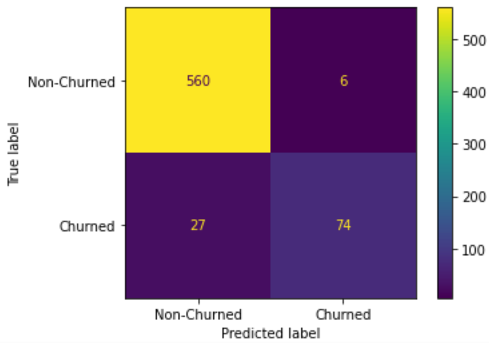
```
Decision Tree Model Cross Validation Score: 0.9163529172024651
Accuracy: 0.9370314842578711
Precision: 0.8105263157894737
Recall: 0.7623762376237624
F1-score: 0.7857142857142857
```



Random forest

```
cv_rf = perform_cross_validation(X_train_selected, y_train, model_type='random_forest')
print("Random Forest Cross Validation Score:", cv_rf)

evaluate_model(y_test, y_pred_3)
```

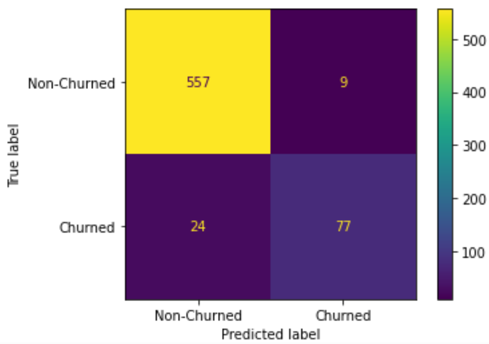
```
Random Forest Cross Validation Score: 0.9561123173893795
Accuracy: 0.9505247376311844
Precision: 0.925
Recall: 0.7326732673267327
F1-score: 0.8176795580110497
```



Gradient Boosting Classifier

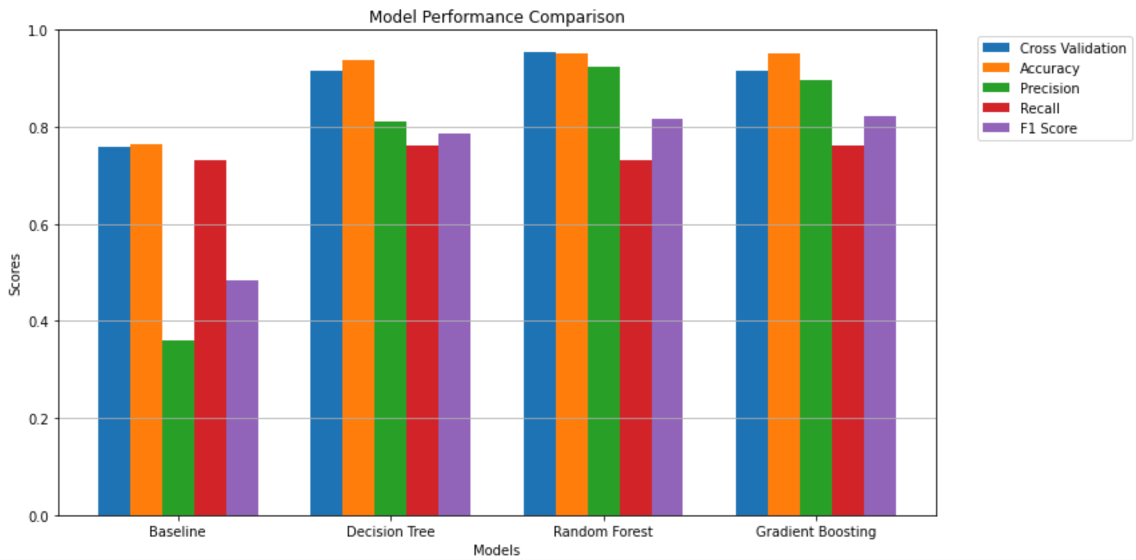
```
cv_gc = perform_cross_validation(X_train_selected, y_train, model_type='gradient_boosting')
print("Gradient Boosting Classifier Cross Validation Score", cv_gc)

evaluate_model(y_test, y_pred_4)
```

```
Gradient Boosting Classifier Cross Validation Score 0.9182283871239749
Accuracy: 0.9505247376311844
Precision: 0.8953488372093024
Recall: 0.7623762376237624
F1-score: 0.823529411764706
```



Based on the above insights, the Gradient Boosting Classifier outperforms the other models, showing the highest scores for cross-validation, accuracy, precision, recall, and F1-score.

```
models = ['Baseline', 'Decision Tree', 'Random Forest', 'Gradient Boosting']
metrics = ['Cross Validation', 'Accuracy', 'Precision', 'Recall', 'F1 Score']
cross_validation_scores = [0.7583, 0.9171, 0.9550, 0.9171]
accuracy_scores = [0.7631, 0.9370, 0.9505, 0.9505]
precision_scores = [0.3610, 0.8105, 0.9250, 0.8953]
recall_scores = [0.7327, 0.7624, 0.7327, 0.7624]
f1_scores = [0.4837, 0.7857, 0.8177, 0.8235]
values = np.array([cross_validation_scores,accuracy_scores,precision_scores,recall_scores,f1_scores])

# Plotting
plt.figure(figsize=(12, 6))
x = np.arange(len(models))
width = 0.15
for i in range(len(metrics)):
    plt.bar(x + i * width, values[i], width, label=metrics[i])
    plt.title('Model Performance Comparison')
plt.xlabel('Models')
plt.ylabel('Scores')
plt.xticks(x + width * 2, models)
plt.ylim(0, 1)
plt.legend(bbox_to_anchor=(1.04, 1), loc="upper left")
plt.tight_layout()
plt.grid(axis='y')

plt.show()
```



Hyperparameter tuning

I loaded the required packages, specified hyperparameters to test, created a GradientBoostingClassifier instance, used GridSearchCV with cross-validation and scoring, ran the grid search to find the best parameters, extracted the optimal hyperparameters and score, fitted the tuned model on the full training data, predicted on the test data, validated the model with cross-validation, and assessed performance using metrics.

```
from sklearn.model_selection import GridSearchCV

# Defining the parameter grid
param_grid = {
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7]
}

# Creating an instance of the Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier()

# Creating a GridSearchCV object
grid_search = GridSearchCV(gb_classifier, param_grid, cv=5, scoring='accuracy')

# Fitting the grid search to the training data
grid_search.fit(X_train_selected, y_train)

# Getting the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

```
# Printing the best parameters and best score
print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

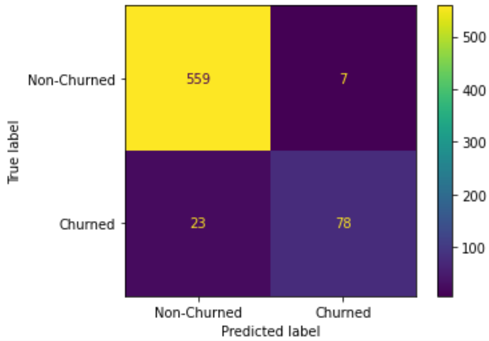
```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300}
Best Score: 0.9549859111382817
```

```
# Creating an instance of the Gradient Boosting Classifier with the best parameters
gb_classifier_1 = GradientBoostingClassifier(max_depth=5, n_estimators=300)

# Training the classifier on the entire training dataset
gb_classifier_1.fit(X_train_selected, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(max_depth=5, n_estimators=300)
```

```
y_pred_5 = gb_classifier_1.predict(X_test_selected)

cv_gc_1 = perform_cross_validation(X_train_selected, y_train, model_type='gradient_boosting', max_depth=5, n_estimators=300)
print("Gradient Boosting Classifier Tuned model Cross Validation Score", cv_gc_1)

# Evaluating tuned model
evaluate_model(y_test, y_pred_5)
```

```
Gradient Boosting Classifier Tuned model Cross Validation Score 0.9549873165110215
Accuracy: 0.9550224887556222
Precision: 0.9176470588235294
Recall: 0.7722772277227723
F1-score: 0.8387096774193548
```



The target accuracy score of 80% has been achieved, indicating that the churn prediction model successfully meets the objective for accuracy. This ensures accurate identification of customers at risk of churn, enabling effective implementation of targeted retention strategies. The model's performance demonstrates its potential to reduce customer churn and improve retention rates, positively impacting the company.

```
def plot_feature_importances(model):

    n_features = X_train_selected.shape[1]
    feature_importances = model.feature_importances_
    sorted_indices = np.argsort(feature_importances)

    plt.figure(figsize=(8, 8))
    plt.barh(range(n_features), feature_importances[sorted_indices], align='center')
    plt.yticks(range(n_features), X_train_selected.columns[sorted_indices])
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importances')
    plt.show()
```

```
plot_feature_importances(gb_classifier_1)
```