predict the whether a user will close their account given usage history and demographic features

Target variable is Acct Closed

In [ ]:
```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,precision_score,recall_score
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
df = pd.read_csv("/content/sample_data/UserRetentionData.csv")
```

Data Understanding

In [ ]:
```
df.head()
```

Out[ ]:

	Region	Tenure	Neighborhood	Trunk Calling Facility	Voice Messaging	Number voice messages	Minutes Peak Hrs	Calls Peak Hrs	Bill Peak Hrs	Mi
0	KS	128	415	No	Yes	25	265.1	110	45.07	
1	OH	107	415	No	Yes	26	161.6	123	27.47	
2	NJ	137	415	No	No	0	243.4	114	41.38	
3	OH	84	408	Yes	No	0	299.4	71	50.90	
4	OK	75	415	Yes	No	0	166.7	113	28.34	

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2666 entries, 0 to 2665
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Region                          2666 non-null   object 
 1   Tenure                          2666 non-null   int64  
 2   Neighborhood                    2666 non-null   int64  
 3   Trunk Calling Facility          2666 non-null   object 
 4   Voice Messaging                 2666 non-null   object 
 5   Number voice messages           2666 non-null   int64  
 6   Minutes Peak Hrs                2666 non-null   float64
 7   Calls Peak Hrs                  2666 non-null   int64  
 8   Bill Peak Hrs                   2666 non-null   float64
 9   Minutes Off Peak                2666 non-null   float64
 10  Calls Off Peak                  2666 non-null   int64  
 11  Bill Off Peak                   2666 non-null   float64
 12  Minutes Night                   2666 non-null   float64
 13  Calls Night                     2666 non-null   int64  
 14  Bill Night                      2666 non-null   float64
 15  Trunk Call Minutes              2666 non-null   float64
 16  Trunk Calls                     2666 non-null   int64  
 17  Trunk Call Bill                 2666 non-null   float64
 18  Contact for Grievances/Changes  2666 non-null   int64  
 19  Acct Closed?                    2666 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 398.5+ KB
```

In [ ]:
```
df.describe()
```

Out[ ]:

	Tenure	Neighborhood	Number voice messages	Minutes Peak Hrs	Calls Peak Hrs	Bill Peak Hrs	2
count	2666.000000	2666.000000	2666.000000	2666.00000	2666.000000	2666.000000	
mean	100.620405	437.438860	8.021755	179.48162	100.310203	30.512404	
std	39.563974	42.521018	13.612277	54.21035	19.988162	9.215733	
min	1.000000	408.000000	0.000000	0.00000	0.000000	0.000000	
25%	73.000000	408.000000	0.000000	143.40000	87.000000	24.380000	
50%	100.000000	415.000000	0.000000	179.95000	101.000000	30.590000	
75%	127.000000	510.000000	19.000000	215.90000	114.000000	36.700000	
max	243.000000	510.000000	50.000000	350.80000	160.000000	59.640000	

Data Cleaning

In [ ]:
```
# Checking if there are any missing values
df.isnull().sum()
```

Out[ ]:
```
Region                            0
Tenure                            0
Neighborhood                      0
Trunk Calling Facility            0
Voice Messaging                   0
Number voice messages             0
Minutes Peak Hrs                  0
Calls Peak Hrs                    0
Bill Peak Hrs                     0
Minutes Off Peak                  0
Calls Off Peak                    0
Bill Off Peak                     0
Minutes Night                     0
Calls Night                       0
Bill Night                        0
Trunk Call Minutes                0
Trunk Calls                       0
Trunk Call Bill                   0
Contact for Grievances/Changes    0
Acct Closed?                      0
dtype: int64
```

In [ ]:
```
# Checking if there are any duplicate values
df.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
df['Acct Closed?'].value_counts()
```

Out[ ]:
```
False    2278
True      388
Name: Acct Closed?, dtype: int64
```

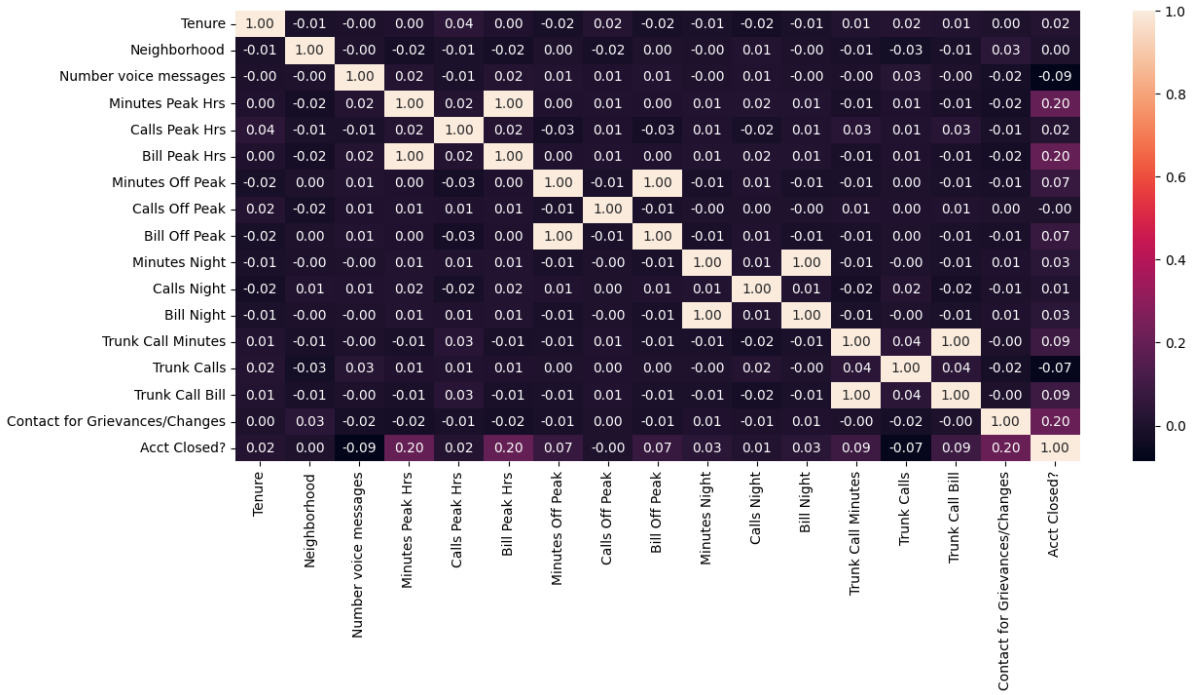
In [ ]:
```
# Data Imbalance
(df['Acct Closed?'].value_counts()/df['Acct Closed?'].count())
```

Out[ ]:
```
False    0.854464
True     0.145536
Name: Acct Closed?, dtype: float64
```

In [ ]:
```
# sns.pairplot(df)
```

In [ ]:
```
plt.figure(figsize=(14,6))
sns.heatmap(df.corr(), annot=True, fmt='0.2f')
```

Out[ ]:
```
<Axes: >
```



Columns with high correlation with each other

Bill Peak Hrs-Minutes Peak Hrs, Bill Off Peak- Minutes Off Peak, Bill Night - Minutes Night, Trunk Call Bill-Trunk Call Minutes

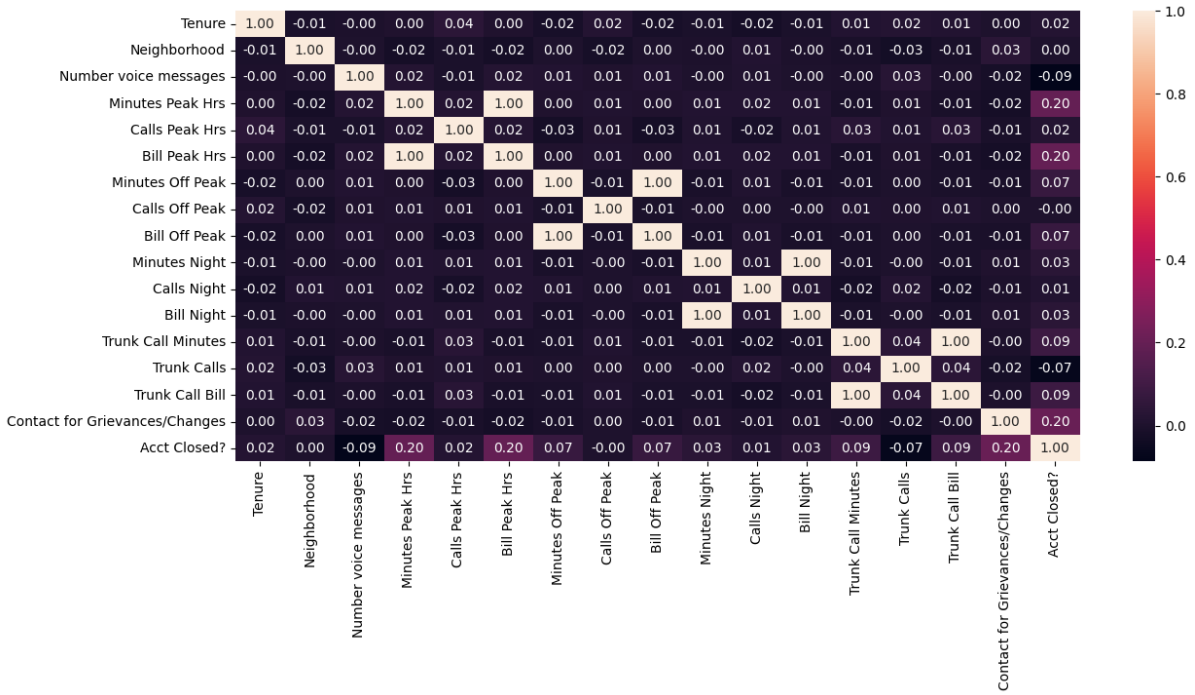
In [ ]:
```
# df = df.drop(columns=['Bill Peak Hrs', 'Bill Off Peak', 'Bill Night', 'Tru
```

In [ ]:
```
df.columns.shape[0]
```

Out[ ]:
```
20
```

In [ ]:
```
plt.figure(figsize=(14,6))
sns.heatmap(df.corr(), annot=True, fmt='0.2f')
```

Out[ ]:
```
<Axes: >
```



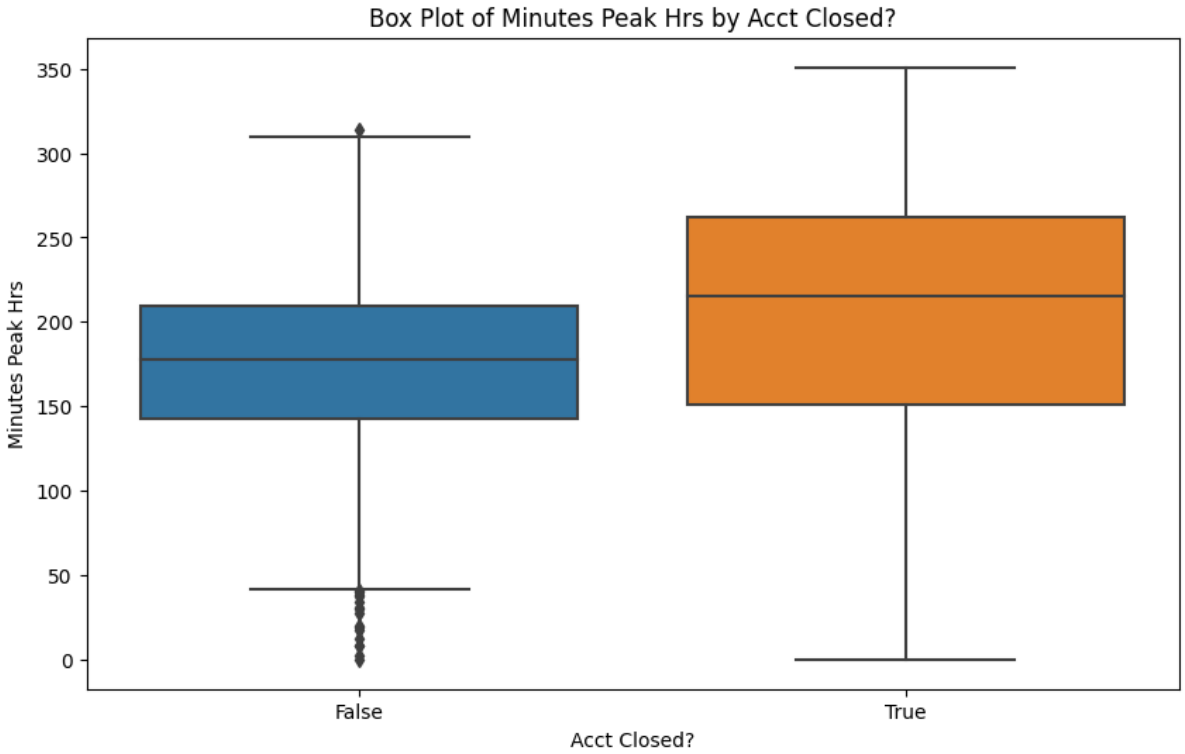
In [ ]:
```
df['Neighborhood'].unique()
```

Out[ ]:
```
array([415, 408, 510])
```

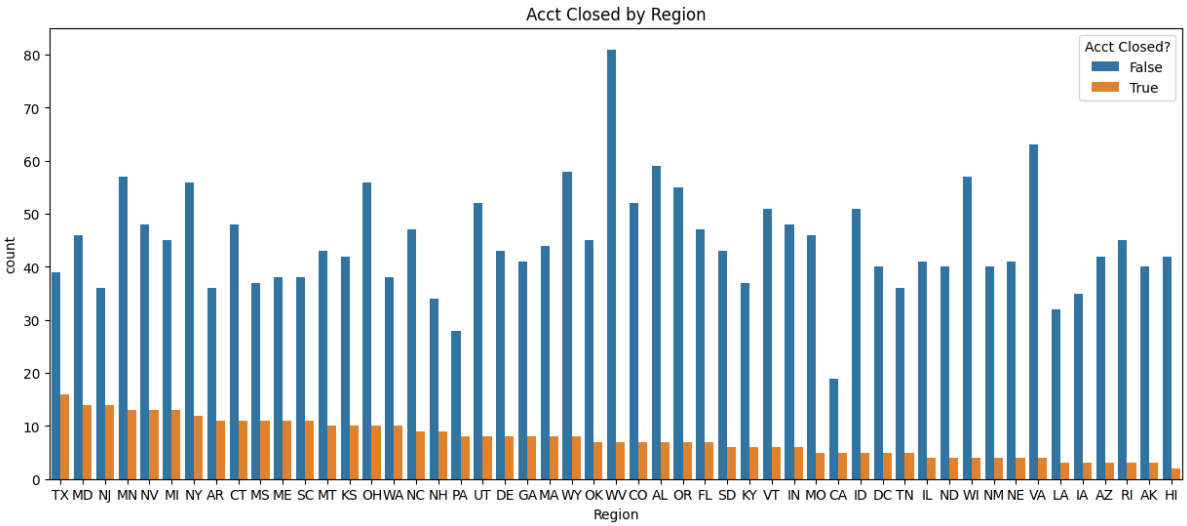
In [ ]:
```
df['Region'].unique().shape[0]
```

Out[ ]:
```
51
```

In [ ]:
```
# For numerical variables vs. target
plt.figure(figsize=(10, 6))
sns.boxplot(x='Acct Closed?', y='Minutes Peak Hrs', data=df)
plt.title('Box Plot of Minutes Peak Hrs by Acct Closed?')
plt.show()
```



In [ ]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Region', hue='Acct Closed?', data=df, order = df.loc[df['Ac
plt.title('Acct Closed by Region')
plt.show()
```



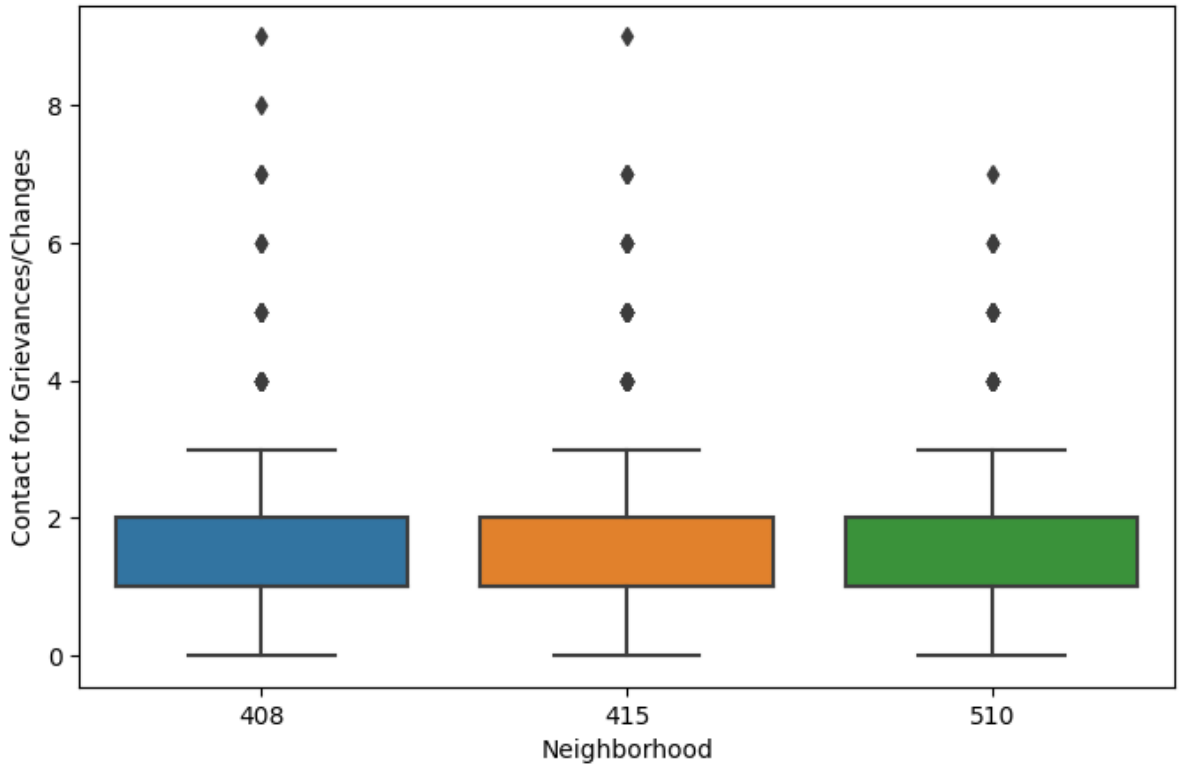
In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Region', 'Tenure', 'Neighborhood', 'Trunk Calling Facility',
       'Voice Messaging', 'Number voice messages', 'Minutes Peak Hrs',
       'Calls Peak Hrs', 'Bill Peak Hrs', 'Minutes Off Peak', 'Calls Off Pe
ak',
       'Bill Off Peak', 'Minutes Night', 'Calls Night', 'Bill Night',
       'Trunk Call Minutes', 'Trunk Calls', 'Trunk Call Bill',
       'Contact for Grievances/Changes', 'Acct Closed?'],
      dtype='object')
```

In [ ]:
```
# # Pair plot for multiple variables with the target variable
# sns.pairplot(df[['Minutes Peak Hrs', 'Calls Peak Hrs', 'Trunk Call Minutes
# plt.show()
```

In [ ]:
```
# # Pair plot for multiple variables with the target variable
# sns.pairplot(df[['Minutes Off Peak', 'Calls Off Peak','Acct Closed?']], hu
# plt.show()
```

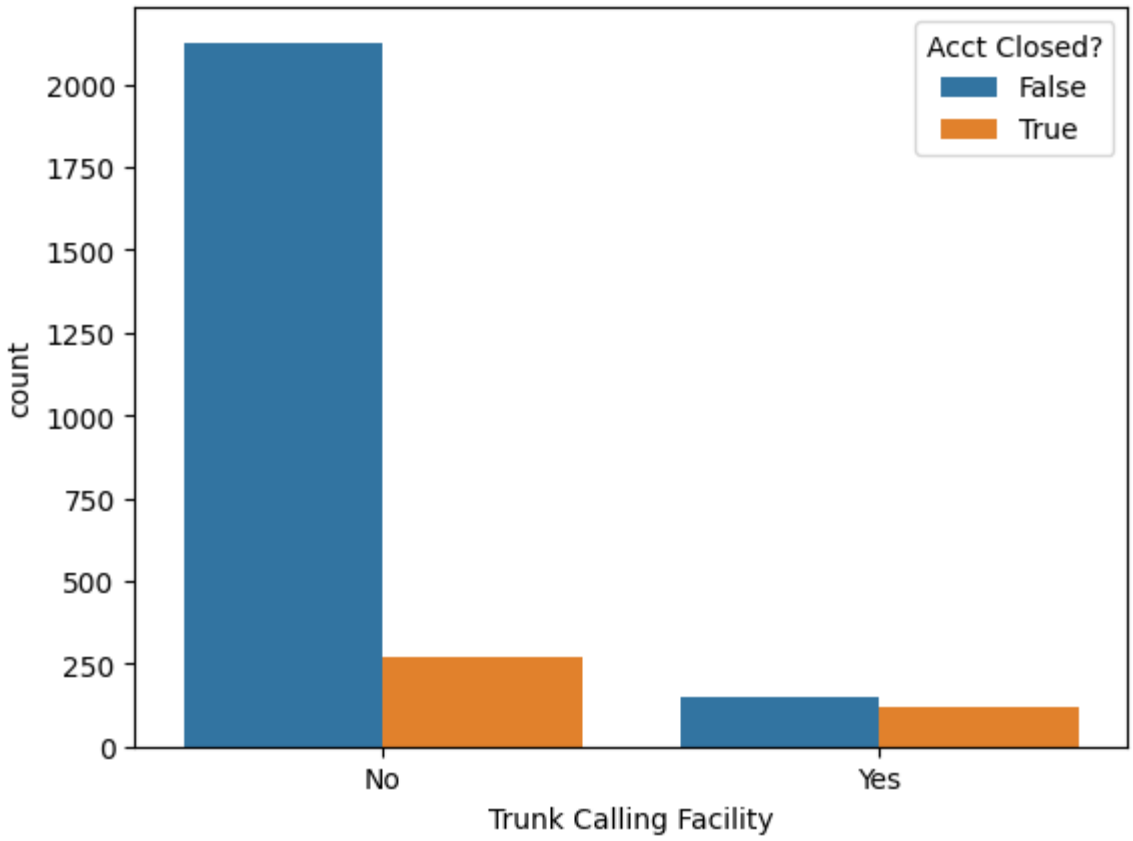
In [ ]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['Neighborhood'], y=df['Contact for Grievances/Changes'])
plt.show()
```



Trunk Calling Facility: Indicates whether the customer has a trunk calling facility

In [ ]:
```
sns.countplot(x='Trunk Calling Facility', hue='Acct Closed?', data=df)
```

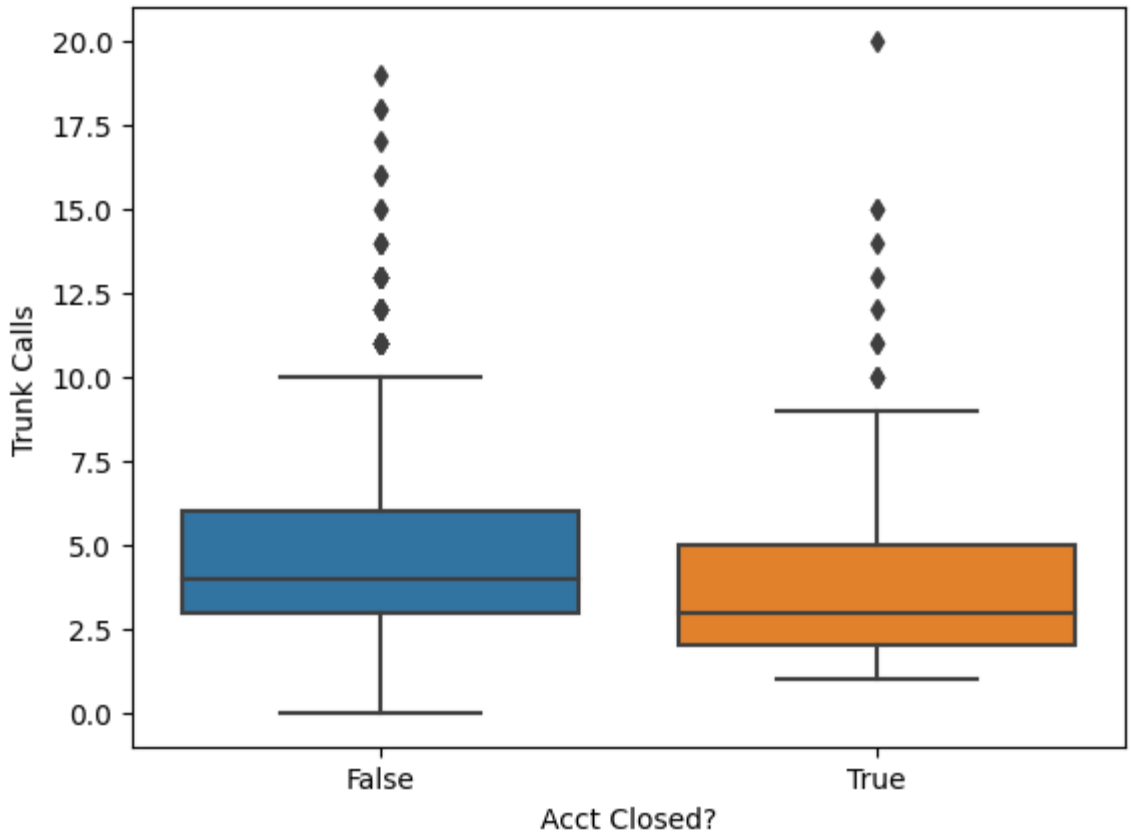
Out[ ]:
```
<Axes: xlabel='Trunk Calling Facility', ylabel='count'>
```



Trunk Calls: The number of trunk calls made.

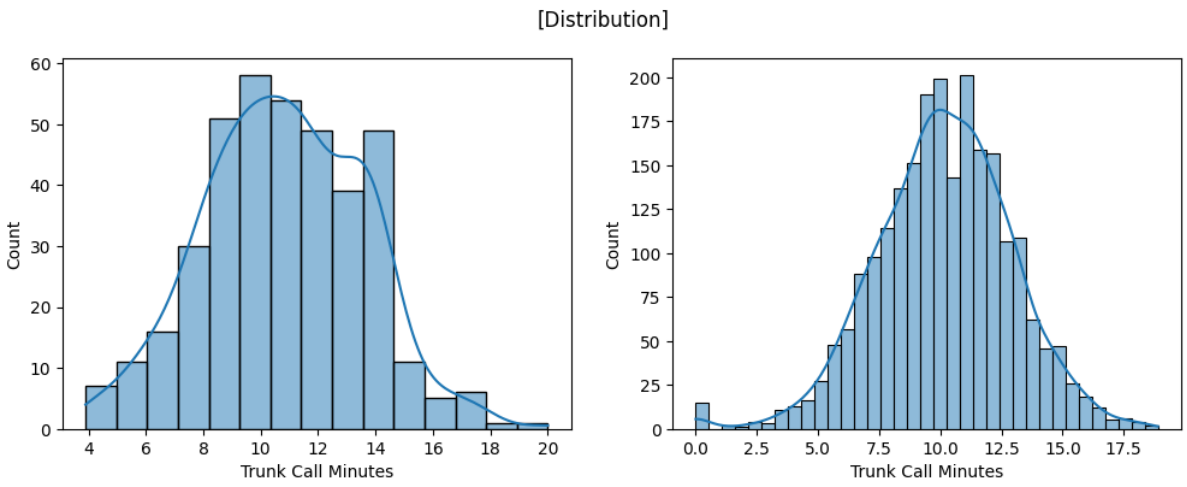
In [ ]:
```
sns.boxplot(y='Trunk Calls', x = 'Acct Closed?',data=df)
```

Out[ ]:
```
<Axes: xlabel='Acct Closed?', ylabel='Trunk Calls'>
```



Trunk Call Minutes: The total number of minutes for trunk calls.

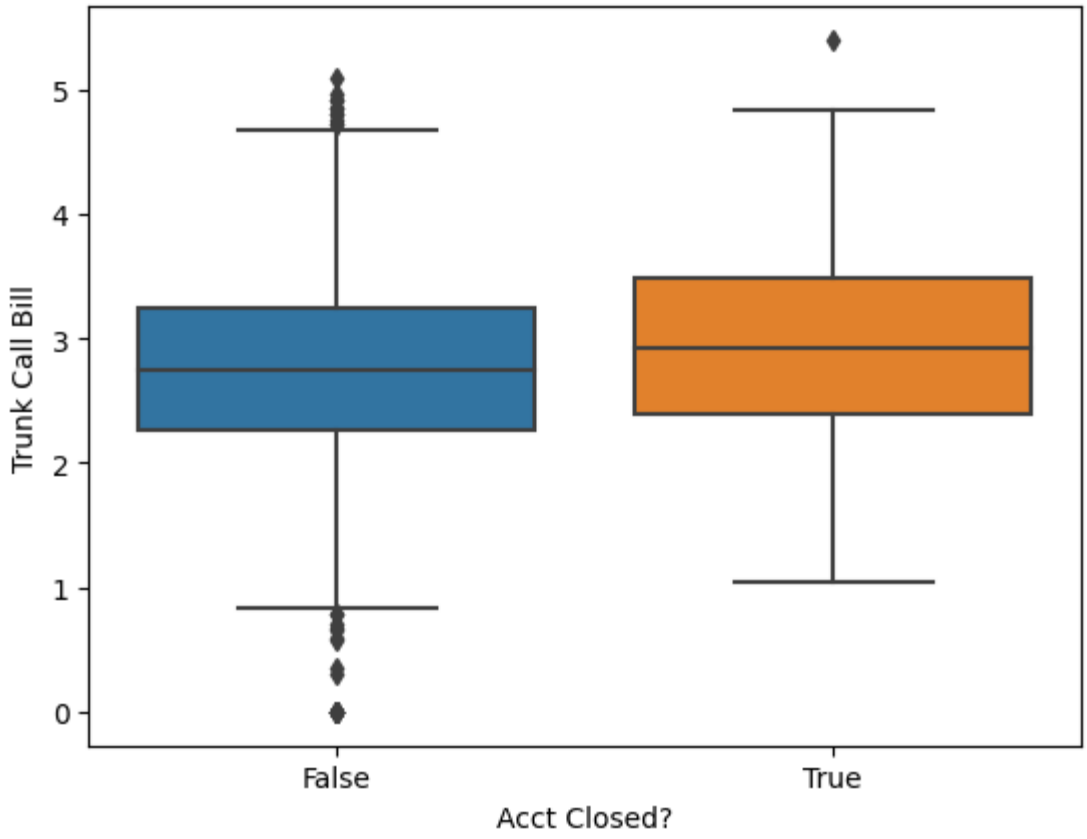
In [ ]:
```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
sns.histplot(df, x=df.loc[df['Acct Closed?']==True]['Trunk Call Minutes'], k

plt.subplot(1,2,2)
sns.histplot(df, x=df.loc[df['Acct Closed?']==False]['Trunk Call Minutes'],

plt.suptitle('[Distribution]')
plt.show()
```



Trunk Call Bill: The bill amount associated with trunk calls.

In [ ]:
```
sns.boxplot(y='Trunk Call Bill', x = 'Acct Closed?',data=df)
```

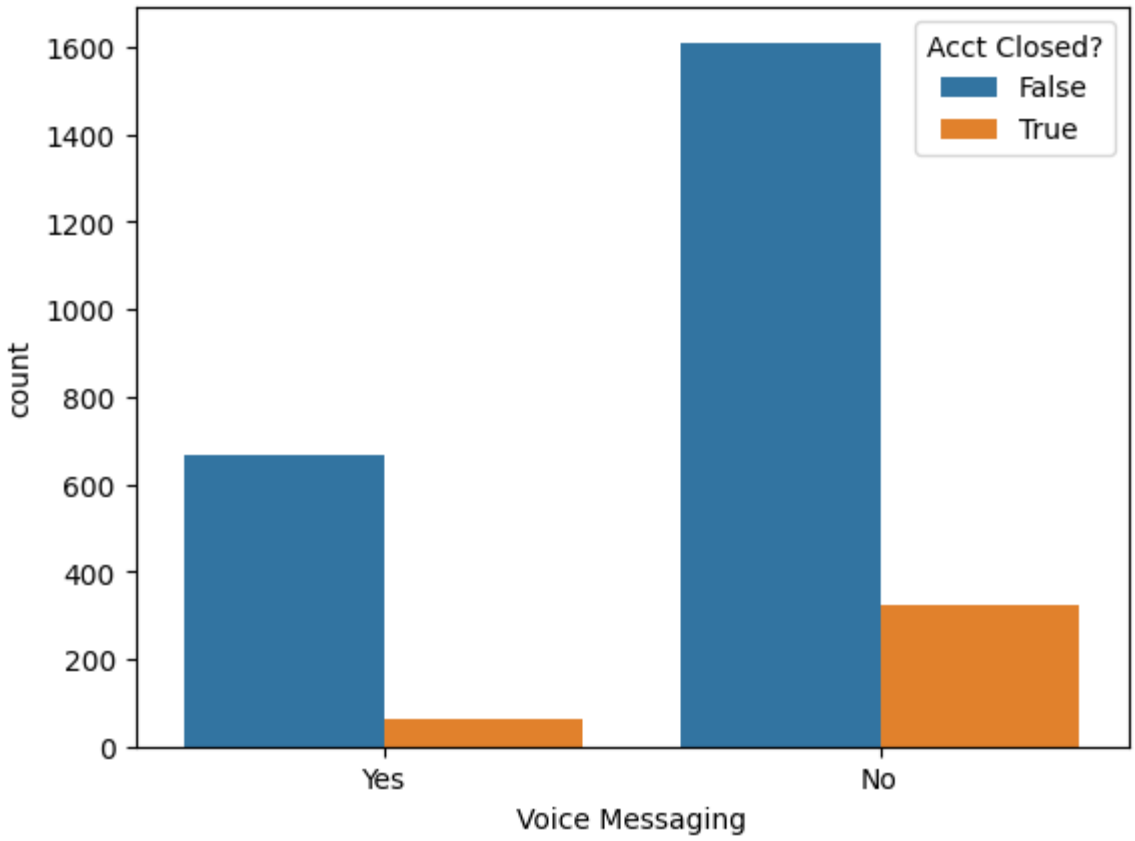
Out[ ]:
```
<Axes: xlabel='Acct Closed?', ylabel='Trunk Call Bill'>
```



Voice Messaging: Indicates whether the customer has voice messaging enabled.

In [ ]:
```
sns.countplot(x='Voice Messaging', hue='Acct Closed?', data=df)
```

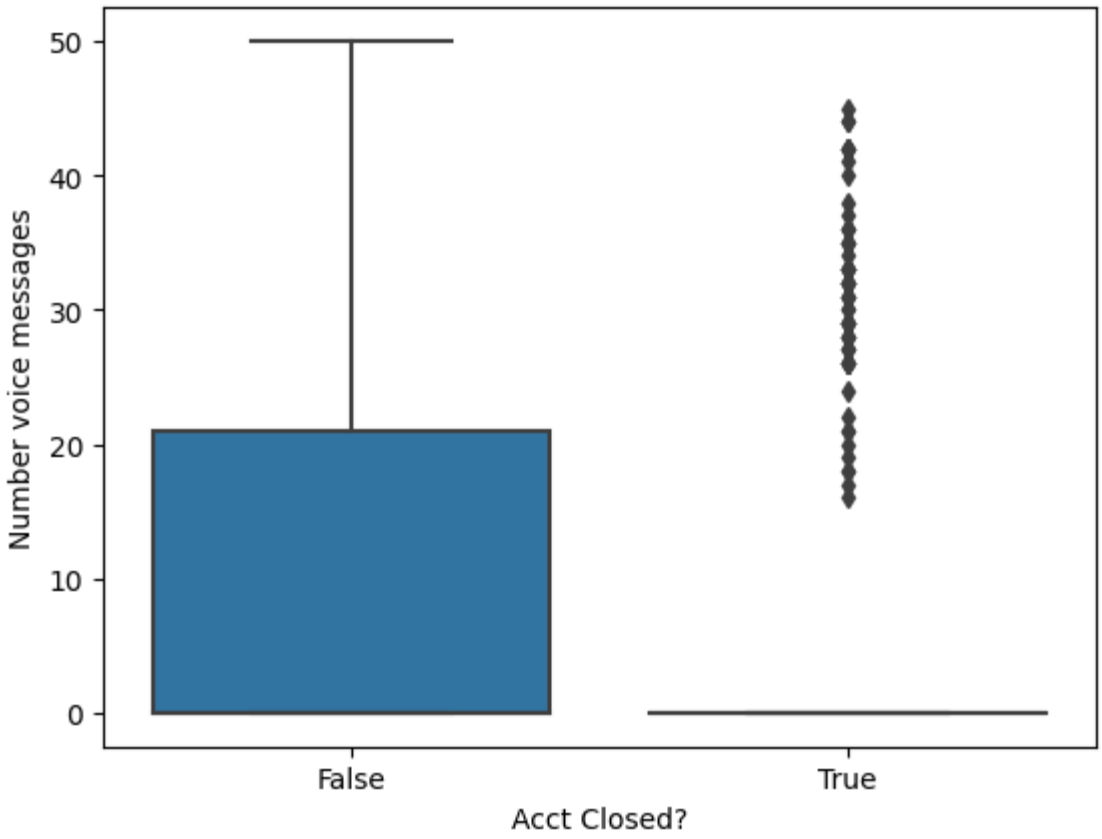
Out[ ]:
```
<Axes: xlabel='Voice Messaging', ylabel='count'>
```



Number voice messages: The number of voice messages sent by the customer.

In [ ]:
```
sns.boxplot(y='Number voice messages', x = 'Acct Closed?',data=df)
```

Out[ ]:
```
<Axes: xlabel='Acct Closed?', ylabel='Number voice messages'>
```



Feature Engineering

Handling Missing and Duplicate values

- There are no missing and duplicate values in the data

Feature Reduction

- 1. Removing "Voice Messaging column" as its information is already available in "Number voice messages" column. e.g. 0 in Number voice messages column indicates No voice messaging facilities.

In [ ]:
```
new_df = df.drop(columns=['Voice Messaging'])
```

- 1. Merge columns of Minutes data and Bill data as it provides similar information

In [ ]:
```
new_df['Merged_Minutes_Calls_Peak_Hrs'] = df['Minutes Peak Hrs'] * df['Calls
new_df['Merged_Minutes_Calls_Off_Peak'] = df['Minutes Off Peak'] * df['Calls
new_df['Merged_Minutes_Calls_Night'] = df['Minutes Night'] * df['Calls Night
new_df['Merged_Minutes_Calls_Trunk'] = df['Trunk Call Minutes'] * df['Trunk
```

- 1. Merge Bill Columns to create new feature

In [ ]:
```
new_df['Total_Bill'] = df['Bill Peak Hrs'] + df['Bill Off Peak'] + df['Bill
```

- 1. Drop the columns which was used to create merged feature, as it will be highly correated with each other.

In [ ]:
```
new_df = new_df.drop(columns=['Minutes Peak Hrs', 'Calls Peak Hrs', 'Minutes
                              'Calls Night', 'Trunk Call Minutes', 'Trunk Ca
                              'Bill Night', 'Trunk Call Bill'])
```

In [ ]:
```
plt.figure(figsize=(10,6))
sns.heatmap(new_df.corr(), annot=True, fmt='0.2f')
```

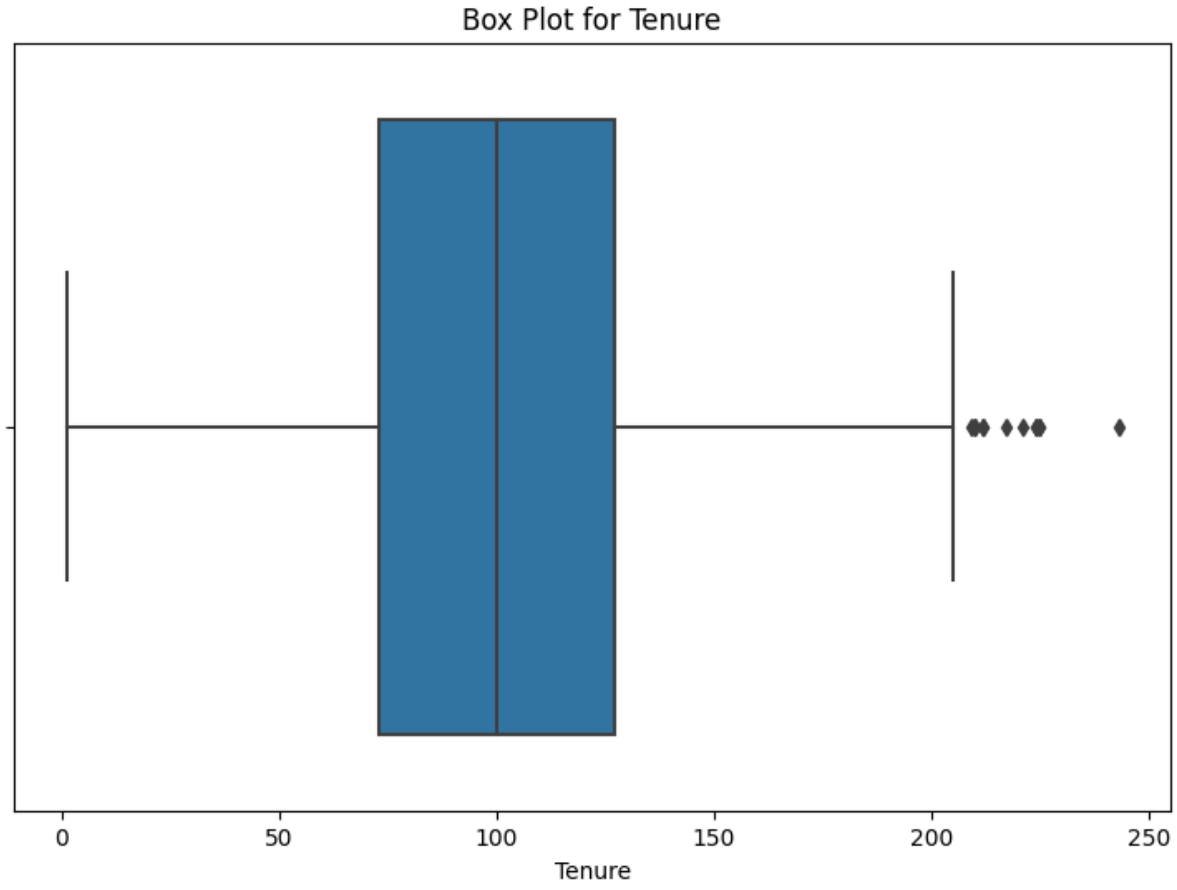
Out[ ]:
```
<Axes: >
```

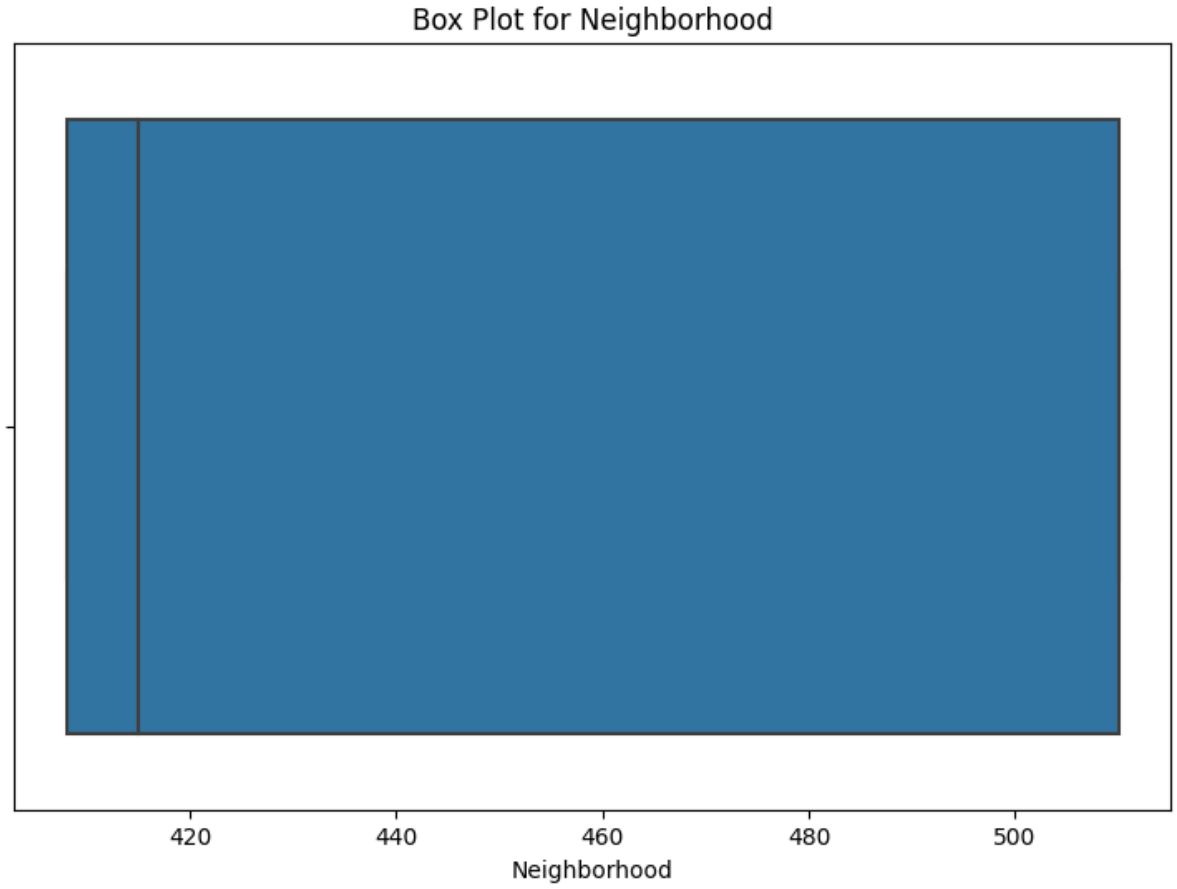
Identifying Outliers

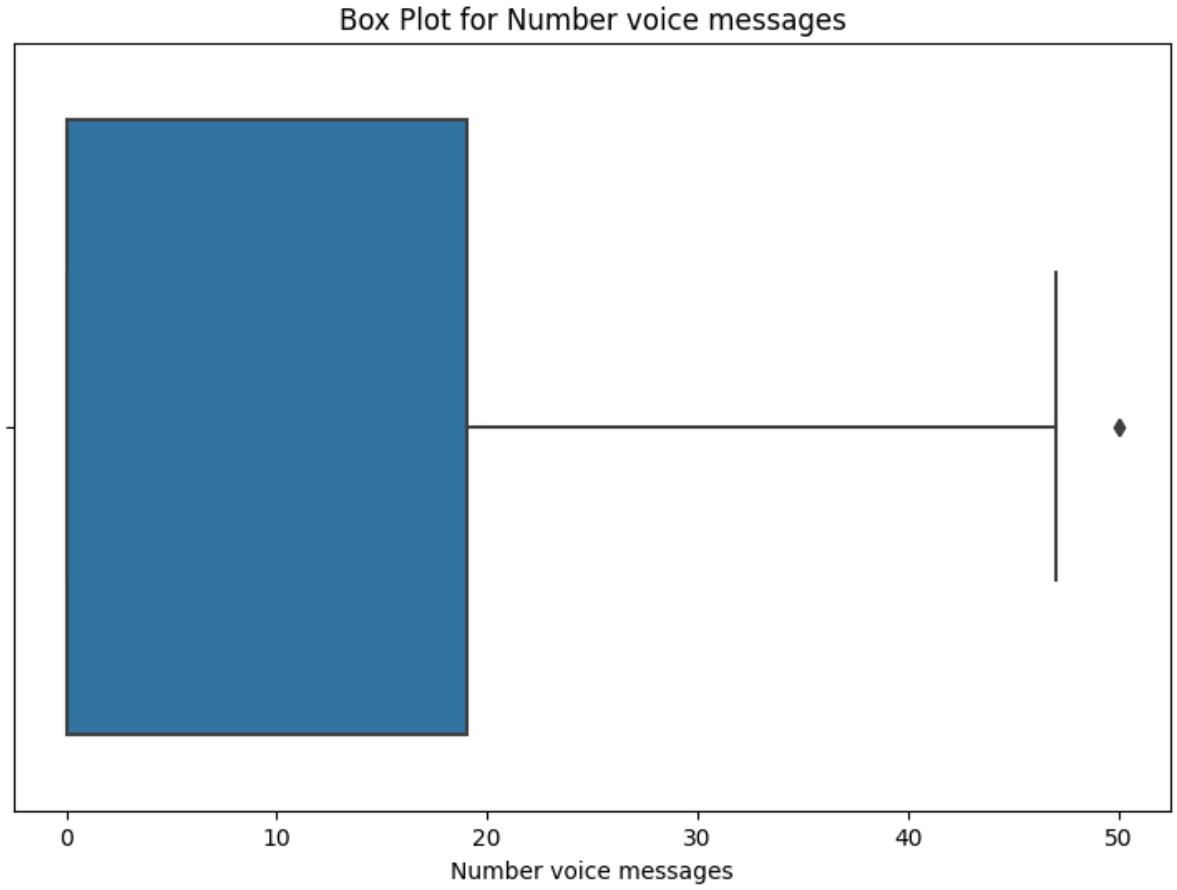
In [ ]:
```
# Taking numeric columns
numeric_columns = new_df.select_dtypes(include=['float64', 'int64']).columns

# Calculateing IQR
Q1 = new_df[numeric_columns].quantile(0.25)
Q3 = new_df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1

# Outlier threshold - 1.5 times IQR
threshold = 1.5

# Finding the outliers using the threshold value
outliers = np.logical_or(new_df[numeric_columns] < (Q1 - threshold * IQR), r

# Total outliers in each numerical columns
outliers_count = outliers.sum()
print(outliers_count.sort_values())
```

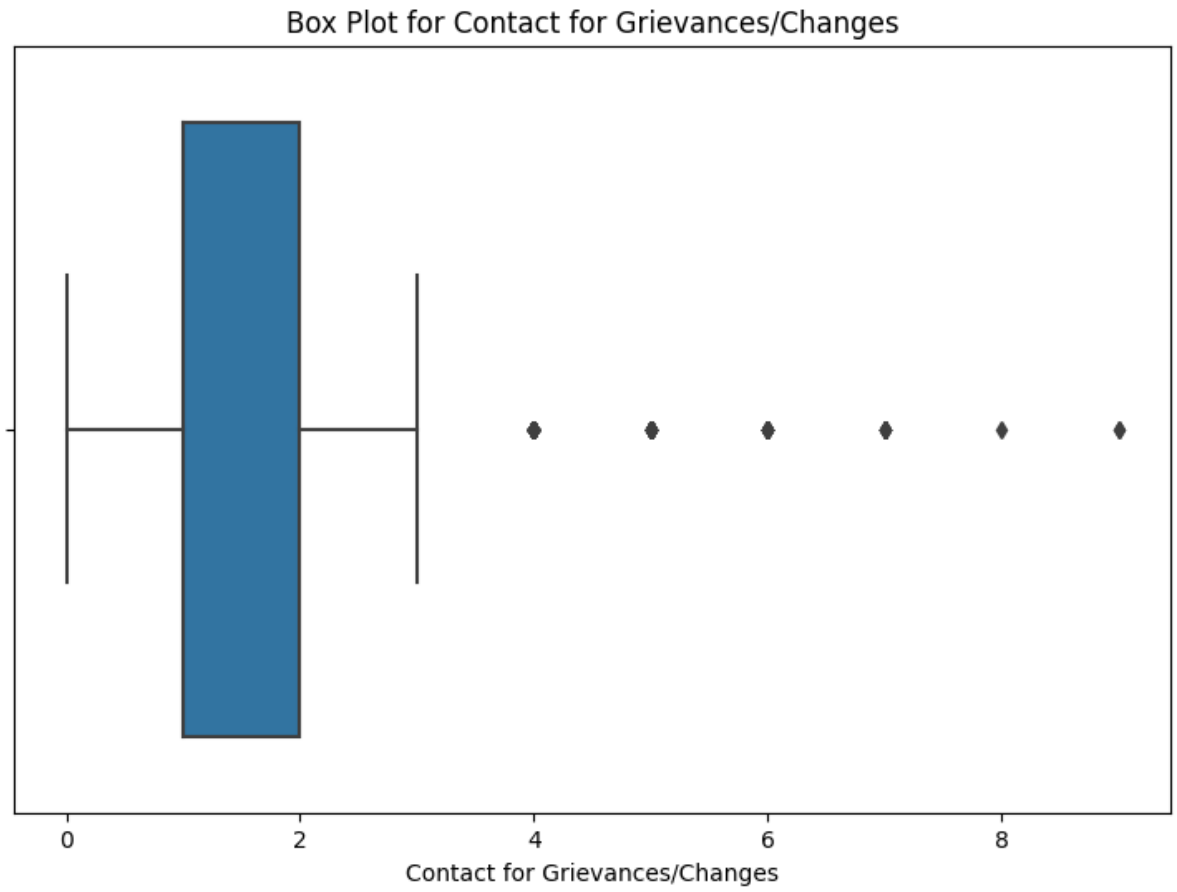
```
Neighborhood                        0
Number voice messages               2
Tenure                             12
Total_Bill                         21
Merged_Minutes_Calls_Off_Peak      24
Merged_Minutes_Calls_Peak_Hrs      25
Merged_Minutes_Calls_Night         27
Merged_Minutes_Calls_Trunk         97
Contact for Grievances/Changes    210
dtype: int64
```

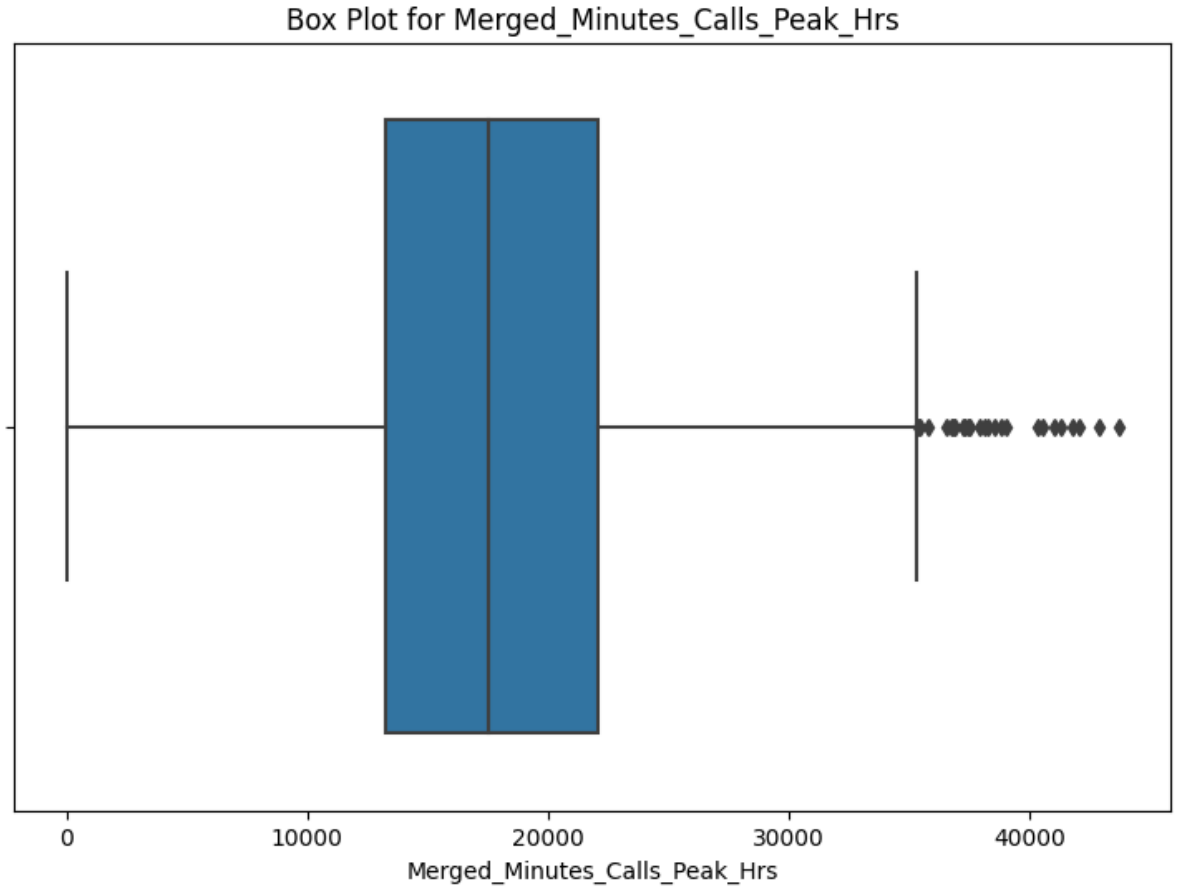
In [ ]:
```
# ploting each column using box plot
for column in numeric_columns:
    plt.figure(figsize=(9, 6))
    sns.boxplot(x=new_df[column])
    plt.title(f'Box Plot for {column}')
    plt.show()
```

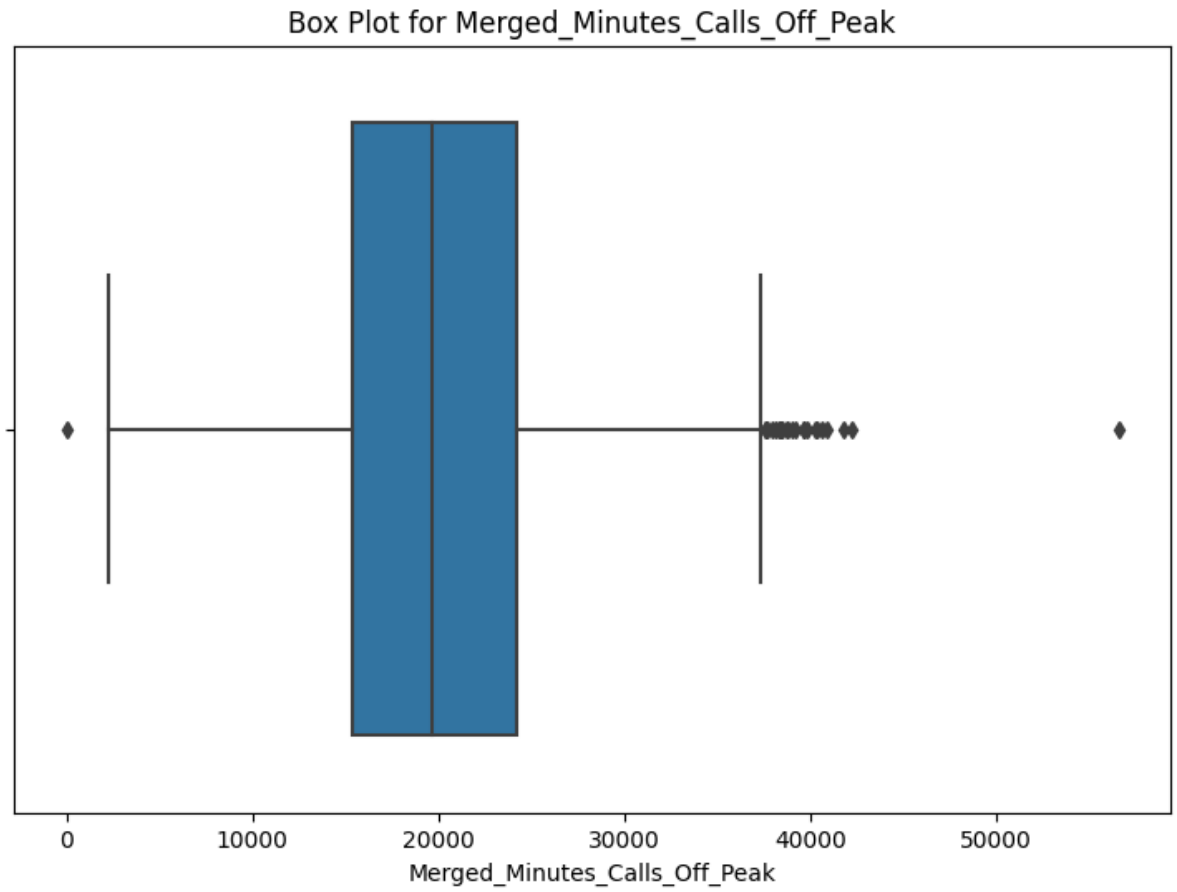

Box Plot for Tenure
0
50
100
150
200
250
Tenure

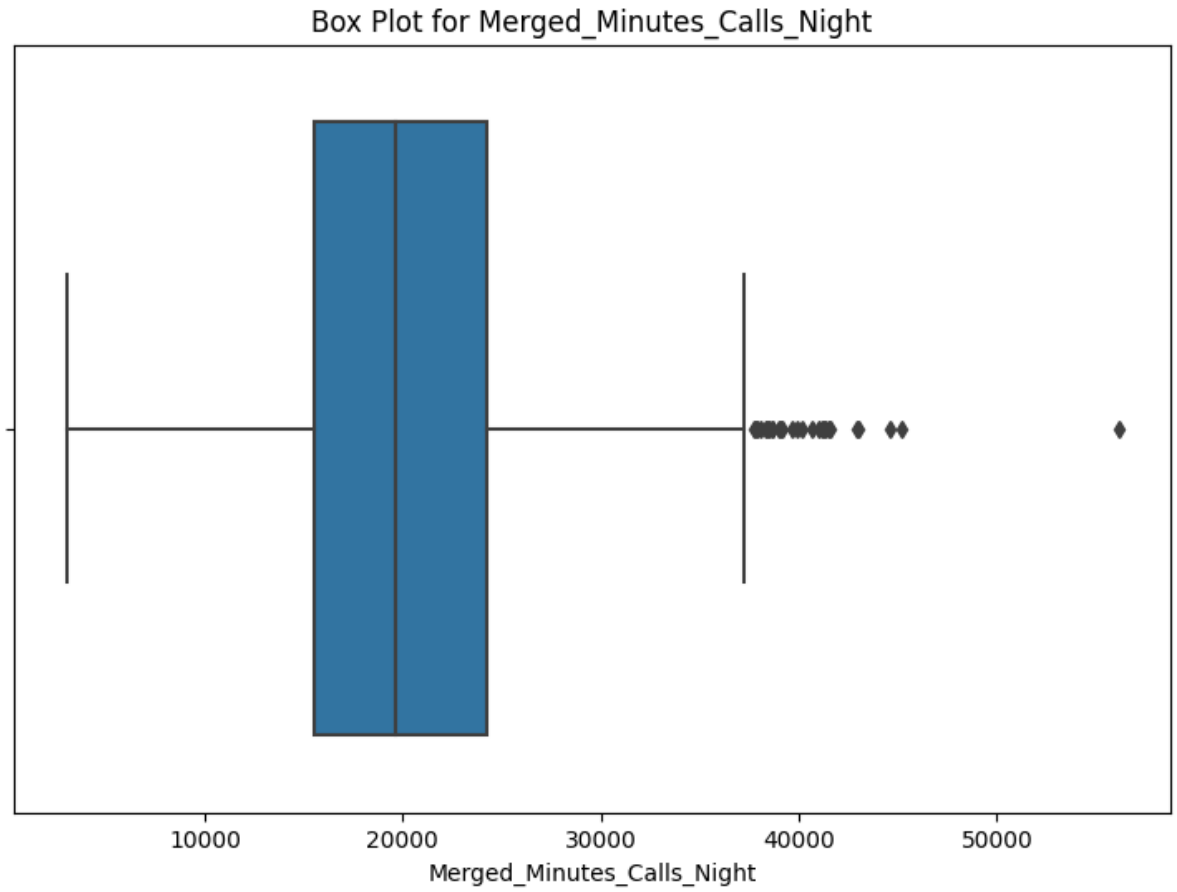

Box Plot for Neighborhood
420
440
460
480
500
Neighborhood

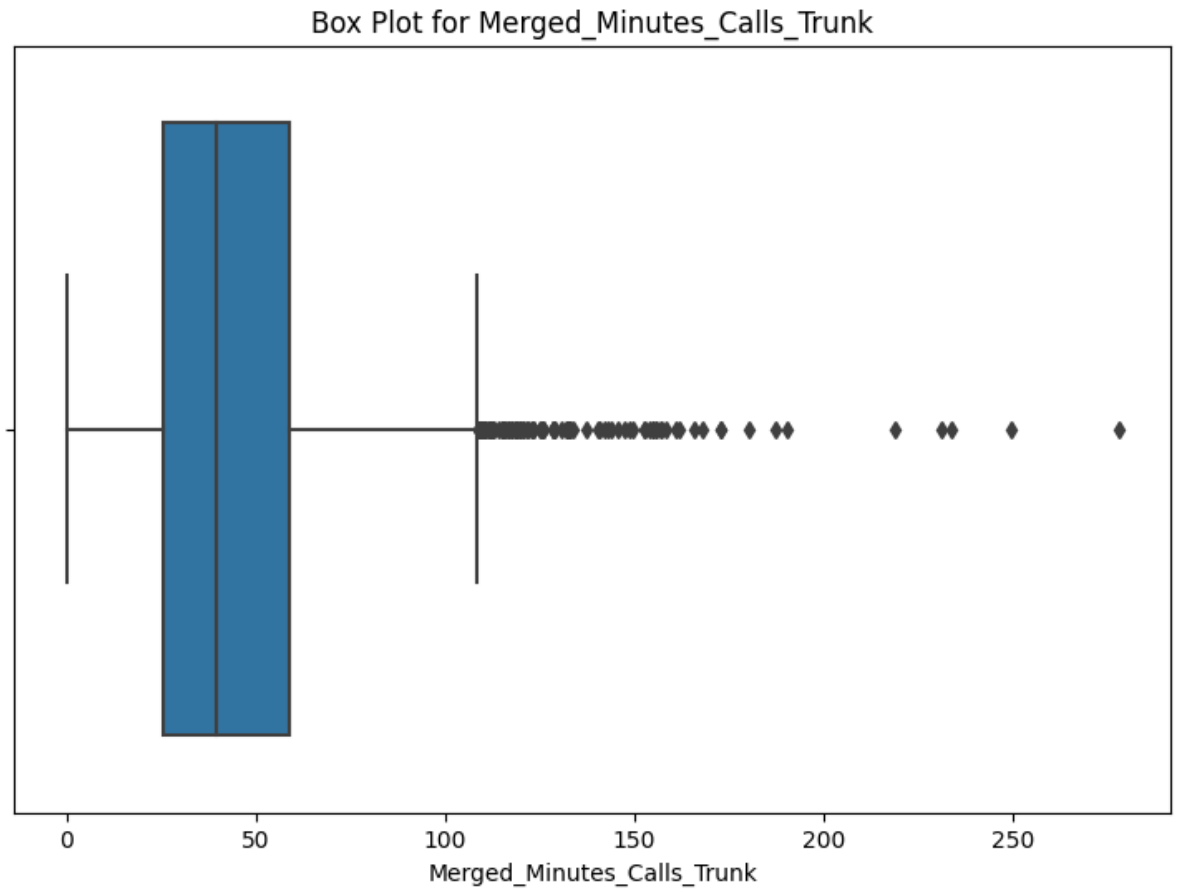

Box Plot for Number voice messages
0
10
20
30
40
50
Number voice messages

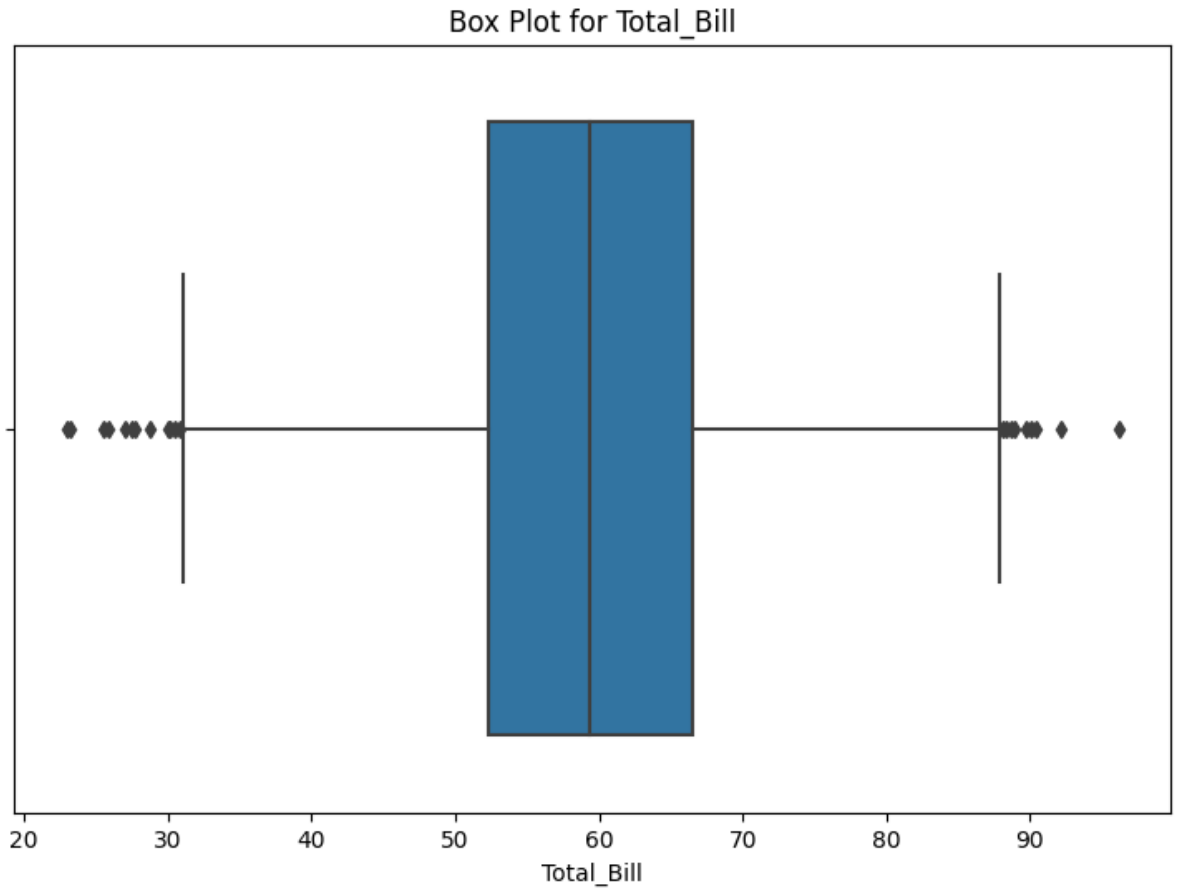

Box Plot for Contact for Grievances/Changes
0
2
4
6
8
Contact for Grievances/Changes


Box Plot for Merged_Minutes_Calls_Peak_Hrs
0
10000
20000
30000
40000
Merged_Minutes_Calls_Peak_Hrs


Box Plot for Merged_Minutes_Calls_Off_Peak
0
10000
20000
30000
40000
50000
Merged_Minutes_Calls_Off_Peak


Box Plot for Merged_Minutes_Calls_Night
10000
20000
30000
40000
50000
Merged_Minutes_Calls_Night


Box Plot for Merged_Minutes_Calls_Trunk
0
50
100
150
200
250
Merged_Minutes_Calls_Trunk



Removing Outliers From Quantitative Columns

In [ ]:
```
# Taking numeric columns which are quantitative in nature
numeric_columns = new_df.select_dtypes(include=['float64', 'int64']).columns

# Calculateing IQR
Q1 = new_df[numeric_columns].quantile(0.25)
Q3 = new_df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1

# Outlier threshold - 1.5 times IQR
threshold = 1.5

# Finding the outliers using the threshold value
outliers_num = np.logical_or(new_df[numeric_columns] < (Q1 - threshold * IQR

# Total outliers in each numerical columns
outliers_count_num = outliers_num.sum()
print(outliers_count_num.sort_values())
```

```
Number voice messages            2
Tenure                          12
Total_Bill                      21
Merged_Minutes_Calls_Off_Peak   24
Merged_Minutes_Calls_Peak_Hrs   25
Merged_Minutes_Calls_Night      27
Merged_Minutes_Calls_Trunk      97
dtype: int64
```

In [ ]:
```
# Remove the outliers
df_no_outliers_num = new_df[~outliers_num.any(axis=1)]
```

In [ ]:
```
# Display the shape
print(f"Shape before: {new_df.shape}")
print(f"Shape after: {df_no_outliers_num.shape}")
```

```
Shape before: (2666, 12)
Shape after: (2466, 12)
```

Removing Outliers From Qualitative Integer Column

- "Contact for Grievances/Changes" column has outliers, but as it seems like a nominal data ranges from 0 to 9. And as there is no much information is mentioned, it is very challenging to precisely interpret the meaning of these values. Therefore I am not performing any operation as of now. If it is degrading the model's performance then we can remove these outliers and check the performance again.

One Hot Encoding and data splitting

In [ ]:
```
# Identifying Categorical and boolean Columns
list(df_no_outliers_num.select_dtypes(include=['object', 'bool']).columns)
```

Out[ ]:
```
['Region', 'Trunk Calling Facility', 'Acct Closed?']
```

In [ ]:
```
df_encoded = pd.get_dummies(df_no_outliers_num, columns = ['Region'])
```

In [ ]:
```
df_encoded['Trunk Calling Facility'] = df_encoded['Trunk Calling Facility'].
```

In [ ]:
```
df_encoded['Acct Closed?'] = df_encoded['Acct Closed?'].astype('int')
```

In [ ]:
```
df_encoded.head()
```

Out[ ]:

	Tenure	Neighborhood	Trunk Calling Facility	Number voice messages	Contact for Grievances/Changes	Acct Closed?	Merged_Minute
0	128	415	0	25	1	0	
1	107	415	0	26	1	0	
2	137	415	0	0	0	0	
3	84	408	1	0	2	0	
4	75	415	1	0	3	0	

5 rows × 62 columns

In [ ]:
```
# Splitting the data into train and test
X = df_encoded.drop(columns=['Acct Closed?'])

y = df_encoded['Acct Closed?']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(1972, 61)
```

In [ ]:
```
y_train.shape
```

Out[ ]:
```
(1972,)
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(494, 61)
```

In [ ]:
```
y_test.shape
```

Out[ ]:
```
(494,)
```

Data Scaling

In [ ]:
```
# Scaling the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Radom Over Sampling on imbalanced data

In [ ]:
```
!pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.1
0/dist-packages (0.10.1)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.6/235.6 kB 4.8 MB/s eta
0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/d
ist-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/di
st-packages (from imbalanced-learn) (1.11.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python
3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/d
ist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
n3.10/dist-packages (from imbalanced-learn) (3.2.0)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.11.0
```

In [ ]:
```
# Over sampling
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

In [ ]:
```
# Data Imbalance
df_encoded['Acct Closed?'].value_counts()
```

Out[ ]:
```
0    2126
1     340
Name: Acct Closed?, dtype: int64
```

In [ ]:
```
# Data Imbalance
(df_encoded['Acct Closed?'].value_counts()/df_encoded['Acct Closed?'].count(
```

Out[ ]:
```
0    0.862125
1    0.137875
Name: Acct Closed?, dtype: float64
```

In [ ]:
```
# Randomly over sample the minority class
os = RandomOverSampler(sampling_strategy='minority', random_state=42)
X_train_os, y_train_os = os.fit_resample(X_train, y_train)

# Check the number of records after over sampling
print(sorted(Counter(y_train_os).items()))
```

```
[(0, 1696), (1, 1696)]
```

Grid Search

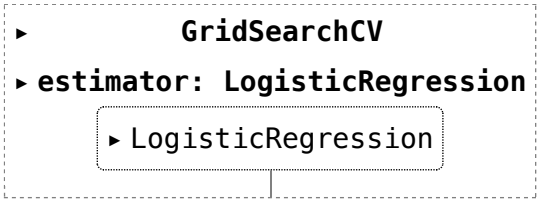
1. Create param grid: where we first add hyperparameters and penalty for each wrong analysis
2. Then we create LogisticRegression()
3. Which can be passed to grid search inbuilt function
4. Basically grid search function needs logestic algo object, param grid, number of iteration and scoring
5. Then we can fit X train and Y train to find best hyper-parameter.

In [ ]:
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty': ['l1', '
```

In [64]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

In [ ]:
```
logreg_model = LogisticRegression()
grid_search = GridSearchCV(logreg_model, param_grid, cv = 5, scoring = 'accu
grid_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
best_params = grid_search.best_params_
```

In [ ]:
```
best_params
```

Out[ ]:
```
{'C': 0.01, 'penalty': 'l2'}
```

K-Fold Cross-Validation with Best Hyperparameters

In [61]:
```
best_logreg_model = LogisticRegression(**best_params)
```

In [62]:
```
K_folds = 5
```

In [66]:
```
cv_score = cross_val_score(best_logreg_model, X_train, y_train, cv = K_folds
```

In [68]:
```
print("Cross-validation scores:", cv_score)
```

```
Cross-validation scores: [0.86582278 0.85822785 0.86548223 0.86294416 0.855
32995]
```

In [71]:
```
print(f"mean accuracy: {cv_score.mean():.2f}")
print(f"Standard deviation: {cv_score.std():.2f}")
```

```
mean accuracy: 0.86
Standard deviation: 0.00
```

Now train the final model on entire dataset

In [72]:
```
final_model = LogisticRegression(**best_params)
final_model.fit(X_train, y_train)
```

Out[72]:
```
▾  LogisticRegression

LogisticRegression(C=0.01)
```

Evaluate the final model

In [75]:
```
final_test_accuracy = final_model.score(X_test, y_test)
print(f"Final Test accuracy: {final_test_accuracy:.3f}")
```

```
Final Test accuracy: 0.874
```