In [15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore") # suppress warnings for readability
```

In [16]:
```
# Import churn.csv
telco = pd.read_csv('../data/churn_data.csv')
telco
```

Out[16]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	Intl_Mins
0	128	25	265.1	197.4	244.7	10.0
1	107	26	161.6	195.5	254.4	13.7
2	137	0	243.4	121.2	162.6	12.2
3	84	0	299.4	61.9	196.9	6.6
4	75	0	166.7	148.3	186.9	10.1
...	...	...	...	...	...	...
3328	192	36	156.2	215.5	279.1	9.9
3329	68	0	231.1	153.4	191.3	9.6
3330	28	0	180.8	288.8	191.9	14.1
3331	184	0	213.8	159.6	139.2	5.0
3332	74	25	234.4	265.9	241.4	13.7

3333 rows × 21 columns

EDA

In [17]:
```
telco.head()
```

Out[17]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	Intl_Mins	Cus
0	128	25	265.1	197.4	244.7	10.0	
1	107	26	161.6	195.5	254.4	13.7	
2	137	0	243.4	121.2	162.6	12.2	
3	84	0	299.4	61.9	196.9	6.6	
4	75	0	166.7	148.3	186.9	10.1	

5 rows × 21 columns

In [18]:
```
telco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Account_Length  3333 non-null   int64  
 1   Vmail_Message   3333 non-null   int64  
 2   Day_Mins        3333 non-null   float64
 3   Eve_Mins        3333 non-null   float64
 4   Night_Mins      3333 non-null   float64
 5   Intl_Mins       3333 non-null   float64
 6   CustServ_Calls  3333 non-null   int64  
 7   Churn           3333 non-null   object 
 8   Intl_Plan       3333 non-null   object 
 9   Vmail_Plan      3333 non-null   object 
 10  Day_Calls       3333 non-null   int64  
 11  Day_Charge      3333 non-null   float64
 12  Eve_Calls       3333 non-null   int64  
 13  Eve_Charge      3333 non-null   float64
 14  Night_Calls     3333 non-null   int64  
 15  Night_Charge    3333 non-null   float64
 16  Intl_Calls      3333 non-null   int64  
 17  Intl_Charge     3333 non-null   float64
 18  State           3333 non-null   object 
 19  Area_Code       3333 non-null   int64  
 20  Phone           3333 non-null   object 
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
```

In [19]:
```
telco.describe()
```

Out[19]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	33
mean	101.064806	8.099010	179.775098	200.980348	200.872037	
std	39.822106	13.688365	54.467389	50.713844	50.573847	
min	1.000000	0.000000	0.000000	0.000000	23.200000	
25%	74.000000	0.000000	143.700000	166.600000	167.000000	
50%	101.000000	0.000000	179.400000	201.400000	201.200000	
75%	127.000000	20.000000	216.400000	235.300000	235.300000	
max	243.000000	51.000000	350.800000	363.700000	395.000000	

In [20]:

```
telco['Churn'].value_counts()
```

Out[20]:

```
Churn
no     2850
yes     483
Name: count, dtype: int64
```

In [21]:

```
# Group telco by 'Churn' and compute the mean
telco.groupby(['Churn']).mean(numeric_only=True)

# Adapt your code to compute the standard deviation
telco.groupby(['Churn']).std(numeric_only=True)
```

Out[21]:

Churn	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	Intl_Mins
no	39.88235	13.913125	50.181655	50.292175	51.105032	2.784489
yes	39.46782	11.860138	68.997792	51.728910	47.132825	2.793190

In [22]:

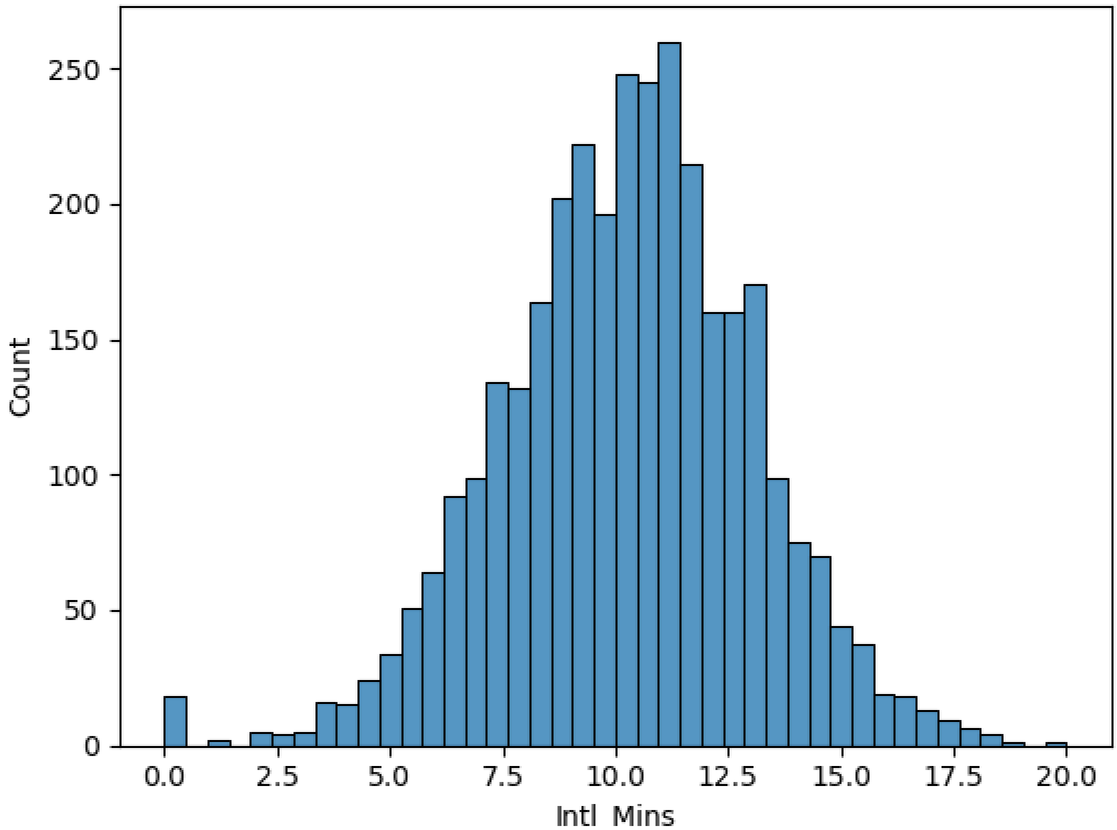
```
# Count the number of churners and non-churners by State
telco.groupby('State')['Churn'].value_counts()

# Filter the data to show only California
telco[telco['State'] == 'CA'].groupby('State')['Churn'].value_counts()
```

Out[22]:

```
State  Churn
CA     no       25
       yes       9
Name: count, dtype: int64
```

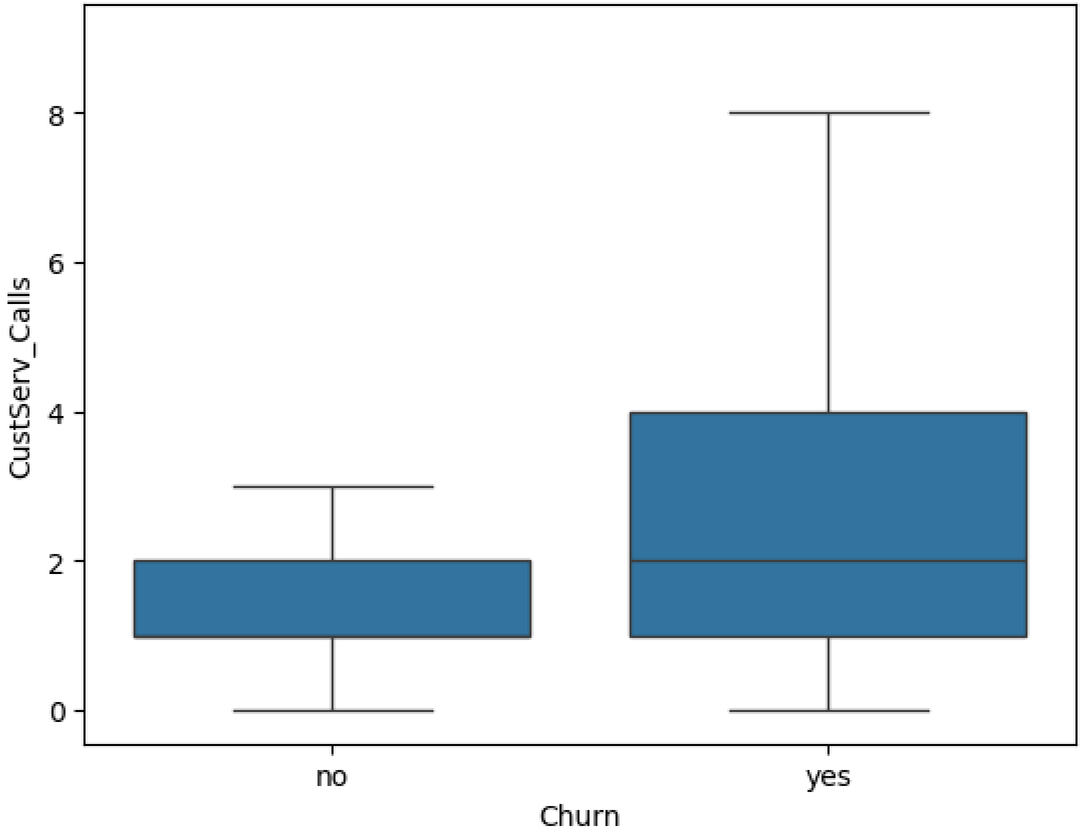
In [23]:

```
# Visualize the distribution of 'Intl_Mins'
sns.histplot(telco['Intl_Mins'])

# Display the plot
plt.show()
```



In [24]:

```
# Add "Intl_Plan" as a third variable
sns.boxplot(x = 'Churn',
            y = 'CustServ_Calls',
            data = telco,
            fliersize = 0, # Remove outliers
            # hue = "Intl_Plan" # Add third variable
            )

# Display the plot
plt.show()
```



Data Preprocessing

Encoding Binary Features

In [25]:
```
pd.set_option('future.no_silent_downcasting', True)

# Replace 'no' with 0 and 'yes' with 1 in 'Vmail_Plan'
# telco['Vmail_Plan'] = telco['Vmail_Plan'].replace({'no': 0 , 'yes': 1})
telco['Vmail_Plan'] = telco['Vmail_Plan'].replace({'no': 0 , 'yes': 1}).infe

# Replace 'no' with 0 and 'yes' with 1 in 'Churn'
# telco['Churn'] = telco['Churn'].replace({'no': 0 , 'yes': 1})
telco['Churn'] = telco['Churn'].replace({'no': 0 , 'yes': 1}).infer_objects(

# Print the results to verify
print(telco['Vmail_Plan'].head())
print(telco['Churn'].head())
```

```
0    1
1    1
2    0
3    0
4    0
Name: Vmail_Plan, dtype: int64
0    0
1    0
2    0
3    0
4    0
Name: Churn, dtype: int64
```

Encoding Categorical Features using One-Hot Encoding

In [26]:
```
telco_state = pd.get_dummies(telco, columns=['State'], drop_first=True)
telco_state
```

Out[26]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	Intl_Mins
0	128	25	265.1	197.4	244.7	10.0
1	107	26	161.6	195.5	254.4	13.7
2	137	0	243.4	121.2	162.6	12.2
3	84	0	299.4	61.9	196.9	6.6
4	75	0	166.7	148.3	186.9	10.1
...	...	...	...	...	...	...
3328	192	36	156.2	215.5	279.1	9.9
3329	68	0	231.1	153.4	191.3	9.6
3330	28	0	180.8	288.8	191.9	14.1
3331	184	0	213.8	159.6	139.2	5.0
3332	74	25	234.4	265.9	241.4	13.7

3333 rows × 70 columns

In [27]:
```
from sklearn.preprocessing import LabelEncoder

# Encode categorical columns
telco['Intl_Plan'] = LabelEncoder().fit_transform(telco['Intl_Plan'])
```

Feature Scaling using Standardization

In [28]:
```
# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Select only the numeric columns for scaling
numeric_cols = telco.select_dtypes(include=['float64', 'int64']).columns
```

```python
# Scale telco
telco_scaled = StandardScaler().fit_transform(telco[numeric_cols])

# Add column names back for readability
telco_scaled_df = pd.DataFrame(telco_scaled, columns=numeric_cols)

# Print summary statistics
print(telco_scaled_df.describe())
```

```
       Account_Length  Vmail_Message      Day_Mins      Eve_Mins  \
count    3.333000e+03   3.333000e+03  3.333000e+03  3.333000e+03   
mean     1.470971e-16   7.035077e-17  7.312216e-16 -6.821892e-17   
std      1.000150e+00   1.000150e+00  1.000150e+00  1.000150e+00   
min     -2.513172e+00  -5.917599e-01 -3.301096e+00 -3.963622e+00   
25%     -6.797448e-01  -5.917599e-01 -6.624241e-01 -6.780300e-01   
50%     -1.627644e-03  -5.917599e-01 -6.887677e-03  8.276141e-03   
75%      6.513740e-01   8.695542e-01  6.725198e-01  6.768330e-01   
max      3.564766e+00   3.134591e+00  3.140422e+00  3.209066e+00   

         Night_Mins     Intl_Mins  CustServ_Calls         Churn     Intl_Pla
n  \
count  3.333000e+03  3.333000e+03    3.333000e+03  3.333000e+03  3.333000e+0
3   
mean   7.887813e-17 -3.336332e-16    8.527366e-18  5.542788e-17 -4.796643e-1
7   
std    1.000150e+00  1.000150e+00    1.000150e+00  1.000150e+00  1.000150e+0
0   
min   -3.513648e+00 -3.667413e+00   -1.188218e+00 -4.116718e-01 -3.275805e-0
1   
25%   -6.698545e-01 -6.223690e-01   -4.279320e-01 -4.116718e-01 -3.275805e-0
1   
50%    6.485803e-03  2.246393e-02   -4.279320e-01 -4.116718e-01 -3.275805e-0
1   
75%    6.808485e-01  6.672969e-01    3.323545e-01 -4.116718e-01 -3.275805e-0
1   
max    3.839081e+00  3.497397e+00    5.654360e+00  2.429119e+00  3.052685e+0
0   

         Vmail_Plan     Day_Calls    Day_Charge     Eve_Calls    Eve_Charge
\
count  3.333000e+03  3.333000e+03  3.333000e+03  3.333000e+03  3.333000e+03   
mean  -6.608708e-17 -1.934646e-16 -2.835349e-16  3.288365e-16  1.385697e-16   
std    1.000150e+00  1.000150e+00  1.000150e+00  1.000150e+00  1.000150e+00   
min   -6.183963e-01 -5.005247e+00 -3.301162e+00 -5.025911e+00 -3.963679e+00   
25%   -6.183963e-01 -6.695701e-01 -6.623760e-01 -6.583610e-01 -6.783123e-01   
50%   -6.183963e-01  2.812491e-02 -6.730063e-03 -5.738630e-03  8.459274e-03   
75%    1.617086e+00  6.759846e-01  6.726790e-01  6.970854e-01  6.766695e-01   
max    1.617086e+00  3.217588e+00  3.140803e+00  3.508382e+00  3.207980e+00   

        Night_Calls  Night_Charge    Intl_Calls   Intl_Charge     Area_Code  
count  3.333000e+03  3.333000e+03  3.333000e+03  3.333000e+03  3.333000e+03  
mean  -4.903235e-17 -4.370275e-17 -8.527366e-18  2.728757e-16  4.221046e-16  
std    1.000150e+00  1.000150e+00  1.000150e+00  1.000150e+00  1.000150e+00  
min   -3.429870e+00 -3.515366e+00 -1.820289e+00 -3.668210e+00 -6.888343e-01  
25%   -6.699340e-01 -6.676792e-01 -6.011951e-01 -6.164341e-01 -6.888343e-01  
50%   -5.505089e-03  4.691242e-03 -1.948306e-01  2.045823e-02 -5.236033e-01  
75%    6.589239e-01  6.814562e-01  6.178983e-01  6.706192e-01  1.718817e+00  
max    3.827739e+00  3.836763e+00  6.307001e+00  3.496829e+00  1.718817e+00  
```

In [29]:

```python
# Select only the "Intl_Calls" and "Night_Mins" columns
selected_cols = ['Intl_Calls', 'Night_Mins']
telco_selected_scaled_df = telco_scaled_df[selected_cols]
telco_selected_scaled_df.describe()
```

Out[29]:

	Intl_Calls	Night_Mins
count	3.333000e+03	3.333000e+03
mean	-8.527366e-18	7.887813e-17
std	1.000150e+00	1.000150e+00
min	-1.820289e+00	-3.513648e+00
25%	-6.011951e-01	-6.698545e-01
50%	-1.948306e-01	6.485803e-03
75%	6.178983e-01	6.808485e-01
max	6.307001e+00	3.839081e+00

Feature Selection

In [30]:

```
# Drop the unnecessary features
telco = telco.drop(telco[['Area_Code','Phone']], axis=1)

# Verify dropped features
print(telco.columns)
```

```
Index(['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins', 'Night_Min
s',
       'Intl_Mins', 'CustServ_Calls', 'Churn', 'Intl_Plan', 'Vmail_Plan',
       'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge', 'Night_Calls',
       'Night_Charge', 'Intl_Calls', 'Intl_Charge', 'State'],
      dtype='object')
```

In [31]:

```
telco = telco.drop(telco[['State']], axis=1)
```

In [32]:

```
# Select only numeric columns for correlation computation
numeric_telco = telco.select_dtypes(include=['float64', 'int64'])
numeric_telco.corr()  # Identify correlated features
```

Out[32]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins
Account_Length	1.000000	-0.004628	0.006216	-0.006757	-0.008955
Vmail_Message	-0.004628	1.000000	0.000778	0.017562	0.007681
Day_Mins	0.006216	0.000778	1.000000	0.007043	0.004323
Eve_Mins	-0.006757	0.017562	0.007043	1.000000	-0.012584
Night_Mins	-0.008955	0.007681	0.004323	-0.012584	1.000000
Intl_Mins	0.009514	0.002856	-0.010155	-0.011035	-0.015207
CustServ_Calls	-0.003796	-0.013263	-0.013423	-0.012985	-0.009288
Churn	0.016541	-0.089728	0.205151	0.092796	0.035493
Intl_Plan	0.024735	0.008745	0.049396	0.019100	-0.028905
Vmail_Plan	0.002918	0.956927	-0.001684	0.021545	0.006079
Day_Calls	0.038470	-0.009548	0.006750	-0.021451	0.022938
Day_Charge	0.006214	0.000776	1.000000	0.007050	0.004324
Eve_Calls	0.019260	-0.005864	0.015769	-0.011430	-0.002093
Eve_Charge	-0.006745	0.017578	0.007029	1.000000	-0.012592
Night_Calls	-0.013176	0.007123	0.022972	0.007586	0.011204
Night_Charge	-0.008960	0.007663	0.004300	-0.012593	0.999999
Intl_Calls	0.020661	0.013957	0.008033	0.002541	-0.012353
Intl_Charge	0.009546	0.002884	-0.010092	-0.011067	-0.015180

In [33]:

```
# Compute the correlation matrix
corr_matrix = numeric_telco.corr().abs()

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

Out[33]:

	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins
Account_Length	NaN	0.004628	0.006216	0.006757	0.008955
Vmail_Message	NaN	NaN	0.000778	0.017562	0.007681
Day_Mins	NaN	NaN	NaN	0.007043	0.004323
Eve_Mins	NaN	NaN	NaN	NaN	0.012584
Night_Mins	NaN	NaN	NaN	NaN	NaN
Intl_Mins	NaN	NaN	NaN	NaN	NaN
CustServ_Calls	NaN	NaN	NaN	NaN	NaN
Churn	NaN	NaN	NaN	NaN	NaN
Intl_Plan	NaN	NaN	NaN	NaN	NaN
Vmail_Plan	NaN	NaN	NaN	NaN	NaN
Day_Calls	NaN	NaN	NaN	NaN	NaN
Day_Charge	NaN	NaN	NaN	NaN	NaN
Eve_Calls	NaN	NaN	NaN	NaN	NaN
Eve_Charge	NaN	NaN	NaN	NaN	NaN
Night_Calls	NaN	NaN	NaN	NaN	NaN
Night_Charge	NaN	NaN	NaN	NaN	NaN
Intl_Calls	NaN	NaN	NaN	NaN	NaN
Intl_Charge	NaN	NaN	NaN	NaN	NaN

In [34]:

```
# Find features with correlation greater than a specified threshold (e.g., 0
threshold = 0.8
to_drop = [column for column in upper.columns if any(upper[column] > thresho
to_drop
```

Out[34]: ['Vmail_Plan', 'Day_Charge', 'Eve_Charge', 'Night_Charge', 'Intl_Charge']

In [35]:

```
# # Drop the highly correlated features
# numeric_telco_filtered = numeric_telco.drop(columns=to_drop)

# # Display the filtered dataframe
# numeric_telco_filtered.head()
```

In [36]:

```
# Create the new feature
telco['Avg_Night_Calls'] = telco['Night_Mins'] / telco['Night_Calls']

# Print the first five rows of 'Avg_Night_Calls'
print(telco['Avg_Night_Calls'].head())
```

```
0    2.689011
1    2.469903
2    1.563462
3    2.212360
4    1.544628
Name: Avg_Night_Calls, dtype: float64
```

Churn Prediction

Model Selection

- Logistic Regression: Good baseline, interpretable.
- Random Forests: Captures complex relationships.
- Support Vector Machines (SVMs): Effective with high-dimensional data.

In [37]:
```
# Define the features
features = ['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins', 'Nigh

# Define a new customer for prediction
new_customer = pd.DataFrame({
        'Account_Length': [100],
        'Vmail_Message': [20],
        'Day_Mins': [200],
        'Eve_Mins': [150],
        'Night_Mins': [180],
        'Intl_Mins': [10],
        'CustServ_Calls': [2],
        'Day_Calls': [100],
        'Eve_Calls': [100],
        'Night_Calls': [100],
        'Intl_Calls': [5],
        # 'Area_Code': [415]
})
```

In [38]:
```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate the classifier
clf = LogisticRegression()

# Fit the classifier
clf.fit(telco[features], telco['Churn'])

# Predict the label of new_customer
print(clf.predict(new_customer))
```

```
[0]
```

In [39]:
```
# Import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

# Instantiate the classifier
clf = DecisionTreeClassifier()
```

```python
# Fit the classifier
clf.fit(telco[features], telco['Churn'])

# Predict the label of new_customer
print(clf.predict(new_customer))
```

```
[0]
```

Model Evaluation

In [40]:
```python
# Import train_test_split
from sklearn.model_selection import train_test_split

# Create feature variable
X = telco.drop('Churn', axis=1)

# Create target variable
y = telco['Churn']

# Create training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [41]:
```python
print(X_train.shape)
print(X_test.shape)
print(len(X_test))
```

```
(2333, 18)
(1000, 18)
1000
```

In [42]:
```python
from sklearn.preprocessing import LabelEncoder

# # Encode 'Intl_Plan' column
# le = LabelEncoder()
# X_train['Intl_Plan'] = le.fit_transform(X_train['Intl_Plan'])
# X_test['Intl_Plan'] = le.transform(X_test['Intl_Plan'])

# Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)

# Compute accuracy
print(clf.score(X_test, y_test))
```

```
0.954
```

In [43]:
```python
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

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

```
# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[848   7]
 [ 39 106]]
```

In [44]:

```
# Create feature variable
X = telco.drop('Churn', axis=1)

# Create target variable
y = telco['Churn']
```

In [45]:

```
# Import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Create training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Print confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[572   5]
 [ 21  69]]
```

In [46]:

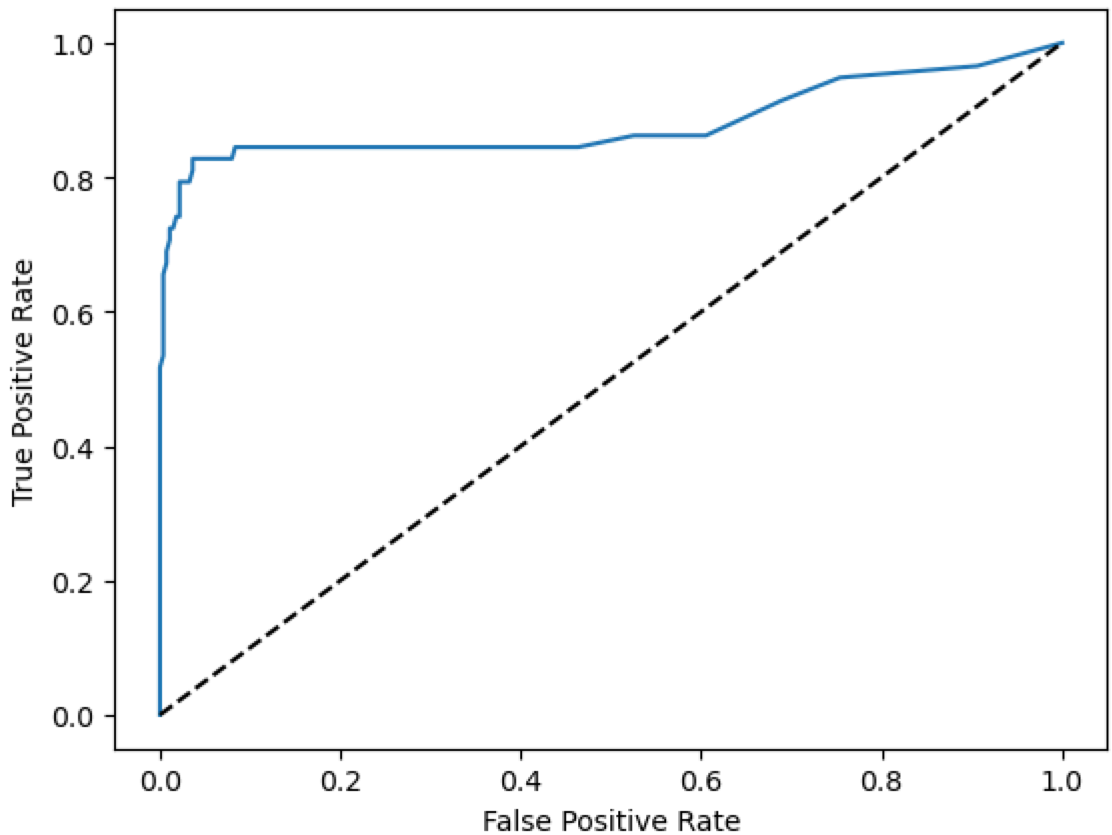
```
# Import train_test_split
from sklearn.model_selection import train_test_split

# Create training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

# Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)
```

```python
# Predict the labels of the test set
y_pred = clf.predict(X_test)
```

In [47]:
```python
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

# Print the precision
print(precision_score(y_test, y_pred))

# Print the recall
print(recall_score(y_test, y_pred))
```

```
0.9512195121951219
0.6724137931034483
```

In [48]:
```python
# Generate the probabilities
y_pred_prob = clf.predict_proba(X_test)[:, 1]

# Import roc_curve
from sklearn.metrics import roc_curve

# Calculate the roc metrics
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Plot the ROC curve
plt.plot(fpr, tpr)

# Add labels and diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot([0, 1], [0, 1], "k--")
plt.show()
```



In [49]:
```
# Import roc_auc_score
from sklearn.metrics import roc_auc_score

# Print the AUC
print(roc_auc_score(y_test, y_pred_prob))
```

```
0.8809345327336333
```

In [50]:
```
# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Import f1_score
from sklearn.metrics import f1_score

# Print the F1 score
print(f1_score(y_test, y_pred))
```

```
0.7628865979381443
```

Model Tuning

In [51]:
```
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV
```

```python
# Create the hyperparameter grid
param_grid = {'max_features': ['auto', 'sqrt', 'log2']}

# Call GridSearchCV
grid_search = GridSearchCV(clf, param_grid, cv=3)

# Fit the model
grid_search.fit(X, y)

# Print the optimal parameters
print(grid_search.best_params_)
```

```
{'max_features': 'sqrt'}
```

In [52]:
```python
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Create the hyperparameter grid
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Call GridSearchCV
grid_search = GridSearchCV(clf, param_grid, cv=3)

# Fit the model
grid_search.fit(X, y)

# Print the best hyperparameters
print(grid_search.best_params_)
```

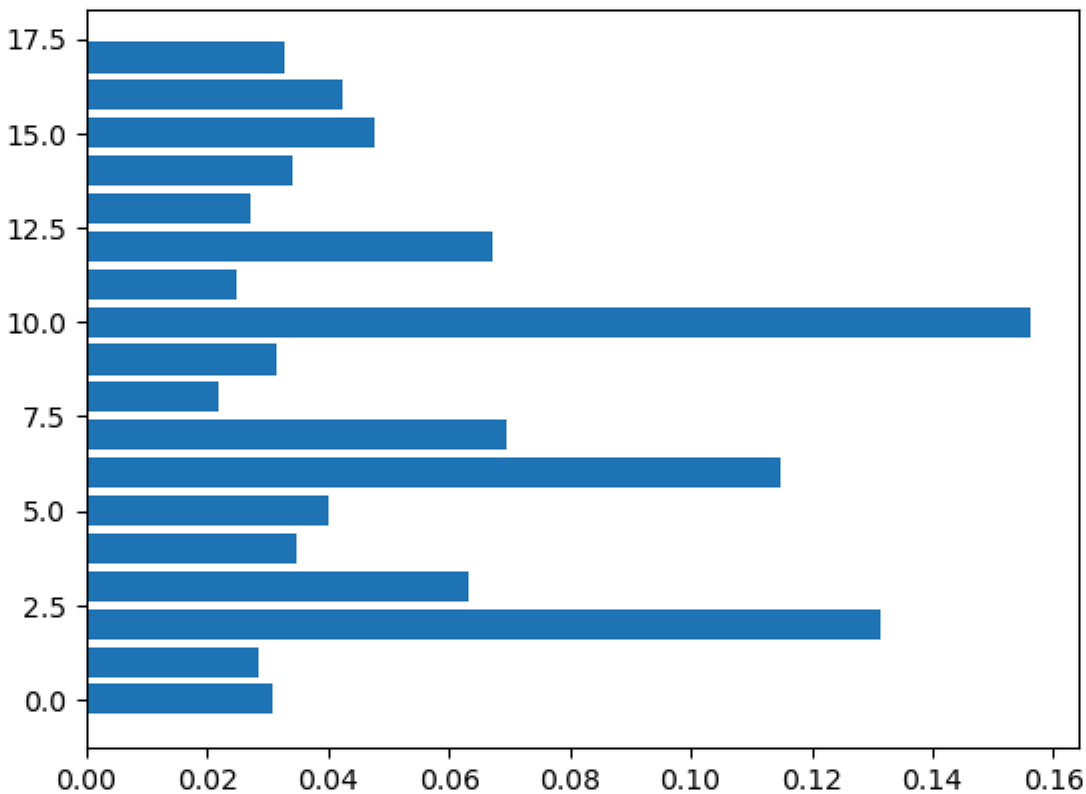
```
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_feature
s': 10}
```

In [53]:
```python
# Import RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Create the hyperparameter grid
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Call RandomizedSearchCV
random_search = RandomizedSearchCV(clf, param_dist, cv=3)

# Fit the model
random_search.fit(X, y)

# Print best parameters
print(random_search.best_params_)
```

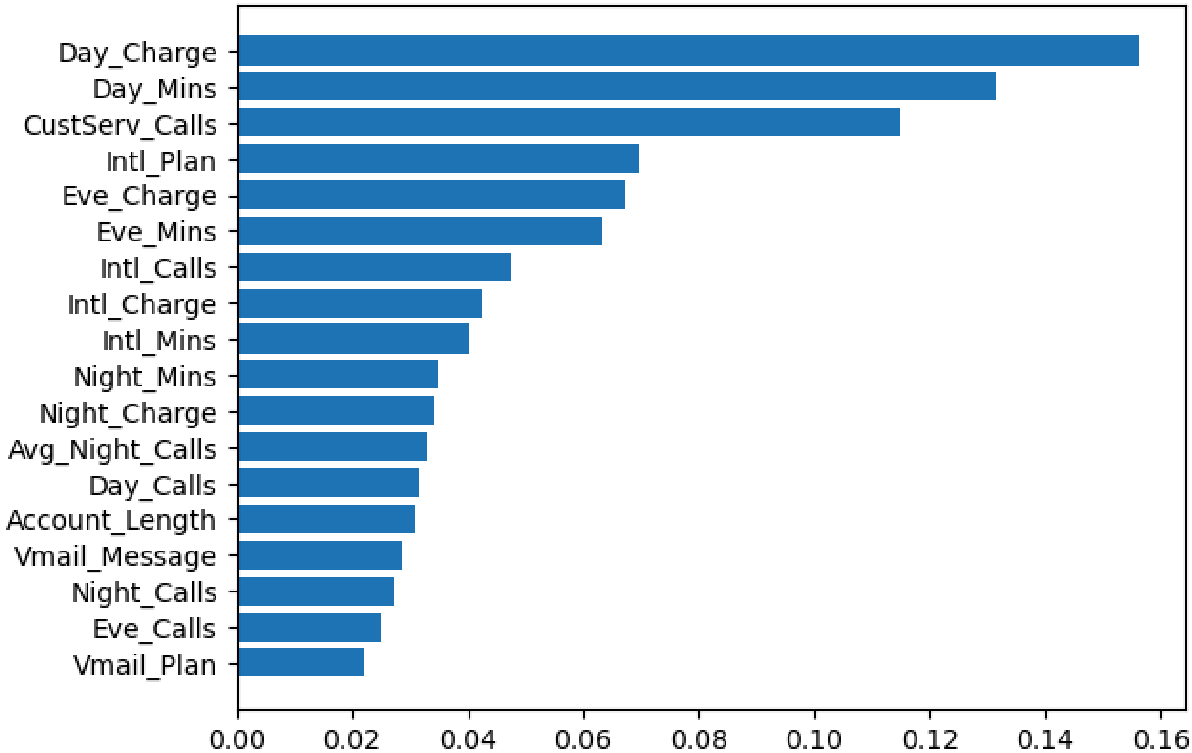
```
{'bootstrap': False, 'criterion': 'entropy', 'max_depth': None, 'max_feature
s': 6}
```

Feature Importance

In [54]:
```
# Calculate feature importances
importances = clf.feature_importances_

# Create plot
plt.barh(range(X.shape[1]), importances)
plt.show()
```



In [55]:
```
# Sort importances
sorted_index = np.argsort(importances)

# Create labels
labels = X.columns[sorted_index]

# Clear current plot
plt.clf()

# Create plot
plt.barh(range(X.shape[1]), importances[sorted_index], tick_label=labels)
plt.show()
```



In [56]:
```
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Create training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the data
clf.fit(X_train, y_train)

# Print the accuracy
print(clf.score(X_test, y_test))
```

```
0.949
```

In [57]:
```
# Import f1_score
from sklearn.metrics import f1_score

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the data
clf.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Print the F1 score
print(f1_score(y_test, y_pred))
```

0.8106060606060606