Customer Churn Prediction for SyriaTel: A Machine Learning Approach to Enhance Retention Strategies.

Business understanding

Customer churn is a major challenge for companies in the telecommunications industry, including SyriaTel. The objective is to develop a predictive model that can determine whether a customer is likely to terminate their services. This binary classification task focuses on identifying patterns in customer behavior and demographic data that may signal a risk of churn. The ultimate goal is to help SyriaTel mitigate the financial impact of churn by enabling proactive retention strategies.

Problem statement

SyriaTel is confronted with the challenge of retaining customers in a highly competitive telecommunications market. Customer churn results in revenue loss and negatively impacts the company's reputation and market position. The goal is to create a predictive model that can accurately identify customers at risk of churning, allowing SyriaTel to take proactive steps with targeted retention strategies.

Objectives

The goal is to develop the most effective model for predicting customer churn at SyriaTel, with the aim of minimizing the financial impact of churn through the implementation of targeted retention strategies.

Data source

The data is sourced from https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset)

Data loading

This step is to help us understand our data for more detailed modeling and visualizations.

In [205]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
df = pd.read_csv('SyriaTel Customer Churn.csv')
df.head()
```

Out[205]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota ev call
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	9
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	10
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	11
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	8
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	12

5 rows × 21 columns

In [206]:
```
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
```

In [207]:
```
categorical_columns = df.select_dtypes(exclude='number')
categorical_columns.columns
```

Out[207]:
```
Index(['state', 'phone number', 'international plan', 'voice mail plan',
       'churn'],
      dtype='object')
```

In [208]:
```
numeric_columns = df.select_dtypes(include='number')
numeric_columns.columns
```

Out[208]:
```
Index(['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', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls'],
      dtype='object')
```

Data cleaning

Checking for missing values

In [209]:
```
# Check for missing values
missing_values = df.isnull().sum()
print( 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
```

The data has no missing values

Checking for duplicate values

In [210]:
```
# Check for duplicate rows
duplicates = df[df.duplicated()]

# Display the number of duplicate rows
print(f"Number of duplicate rows: {duplicates.shape[0]}")

# Display the duplicate rows, if any
if not duplicates.empty:
    print("duplicate rows:")
    print(duplicates)
else:
    print("No duplicate rows found.")
```

```
Number of duplicate rows: 0
No duplicate rows found.
```

The data has no duplicates

In [211]:
```
df.describe()
```

Out[211]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total ev minute
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.00000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.98034
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.71384
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.60000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.40000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.30000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.70000

Dropping unimportant data

In [238]:
```
# Drop the 'phone number' column from the dataset
data = df.drop(columns=['phone number'])

# Verify that the column has been removed
data.head()
```

Out[238]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total e minu
0	KS	0.524793	0.068627	no	yes	0.490196	0.755701	0.666667	0.755701	0.542
1	OH	0.438017	0.068627	no	yes	0.509804	0.460661	0.745455	0.460597	0.537
2	NJ	0.561983	0.068627	no	no	0.000000	0.693843	0.690909	0.693830	0.333
3	OH	0.342975	0.000000	yes	no	0.000000	0.853478	0.430303	0.853454	0.170
4	OK	0.305785	0.068627	yes	no	0.000000	0.475200	0.684848	0.475184	0.407

Identify numeric and categorical variables

In [212]:
```
# Identify numeric and categorical variables
numeric_columns = df.select_dtypes(include=['number']).columns
categorical_columns = df.select_dtypes(include=['object', 'category']).columns

# Display the results
print("Numeric Variables:")
print(numeric_columns)

print("\nCategorical Variables:")
print(categorical_columns)
```

```
Numeric Variables:
Index(['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', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls'],
      dtype='object')

Categorical Variables:
Index(['state', 'phone number', 'international plan', 'voice mail plan'], dty
pe='object')
```

In [213]:
```
df.head()
```

Out[213]:

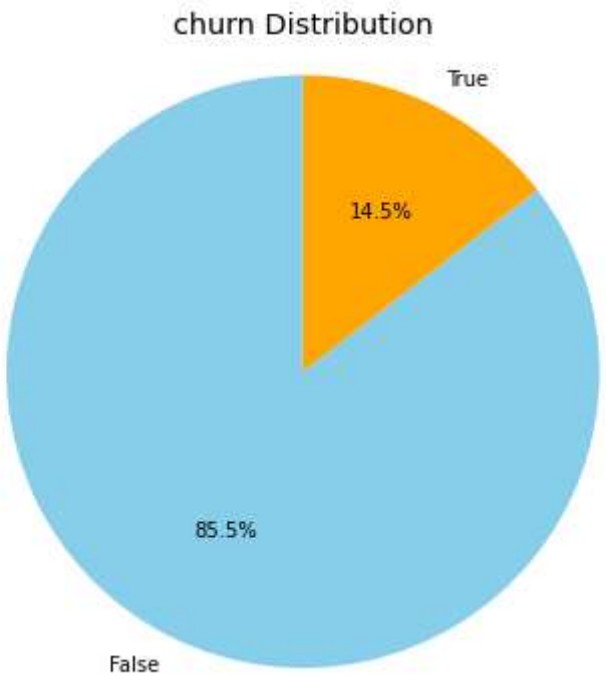
	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota ev call
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	9
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	10
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	11
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	8
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	12

5 rows × 21 columns

In [214]:
```
# Check the count of each unique value in the 'Churn' column
if 'churn' in df.columns:
    churn_counts = df['churn'].value_counts()
    print("churn counts:")
    print(churn_counts)
else:
    print("The column 'churn' is not found in the dataset. Please check the co
lumn name.")
```

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

In [ ]:
```
# Check if the 'churn' column exists
if 'churn' in df.columns:
    # Calculate the churn counts
    churn_counts = df['churn'].value_counts()

    # Create a pie chart
    plt.figure(figsize=(8, 6))
    plt.pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%', starta
ngle=90, colors=['skyblue', 'orange'])
    plt.title('churn Distribution')
    plt.axis('equal')
    plt.show()
else:
    print("The column 'churn' is not found in the dataset. Please check the co
lumn name.")
```



In [239]:
```
# Summary statistics for numerical columns
summary_statistics = data.describe()

# Display the summary statistics
summary_statistics
```

Out[239]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total ev minute
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.00000
mean	0.413491	0.286102	0.158804	0.512472	0.608701	0.512446	0.55259
std	0.164554	0.415405	0.268399	0.155266	0.121631	0.155255	0.13943
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.301653	0.000000	0.000000	0.409635	0.527273	0.409624	0.45807
50%	0.413223	0.068627	0.000000	0.511403	0.612121	0.511402	0.55375
75%	0.520661	1.000000	0.392157	0.616876	0.690909	0.616868	0.64696
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

In [240]:
```
# Convert 'international plan' and 'voice mail plan' to binary numerical value
s
data['international plan'] = data['international plan'].apply(lambda x: 1 if x
== 'yes' else 0)
data['voice mail plan'] = data['voice mail plan'].apply(lambda x: 1 if x == 'y
es' else 0)

# Encode 'state' using one-hot encoding or label encoding
# One-hot encoding
data = pd.get_dummies(data, columns=['state'], drop_first=True)

# Verify the changes
data.head()
```

Out[240]:

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	to
0	0.524793	0.068627	0	1	0.490196	0.755701	0.666667	0.755701	0.542755	0.
1	0.438017	0.068627	0	1	0.509804	0.460661	0.745455	0.460597	0.537531	0.
2	0.561983	0.068627	0	0	0.000000	0.693843	0.690909	0.693830	0.333242	0.
3	0.342975	0.000000	1	0	0.000000	0.853478	0.430303	0.853454	0.170195	0.
4	0.305785	0.068627	1	0	0.000000	0.475200	0.684848	0.475184	0.407754	0.

5 rows × 69 columns

In [216]:
```
df.corr
```

Out[216]:
```
<bound method DataFrame.corr of      state  account length  area code phone n
umber international plan  \
0       KS             128        415     382-4657                 no
1       OH             107        415     371-7191                 no
2       NJ             137        415     358-1921                 no
3       OH              84        408     375-9999                yes
4       OK              75        415     330-6626                yes
...    ...             ...        ...          ...                ...
3328    AZ             192        415     414-4276                 no
3329    WV              68        415     370-3271                 no
3330    RI              28        510     328-8230                 no
3331    CT             184        510     364-6381                yes
3332    TN              74        415     400-4344                 no

     voice mail plan  number vmail messages  total day minutes  \
0                yes                     25              265.1
1                yes                     26              161.6
2                 no                      0              243.4
3                 no                      0              299.4
4                 no                      0              166.7
...              ...                    ...                ...
3328             yes                     36              156.2
3329              no                      0              231.1
3330              no                      0              180.8
3331              no                      0              213.8
3332             yes                     25              234.4

      total day calls  total day charge  ...  total eve calls  \
0                 110             45.07  ...               99
1                 123             27.47  ...              103
2                 114             41.38  ...              110
3                  71             50.90  ...               88
4                 113             28.34  ...              122
...               ...               ...  ...              ...
3328               77             26.55  ...              126
3329               57             39.29  ...               55
3330              109             30.74  ...               58
3331              105             36.35  ...               84
3332              113             39.85  ...               82

      total eve charge  total night minutes  total night calls  \
0                16.78                244.7                 91
1                16.62                254.4                103
2                10.30                162.6                104
3                 5.26                196.9                 89
4                12.61                186.9                121
...                ...                  ...                ...
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
0                  11.01                10.0                 3
1                  11.45                13.7                 3
2                   7.32                12.2                 5
```

3	8.86	6.6	7
4	8.41	10.1	3
...	...	...	...
3328	12.56	9.9	6
3329	8.61	9.6	4
3330	8.64	14.1	6
3331	6.26	5.0	10
3332	10.86	13.7	4

	total intl charge	customer service calls	churn
0	2.70	1	False
1	3.70	1	False
2	3.29	0	False
3	1.78	2	False
4	2.73	3	False
...	...	...	...
3328	2.67	2	False
3329	2.59	3	False
3330	3.81	2	False
3331	1.35	2	False
3332	3.70	0	False

[3333 rows x 21 columns]>

In [217]:
```
# Compute the correlation matrix for numeric variables
correlation_matrix = df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

Correlation Matrix:

	account length	area code	number vmail messages	\
account length	1.000000	-0.012463	-0.004628	
area code	-0.012463	1.000000	-0.001994	
number vmail messages	-0.004628	-0.001994	1.000000	
total day minutes	0.006216	-0.008264	0.000778	
total day calls	0.038470	-0.009646	-0.009548	
total day charge	0.006214	-0.008264	0.000776	
total eve minutes	-0.006757	0.003580	0.017562	
total eve calls	0.019260	-0.011886	-0.005864	
total eve charge	-0.006745	0.003607	0.017578	
total night minutes	-0.008955	-0.005825	0.007681	
total night calls	-0.013176	0.016522	0.007123	
total night charge	-0.008960	-0.005845	0.007663	
total intl minutes	0.009514	-0.018288	0.002856	
total intl calls	0.020661	-0.024179	0.013957	
total intl charge	0.009546	-0.018395	0.002884	
customer service calls	-0.003796	0.027572	-0.013263	
churn	0.016541	0.006174	-0.089728	

	total day minutes	total day calls	total day charge	\
account length	0.006216	0.038470	0.006214	
area code	-0.008264	-0.009646	-0.008264	
number vmail messages	0.000778	-0.009548	0.000776	
total day minutes	1.000000	0.006750	1.000000	
total day calls	0.006750	1.000000	0.006753	
total day charge	1.000000	0.006753	1.000000	
total eve minutes	0.007043	-0.021451	0.007050	
total eve calls	0.015769	0.006462	0.015769	
total eve charge	0.007029	-0.021449	0.007036	
total night minutes	0.004323	0.022938	0.004324	
total night calls	0.022972	-0.019557	0.022972	
total night charge	0.004300	0.022927	0.004301	
total intl minutes	-0.010155	0.021565	-0.010157	
total intl calls	0.008033	0.004574	0.008032	
total intl charge	-0.010092	0.021666	-0.010094	
customer service calls	-0.013423	-0.018942	-0.013427	
churn	0.205151	0.018459	0.205151	

	total eve minutes	total eve calls	total eve charge	\
account length	-0.006757	0.019260	-0.006745	
area code	0.003580	-0.011886	0.003607	
number vmail messages	0.017562	-0.005864	0.017578	
total day minutes	0.007043	0.015769	0.007029	
total day calls	-0.021451	0.006462	-0.021449	
total day charge	0.007050	0.015769	0.007036	
total eve minutes	1.000000	-0.011430	1.000000	
total eve calls	-0.011430	1.000000	-0.011423	
total eve charge	1.000000	-0.011423	1.000000	
total night minutes	-0.012584	-0.002093	-0.012592	
total night calls	0.007586	0.007710	0.007596	
total night charge	-0.012593	-0.002056	-0.012601	
total intl minutes	-0.011035	0.008703	-0.011043	
total intl calls	0.002541	0.017434	0.002541	
total intl charge	-0.011067	0.008674	-0.011074	

customer service calls	-0.012985	0.002423	-0.012987
churn	0.092796	0.009233	0.092786

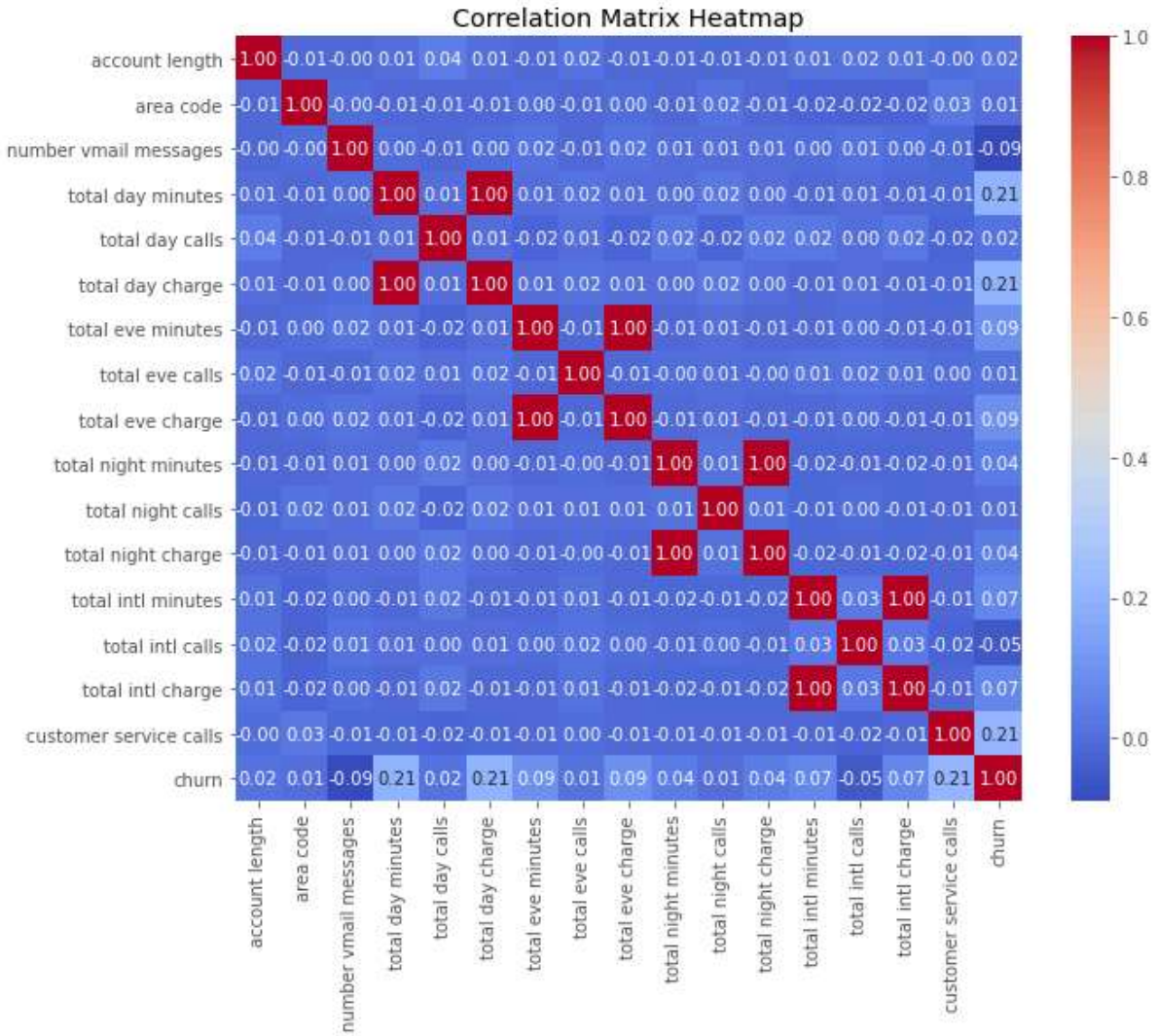
	total night minutes	total night calls	\
account length	-0.008955	-0.013176	
area code	-0.005825	0.016522	
number vmail messages	0.007681	0.007123	
total day minutes	0.004323	0.022972	
total day calls	0.022938	-0.019557	
total day charge	0.004324	0.022972	
total eve minutes	-0.012584	0.007586	
total eve calls	-0.002093	0.007710	
total eve charge	-0.012592	0.007596	
total night minutes	1.000000	0.011204	
total night calls	0.011204	1.000000	
total night charge	0.999999	0.011188	
total intl minutes	-0.015207	-0.013605	
total intl calls	-0.012353	0.000305	
total intl charge	-0.015180	-0.013630	
customer service calls	-0.009288	-0.012802	
churn	0.035493	0.006141	

	total night charge	total intl minutes	\
account length	-0.008960	0.009514	
area code	-0.005845	-0.018288	
number vmail messages	0.007663	0.002856	
total day minutes	0.004300	-0.010155	
total day calls	0.022927	0.021565	
total day charge	0.004301	-0.010157	
total eve minutes	-0.012593	-0.011035	
total eve calls	-0.002056	0.008703	
total eve charge	-0.012601	-0.011043	
total night minutes	0.999999	-0.015207	
total night calls	0.011188	-0.013605	
total night charge	1.000000	-0.015214	
total intl minutes	-0.015214	1.000000	
total intl calls	-0.012329	0.032304	
total intl charge	-0.015186	0.999993	
customer service calls	-0.009277	-0.009640	
churn	0.035496	0.068239	

	total intl calls	total intl charge	\
account length	0.020661	0.009546	
area code	-0.024179	-0.018395	
number vmail messages	0.013957	0.002884	
total day minutes	0.008033	-0.010092	
total day calls	0.004574	0.021666	
total day charge	0.008032	-0.010094	
total eve minutes	0.002541	-0.011067	
total eve calls	0.017434	0.008674	
total eve charge	0.002541	-0.011074	
total night minutes	-0.012353	-0.015180	
total night calls	0.000305	-0.013630	
total night charge	-0.012329	-0.015186	
total intl minutes	0.032304	0.999993	
total intl calls	1.000000	0.032372	
total intl charge	0.032372	1.000000	

```
customer service calls         -0.017561          -0.009675
churn                          -0.052844           0.068259

                        customer service calls     churn
account length                       -0.003796  0.016541
area code                             0.027572  0.006174
number vmail messages                -0.013263 -0.089728
total day minutes                    -0.013423  0.205151
total day calls                      -0.018942  0.018459
total day charge                     -0.013427  0.205151
total eve minutes                    -0.012985  0.092796
total eve calls                       0.002423  0.009233
total eve charge                     -0.012987  0.092786
total night minutes                  -0.009288  0.035493
total night calls                    -0.012802  0.006141
total night charge                   -0.009277  0.035496
total intl minutes                   -0.009640  0.068239
total intl calls                     -0.017561 -0.052844
total intl charge                    -0.009675  0.068259
customer service calls                1.000000  0.208750
churn                                 0.208750  1.000000
```

In [218]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



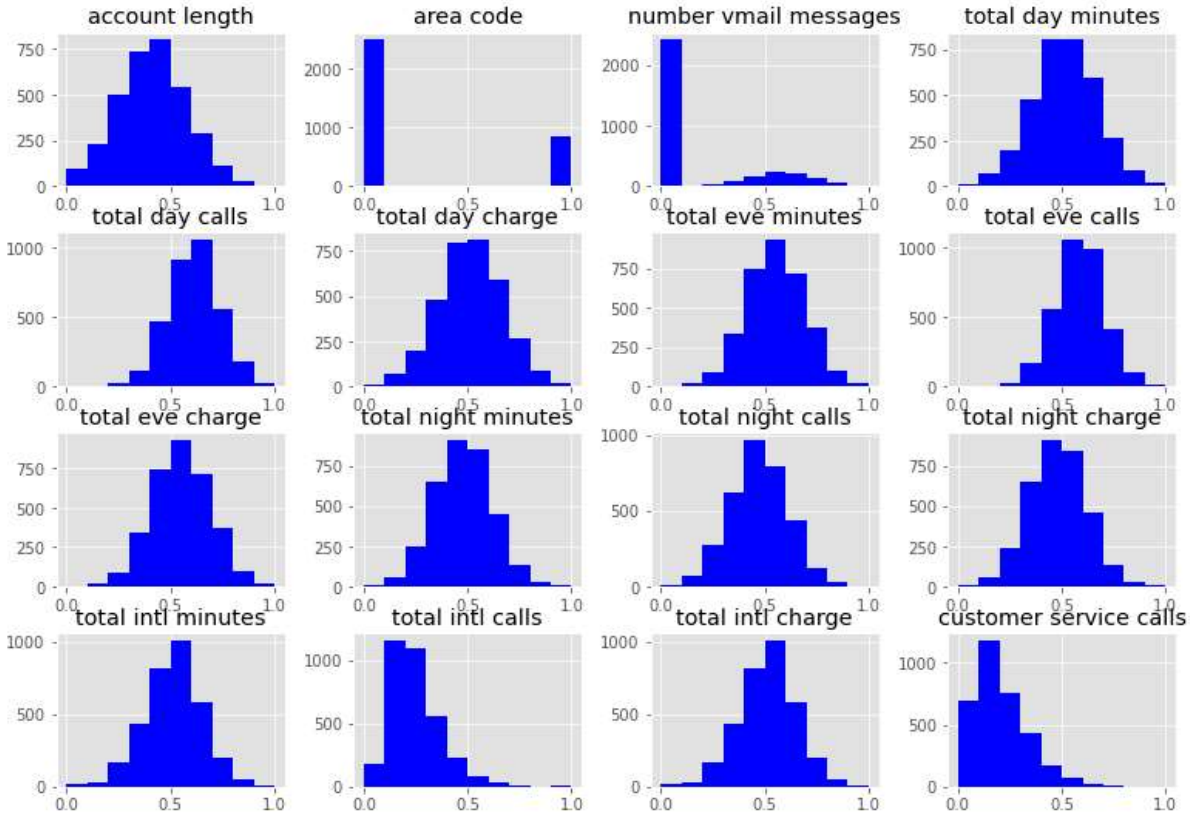
In [284]:
```
# drop the columns with high correlation
cols_drop = ['total day charge', 'total eve charge', 'total night charge', 'to
tal intl charge']
df = df.drop(cols_drop, axis=1)
df.head(5)
```

Out[284]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total ev minut
0	KS	0.524793	0.068627	382-4657	no	yes	0.490196	0.755701	0.666667	0.5427
1	OH	0.438017	0.068627	371-7191	no	yes	0.509804	0.460661	0.745455	0.5375
2	NJ	0.561983	0.068627	358-1921	no	no	0.000000	0.693843	0.690909	0.3332
3	OH	0.342975	0.000000	375-9999	yes	no	0.000000	0.853478	0.430303	0.1701
4	OK	0.305785	0.068627	330-6626	yes	no	0.000000	0.475200	0.684848	0.4077

In [219]:
```
#Converting churn to binary
df['churn'] = pd.DataFrame(df['churn'].map({False: 0, True: 1}))
```

In [283]:
```
# Distribution of features excluding 'churn' column

df.drop(columns='churn').hist(figsize=(13,9), color='blue')
plt.show()
```



In [221]:
```
df.select_dtypes(include=['object']).columns
```

Out[221]:
```
Index(['state', 'phone number', 'international plan', 'voice mail plan'], dty
pe='object')
```

In [223]:
```
print(df.columns)
```

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

Data splitting

To start modeling we need to split our data features and target variable.

In [225]:
```
X = df.drop(columns=['churn'])
y = df['churn']
```

In [226]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=42)

# Display the shapes of the resulting splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (2666, 20)
X_test shape: (667, 20)
y_train shape: (2666,)
y_test shape: (667,)
```

In [228]:
```
import numpy as np
from scipy.stats import zscore

# Calculate the Z-scores for each numeric column
numeric_columns = df.select_dtypes(include=np.number).columns
df[numeric_columns] = df[numeric_columns].apply(zscore)

# Set a threshold for outliers (commonly 3)
threshold = 3

# Remove rows where any Z-score is greater than the threshold or less than the negative threshold
df_cleaned = df[(df[numeric_columns].abs() <= threshold).all(axis=1)]

# Check the size of the DataFrame before and after removing outliers
print(f"Before removing outliers, data length: {len(df)}")
print(f"After removing outliers, data length: {len(df_cleaned)}")

# Optionally, inspect the first few rows of the cleaned data
df_cleaned.head()
```

```
Before removing outliers, data length: 3333
After removing outliers, data length: 3169
```

Out[228]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tota cl
0	KS	0.676489	-0.523603	382-4657	no	yes	1.234883	1.566767	0.476643	1.56
1	OH	0.149065	-0.523603	371-7191	no	yes	1.307948	-0.333738	1.124503	-0.33
2	NJ	0.902529	-0.523603	358-1921	no	no	-0.591760	1.168304	0.675985	1.16
3	OH	-0.428590	-0.688834	375-9999	yes	no	-0.591760	2.196596	-1.466936	2.19
4	OK	-0.654629	-0.523603	330-6626	yes	no	-0.591760	-0.240090	0.626149	-0.24

5 rows × 21 columns

In [229]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_columns = df.select_dtypes(include=[np.number]).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
df.head()
```

Out[229]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total da charg
0	KS	0.524793	0.068627	382-4657	no	yes	0.490196	0.755701	0.666667	0.7557(
1	OH	0.438017	0.068627	371-7191	no	yes	0.509804	0.460661	0.745455	0.4605(
2	NJ	0.561983	0.068627	358-1921	no	no	0.000000	0.693843	0.690909	0.6938:
3	OH	0.342975	0.000000	375-9999	yes	no	0.000000	0.853478	0.430303	0.8534(
4	OK	0.305785	0.068627	330-6626	yes	no	0.000000	0.475200	0.684848	0.4751(

5 rows × 21 columns

In [230]:
```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = X_train.quantile(0.25)
Q3 = X_train.quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify rows that have outliers in any feature
outliers = ((X_train < lower_bound) | (X_train > upper_bound)).any(axis=1)

# Remove rows with outliers
X_train_no_outliers = X_train[~outliers]

# Check the shape before and after removing outliers
print(f"Original dataset shape: {X_train.shape}")
print(f"Dataset shape after removing outliers: {X_train_no_outliers.shape}")
```

```
Original dataset shape: (2666, 20)
Dataset shape after removing outliers: (1673, 20)
```

In [232]:
```
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   float64
 2   area code               3333 non-null   float64
 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   float64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   float64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   float64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   float64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   float64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   float64
 20  churn                   3333 non-null   float64
dtypes: float64(17), object(4)
memory usage: 546.9+ KB
```

Data scaling

In [264]:
```
from sklearn.preprocessing import StandardScaler
# Create StandardScaler object
scaler = StandardScaler()

# Fit and transform the features
X_scaled = scaler.fit_transform(X)

# Convert the scaled features back to a DataFrame
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

X_scaled.head()
```

Out[264]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total ev charg
0	0.676489	-0.523603	1.234883	1.566767	0.476643	1.567036	-0.070610	-0.055940	-0.07042
1	0.149065	-0.523603	1.307948	-0.333738	1.124503	-0.334013	-0.108080	0.144867	-0.10754
2	0.902529	-0.523603	-0.591760	1.168304	0.675985	1.168464	-1.573383	0.496279	-1.57390
3	-0.428590	-0.688834	-0.591760	2.196596	-1.466936	2.196759	-2.742865	-0.608159	-2.74326
4	-0.654629	-0.523603	-0.591760	-0.240090	0.626149	-0.240041	-1.038932	1.098699	-1.03793

5 rows × 72 columns

In [287]:
```
#initialize SMOTE object

smote = SMOTE(random_state=42)

#Apply SMOTE to training Data

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Initialize the logistic regression model

model = LogisticRegression()

#Train  model on the resampled data

model.fit(X_train_smote, y_train_smote)
```

```
c:\Users\pc\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_
logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  n_iter_i = _check_optimize_result(
```

Out[287]:
```
LogisticRegression()
```

In [265]:
```
from sklearn.preprocessing import StandardScaler

# Scale the training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_smote)

# Train the model on the scaled data
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train_smote)
```

Out[265]:
```
LogisticRegression(max_iter=1000)
```

Modelling

1.Logistic regression

In [233]:
```
X_test.select_dtypes(include=['object']).columns
```

Out[233]:
```
Index(['state', 'phone number', 'international plan', 'voice mail plan'], dty
pe='object')
```

In [234]:
```
X_train = pd.get_dummies(X_train, drop_first=True)
```

In [267]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_m
atrix

# Initialize the Logistic Regression model
log_model = LogisticRegression(max_iter=10000, random_state=42)

# Train the model
log_model.fit(X_train, y_train)
print(log_model)
```

```
LogisticRegression(max_iter=10000, random_state=42)
```

In [268]:
```
#initialize SMOTE object
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
smote = SMOTE(random_state=42)

#Apply SMOTE to training Data

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Initialize the logistic regression model

model = LogisticRegression()

#Train  model on the resampled data

model.fit(X_train_smote, y_train_smote)
```

```
c:\Users\pc\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_
logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  n_iter_i = _check_optimize_result(
```

Out[268]:
```
LogisticRegression()
```

In [246]:
```
# Import the necessary classifiers
from sklearn.linear_model import LogisticRegression
# Create a logistic regression model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')

# Fit the model on the resampled training data
logreg.fit(X_train, y_train)
```

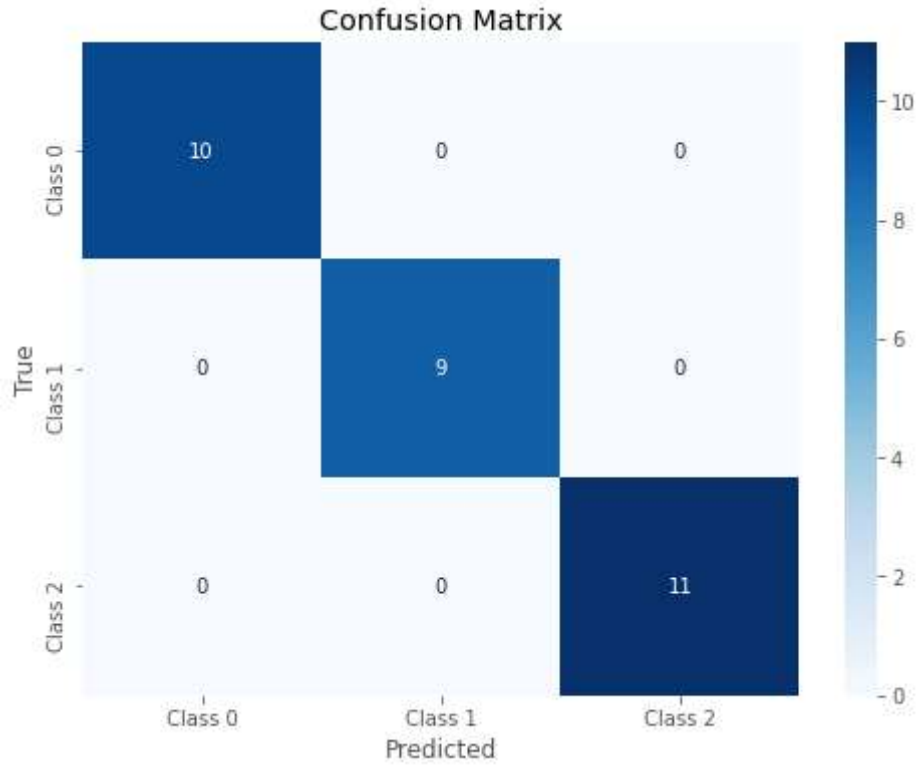
Out[246]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinea
r')
```

In [259]:
```
# Generate predictions on the test set
y_pred = logreg.predict(X_test)
```

In [289]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score, roc_auc_score

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Precision (Adjust for multiclass classification)
precision = precision_score(y_test, y_pred, average='weighted')  # Or 'micro',
'weighted'

# Recall (Adjust for multiclass classification)
recall = recall_score(y_test, y_pred, average='weighted')  # Or 'micro', 'weig
hted'

# F1-score (Adjust for multiclass classification)
f1 = f1_score(y_test, y_pred, average='weighted')  # Or 'micro', 'weighted'

# AUC for Logistic Regression (requires predicted probabilities)
y_pred_prob = model.predict_proba(X_test)  # Get the predicted probabilities
auc_lr = roc_auc_score(y_test, y_pred_prob, multi_class='ovr')  # Use 'ovr' fo
r multiclass

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("AUC (Logistic Regression):", auc_lr)
```

```
Accuracy: 0.9666666666666667
Precision: 0.9700000000000001
Recall: 0.9666666666666667
F1-score: 0.966750208855472
AUC (Logistic Regression): 1.0
```

In [249]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(y_test, predictions)

# Plotting the confusion matrix using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Cl
ass 1', 'Class 2'], yticklabels=['Class 0', 'Class 1', 'Class 2'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



2. Decision tree

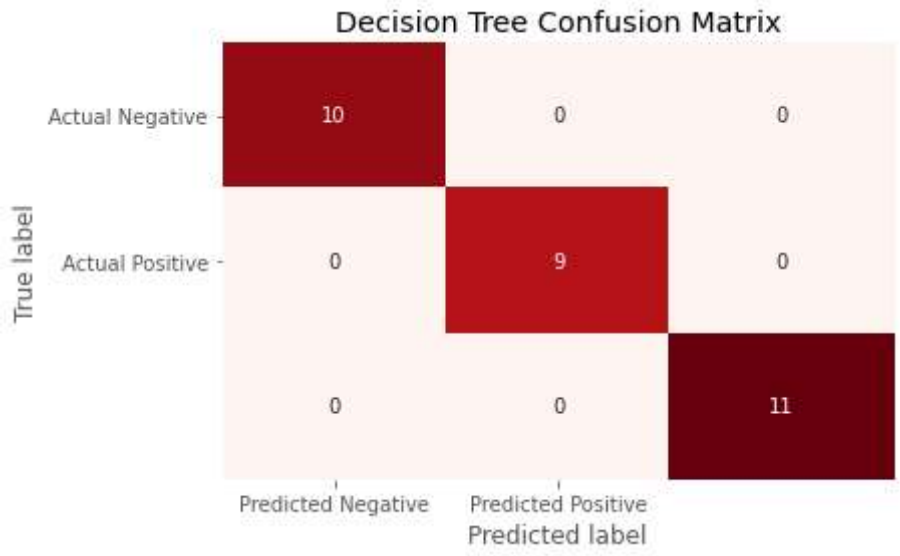
In [276]:
```
# Create an instance of DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
dt = DecisionTreeClassifier(random_state=42)

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

Out[276]:
```
DecisionTreeClassifier(random_state=42)
```

In [290]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score

# Generate predictions on the test set
precision_dt = precision_score(y_test, y_pred, average='weighted')
recall_dt = recall_score(y_test, y_pred, average='weighted')
accuracy_dt = accuracy_score(y_test, y_pred)

print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("Accuracy:", accuracy_dt)
```

```
Precision: 0.9700000000000001
Recall: 0.9666666666666667
Accuracy: 0.9666666666666667
```

In [278]:
```
# Generate predictions on the test set
y_pred_dt = dt.predict(X_test)

# Build confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# Visualize the Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_dt, annot=True, cmap='Reds', fmt='g', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```



3.Random forest model

In [280]:
```
#Initializing the Random Forest model

rf_model = RandomForestClassifier(random_state=42)
#Train the Random Forest model on the training data

rf_model.fit(X_train, y_train)
```

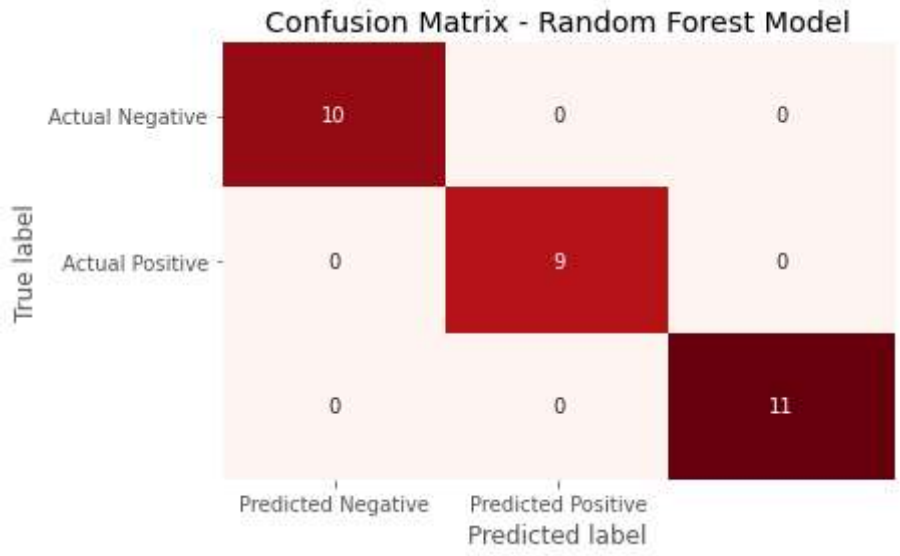
Out[280]:
```
RandomForestClassifier(random_state=42)
```

In [281]:
```
#Generate predictions on the test data

y_pred_rf = rf_model.predict(X_test)
```

In [291]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score, roc_auc_score

# Accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Precision (Adjust for multiclass classification)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')  # Or 'm
icro', 'weighted'

# Recall (Adjust for multiclass classification)
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')  # Or 'micro',
'weighted'

# F1-score (Adjust for multiclass classification)
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')  # Or 'micro', 'weight
ed'

# ROC AUC Score for Random Forest (requires predicted probabilities)
y_pred_prob_rf = rf_model.predict_proba(X_test)  # Get predicted probabilities
roc_auc_rf = roc_auc_score(y_test, y_pred_prob_rf, multi_class='ovr')  # 'ovr'
for multiclass

# Print the performance metrics
print("Random Forest Metrics:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1-score:", f1_rf)
print("ROC AUC Score:", roc_auc_rf)
```

```
Random Forest Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
ROC AUC Score: 1.0
```

In [298]:
```
# Confusion matrix for Random Forest model
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Visualize the matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_rf, annot=True, cmap='Reds', fmt='g', cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix - Random Forest Model')
plt.show()
```



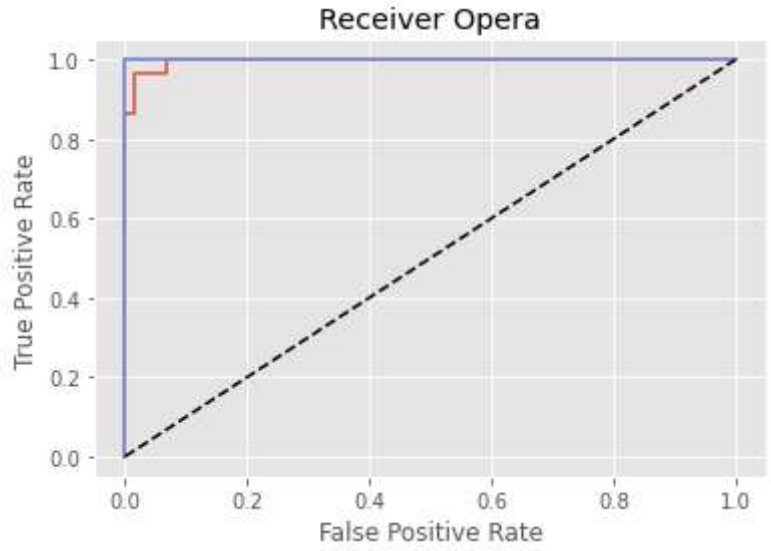
In [307]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)  # Initialize Random Forest
rf.fit(X_train, y_train)  # Train the model
```

Out[307]:
```
RandomForestClassifier(random_state=42)
```

plot a ROC curve to compare performance of the models

In [311]:
```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize

# Binarize the labels for multiclass classification
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])  # Assuming 3 classes,
update for your case

# Logistic Regression
logreg_probs = logreg.predict_proba(X_test)
logreg_fpr, logreg_tpr, _ = roc_curve(y_test_bin.ravel(), logreg_probs.ravel
())  # Flatten arrays

# Random Forest
rf_probs = rf.predict_proba(X_test)
rf_fpr, rf_tpr, _ = roc_curve(y_test_bin.ravel(), rf_probs.ravel())

# Decision Tree
dt_probs = dt.predict_proba(X_test)
dt_fpr, dt_tpr, _ = roc_curve(y_test_bin.ravel(), dt_probs.ravel())

# Plotting the ROC curves for each class
plt.plot(logreg_fpr, logreg_tpr, label='Logistic Regression')
plt.plot(rf_fpr, rf_tpr, label='Random Forest')
plt.plot(dt_fpr, dt_tpr, label='Decision Tree')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')  # Diagonal line for r
andom classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Opera')
```

Out[311]:
```
Text(0.5, 1.0, 'Receiver Opera')
```



Conclusion

The Random Forest model outperforms Logistic Regression and decision tree on all metrics, achieving perfect scores (1.0) for accuracy, precision, recall, F1-score, and ROC AUC.

Recommendations

1.Introduce loyalty programs that reward long-term customers with benefits such as free upgrades, exclusive services, or special discounts.

2.Customers with frequent customer service calls may indicate unresolved issues. Streamline complaint resolution processes to address their concerns promptly.

3.Design personalized offers such as discounts, loyalty rewards, or improved service plans to retain these customers