SYRIATEL CUSTOMER CHURN RATE ANALYSIS

Business Understanding

Introduction: The project aims to address a critical issue faced by SyriaTel, a telecommunications company, by developing a classifier to predict which customers are likely to stop using their services in the near future. This binary classification problem focuses on identifying patterns that signal customer churn.

Stakeholders: The primary stakeholders for this project are the management and customer retention teams at SyriaTel. These stakeholders can utilize the insights gained from the classifier to pinpoint at-risk customers more accurately. By understanding the predictive patterns of customer behavior, they can implement targeted interventions to improve retention rates and reduce revenue loss.

Conclusion: By uncovering and analyzing predictable patterns in customer churn, this project provides valuable information that can help SyriaTel enhance their customer retention strategies. The implications for the company include reduced financial losses associated with customer attrition and improved overall profitability through more effective customer engagement and retention efforts.

Data Understanding

Data Sources and Suitability:

The dataset for this project is sourced from Kaggle and pertains to customer churn rates within a telecommunications context. It provides detailed information about customer behaviors and attributes, which are essential for predicting churn. This dataset includes various features related to customer demographics, service usage, and account details, making it a comprehensive resource for analyzing patterns that lead to customer attrition.

Dataset Size and Descriptive Statistics:

The dataset comprises 3333 records and includes several features such as:

- account length
- area code
- phone number
- the diffferent plans
- Number of messages and calls
- The call charges
- Customer service calls
- Churn

Feature Justification:

Features included in the dataset are justified based on their relevance to understanding customer behavior. For instance, customer tenure is a critical indicator of potential churn, as longer tenures often correlate with higher retention rates. Service plans and billing information provide insights into customer satisfaction and usage patterns, which are pivotal for identifying churn risks. Each feature contributes valuable information for developing a predictive model.

Data Limitations:

Despite its utility, the dataset has some limitations. It may not capture all factors influencing churn, such as external market conditions or changes in service quality. Additionally, if the dataset is not representative of the entire customer base, predictions may be biased. These limitations should be considered when interpreting the results and developing strategies based on the model's predictions.

Load and understand the data

In [1]:
```
#import libraries
# Data manipulation
import pandas as pd
import numpy as np

# Machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r
eport

# Data visualization (optional)
import matplotlib.pyplot as plt
import seaborn as sns


# Preprocessing (optional)
from sklearn.preprocessing import StandardScaler
```

The next step will be to explore the dataset, our DataFrame will be df

In [2]:
```
#load the data
df = pd.read_csv('./SyriaTel.csv')
```

Displaying part of the dataset

In [3]:
```
df.head(5)
```

Out[3]:

	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 [4]:
```
df.shape
```

Out[4]:
```
(3333, 21)
```

The dataframe has 3333 rows and 21 columns, further checking what information is displayed in the dataset

In [5]:
```
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 [6]:
```
df.describe
```

Out[6]:
```
<bound method NDFrame.describe of      state  account length  area code phone
number 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]>
```

Data cleaning

The customer's phone number is the unique identifier. There should be no duplicate phone numbers.

In [7]:
```
# check for duplicates using the phone number
df.duplicated(subset='phone number').value_counts()
```

Out[7]:
```
False    3333
dtype: int64
```

The data has no duplicates

Checking whether the data has any missing data before any exploration is done

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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
```

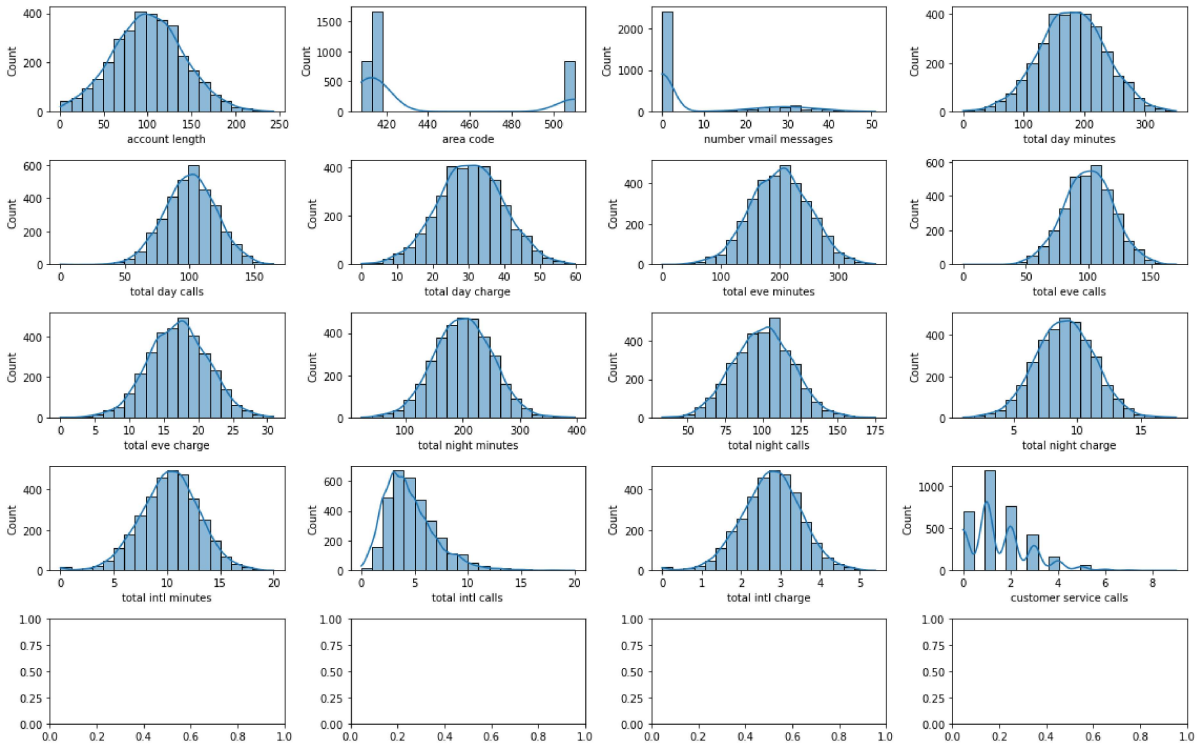
Data appears to have no missing values

Exploratory Data Analysis

Let's display the features on our dataset:

In [9]:
```
# checking for data types
df2 = df.select_dtypes(include=['float64','int64'])
```

In [10]:
```
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(16, 10))

# Flatten the axes for easy iteration
axes = axes.flatten()

# Iterate through numeric columns and plot histograms
for i, feature in enumerate(df2):
    sns.histplot(df[feature], ax=axes[i], kde=True, bins=20)
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Count')

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

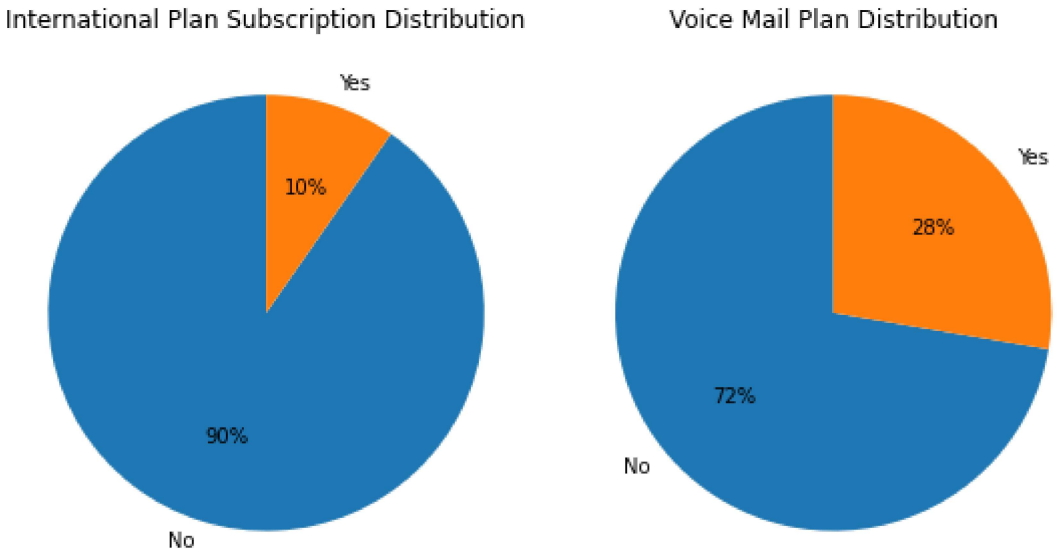


Plans on subscription:

Below is how the distribution on international plan and voice mail plan were recorded:

In [11]:
```
plt.figure(figsize=(8, 6))

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

# Plot the first pie chart
plt.subplot(1, 2, 1)
ax1 = df['international plan'].value_counts()
plt.pie(ax1, labels=['No', 'Yes'], autopct='%.0f%%', startangle=90)
plt.title('International Plan Subscription Distribution')

# Plot the second pie chart
plt.subplot(1, 2, 2)
ax2 = df['voice mail plan'].value_counts()
plt.pie(ax2, labels=['No', 'Yes'], autopct='%.0f%%', startangle=90)
plt.title('Voice Mail Plan Distribution')

# Adjust layout for better spacing
plt.tight_layout()

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

```
<Figure size 576x432 with 0 Axes>
```

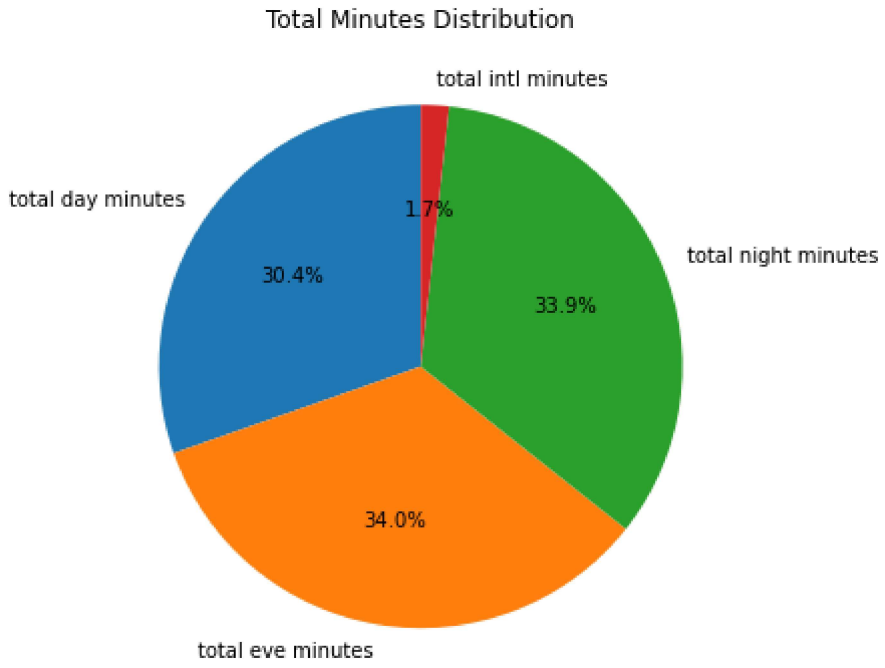


Minutes distribution:

How the minutes were distributed:

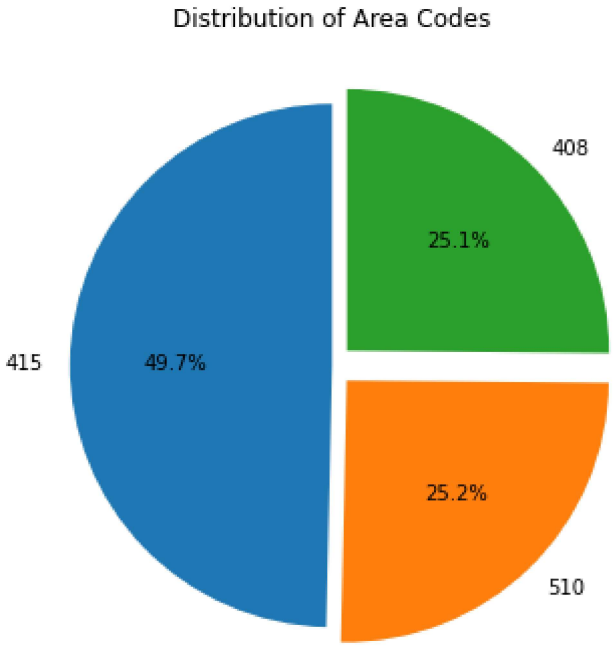
In [12]:
```
#function to create a pie chart
def pie (title, x_axis, labels):
    gsdfgdf
    plt.figure(figsize=(6, 6))
```

In [13]:
```
col_sum = ['total day minutes', 'total eve minutes', 'total night minutes', 't
otal intl minutes']

# Calculate the sum for each column
sums = df[col_sum].sum()

# Plot a pie chart
plt.figure(figsize=(6, 6))
plt.pie(sums, labels=sums.index, autopct='%1.1f%%', startangle=90)
plt.title('Total Minutes Distribution')
plt.show()
```



Area Code Distribution

In [14]:
```
# Update your code to reflect the exact column name
area_code_counts = df['area code'].value_counts()

# Plot a pie chart
plt.figure(figsize=(6, 6))
plt.pie(area_code_counts, labels=area_code_counts.index, autopct='%1.1f%%', st
artangle=90, explode=(0, 0.08, 0.08))
plt.title('Distribution of Area Codes')
plt.show()
```



Calculating Churn rate

Feature Engineering

Converting the Categorical Variables

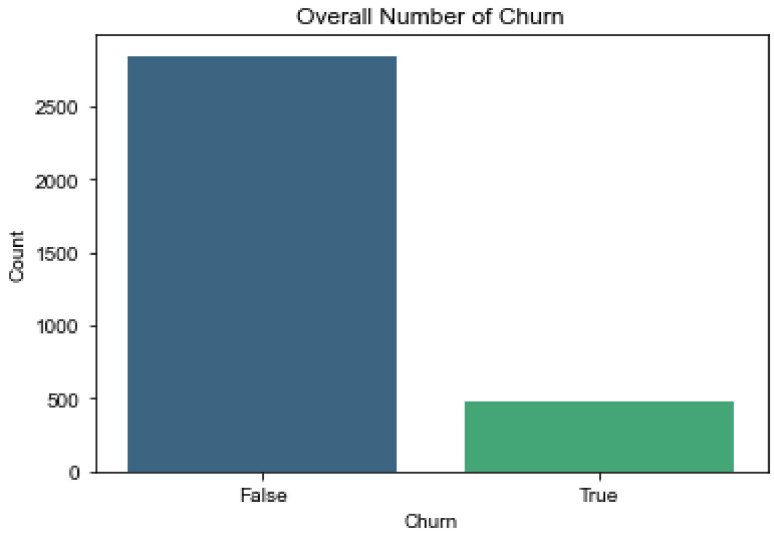
In [15]:
```
from sklearn.preprocessing import OneHotEncoder

df_encoded = pd.get_dummies(df, columns=['state', 'international plan', 'voice mail plan'])
df_encoded
```

Out[15]:

	account length	area code	phone number	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	...	sta
0	128	415	382-4657	25	265.1	110	45.07	197.4	99	16.78	...	
1	107	415	371-7191	26	161.6	123	27.47	195.5	103	16.62	...	
2	137	415	358-1921	0	243.4	114	41.38	121.2	110	10.30	...	
3	84	408	375-9999	0	299.4	71	50.90	61.9	88	5.26	...	
4	75	415	330-6626	0	166.7	113	28.34	148.3	122	12.61	...	
...	...	...	...	...	...	...	...	...	...	...	...	
3328	192	415	414-4276	36	156.2	77	26.55	215.5	126	18.32	...	
3329	68	415	370-3271	0	231.1	57	39.29	153.4	55	13.04	...	
3330	28	510	328-8230	0	180.8	109	30.74	288.8	58	24.55	...	
3331	184	510	364-6381	0	213.8	105	36.35	159.6	84	13.57	...	
3332	74	415	400-4344	25	234.4	113	39.85	265.9	82	22.60	...	

3333 rows × 73 columns

In [16]:
```
# Calculate churn value counts
churn = df['churn'].value_counts()

# Plot churn distribution
sns.barplot(x=churn.index, y=churn.values, palette='viridis')
plt.title('Overall Number of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.style.use('seaborn-darkgrid')
plt.show()

# Calculate churn rate
churn_rate = (df['churn'].mean() * 100)
print('Overall Churn Rate is', round(churn_rate, 2), '%')
```

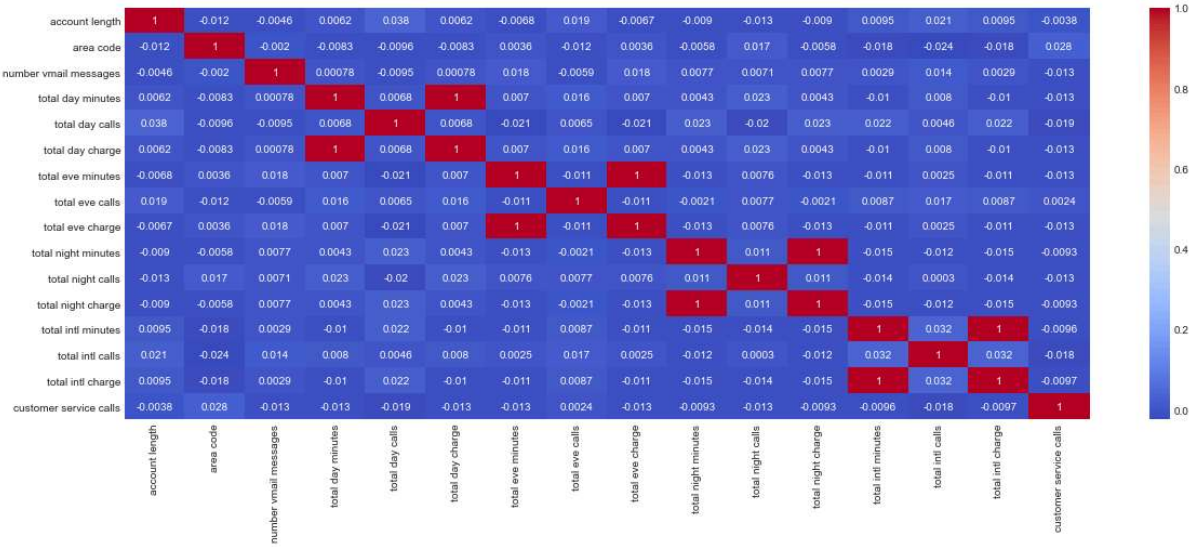



```
Overall Churn Rate is 14.49 %
```

From the graph above, it shows we have an imbalanced issue as there are more unchurned customers than churned one.

How features in this dataset correlate is as below:

In [17]:
```
df2 = df.select_dtypes(include=['float64','int64'])
correlation_matrix = df2.corr()

plt.figure(figsize=(20,7))
sns.heatmap(correlation_matrix, annot = True, cmap='coolwarm')
plt.show()
```



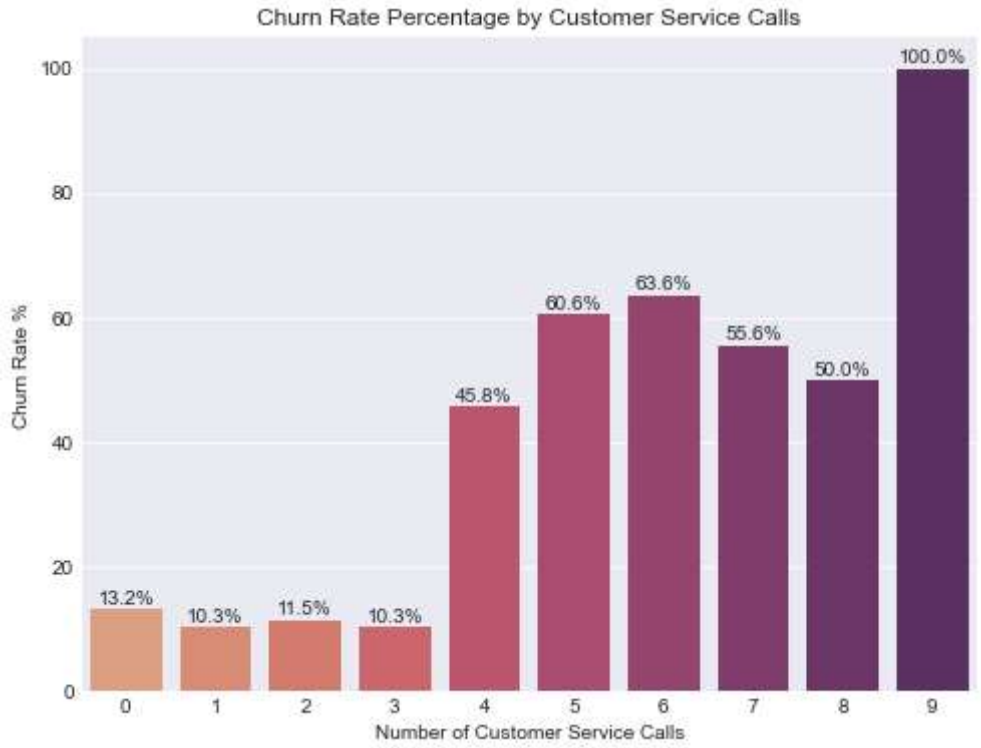
In [18]:
```
corr_matrix = df.corr()
churn_corr = corr_matrix['churn'].sort_values(ascending=False)
print(churn_corr)
```

```
churn                    1.000000
customer service calls    0.208750
total day minutes         0.205151
total day charge          0.205151
total eve minutes         0.092796
total eve charge          0.092786
total intl charge         0.068259
total intl minutes        0.068239
total night charge        0.035496
total night minutes       0.035493
total day calls           0.018459
account length            0.016541
total eve calls           0.009233
area code                 0.006174
total night calls         0.006141
total intl calls         -0.052844
number vmail messages    -0.089728
Name: churn, dtype: float64
```

In [19]:
```
df.columns
```

Out[19]:
```
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')
```

In [20]:
```
# Calculate churn rate percentage for each number of customer service calls
churn_rate = df.groupby('customer service calls')['churn'].mean() * 100

# Plotting a bar plot
plt.figure(figsize=(8, 6))
ax = sns.barplot(x=churn_rate.index, y=churn_rate.values, palette='flare')

# Manually add labels
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height:.1f}%',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='center',
                xytext=(0, 5),
                textcoords='offset points')

# Adding title and labels
plt.title('Churn Rate Percentage by Customer Service Calls')
plt.xlabel('Number of Customer Service Calls')
plt.ylabel('Churn Rate %')

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



There is a positive correlation between the number of customer service calls and the likelihood of churn.

- As the number of customer service calls increases, customers are more likely to churn, or stop using the service. However, there is a disparity in the trend. Specifically, there is a noticeable increase in churn at the 6th customer service call.
- This observation indicates that there might be a particular threshold or point where additional customer service interactions start to have a negative impact on customer retention, potentially leading to a higher churn rate.
- While there is a general trend of increasing churn with more customer service calls, it highlights a potential anomaly or critical point at the 6th customer service call where churn rate spikes, suggesting that this specific interaction might be a key factor in customer decision-making regarding retention.

Data Preparation

Feature Scaling (Optional) For logistic regression, it's often useful to scale numerical features to improve model performance:

In [21]:
```
# Initialize scaler
scaler = StandardScaler()

# Select numerical features
numerical_features = df_encoded.select_dtypes(include=['float64', 'int64']).co
lumns

# Scale numerical features
df_encoded[numerical_features] = scaler.fit_transform(df_encoded[numerical_fea
tures])
```

Data Modelling

We start with a Logistic Regression for our baseline model

Split Data for Training and Testing

In [22]:
```
# Define features and target
X = df_encoded.drop(['churn', 'phone number'], axis=1)
y = df_encoded['churn']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=42)
```

In [23]:
```
# preview of X_train
X_train.head()
```

Out[23]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total cha
2016	-0.529052	1.718817	-0.591760	0.415447	0.875326	0.415605	1.167897	-1.662395	1.168
1362	-0.956015	1.718817	-0.591760	-0.860738	1.074667	-0.861122	-2.642285	-1.913404	-2.641
2670	0.375104	1.718817	0.285029	0.756987	0.376972	0.756930	-0.985684	0.897892	-0.984
2210	-0.755091	-0.523603	-0.591760	1.820168	0.476643	1.819789	-0.212604	-1.662395	-0.211
1846	0.475566	1.718817	2.550065	-0.034431	0.825491	-0.034814	-0.510398	-1.511790	-0.511

5 rows × 71 columns

SMOTE FOR CLASS IMBALANCE

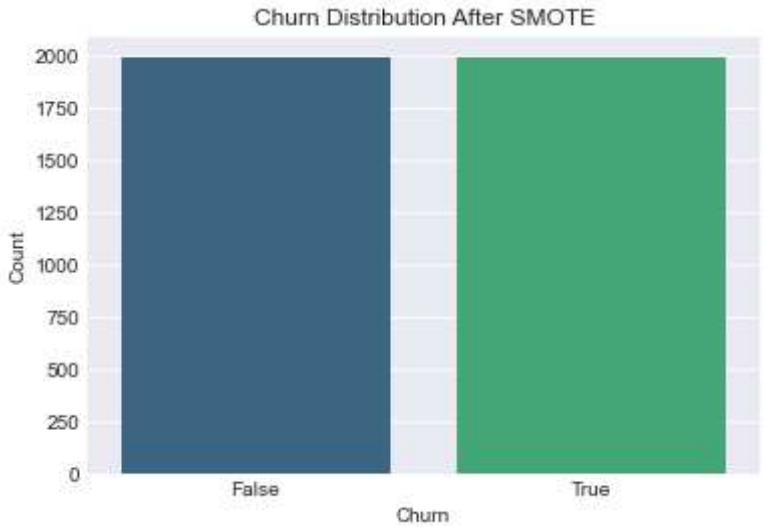
As we had seen above, there was class imbalance, using SMOTE to handle class imbalance problems by oversampling the minority class.

In [25]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state = 42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Preview synthetic sample class distribution
print(pd.Series(y_train_resampled).value_counts())
```

```
True     1993
False    1993
Name: churn, dtype: int64
```

Checking to see whether the data balanced

In [26]:
```
# Convert y_train_resampled to a DataFrame to use value_counts
y_train_resampled_df = pd.DataFrame(y_train_resampled, columns=['churn'])

# Calculate churn value counts for resampled data
churn_resampled = y_train_resampled_df['churn'].value_counts()

# Plot churn distribution for resampled data
sns.barplot(x=churn_resampled.index, y=churn_resampled.values, palette='viridis')
plt.title('Churn Distribution After SMOTE')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.style.use('seaborn-darkgrid')
plt.show()

# Calculate churn rate for resampled data
churn_rate_resampled = (y_train_resampled.mean() * 100)
print('Overall Churn Rate After SMOTE is', round(churn_rate_resampled, 2), '%')
```




```
Overall Churn Rate After SMOTE is 50.0 %
```

The data is now balanced

1.LOGISTIC REGRESSION

Train and Evaluate the Logistic Regression Model

In [27]:
```
# perform train test split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rand
om_state=42)

# Create a logistic regression model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')

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

Out[27]:
```
LogisticRegression

LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinea
r')
```

In [28]:
```
# Generate predictions
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```

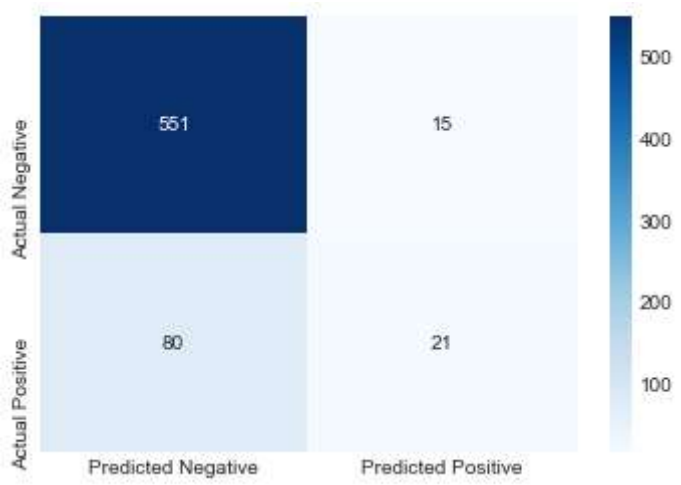
In [29]:
```
# Calulate the performance metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score, roc_curve, auc

# Accuracy
accuracy = accuracy_score(y_test, y_hat_test)
print(f"Accuracy: {accuracy}")

# Precision
precision = precision_score(y_test, y_hat_test)
print(f"Precision: {precision}")

# Recall
recall = recall_score(y_test, y_hat_test)
print(f"Recall: {recall}")

# F1-score
f1 = f1_score(y_test, y_hat_test)
print(f"F1-score: {f1}")

# False positive rate(fpr) and true positive rate(tpr)
fpr, tpr, thresholds = roc_curve(y_test, y_hat_test)

# calculate the AUC
auc = auc(fpr, tpr)
print(f"AUC: {auc}")
```

```
Accuracy: 0.8575712143928036
Precision: 0.5833333333333334
Recall: 0.2079207920792079
F1-score: 0.30656934306569344
AUC: 0.5907095126473778
```

- Accuracy: Our classifier shows that our model 85% accurate
- Presicion: Out of all the instances our model predicted as positive, approximately 58.33% were actually positive.
- Recall: Being approximately 20.79%, means that out of all the actual positive instances, our model identified approximately 20.79% correctly.
- F1 score provides a balance between precision and recall. In our case, the F1-score is approximately 30.65%.
- AUC is approximately 0.59, suggesting that our model's ability to distinguish between positive and negative classes is more or less the same as random guessing.

In [30]:
```
# Build a confusion matrix
from sklearn.metrics import confusion_matrix

# Confusion matrix
cm = confusion_matrix(y_test, y_hat_test)

#  Visualize
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
```

Out[30]: <AxesSubplot:>

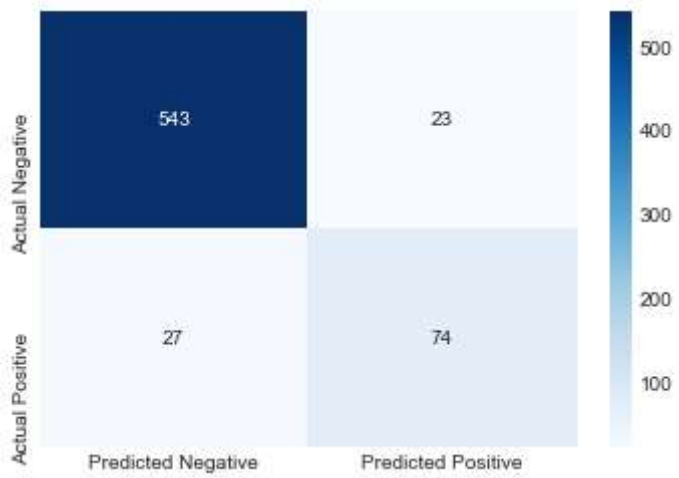


- The model predicted 21 True Positives, 551 True Negatives, 15 False Positives and 80 False Negatives.
- The model predicted 21 customers would churn and they did.
- The model predicted that 551 customers would not churn and they didn't
- The model predicted that 15 customers would churn but they didn't
- The model incorrectly predicted that 80 customers would not churn but they actually churned

Trying an alternative model to see whether there will be improved model performance

2.DECISION TREES

In [32]:
```
from sklearn.tree import DecisionTreeClassifier
#Instantiate DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(random_state=42)
#Fit on the training data
dt_clf.fit(X_train, y_train)

#predict on the test set
dt_y_pred = dt_clf.predict(X_test)
cm2 = confusion_matrix(y_test, dt_y_pred)
# Visualize the confusion matrix
sns.heatmap(cm2, annot=True, fmt="d",  cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.show()
```



We can see that the model made correct predictions for 543 negative cases and 74 positive cases. However, it incorrectly predicted 27 positive cases as negative and 23 negative cases as positive.

In [34]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_recall_curve,
roc_auc_score

# Assuming y_test and dt_y_pred are already defined
accuracy = accuracy_score(y_test, dt_y_pred)
print(f"Accuracy: {accuracy}")

f1 = f1_score(y_test, dt_y_pred)
print(f"F1-score: {f1}")

# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_test, dt_y_pred)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, dt_y_pred)

print(f"ROC AUC: {roc_auc}")
```

```
Accuracy: 0.9250374812593704
F1-score: 0.7474747474747475
ROC AUC: 0.846018612461953
```

- Accuracy: Our classifier shows that our model 92.5% accurate
- The F1-score is approximately 74.74%.
- AUC is approximately 0.846

3.RANDOM FOREST

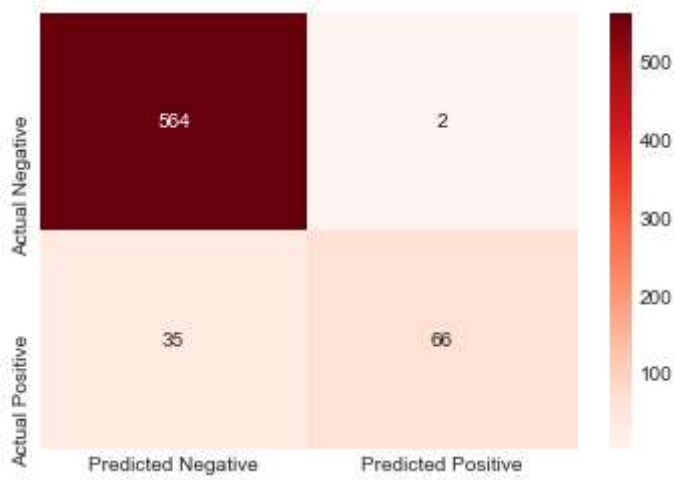
In [35]:
```
# Create and fit a random forest classifier
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
```

In [36]:
```
# Predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print(f"F1-score: {f1}")

# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_pred)

print(f"ROC AUC: {roc_auc}")
```

```
Accuracy: 0.9445277361319341
F1-score: 0.7810650887573966
ROC AUC: 0.82496588881503
```

- F1-score being 78.1% is moderately good performance. AUC of 82% is okay for the imbalanced data. This gives a generally good performance of the model.
- Having an accuracy of 94.45%, performs better than that of Logistic Regression Model, 85.3% and decision trees accuracy of 92.5%

In [37]:
```
from sklearn.metrics import confusion_matrix

cm1 = confusion_matrix(y_test, y_pred)
# Visualize the confusion matrix
sns.heatmap(cm1, annot=True, fmt="d",  cmap='Reds',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.show()
```



- The model predicted 66 True Positives, 564 True Negatives, 2 False Positives and 35 False Negatives.
- The model predicted 66 customers would churn and they did.
- The model predicted that 564 customers would not churn and they didn't
- The model predicted that 2 customers would churn but they didn't
- The model incorrectly predicted that 35 customers would not churn but they actually churned

Random Forest has performed better both of the models before, let's use hyperparameter tuning.

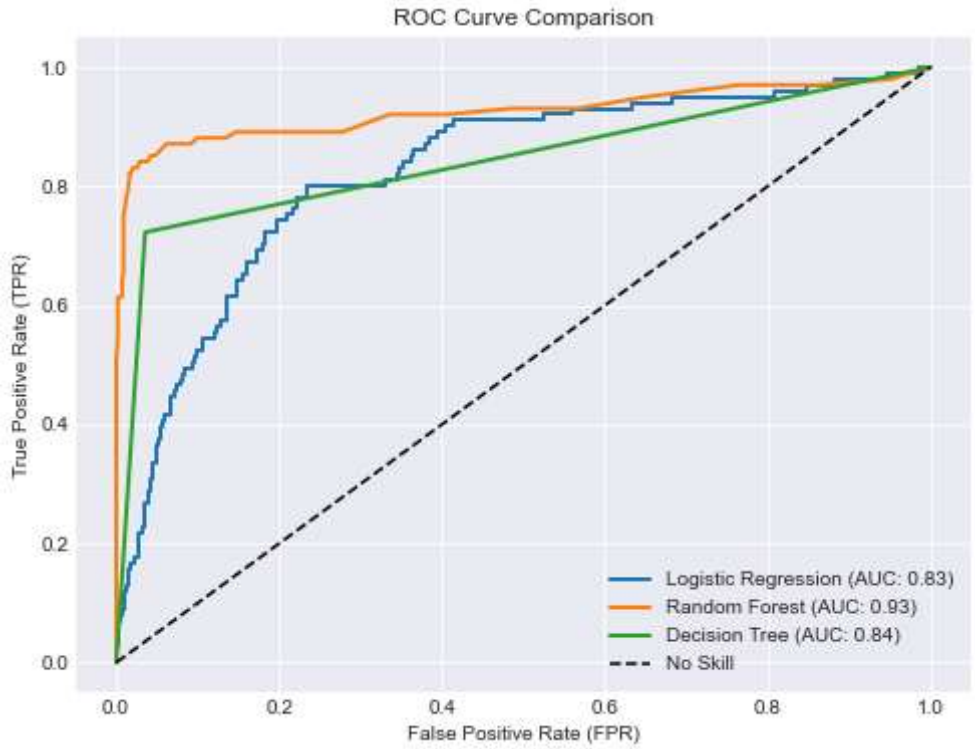
Hyperparameter tuning

In [38]:
```
# Define hyperparameter grid
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 125, 150],
    'max_depth': [5, 10, 15, 20, 25],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy'],
}

# Create Random Forest model
clf = RandomForestClassifier(random_state=0)

# Initialize GridSearchCV with early stopping and smaller training fraction
grid_search = GridSearchCV(clf, param_grid, scoring='f1', cv=3, n_jobs=-1)

# Fit the model on a smaller portion of training data (replace 0.8 with a suit
able fraction)
grid_search.fit(X_train[:int(0.8 * len(X_train))], y_train[:int(0.8 * len(y_tr
ain))])

 # Get best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

 # Evaluate on the testing set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"F1-score: {f1}")
print(f"Best Hyperparameters: {best_params}")
```

```
Accuracy: 0.9325337331334332
F1-score: 0.7169811320754716
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 25, 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

Displaying the AUC values of Logistic Regression, Decision trees and Random Forest models

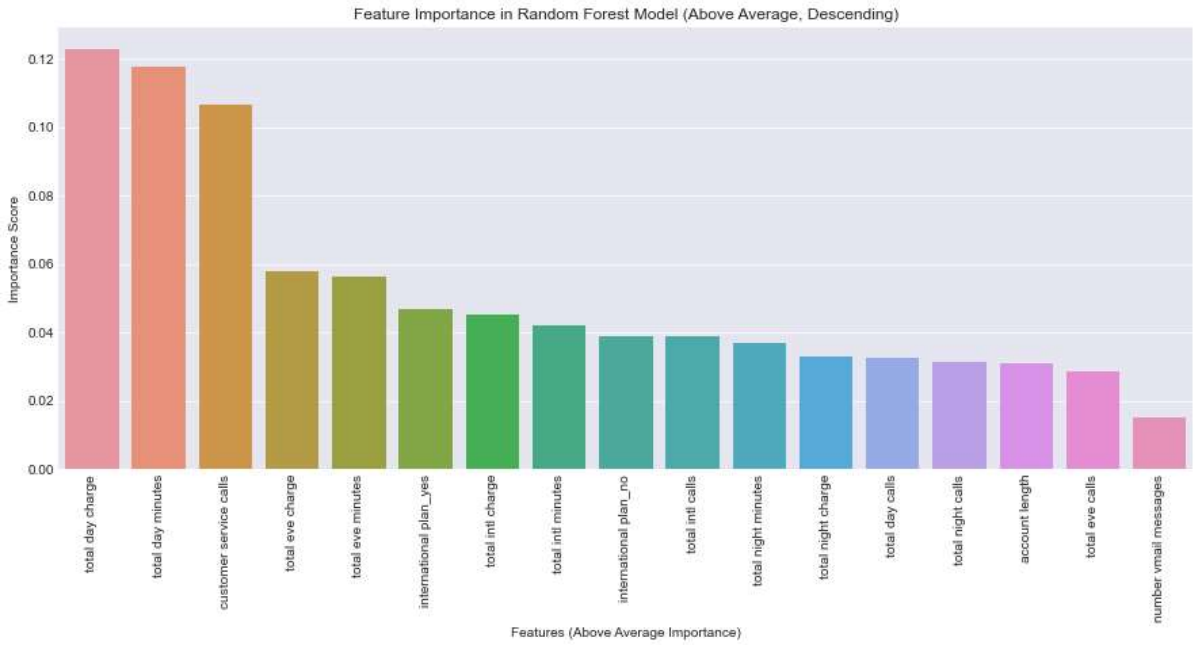
In [46]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Fit your models (assuming X_train and y_train are defined)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize and fit models
lr = LogisticRegression()
rf = RandomForestClassifier()
dt = DecisionTreeClassifier()

lr.fit(X_train, y_train)
rf.fit(X_train, y_train)
dt.fit(X_train, y_train)

# Predict probabilities for the test set
lr_predictions_proba = lr.predict_proba(X_test)[:, 1]  # Probabilities for the
positive class
rf_predictions_proba = rf.predict_proba(X_test)[:, 1]
dt_predictions_proba = dt.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC for each model
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_predictions_proba)
roc_auc_lr = auc(fpr_lr, tpr_lr)

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_predictions_proba)
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_predictions_proba)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC: {:.2f})'.format(roc_
auc_lr), linewidth=2)
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC: {:.2f})'.format(roc_auc_r
f), linewidth=2)
plt.plot(fpr_dt, tpr_dt, label='Decision Tree (AUC: {:.2f})'.format(roc_auc_d
t), linewidth=2)

plt.plot([0, 1], [0, 1], linestyle='--', color='black', label='No Skill')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve Comparison')
plt.legend()
plt.grid(True)
plt.show()
```



In [40]:
```
# first drop the 'state' columns
feature_names = df2.drop(columns=[col for col in df2.columns if 'state' in co
l])
feature_names.columns
```

Out[40]:
```
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')
```

In [42]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier

# Define and fit the model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)  # Fit the model with training data

# Now you can access feature_importances_
feature_importances = clf.feature_importances_
feature_names = X_train.columns

# Calculate average importance score
avg_importance = feature_importances.mean()

# Filter features with above-average importance
important_features = feature_names[feature_importances > avg_importance]
important_importances = feature_importances[feature_importances > avg_importan
ce]

# Sort features and importances in descending order
sorted_idx = important_importances.argsort()[::-1]  # Reverse order for descen
ding
sorted_features = important_features[sorted_idx]
sorted_importances = important_importances[sorted_idx]

# Create the bar plot
plt.figure(figsize=(15, 6))
sns.barplot(x=sorted_features, y=sorted_importances)
plt.xlabel('Features (Above Average Importance)')
plt.ylabel('Importance Score')
plt.title('Feature Importance in Random Forest Model (Above Average, Descendin
g)')

# Rotate feature names for better readability
plt.xticks(rotation=90)

plt.show()
```



- From the graph, we can conclude the top features are:
  - Total day charges and minutes
  - Customer service calls
  - Total evening charge and minutes
  - Interntional plan

CONCLUSION

Random Forest model appears to be the best model to predict the customers likely to churn.

- The churn prediction analysis conducted for SyriaTel aimed to develop a classifier to identify customers likely to terminate their services. Through comprehensive data exploration, preparation, and modeling, several key findings emerged:
  - Model Performance: Random Forest emerged as the most effective model for churn prediction, outperforming Logistic Regression, and Decision Trees. It exhibited superior accuracy and predictive power, making it the preferred choice for SyriaTel's churn prediction system.
  - Key Predictive Features: Total day charges and minutes, customer service calls, total evening chearges and minutes and subscription to the international plan were identified as crucial indicators of churn. These insights provide valuable guidance for SyriaTel in devising proactive retention strategies targeted at high-risk customers.

RECOMMENDATIONS

- Improve Call Quality: Invest in advanced infrastructure and technology to boost call quality, thereby providing a superior customer experience.
- Improve Customer Service: Prioritize improvements in customer service by shortening response times, boosting efficiency in resolving issues, and providing personalized support.
- Customized Plans for International Subscribers: Develop appealing plans and offers tailored specifically for international subscribers to enhance satisfaction and lower churn rates.
- Retention Measures: Deploy proactive strategies, including targeted promotions, loyalty rewards, and personalized communication, to effectively retain customers who are at risk of churning.
- Ongoing Monitoring: Consistently analyze customer behavior and churn trends, and update models and strategies regularly to stay aligned with evolving market conditions.