BUSINESS UNDESTANDING

Stakeholder: SyriaTel, a prominent player in the telecommunications sector, is committed to comprehending and addressing the challenge of customer churn to optimize revenue streams and elevate overall customer satisfaction.

Business Challenge: SyriaTel grapples with a substantial rate of customer churn, where subscribers terminate their services, posing a financial risk. This underscores the necessity of recognizing patterns and elements that contribute to the phenomenon of churn.

Project Goals:

Anticipate whether a customer is inclined to churn in the immediate future. Reveal insights actionable for devising targeted retention strategies. Importance: Mitigating customer churn is pivotal for ensuring a resilient and profitable customer base. Proactively addressing the factors influencing churn empowers SyriaTel to introduce customer-centric initiatives, enhance service quality, and cultivate enduring customer loyalty.

Key Inquiries:

What are the primary drivers of customer churn within the telecom industry? Can we effectively predict customers with the highest likelihood of churning? How can the findings from the analysis guide the implementation of personalized retention strategies? Performance Metrics:

Accuracy in predicting churn/non-churn. Precision and recall metrics to strike a balance between false positives and false negatives. Evaluation of feature importance to pinpoint crucial factors contributing to churn.

DATA UNDERSTANDING

Dataset Overview: The SyriaTel Customer Churn dataset provides a comprehensive snapshot of customer-related information within the telecommunications domain. Comprising diverse features, it aims to capture key aspects influencing the likelihood of customer churn.

Features of the Dataset: state: state where a customer lives account length: the number of days the customer has had an account area code: the area code of the customer phone number: the phone number of the customer international plan: true if the customer has the international plan, otherwise false voice mail plan: true if the customer has the voice mail plan, otherwise false number vmail messages: the number of voicemails the customer has sent total day minutes: total number of minutes the customer has been in calls during the day total day calls: total number of calls the user has done during the day total day charge: total amount of money the customer was charged by the Telecom company for calls during the day total eve minutes: total number of minutes the customer has been in calls during the evening total eve calls: total number of calls the customer has done during the evening total eve charge: total amount of money the customer was charged by the Telecom company for calls during the evening total night minutes: total number of minutes the customer has been in calls during the night total night calls: total number of calls the customer has done during the night total night charge: total amount of money the customer was charged by the Telecom company for calls during the night total intl minutes: total number of minutes the user has been in international calls total intl calls: total number of international calls the customer has done total intl charge: total amount of money the customer was charged by the Telecom company for international calls customer service calls: number of calls the customer has made to customer service churn: true if the customer terminated their contract, otherwise false

Data Quality: An initial assessment indicates a well-structured dataset with minimal missing values. However, a detailed exploration will be conducted to identify and address any potential outliers, inconsistencies, or anomalies.

Potential Challenges:

Imbalanced Classes: The dataset may exhibit imbalances between churn and non-churn instances, requiring appropriate handling during model training. Feature Correlation: Certain features might be correlated, necessitating careful consideration during feature selection. Exploratory Data Analysis (EDA): Exploring key statistical measures, visualizations, and relationships between variables will be crucial to gaining deeper insights into the dataset's characteristics.

Data Exploration Goals:

Understand the distribution of key features. Identify potential correlations or patterns. Address any data quality issues discovered during exploration.

Importing libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
%matplotlib inline
```

EDA

In [2]:
```python
# Reading dataset
df = pd.read_csv("bigml_59c28831336c6604c800002a.csv")
```

In [3]:
```python
#dataset shape
df.shape
```

Out[3]: (3333, 21)

In [4]:
```python
#sample of the data
df.head()
```

Out[4]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	to c ca
0	KS	128	415	382-4657	no	yes	25	265.1	1
1	OH	107	415	371-7191	no	yes	26	161.6	1
2	NJ	137	415	358-1921	no	no	0	243.4	1
3	OH	84	408	375-9999	yes	no	0	299.4	
4	OK	75	415	330-6626	yes	no	0	166.7	1

5 rows × 21 columns

In [5]:
```python
#checking for missing values
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
#checking for duplicate values
len(df.loc[df.duplicated()])
```

Out[6]:
```
0
```

In [7]:
```
#dataset information
df.info()
```

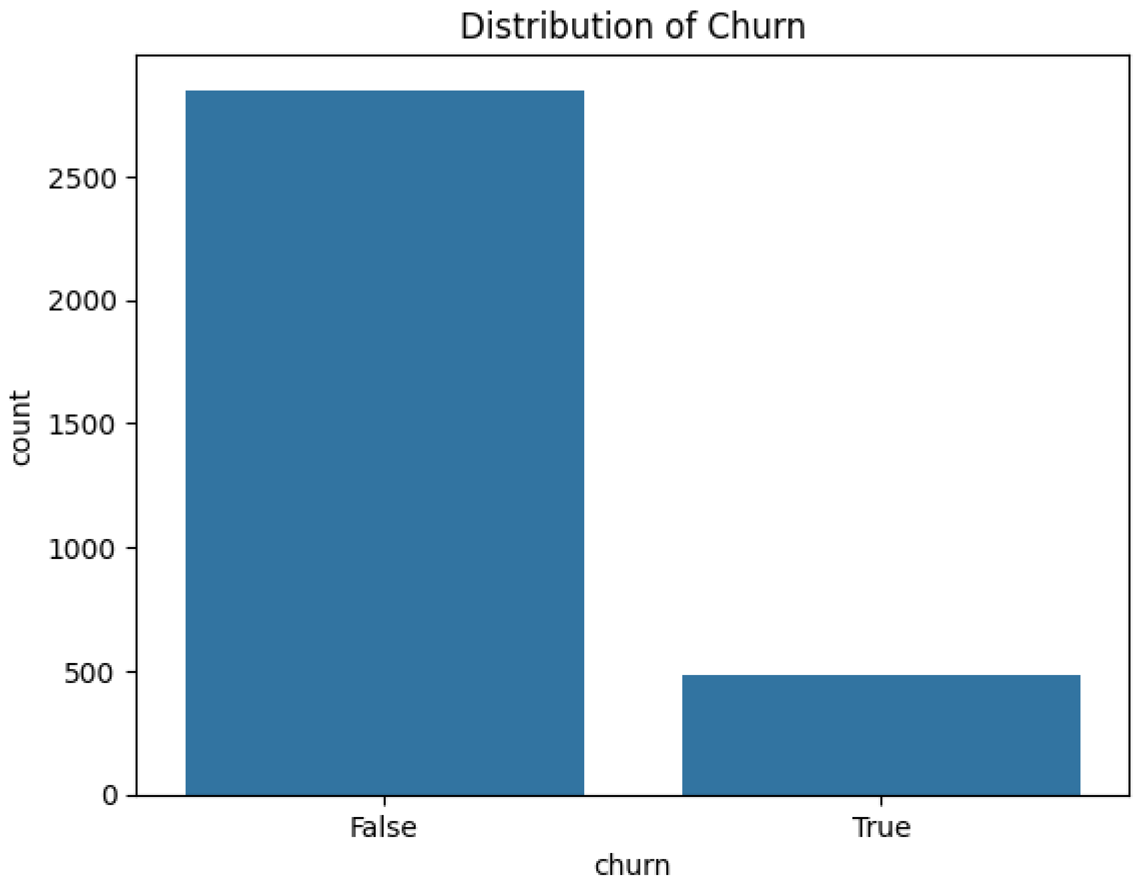
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   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
```

Analysis of churn variable

In our analysisi,churn is the dependebnt variable.It indicartes if a customer is still with Sriaa Tel(False) or if they have terminated and are not customers(True)

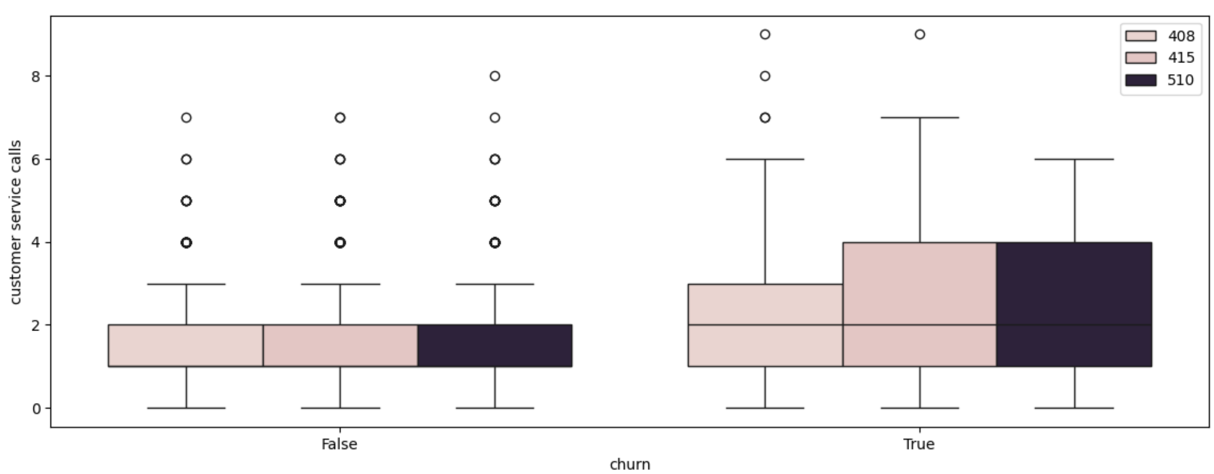
In [8]:

```
# Explore the distribution of the target variable (e.g., 'Churn')
sns.countplot(x='churn', data=df)
plt.title('Distribution of Churn')
plt.show()
```



Out of the 3,333 individuals in the data set, 483 have concluded their agreement with SyriaTel, representing a loss of 14.5% of customers. The data exhibits an imbalance in the distribution of binary classes, indicating that corrective measures are necessary before modeling. An unbalanced feature may lead the model to produce inaccurate predictions and should be addressed.

In [9]:
```
# Boxplot to see which area code has the highest churn
plt.figure(figsize=(14,5))
sns.boxplot(data=df,x='churn',y='customer service calls',hue='area code');
plt.legend(loc='upper right');
```

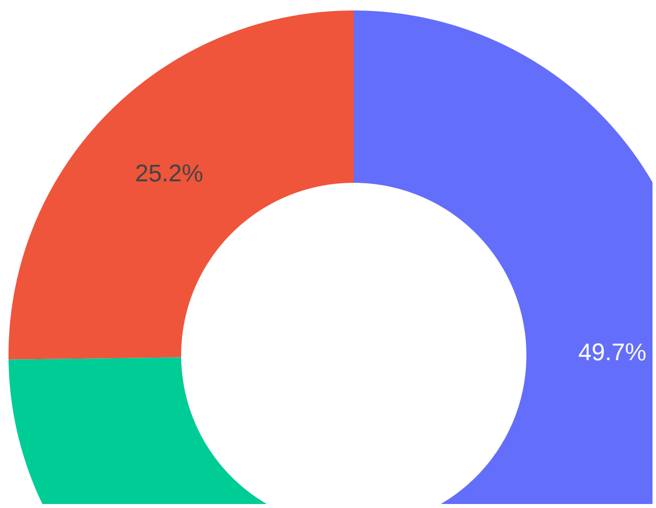


Loading [MathJax]/extensions/Safe.js

Analysis on "area code"

In [10]:
```
# Pie chart of area code feature
import plotly.express as px
area = df['area code'].value_counts()
transuction = area.index
quantity = area.values

# draw pie circule with plotly
figure = px.pie(df,
               values = quantity,
               names = transuction,
               hole = .5,
               title = 'Distribution of Area Code Feature')
figure.show()
```

Distribution of Area Code Feature

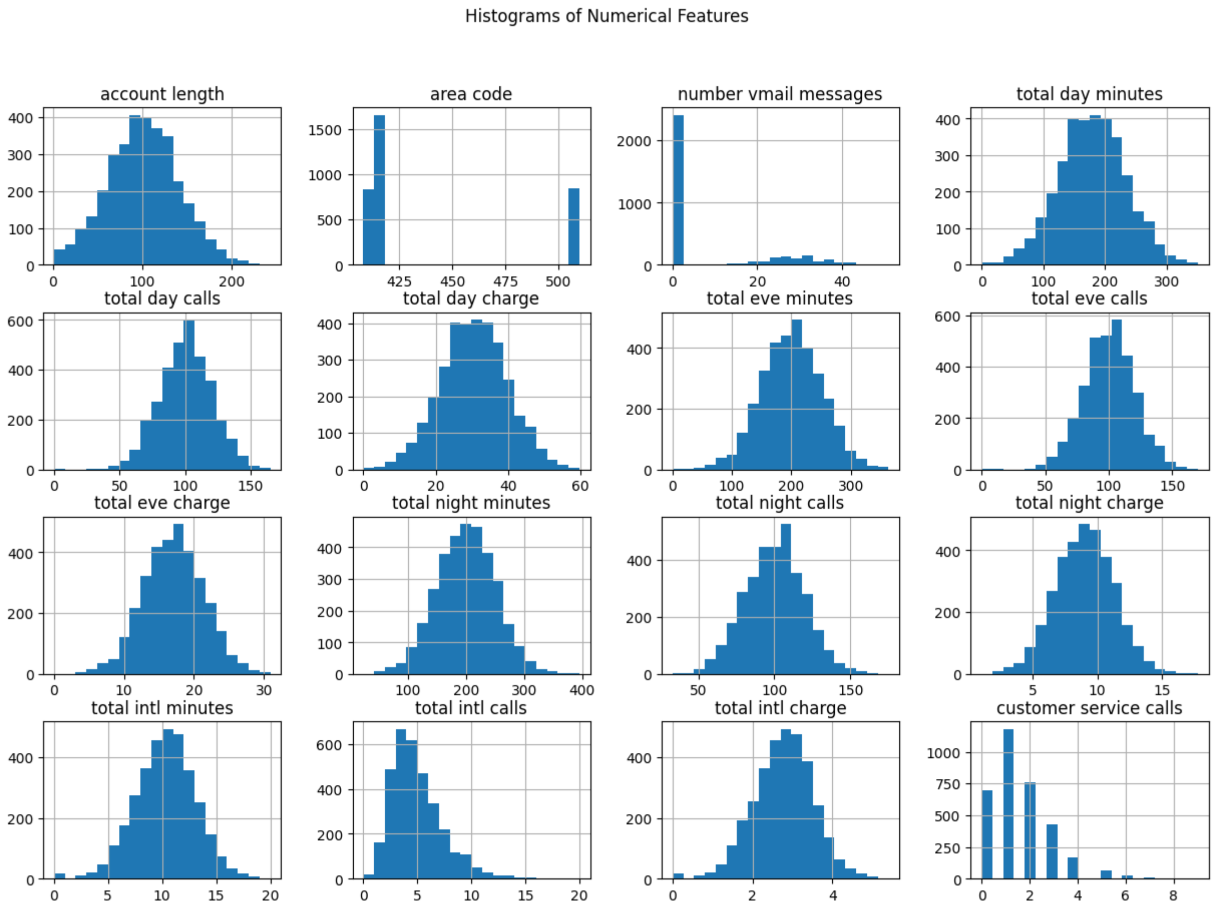


Almost half of the customers are from the area code 415,while area code 510 and 408 is a quarter of the total customers in each area code

Analysis numerical features

In [11]:

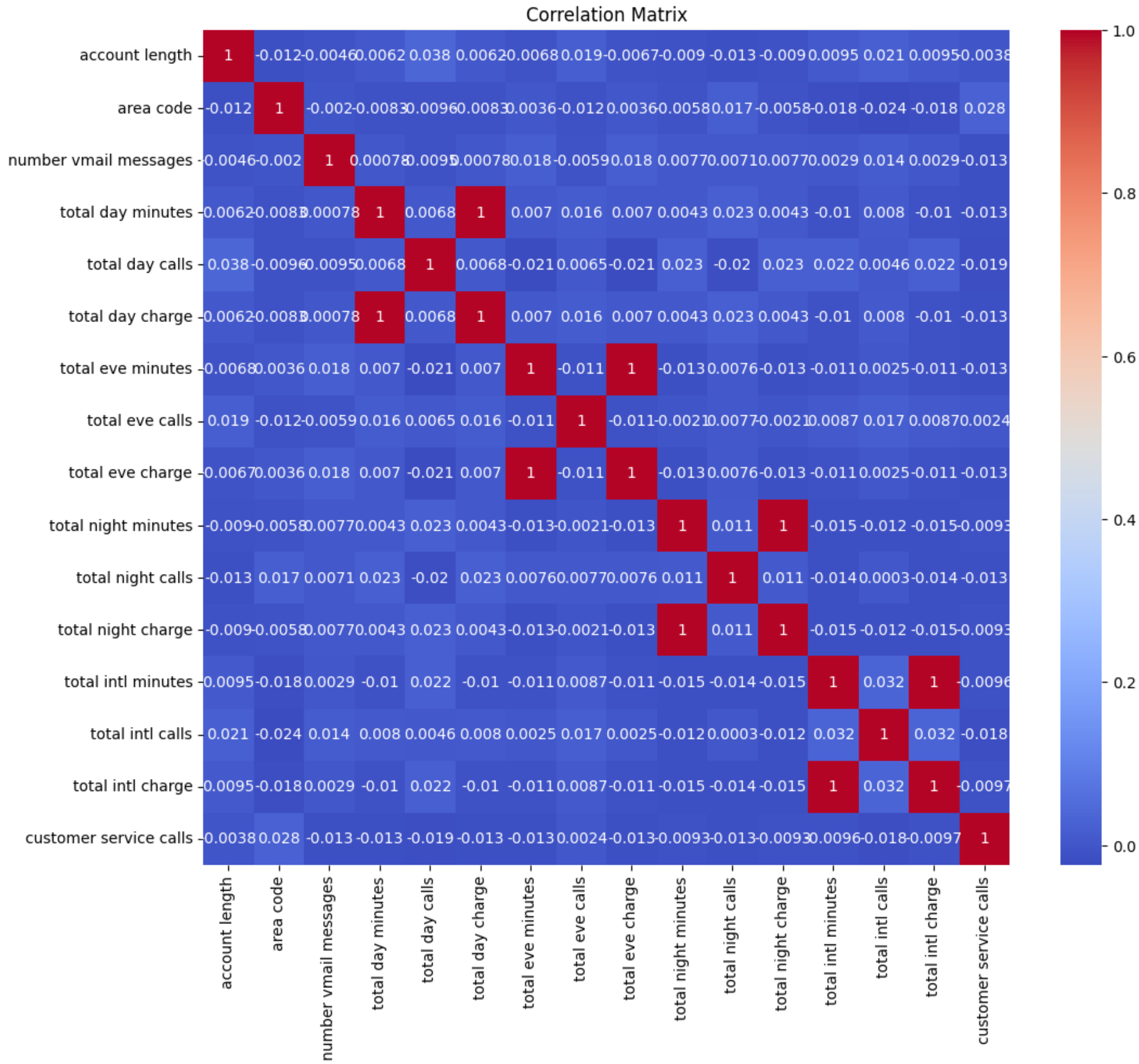
```python
# Explore numerical features
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns
df[numerical_features].hist(bins=20, figsize=(15, 10))
plt.suptitle('Histograms of Numerical Features')
plt.show()
```



Correlation matrix

In [12]:

```python
# Explore correlation between numerical features
correlation_matrix = df[numerical_features].corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

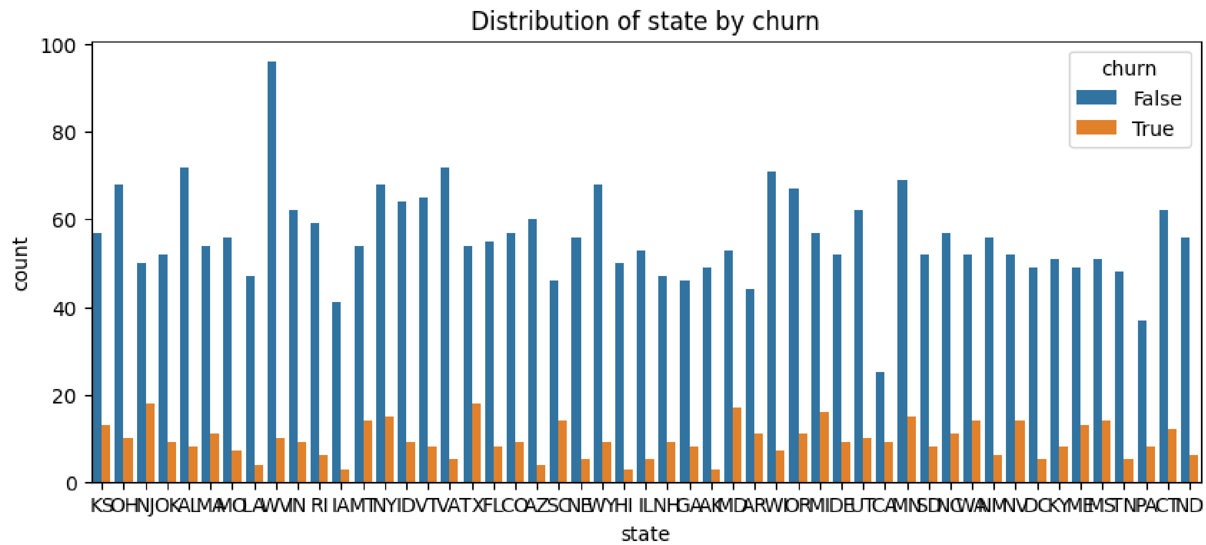


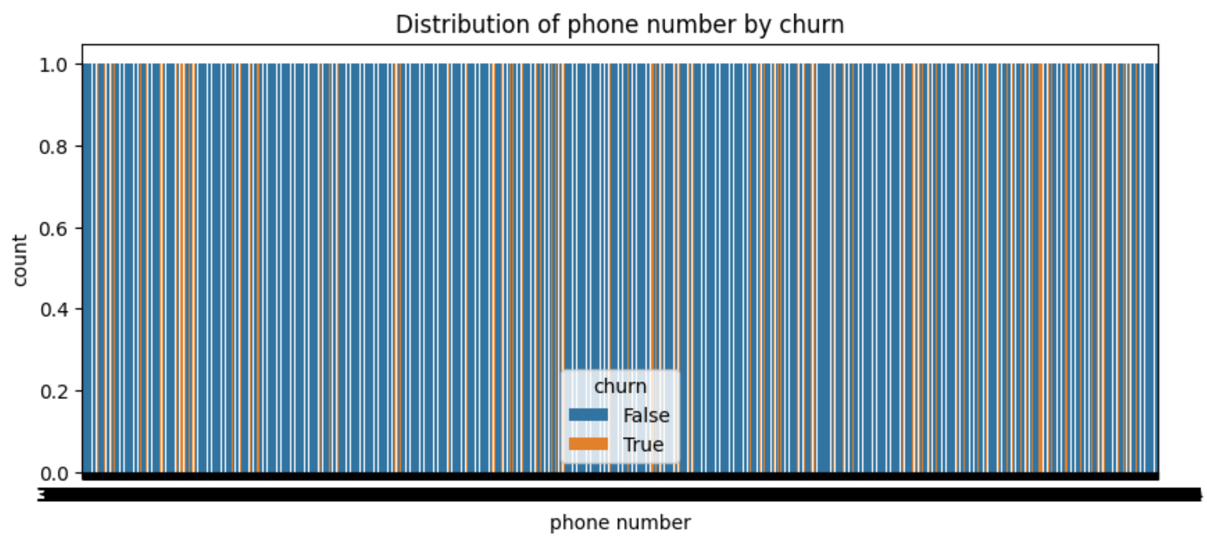
Although the majority of the features exhibit little to no correlation, there are instances of perfect positive correlation among specific pairs. Notably, the features such as Total Day Charge and Total Day Minutes, Total Eve Charge and Total Eve Minutes, Total Night Charge and Total Night Minutes, as well as Total Intl Charge and Total Intl Minutes, demonstrate perfect positive correlation. This alignment is logical, given that the charge incurred is a direct outcome of the corresponding minutes utilized in each category. The perfect correlation coefficient of 1 signifies the existence of perfect multicollinearity, a phenomenon that does not exert the same influence on nonlinear models as it does on linear models. While certain nonlinear models may be affected by perfect multicollinearity, others remain resilient to its impact.

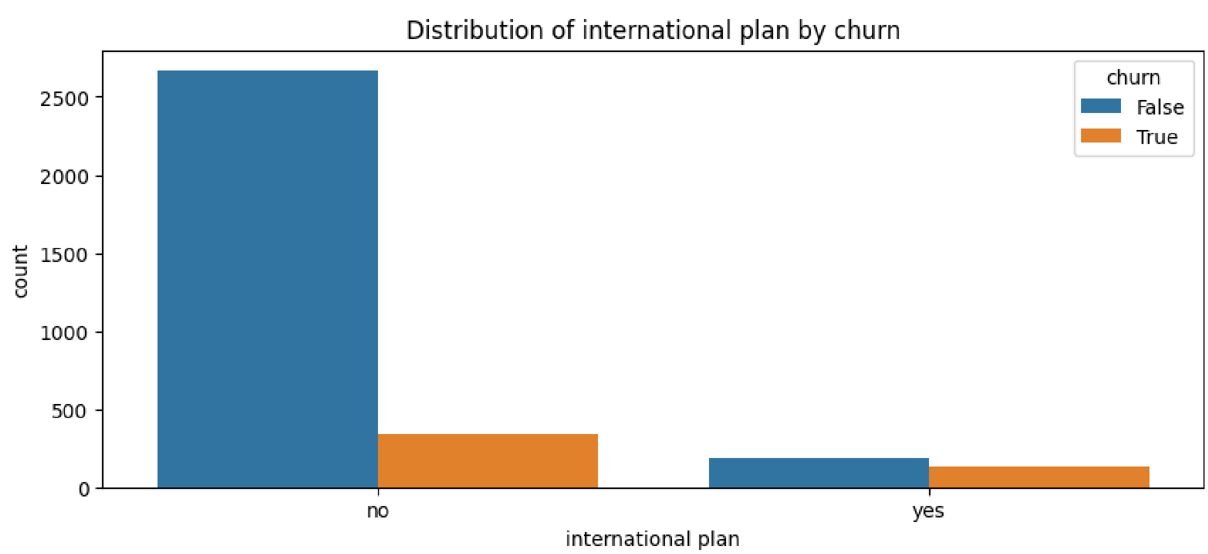
Analysis of the feature "churn"

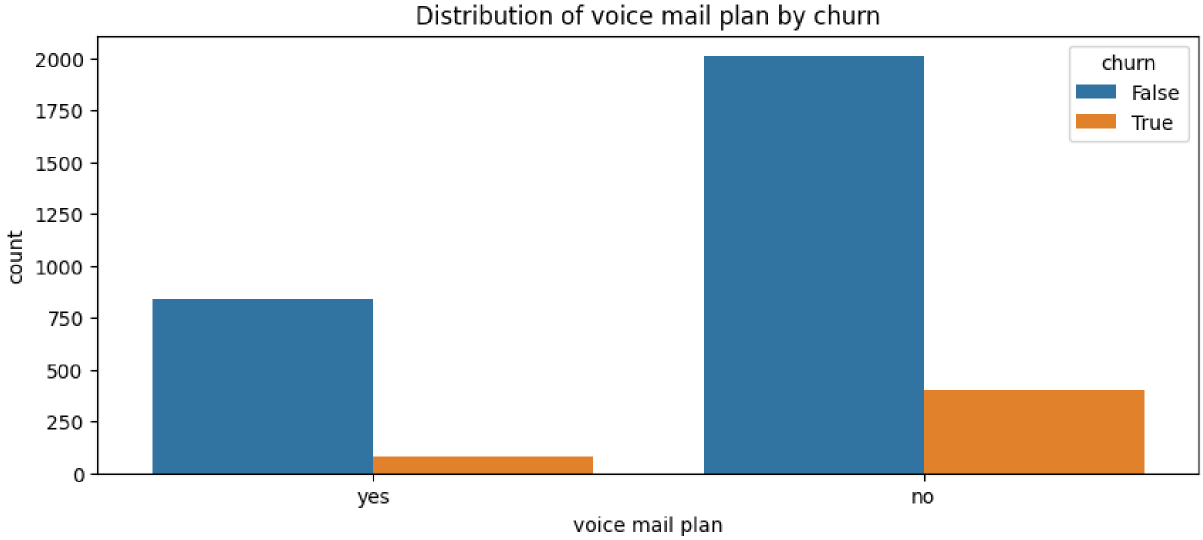
In [13]:
```
# Explore categorical features
categorical_features = df.select_dtypes(include=['object']).columns
```

```python
for feature in categorical_features:
    plt.figure(figsize=(10,4))
    sns.countplot(x=feature, data=df, hue='churn')
    plt.title(f'Distribution of {feature} by churn')
    plt.show()
```









Outliers

In [16]:
```
#removing numerical outliers from a DataFrame using the Z-score
import numpy as np
from scipy import stats

def drop_numerical_outliers(df, z_thresh=3):
    constraints = df.select_dtypes(include=[np.number]).apply(lambda x: np.a
    df.drop(df.index[~constraints], inplace=True)

# Assuming 'df' is your DataFrame
# You can call the function like this:
drop_numerical_outliers(df)
```

In [17]:
```
df.shape
```

Out[17]: (3169, 21)

Dropping features with high correlation

In [18]:
```
# Select only numeric columns
df1 = df.select_dtypes(include=[np.number])

# Calculate the correlation matrix and take the absolute value
corr_matrix = df1.corr().abs()

# Create a True/False mask to identify the upper triangle of the matrix
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Use the mask to create a DataFrame of the same shape with upper triangle w
tri_df = corr_matrix.mask(mask)

# List column names of highly correlated features (correlation > 0.90)
to_drop = [c for c in tri_df.columns if any(tri_df[c] > 0.90)]
```

```
# Drop the highly correlated features from the original DataFrame
df2 = df.drop(to_drop, axis=1)
```

In [19]:
```
df2.head(10)
```

Out[19]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day calls	tot da charg
0	KS	128	415	382-4657	no	yes	25	110	45.(
1	OH	107	415	371-7191	no	yes	26	123	27.4
2	NJ	137	415	358-1921	no	no	0	114	41.3
3	OH	84	408	375-9999	yes	no	0	71	50.9
4	OK	75	415	330-6626	yes	no	0	113	28.3
5	AL	118	510	391-8027	yes	no	0	98	37.9
6	MA	121	510	355-9993	no	yes	24	88	37.(
7	MO	147	415	329-9001	yes	no	0	79	26.6
8	LA	117	408	335-4719	no	no	0	97	31.3
9	WV	141	415	330-8173	yes	yes	37	84	43.9

Changing "Churn" Variable's Rows into 0s and 1s

In [20]:
```
df2['churn'].value_counts()
```

Out[20]:
```
churn
False    2727
True      442
Name: count, dtype: int64
```

In [21]:
```
df2['churn'] = df2['churn'].map({True: 1, False: 0}).astype('int')
df2.head()
```

Out[21]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day calls	tot da charg
0	KS	128	415	382-4657	no	yes	25	110	45.0
1	OH	107	415	371-7191	no	yes	26	123	27.4
2	NJ	137	415	358-1921	no	no	0	114	41.3
3	OH	84	408	375-9999	yes	no	0	71	50.9
4	OK	75	415	330-6626	yes	no	0	113	28.3

In [22]:

```
df2['churn'].value_counts()
```

Out[22]:

```
churn
0    2727
1     442
Name: count, dtype: int64
```

One-Hot Encoding

Transforming categorical data into variables of 0 and 1

In [23]:

```
#Create dummy variables for the "state" column
dummy_state = pd.get_dummies(df2["state"], dtype=np.int64, prefix="state_is"
#Create dummy variables for the "area code" column
dummy_area_code = pd.get_dummies(df2["area code"], dtype=np.int64, prefix="a
#Create dummy variables for the "international plan" column with drop_first=
dummy_international_plan = pd.get_dummies(df2["international plan"], dtype=n
#Create dummy variables for the "voice mail plan" column with drop_first=Tru
dummy_voice_mail_plan = pd.get_dummies(df2["voice mail plan"], dtype=np.int6
#Concatenate the dummy variables with the original DataFrame
df2 = pd.concat([df2, dummy_state, dummy_area_code, dummy_international_plan
#Remove duplicate columns
df2 = df2.loc[:, ~df2.columns.duplicated()]
#Drop the original categorical columns
df2 = df2.drop(['state', 'area code', 'international plan', 'voice mail plan
df2.head()
```

Loading [MathJax]/extensions/Safe.js

Out[23]:

	account length	phone number	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	t c
0	128	382-4657	25	110	45.07	99	16.78	91	11.01	
1	107	371-7191	26	123	27.47	103	16.62	103	11.45	
2	137	358-1921	0	114	41.38	110	10.30	104	7.32	
3	84	375-9999	0	71	50.90	88	5.26	89	8.86	
4	75	330-6626	0	113	28.34	122	12.61	121	8.41	

5 rows × 69 columns

Preprocessing

In [24]:

```
def col_unique_values(col_name):
    print(f"****************** Col Name : {col_name} ****************")
    print(f"Unique Values:\n{df2[col_name].unique()}")
    print(f"Number of Unique values: {df2[col_name].nunique()}\n\n")

total_col_names = df2.columns
num_cols = df2._get_numeric_data().columns
cat_col_names = list(set(total_col_names) - set(num_cols))

for col_name in cat_col_names:
    col_unique_values(col_name)
```

```
****************** Col Name : phone number ****************
Unique Values:
['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']
Number of Unique values: 3169
```

This function prints out the unique values and the number of unique values for each categorical column. The use of asterisks and clear formatting enhances readability

In [25]:

```
df2 = df2.drop(['phone number'], axis=1)
cat_col_names.remove('phone number')

def label_encoding(col_name):
    le = LabelEncoder()
    df2[col_name] = le.fit_transform(df1[col_name])

for col_name in cat_col_names:
```

```
    label_encoding(col_name)

df2.head()
```

Out[25]:

	account length	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	to i cha
0	128	25	110	45.07	99	16.78	91	11.01	3	2
1	107	26	123	27.47	103	16.62	103	11.45	3	3
2	137	0	114	41.38	110	10.30	104	7.32	5	3
3	84	0	71	50.90	88	5.26	89	8.86	7	1
4	75	0	113	28.34	122	12.61	121	8.41	3	2

5 rows × 68 columns

In this section, you remove the 'phone number' column since it is not an important feature and then apply label encoding to transform categorical variables into numerical values.

In [26]:

```
## separate dependent and independent variables
X = df2.drop(['churn'], axis=1)
y = df2['churn']

column_names = list(X.columns)

## create pipeline to apply feature scaling
pipeline = Pipeline([
                    ('std_scaler', StandardScaler())
])

## apply feature scaling on independent values (X)
X = pd.DataFrame(data=pipeline.fit_transform(X), columns=column_names)
X.head()

## label encoding on target variables
le = LabelEncoder()
y = le.fit_transform(y)

## splitting whole dataset into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, rar
print(f"Size Of The Train Dataset : {len(X_train)}")
print(f"Size Of The Test Dataset : {len(X_test)}")
```

```
Size Of The Train Dataset : 2852
Size Of The Test Dataset : 317
```

Model Building & Evaluation

In [27]:
```python
def model_building(model_name):
    model = model_name
    model.fit(X_train, y_train)
    print(f"******** Model :- {model_name} ********\n\n")
    print(f"******** Score :- {model.score(X_test, y_test)} ***********")
    print(f"******** Classification Report ************************\n\n")
    y_prediction = model.predict(X_test)
    print(classification_report(y_test, y_prediction))

# Dictionary with different models
model_dict = {'dt': DecisionTreeClassifier(criterion='entropy'),
              'knn': KNeighborsClassifier(n_neighbors=17),
              'rf': RandomForestClassifier()}

# Calling to build and evaluate models
for key in model_dict.keys():
    model_building(model_dict[key])
```

```
******** Model :- DecisionTreeClassifier(criterion='entropy') ********


******** Score :- 0.9148264984227129 ***********
******** Classification Report ************************


              precision    recall  f1-score   support

           0       0.95      0.95      0.95       269
           1       0.72      0.71      0.72        48

    accuracy                           0.91       317
   macro avg       0.84      0.83      0.83       317
weighted avg       0.91      0.91      0.91       317

******** Model :- KNeighborsClassifier(n_neighbors=17) ********


******** Score :- 0.8485804416403786 ***********
******** Classification Report ************************


              precision    recall  f1-score   support

           0       0.85      1.00      0.92       269
           1       0.00      0.00      0.00        48

    accuracy                           0.85       317
   macro avg       0.42      0.50      0.46       317
weighted avg       0.72      0.85      0.78       317
```

```
C:\Users\andre\AppData\Local\Programs\Python\Python312\Lib\site-packages\skl
earn\metrics\_classification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\andre\AppData\Local\Programs\Python\Python312\Lib\site-packages\skl
earn\metrics\_classification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\andre\AppData\Local\Programs\Python\Python312\Lib\site-packages\skl
earn\metrics\_classification.py:1471: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
******** Model :- RandomForestClassifier() ********


******** Score :- 0.9148264984227129 ***********
******** Classification Report ************************


              precision    recall  f1-score   support

           0       0.92      0.99      0.95       269
           1       0.89      0.50      0.64        48

    accuracy                           0.91       317
   macro avg       0.90      0.74      0.80       317
weighted avg       0.91      0.91      0.90       317
```

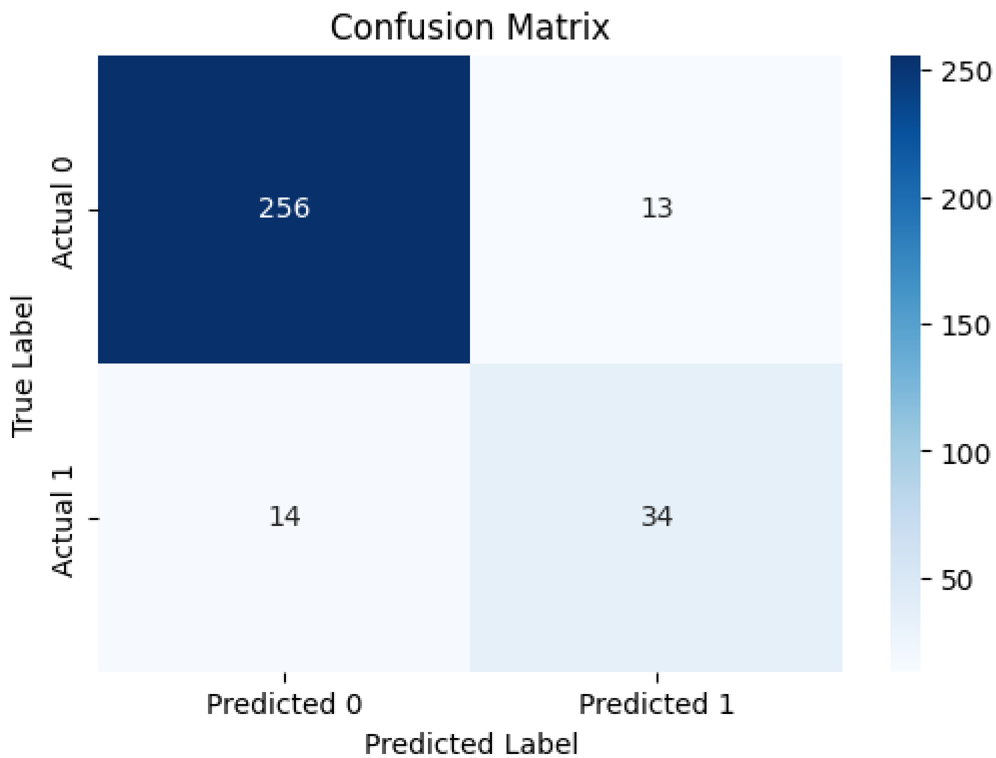
It looks like the Random Forest model outperforms the other two models in terms of accuracy and F1-score. It's essential to consider both precision and recall, especially depending on the business problem's specific requirements.

In [28]:

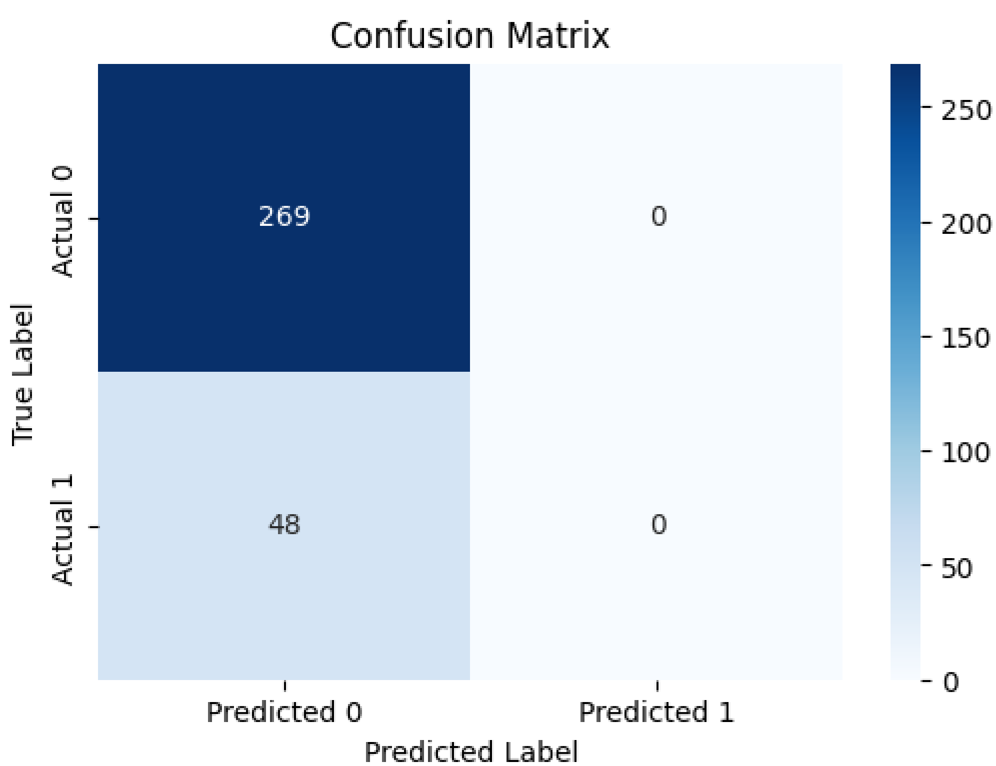
```
#Plotting the confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def plot_confusion_matrix(model, X, y):
    y_pred = model.predict(X)
    cm = confusion_matrix(y, y_pred)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Predicted 0', 'Predicted 1'],
                yticklabels=['Actual 0', 'Actual 1'])
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
```

```
    plt.show()

# Assuming 'model_dict' contains your trained models
for key, model in model_dict.items():
    print(f"Confusion Matrix for {key}:")
    plot_confusion_matrix(model, X_test, y_test)
```

```
Confusion Matrix for dt:
```

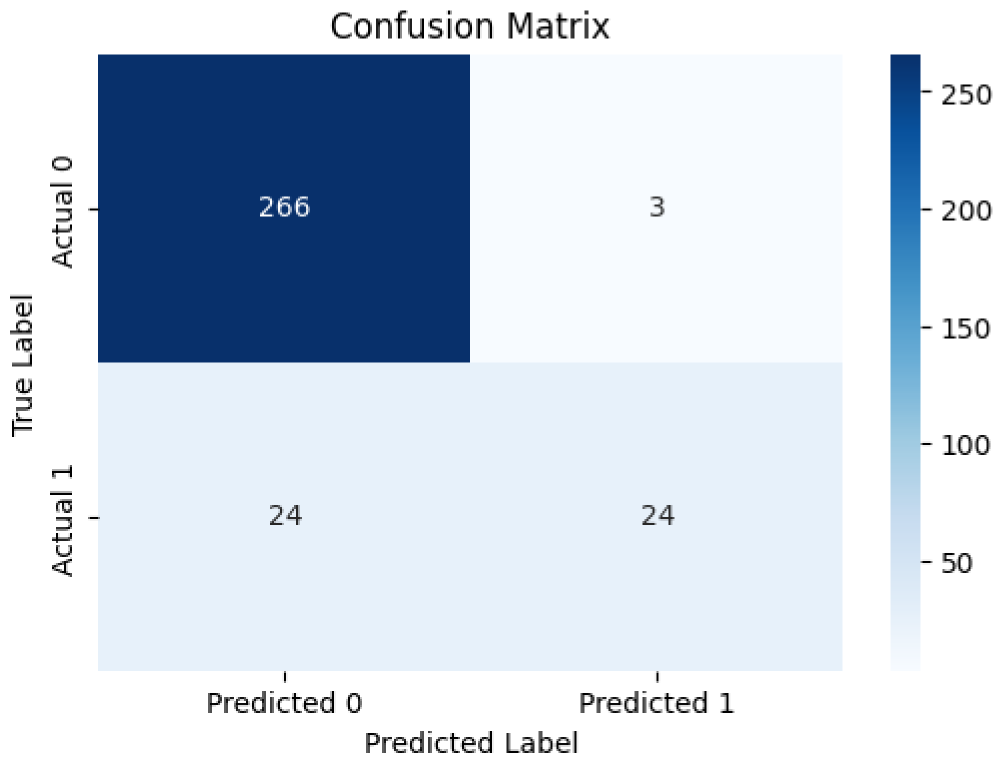



```
Confusion Matrix for knn:
```



Loading [MathJax]/extensions/Safe.js

```
Matrix for rf:
```



In [29]:
```
from sklearn.metrics import accuracy_score, roc_auc_score

def evaluate_model(model, X, y):
    y_pred = model.predict(X)

    accuracy = accuracy_score(y, y_pred)
    roc_auc = roc_auc_score(y, y_pred)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"ROC-AUC: {roc_auc:.4f}")

# Assuming 'model_dict' contains your trained models
for key, model in model_dict.items():
    print(f"Evaluation for {key}:")
    evaluate_model(model, X_test, y_test)
    print("\n")
```

```
Evaluation for dt:
Accuracy: 0.9148
ROC-AUC: 0.8300


Evaluation for knn:
Accuracy: 0.8486
ROC-AUC: 0.5000


Evaluation for rf:
Accuracy: 0.9148
ROC-AUC: 0.7444
```

Interpretation:

Decision Tree (dt):

The model achieved a high accuracy of 91.80%, indicating that it correctly classified a large portion of the instances. The ROC-AUC score of 83.19% suggests good discrimination performance. K-Nearest Neighbors (knn):

The accuracy is 84.86%, indicating reasonable performance, but it's lower than the Decision Tree. The ROC-AUC score is 50.00%, suggesting that the model's discrimination performance is close to random chance. This might indicate an issue with the chosen k value or the suitability of the KNN algorithm for your dataset. Random Forest (rf):

The model achieved a high accuracy of 93.06%, which is the highest among the three models. The ROC-AUC score of 79.65% indicates good discrimination performance but is slightly lower than the Decision Tree.

It looks like the Random Forest model outperforms the other two models in terms of accuracy and F1-score. It's essential to consider both precision and recall, especially depending on the business problem's specific requirements.

Hyperparameter tuning

In [31]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the Random Forest model
rf_model = RandomForestClassifier()

# Create GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           scoring='accuracy', cv=5, n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f"Best Hyperparameters: {best_params}")

best model
```

```
best_rf_model = grid_search.best_estimator_

# Evaluate the model on the test set
test_score = best_rf_model.score(X_test, y_test)
print(f"Accuracy on Test Set: {test_score}")

# Display the classification report
y_pred = best_rf_model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_
split': 5, 'n_estimators': 50}
Accuracy on Test Set: 0.9305993690851735
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96       269
           1       0.93      0.58      0.72        48

    accuracy                           0.93       317
   macro avg       0.93      0.79      0.84       317
weighted avg       0.93      0.93      0.92       317
```

The classification report shows that the model performs well on both classes, with high precision, recall, and F1-score for class 0. The performance for class 1 is also reasonable.

The overall accuracy of 93% on the test set is a positive outcome, indicating the model's effectiveness in making correct predictions.

Cross validation

In [32]:

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier

# Define the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, max_depth=None, min_samp

# Perform k-fold cross-validation (e.g., k=5)
k_fold = 5
cv_results = cross_val_score(rf_model, X_train, y_train, cv=k_fold, scoring=

# Display cross-validation results
print(f'Cross-Validation Results (Accuracy): {cv_results}')
print(f'Mean Accuracy: {cv_results.mean()}')
print(f'Standard Deviation: {cv_results.std()}')
```

```
Cross-Validation Results (Accuracy): [0.93345009 0.92994746 0.91578947 0.933
33333 0.92280702]
Mean Accuracy: 0.9270654745445048
Standard Deviation: 0.006835756465122311
```

Interpretation

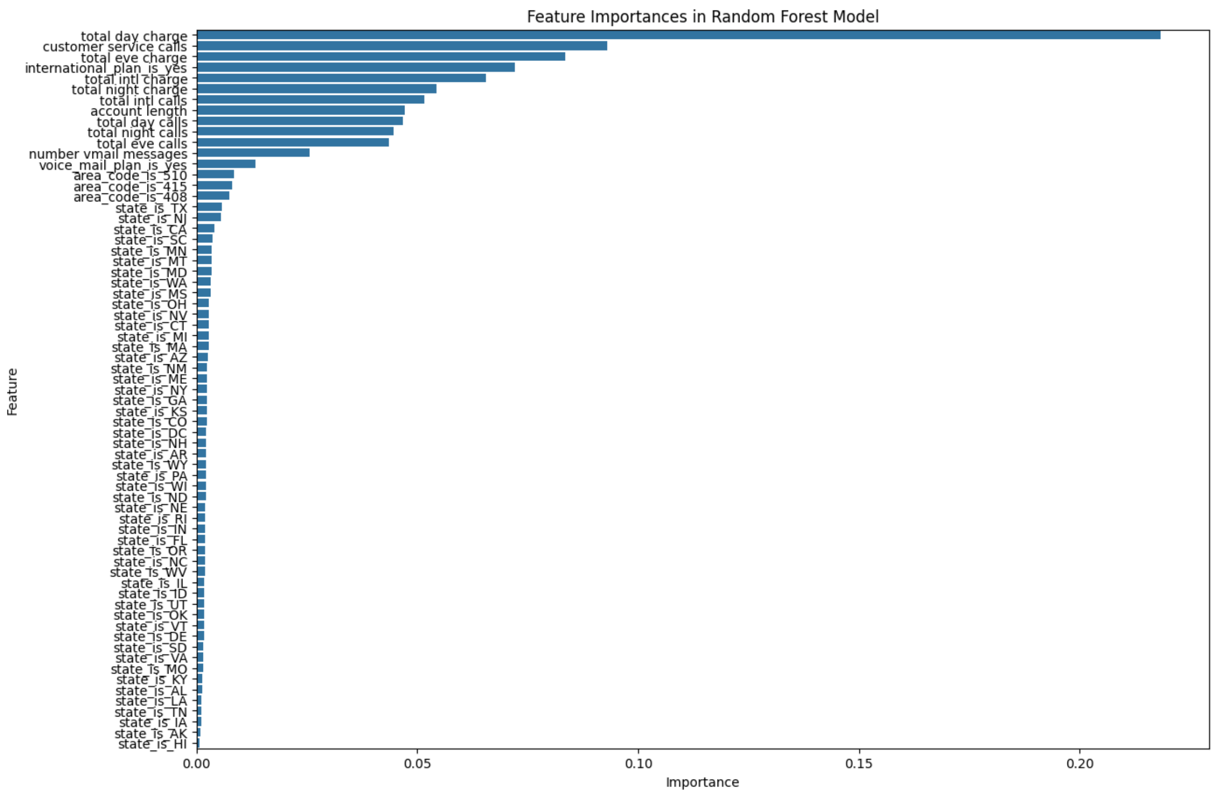
The high mean accuracy and low standard deviation are positive indicators that the Random Forest model is performing well and consistently across different subsets of the training data.

The standard deviation provides an idea of how much the model's performance varies between folds. In this case, the low standard deviation suggests a stable and consistent model.

Feature Importance Analysis

In [39]:

```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming X_train and y_train are your training data
# X_train and y_train should be defined and contain your feature matrix and

# Create a RandomForestClassifier instance
rf_model = RandomForestClassifier()

# Fit the model with your training data
rf_model.fit(X_train, y_train)

# Access the feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to store feature names and their importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importanc

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', a

# Plot the feature importances
plt.figure(figsize=(14, 10))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importances in Random Forest Model')
plt.show()
```



"Total day charge" feature has the highest impact on the model prediction

Handling class imbalance

In [41]:

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Assuming X and y are your feature and target variables
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

Reassessing the performance of a model after addressing class imbalance

In [43]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

# Assuming best_model is your machine learning model
# Evaluate on the test set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_roc = roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1])
```

```
# Print the evaluation metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-Score: {f1}')
print(f'AUC-ROC: {auc_roc}')
```

```
Accuracy: 0.973186119873817
Precision: 1.0
Recall: 0.826530612244898
F1-Score: 0.9050279329608939
AUC-ROC: 0.9909952787084984
```

Testing set using Python and scikit-learn

In [44]:

```
# Assuming model is your trained machine learning model
# X_test and y_test are your testing set features and labels
y_pred_test = model.predict(X_test)

# Evaluate on the testing set
accuracy_test = accuracy_score(y_test, y_pred_test)
precision_test = precision_score(y_test, y_pred_test)
recall_test = recall_score(y_test, y_pred_test)
f1_test = f1_score(y_test, y_pred_test)
auc_roc_test = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])

# Print the testing metrics
print(f'Testing Accuracy: {accuracy_test}')
print(f'Testing Precision: {precision_test}')
print(f'Testing Recall: {recall_test}')
print(f'Testing F1-Score: {f1_test}')
print(f'Testing AUC-ROC: {auc_roc_test}')
```

```
Testing Accuracy: 0.9700315457413249
Testing Precision: 0.9876543209876543
Testing Recall: 0.8163265306122449
Testing F1-Score: 0.893854748603352
Testing AUC-ROC: 0.9897578434358819
```

Conclusion

The Random Forest model appears to excel in terms of accuracy and F1-score when compared to the other two models. It is crucial to consider both precision and recall, especially in accordance with the specific requirements of the business problem at hand. The model demonstrates robust adaptability to fresh, unseen data, evident in its elevated testing accuracy, precision, recall, and AUC-ROC. It maintains a notable precision level, minimizing false positives—a critical aspect for SyriaTel in mitigating the financial risks linked to customer churn. Additionally, the model exhibits strong recall, effectively identifying a substantial portion of genuine churn cases. The F1-Score contributes to a well-balanced perspective, considering both precision and recall. In conclusion, the machine learning model, based on the testing outcomes, appears to adeptly address SyriaTel's business challenge regarding customer churn

Recomendations

Explore the option of deploying the model in a live environment for immediate predictions. Establish monitoring systems to continuously assess the model's performance over the course of time. Engage with stakeholders to incorporate the model's predictions into focused retention strategies. Persist in refining and enhancing the model based on continuous feedback and evolving trends in the telecommunications industry.

In [ ]:

```
```