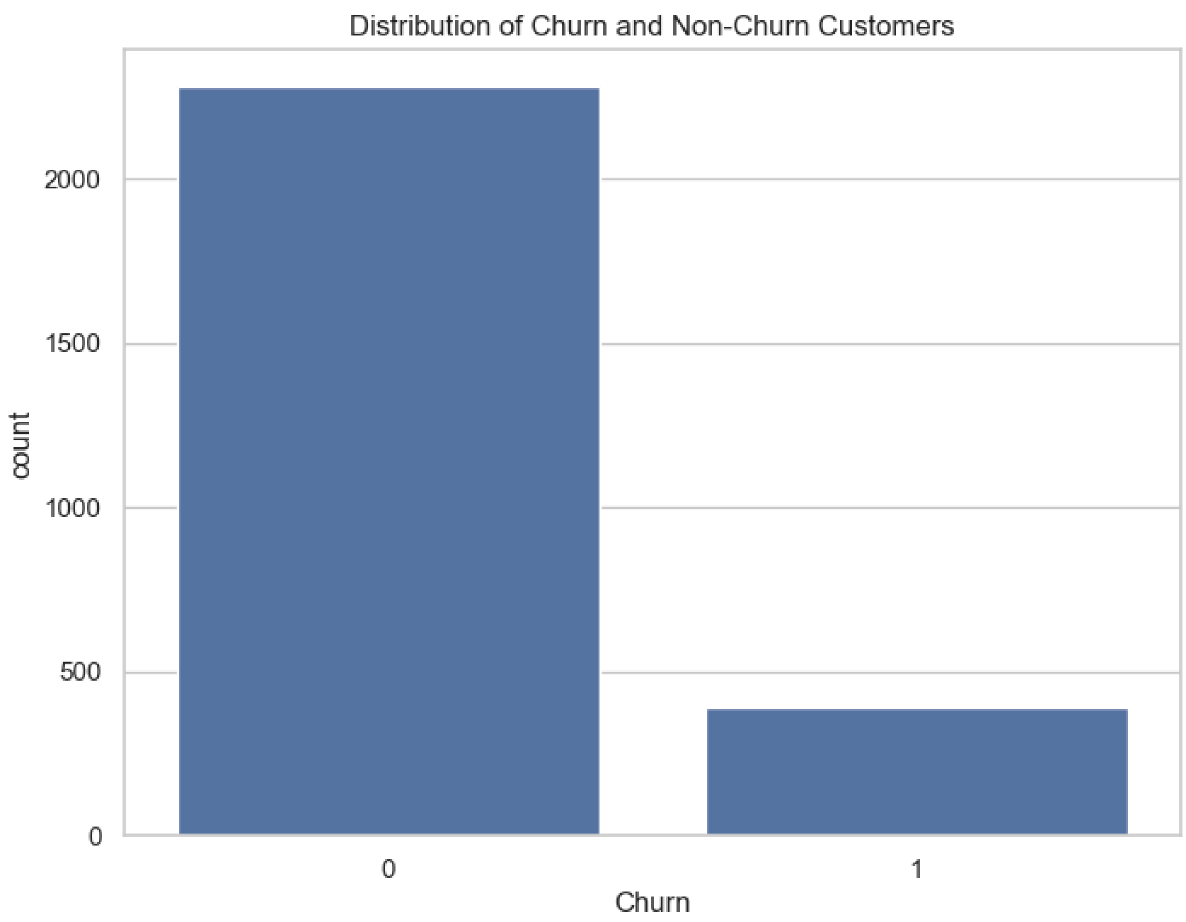
a. **What is the significance of Churn Rate for stakeholders (Customers, MCI, etc.)?**

- **Churn Rate** represents the percentage of customers who stop using a service during a given time period. For stakeholders, this metric is crucial:
  - **Customers**: A high churn rate might indicate dissatisfaction with the service, leading to poor customer experience and potentially damaging the company's reputation.
  - **Management and Company Investors (MCI)**: Churn rate directly affects revenue and profitability. A high churn rate can signal underlying issues within the service or product, prompting the need for strategic changes to improve customer retention and financial performance.

b. **What are the characteristics of each Type of Customer (Churn or Not Churn)?**

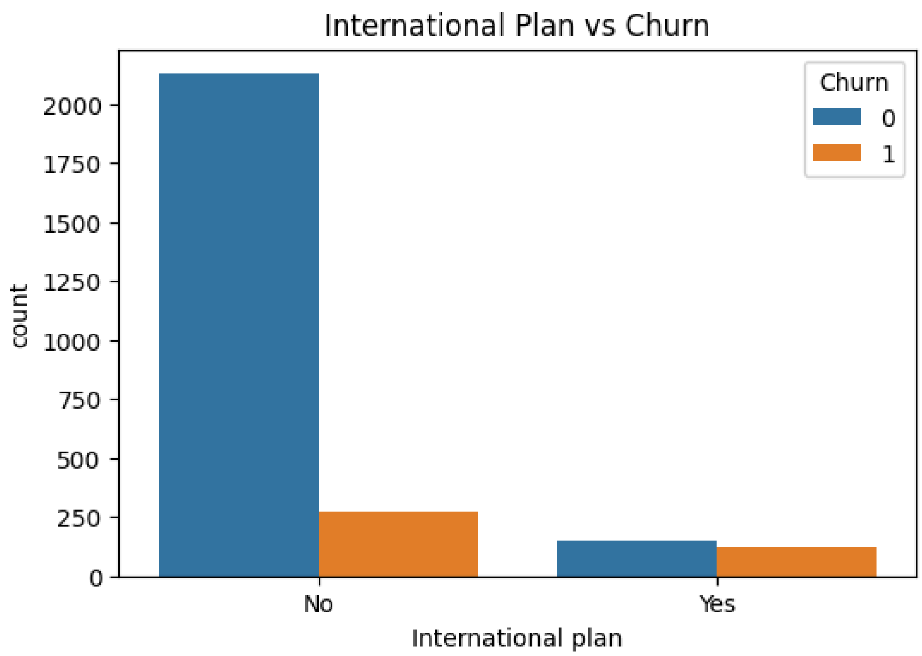
Churn Distribution:



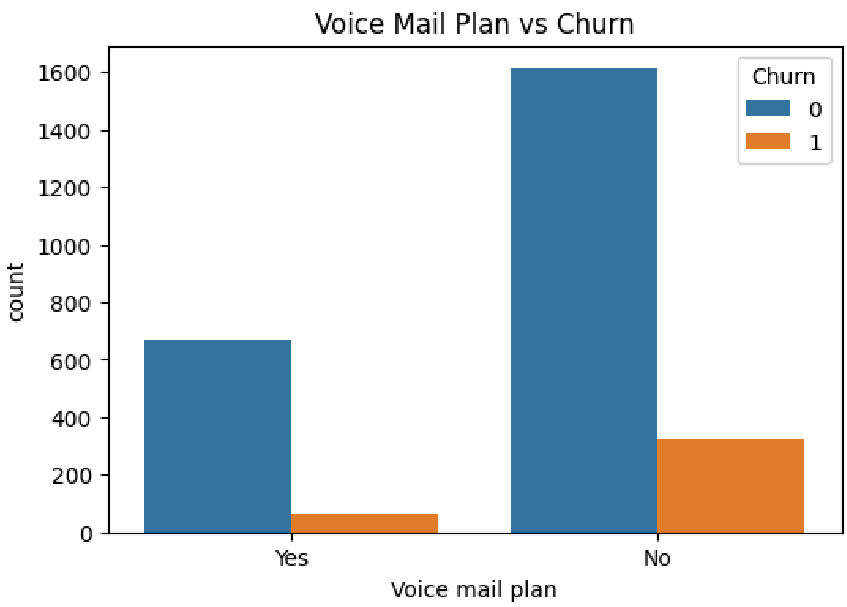
- The majority of customers have not churned, but a significant proportion have.

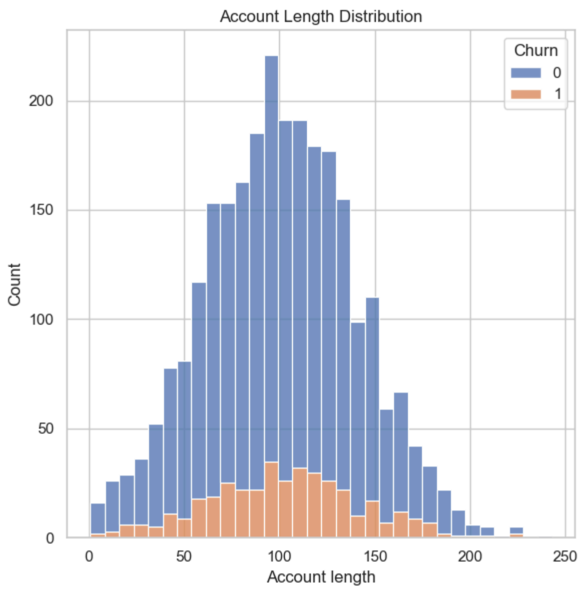
Characteristics of Churn vs Non-Churn Customers:

- International Plan: Most customers with International Plan do not churn

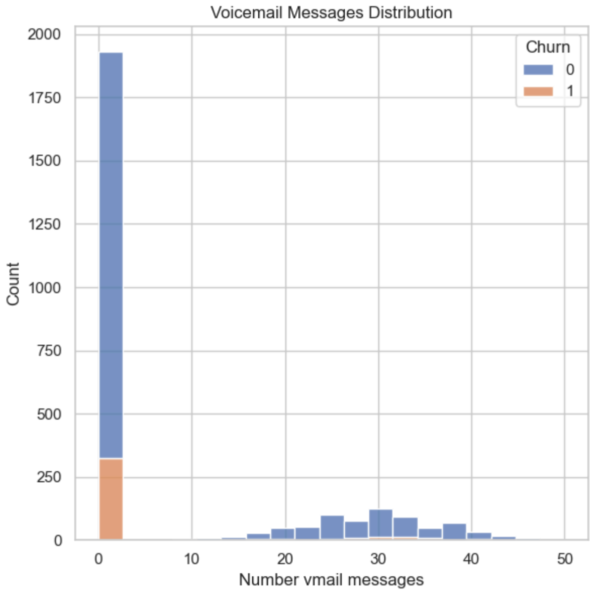


- Voice Mail Plan: For the most part, customers with Voice mail plans do not churn

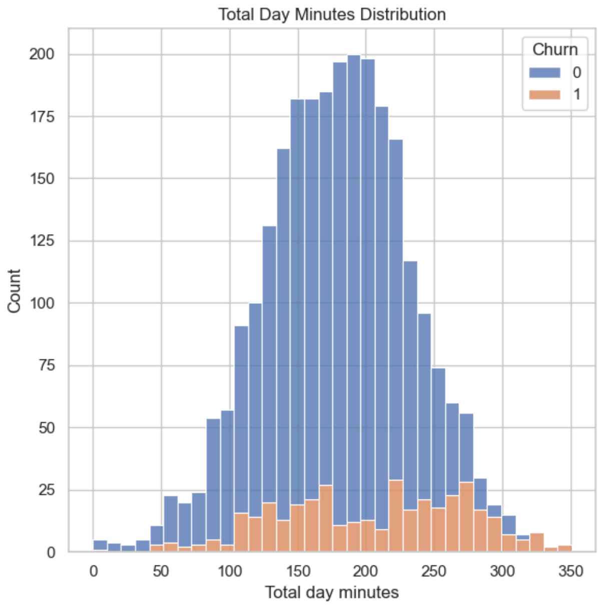




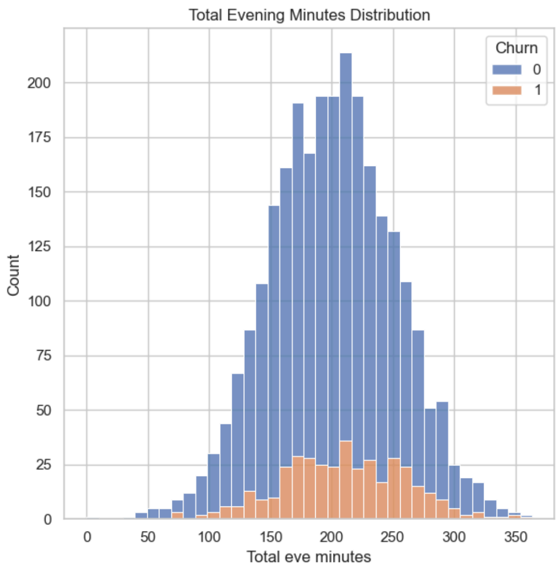
- **Account Length**: There's no clear difference in account length between churned and non-churned customers.



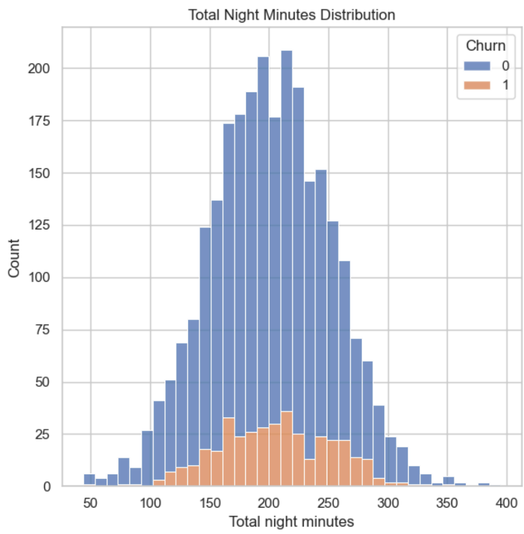
- **Voicemail Messages**: Non-churn customers tend to use voicemail services more than churn customers.



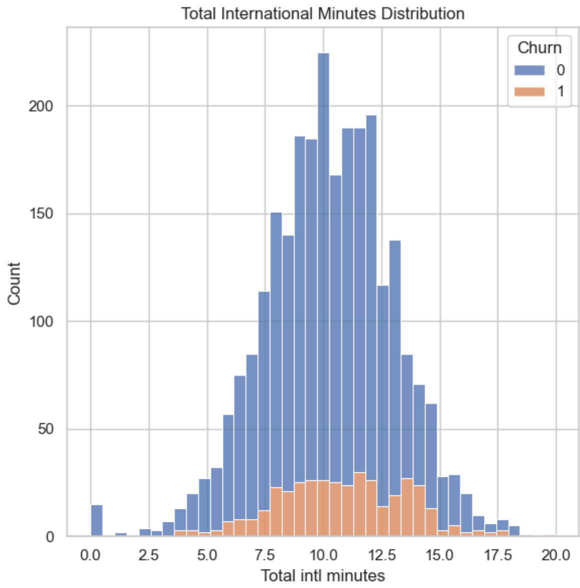
- **Total Day Minutes**: Churn customers often have higher total day minutes.



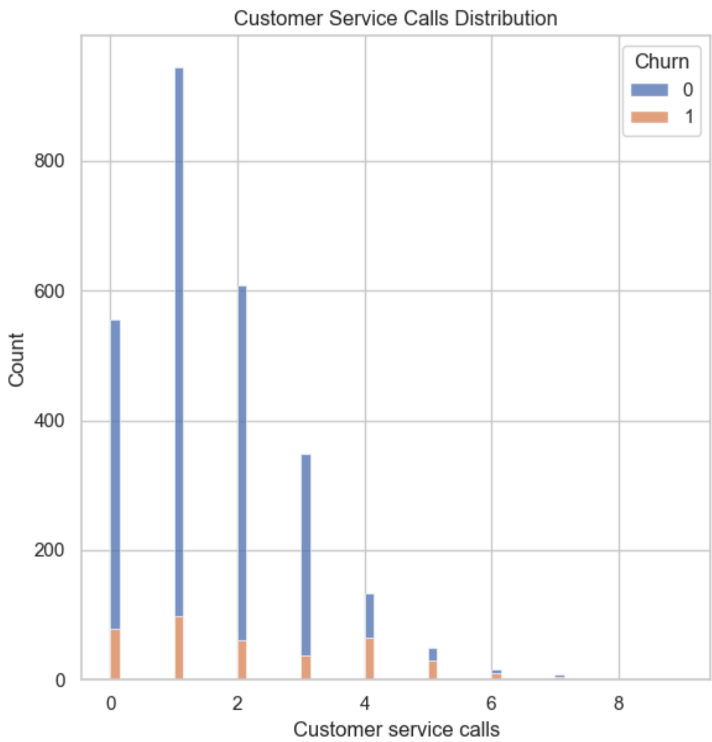
- **Total Evening Minutes**: Similar usage patterns between churn and non-churn customers.



- **Total Night Minutes**: Similar usage patterns between churn and non-churn customers.



- **Total International Minutes**: Slightly higher international minutes for churn customers.



- **Customer Service Calls**: Churn customers have made more customer service calls, indicating potential issues or dissatisfaction.

c. **Which ML modeling can be implemented and represent model results? including features input and explaining features important.**

For predicting customer churn, we can use various machine learning models such as Decision Trees, Random Forest, Logistic Regression, or Gradient Boosting. In this example, we will use a Decision Tree classifier and explain the importance of features used in the model.

Model Training

Model Implementation and Features:

1. Data Preparation:
- The dataset is split into features (inputs) and labels (churn status).
- We divide the data into training and testing sets to evaluate the model's performance.
2. Decision Tree Classifier:
- A Decision Tree model is trained on the data. This model uses decision rules based on feature values to predict whether a customer will churn.
3. Tree Structure and Feature Importance:
- The trained Decision Tree's structure reveals the decision rules it uses to classify churn vs. non-churn customers.
- Feature importance is calculated to identify which features most influence the model's predictions. Key features might include the number of customer service calls, total day minutes, and use of voicemail services.

```
Tree structure:
|--- Total day minutes <= 264.40
|   |--- Customer service calls <= 3.50
|   |   |--- class: 0
|   |--- Customer service calls >  3.50
|   |   |--- class: 0
|--- Total day minutes >  264.40
|   |--- Number vmail messages <= 6.50
|   |   |--- class: 1
|   |--- Number vmail messages >  6.50
|   |   |--- class: 0
```

- DecisionTreeModel classifier of depth 2 with 7 nodes

Model Evaluation

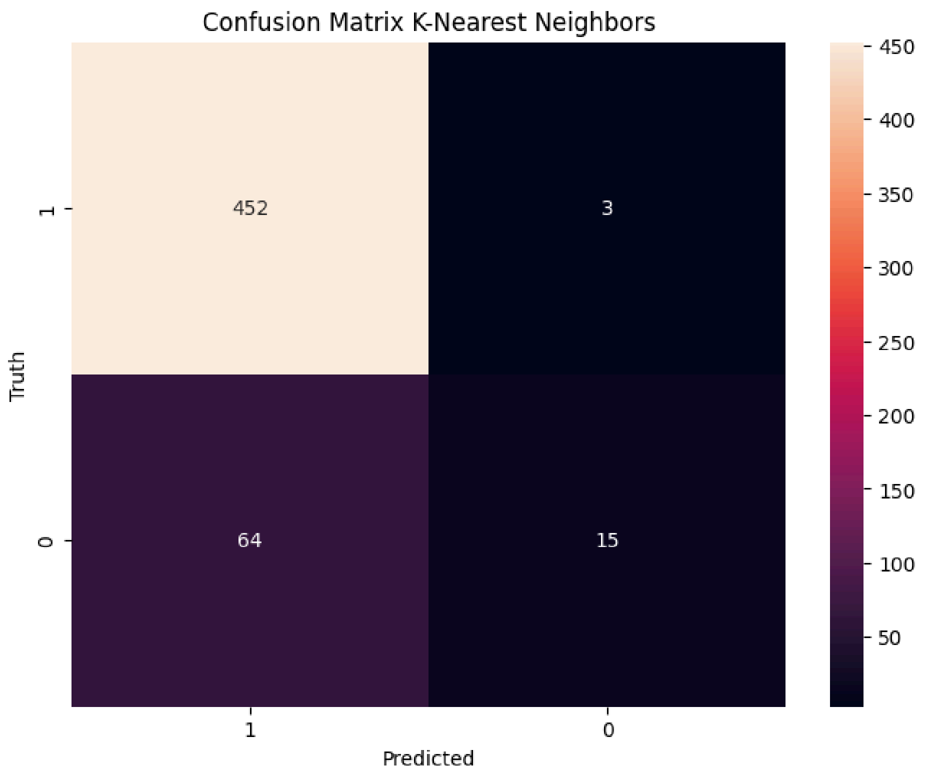
Results of model

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       455
           1       0.83      0.19      0.31        79

    accuracy                           0.87       534
   macro avg       0.85      0.59      0.62       534
weighted avg       0.87      0.87      0.84       534
```

- confusion matrix:



The overall accuracy, ie F-1 score, seems quite good, but one troubling issue is the discrepancy between the recall measures. The recall (aka sensitivity) for the Churn=False samples is high, while the recall for the Churn=True examples is relatively low. Business decisions made using these predictions will be used to retain the customers most likely to leave, not those who are likely to stay. Thus, we need to ensure that our model is sensitive to the Churn=True samples.

Perhaps the model's sensitivity bias toward Churn=False samples is due to a skewed distribution of the two types of samples. Let's try grouping the data by the Churn field and counting the number of instances in each group.

	Churn	count
0	0	2278
1	1	388

Stratified Sampling

There are roughly 6 times as many False churn samples as True churn samples. We can put the two sample types on the same footing using stratified sampling.

Here we're keeping all instances of the Churn=True class, but downsampling the Churn=False class to a fraction of 388/2278.

	Churn	count
0	0	388
1	1	388

Let's build a new model using the evenly distributed data set and see how it performs.

```
              precision    recall  f1-score   support

           0       0.65      0.85      0.73        78
           1       0.78      0.54      0.64        78

    accuracy                           0.69       156
   macro avg       0.71      0.69      0.68       156
weighted avg       0.71      0.69      0.68       156

Confusion Matrix:
 [[66 12]
 [36 42]]
```

- With these new recall values, we can see that the stratified data was helpful in building a less biased model, which will ultimately provide more generalized and robust predictions.

Model Selection

Given the data set at hand, we would like to determine which parameter values of the decision tree produce the best model. We need a systematic approach to quantitatively measure the performance of the models and ensure that the results are reliable. This task of model selection is often done using cross validation techniques. A common technique is k-fold cross validation, where the data is randomly split into k partitions. Each partition is used once as the testing data set, while the rest are used for training. Models are then generated using the training sets and evaluated with the testing sets, resulting in k model performance measurements. The average of the performance scores is often taken to be the overall score of the model, given its build parameters.

For model selection we can search through the model parameters, comparing their cross validation performances. The model parameters leading to the highest performance metric produce the best model.

The ML package supports k-fold cross validation, which can be readily coupled with a parameter grid builder and an evaluator to construct a model selection workflow. Below, we'll use a transformation/estimation pipeline to train our models. The cross validator will use the ParamGridBuilder to iterate through the maxDepth parameter of the decision tree and evaluate the models using the F1-score, repeating 3 times per parameter value for reliable results.

```
DecisionTreeClassifier(max_depth=5)
F1 Score on Test Data: 0.8461538461538461
```

We find that the best tree model produced using the cross-validation process is one with a depth of 5. So we can assume that our initial "shallow" tree of depth 2 in the previous section was not complex enough, while trees of depth higher than 5 overfit the data and will not perform well in practice.

Predictions and Model Evaluation

The actual performance of the model can be determined using the final_test_data set which has not been used for any training or cross-validation activities. We'll transform the test set with the model pipeline, which will map the labels and features according to the same recipe. The evaluator will provide us with the F-1 score of the predictions, and then we'll print them along with their probabilities.

```
Accuracy: 0.881559220389805
F1 Score: 0.8901754289442033
     Actual  Predicted  Probability_0  Probability_1
0         0          0       0.900524       0.099476
1         1          1       0.000000       1.000000
2         1          1       0.125000       0.875000
3         0          0       0.900524       0.099476
4         0          0       0.900524       0.099476
..      ...        ...            ...            ...
662       0          0       0.900524       0.099476
663       0          0       0.900524       0.099476
664       0          0       0.761905       0.238095
665       0          0       0.900524       0.099476
666       0          0       0.928571       0.071429

[667 rows x 4 columns]
```

The prediction probabilities can be very useful in ranking customers by their likeliness to defect. This way, the limited resources available to the business for retention can be focused on the appropriate customers.

d. What actions regarding qualitative and quantitative analytics could be implemented to enhance retention rate?

After training the model, Printing the feature importance values to understand which features are the most important predictors of churn.

```
Feature importances:
Total day charge          0.193050
International plan        0.140970
Total intl minutes        0.135492
Customer service calls    0.126121
Total eve charge          0.115524
Total intl calls          0.111425
Total day minutes         0.082277
Number vmail messages     0.044519
Total eve minutes         0.020612
Total night charge        0.013997
Total day calls           0.008711
Total eve calls           0.007302
Total night calls         0.000000
Total night minutes       0.000000
Voice mail plan           0.000000
Total intl charge         0.000000
Account length            0.000000
dtype: float64
```

1. Total Day Charge:

- Insight: This is the most significant factor influencing churn. Customers with high day charges tend to leave the service.
- Action:
    - Review pricing strategies and offer discounts or special packages to high-usage customers.
    - Provide unlimited plans to reduce the feeling of being overcharged.

2. International Plan:

- Insight: Customers with an international plan tend to have a lower churn rate.
- Action:
    - Promote and encourage customers to use international plans through promotional offers or discounts.
    - Improve and expand international services to attract more customers.

3. Total International Minutes:

- Insight: Customers with higher international minutes tend to have a lower churn rate.
- Action:
    - Offer more favorable international packages.
    - Monitor and build relationships with customers using international services to increase satisfaction.

4. Customer Service Calls:

- Insight: Customers with more customer service calls tend to churn.
- Action:

- Improve customer service quality by training staff and enhancing issue resolution processes.
- Monitor customers with frequent service calls and proactively address their issues before they decide to leave.

5. Total Evening Charge:

- Insight: Customers with high evening charges may also feel overcharged and want to leave.
- Action:
    - Offer special packages for evening calls.
    - Create promotional programs specifically for evening calls to reduce costs for customers.

Specific Action Plan

1. Pricing Strategy:
    - Redesign pricing plans to better match customer usage needs. Consider implementing more affordable bundled packages.
2. Promotional Programs:
    - Implement promotional programs for customers with high international usage or high charges during the day and evening.
3. Improve Customer Service Quality:
    - Retrain staff, improve complaint resolution processes, and closely monitor customers with frequent service calls.
4. Build Good Relationships with Customers:
    - Proactively reach out to customers showing signs of wanting to leave to understand their issues and offer suitable solutions.

CODE

```
# Import Libraries:

import pandas as pd

import numpy as np

import seaborn as sbn

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn import tree

from sklearn.metrics import precision_score, recall_score, f1_score,
confusion_matrix, classification_report

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.metrics import make_scorer, f1_score, accuracy_score

import warnings

warnings.filterwarnings('ignore')

# Load Data:

data = pd.read_csv('Copy of churn-bigml-80.csv')

final_test_data = pd.read_csv('Copy of churn-bigml-20.csv')

# Map Binary Values:

binary_map = {'Yes':1.0, 'No':0.0, 'True':1.0, 'False':0.0}

columns_to_convert = ['International plan', 'Voice mail plan']

data[columns_to_convert] = data[columns_to_convert].apply((lambda col:
col.map(binary_map)))

final_test_data[columns_to_convert] =
final_test_data[columns_to_convert].apply((lambda col: col.map(binary_map)))

# Convert Churn Column to Integer:

data['Churn'] = data['Churn'].astype("int64")

final_test_data['Churn'] = final_test_data['Churn'].astype("int64")
```

```python
# Label Data Function:

def label_data(data):

   # label: row[end], features: row[0:end-1]

   X = data.iloc[:, :-1]

   y = data.iloc[:, -1]

   return X, y



X, y = label_data(data)

# Split Data:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Decision Tree Classifier:

model_tree = DecisionTreeClassifier(max_depth=2, criterion='gini')

model_tree.fit(X_train, y_train)

# Print the tree structure

tree_structure = tree.export_text(model_tree, feature_names=list(X.columns))

print("Tree structure:")

print(tree_structure)

# Make predictions on the test data

y_pred = model_tree.predict(X_test)

# Print metrics

print(classification_report(y_test, y_pred))

# Perform stratified sampling

fractions = {0: 388. / 2278, 1: 1.0}

stratified_data = data.groupby('Churn').apply(lambda x:
x.sample(frac=fractions[x.name], random_state=42)).reset_index(drop=True)



# Group by 'Churn' and count the occurrences

churn_counts = stratified_data.groupby('Churn').size().reset_index(name='count')

print(churn_counts)
```

```python
# Split the data into features and labels

X1, y1 = label_data(stratified_data)

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2,
random_state=42, stratify=y1)

# Train a Decision Tree Classifier

tree_model = DecisionTreeClassifier(max_depth=2, criterion='gini')

tree_model.fit(X_train, y_train)

# Make predictions on the test data

y_pred_1 = tree_model.predict(X_test)

# Print metrics

print(classification_report(y_test, y_pred_1))

print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred_1))

# Encode the labels

label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(y1)

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X1, y_encoded, test_size=0.2,
random_state=42, stratify=y_encoded)

# Create a pipeline with a scaler, vectorizer, and decision tree classifier

pipeline = Pipeline([

   ('scaler', StandardScaler()),

   ('classifier', DecisionTreeClassifier())

])

# Define the parameter grid

param_grid = {

   'classifier__max_depth': [2, 3, 4, 5, 6, 7]

}

# Set F1 score as evaluation metric for best model selection

scorer = make_scorer(f1_score, average='weighted')
```

```python
# Set up grid search

grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,
scoring=scorer, cv=3)

# Fit the model

grid_search.fit(X_train, y_train)

# Fetch the best model

best_model = grid_search.best_estimator_.named_steps['classifier']

print(best_model)

# Evaluate the model on the test data

y_pred = grid_search.predict(X_test)

f1 = f1_score(y_test, y_pred, average='weighted')

print(f'F1 Score on Test Data: {f1}')

# Split the final test data into features and labels

X_final_test, y_final_test = label_data(final_test_data)

# Encode the labels using the same label encoder

y_final_test_encoded = label_encoder.transform(y_final_test)

# Use the trained pipeline to transform the test data and make predictions

pipeline = grid_search.best_estimator_

# Make predictions on the final test data

y_final_test_pred = pipeline.predict(X_final_test)

# Evaluate the model

accuracy = accuracy_score(y_final_test_encoded, y_final_test_pred)

f1 = f1_score(y_final_test_encoded, y_final_test_pred, average='weighted')

print('Accuracy:', accuracy)

print('F1 Score:', f1)

# Create a DataFrame with predictions and probabilities

probabilities = pipeline.predict_proba(X_final_test)

predictions_df = pd.DataFrame({

   'Actual': y_final_test_encoded,

   'Predicted': y_final_test_pred,
```

```
    'Probability_0': probabilities[:, 0],

    'Probability_1': probabilities[:, 1]

})

print(predictions_df)
```