Final project submission

Please fill out:

- Student Name: Esther Terry Munene
- Student Pace: Full time hybrid
- Instructor Name: Maryanne Mwikali
- Due Date: 7/06/2024

Syriatel Telcommunications Churn Rate Analysis

Syriatel is a telecommunications company operating in Syria, and they are currently working on reducing their churn rate. Churn rate refers to the number of customers who unsubscribe or discontinue a specific service over a given period of time. Syriatel aims to focus on improving their phone service sector while keeping the internet services separate for now.

The goals of this study are as follows:

Develop a Model to Minimize Churn: Create a predictive model that accurately identifies customers at risk of churning without them being aware of SyriaTel's intention to retain them.

Identify Key Factors: Determine the key factors and patterns that contribute to a higher churn rate.

By achieving these goals, Syriatel will be able to segment their customer base and implement targeted strategies to improve user retention.

The analysis involves the use of four different classification models: Naive Bayes, Decision Tree Classifier, Random Forest, and Logistic Regression. The notebook is organized into the following sections:

- Exploratory Data Analysis (EDA)
- Baseline Modeling
- Optimization of three Classification Models
- Performance Evaluation
- Findings and Feature Importance of the winning model

In the EDA section, various graphs and visualizations are presented to provide insights.

Data Preparation

In [2]:
```
# import neccesary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Data Preprocessing

from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, LabelBinarizer, OneHotEncode

# Model Evaluation Metrics

from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, recall_
from sklearn.metrics import roc_curve, confusion_matrix, precision_score

# Data Splitting

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, StratifiedKFold

# Machine Learning Algorithms
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

Load data

In [3]:
```
# importing initial dataset
data = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
```

Data Exploration

In [4]:
```
# Displaying DataFrame information
data.info
```

Out[4]:
```
<bound method DataFrame.info of       state  account length  area code phon
e 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]>
```

In [6]:
```
# Displaying the first few rows of the dataset
data.head()
```

Out[6]:

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

5 rows × 21 columns

In [7]:
```
# Checking the dimensions of the dataset
data.shape
```

Out[7]: (3333, 21)

In [8]:
```
# Checking the data types of the columns
data.dtypes
```

Out[8]:
```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

In [9]:
```
# Checking for missing values
data.isnull().sum()
```

Out[9]:
```
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 [10]:
```
# Checking if customers have single number
data['phone number'].nunique()
```

Out[10]:
```
3333
```

Data Cleaning and EDA

In [11]:
```
def clean_data(df):
    """
    Cleans the given DataFrame.

    Parameters:
    - df: DataFrame

    Returns:
    - None
    """
    # Drop the 'phone number', 'area code', and 'state' columns as they are
    df.drop(['phone number', 'area code', 'state'], axis=1, inplace=True)

    # Convert non-numerical columns to categorical
    categorical_cols = ['international plan', 'voice mail plan']
    label_encoder = preprocessing.LabelEncoder()
    for col in categorical_cols:
        df[col] = label_encoder.fit_transform(df[col])

clean_data(data)
```

In [12]:
```
# Statistical summary of the dataset
data.describe()
```

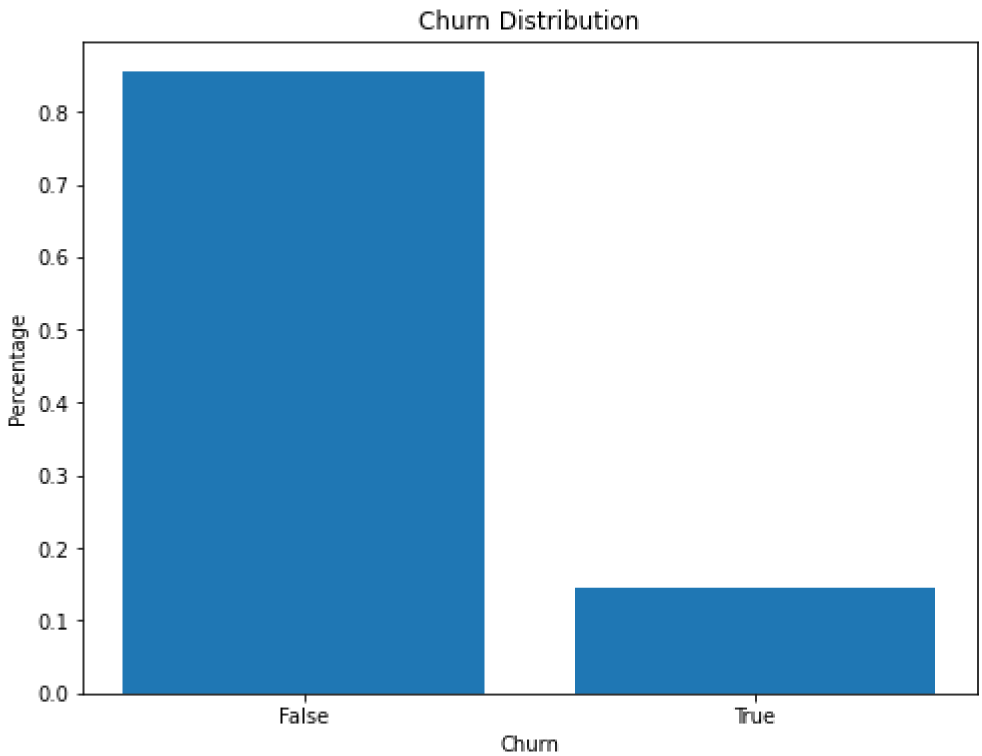
Out[12]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tot c
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0
mean	101.064806	0.096910	0.276628	8.099010	179.775098	100.435644	30.5
std	39.822106	0.295879	0.447398	13.688365	54.467389	20.069084	9.2
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	74.000000	0.000000	0.000000	0.000000	143.700000	87.000000	24.4
50%	101.000000	0.000000	0.000000	0.000000	179.400000	101.000000	30.5
75%	127.000000	0.000000	1.000000	20.000000	216.400000	114.000000	36.7
max	243.000000	1.000000	1.000000	51.000000	350.800000	165.000000	59.6

In [14]:
```
# Analyzing the target variable - 'churn'
data['churn'].value_counts(normalize=True)
```

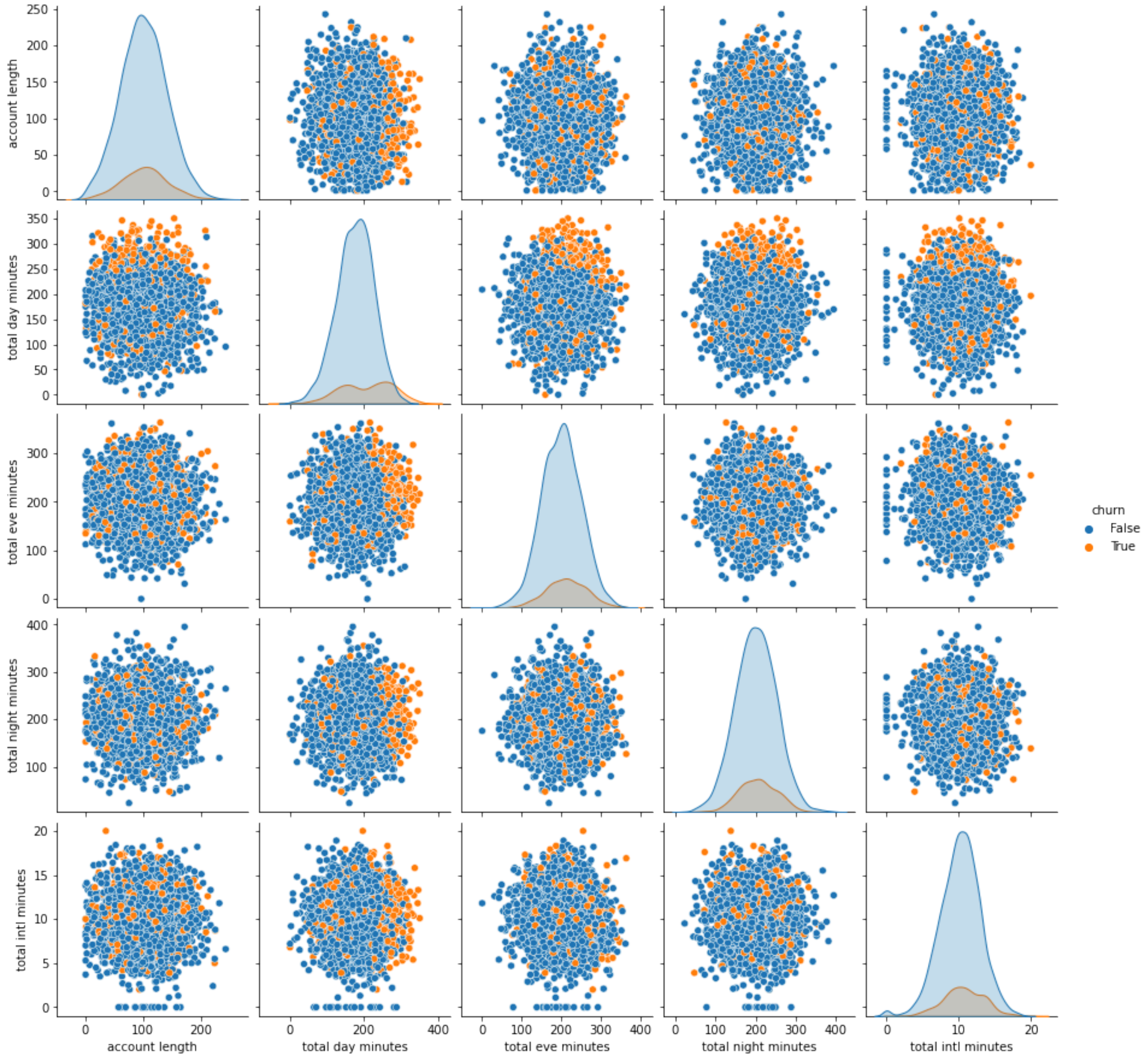
Out[14]:
```
False    0.855086
True     0.144914
Name: churn, dtype: float64
```

In [16]:
```
# Analyzing the target variable - 'churn'
churn_counts = data['churn'].value_counts(normalize=True)

# Plotting the bar chart
plt.figure(figsize=(8, 6))
churn_labels = ['False', 'True']
plt.bar(churn_labels, churn_counts.values)
plt.xlabel('Churn')
plt.ylabel('Percentage')
plt.title('Churn Distribution')
plt.show()

churn_rate = ((sum(data['churn'] == True)/ len(data['churn'])*100))
print('Overall Churn rate is ', round(churn_rate, 2), '%')
```

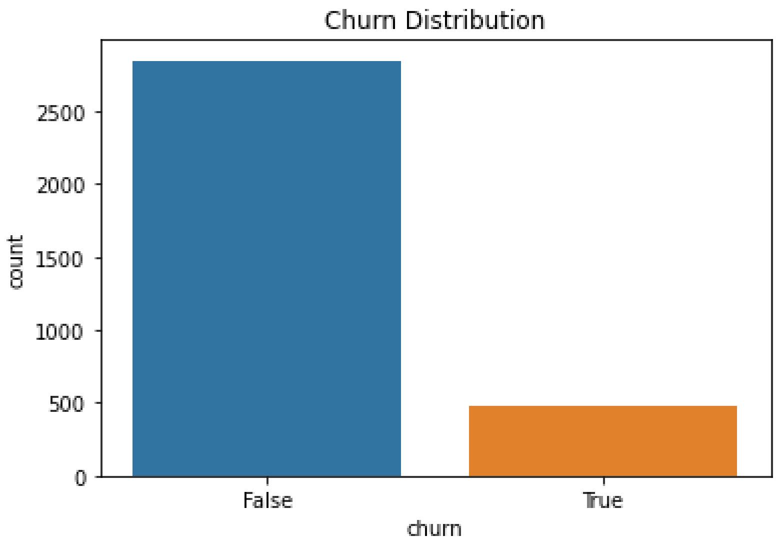


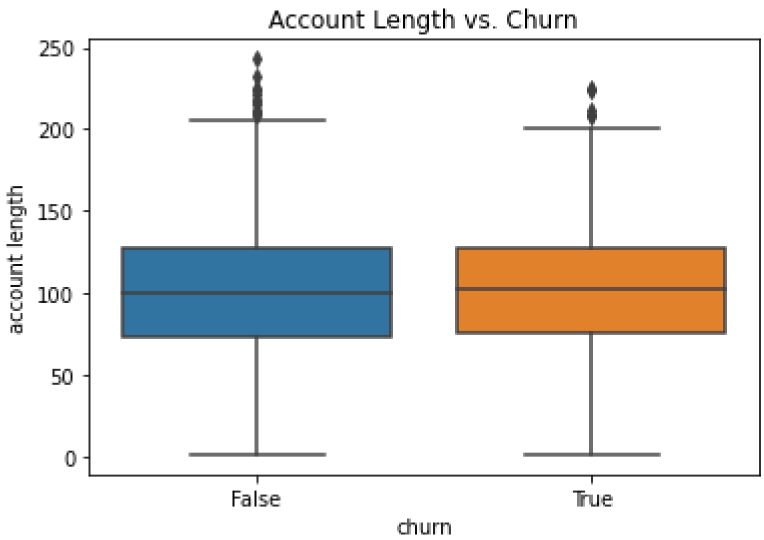

```
Overall Churn rate is  14.49 %
```

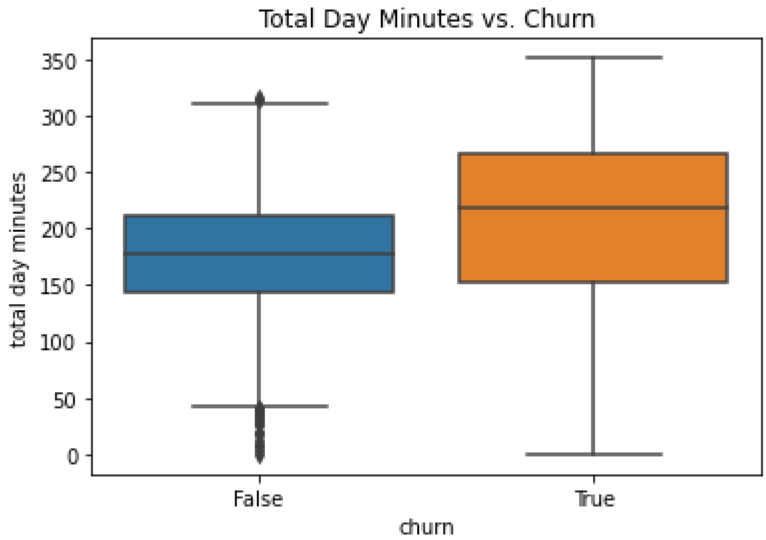
In [20]:
```
sns.pairplot(data, vars=['account length', 'total day minutes', 'total eve m
plt.show()
```

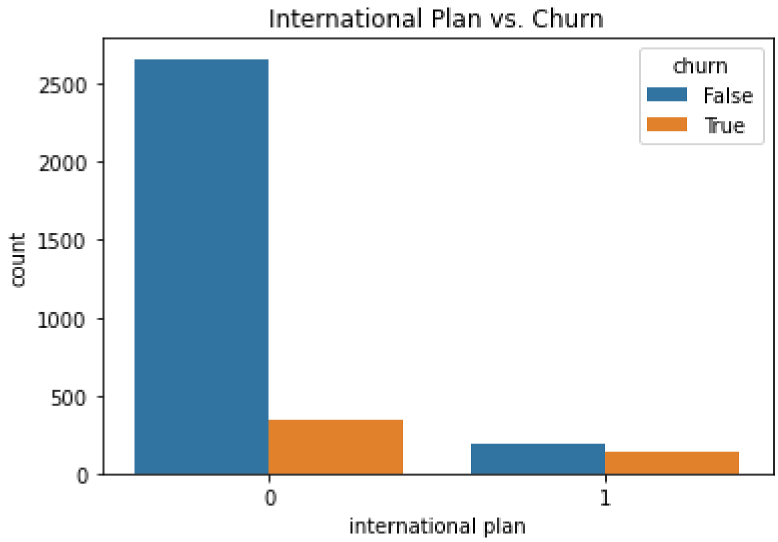


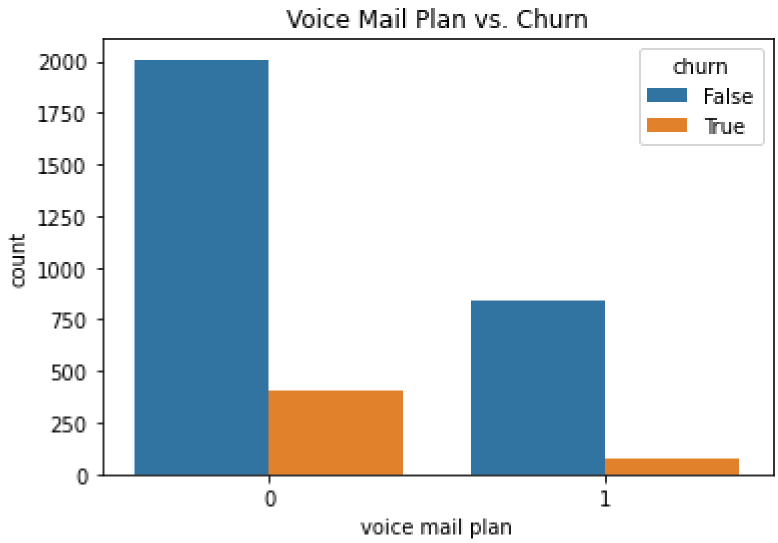
In [21]:
```
# Visualize the distribution of the target variable
sns.countplot(x='churn', data=data)
plt.title('Churn Distribution')
plt.show()

# Explore the relationship between  features and churn
sns.boxplot(x='churn', y='account length', data=data)
plt.title('Account Length vs. Churn')
plt.show()

sns.boxplot(x='churn', y='total day minutes', data=data)
plt.title('Total Day Minutes vs. Churn')
plt.show()

sns.countplot(x='international plan', hue='churn', data=data)
plt.title('International Plan vs. Churn')
plt.show()

sns.countplot(x='voice mail plan', hue='churn', data=data)
plt.title('Voice Mail Plan vs. Churn')
plt.show()
```






Total Day Minutes vs. Churn
total day minutes
350
300
250
200
150
100
50
0
False
True
churn


International Plan vs. Churn
count
2500
2000
1500
1000
500
0
0
1
international plan
churn
False
True


Voice Mail Plan vs. Churn
count
2000
1750
1500
1250
1000
750
500
250
0
0
1
voice mail plan
churn
False
True

In [22]:
```
data.corr(method='pearson').style.format("{:.2}").background_gradient(cmap=p
```

Out[22]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
account length	1.0	0.025	0.0029	-0.0046	0.0062	0.038	0.0062	-0.0068
international plan	0.025	1.0	0.006	0.0087	0.049	0.0038	0.049	0.019
voice mail plan	0.0029	0.006	1.0	0.96	-0.0017	-0.011	-0.0017	0.022
number vmail messages	-0.0046	0.0087	0.96	1.0	0.00078	-0.0095	0.00078	0.018
total day minutes	0.0062	0.049	-0.0017	0.00078	1.0	0.0068	1.0	0.007
total day calls	0.038	0.0038	-0.011	-0.0095	0.0068	1.0	0.0068	-0.021
total day charge	0.0062	0.049	-0.0017	0.00078	1.0	0.0068	1.0	0.007
total eve minutes	-0.0068	0.019	0.022	0.018	0.007	-0.021	0.007	1.0
total eve calls	0.019	0.0061	-0.0064	-0.0059	0.016	0.0065	0.016	-0.011
total eve charge	-0.0067	0.019	0.022	0.018	0.007	-0.021	0.007	1.0
total night minutes	-0.009	-0.029	0.0061	0.0077	0.0043	0.023	0.0043	-0.013
total night calls	-0.013	0.012	0.016	0.0071	0.023	-0.02	0.023	0.0076
total night charge	-0.009	-0.029	0.0061	0.0077	0.0043	0.023	0.0043	-0.013
total intl minutes	0.0095	0.046	-0.0013	0.0029	-0.01	0.022	-0.01	-0.011
total intl calls	0.021	0.017	0.0076	0.014	0.008	0.0046	0.008	0.0025
total intl charge	0.0095	0.046	-0.0013	0.0029	-0.01	0.022	-0.01	-0.011
customer service calls	-0.0038	-0.025	-0.018	-0.013	-0.013	-0.019	-0.013	-0.013
churn	0.017	0.26	-0.1	-0.09	0.21	0.018	0.21	0.093

Modelling

Baseline Model - Naive Bayes

In [23]:
```
#Converting Target Variable into Integers
data['churn'] = data['churn'].astype(int)
data['churn'].value_counts()
```

Out[23]:
```
0    2850
1     483
Name: churn, dtype: int64
```

In [24]:
```
#Setting up the Target and Dataset
y = data['churn']
X = data.drop('churn', axis=1)
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account length          3333 non-null   int64  
 1   international plan      3333 non-null   int32  
 2   voice mail plan         3333 non-null   int32  
 3   number vmail messages   3333 non-null   int64  
 4   total day minutes       3333 non-null   float64
 5   total day calls         3333 non-null   int64  
 6   total day charge        3333 non-null   float64
 7   total eve minutes       3333 non-null   float64
 8   total eve calls         3333 non-null   int64  
 9   total eve charge        3333 non-null   float64
 10  total night minutes     3333 non-null   float64
 11  total night calls       3333 non-null   int64  
 12  total night charge      3333 non-null   float64
 13  total intl minutes      3333 non-null   float64
 14  total intl calls        3333 non-null   int64  
 15  total intl charge       3333 non-null   float64
 16  customer service calls  3333 non-null   int64  
dtypes: float64(8), int32(2), int64(7)
memory usage: 416.8 KB
```

In [25]:
```
#Splitting Training and Test Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, ranc

print(f'My training set is {X_train.shape}')
print(f'My final test set is {X_test.shape}')
print(f'My training set dependent variable is {y_train.shape}')
print(f'My test set dependent variable is {y_test.shape}')
```

```
My training set is (2333, 17)
My final test set is (1000, 17)
My training set dependent variable is (2333,)
My test set dependent variable is (1000,)
```

In [27]:
```
# baseline model using bayes naive learner

# setting up the learner
gnb = GaussianNB()

# fitting the model and predict
model_naive = gnb.fit(X_train, y_train)

y_pred = model_naive.predict_proba(X_train)[:,1]
# y_pred_50 = model_naive.predict(X_train)
# len(y_pred)
# model_naive

roc_auc_score (y_train, y_pred)
```

Out[27]: 0.8433266432513798

In [28]:
```
# Baseline Performance Evaluation Naive Bayes
# Instantiate a stratified k-fold object
skf = StratifiedKFold(n_splits=10, shuffle=True)

param_grid = {'var_smoothing': [1e-09]}

# GridSearchCV for hyperparameter tuning
opt_model_base = GridSearchCV(model_naive,
                              param_grid,
                              cv=skf,
                              scoring='roc_auc',
                              return_train_score=True)

# Fit the model with GridSearchCV
opt_model_base.fit(X_train, y_train)

# Display the GridSearchCV results
opt_model_base.cv_results_
# The validation baseline ROC AUC score mean and std
validation_mean_score = opt_model_base.cv_results_['mean_test_score'][0]
validation_std_score = opt_model_base.cv_results_['std_test_score'][0]

# The training baseline ROC AUC score mean and std
training_mean_score = opt_model_base.cv_results_['mean_train_score'][0]
training_std_score = opt_model_base.cv_results_['std_train_score'][0]

print("The validation baseline roc_auc score is mean {:.4f} std {:.4f}".form
print("The training baseline roc_auc_score is mean {:.4f} std {:.4f}".format

# Convert the CV results into a DataFrame
pd.DataFrame(opt_model_base.cv_results_)
```

```
The validation baseline roc_auc score is mean 0.8358 std 0.0334
The training baseline roc_auc_score is mean 0.8436 std 0.0039
```

Out[28]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_var_smoothing	
0	0.005584	0.001681	0.004993	0.001665	1e-09	{'var_

1 rows × 31 columns

Hyperparameter Tuning on 3 Classification Models

Decision Tree and Hyperparameter Tuning

In [29]:
```
# Set up the learner
model_tree = DecisionTreeClassifier(max_depth=2, min_samples_leaf=10, randon
```

In [30]:
```
# Fit the model
model_tree.fit(X_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=2,
                       min_samples_leaf=10, random_state=40)
```

In [31]:
```
# Perform training/validation test with the stratified k-fold object and fix
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=600)
param_grid = {'max_depth': [2], 'min_samples_leaf': [10]}

basic_model_tree = GridSearchCV(
    DecisionTreeClassifier(random_state=40, class_weight='balanced'),
    param_grid,
    cv=skf,
    scoring='roc_auc',
    return_train_score=True
)

basic_model_tree.fit(X_train, y_train)
```

Out[31]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=10, random_state=600, shuffle=Tru
e),
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              random_state=40),
             param_grid={'max_depth': [2], 'min_samples_leaf': [10]},
             return_train_score=True, scoring='roc_auc')
```

In [32]:
```
# Print results of unoptimized model
unoptimized_mean_test_score = basic_model_tree.cv_results_['mean_test_score'
unoptimized_std_test_score = basic_model_tree.cv_results_['std_test_score'][
unoptimized_mean_train_score = basic_model_tree.cv_results_['mean_train_scor
unoptimized_std_train_score = basic_model_tree.cv_results_['std_train_score'

print(f"The validation unoptimized Decision Tree roc_auc score is mean {unop
      f"std {unoptimized_std_test_score:.3f}")
print(f"The training unoptimized Decision Tree roc_auc_score is mean {unopti
      f"std {unoptimized_std_train_score:.3f}")
```

```
The validation unoptimized Decision Tree roc_auc score is mean 0.7443 std
0.063
The training unoptimized Decision Tree roc_auc_score is mean 0.7637 std 0.
013
```

In [33]:
```
# Apply hyperparameter optimization
skf = StratifiedKFold(n_splits=10, random_state=600, shuffle=True)
param_grid = {'max_depth': range(1, 15), 'min_samples_leaf': [5, 10, 15, 20,

opt_model_tree = GridSearchCV(
    DecisionTreeClassifier(random_state=40, class_weight='balanced'),
    param_grid,
    cv=skf,
    scoring='roc_auc',
    return_train_score=True
)

opt_model_tree.fit(X_train, y_train)

# Get results of hyperparameter optimization
opt_model_tree_results = pd.DataFrame(opt_model_tree.cv_results_)
opt_model_tree_results
```

Out[33]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_
0	0.021542	0.007880	0.011868	0.003950	1	
1	0.007880	0.001808	0.005286	0.002098	1	
2	0.011376	0.004067	0.005483	0.003034	1	
3	0.008579	0.001200	0.004286	0.001482	1	
4	0.006892	0.000303	0.003889	0.000940	1	
...	...	...	...	...	...	
121	0.021851	0.001280	0.003679	0.000631	14	
122	0.022141	0.001163	0.003693	0.000640	14	
123	0.021851	0.002155	0.003387	0.000487	14	
124	0.022934	0.003273	0.003791	0.000978	14	
125	0.020936	0.001773	0.003194	0.000398	14	

126 rows × 32 columns

In [34]:
```
# Print best hyperparameters and roc_auc score
best_hyperparameters = opt_model_tree.best_params_
best_roc_auc_score = opt_model_tree.best_score_

print("Values of the optimized hyperparameters for the best model found:")
print(best_hyperparameters)
print(f"Best roc_auc score: {best_roc_auc_score:.4f}")
```

```
Values of the optimized hyperparameters for the best model found:
{'max_depth': 6, 'min_samples_leaf': 35}
Best roc_auc score: 0.8852
```

Random Forest Classifier and Hyperparameters Tuning

In [35]:
```
# setting up the learner and fitting
model_random_forest = RandomForestClassifier(n_estimators=100,
                                             random_state = 11,
                                             class_weight= 'balanced',
                                             max_depth = 15,
                                             min_samples_leaf= 20
                                             )
model_random_forest.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier(class_weight='balanced', max_depth=15,
                       min_samples_leaf=20, random_state=11)
```

In [36]:
```
#  estimating initial performance with only two fixed parameters max_depth 1

skf = StratifiedKFold(n_splits=10, random_state=600, shuffle=True)

param_grid = {'max_depth': [15], 'min_samples_leaf': [20] }

basic_model_random = GridSearchCV(RandomForestClassifier(random_state=11, cl
                         param_grid,
                         cv=skf,
                         scoring='roc_auc',
                         return_train_score=True)

# fitting the initial model
basic_model_random.fit(X_train,y_train)

basic_model_random.cv_results_
```

Out[36]:
```
{'mean_fit_time': array([0.52151237]),
 'std_fit_time': array([0.11224904]),
 'mean_score_time': array([0.02360296]),
 'std_score_time': array([0.00980857]),
 'param_max_depth': masked_array(data=[15],
              mask=[False],
        fill_value='?',
             dtype=object),
 'param_min_samples_leaf': masked_array(data=[20],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 15, 'min_samples_leaf': 20}],
 'split0_test_score': array([0.78970588]),
 'split1_test_score': array([0.88897059]),
 'split2_test_score': array([0.89985294]),
 'split3_test_score': array([0.88693467]),
 'split4_test_score': array([0.91028673]),
 'split5_test_score': array([0.91959799]),
 'split6_test_score': array([0.9255099]),
 'split7_test_score': array([0.90836536]),
 'split8_test_score': array([0.85737511]),
 'split9_test_score': array([0.90230565]),
 'mean_test_score': array([0.88889048]),
 'std_test_score': array([0.03775683]),
 'rank_test_score': array([1]),
 'split0_train_score': array([0.97010159]),
 'split1_train_score': array([0.96869999]),
 'split2_train_score': array([0.96945091]),
 'split3_train_score': array([0.96882309]),
 'split4_train_score': array([0.96838408]),
 'split5_train_score': array([0.96787039]),
 'split6_train_score': array([0.96683936]),
 'split7_train_score': array([0.96778295]),
 'split8_train_score': array([0.97026945]),
 'split9_train_score': array([0.96974119]),
 'mean_train_score': array([0.9687963]),
 'std_train_score': array([0.00105304])}
```

In [37]:
```
print('The validation unoptimized Random Forest Tree roc_auc score Mean:', 0
print('The validation unoptimized Random Forest Tree roc_auc score Std:', 0.
print('The training unoptimized Random Forest Tree roc_auc_score Mean:', 0.9
print('The training unoptimized Random Forest Tree roc_auc_score Std:', 0.00
```

```
The validation unoptimized Random Forest Tree roc_auc score Mean: 0.896815
03
The validation unoptimized Random Forest Tree roc_auc score Std: 0.0288162
2
The training unoptimized Random Forest Tree roc_auc_score Mean: 0.97061469
The training unoptimized Random Forest Tree roc_auc_score Std: 0.0010613
```

In [38]:
```
# applying hyperparameter optimisations
skf = StratifiedKFold(n_splits=10, random_state=600, shuffle=True)

param_grid = {'max_depth': range(1,15), 'min_samples_leaf': [5,10,15,20,25]}

# change all the parameters in the GridSearch CV!
opt_model_forest = GridSearchCV(RandomForestClassifier(random_state=11, clas
                         param_grid,
                         cv=skf,
                         scoring='roc_auc',
                         return_train_score=True)
# fitting the optimal model
opt_model_forest.fit(X_train,y_train)

# turning the GridSearch CV into a dataframe
pd.DataFrame(opt_model_forest.cv_results_)
```

Out[38]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_r
0	0.266873	0.045133	0.018267	0.003744	1	
1	0.247352	0.013708	0.017456	0.002874	1	
2	0.239949	0.009864	0.016551	0.001563	1	
3	0.229490	0.008249	0.015752	0.000591	1	
4	0.245936	0.030172	0.017249	0.003432	1	
...	...	...	...	...	...	
65	0.522392	0.009514	0.017767	0.000745	14	
66	0.496471	0.011100	0.019138	0.004967	14	
67	0.481111	0.010174	0.018054	0.001043	14	
68	0.466466	0.008684	0.018156	0.001404	14	
69	0.458256	0.009739	0.017764	0.000864	14	

70 rows × 32 columns

In [39]:
```
print('Values of the optimised hyperparameters\nfor the best model found:\n'
opt_model_forest.best_score_
```

```
Values of the optimised hyperparameters
for the best model found:
 {'max_depth': 10, 'min_samples_leaf': 5}
```

Out[39]:
```
0.9033033550103458
```

Logistic Regression

In [40]:
```
# Setting up the learner and fitting the model
log_model = LogisticRegression(class_weight='balanced', penalty='l2', randon
log_model.fit(X_train, y_train)
```

Out[40]:
```
LogisticRegression(class_weight='balanced', random_state=15, solver='libli
near')
```

In [42]:
```
# Defining the cross-validation strategy
skf = StratifiedKFold(n_splits=10, random_state=600, shuffle=True)

# Defining the parameter grid for grid search
param_grid = {'penalty': ['l2']}

# Creating a grid search object with logistic regression model
basic_model_logistic = GridSearchCV(LogisticRegression(class_weight='balance
                                    param_grid,
                                    cv=skf,
                                    scoring='roc_auc',
                                    return_train_score=True)

# Fitting the initial model
basic_model_logistic.fit(X_train, y_train)

# Accessing the cross-validation results
basic_model_logistic.cv_results_
```

Out[42]:
```
{'mean_fit_time': array([0.04507983]),
 'std_fit_time': array([0.0077169]),
 'mean_score_time': array([0.00408747]),
 'std_score_time': array([0.00103807]),
 'param_penalty': masked_array(data=['l2'],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'penalty': 'l2'}],
 'split0_test_score': array([0.71691176]),
 'split1_test_score': array([0.81529412]),
 'split2_test_score': array([0.80514706]),
 'split3_test_score': array([0.8380136]),
 'split4_test_score': array([0.85087201]),
 'split5_test_score': array([0.7970736]),
 'split6_test_score': array([0.79677801]),
 'split7_test_score': array([0.83978717]),
 'split8_test_score': array([0.84303872]),
 'split9_test_score': array([0.7970736]),
 'mean_test_score': array([0.80999897]),
 'std_test_score': array([0.0369596]),
 'rank_test_score': array([1]),
 'split0_train_score': array([0.83142686]),
 'split1_train_score': array([0.82060591]),
 'split2_train_score': array([0.81978938]),
 'split3_train_score': array([0.81890616]),
 'split4_train_score': array([0.81672387]),
 'split5_train_score': array([0.82149285]),
 'split6_train_score': array([0.82024687]),
 'split7_train_score': array([0.8178642]),
 'split8_train_score': array([0.81876225]),
 'split9_train_score': array([0.82192821]),
 'mean_train_score': array([0.82077466]),
 'std_train_score': array([0.00386024])}
```

In [43]:
```
print('The validation unoptimized Random Forest Tree roc_auc score Mean:', 0
print('The validation unoptimized Random Forest Tree roc_auc score Std:', 0.
print('The training unoptimized Random Forest Tree roc_auc_score Mean:', 0.8
print('The training unoptimized Random Forest Tree roc_auc_score Std:', 0.00
```

```
The validation unoptimized Random Forest Tree roc_auc score Mean: 0.814307
51
The validation unoptimized Random Forest Tree roc_auc score Std: 0.0351649
9
The training unoptimized Random Forest Tree roc_auc_score Mean: 0.8229525
The training unoptimized Random Forest Tree roc_auc_score Std: 0.00390355
```

In [44]:
```
# hyperparameter tuning on Logistic regression
skf = StratifiedKFold(n_splits=10, random_state=600, shuffle=True)

param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100]
             }

opt_model_logistic = GridSearchCV(LogisticRegression(class_weight='balanced'
                                  param_grid,
                                  cv=skf,
                                  scoring='roc_auc',
                                  return_train_score=True)

# Fitting the optimal model
opt_model_logistic.fit(X_train, y_train)

pd.DataFrame(opt_model_logistic.cv_results_)
```

```
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\svm\_ba
se.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\svm\_ba
se.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\svm\_ba
se.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\svm\_ba
se.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\svm\_ba
se.py:976: ConvergenceWarning: Liblinear failed to converge, increase th
```

In [45]:
```
print('Values of the optimized hyperparameters for the best model found:')
print(opt_model_logistic.best_params_)
print('Best ROC AUC score: {:.4f}'.format(opt_model_logistic.best_score_))
```

```
Values of the optimized hyperparameters for the best model found:
{'C': 10, 'penalty': 'l1'}
Best ROC AUC score: 0.8116
```

Evaluation of The Winning Model - Desicion Tree

Based on the performance of the models, it appears that the Decision Tree classifier outperforms the Random Forest classifier. Although the difference in ROC scores is not significant, I have decided to adopt the Decision Tree as our final model due to its higher interpretability.

Estimating the underlying costs for TP, FP, TN and FN

The average cost for telco prospecting in the US is around 315 US dollars made up of marketing initiatives dedicated to make our prospects convert. Retaining an existing customer (and generally speaking keeping them satisfied) is roughly 5 times cheaper with an estimate of $60 per customer. Below costs associated to each scenario.

FN = That would be when the model predicted the user wouldn't churn when they actually would. After some research we have found that the cost per acquisition of a new customer to replace the lost one is around $315. This is the most expensive scenario and what Syriatel wants to avoid the most.

TP = In this case, model would predict that the customer is churning when they actually would and we need to spend $60 to keep them happy.

FP = Model is predicting that the customer would churn but in reality, they wouldn't. We still spend $60 to keep them happy.

TN = This is the scenario with less impact as we are corretly identifying happy customers ($0).

The m (Metz) parameter that we need to calculate the ideal threshold is given by the following formula:

In [46]:
```
prevalence = .22
FN = 315
TP = 60
FP = 60
TN = 0

m = ((1.0 - prevalence)/(prevalence)) * ((60-0)/(315-60))
print(f'Metz parameter is {m}' )
```

```
Metz parameter is 0.8342245989304813
```

Identifying optimal threshold given our Metz value

In [47]:
```
# refitting my best model with optimal max_depth 5 and min_sample_leafs 35

model_tree_f = DecisionTreeClassifier(max_depth=5,min_samples_leaf=35,randon
# fitting the model
model_tree_f.fit(X_train, y_train)

# TESTING ON TEST DATA, good results
y_hat_decision_tree = model_tree_f.predict_proba(X_test)[:,1]
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_hat_decision_tree)


# good roc score on test dataset
roc_auc_score(y_test, y_hat_decision_tree)
```

Out[47]:
```
0.9093397850690733
```

In [48]:
```
# Calculating the F-Measure and Thresholds
fm_list = (tpr_test) -(m*(fpr_test))
list(zip(fm_list.tolist(), thresholds_test.tolist()))
```

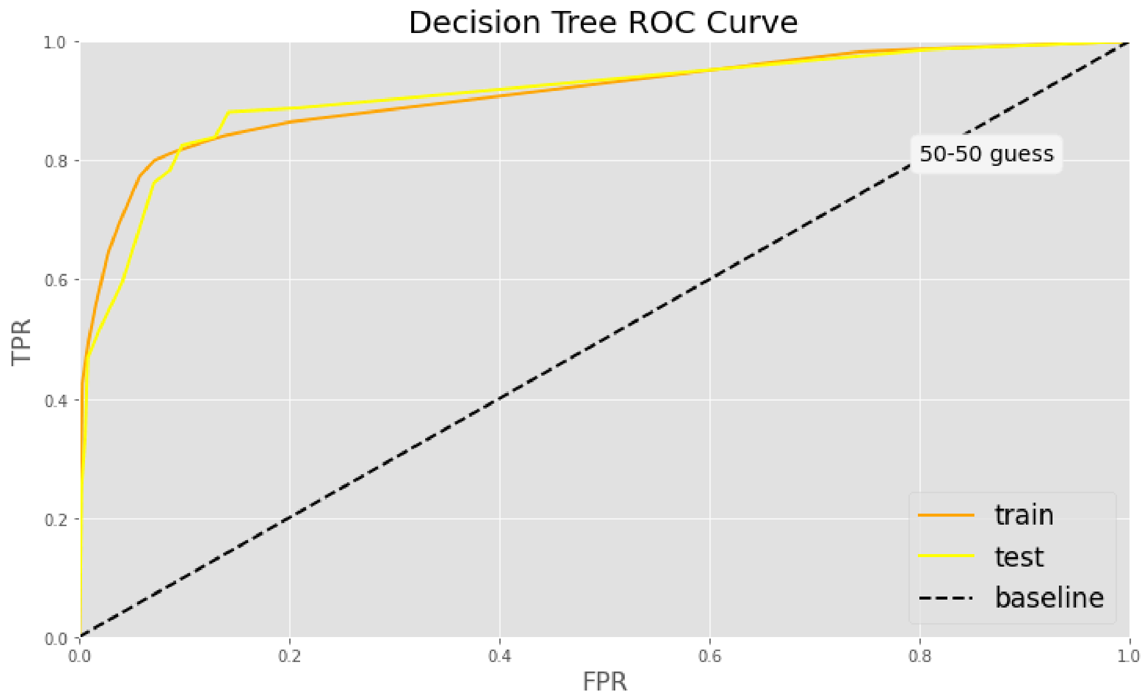
Out[48]:
```
[(0.0, 1.9969078259287891),
 (0.1039216806256411, 0.996907825928789),
 (0.24980140320932417, 0.9907804944185836),
 (0.33079721431701664, 0.9903457088095408),
 (0.4617174986452219, 0.9214054553860379),
 (0.5009343049016328, 0.8978825649496921),
 (0.5603357449743612, 0.8808235670634135),
 (0.6255777306638725, 0.8259428097803564),
 (0.7028588818004701, 0.8022990562240014),
 (0.7111833872764616, 0.6532760316130677),
 (0.7434071865398656, 0.4290635091496232),
 (0.7311107452503568, 0.3695531244205451),
 (0.7623611202442969, 0.3476364904936333),
 (0.7128955196084031, 0.26967052296867594),
 (0.338507809713795, 0.17837644321131577),
 (0.31240439154503263, 0.07432961623093275),
 (0.1657754010695187, 0.06271831828051735)]
```

Plotting Winning Decision Tree Model ROC Curve

In [49]:
```
# evaluating TPRs, FPRs and thresholds for both the training and test sets
base_pred_train = model_tree_f.predict_proba(X_train)[:,1]
base_fpr_train, base_tpr_train, base_thresh_train = roc_curve(y_train, base_

y_hat_decision_tree = model_tree_f.predict_proba(X_test)[:,1]
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_hat_decision_tree)
```

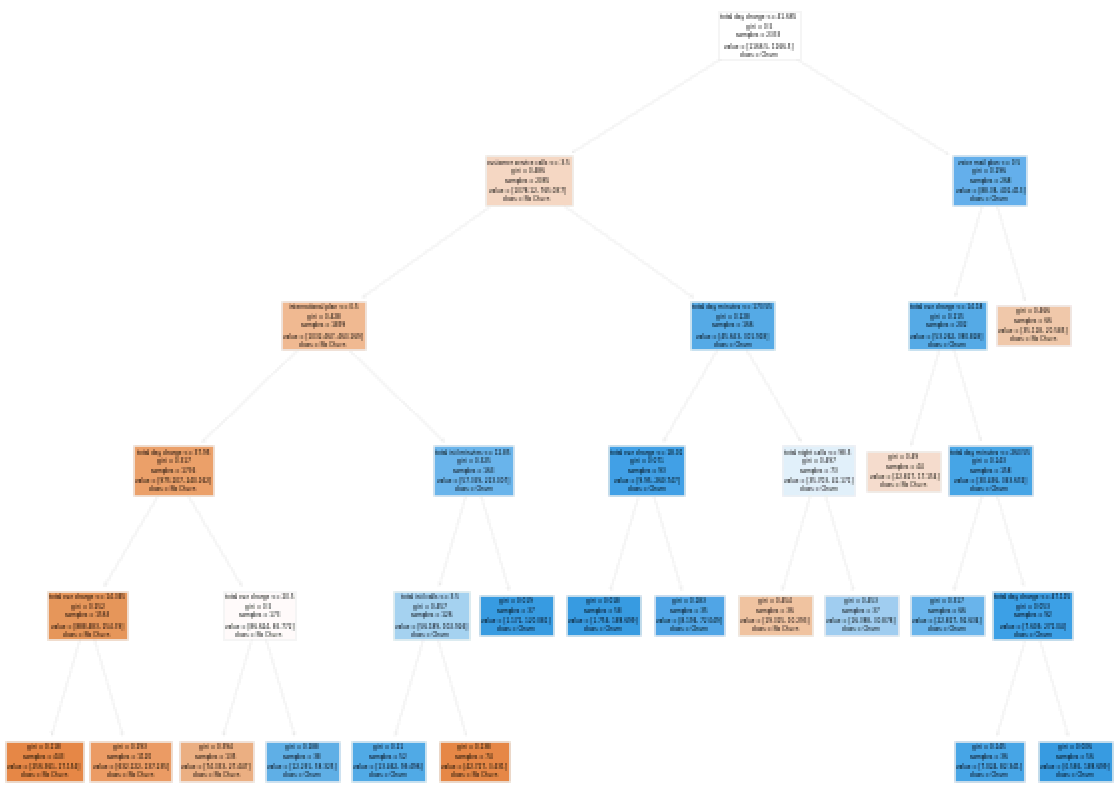
In [50]:
```
# Plotting the ROC Curve

plt.style.use('ggplot')
plt.figure(figsize=(12,7))
ax1 = sns.lineplot(base_fpr_train, base_tpr_train, label='train',)
ax1.lines[0].set_color("orange")
ax1.lines[0].set_linewidth(2)

ax2 = sns.lineplot(fpr_test, tpr_test, label='test')
ax2.lines[1].set_color("yellow")
ax2.lines[1].set_linewidth(2)

ax3 = sns.lineplot([0,1], [0,1], label='baseline')
ax3.lines[2].set_linestyle("--")
ax3.lines[2].set_color("black")
ax3.lines[2].set_linewidth(2)

plt.title('Decision Tree ROC Curve', fontsize=20)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim(0,1)
plt.ylim(0,1)
plt.text(x=0.8, y=0.8, s="50-50 guess", fontsize=14,
         bbox=dict(facecolor='whitesmoke', boxstyle="round, pad=0.4"))
plt.legend(loc=4, fontsize=17)
plt.show();
```

```
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorato
rs.py:36: FutureWarning: Pass the following variables as keyword args: x,
y. From version 0.12, the only valid positional argument will be `data`, a
nd passing other arguments without an explicit keyword will result in an e
rror or misinterpretation.
  warnings.warn(
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorato
rs.py:36: FutureWarning: Pass the following variables as keyword args: x,
y. From version 0.12, the only valid positional argument will be `data`, a
nd passing other arguments without an explicit keyword will result in an e
rror or misinterpretation.
  warnings.warn(
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorato
rs.py:36: FutureWarning: Pass the following variables as keyword args: x,
y. From version 0.12, the only valid positional argument will be `data`, a
nd passing other arguments without an explicit keyword will result in an e
rror or misinterpretation.
  warnings.warn(
```



In [51]:

```
# Visualizing Decision Tree

from sklearn.tree import plot_tree
plt.figure(figsize=(10, 8))
plot_tree(model_tree_f, filled=True, feature_names=X.columns, class_names=[
plt.show()
```



Plotting Confusion Matrix for selected threshold

In [52]:
```
# creating a new list with threshold 0.53 separating churn 1 and non-churn 0
probs_list_test = model_tree_f.predict_proba(X_test)[:,1]

final_res =[]
for x in probs_list_test:
    if x > 0.3476364904936333:
        final_res.append(1)
    else:
        final_res.append(0)
final_res
len(final_res)
```

Out[52]:
```
1000
```

In [53]:
```
# plotting the confusion matrix for the .53 threshold
confusion_matrix(y_test, final_res)
```

Out[53]:
```
array([[746, 111],
       [ 23, 120]], dtype=int64)
```

In [54]:
```
# listing all the TN, FP, FN, TP
tn, fp, fn, tp = confusion_matrix(y_test, final_res).ravel()
tn, fp, fn, tp
```

Out[54]:
```
(746, 111, 23, 120)
```

In [55]:
```
# evaluating performance on this specific confusion matrix
accuracy = print('Accuracy Score', accuracy_score(y_test, final_res))
roc_score = print('ROC_score ', roc_auc_score(y_test, y_hat_decision_tree))
precision = print('Precision ', precision_score(y_test, final_res))
recall= print('Recall or TPR ', recall_score(y_test, final_res))
f1__score = print('F1 score ', f1_score(y_test, final_res))

# power
# alpha
# precision
```

```
Accuracy Score 0.866
ROC_score  0.9093397850690733
Precision  0.5194805194805194
Recall or TPR  0.8391608391608392
F1 score  0.6417112299465241
```

In [56]:
```
# additional metrics including Type I error (alpha), statistical power (1-Be
alpha = 111/ (111 + 746)
print("alpha = ", alpha)

power = 120/(120 + 23)
print("power = ", power)

precision = 120/(120 +111)
print("precision = ", precision)

accuracy = (746 + 120 )/(111+ 23 + 746 + 120)
print("accuracy = ", accuracy)
```

```
alpha =  0.1295215869311552
power =  0.8391608391608392
precision =  0.5194805194805194
accuracy =  0.866
```

- The alpha value represents the probability of our model falsely predicting that a customer will churn when they actually wouldn't. This misclassification occurs approximately once in every ten predictions.
- The power of our model refers to its ability to accurately identify customers who are likely to churn, achieving a correct prediction rate of 80% out of all churn instances.

Although our model correctly predicts churn for only half of the customers it identifies, it prioritizes minimizing potential losses by erring on the side of caution. The cost of incorrectly identifying a non-churning customer is $60, while failing to recognize an impending churn results in five times higher marketing expenditure. Fortunately, this failure to identify churn only occurs in approximately two out of ten customers (miss rate or 1 minus beta).

Our model achieves around 90% accuracy rate in correctly identifying both churn and non-churn customers.

How much money could this model save you?

Pre-Model Loss

We know churn rate is overall 14%. Out of a thousand people in a pre-model scenario, that would incur in the cost of losing 144 customers without doing anything about it and thus having to spend `$ 315 for each of their replacements.

Total Loss for Churning Customers > 144 x 315 = $ 45,360

After-Model Loss

Out of the same 1000 people sample we would still mistakenly think that 23 people would not churn when they will (total cost 315*23 = 7245). At the same time this model would make you mistakenly spend spend 60 *111 = 6660 on people we thought would churn but they won't. Lastly, it would correctly take preventive measures and spend the 60 marketing on 114 people who were actually about to churn and we will try to retain (60 *120).

Summing all the costs above > 7245 + 6660 + 7200 = $ 21,105

We would be saving on average (45,360 - 21,105) $24,255 per 1000 customers

Understanding Features Importance

In [57]:
```
# listing all the decision tree coefficients
model_tree_f.feature_importances_
```

Out[57]:
```
array([0.        , 0.24420359, 0.03817875, 0.        , 0.03996007,
       0.        , 0.24514141, 0.        , 0.        , 0.07479717,
       0.        , 0.00510119, 0.        , 0.02435269, 0.06393326,
       0.        , 0.26433187])
```

In [58]:
```
# creating a zip obejct with column names and decision tree coefficients
all_coef = dict(zip(data.columns, model_tree_f.feature_importances_ ))
all_coef
```

Out[58]:
```
{'account length': 0.0,
 'international plan': 0.2442035905919021,
 'voice mail plan': 0.03817875128415697,
 'number vmail messages': 0.0,
 'total day minutes': 0.039960070755545044,
 'total day calls': 0.0,
 'total day charge': 0.2451414096392877,
 'total eve minutes': 0.0,
 'total eve calls': 0.0,
 'total eve charge': 0.07479717265040517,
 'total night minutes': 0.0,
 'total night calls': 0.005101187869800938,
 'total night charge': 0.0,
 'total intl minutes': 0.024352685207632112,
 'total intl calls': 0.06393325759355394,
 'total intl charge': 0.0,
 'customer service calls': 0.264331874407716}
```

In [59]:
```
# converting column names into a list and slicing the last column 'churn' ou
x = list(data.columns)
x = x[:-1]
x
```

Out[59]:
```
['account length',
 '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']
```

In [60]:
```
# created a dataframe for all the relevant columns names for features
df_feature_importance = pd.DataFrame(model_tree_f.feature_importances_, colu
df_feature_importance

second_ = pd.DataFrame(x, columns=['feature'])
second_
```

Out[60]:

	feature
0	account length
1	international plan
2	voice mail plan
3	number vmail messages
4	total day minutes
5	total day calls
6	total day charge
7	total eve minutes
8	total eve calls
9	total eve charge
10	total night minutes
11	total night calls
12	total night charge
13	total intl minutes
14	total intl calls
15	total intl charge
16	customer service calls

In [61]:
```
second_ = pd.DataFrame(x, columns=['feature'])
second_
```

Out[61]:

	feature
0	account length
1	international plan
2	voice mail plan
3	number vmail messages
4	total day minutes
5	total day calls
6	total day charge
7	total eve minutes
8	total eve calls
9	total eve charge
10	total night minutes
11	total night calls
12	total night charge
13	total intl minutes
14	total intl calls
15	total intl charge
16	customer service calls

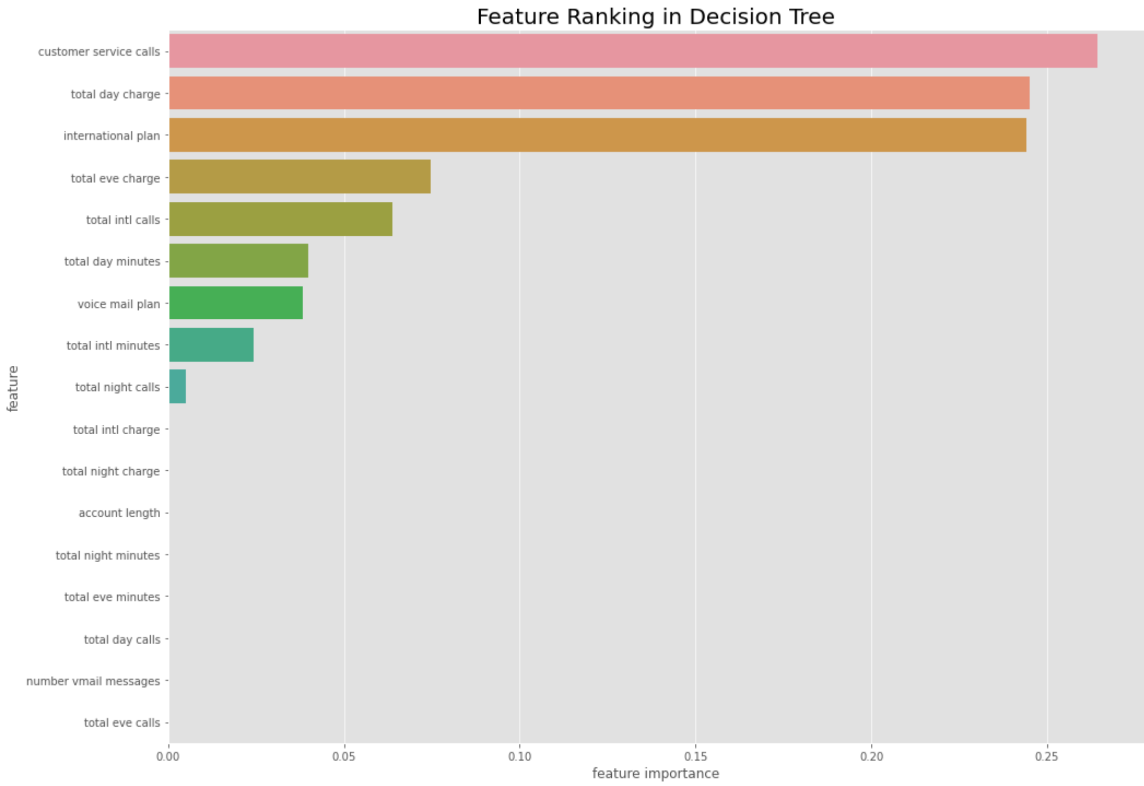
In [62]:
```
# creating a dataframe with feature names and importance combined
mini_df_features = pd.concat([second_, df_feature_importance], axis = 1)
ordered = mini_df_features.sort_values(by ='feature importance', ascending =

ordered.drop('index', axis =1)
```

Out[62]:

	feature	feature importance
0	customer service calls	0.264332
1	total day charge	0.245141
2	international plan	0.244204
3	total eve charge	0.074797
4	total intl calls	0.063933
5	total day minutes	0.039960
6	voice mail plan	0.038179
7	total intl minutes	0.024353
8	total night calls	0.005101
9	total intl charge	0.000000
10	total night charge	0.000000
11	account length	0.000000
12	total night minutes	0.000000
13	total eve minutes	0.000000
14	total day calls	0.000000
15	number vmail messages	0.000000
16	total eve calls	0.000000

In [63]:
```
# plotting the decision tree importance features
plt.figure(figsize=(16,12))
sns.barplot(ordered['feature importance'],
            ordered['feature'],
            orient = 'h',
            )

plt.title ('Feature Ranking in Decision Tree', fontsize = 20)

plt.show()
```

```
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorato
rs.py:36: FutureWarning: Pass the following variables as keyword args: x,
y. From version 0.12, the only valid positional argument will be `data`, a
nd passing other arguments without an explicit keyword will result in an e
rror or misinterpretation.
  warnings.warn(
```



Sample Customers

After training the model, tuning hyperparameters, and selecting thresholds, we can examine the insights provided by these predictions regarding the underlying factors. To do so, we will randomly select an observation from our dataset as our "baseline customer" and analyze how altering the top three features influences the churn probability for our customers.

These modifications represent the potential impact of interventions on SyriaTel's customers. These interventions encompass various strategies such as marketing campaigns, loyalty rewards, price reductions, or any other incentives aimed at encouraging existing customers to retain their loyalty to the company.

In [65]:
```
#Ease of life functions


# function that takes in the example variables as one row of a dataframe and
# before and after the changes, as well as the change in probability as a re
def prediction(x):
    prediction = model_tree_f.predict_proba(x)[:,1]
    prob1 = prediction[0]*100
    prob2 = prediction[1]*100
    perc_diff = ((prob2 - prob1)/prob1) *100

    if perc_diff <= 0:
        symbol = ''
    else:
        symbol = '+'

    return print(f'Original predicted probability of this customer churning:
                 f'{ round(prob1,2) }%', '\n', '\n',
                 f'New predicted probability of this customer churning:', '\
                 f'{ round(prob2,2) }%', '\n', '\n',
                 f'Percentage difference:', '\n',
                 f'{symbol}{ round(perc_diff,2) }%')




#function creates a dataframe with 2 identical rows by duplicating an arbitr
#this will form the basis for the before and after comparisons made below
def fresh_comparison():
    # Randomly chosen row in the data, acting as an example customer
    example1 = list(X_train.loc[2,:])

    # example customer being duplicated and both placed in a df
    examples_df = pd.DataFrame([example1, example1],
                               columns=X_train.columns,
                               index=['customer 1', 'customer 2'])
    return examples_df




#function that outputs a slice of the customer comparison dataframe showing
def print_summary(feature):
    return print(examples_df[[feature]], '\n',
                 '(All other variables are controlled for i.e they\'re ident
                 '-----------------------------------------------------------
                )
```

What happens if a customer makes more (or less) customer service calls?

In [66]:
```
# new dataframe of identical rows i.e customers
examples_df = fresh_comparison()


# new duplicated example customer now with 4 more customer service calls mac
examples_df.loc['customer 2', 'customer service calls'] += 2


print_summary('customer service calls')
prediction(examples_df)
```

```
            customer service calls
customer 1                     0.0
customer 2                     2.0
 (All other variables are controlled for i.e they're identical)
 ------------------------------------------------------------------

Original predicted probability of this customer churning:
26.97%

 New predicted probability of this customer churning:
 26.97%

 Percentage difference:
 0.0%
```

In [67]:
```
# new dataframe of identical rows i.e customers
examples_df = fresh_comparison()


# new duplicated example customer now with 7 more customer service calls mac
examples_df.loc['customer 2', 'customer service calls'] += 4


print_summary('customer service calls')
prediction(examples_df)
```

```
            customer service calls
customer 1                     0.0
customer 2                     4.0
 (All other variables are controlled for i.e they're identical)
 ------------------------------------------------------------------

Original predicted probability of this customer churning:
26.97%

 New predicted probability of this customer churning:
 65.33%

 Percentage difference:
 +142.25%
```

In [68]:
```
# new dataframe of identical rows i.e customers
examples_df = fresh_comparison()


# new duplicated example customer now with 7 more customer service calls mac
examples_df.loc['customer 2', 'customer service calls'] += 6


print_summary('customer service calls')
prediction(examples_df)
```

```
            customer service calls
customer 1                     0.0
customer 2                     6.0
 (All other variables are controlled for i.e they're identical)
 ------------------------------------------------------------------

Original predicted probability of this customer churning:
26.97%

 New predicted probability of this customer churning:
 65.33%

 Percentage difference:
 +142.25%
```

Insights Breakdown:

In this analysis, we observe the impact of customer service calls on the probability of churn for a specific customer. When the number of customer service calls is between 0 and 3, the probability of churn remains constant at 26.97% for this particular customer. However, if the number of calls increases to 4 or more, the probability of churn rises significantly to 65.33%.

This difference can be attributed to the customer's level of comfort in reaching out for inquiries and the effectiveness of issue resolution. When a customer feels comfortable making calls and has their inquiries promptly resolved without recurring problems, the likelihood of churn remains relatively low. Conversely, if a customer needs to make multiple calls due to unresolved issues or a continuous emergence of new problems, the chances of losing that customer increase substantially.

Potential Solution: Implementing a marketing campaign that highlights SyriaTel's customer services as the friendliest and most approachable in the industry could help address this issue. By emphasizing the company's commitment to resolving customer inquiries efficiently and effectively, it may alleviate concerns and reduce the likelihood of customer churn.

What happens if the customers total day charge were to increase?

In [69]:
```
# new dataframe of identical rows i.e customers
examples_df = fresh_comparison()


# new duplicated example customer now with 50% increase in total day charge
examples_df.loc['customer 2', 'total day charge']*= 1.5


print_summary('total day charge')
prediction(examples_df)
```

```
            total day charge
customer 1             41.38
customer 2             62.07
 (All other variables are controlled for i.e they're identical)
 -----------------------------------------------------------------

Original predicted probability of this customer churning:
26.97%

 New predicted probability of this customer churning:
 42.91%

 Percentage difference:
 +59.11%
```

Insights Breakdown:

Upon analysis, we observe an interesting trend where a 50% increase in the total price charged for a day results in a more significant decrease in the probability of a customer leaving. Initially, this may seem counterintuitive – one would expect that increasing the cost of services would potentially drive customers away. However, it's important to consider that the total day charge is influenced by both the price per minute and the duration of service usage.

Therefore, it is plausible to conclude that this decrease in the probability of churn is more accurately explained by an increase in the customer's usage of the services, rather than the price alone. As customers spend more time utilizing the services, their likelihood of churn diminishes, potentially indicating that they are deriving more value and satisfaction from the extended usage.

Potential Solution: To address customers identified as at risk of churning, a loyalty bonus program could be implemented. This program would reward customers with discounts on their monthly subscription fee if they surpass a certain threshold of service usage. For example, customers who spend more than three hours on the phone with another SyriaTel customer in a month could receive a 50% discount on their subscription fee for that month. This incentive would encourage increased usage, fostering customer loyalty and reducing the likelihood of churn.