Capstone Project To Predicting Churning Customers Inside 21st Century

Autor: Henrique Martins Prado

- **Objective: The development of this project aimed to identify the churn generation of customers.**
- **Motivation: The project's motivation was to analyze patterns, trends and predictions extracted from the data using machine learning models capable of identifying the significant decrease in the use of services and products by customers.**
- **Importance: The importance of development is precisely to provide companies with the possibility to identify the decrease in consumption of their businesses and to act in order to identify the problems that caused this decrease, as well as to explore new possibilities to retain and attract new customers with differentiated services and new products.**
- **Approach: In the development of the project, the 6 stages of the CRISP-DM methodology were used, precisely because it is a very consolidated methodology in the area of data mining and with a completeness of topics to be developed and explored that allows to create data solutions.**

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas_profiling

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

from sklearn.inspection import permutation_importance
from matplotlib import pyplot

from sklearn.metrics import confusion_matrix, accuracy_score, classification_r
eport
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from yellowbrick.classifier import ConfusionMatrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\utils\deprecation.py:143: FutureWarning: The sklea
rn.metrics.classification module is  deprecated in version 0.22 a
nd will be removed in version 0.24. The corresponding classes / f
unctions should instead be imported from sklearn.metrics. Anythin
g that cannot be imported from sklearn.metrics is now part of the
private API.
  warnings.warn(message, FutureWarning)
```

In [2]:

```
!python --version
```

```
Python 3.8.5
```

Jupyer Notebook Configuration

In [3]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container {width:90% !important;}</style>"))

pd.set_option('display.max_columns', 50)
pd.set_option('display.max_rows', 2000)
```

Phases of CRISP-DM are going to use as reference to develop this project

================== Phase 1: Understanding the Business ===================

What problem I would like to solve?
Predict Churning customers

Questions to guide the business:

- ***Background: In which situation the company is and how the project is going to be conducted to solve the problem?***
- ***Goal: What is the major goal of this project?***
- ***Sucess Criteria: What metric will be use that will inform your project got the sucess?***

Main questions that I need to answer:

- ***Is it possible to achieve a hit rate above 80% in the customer's churn forecast?***
- ***What is the level of correlation between the variables age, year and total_pucharse?***
- ***Is the solution developed passive to be applied in a way that the business identified benefits?***

==================== Phase 2: Understanding the Data ===================

This part is responsible for collect and treat the data. Normally, this part is responsilble for 70% of the time that you will spend in a project.

- In this part I'll collect, describe, explore and check the quality of the data.

In [4]:

```
df = pd.read_csv(r'C:\Users\hik_m\Documents\Jornada do Conhecimento\Projetos\C
apstone Projetct Churning Customers\Datasets\Customer_Churn.csv')
# Shows the type of the object
type(df)
```

Out[4]:

```
pandas.core.frame.DataFrame
```

In [5]:

```
# Show the top 5 rows
df.head()
```

Out[5]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboard
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-07:
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-00:
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06:
3	Phillip White	42.0	8010.76	0	6.71	10.0	2014-12:
4	Cynthia Norton	37.0	9191.58	0	5.56	9.0	2016-15:

In [6]:

```
# Show the last 5 rows
df.tail()
```

Out[6]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboar
895	Paul Miller	42.0	12800.82	1	3.62	8.0	200 1
896	Natalie Hodges	52.0	9893.92	0	6.91	7.0	200 1
897	Ana Smith	45.0	12056.18	0	5.46	4.0	201 0
898	Justin Leonard	51.0	6517.93	1	5.47	10.0	201 0
899	Joseph Williams	39.0	9315.60	1	5.02	10.0	201 1

In [7]:

```
df = df.iloc[:, 0:21]
df.head()
```

Out[7]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboard
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-07:
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-00:
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06:
3	Phillip White	42.0	8010.76	0	6.71	10.0	2014-12:
4	Cynthia Norton	37.0	9191.58	0	5.56	9.0	2016-15:

In [8]:

```
# Total of rows and columns
print('Total amount of rows: ', df.shape[0])
print('Total amount of columns: ', df.shape[1])
```

```
Total amount of rows:  900
Total amount of columns:  10
```

In [9]:

```
# General information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Names            900 non-null    object 
 1   Age              900 non-null    float64
 2   Total_Purchase   900 non-null    float64
 3   Account_Manager  900 non-null    int64  
 4   Years            900 non-null    float64
 5   Num_Sites        900 non-null    float64
 6   Onboard_date     900 non-null    object 
 7   Location         900 non-null    object 
 8   Company          900 non-null    object 
 9   Churn            900 non-null    int64  
dtypes: float64(4), int64(2), object(4)
memory usage: 70.4+ KB
```

In [10]:

```
# Show all columns in my dataset called df
#df.columns

# Show the datatype of each column
#df.dtypes.to_frame()

def total_amount_unique_values(column):
    '''
        Checking the amount values of each column
    '''
    return len(np.unique(df[['column']].values))

# Total amount of Unique values in each column
df.nunique(axis=0)
```

Out[10]:

```
Names              899
Age                 36
Total_Purchase     900
Account_Manager      2
Years              418
Num_Sites           12
Onboard_date       900
Location           900
Company            873
Churn                2
dtype: int64
```

In [11]:

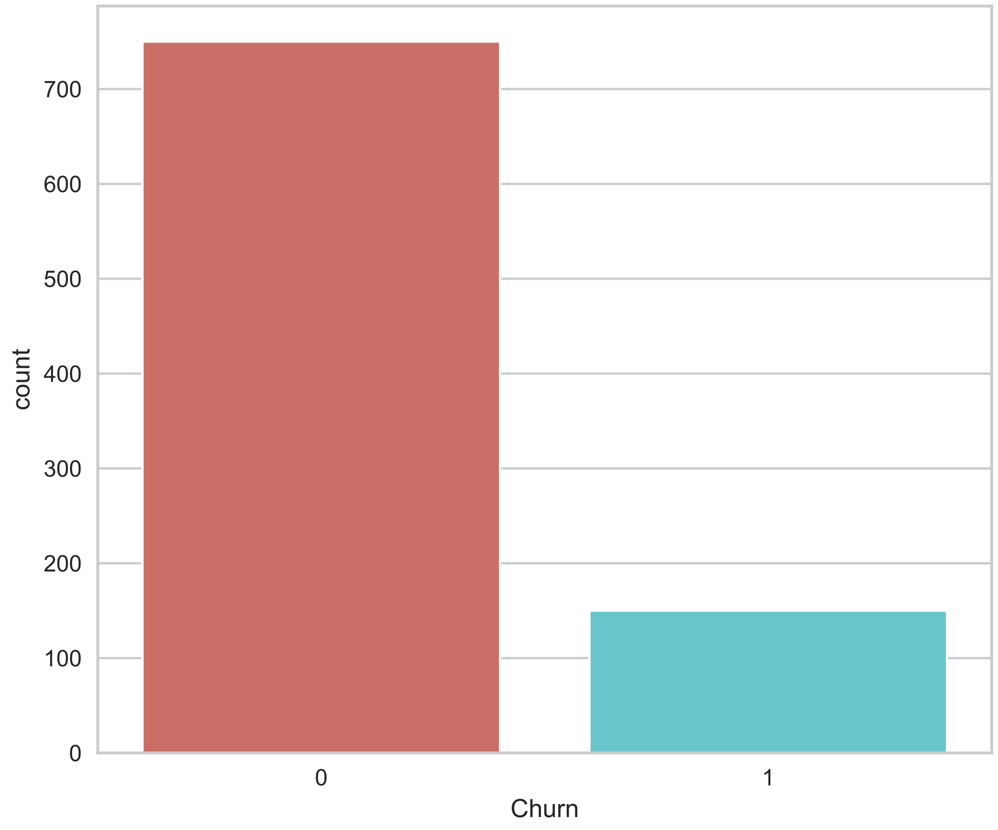
```
# Statistics about my dataset
df.describe()
```

Out[11]:

	Age	Total_Purchase	Account_Manager	Years	Num_Sites	
count	900.000000	900.000000	900.000000	900.000000	900.000000	90
mean	41.816667	10062.824033	0.481111	5.273156	8.587778	
std	6.127560	2408.644532	0.499921	1.274449	1.764836	
min	22.000000	100.000000	0.000000	1.000000	3.000000	
25%	38.000000	8497.122500	0.000000	4.450000	7.000000	
50%	42.000000	10045.870000	0.000000	5.215000	8.000000	
75%	46.000000	11760.105000	1.000000	6.110000	10.000000	
max	65.000000	18026.010000	1.000000	9.150000	14.000000	

In [12]:

```
# Checking the churn distribuition
churn_distr = df['Churn'].value_counts().plot(kind='bar',  figsize=(7, 6), rot
=0, color='blue')
sns.set(font_scale=1.3)
sns.countplot(x = 'Churn', data = df, palette = 'hls')
plt.show()
plt.savefig('count_plot')
churn_distr
```



Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a3af6fa220>
```

```
<Figure size 576x396 with 0 Axes>
```

In [13]:

```
# Checking the churn distribuition
churn_distr = df['Churn'].value_counts()*100/df.shape[0]
churn_distr
```

Out[13]:

```
0    83.333333
1    16.666667
Name: Churn, dtype: float64
```

In [14]:

```
# Create histograma for each variable
#df.hist()
```

In [15]:

```
# Correlation between variables
corr = df.corr()
corr
```

Out[15]:

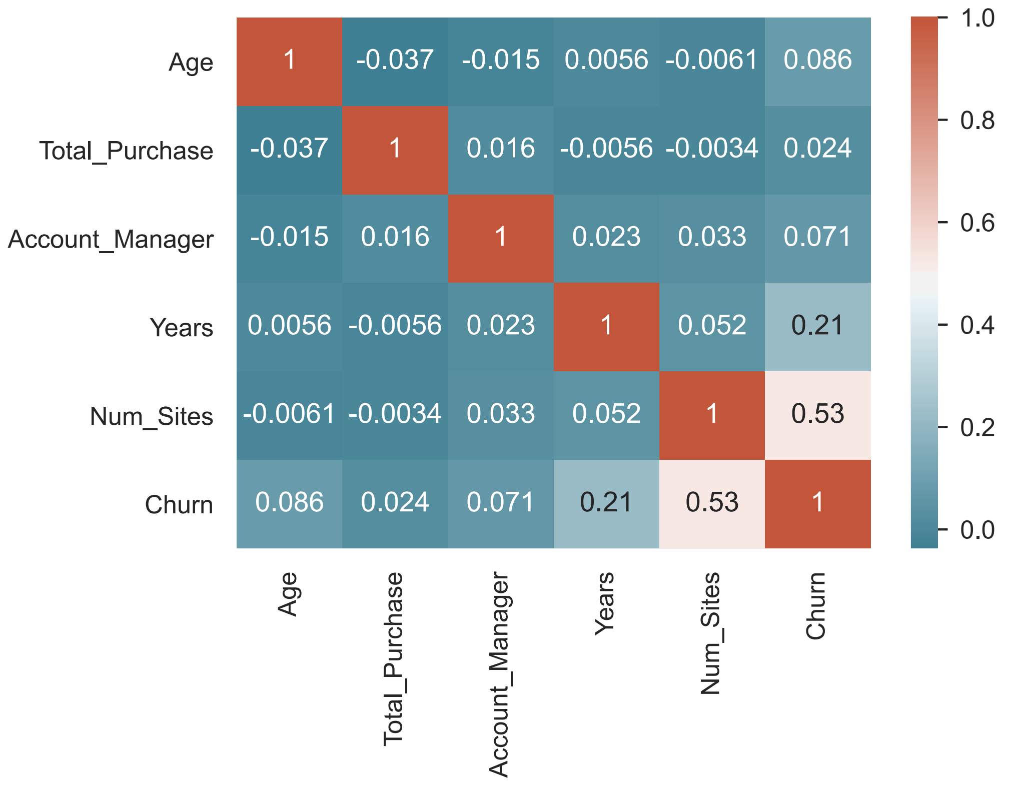
	Age	Total_Purchase	Account_Manager	Years	Num_S
Age	1.000000	-0.037208	-0.014749	0.005625	-0.00
Total_Purchase	-0.037208	1.000000	0.015856	-0.005623	-0.00
Account_Manager	-0.014749	0.015856	1.000000	0.022930	0.03
Years	0.005625	-0.005623	0.022930	1.000000	0.05
Num_Sites	-0.006070	-0.003390	0.033401	0.051642	1.00
Churn	0.085926	0.024031	0.070611	0.214329	0.52

In [16]:

```
# Fazendo o plot da correção acima
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=Tr
ue, cmap=sns.diverging_palette(220, 20, as_cmap=True))
```

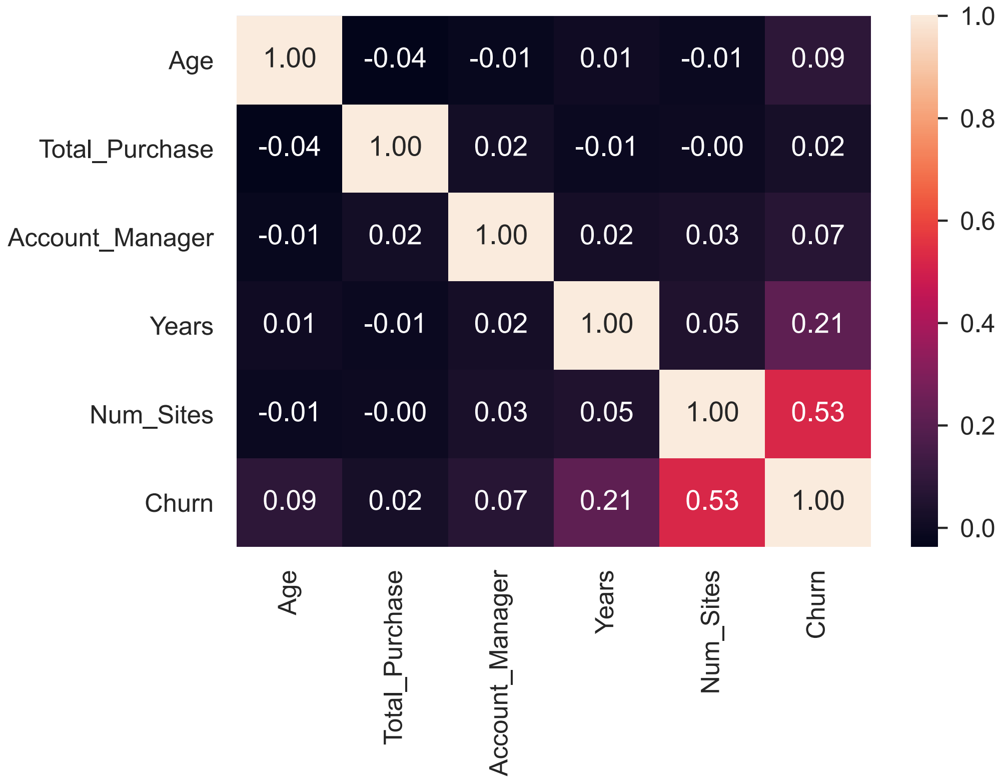
Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a3af6fa4f0>
```



In [17]:

```
# Ploting the correlation using seaborn module
sns.heatmap(df.corr(), annot=True, fmt=".2f");
```

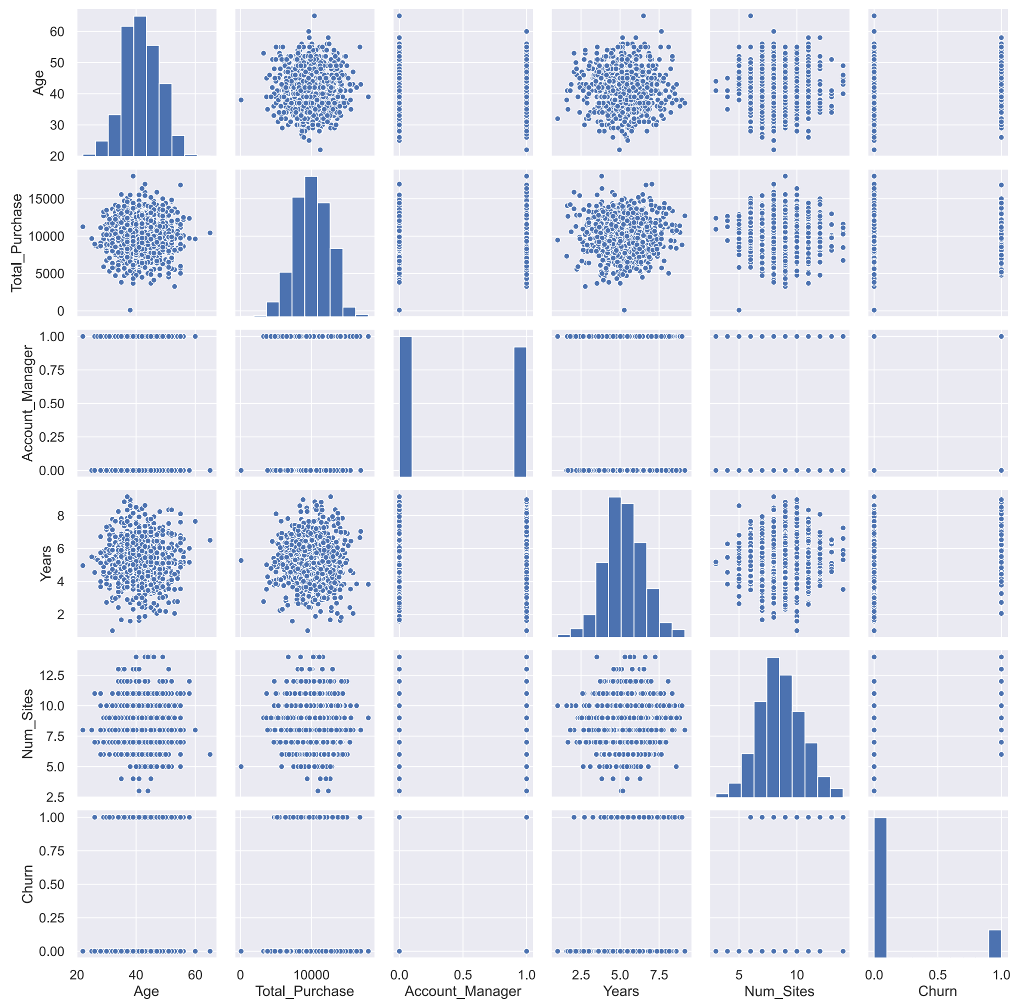


In [18]:

```
sns.pairplot(df)
```

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x2a3c02dc2e0>
```



In [19]:

```
# EDA
df.profile_report()
Projetc_EDA = df.profile_report(title='EDA - Exploratory Data Analysis')
Projetc_EDA.to_file(output_file=r'C:\Users\hik_m\Documents\Jornada do Conhecim
ento\Projetos\Capstone Projetct Churning Customers\EDA - Capstone Project To P
redicting Churning Customers Inside 21st Century.html')
```

```
Summarize dataset: 100%|██████████| 24/24 [00:16<00:00,  1.42it/
s, Completed]
Generate report structure: 100%|██████████| 1/1 [00:04<00:00,  4.
85s/it]
Render HTML: 100%|██████████| 1/1 [00:02<00:00,  2.56s/it]
Export report to file: 100%|██████████| 1/1 [00:00<00:00, 58.82i
t/s]
```

===================== Phase 3: Preparing the Data =====================

In this phase, I'm going to prepar the whole data to the next phase, modeling.

- Data Selection

  Here, I need to select the data that I'll use to train and test my model.

  Will I use:
  - Ouliers?
    The variables are able to use, thre is no high importance to handle with outliers in this dataset
  - All rows and columns?
    I'll use all columns, because there weren't a high correlation between them

In [20]:

```
df.columns
```

Out[20]:

```
Index(['Names', 'Age', 'Total_Purchase', 'Account_Manager', 'Year
s',
       'Num_Sites', 'Onboard_date', 'Location', 'Company', 'Chur
n'],
      dtype='object')
```

In [21]:

```
#df = df.drop(['Onboard_date'], axis = 1)
#df = df.drop(['Location'], axis = 1)
```

In [22]:

```
df.columns
```

Out[22]:

```
Index(['Names', 'Age', 'Total_Purchase', 'Account_Manager', 'Year
s',
       'Num_Sites', 'Onboard_date', 'Location', 'Company', 'Chur
n'],
      dtype='object')
```

In [23]:

```
# Segregating the dataset with predictive variables (independent) of my variab
le to be predicted (label / dependent)

# Predictive
previsoras = df.iloc[:, 1:9].values
previsoras
```

Out[23]:

```
array([[42.0, 11066.8, 0, ..., '2013-08-30 07:00:40',
        '10265 Elizabeth Mission Barkerburgh, AK 89518', 'Harvey
LLC'],
       [41.0, 11916.22, 0, ..., '2013-08-13 00:38:46',
        '6157 Frank Gardens Suite 019 Carloshaven, RI 17756',
        'Wilson PLC'],
       [38.0, 12884.75, 0, ..., '2016-06-29 06:20:07',
        '1331 Keith Court Alyssahaven, DE 90114',
        'Miller, Johnson and Wallace'],
       ...,
       [45.0, 12056.18, 0, ..., '2014-06-20 05:10:09',
        'Unit 8633 Box 8738 DPO AA 14126-5026', 'Schneider-Smit
h'],
       [51.0, 6517.93, 1, ..., '2012-05-30 00:15:43',
        '49800 Torres Ways Suite 886 West Bradleybury, LA 05945-2
648',
        'Robles-Abbott'],
       [39.0, 9315.6, 1, ..., '2010-09-25 12:16:08',
        '27252 Olivia Burgs Rivasmouth, MN 80121-6348', 'Davis Gr
oup']],
      dtype=object)
```

In [24]:

```
previsoras[0]
```

Out[24]:

```
array([42.0, 11066.8, 0, 7.22, 8.0, '2013-08-30 07:00:40',
       '10265 Elizabeth Mission Barkerburgh, AK 89518', 'Harvey L
LC'],
      dtype=object)
```

In [25]:

```
# Label variable
classe = df.iloc[:, 9].values
classe
```

Out[25]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
      dtype=int64)
```

I checked the feature importance in each model after fit. You can check this information in phase 4 - Modeling

- Data Cleaning

Here, it's possible to check the format of each data like numbers interpreted as string, missing values, transformations and etc.

Missing values

In [26]:

```python
def missing_values(dataset):
    '''
        Checking missing values in each colum
    '''
    return dataset.isnull().sum()
```

In [27]:

```
print(missing_values(df))
```

```
Names              0
Age                0
Total_Purchase     0
Account_Manager    0
Years              0
Num_Sites          0
Onboard_date       0
Location           0
Company            0
Churn              0
dtype: int64
```

In this case, there is no need to handle with null values. The data set has all columns filled

Working with categorical variables

To work with the categorical values of the data set I used the label encode which transforms categorical values into numeric values. The generated numerical values characterize each category in the transformed column. There are other ways to act with categorical values such as the One Hot Enconder which distributes categorical values creating new columns in the data set, however for this project the use of Label Encoder provides the transformation without sending data without creating problems to enumerate the existing data and also, this transformation allows data to be stored using less disk space while maintaining the same number of columns in the data set.

In [28]:

```
df.columns
```

Out[28]:

```
Index(['Names', 'Age', 'Total_Purchase', 'Account_Manager', 'Year
s',
       'Num_Sites', 'Onboard_date', 'Location', 'Company', 'Chur
n'],
      dtype='object')
```

In [29]:

```
# I create a variable called labelencoder of the class we just imported (Label
Encoder)
labelencoder = LabelEncoder ()
```

In [30]:

```
# Do the conversion
previsoras[:, 0] = labelencoder.fit_transform(previsoras[:, 0])
previsoras[:, 5] = labelencoder.fit_transform(previsoras[:, 5])
previsoras[:, 6] = labelencoder.fit_transform(previsoras[:, 6])
previsoras[:, 7] = labelencoder.fit_transform(previsoras[:, 7])
#previsoras[:, 7] = labelencoder.fit_transform(previsoras[:, 7])
```

In [31]:

```
previsoras[0]
```

Out[31]:

```
array([18, 11066.8, 0, 7.22, 8.0, 634, 81, 325], dtype=object)
```

In [32]:

```
previsoras
```

Out[32]:

```
array([[18, 11066.8, 0, ..., 634, 81, 325],
       [17, 11916.22, 0, ..., 631, 487, 849],
       [14, 12884.75, 0, ..., 871, 114, 499],
       ...,
       [21, 12056.18, 0, ..., 706, 895, 680],
       [27, 6517.93, 1, ..., 541, 402, 648],
       [15, 9315.6, 1, ..., 417, 206, 189]], dtype=object)
```

In [33]:

```
# Checking the unique values e proportion of each one

#print('Names', df['Names'].unique())
#print('Age', df['Age'].unique())
#print('Total_Purchase', df['Total_Purchase'].unique())
#print('Account_Manager', df['Account_Manager'].unique())
#print('Years', df['Years'].unique())
#print('Num_Sites', df['Num_Sites'].unique())
#print('Onboard_date', df['Onboard_date'].unique())
#print('Location', df['Location'].unique())
#print('Company', df['Company'].unique())
#print('Churn', df['Churn'].unique())
```

In [34]:

```
# Using the train_test_split function to separate training and test data
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsoras,
classe, test_size = 0.3, random_state = 0)
```

In [35]:

```
# Checking data segregation
print('__________ Treinamento __________', '\nDados de Treinamento:' , X_trein
amento.shape, '\nLabel de Treinamento:', y_treinamento.shape)
print('\n_____________ Teste _____________', '\nDados de Teste:' , X_teste.sha
pe, '\nLabel de Teste:', y_teste.shape)
```

```
__________ Treinamento __________
Dados de Treinamento: (630, 8)
Label de Treinamento: (630,)

_____________ Teste _____________
Dados de Teste: (270, 8)
Label de Teste: (270,)
```

- Construct Data

  Here, it's possible to create columns as you need. In this part, I can use the feature engineering the most

  For this exemplo, there wasn't a need to construct more variables

- Integrating Data

  This is an important step when you need to integrate differents datasets and also diferrents orgins data

  For this exemplo, there wasn't a need to integratind data with other sources

======================= Phase 4: Modeling =======================

As the problem I am dealing with in the project to predict customer churn has a label column in which which customers have churned or not, it is inferred in a supervised learning task, where a mathematical function maps inputs (independent variables) to an output (dependent variable). To generate the solution, four machine learning models were tested, chosen at random and which are models used in supervised tasks, they were: Logistic Regression, Naive-Bayes, Random Forest and Decision Trees.

It is important to note that there are a variety of models that can be used to handle the supervised learning task and that it is always valid to use more than one model for the purpose of comparing performance in order to find an optimal solution to the problem addressed. In addition, adjustments can be applied to model parameters to adjust their performance.

Model: Logistic Regression

In [36]:

```
# Creating the logistic regression model
reglog = LogisticRegression()
```

In [37]:

```
# Making the fit of my model. In other words, learning process of my logistic
 regression algorithm
reglog.fit(X_treinamento, y_treinamento)
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[37]:

```
LogisticRegression()
```

In [38]:

```
# Feature Importance - Logistic Regression Model
importances_lr = reglog.coef_
importances_lr
```

Out[38]:

```
array([[-5.50046297e-02, -2.05057256e-04,  2.29853165e-02,
        -8.90824553e-02,  3.87629183e-01, -1.84065095e-03,
        -7.88915406e-04, -6.71747277e-04]])
```

In [39]:

```
# Carrying out predictions using the test model and data
previsoes_reglog = reglog.predict(X_teste)
previsoes_reglog
```

Out[39]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 1], dtype=int64)
```

Model: Naive Bayes - Gaussian

In [40]:

```
# Creating the NaiveBayes object
naive_bayes = GaussianNB()
```

In [41]:

```
# Let's do the model training with the "fit", that is, with the fit I will try
to fit the training data into the model
#tTrying to relate the training data (X_training) to the responses (y_trainin
g)
naive_bayes.fit(X_treinamento, y_treinamento)

# Note: This python algorithm does not allow you to check the probabilities th
at have been generated, however the probability table has already been create
d.
# So, I will no longer need the X_training data, because the model has already
been created and also trained through "fit" with the training data
```

Out[41]:

```
GaussianNB()
```

In [42]:

```
# Feature Importance - Naive Bayes Model
importances_nb = permutation_importance(naive_bayes, X_teste, y_teste)
print(importances_nb.importances_mean)
```

```
[ 0.00222222 -0.00148148 -0.00074074  0.01037037  0.12740741  0.0
0148148
  0.          0.        ]
```

In [43]:

```
# Making predictions
previsoes_nb = naive_bayes.predict (X_teste)
```

In [44]:

```
# Opening the found forecast values
previsoes_nb[:]
```

Out[44]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 1], dtype=int64)
```

In [45]:

```
previsoes_nb.shape
```

Out[45]:

```
(270,)
```

Model: Random Forest

In [46]:

```
# I create my object which receives the algorithm
floresta = RandomForestClassifier(n_estimators = 10000)

# Obs .: The parameter n_estimator shows the number of trees that my model wil
l use to classify
```

In [47]:

```
# 'Fit' my forest model with the training data.
floresta.fit(X_treinamento, y_treinamento)
```

Out[47]:

```
RandomForestClassifier(n_estimators=10000)
```

In [48]:

```
# Feature Importance - Random Forest Model
importances_rf = floresta.feature_importances_
importances_rf
```

Out[48]:

```
array([0.09548978, 0.11053139, 0.02232726, 0.15743981, 0.2963278
,
       0.11134233, 0.10302261, 0.10351902])
```

In [49]:

```
# Making Predictions
previsoes_rf = floresta.predict(X_teste)
previsoes_rf
```

Out[49]:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0,
       0, 0, 0, 0, 0, 0], dtype=int64)
```

Model: Decision Tree

In [50]:

```
arvore = DecisionTreeClassifier()
```

In [51]:

```
# Doing model learning
arvore.fit(X_treinamento, y_treinamento)
```

Out[51]:

```
DecisionTreeClassifier()
```

In [52]:

```
# Feature Importance - Decision Tree
imporrances_dt = arvore.feature_importances_
imporrances_dt
```

Out[52]:

```
array([0.09486311, 0.1399562 , 0.03286942, 0.15780677, 0.3331617
5,
       0.07970784, 0.10253896, 0.05909596])
```

In [53]:

```
# Making predictions
previsoes_ad = arvore.predict (X_teste)
previsoes_ad
```

Out[53]:

```
array([1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
0, 1, 0,
       1, 0, 1, 0, 0, 0], dtype=int64)
```

======================= Phase 5: Evaluation =======================

Now, it's the time to verify if the success criteria defined in first phase was accomplished. Here, I can use some metrics to check the performance of my model like ROC, mean square error and etc.

I create some functions to make tha code reusable. All functions can be checked below, before applying that in each model

In [54]:

```
def func_taxa_acerto(y, previsoes):
    '''
        Checking the hit rate
    '''
    return  accuracy_score(y, previsoes)

def func_confusion_matrix(y, previsoes):
    '''
        Using the confusion matrix to compare the values found or predicted wi
th the variable y_test
    '''
    conf_matrix =  confusion_matrix(y_teste, previsoes)
    return conf_matrix

def plot_confusion_matrix(model, X_trein, y_trein, X_test, y_test):
    '''
        I create the visualization object v using the yellowbrick library to m
ake the results more pleasant
    '''
    v = ConfusionMatrix(model)
    v.fit(X_trein, y_trein)
    v.score(X_test, y_test)
    return v.poof()

def metrics(y, previsoes):
    '''
        Checking the precision, recall, f1-score and support
    '''
    return classification_report(y, previsoes)

def roc(y_test, model, X_test):
    '''
        Checking the ROC curve
    '''
    logit_roc_auc = roc_auc_score(y_teste, model.predict(X_teste))
    fpr, tpr, thresholds = roc_curve(y_teste, model.predict_proba(X_teste)[:,1
])
    plt.figure()
    plt.plot(fpr, tpr, label='ROC Curve (area = %0.2f)' % logit_roc_auc)
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.savefig('Log_ROC')
    plt.show()
```

Model: Logistic Regression

In [55]:

```
# Checking the hit rate
#taxa_acerto_reglog = accuracy_score(y_teste, previsoes_reglog)
#taxa_acerto_reglog

print('Taxa de Acerto:', func_taxa_acerto(y_teste, previsoes_reglog))
```

```
Taxa de Acerto: 0.8444444444444444
```

In [56]:

```
# Checking the error rate
#taxa_erro_reglog = 1-taxa_acerto_reglog
#taxa_erro_reglog

print('Taxa de Erro:', 1-(func_taxa_acerto(y_teste, previsoes_reglog)))
```

```
Taxa de Erro: 0.15555555555555556
```

In [57]:

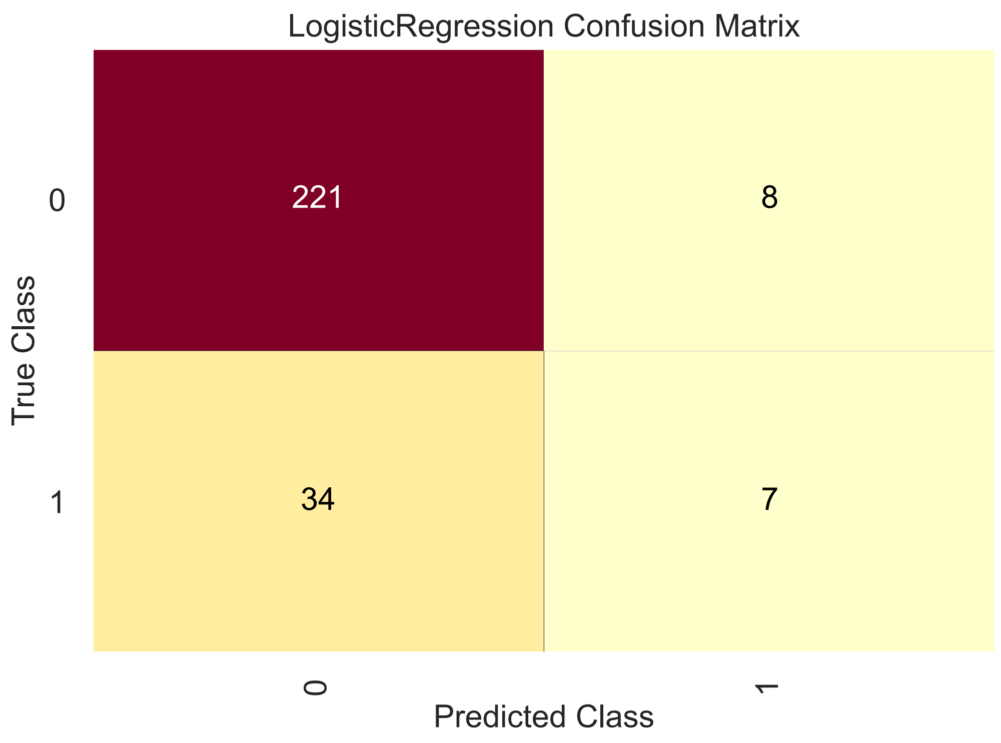
```
# Using the confusion matrix to compare the values found or predicted with the
variable y_test
#confusao_reglog = confusion_matrix(y_teste, previsoes_reglog)
#confusao_reglog

print('Confusion Matrix: ', func_confusion_matrix(y_teste, previsoes_reglog))
```

```
Confusion Matrix:  [[221   8]
 [ 34   7]]
```

In [58]:

```
# I create the visualization object v using the yellowbrick library to make th
e results more pleasant
#v = ConfusionMatrix(LogisticRegression())
#v.fit(X_treinamento, y_treinamento)
#v.score(X_teste, y_teste)
#v.poof()

print(plot_confusion_matrix(LogisticRegression(), X_treinamento, y_treinamento
, X_teste, y_teste))
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\base.py:209: FutureWarning: From version 0.24, get
_params will raise an AttributeError if a parameter cannot be ret
rieved as an instance attribute. Previously it would return None.
  warnings.warn('From version 0.24, get_params will raise an '
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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



```
AxesSubplot(0.125,0.125;0.775x0.755)
```

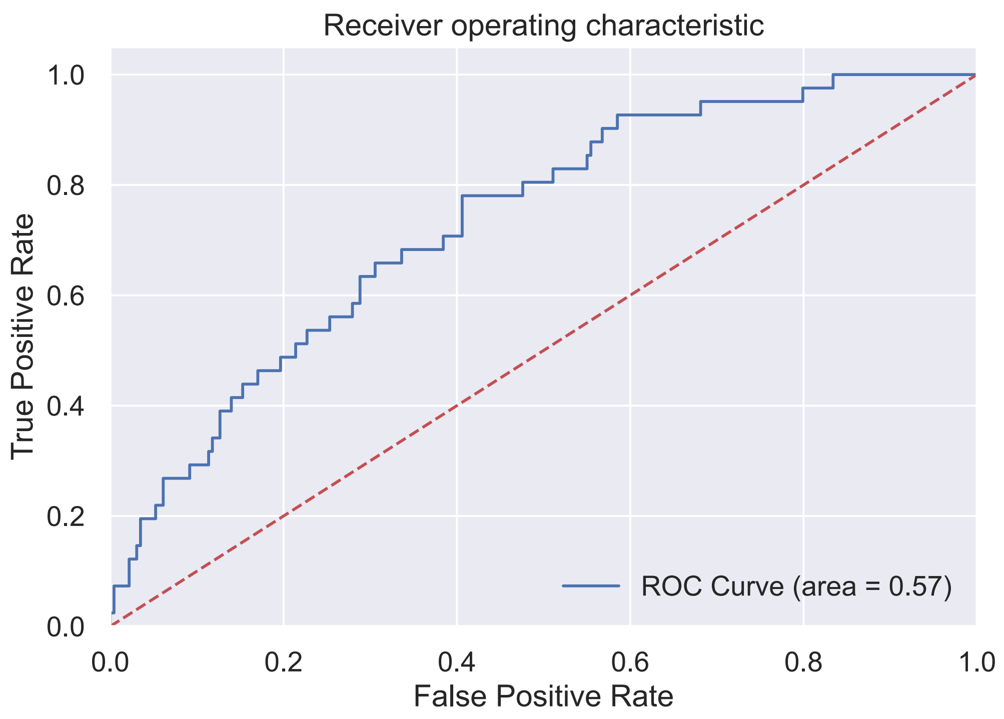
In [59]:

```
# Checking the precision, recall, f1-score and support
#print(classification_report(y_teste, previsoes_reglog))

print(metrics(y_teste, previsoes_reglog))
```

```
              precision    recall  f1-score   support

           0       0.87      0.97      0.91       229
           1       0.47      0.17      0.25        41

    accuracy                           0.84       270
   macro avg       0.67      0.57      0.58       270
weighted avg       0.81      0.84      0.81       270
```

In [60]:

```
# Checking the ROC curve
#logit_roc_auc = roc_auc_score(y_teste, reglog.predict(X_teste))
#fpr, tpr, thresholds = roc_curve(y_teste, reglog.predict_proba(X_teste)[:,1])
#plt.figure()
#plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_au
c)
#plt.plot([0, 1], [0, 1],'r--')
#plt.xlim([0.0, 1.0])
#plt.ylim([0.0, 1.05])
#plt.xlabel('False Positive Rate')
#plt.ylabel('True Positive Rate')
#plt.title('Receiver operating characteristic')
#plt.legend(loc="lower right")
#plt.savefig('Log_ROC')
#plt.show()

print(roc(y_teste, reglog, X_teste))
```



```
None
```

Model: Naive Bayes - Gaussian

In [61]:

```
# Checking the hit rate
print('Taxa de Acerto:', func_taxa_acerto(y_teste, previsoes_nb))
```

```
Taxa de Acerto: 0.9185185185185185
```

In [62]:

```
# Checking the error rate
print('Taxa de Erro:', 1-(func_taxa_acerto(y_teste, previsoes_nb)))
```

```
Taxa de Erro: 0.08148148148148149
```

In [63]:

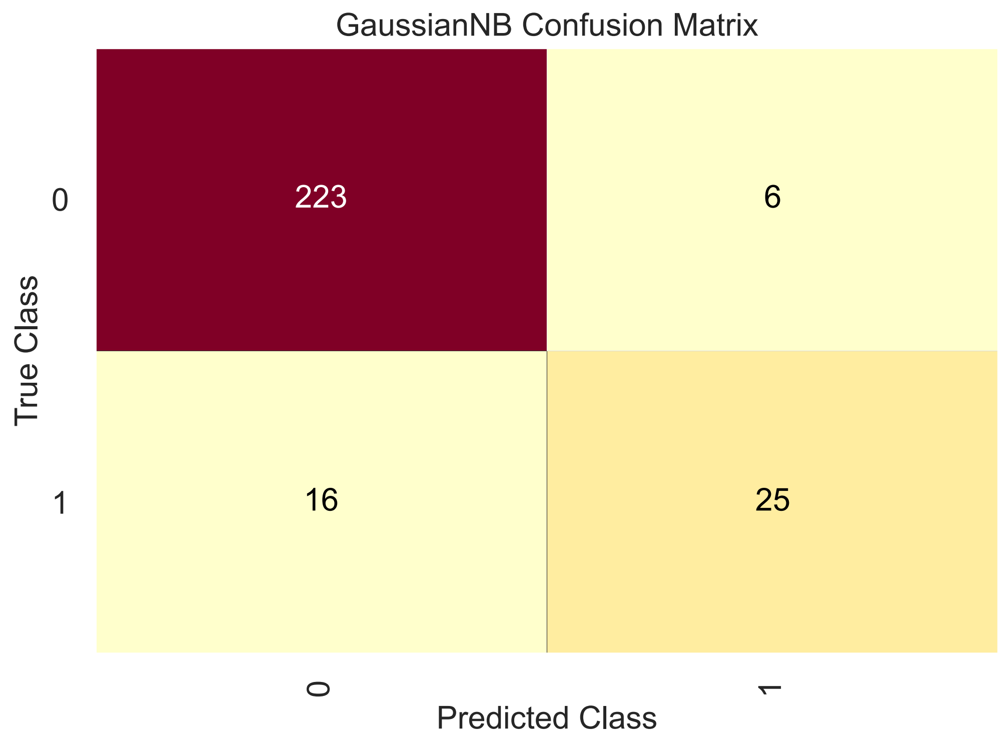
```
# Using the confusion matrix to compare the values found or predicted with the
variable y_test
print('Confusion Matrix: ', func_confusion_matrix(y_teste, previsoes_nb))
```

```
Confusion Matrix:  [[223   6]
 [ 16  25]]
```

In [64]:

```
# I create the visualization object v using the yellowbrick library to make th
e results more pleasant
print(plot_confusion_matrix(GaussianNB(), X_treinamento, y_treinamento, X_test
e, y_teste))
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\base.py:209: FutureWarning: From version 0.24, get
_params will raise an AttributeError if a parameter cannot be ret
rieved as an instance attribute. Previously it would return None.
  warnings.warn('From version 0.24, get_params will raise an '
```



```
AxesSubplot(0.125,0.125;0.775x0.755)
```

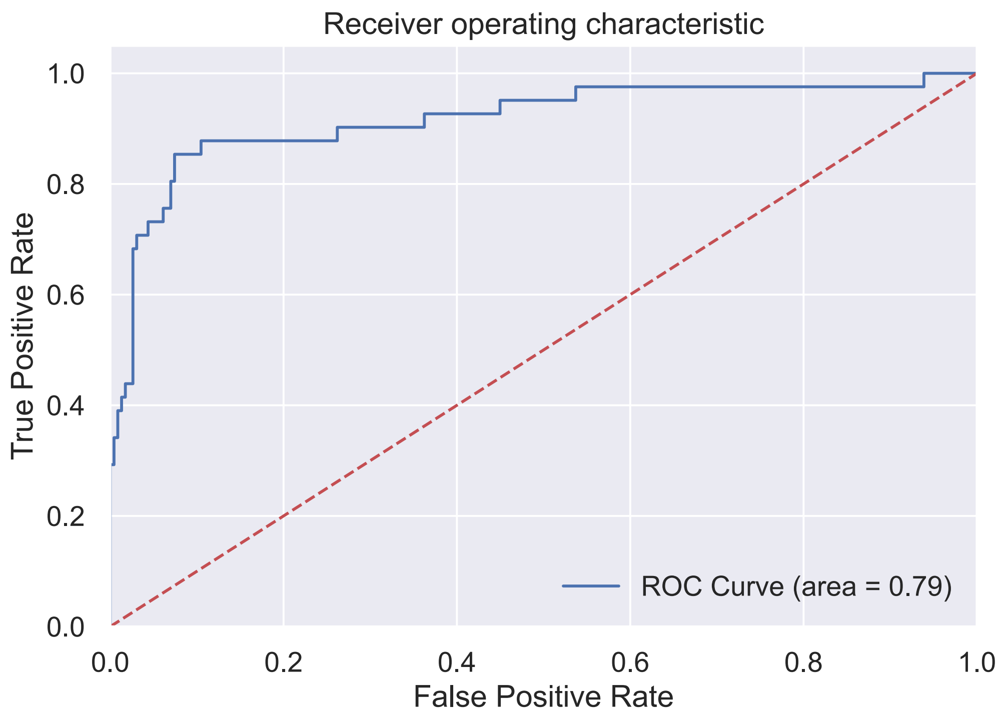
In [65]:

```
# Checking the precision, recall, f1-score and support
print(metrics(y_teste, previsoes_nb))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       229
           1       0.81      0.61      0.69        41

    accuracy                           0.92       270
   macro avg       0.87      0.79      0.82       270
weighted avg       0.91      0.92      0.91       270
```

In [66]:

```
# Checking the ROC curve
print(roc(y_teste, naive_bayes, X_teste))
```



```
None
```

Model: Random Forest

In [67]:

```
# Checking the hit rate
print('Taxa de Acerto:', func_taxa_acerto(y_teste, previsoes_rf))
```

```
Taxa de Acerto: 0.9
```

In [68]:

```python
# Checking the error rate
print('Taxa de Erro:', 1-(func_taxa_acerto(y_teste, previsoes_rf)))
```

```
Taxa de Erro: 0.09999999999999998
```

In [69]:

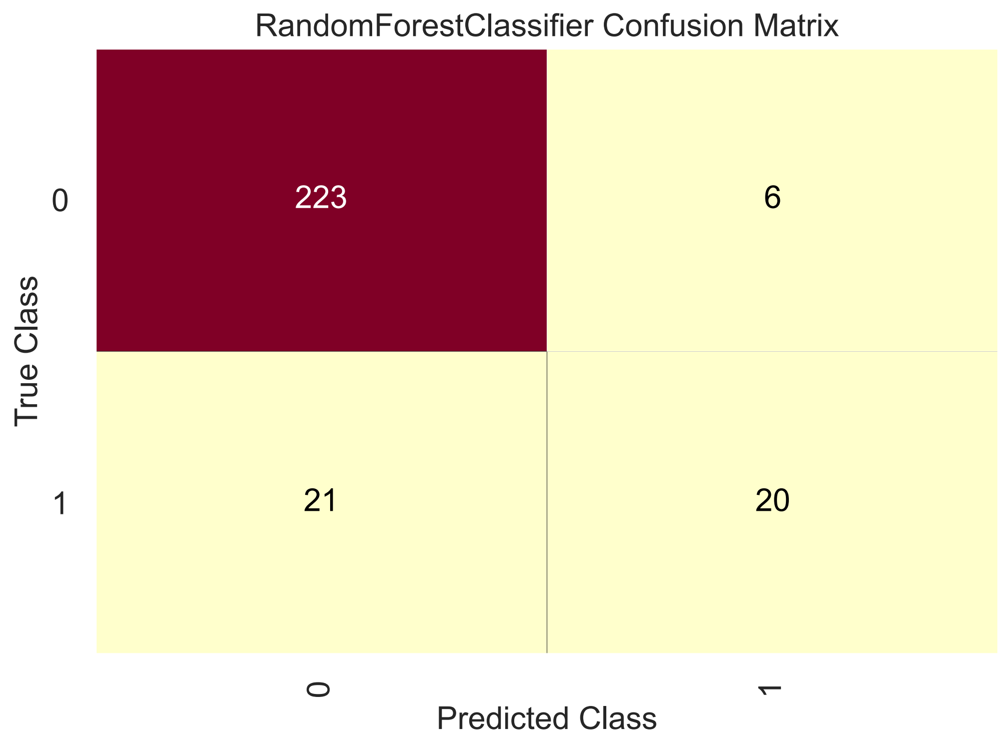
```python
# Confusion Matrix
print('Confusion Matrix: ', func_confusion_matrix(y_teste, previsoes_rf))
```

```
Confusion Matrix:  [[222   7]
 [ 20  21]]
```

In [70]:

```python
# Creating the preview v object
print(plot_confusion_matrix(RandomForestClassifier(), X_treinamento, y_treinam
ento, X_teste, y_teste))
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\base.py:209: FutureWarning: From version 0.24, get
_params will raise an AttributeError if a parameter cannot be ret
rieved as an instance attribute. Previously it would return None.
  warnings.warn('From version 0.24, get_params will raise an '
```



```
AxesSubplot(0.125,0.125;0.775x0.755)
```

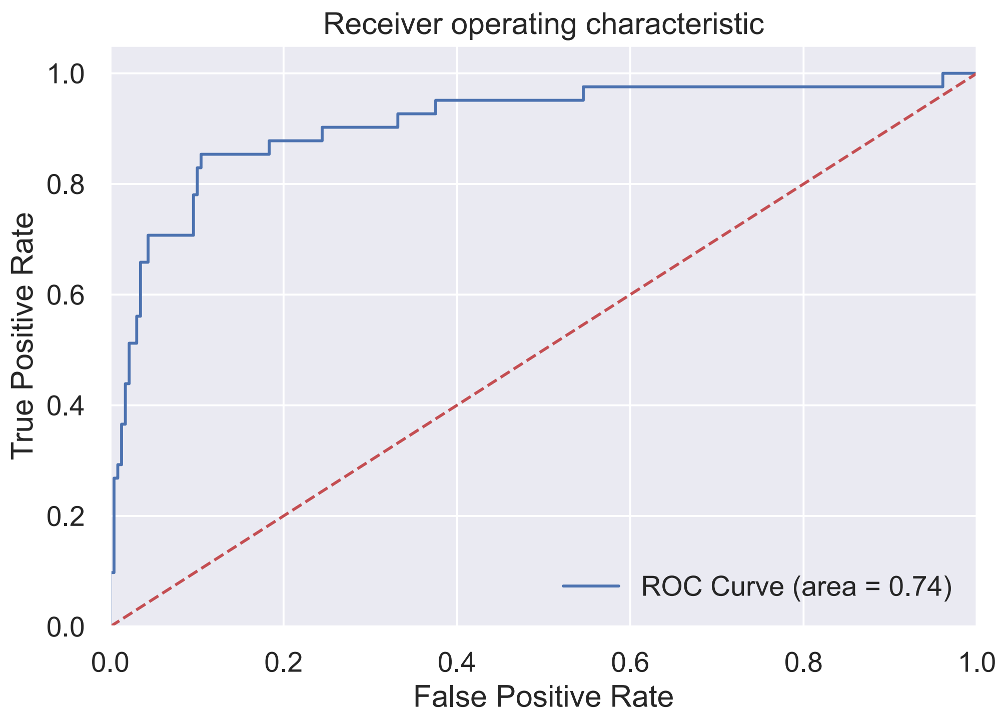
```
In [71]:
```

```
# Checking the precision, recall, f1-score and support
print(metrics(y_teste, previsoes_rf))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       229
           1       0.75      0.51      0.61        41

    accuracy                           0.90       270
   macro avg       0.83      0.74      0.78       270
weighted avg       0.89      0.90      0.89       270
```

```
In [72]:
```

```
# Checking the ROC curve
print(roc(y_teste, floresta, X_teste))
```



```
None
```

Model: Decision Tree

```
In [73]:
```

```
# Checking the hit rate
print('Taxa de Acerto:', func_taxa_acerto(y_teste, previsoes_ad))
```

```
Taxa de Acerto: 0.8185185185185185
```

In [74]:

```
# Checking the error rate
print('Taxa de Erro:', 1-(func_taxa_acerto(y_teste, previsoes_ad)))
```

```
Taxa de Erro: 0.18148148148148147
```

In [75]:

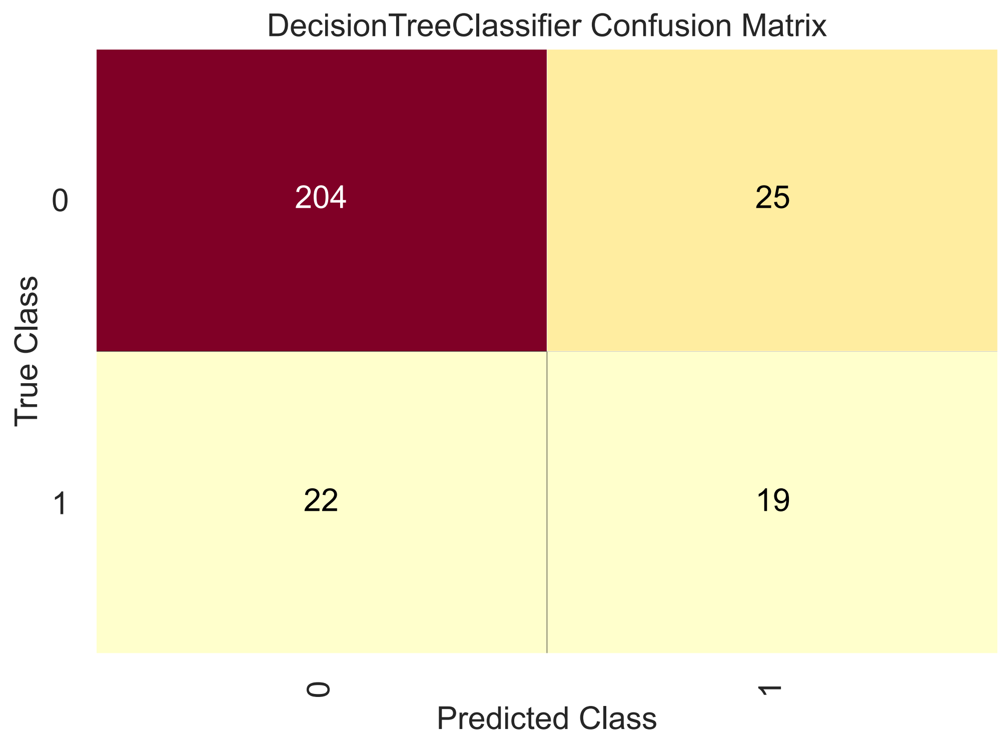
```
# Confusion Matrix
print('Confusion Matrix: ', func_confusion_matrix(y_teste, previsoes_ad))
```

```
Confusion Matrix:  [[203  26]
 [ 23  18]]
```

In [76]:

```
# I create the visualization v object
print(plot_confusion_matrix(DecisionTreeClassifier(), X_treinamento, y_treinam
ento, X_teste, y_teste))
```

```
C:\Users\hik_m\AppData\Local\Programs\Python\Python38\lib\site-pa
ckages\sklearn\base.py:209: FutureWarning: From version 0.24, get
_params will raise an AttributeError if a parameter cannot be ret
rieved as an instance attribute. Previously it would return None.
  warnings.warn('From version 0.24, get_params will raise an '
```



```
AxesSubplot(0.125,0.125;0.775x0.755)
```

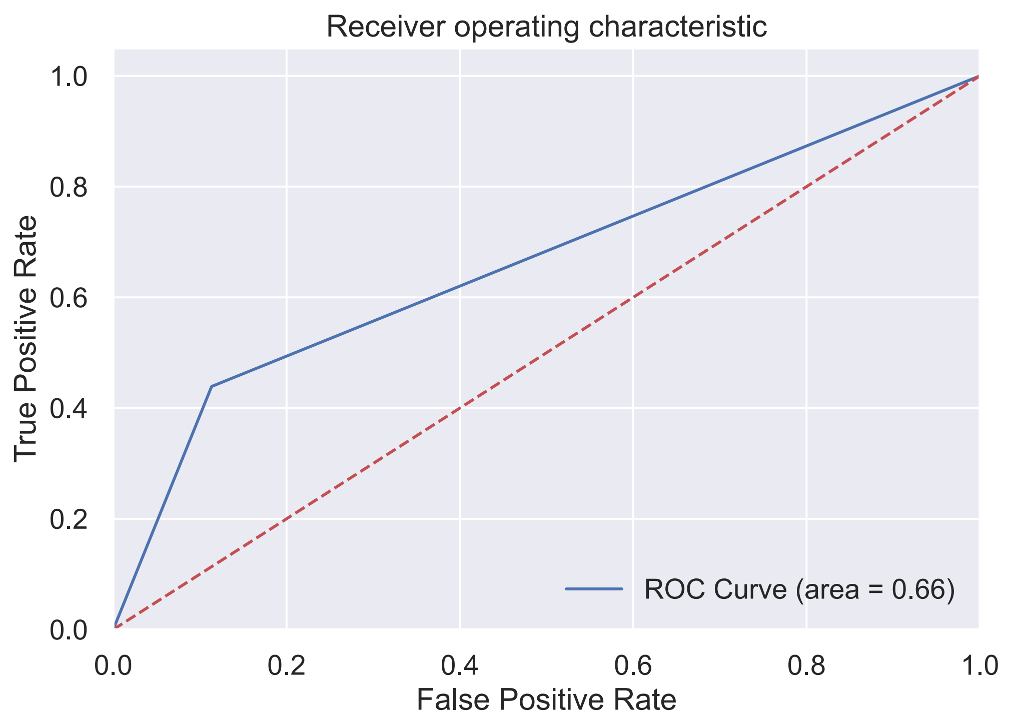
In [77]:

```
# Checking for precision, recall, f1-score and support
print(metrics(y_teste, previsoes_ad))
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.89       229
           1       0.41      0.44      0.42        41

    accuracy                           0.82       270
   macro avg       0.65      0.66      0.66       270
weighted avg       0.82      0.82      0.82       270
```

In [78]:

```
# Checking the ROC curve
print(roc(y_teste, arvore, X_teste))
```



```
None
```

===================== Phase 6: Deployement =====================

In this phase I can deploy my model in production environment and share my solution inside the company.
Important things here are:

- Monitoring the results
- Adapt the model when necessary

The last phase, phase 6 - Deploy, is where the implementation is made, it can be delivered in several ways, for example, creating an application for data consumption by those involved, making the model available through the persistence of the trained model using the Pickle library python and etc. For this project, only the development of the case study was generated. The availability for consumption of the data will be developed as a point to improve the structure of this project.

Conclusions

```
a) I used the train_test_split command to segregate training data and t
est data. I segregated the df dataset with 70% to do the models trainin
g and 30% (rest) to do tests.

b) I applied the Logistic Regression, Naive Bayes, Random Forest and De
cision Tree models and presented the results found with the predictions
found by the model.

c) The models used are models applied to classification in supervised l
earning.
    Objectively, the model:
        -Logistic Regression: Logistic regression is a resource that al
lows estimating the probability associated with the occurrence of a giv
en event in the face of a set of explanatory variables.
        -Naive Bayes: Generates a probability table. The classification
of the instance or record is done using this table.
        -Random Forest: Classifies the new rescuer based on several dec
ision trees. The amount I used was 10,000 trees.
        -Decision tree: Classifies the new record by going through each
node of the tree.

d) To compare and comment on the results obtained through the creation
 of the models, I used the confusion matrix, checking their correctness
and error rates, accuracy, recall, f1-score, support and ROC curve.

    Below I share a table with ranking of the models according to the h
it and error rates found:

        Rank    Model                   HIt Rate (TA)           TA(~)
Error Rate (TE)         TE(~)
        ----    --------------------    -------------------     -----
-----------------       -----
        1       Decision Trees          0,9185                  92%
0,0814                  08%
        2       Logistic Regression     0,9037                  90%
0,0962                  10%
        3       Random Forest           0,8444                  84%
0,1555                  16%
        4       Naive-Bayes             0,8296                  83%
0,1703                  17%

It can be observed that the Decision Trees model presented a higher hit
rate than the other models based on the data that were used. It is wort
h mentioning that some techniques could have been used, such as attribu
te selection, treatment of missing values, feature engineering, creatio
n of dummy variables and other approaches, making an exploratory analys
is of the data as well as modifying model parameters to achieve a bette
r fit.

     Another point of analysis was made through precision, recall, f1-s
core and support.
         * Accuracy is the classifier's ability to not label a sample a
```

```
s positive if it is negative.
         * The recall is the classifier's ability to find all positive
 samples
         * The F1-score can be interpreted as a harmonic average betwee
n precision and recall, where the best value is 1 and the worst is 0.
         * Support is the number of occurrences of the class in the y_t
est.

            Logistic Regression ---------------------------------
                          precision    recall  f1-score   support

                       0       0.87      0.97      0.91       229
                       1       0.47      0.17      0.25        41

                accuracy                           0.84       270
               macro avg       0.67      0.57      0.58       270
            weighted avg       0.81      0.84      0.81       270

            Naive Bayes -----------------------------------------
                          precision    recall  f1-score   support

                       0       0.93      0.97      0.95       229
                       1       0.81      0.61      0.69        41

                accuracy                           0.92       270
               macro avg       0.87      0.79      0.82       270
            weighted avg       0.91      0.92      0.91       270

            Random Forest ---------------------------------------
                          precision    recall  f1-score   support

                       0       0.92      0.97      0.94       229
                       1       0.76      0.54      0.63        41

                accuracy                           0.90       270
               macro avg       0.84      0.75      0.79       270
            weighted avg       0.90      0.90      0.90       270

            Decision Trees -----------------------------------
                          precision    recall  f1-score   support

                       0       0.90      0.90      0.90       229
                       1       0.44      0.44      0.44        41

                accuracy                           0.83       270
               macro avg       0.67      0.67      0.67       270
            weighted avg       0.83      0.83      0.83       270

        Looking at the table above, it can be seen that the Naive Bayes
model presented a higher accuracy rate (0.92 or 92%), however, as it wa
s verified through confusion, the Decision Tree model presented a highe
r accuracy rate.
```

```
        Finally, the ROC curve is another tool widely used with binary
 classifiers. The dotted line represents the ROC curve. A good classifi
er is as far away from that line as possible (towards the upper left co
rner). Thus, it can be observed that the ROC curve found in the Naive B
ayes model is what directs us that within the applied models, the Naive
Bayes model is the best model used in this approach.

        Despite the immense value created through the development of py
thon code to handle the data according to the purpose of each stage of
 the CRISP-DM methodology, the greatest value is to answer the question
s defined at the beginning of the project so that they can generate val
ue in decision making. Thus, the questions and respective answers of ea
ch one can be checked below:

            * Is it possible to achieve a hit rate above 80% in the cus
tomer's churn forecast?
              Following the steps of the CRISP-DM methodology, it was p
ossible to carry out an organized and evolutionary process for the deve
lopment of a data solution. Through these steps, it was possible to tra
in and test 4 different models where hit rates between 83% and 92% were
found.

            * What is the level of correlation between the variables ag
e, year and total_pucharse?
              Through step 2 of the CRISP-DM, a correlation was made be
tween the variables and a heatmp graph was generated through the seabor
n library, which presented a map of color and values in relation to the
variables used. In the reading done through it was possible to identify
a low correlation between the variables used in the model. It is import
ant to note that this pattern is possible and tends to benefit machine
 learning models since there is no skewed data.

            * Is the solution developed passive to be applied in a way
 that the business identified benefits?
```