SANTANDER DATA MASTER - CIENTISTA DE DADOS

Introdução: Análise Exploratória dos Dados e Balanceamento das bases

In [ ]:
```
!pip install matplotlib
!pip install seaborn
```

In [2]:
```
#Imports

#Manipulação dos Dados
import pandas as pd
import numpy as np

#Visualização dos Dados
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
from scipy import stats

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


plt.style.use("ggplot")
rcParams['figure.figsize'] = (12,  6)
```

Vamos começar importando as bases de treino e teste disponilizadas.

In [4]:
```
#Importar os arquivos de treino e teste
df_train = pd.read_csv('./train.csv')
df_test = pd.read_csv('./test.csv')

print(df_train.shape, df_test.shape)
```

```
(76020, 371) (75818, 370)
```

Verificando as informações de cada base.

In [5]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76020 entries, 0 to 76019
Columns: 371 entries, ID to TARGET
dtypes: float64(111), int64(260)
memory usage: 215.2 MB
```

In [6]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75818 entries, 0 to 75817
Columns: 370 entries, ID to var38
dtypes: float64(110), int64(260)
memory usage: 214.0 MB
```

Ao avaliar as suas informações:

- Todos os campos são númericos, sendo que a maioria apresenta tipo int64
- Percebe-se que há 371 na base de treino e 370 em teste, sendo a coluna TARGET a diferença das duas. A mesma será utilizada para identificar os clientes satisfeitos (0) e insatisfeitos (1).

Vamos analisar as principais descrições estatísticas das bases.

In [10]:
```
df_train.describe()
```

Out[10]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363067
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831
min	1.000000	-999999.000000	5.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000

8 rows × 371 columns

In [11]:
```
df_test.describe()
```

Out[11]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1
count	75818.000000	75818.000000	75818.000000	75818.000000	75818.000000
mean	75874.830581	-1579.955011	33.138832	83.164329	74.312894
std	43882.370827	39752.473358	12.932000	1694.873886	364.211245
min	2.000000	-999999.000000	5.000000	0.000000	0.000000
25%	37840.250000	2.000000	23.000000	0.000000	0.000000
50%	75810.000000	2.000000	27.000000	0.000000	0.000000
75%	113996.500000	2.000000	39.000000	0.000000	0.000000
max	151837.000000	238.000000	105.000000	240000.000000	21093.960000

8 rows × 370 columns

Logo de início, percebemos um outlier interessante na 'var3' de -999999. É um caso que trataremos mais adiante, já que o máximo dessa coluna é 238 e seus quartis apresentam o mesmo valor (2).

1. Verificar valores nulos

In [25]:
```
df_train.isnull().sum().sum()
```

Out[25]:
```
0
```

In [26]:
```
df_test.isnull().sum().sum()
```

Out[26]:
```
0
```

Não há ocorrências de valores nulos em nenhuma das bases

2. Verificar colunas com variância zero ou nula

In [7]:
```
#código para remover colunas com variância zero ou nula
i=0
for col in df_train.columns:
    if df_train[col].var()==0:
        i+=1
        del df_train[col]
        del df_test[col]
i, df_train.shape, df_test.shape
```

Out[7]:
```
(34, (76020, 337), (75818, 336))
```

Foram removidas 34 variáveis das duas bases.

3. Verificar colunas duplicadas

In [8]:
```
def getDuplicateColumns(df):

    duplicateColumnNames = set()
    for x in range(df.shape[1]):
        col = df.iloc[:, x]
        for y in range(x + 1, df.shape[1]):
            otherCol = df.iloc[:, y]
            if col.equals(otherCol):
                duplicateColumnNames.add(df.columns.values[y])
    return list(duplicateColumnNames)
```

In [9]:
```
duplicateColNames = getDuplicateColumns(df_train)
```

In [10]:
```
duplicateColNames, len(duplicateColNames)
```

Out[10]:
```
(['ind_var29',
  'ind_var13_medio',
  'ind_var32',
  'num_var34',
  'num_var25',
  'ind_var25',
  'delta_num_reemb_var13_1y3',
  'delta_num_trasp_var33_out_1y3',
  'ind_var34',
  'delta_num_trasp_var33_in_1y3',
  'num_var13_medio',
  'delta_num_trasp_var17_out_1y3',
  'ind_var18',
  'num_var39',
  'delta_num_reemb_var33_1y3',
  'num_var37',
  'ind_var29_0',
  'saldo_var29',
  'num_var29',
  'ind_var39',
  'saldo_medio_var13_medio_ult1',
  'delta_num_trasp_var17_in_1y3',
  'num_var18',
  'num_var32',
  'num_var29_0',
  'delta_num_reemb_var17_1y3',
  'ind_var37',
  'ind_var26',
  'num_var26'],
 29)
```

Foram encontradas 29 colunas duplicadas na base de treino.

In [11]:
```
df_train = df_train.drop(columns = getDuplicateColumns(df_train))
```

In [12]:
```
df_train.head()
```

Out[12]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_c
0	1	2	23	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	
3	8	2	37	0.0	195.0	195.0	
4	10	2	39	0.0	0.0	0.0	

5 rows × 308 columns

Retirando as colunas duplicadas, ficamos com 308. Vamos remover na base de teste.

In [13]:
```
df_test = df_test.drop(columns = duplicateColNames)
df_test
```

Out[13]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ul
0	2	2	32	0.0	0.0	0
1	5	2	35	0.0	0.0	0
2	6	2	23	0.0	0.0	0
3	7	2	24	0.0	0.0	0
4	9	2	23	0.0	0.0	0
...	...	...	...	...	...	
75813	151831	2	23	0.0	0.0	0
75814	151832	2	26	0.0	0.0	0
75815	151833	2	24	0.0	0.0	0
75816	151834	2	40	0.0	0.0	0
75817	151837	2	23	0.0	0.0	0

75818 rows × 307 columns

4. Verificar colunas dispersas

In [14]:
```
#removing sparse features
i=0
for col in df_train.columns: #reomving all sparse features
    if np.percentile(df_train[col],99)==0:
        i+=1
        del df_train[col]
        del df_test[col]
```

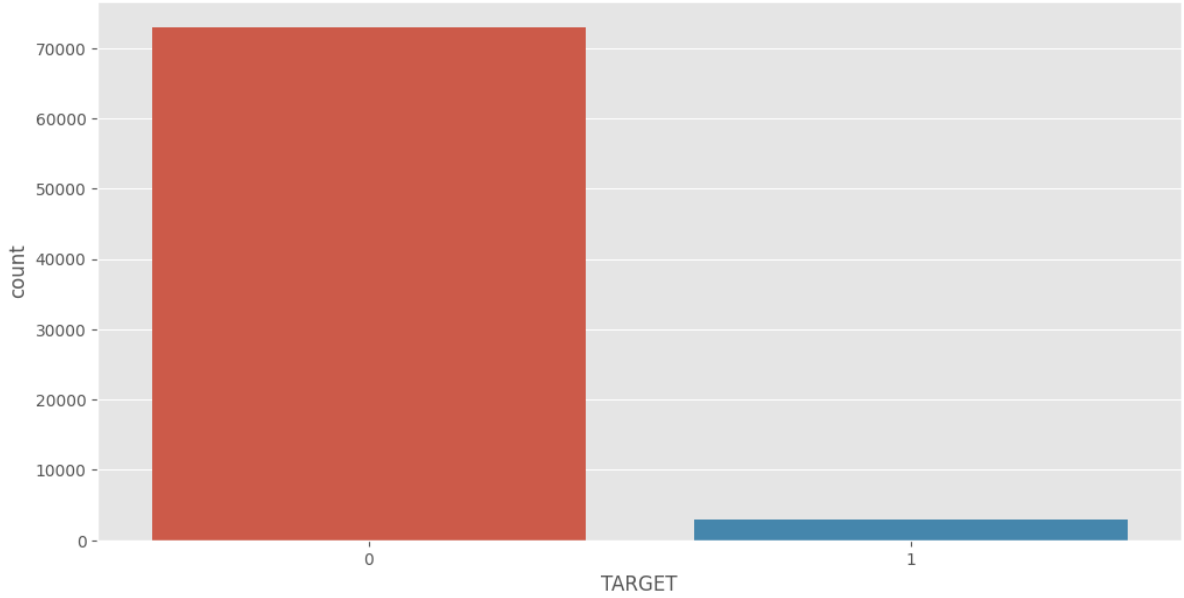
In [15]:
```
df_train.shape, df_test.shape, i
```

Out[15]:
```
((76020, 143), (75818, 142), 165)
```

Removemos 165 variáveis dispersas, agora a quantidade de colunas está mais enxuta! Mas vamos continuar.

5. Entendendo a coluna TARGET

In [17]:
```
sns.countplot(x=df_train['TARGET']);
```



In [18]:
```
np.unique(df_train['TARGET'], return_counts=True)
```

Out[18]:
```
(array([0, 1], dtype=int64), array([73012,  3008], dtype=int64))
```

In [19]:
```
(df_train['TARGET'].value_counts()/df_train['TARGET'].count())*100
```

Out[19]:
```
0    96.043147
1     3.956853
Name: TARGET, dtype: float64
```

Na base de treino, percebe-se que 96% dos registros são de clientes satisfeitos (1). Isso é um ótimo sinal já que indica que o Santander possui um bom atendimento, refletido no alto percentual de satisfação. E temos 4% de clientes insatisfeitos, que será nosso foco de identificá-los para tomarmos as ações de retenção. Outro ponto identificado é o desbalanceamento da base, logo vamos avaliar um processo (undersampling ou oversampling) para equalizar os registros e aprimorar nosso algoritmo.

6. Analisando variáveis específicas

Coluna var3

Como dito anteriormente, essa coluna possui indícios de outliers. Vamos analisar os registros únicos:

In [16]:
```
np.array(sorted(df_train.var3.unique()))
```

Out[16]:
```
array([-999999,       0,       1,       2,       3,       4,       5,
             6,       7,       8,       9,      10,      11,      12,
            13,      14,      15,      16,      17,      18,      19,
            20,      21,      22,      23,      24,      25,      26,
            27,      28,      29,      30,      31,      32,      33,
            34,      35,      36,      38,      40,      41,      42,
            43,      44,      45,      46,      47,      48,      49,
            50,      51,      52,      53,      54,      55,      56,
            57,      58,      59,      60,      61,      62,      63,
            64,      65,      66,      68,      69,      71,      72,
            73,      74,      76,      77,      78,      79,      81,
            82,      84,      85,      86,      87,      88,      89,
            90,      91,      93,      94,      95,      96,      97,
            98,      99,     100,     101,     102,     103,     104,
           105,     106,     107,     108,     110,     111,     112,
           114,     115,     116,     117,     118,     119,     120,
           121,     122,     124,     125,     126,     127,     128,
           129,     130,     131,     132,     133,     134,     135,
           136,     137,     138,     139,     141,     142,     143,
           144,     145,     146,     147,     148,     149,     150,
           151,     152,     153,     154,     156,     157,     158,
           159,     161,     162,     163,     164,     165,     166,
           167,     168,     169,     170,     171,     172,     173,
           174,     175,     176,     177,     178,     181,     182,
           183,     184,     185,     186,     187,     188,     189,
           190,     191,     192,     193,     194,     195,     196,
           197,     198,     199,     200,     201,     204,     205,
           207,     208,     209,     210,     211,     213,     215,
           216,     217,     218,     219,     220,     223,     225,
           228,     229,     231,     235,     238], dtype=int64)
```

In [17]:
```
print("O número de valores únicos em var3 é %i"%(len(np.array(sorted(df_train.var3.
```

```
O número de valores únicos em var3 é 208
```

In [18]:
```
df_test['var3'].value_counts()[:5]
```

```
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\233122141.py:1: FutureWarning: T
he behavior of `series[i:j]` with an integer-dtype index is deprecated. In a futur
e version, this will be treated as *label-based* indexing, consistent with e.g. `s
eries[i]` lookups. To retain the old behavior, use `series.iloc[i:j]`. To get the
future behavior, use `series.loc[i:j]`.
  df_test['var3'].value_counts()[:5]
```

Out[18]:
```
 2         73962
-999999      120
 8           116
 9           108
 13          107
Name: var3, dtype: int64
```

Aqui podemos ver que valores exclusivos variam de 0 a 238 com exceção sendo -999999 (pode estar faltando valor). Isso pode indicar nacionalidade/região para um cliente específico, pois 208 é um número razoável para uma empresa global como o Santander estar presente. Isso foi mais evidenciado da literatura revisada.

In [19]:
```
print("A maior ocorrência em var3 (nacionalidade) na base de treino é o 2 e represe
                                                                                    d
print("A maior ocorrência em var3 (nacionalidade) na base de teste é o 2 e represen
                                                                                    d
```

```
A maior ocorrência em var3 (nacionalidade) na base de treino é o 2 e representa 74
165 (97.56%).
A maior ocorrência em var3 (nacionalidade) na base de teste é o 2 e representa 739
62 (97.55%).
```

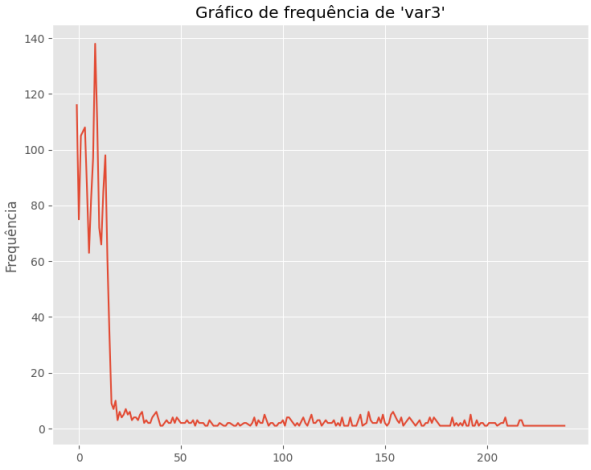
In [20]:
```
missing = dict(df_train['var3'].value_counts())[-999999]*100/df_train.shape[0]
missing_ = dict(df_test['var3'].value_counts())[-999999]*100/df_test.shape[0]
print("O percentual que possuem valores extremos em var3 na base de treino e teste
```

```
O percentual que possuem valores extremos em var3 na base de treino e teste são 0.
153%, 0.158% respectivamente
```

In [21]:
```
def valuecounts_plot(col,train=df_train,test=df_test):
    """
    plots a frequency of occurence vs value line plot for a specific column for bot
    Also prints the top 5 values with highest percenetage occurence
    """
    plt.subplots(1,2,figsize=(15,6))
    #plotting frequency counts for train
    plt.subplot(121)
    df = train[col].value_counts().sort_index()
    sns.lineplot(x=df.index,y=df.values)
    plt.title("Gráfico de frequência de '%s'"%(col))
    plt.ylabel('Frequência')

    #plotting frequency counts for test
    plt.subplot(122)
    df = test[col].value_counts().sort_index()
    sns.lineplot(x=df.index,y=df.values)
    plt.title("Gráfico de frequência de '%s'"%(col))
    plt.ylabel('Frequência')

    plt.tight_layout()
    plt.show()
    print("*"*100)
    print("Valor percentual (5 maiores) na base de treino para '%s':"%(col))
    print("Valor\t Perc%")
    print((train[col].value_counts()*100/train.shape[0]).iloc[:5])
    print("*"*100)
    print("Valor percentual (5 menores) na base de treino para '%s':"%(col))
    print("Valor\t Perc%")
    print((train[col].value_counts()*100/train.shape[0]).iloc[-5:])

    print("*"*100)
    print("Valor percentual (5 maiores) na base de teste para '%s':"%(col))
    print("Valor\t Perc%")
    print((test[col].value_counts()*100/test.shape[0]).iloc[:5])
    print("*"*100)
    print("Valor percentual (5 menores) na base de teste para '%s':"%(col))
    print("Valor\t Perc%")
    print((test[col].value_counts()*100/test.shape[0]).iloc[-5:])
```

Vamos substituir o valor de -999999 para -1 e verificar a frequência de ocorrência de cada valor (excepto o 2).

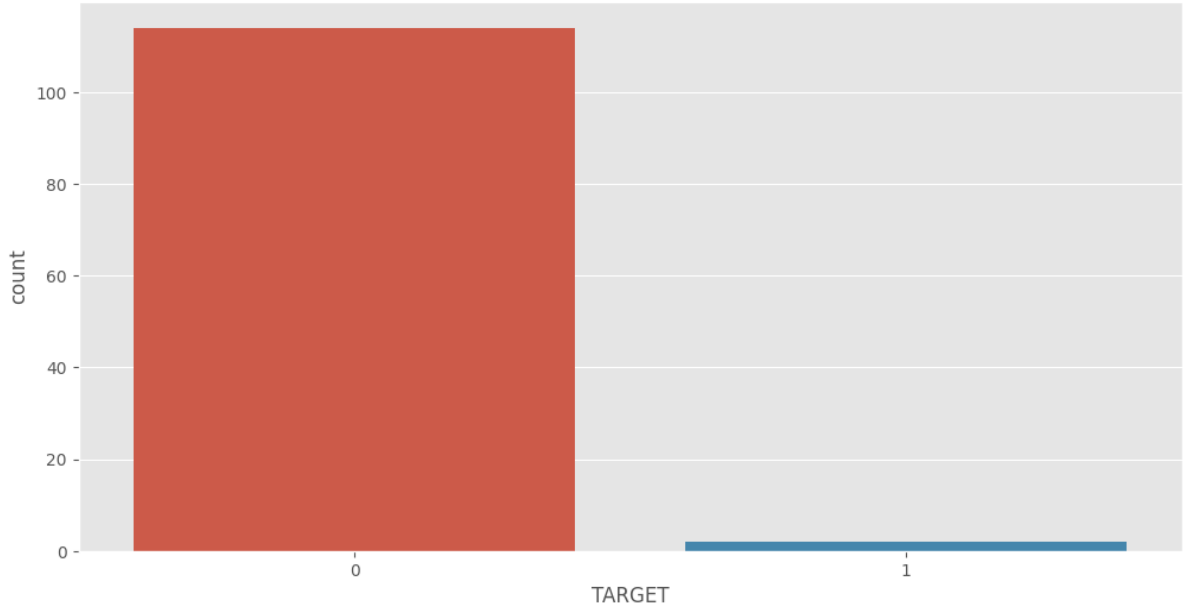
In [22]:
```
df_train['var3'].replace(-999999,-1,inplace=True)
df_test['var3'].replace(-999999,-1,inplace=True)
#plotting frequency with no 2 value for var3 and value_counts with percentage
valuecounts_plot('var3',df_train[ df_train['var3']!=2 ],df_test[ df_test['var3']!=2
```



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'var3':
Valor    Perc%
 8    7.439353
-1    6.253369
 9    5.929919
 3    5.822102
 1    5.660377
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'var3':
Valor    Perc%
188    0.053908
168    0.053908
135    0.053908
159    0.053908
87     0.053908
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'var3':
Valor    Perc%
-1     6.465517
 8     6.250000
 9     5.818966
 3     5.765086
 13    5.765086
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'var3':
Valor    Perc%
199    0.053879
101    0.053879
57     0.053879
134    0.053879
225    0.053879
Name: var3, dtype: float64
```

Com os gráficos, percebe-se muita similaridade nas bases de treino e teste.

In [23]:
```
mask = df_train[df_train['var3']==-1]
sns.countplot(x=mask['TARGET']);
```



In [24]:
```
(mask['TARGET'].value_counts()/mask['TARGET'].count())*100
```

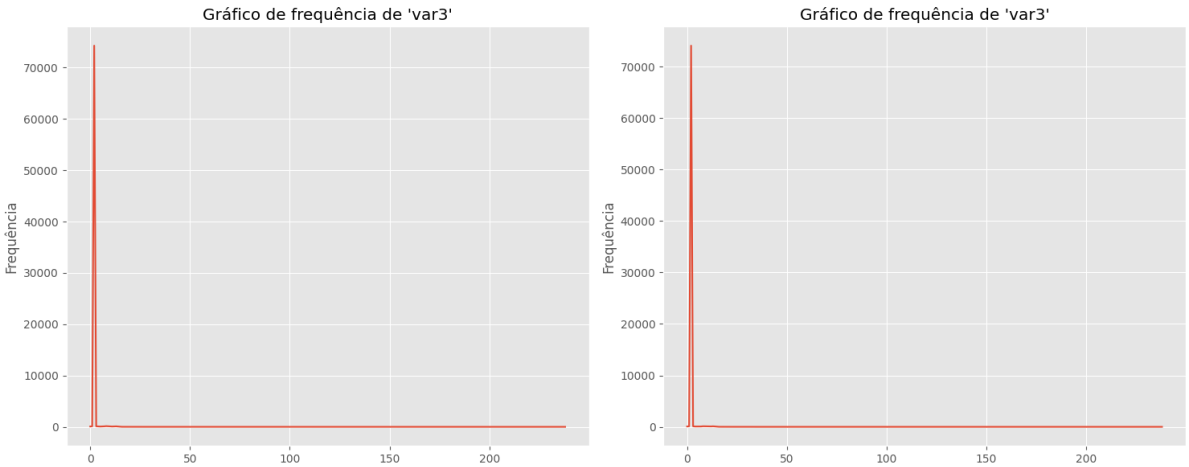
Out[24]:
```
0    98.275862
1     1.724138
Name: TARGET, dtype: float64
```

O número de clientes insatisfeitos nessa amostra são 2. O número de clientes satisfeitos nessa amostra são 114.

Podemos substituir todos os valores -1 por 2 (categoria mais frequente neste recurso) para treinar e testar, pois a partir dos dados de treino também há clientes insatisfeitos e também o número total é muito pequeno

In [25]:
```
df_train['var3'].replace(-1,2,inplace=True)
df_test['var3'].replace(-1,2,inplace=True)
```

In [26]:
```
valuecounts_plot('var3',df_train,df_test)
```



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'var3':
Valor    Perc%
2    97.712444
8     0.181531
9     0.144699
3     0.142068
1     0.138122
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'var3':
Valor    Perc%
231    0.001315
188    0.001315
168    0.001315
135    0.001315
87     0.001315
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'var3':
Valor    Perc%
2     97.710306
8      0.152998
9      0.142446
13     0.141127
3      0.141127
Name: var3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'var3':
Valor    Perc%
199    0.001319
101    0.001319
57     0.001319
134    0.001319
225    0.001319
Name: var3, dtype: float64
```

Coluna Var15

In [27]:
```
max_ = df_train['var15'].max()
min_ = df_train['var15'].min()
print("O valor mínimo de var15 é %i e valor máximo de var15 é %i."%(min_,max_ ))
```

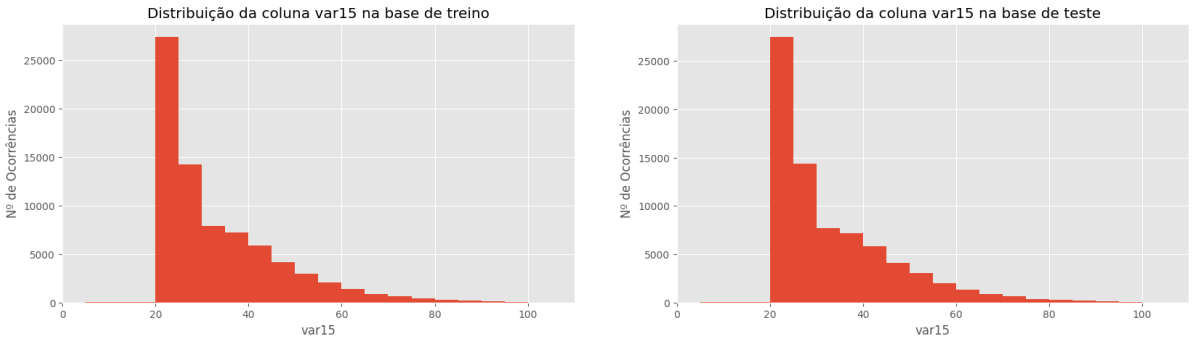
```
O valor mínimo de var15 é 5 e valor máximo de var15 é 105.
```

Como o valor de var15 é de 5 a 105, podemos assumir que esse recurso talvez indique a idade do cliente.

In [28]:
```
def histplot_comb(col,train=df_train,test=df_test,size=(20,5),bins=20):
    """
    Creates a histplots of train and
```

```
    test data for feature var
    """
    var=col
    plt.subplots(1,2,figsize=size)
    #plotting train data
    plt.subplot(121)
    plt.title("Distribuição da coluna {} na base de treino".format(var))
    plt.ylabel('Nº de Ocorrências')
    plt.xlabel(var)
    plt.hist(train[var],bins=bins)
    #plotting test data
    plt.subplot(122)
    plt.title("Distribuição da coluna {} na base de teste".format(var))
    plt.ylabel('Nº de Ocorrências')
    plt.xlabel(var)
    plt.hist(test[var],bins=bins)

    plt.show()
    print("")
```

In [29]:
```
#plotting histplot for train and test data
histplot_comb('var15')


#https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.percentileofscore
percentile = stats.percentileofscore(df_train['var15'].values,30)
print("Clientes cuja idade é abaixo de 30 constitui cerca de %.2f%% da base de trei
print("A idade mínima na base de treino é %i enquanto a idade máxima é %i."%(df_tra
print("")
percentile = stats.percentileofscore(df_test['var15'].values,30)
print("Clientes cuja idade é abaixo de 30 constitui cerca de %.2f%% da base de test
print("A idade mínima na base de teste é %i enquanto a idade máxima é %i."%(df_test
```

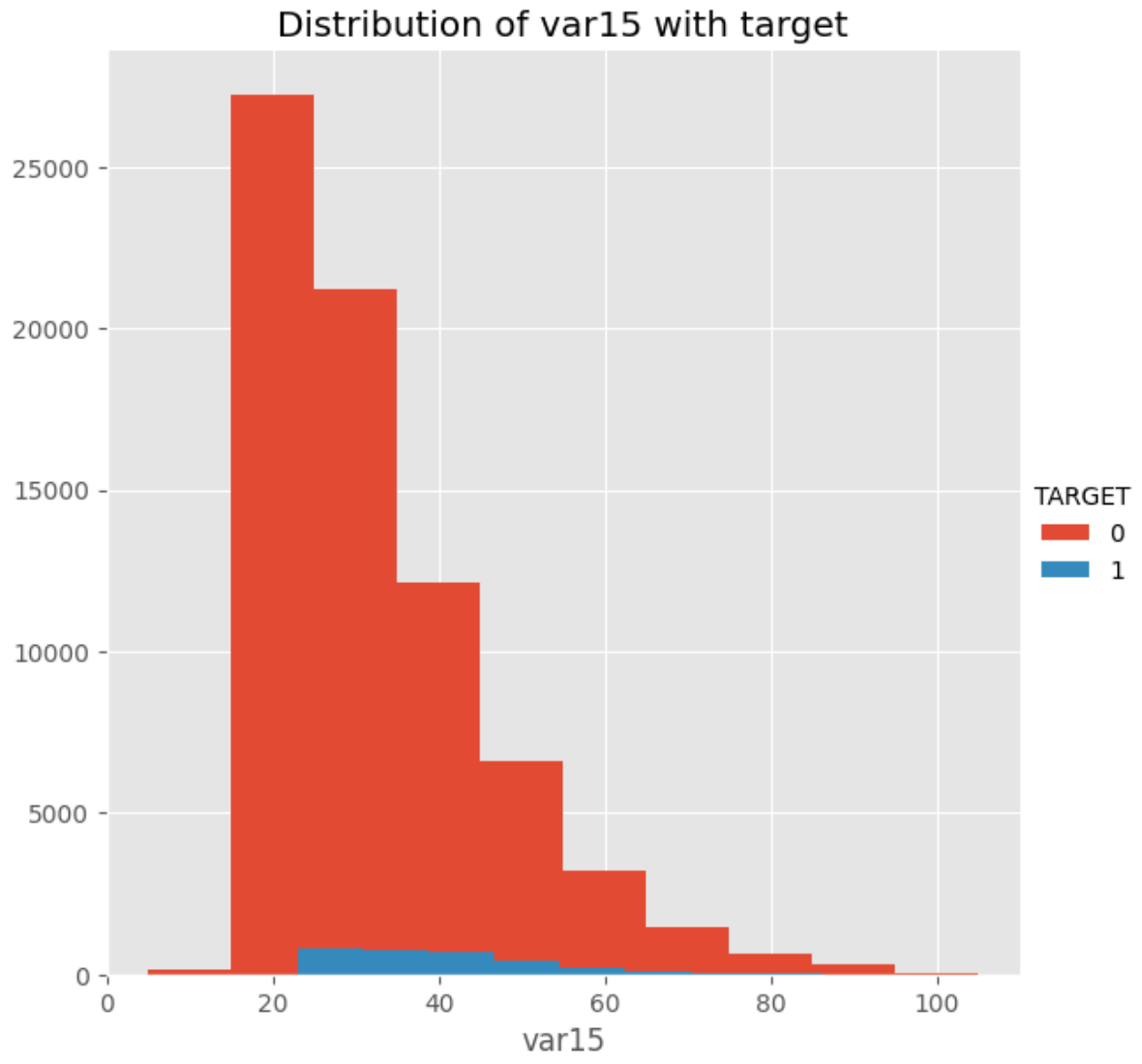


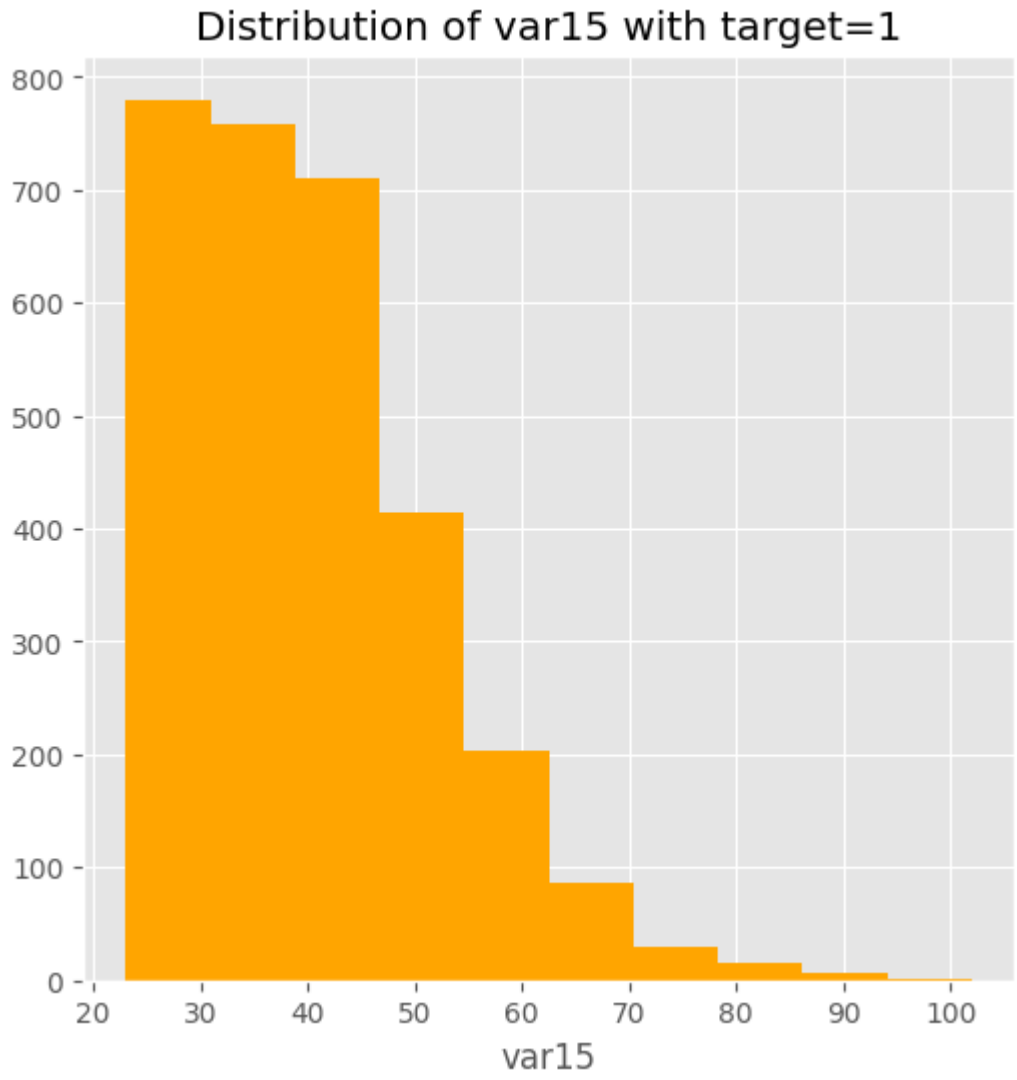

```
Clientes cuja idade é abaixo de 30 constitui cerca de 56.15% da base de treino.
A idade mínima na base de treino é 5 enquanto a idade máxima é 105.

Clientes cuja idade é abaixo de 30 constitui cerca de 56.58% da base de teste.
A idade mínima na base de teste é 5 enquanto a idade máxima é 105.
```

Ambos os dados de treino e teste tiveram distribuições semelhantes, ambos consistindo em clientes mais jovens. Agora vamos ver se todos os mais jovens estão satisfeitos.

In [30]:
```
#plotting the distribution with target
sns.FacetGrid(data=df_train,hue='TARGET',height=6).map(plt.hist,'var15').add_legend
plt.title("Distribution of var15 with target")
```

```
plt.show()

print("")
print("")

mask = df_train[(df_train['TARGET']==1)]
plt.figure(figsize=(6,6))
plt.hist(mask['var15'],color='orange')
plt.title("Distribution of var15 with target=1")
plt.xlabel('var15')
plt.show()

max_ = mask['var15'].max()
min_ = mask['var15'].min()
print("The minimum age of the unsatisfied customer is %i and maximum age of the uns
```






```
The minimum age of the unsatisfied customer is 23 and maximum age of the unsatisfi
ed customer is 102 .
```

Podemos ver no diagrama acima que a maioria dos jovens está satisfeita. Assim, podemos criar uma nova coluna que informa se um cliente tem menos de 23 anos ou não.

In [31]:
```
#create a new feature which tells whether a customer is below 23 years old or not
for df in [df_train,df_test]:
    df['var15_below_23'] = np.zeros(df.shape[0],dtype=int)
    df.loc[df['var15']<23,'var15_below_23']=1
```

In [32]:
```
new_feature = ['var15_below_23']
#binning age feature into 5 bins
_,bins = pd.cut(df_train['var15'].values,5,retbins=True) #getting the bins
print(_ )
```

```
[(4.9, 25.0], (25.0, 45.0], (4.9, 25.0], (25.0, 45.0], (25.0, 45.0], ..., (45.0, 6
5.0], (25.0, 45.0], (4.9, 25.0], (4.9, 25.0], (45.0, 65.0]]
Length: 76020
Categories (5, interval[float64, right]): [(4.9, 25.0] < (25.0, 45.0] < (45.0, 65.
0] < (65.0, 85.0] < (85.0, 105.0]]
```

Esses serão os bins que definiremos.

In [33]:
```
#converting var15 values to bin values
df_train['var15'] = pd.cut(df_train['var15'].values,bins,labels=False)
df_test['var15'] = pd.cut(df_test['var15'].values,bins,labels=False)
```

In [34]:
```
#plotting the binned feature
sns.FacetGrid(data=df_train,hue='TARGET',height=6).map(plt.hist,'var15').add_legend
plt.title("Distribution of var15 with target after binning")
plt.show()
```

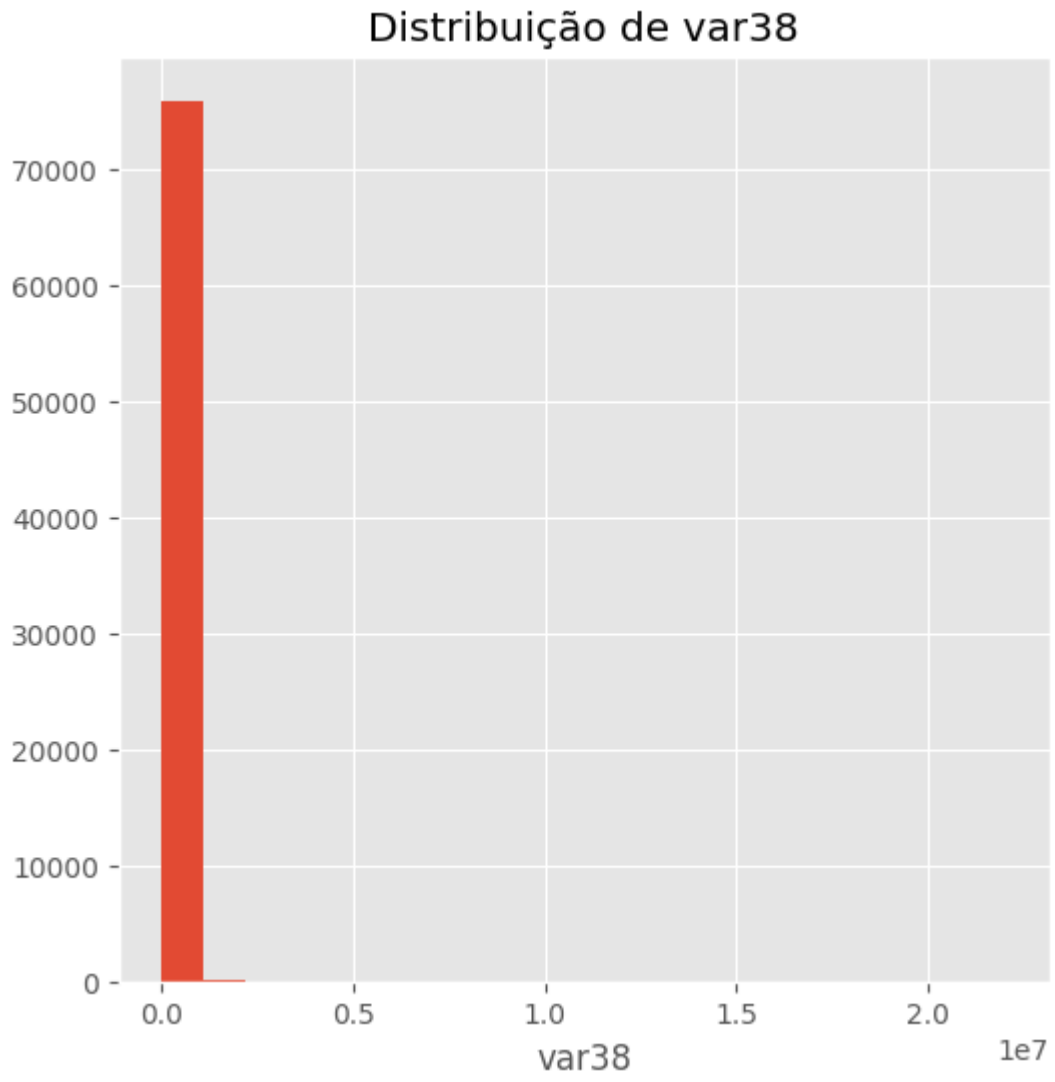
Coluna var38

In [35]:
```
print("O valor mínimo de var38 encontrado foi %.3f e o valor máximo de var38 é %.3f
```

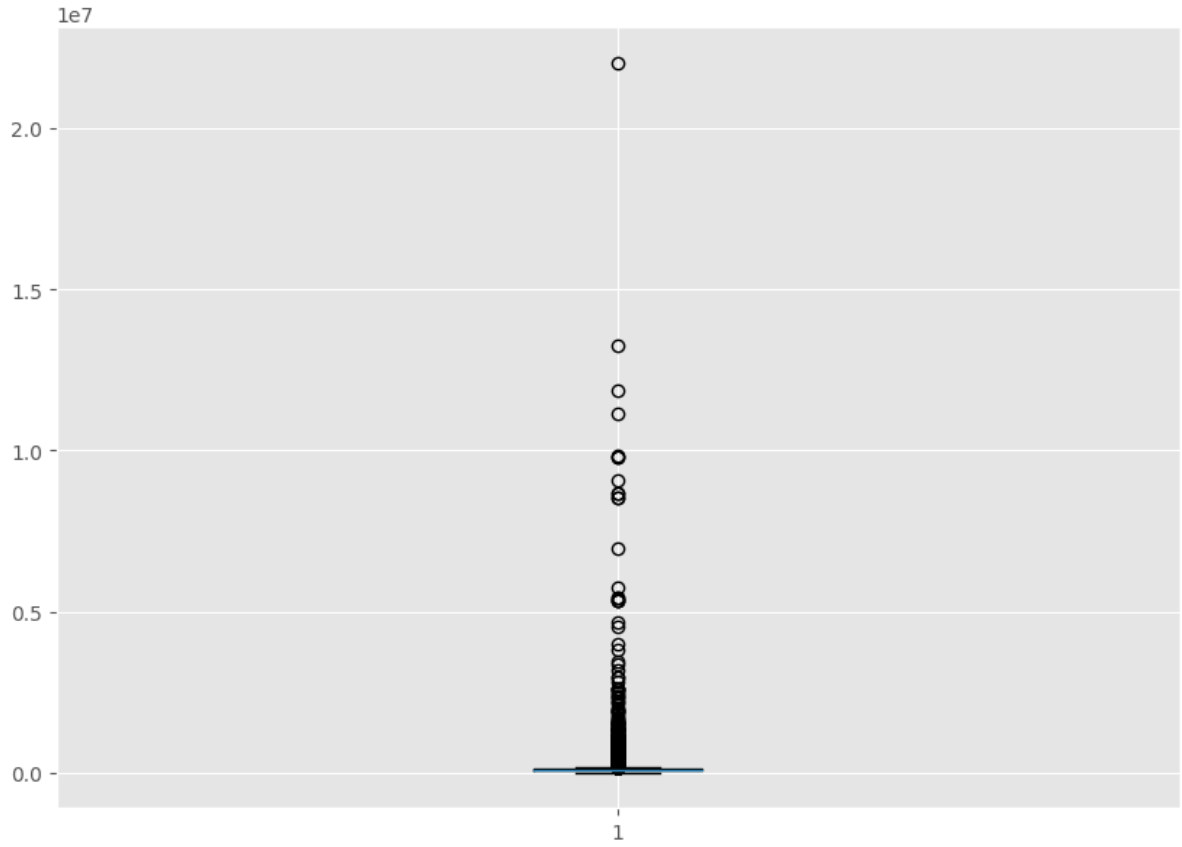
```
O valor mínimo de var38 encontrado foi 5163.750 e o valor máximo de var38 é 220347
38.760
```

In [36]:
```
plt.figure(figsize=(6,6))
plt.hist(df_train['var38'],bins=20)
plt.title("Distribuição de var38")
```

```
plt.xlabel('var38')
plt.show()
```



In [37]:
```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(df_train['var38'])

# show plot
plt.show()
```



In [38]:
```
df_train.var38.value_counts()
```

Out[38]:
```
117310.979016    14868
451931.220000       16
463625.160000       12
288997.440000       11
104563.800000       11
                 ...
89665.500000         1
45876.570000         1
151505.640000        1
74548.170000         1
84278.160000         1
Name: var38, Length: 57736, dtype: int64
```

Aqui não podemos obter nenhuma informação, pois um valor está tendo uma frequência de distribuição muito alta. Vamos imprimir os valores de cada percentil.

In [39]:
```
for i in np.arange(0,1.1,0.1):
    print('%i percentil : %i'%(i*100,np.quantile(df_train.var38.values,i)))
```

```
0 percentil : 5163
10 percentil : 48070
20 percentil : 61496
30 percentil : 74152
40 percentil : 88571
50 percentil : 106409
60 percentil : 117310
70 percentil : 117310
80 percentil : 132859
90 percentil : 182585
100 percentil : 22034738
```

Ahaa! Podemos ver que há uma enorme diferença entre o valor do percentil 0 e o valor do percentil 10. Este é o mesmo caso para o valor do percentil 90 e o valor do percentil 100.

In [40]:
```
for i in np.arange(0,0.11,0.025):
    print('%.2f percentile : %i'%(i*100,np.quantile(df_train.var38.values,i)))
```

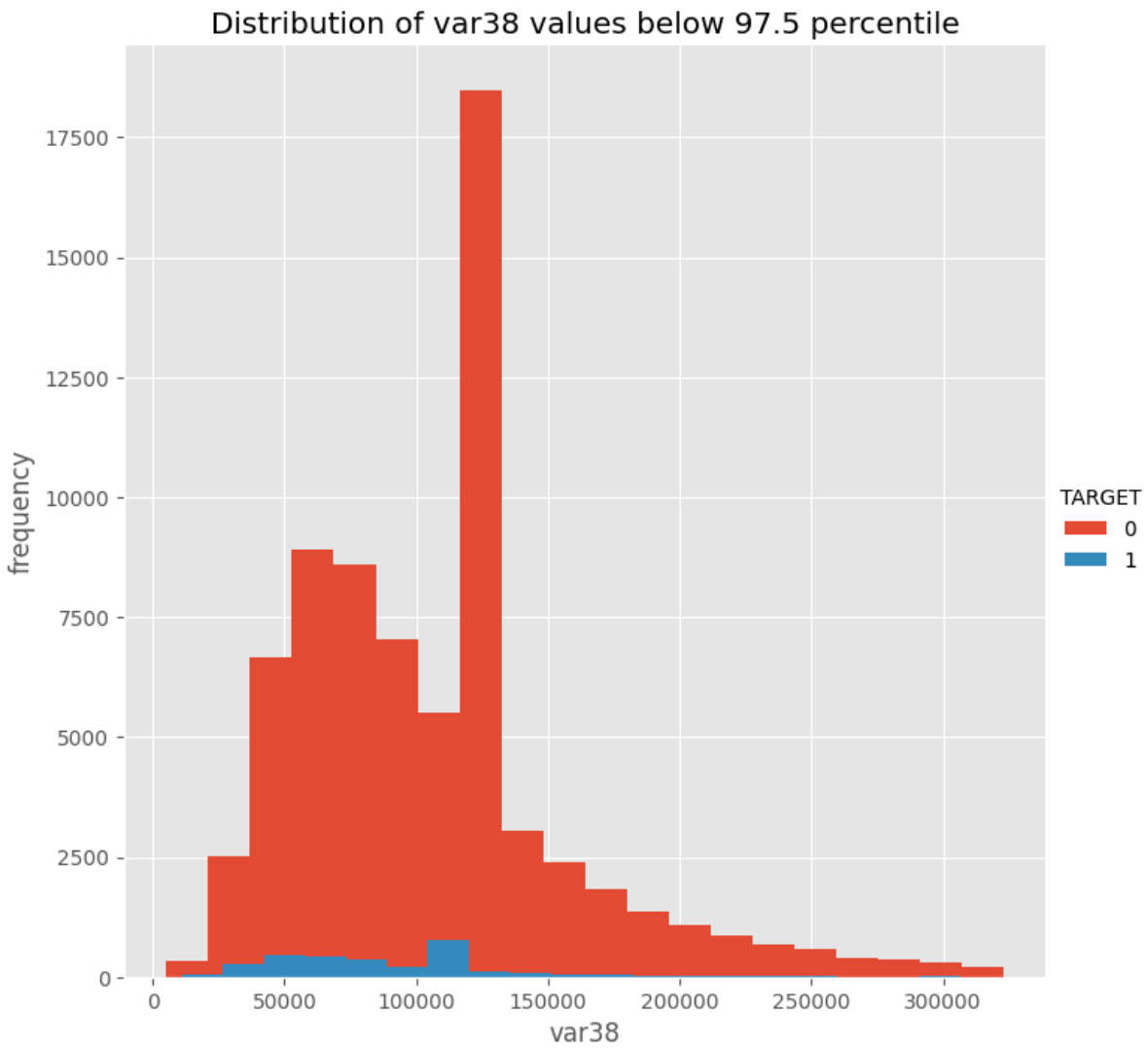
```
0.00 percentile : 5163
2.50 percentile : 32664
5.00 percentile : 39255
7.50 percentile : 44098
10.00 percentile : 48070
```

In [41]:
```
for i in np.arange(0.9,1.01,0.025):
    print('%.2f percentile : %i'%(i*100,np.quantile(df_train.var38.values,i)))
```

```
90.00 percentile : 182585
92.50 percentile : 206707
95.00 percentile : 242780
97.50 percentile : 323173
100.00 percentile : 22034738
```

Podemos ver que o mínimo e o máximo são muito diferentes em magnitude do restante da amostra.

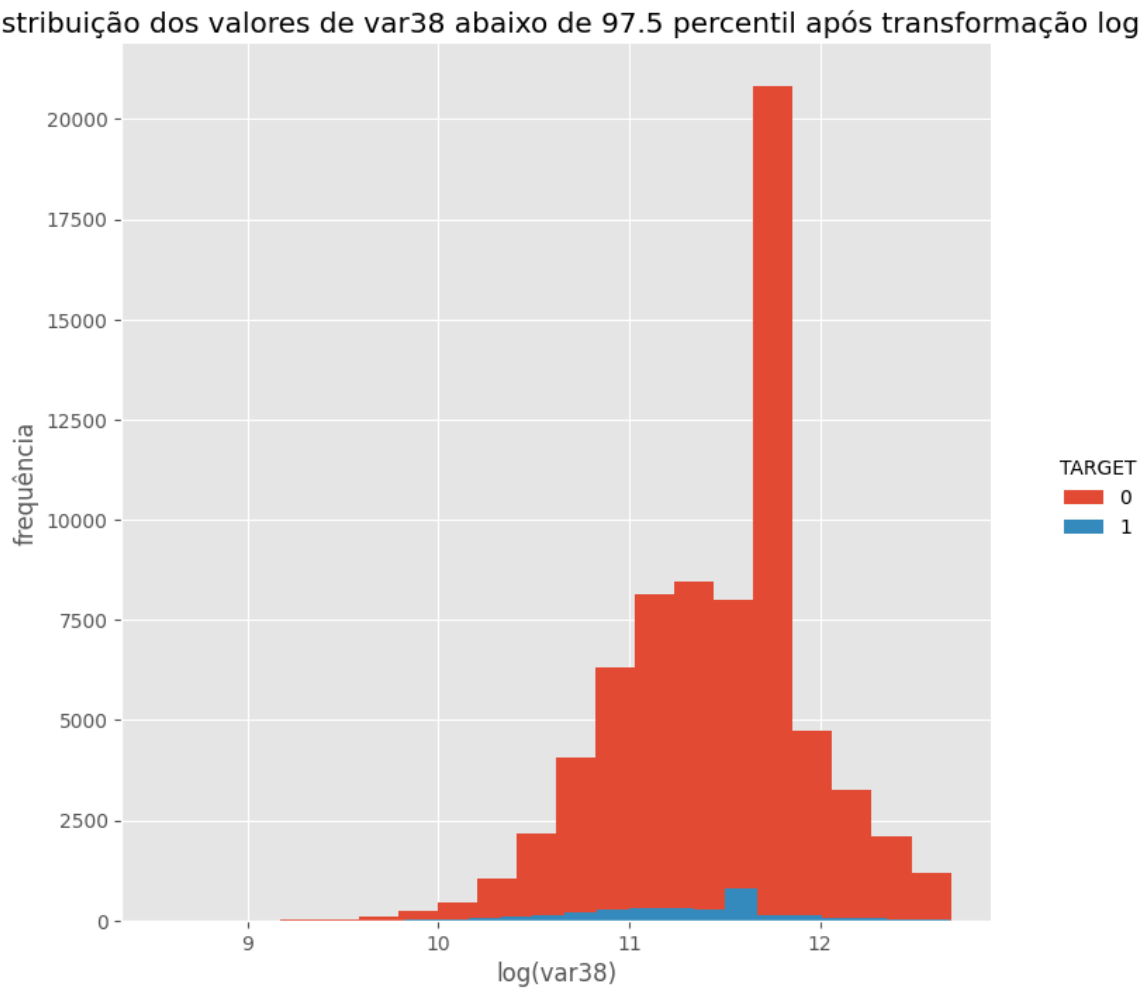
In [42]:
```
mask = df_train[df_train['var38']<=np.quantile(df_train.var38.values,0.975)]
sns.FacetGrid(data=mask,hue='TARGET',height=7).map(plt.hist,'var38',bins=20).add_le
plt.title('Distribution of var38 values below 97.5 percentile')
plt.ylabel("frequency")
plt.show()
```



Podemos ver que o gráfico acima está inclinado para a direita com um pico estranho entre 100.000 e 150.000. Podemos aplicar a transformação de log e verificar a distribuição resultante.

In [43]:
```
mask.loc[:,'var38'] = np.log(mask.var38).values
sns.FacetGrid(data=mask,hue='TARGET',height=7).map(plt.hist,'var38',bins=20).add_le
plt.title("Distribuição dos valores de var38 abaixo de 97.5 percentil após transfor
plt.xlabel('log(var38)')
plt.ylabel("frequência")
plt.show()
```

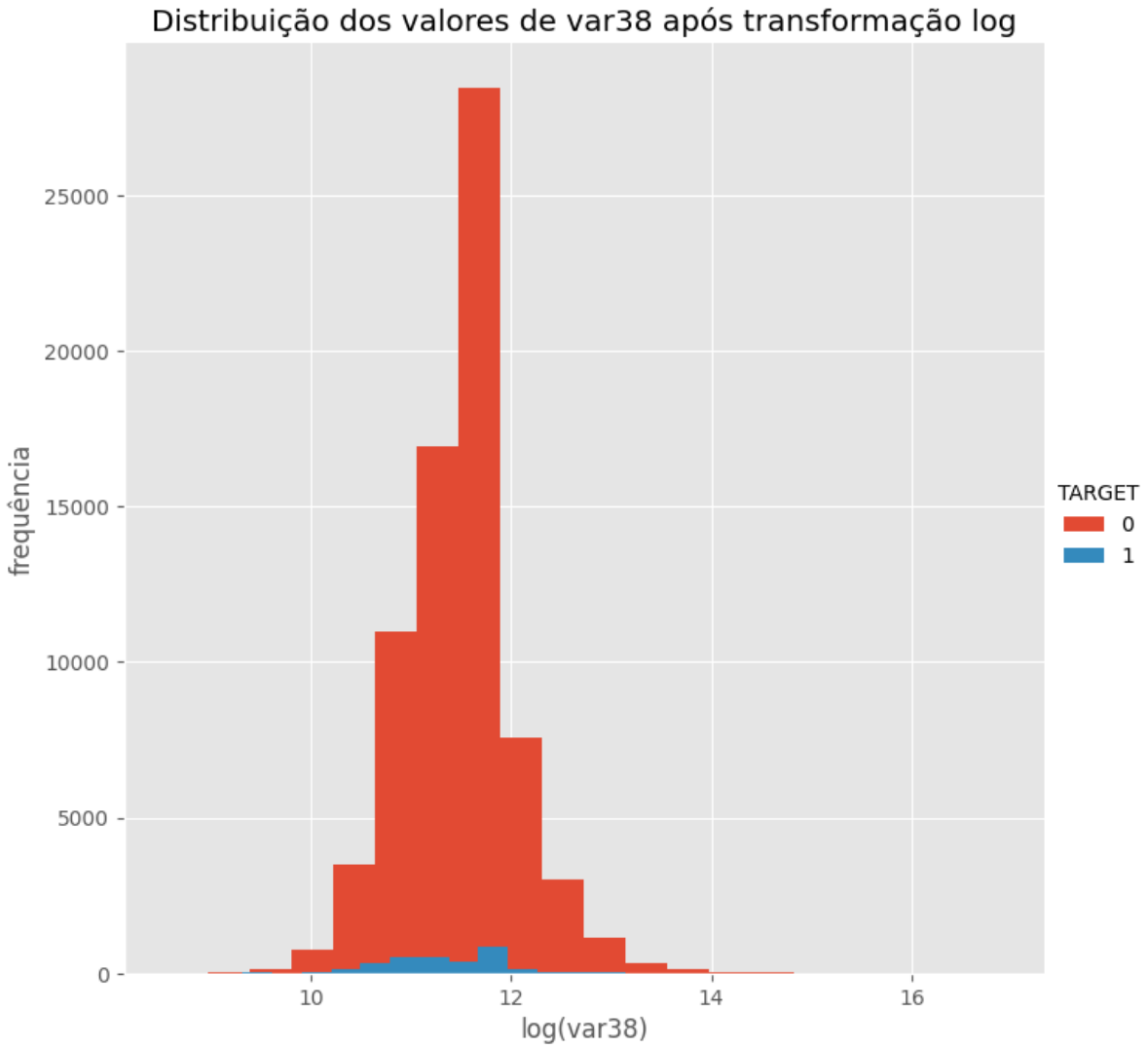
```
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\2568799573.py:1: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  mask.loc[:,'var38'] = np.log(mask.var38).values
```



Isso é muito melhor do que o anterior. Portanto, aplicamos a transformação de log ao recurso var38.

In [44]:
```
df_train.loc[:,'var38'] = np.log(df_train['var38']).values
plt.figure(figsize=(8,6))
sns.FacetGrid(data=df_train,hue='TARGET',height=7).map(plt.hist,'var38',bins=20).ad
plt.title("Distribuição dos valores de var38 após transformação log")
plt.xlabel('log(var38)')
plt.ylabel("frequência")
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



7. Analisando colunas com palavras-chave

In [45]:
```
df_train.columns
```

Out[45]:
```
Index(['ID', 'var3', 'var15', 'imp_ent_var16_ult1', 'imp_op_var39_comer_ult1',
       'imp_op_var39_comer_ult3', 'imp_op_var41_comer_ult1',
       'imp_op_var41_comer_ult3', 'imp_op_var41_efect_ult1',
       'imp_op_var41_efect_ult3',
       ...
       'saldo_medio_var12_hace3', 'saldo_medio_var12_ult1',
       'saldo_medio_var12_ult3', 'saldo_medio_var13_corto_hace2',
       'saldo_medio_var13_corto_hace3', 'saldo_medio_var13_corto_ult1',
       'saldo_medio_var13_corto_ult3', 'var38', 'TARGET', 'var15_below_23'],
      dtype='object', length=144)
```

In [46]:
```
f_keywords = {col.split('_')[0] for col in df_train.columns if (len(col.split('_'))
f_keywords
```

Out[46]:
```
{'imp', 'ind', 'num', 'saldo'}
```

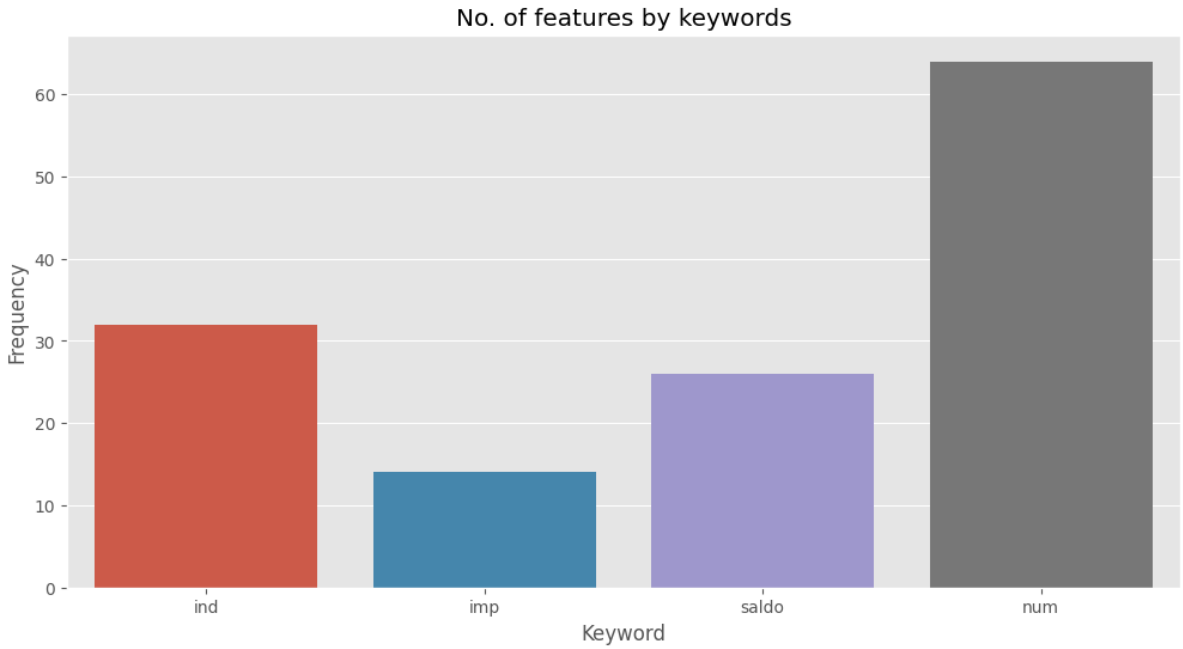
Temos 4 palavras-chaves principais. Vamos analisar alguns casos de cada uma.

In [47]:
```
f_keywords = dict(zip(f_keywords,np.zeros(len(f_keywords),dtype=int)))
for key in f_keywords.keys():
    for col in df_train.columns:
        if key in col:
            f_keywords[key]+=1
f_keywords
```

Out[47]:
```
{'ind': 32, 'imp': 14, 'saldo': 26, 'num': 64}
```

A palavra-chave com mais ocorrências é o 'num'.

In [48]:
```
k = pd.Series(f_keywords)
sns.barplot(x=k.index,y=k.values)
plt.title("No. of features by keywords")
plt.ylabel('Frequency')
plt.xlabel('Keyword')
plt.show()
```



imp features

In [50]:
```
imp = [col for col in df_train.columns if 'imp' in col]
print("O números de colunas com a palavra-chave 'imp': %i"%(len(imp)))
imp
```

```
O números de colunas com a palavra-chave 'imp': 14
```

Out[50]:
```
['imp_ent_var16_ult1',
 'imp_op_var39_comer_ult1',
 'imp_op_var39_comer_ult3',
 'imp_op_var41_comer_ult1',
 'imp_op_var41_comer_ult3',
 'imp_op_var41_efect_ult1',
 'imp_op_var41_efect_ult3',
 'imp_op_var41_ult1',
 'imp_op_var39_efect_ult1',
 'imp_op_var39_efect_ult3',
 'imp_op_var39_ult1',
 'imp_aport_var13_hace3',
 'imp_var43_emit_ult1',
 'imp_trans_var37_ult1']
```

Vamos escolher duas variáveis aleatórias para entender seus registros.

In [52]:
```
import random
random.seed()
imp_feat = random.sample(imp,2)
print("As variáveis escolhidas aleatoriamente são '%s' e '%s'"%(imp_feat[0],imp_fea
```

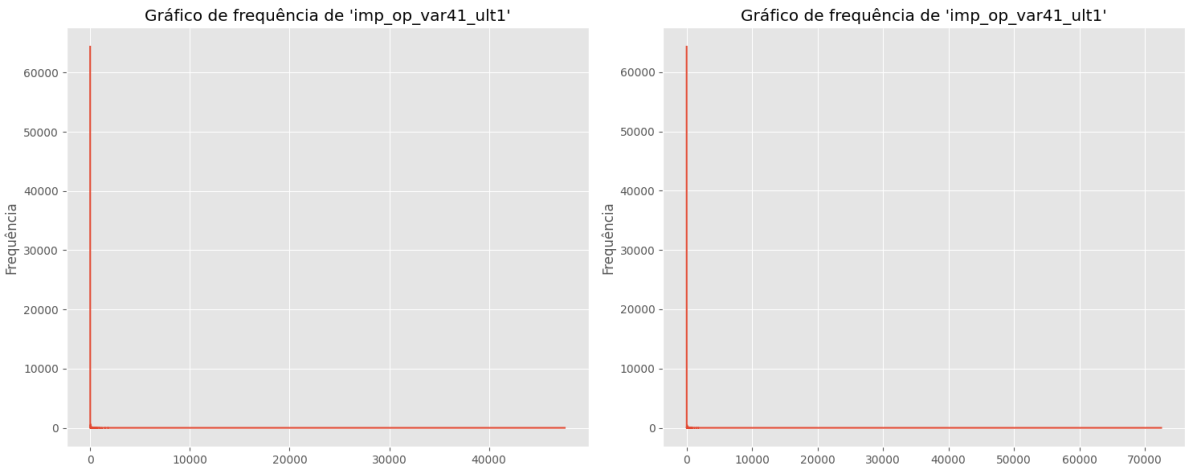
```
As variáveis escolhidas aleatoriamente são 'imp_op_var41_ult1' e 'imp_trans_var37_
ult1'
```

'imp_op_var41_ult1'

In [54]:
```
col = imp_feat[0]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'imp_op_var41_ult1' é 0.0 e 47598.1
O valor mínimo e máximo da base de teste para 'imp_op_var41_ult1' é 0.0 e 72511.8
```

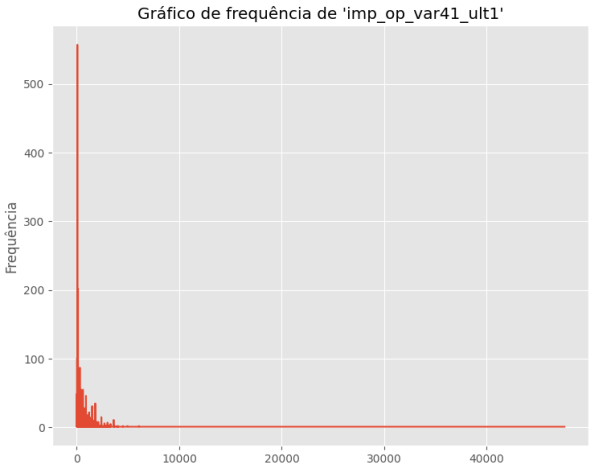
In [55]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```

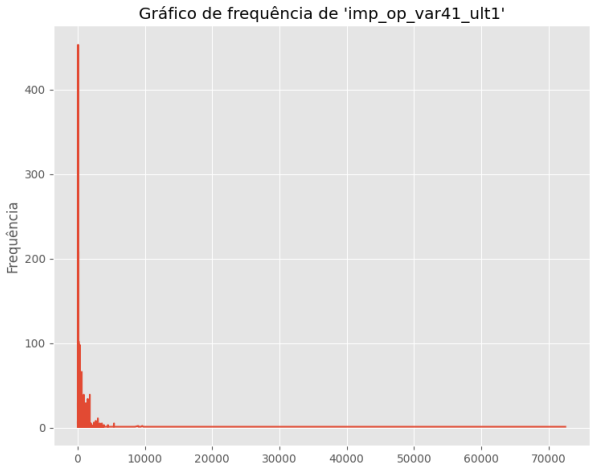


```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'imp_op_var41_ult1':
Valor    Perc%
0.0      84.698763
60.0      0.732702
120.0     0.265720
30.0      0.132860
300.0     0.114444
Name: imp_op_var41_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'imp_op_var41_ult1':
Valor    Perc%
185.55     0.001315
394.89     0.001315
1101.39    0.001315
173.85     0.001315
1575.99    0.001315
Name: imp_op_var41_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'imp_op_var41_ult1':
Valor    Perc%
0.0      84.813633
60.0      0.597483
120.0     0.274341
30.0      0.158274
150.0     0.134533
Name: imp_op_var41_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'imp_op_var41_ult1':
Valor    Perc%
803.94     0.001319
666.09     0.001319
381.12     0.001319
318.18     0.001319
1586.94    0.001319
Name: imp_op_var41_ult1, dtype: float64
```

Vamos tirar as ocorrências de 0 e ver como se comporta o gráfico.

In [56]:
```
col = imp_feat[0]
#plotting frequency plots with no zero (most occuring) values
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```



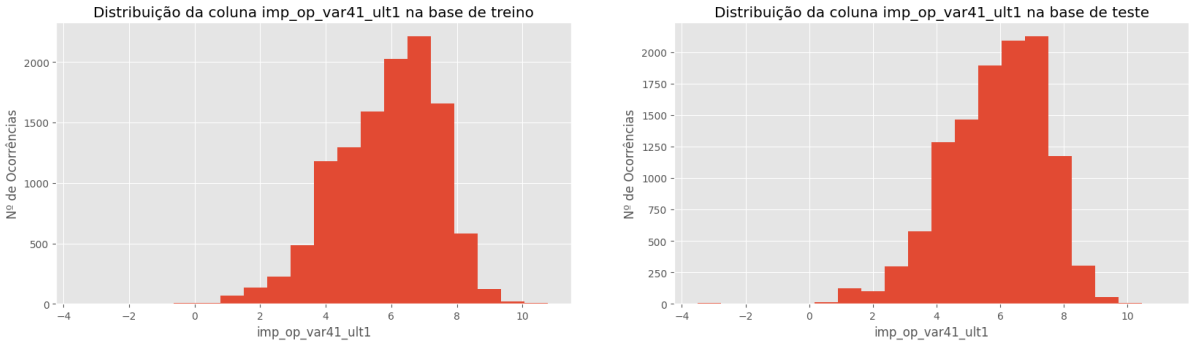



```
**************************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'imp_op_var41_ult1':
Valor    Perc%
60.0     4.788514
120.0    1.736589
30.0     0.868294
300.0    0.747937
180.0    0.696355
Name: imp_op_var41_ult1, dtype: float64
**************************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'imp_op_var41_ult1':
Valor    Perc%
185.55     0.008597
394.89     0.008597
1101.39    0.008597
173.85     0.008597
1575.99    0.008597
Name: imp_op_var41_ult1, dtype: float64
**************************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'imp_op_var41_ult1':
Valor    Perc%
60.0     3.934341
120.0    1.806496
30.0     1.042209
150.0    0.885878
300.0    0.851138
Name: imp_op_var41_ult1, dtype: float64
**************************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'imp_op_var41_ult1':
Valor    Perc%
535.26     0.008685
803.94     0.008685
666.09     0.008685
381.12     0.008685
1586.94    0.008685
Name: imp_op_var41_ult1, dtype: float64
```

Se considerarmos esses registros com exceção do 0 e aplicarmos um log, o resultado fica

assim:

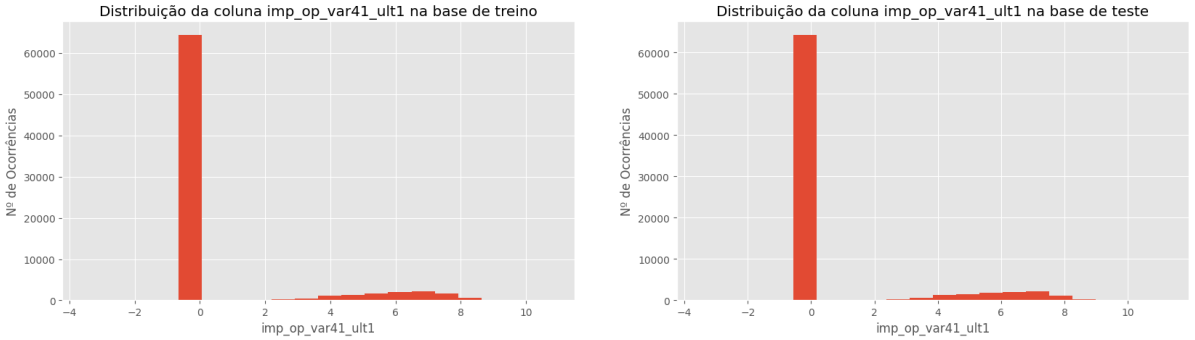
In [57]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Um distribuição mais para a direita.

Se incluirmos o zero:

In [58]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

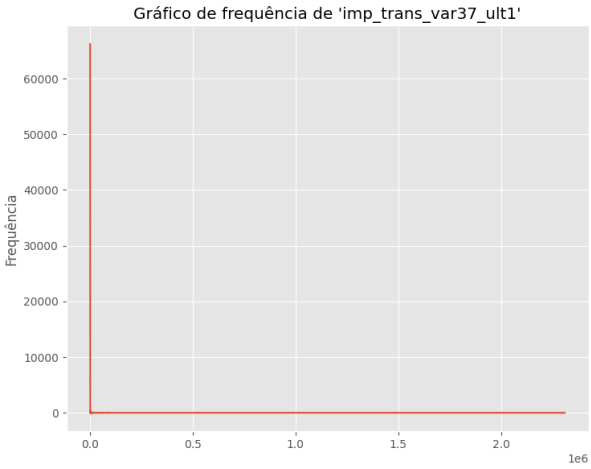


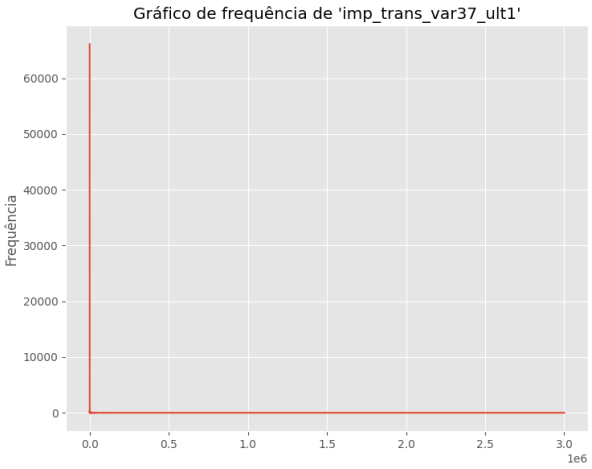
'imp_trans_var37_ult1'

In [59]:
```
col = imp_feat[1]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'imp_trans_var37_ult1' é 0.0 e 2310
003.0
O valor mínimo e máximo da base de teste para 'imp_trans_var37_ult1' é 0.0 e 30000
00.0
```

In [60]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```

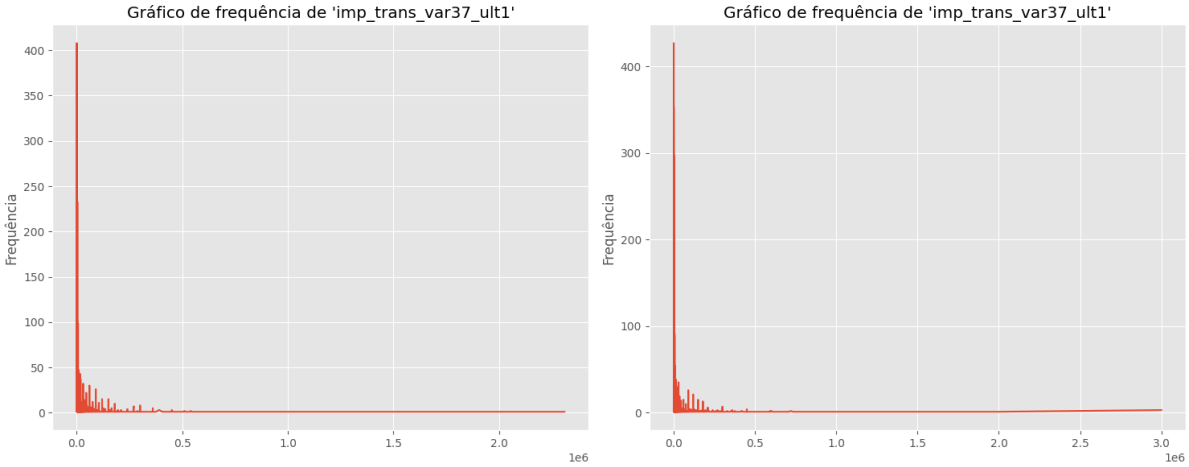





```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'imp_trans_var37_ult1':
Valor    Perc%
0.0       87.069192
300.0      0.536701
600.0      0.420942
1500.0     0.361747
150.0      0.344646
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'imp_trans_var37_ult1':
Valor    Perc%
2.37       0.001315
4171.68    0.001315
7881.96    0.001315
4740.00    0.001315
792.81     0.001315
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'imp_trans_var37_ult1':
Valor    Perc%
0.0       87.235221
300.0      0.563191
600.0      0.465589
1500.0     0.391728
3000.0     0.319185
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'imp_trans_var37_ult1':
Valor    Perc%
3309.84     0.001319
662.88      0.001319
2206.53     0.001319
5576.19     0.001319
10938.60    0.001319
Name: imp_trans_var37_ult1, dtype: float64
```

In [61]:
```
col = imp_feat[1]
```

```
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```

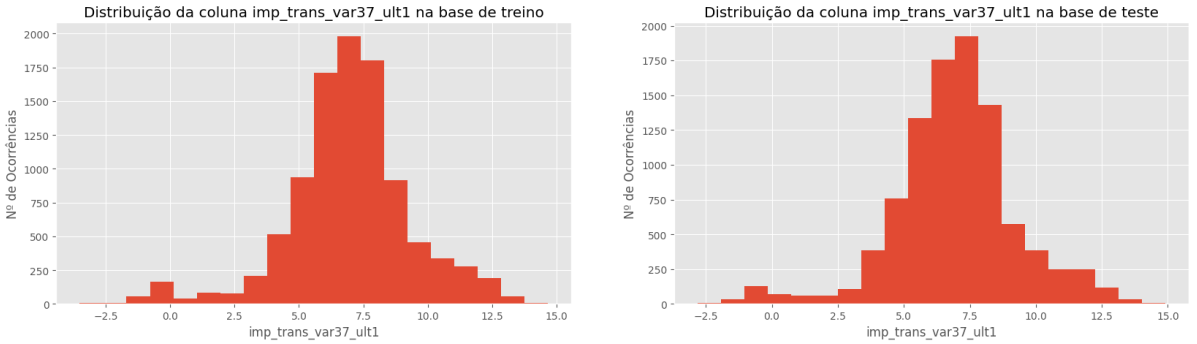



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'imp_trans_var37_ult1':
Valor    Perc%
300.0     4.150560
600.0     3.255341
1500.0    2.797558
150.0     2.665310
900.0     2.492370
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'imp_trans_var37_ult1':
Valor    Perc%
4171.68    0.010173
7881.96    0.010173
4740.00    0.010173
5850.00    0.010173
792.81     0.010173
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'imp_trans_var37_ult1':
Valor    Perc%
300.0     4.412069
600.0     3.647448
1500.0    3.068816
3000.0    2.500517
900.0     2.448853
Name: imp_trans_var37_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'imp_trans_var37_ult1':
Valor    Perc%
3309.84     0.010333
662.88      0.010333
2206.53     0.010333
5576.19     0.010333
10938.60    0.010333
Name: imp_trans_var37_ult1, dtype: float64
```

Tirando os 0's, o 300,0 é o que mais aparece.

Vamos tirar as ocorrências de 0 e ver como se comporta o gráfico.

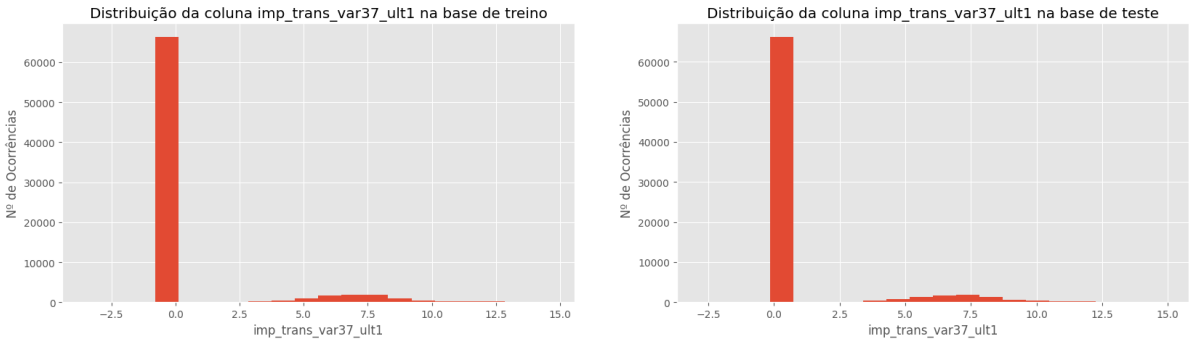
In [62]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Temos uma distribuição mais próxima da normal.

Se incluirmos o 0:

In [63]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Palavra-chave saldo

In [64]:
```
imp = [col for col in df_train.columns if 'saldo' in col]
print("O número de colunas com a palavra-chave saldo é: %i"%(len(imp)))
imp
```

```
O número de colunas com a palavra-chave saldo é: 26
```

Out[64]:
```
['saldo_var5',
 'saldo_var8',
 'saldo_var12',
 'saldo_var13_corto',
 'saldo_var13',
 'saldo_var24',
 'saldo_var26',
 'saldo_var25',
 'saldo_var30',
 'saldo_var37',
 'saldo_var42',
 'saldo_medio_var5_hace2',
 'saldo_medio_var5_hace3',
 'saldo_medio_var5_ult1',
 'saldo_medio_var5_ult3',
 'saldo_medio_var8_hace2',
 'saldo_medio_var8_ult1',
 'saldo_medio_var8_ult3',
 'saldo_medio_var12_hace2',
 'saldo_medio_var12_hace3',
 'saldo_medio_var12_ult1',
 'saldo_medio_var12_ult3',
 'saldo_medio_var13_corto_hace2',
 'saldo_medio_var13_corto_hace3',
 'saldo_medio_var13_corto_ult1',
 'saldo_medio_var13_corto_ult3']
```

In [65]:
```
random.seed()
imp_feat = random.sample(imp,2)
print("As variáveis escolhidas aleatoriamente são '%s' é '%s'"%(imp_feat[0],imp_fea
```

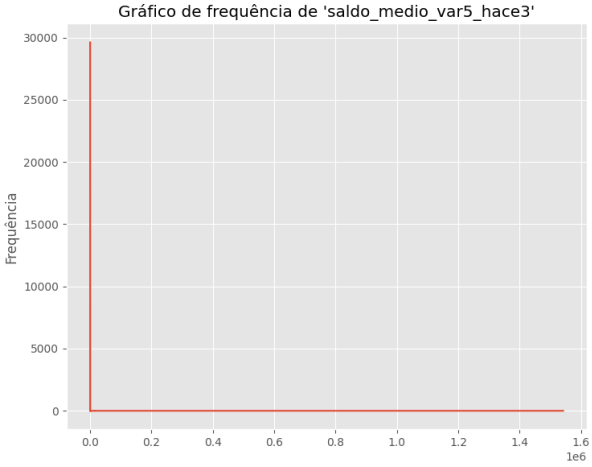
```
As variáveis escolhidas aleatoriamente são 'saldo_medio_var5_hace3' é 'saldo_var2
4'
```

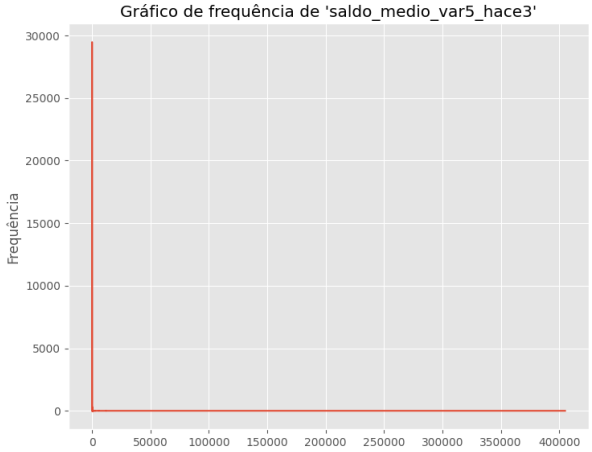
'saldo_medio_var5_hace3'

In [66]:
```
col = imp_feat[0]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'saldo_medio_var5_hace3' é -8.0 e 1
542339.4
O valor mínimo e máximo da base de teste para 'saldo_medio_var5_hace3' é -32.9 e 4
05001.5
```

In [67]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```

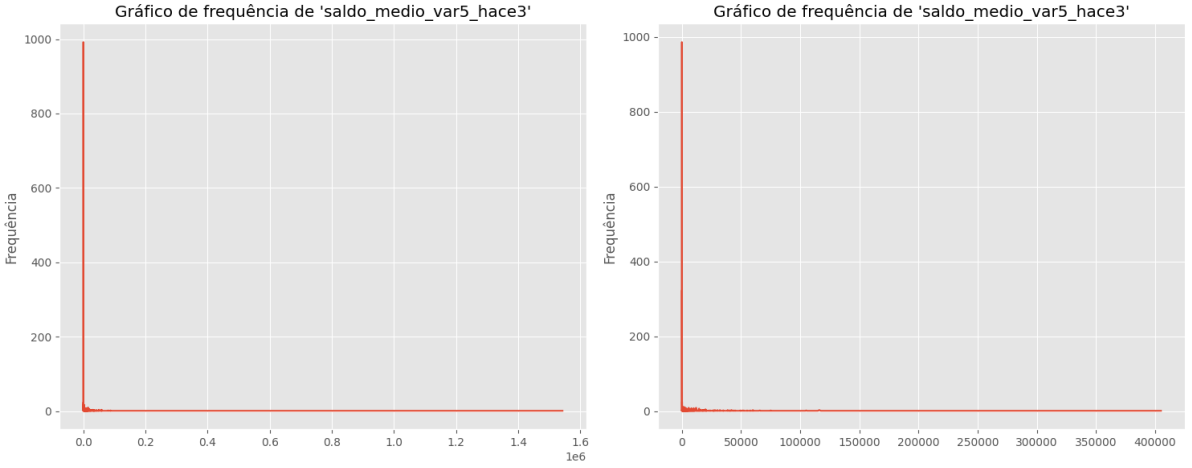





```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'saldo_medio_var5_hace3':
Valor    Perc%
0.00    38.985793
0.09     1.304920
0.18     1.151013
0.99     0.974743
2.61     0.924757
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'saldo_medio_var5_hace3':
Valor    Perc%
28955.13    0.001315
25161.27    0.001315
270.99      0.001315
8001.39     0.001315
885.84      0.001315
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'saldo_medio_var5_hace3':
Valor    Perc%
0.00    38.872036
0.09     1.300483
0.18     1.197605
0.99     0.923264
2.61     0.916669
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'saldo_medio_var5_hace3':
Valor    Perc%
71.01       0.001319
1723.02     0.001319
959.70      0.001319
20001.90    0.001319
3405.15     0.001319
Name: saldo_medio_var5_hace3, dtype: float64
```

Vamos tirar as ocorrências de 0 e ver como se comporta o gráfico.

In [68]:
```
col = imp_feat[0]
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```

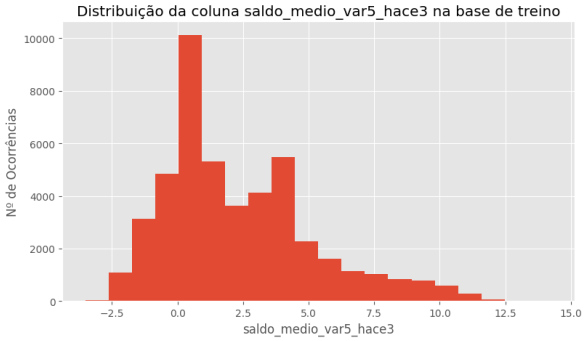


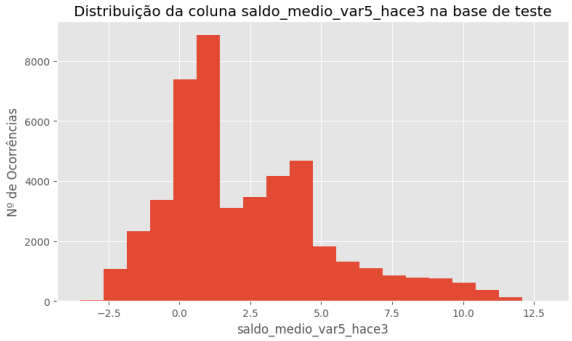
```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'saldo_medio_var5_hace3':
Valor    Perc%
0.09    2.138715
0.18    1.886467
0.99    1.597568
2.61    1.515642
0.27    1.356100
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'saldo_medio_var5_hace3':
Valor    Perc%
28955.13    0.002156
25161.27    0.002156
270.99      0.002156
8001.39     0.002156
885.84      0.002156
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'saldo_medio_var5_hace3':
Valor    Perc%
0.09    2.127476
0.18    1.959177
0.99    1.510378
2.61    1.499590
0.27    1.387390
Name: saldo_medio_var5_hace3, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'saldo_medio_var5_hace3':
Valor    Perc%
71.01       0.002158
1723.02     0.002158
959.70      0.002158
20001.90    0.002158
3405.15     0.002158
Name: saldo_medio_var5_hace3, dtype: float64
```

Se considerarmos esses registros com exceção do 0 e aplicarmos um log, o resultado fica assim:

In [69]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\pandas\cor
e\arraylike.py:402: RuntimeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

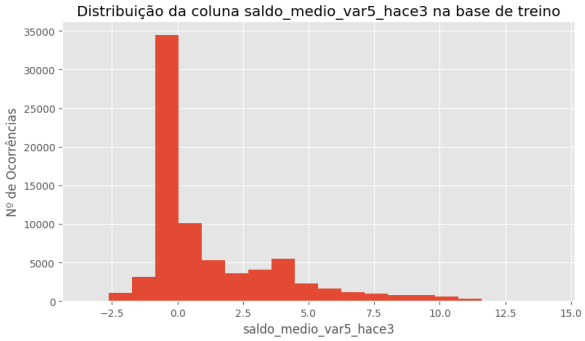


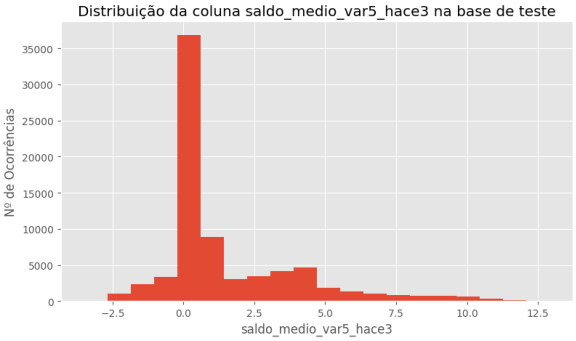


Uma distribuição mais para a direita.

Se incluirmos o zero:

In [70]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



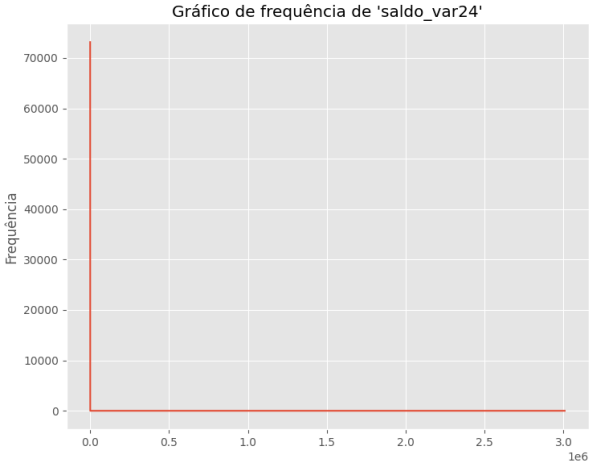


'saldo_var24'

In [71]:
```
col = imp_feat[1]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'saldo_var24' é 0.0 e 3008077.3
O valor mínimo e máximo da base de teste para 'saldo_var24' é 0.0 e 4202599.2
```

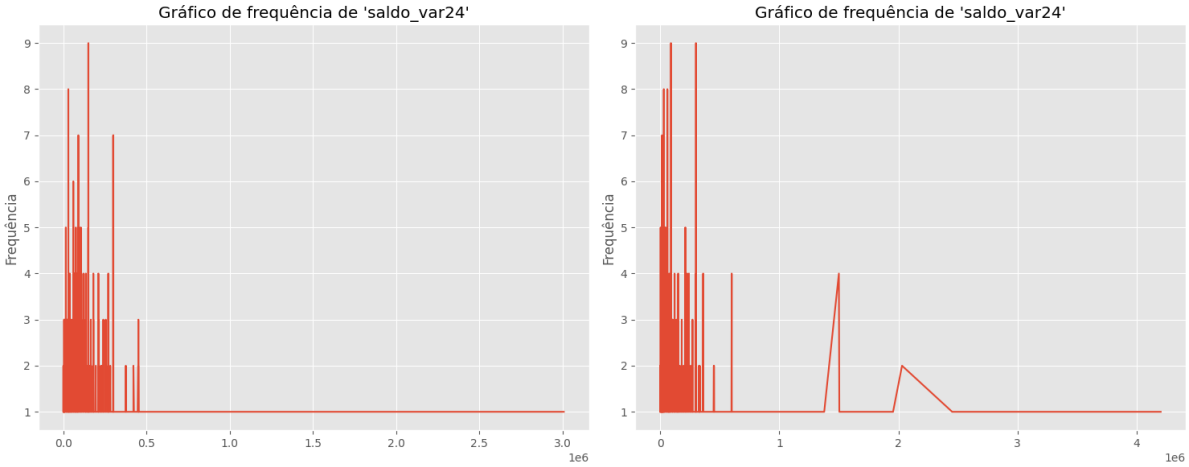
In [72]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```




```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'saldo_var24':
Valor    Perc%
0.00         96.211523
150392.37     0.011839
30000.00      0.010524
300000.00     0.009208
300385.68     0.009208
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'saldo_var24':
Valor    Perc%
180242.31    0.001315
309000.00    0.001315
260143.71    0.001315
120545.01    0.001315
48191.22     0.001315
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'saldo_var24':
Valor    Perc%
0.0          96.194835
90000.0       0.011871
300000.0      0.011871
60000.0       0.010552
30000.0       0.010552
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'saldo_var24':
Valor    Perc%
356699.82    0.001319
208211.82    0.001319
15059.25     0.001319
4085.37      0.001319
24702.63     0.001319
Name: saldo_var24, dtype: float64
```

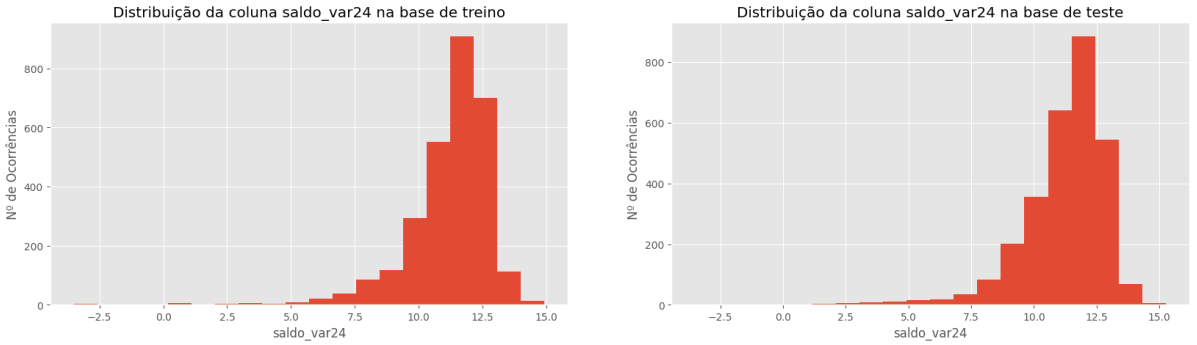
In [73]:
```
col = imp_feat[1]
```

```
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```




```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'saldo_var24':
Valor    Perc%
150392.37    0.312500
30000.00     0.277778
90000.00     0.243056
300000.00    0.243056
300385.68    0.243056
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'saldo_var24':
Valor    Perc%
180242.31    0.034722
309000.00    0.034722
260143.71    0.034722
120545.01    0.034722
48191.22     0.034722
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'saldo_var24':
Valor    Perc%
300000.0    0.311958
90000.0     0.311958
30000.0     0.277296
60000.0     0.277296
15000.0     0.242634
Name: saldo_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'saldo_var24':
Valor    Perc%
208211.82    0.034662
15059.25     0.034662
4085.37      0.034662
240015.00    0.034662
24702.63     0.034662
Name: saldo_var24, dtype: float64
```

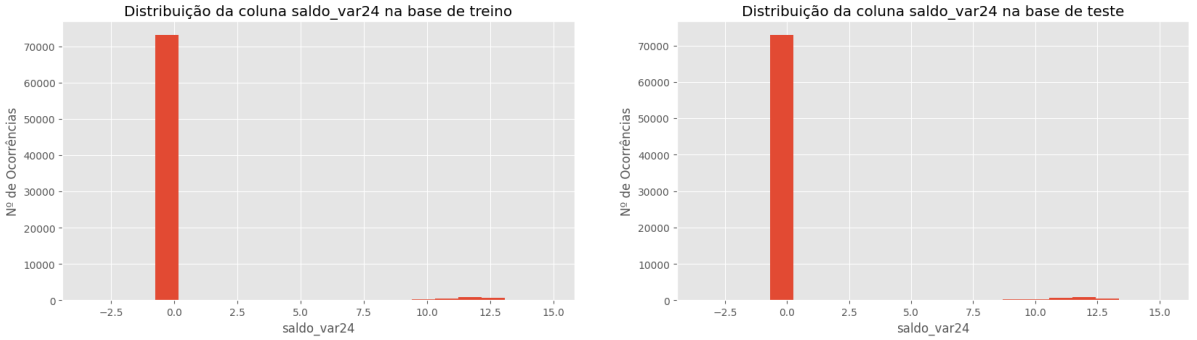
In [74]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Temos uma distribuição mais para a direita.

Se incluirmos o 0:

In [76]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Palavra-chave num

In [77]:
```
imp = [col for col in df_train.columns if 'num' in col]
print("O números de colunas com a palavra-chave 'num': %i"%(len(imp)))
imp
```

```
O números de colunas com a palavra-chave 'num': 64
```

Out[77]:
```
['num_var1_0',
 'num_var4',
 'num_var5_0',
 'num_var5',
 'num_var8_0',
 'num_var8',
 'num_var12_0',
 'num_var12',
 'num_var13_0',
 'num_var13_corto_0',
 'num_var13_corto',
 'num_var13_largo_0',
 'num_var13',
 'num_var14_0',
 'num_var24_0',
 'num_var24',
 'num_var26_0',
 'num_var25_0',
 'num_op_var41_hace2',
 'num_op_var41_hace3',
 'num_op_var41_ult1',
 'num_op_var41_ult3',
 'num_op_var39_hace2',
 'num_op_var39_hace3',
 'num_op_var39_ult1',
 'num_op_var39_ult3',
 'num_var30_0',
 'num_var30',
 'num_var35',
 'num_var37_med_ult2',
 'num_var37_0',
 'num_var39_0',
 'num_var40_0',
 'num_var41_0',
 'num_var42_0',
 'num_var42',
 'num_aport_var13_hace3',
 'num_ent_var16_ult1',
 'num_var22_hace2',
 'num_var22_hace3',
 'num_var22_ult1',
 'num_var22_ult3',
 'num_med_var22_ult3',
 'num_med_var45_ult3',
 'num_meses_var5_ult3',
 'num_meses_var8_ult3',
 'num_meses_var12_ult3',
 'num_meses_var13_corto_ult3',
 'num_meses_var39_vig_ult3',
 'num_op_var39_comer_ult1',
 'num_op_var39_comer_ult3',
 'num_op_var41_comer_ult1',
 'num_op_var41_comer_ult3',
 'num_op_var41_efect_ult1',
 'num_op_var41_efect_ult3',
 'num_op_var39_efect_ult1',
```

```
 'num_op_var39_efect_ult3',
 'num_var43_emit_ult1',
 'num_var43_recib_ult1',
 'num_trasp_var11_ult1',
 'num_var45_hace2',
 'num_var45_hace3',
 'num_var45_ult1',
 'num_var45_ult3']
```

Vamos escolher duas variáveis aleatórias para entender seus registros.

In [79]:
```
import random
random.seed()
imp_feat = random.sample(imp,2)
print("As variáveis escolhidas aleatoriamente são '%s' e '%s'"%(imp_feat[0],imp_fea
```

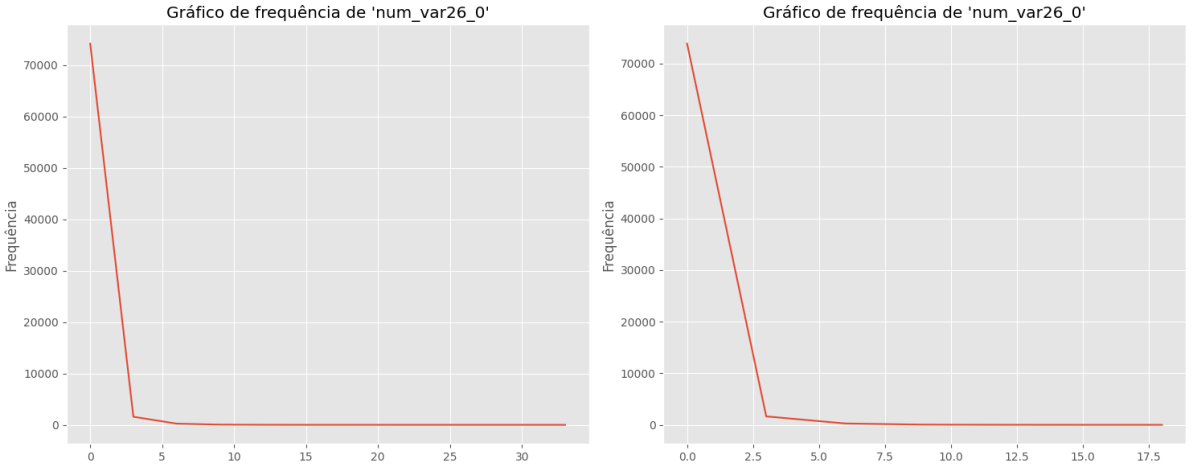
```
As variáveis escolhidas aleatoriamente são 'num_var26_0' e 'num_var24'
```

'num_var26_0'

In [80]:
```
col = imp_feat[0]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'num_var26_0' é 0.0 e 33.0
O valor mínimo e máximo da base de teste para 'num_var26_0' é 0.0 e 18.0
```

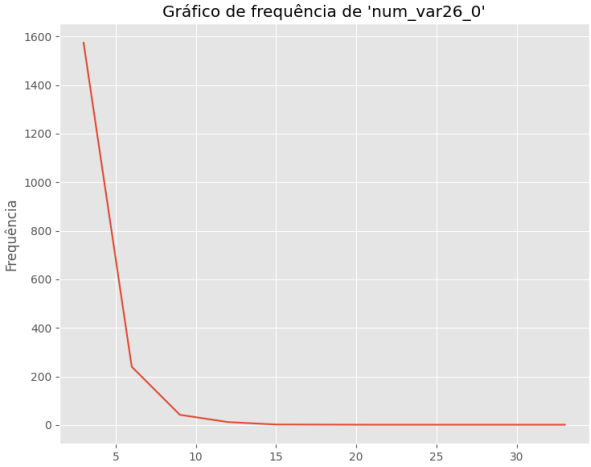
In [81]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```

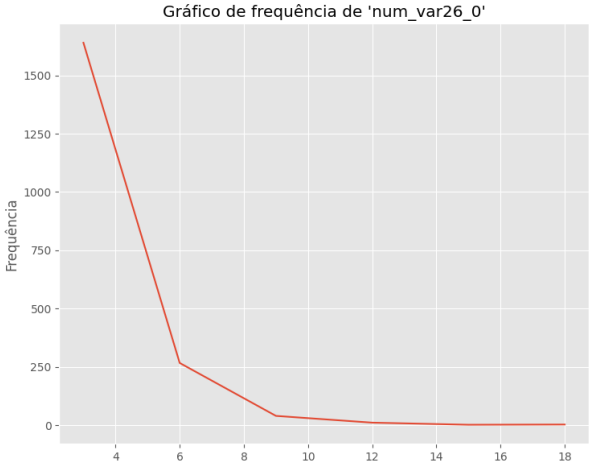


```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'num_var26_0':
Valor    Perc%
0     97.536175
3      2.070508
6      0.315706
9      0.055249
12     0.015785
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'num_var26_0':
Valor    Perc%
12    0.015785
15    0.002631
33    0.001315
21    0.001315
27    0.001315
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'num_var26_0':
Valor    Perc%
0     97.410905
3      2.163075
6      0.352159
9      0.052758
12     0.014508
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'num_var26_0':
Valor    Perc%
6     0.352159
9     0.052758
12    0.014508
18    0.003957
15    0.002638
Name: num_var26_0, dtype: float64
```

Vamos tirar as ocorrências de 0 e ver como se comporta o gráfico.

In [83]:
```
col = imp_feat[0]
#plotting frequency plots with no zero (most occuring) values
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```



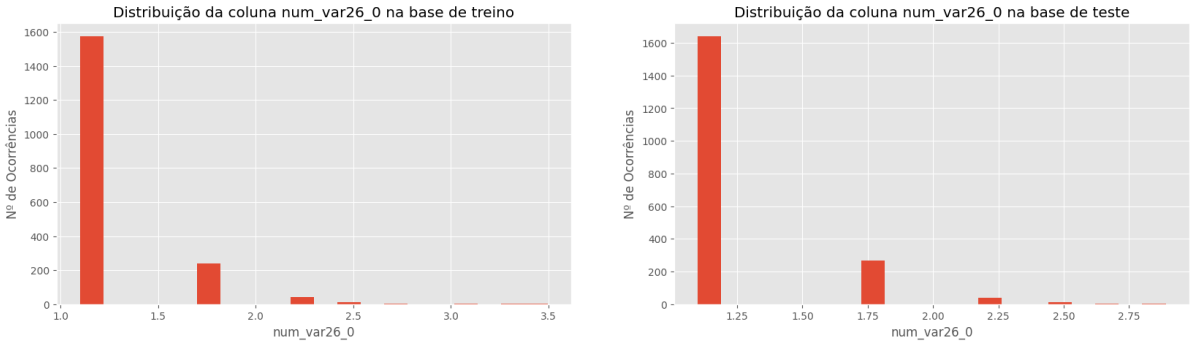



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'num_var26_0':
Valor    Perc%
3     84.036305
6     12.813668
9      2.242392
12     0.640683
15     0.106781
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'num_var26_0':
Valor    Perc%
12    0.640683
15    0.106781
33    0.053390
21    0.053390
27    0.053390
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'num_var26_0':
Valor    Perc%
3     83.545593
6     13.601630
9      2.037697
12     0.560367
18     0.152827
Name: num_var26_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'num_var26_0':
Valor    Perc%
6     13.601630
9      2.037697
12     0.560367
18     0.152827
15     0.101885
Name: num_var26_0, dtype: float64
```

Se considerarmos esses registros com exceção do 0 e aplicarmos um log, o resultado fica

assim:

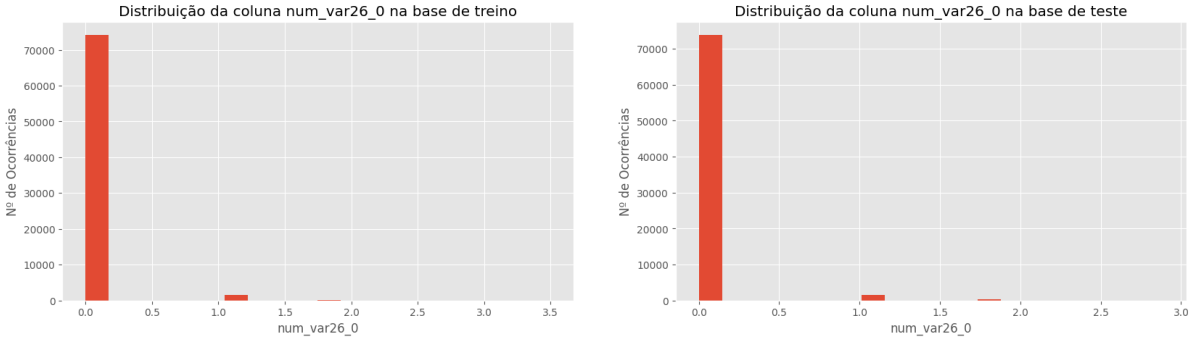
In [84]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



Temos concentrações em valores mais específicos.

Se incluirmos o zero:

In [86]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

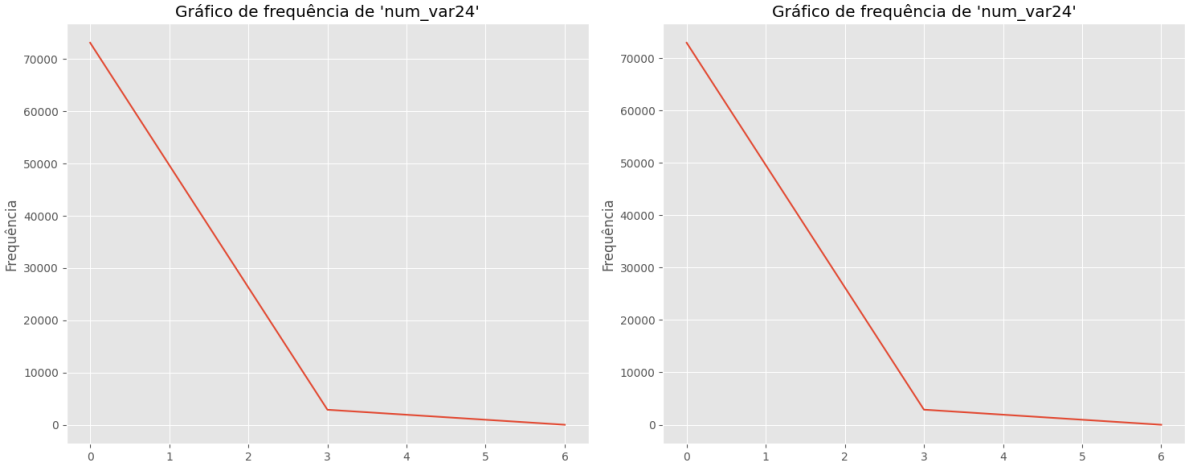


'num_var24'

In [87]:
```
col = imp_feat[1]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

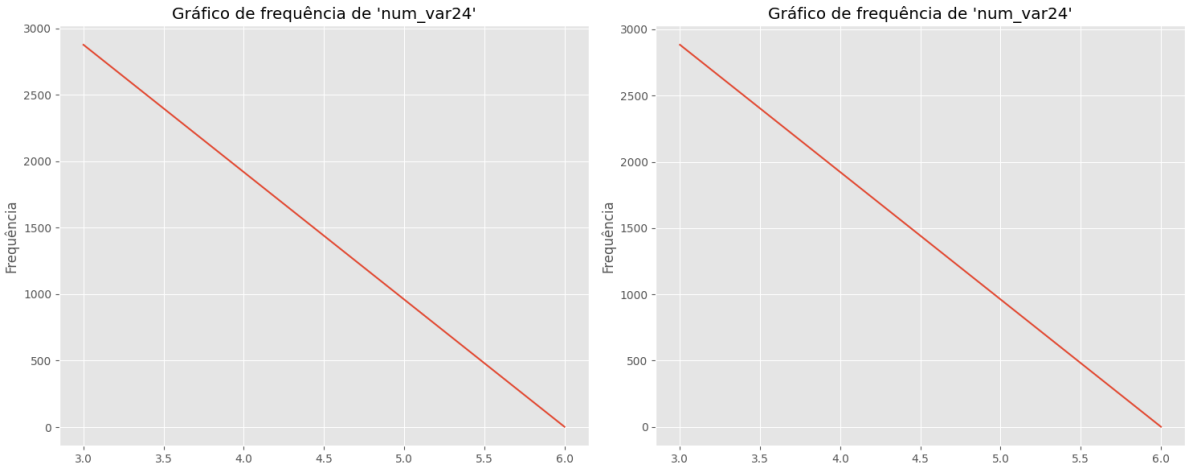
```
O valor mínimo e máximo da base de treino para 'num_var24' é 0.0 e 6.0
O valor mínimo e máximo da base de teste para 'num_var24' é 0.0 e 6.0
```

In [88]:
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```




```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'num_var24':
Valor    Perc%
0    96.211523
3     3.784530
6     0.003946
Name: num_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'num_var24':
Valor    Perc%
0    96.211523
3     3.784530
6     0.003946
Name: num_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'num_var24':
Valor    Perc%
0    96.194835
3     3.803846
6     0.001319
Name: num_var24, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'num_var24':
Valor    Perc%
0    96.194835
3     3.803846
6     0.001319
Name: num_var24, dtype: float64
```

In [89]:
```
col = imp_feat[1]
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```

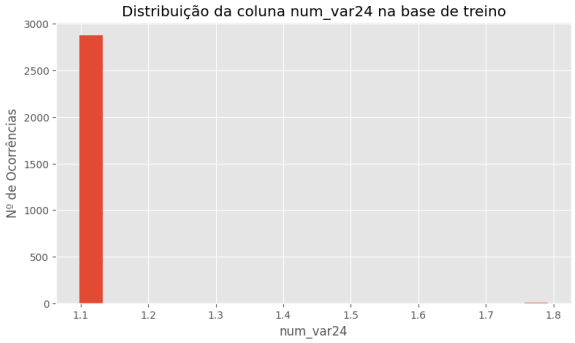


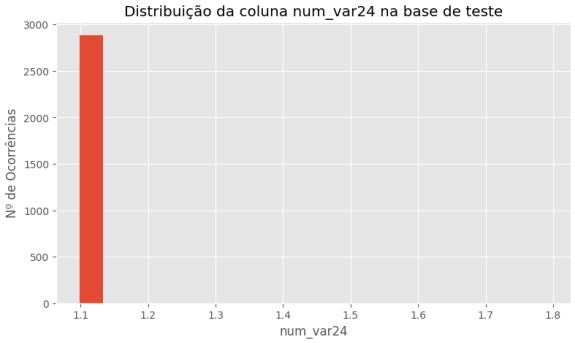

```
**************************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'num_var24':
Valor    Perc%
3    99.895833
6     0.104167
Name: num_var24, dtype: float64
**************************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'num_var24':
Valor    Perc%
3    99.895833
6     0.104167
Name: num_var24, dtype: float64
**************************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'num_var24':
Valor    Perc%
3    99.965338
6     0.034662
Name: num_var24, dtype: float64
**************************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'num_var24':
Valor    Perc%
3    99.965338
6     0.034662
Name: num_var24, dtype: float64
```

Tirando os 0's, temos a distribuição entre o valor 3 e 6.

Vamos tirar as ocorrências de 0 e ver como se comporta o gráfico.

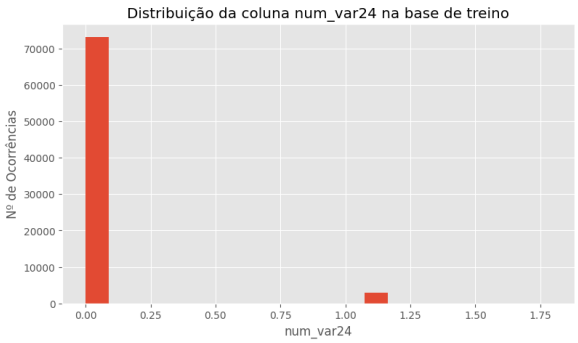
In [90]:
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

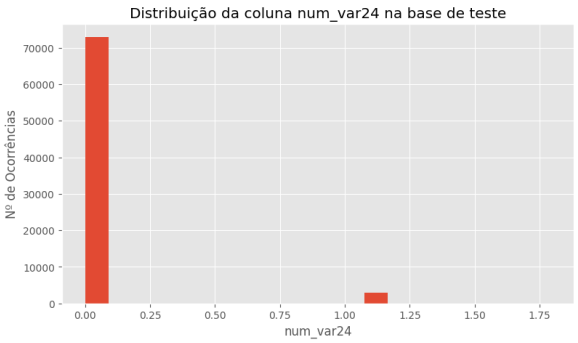




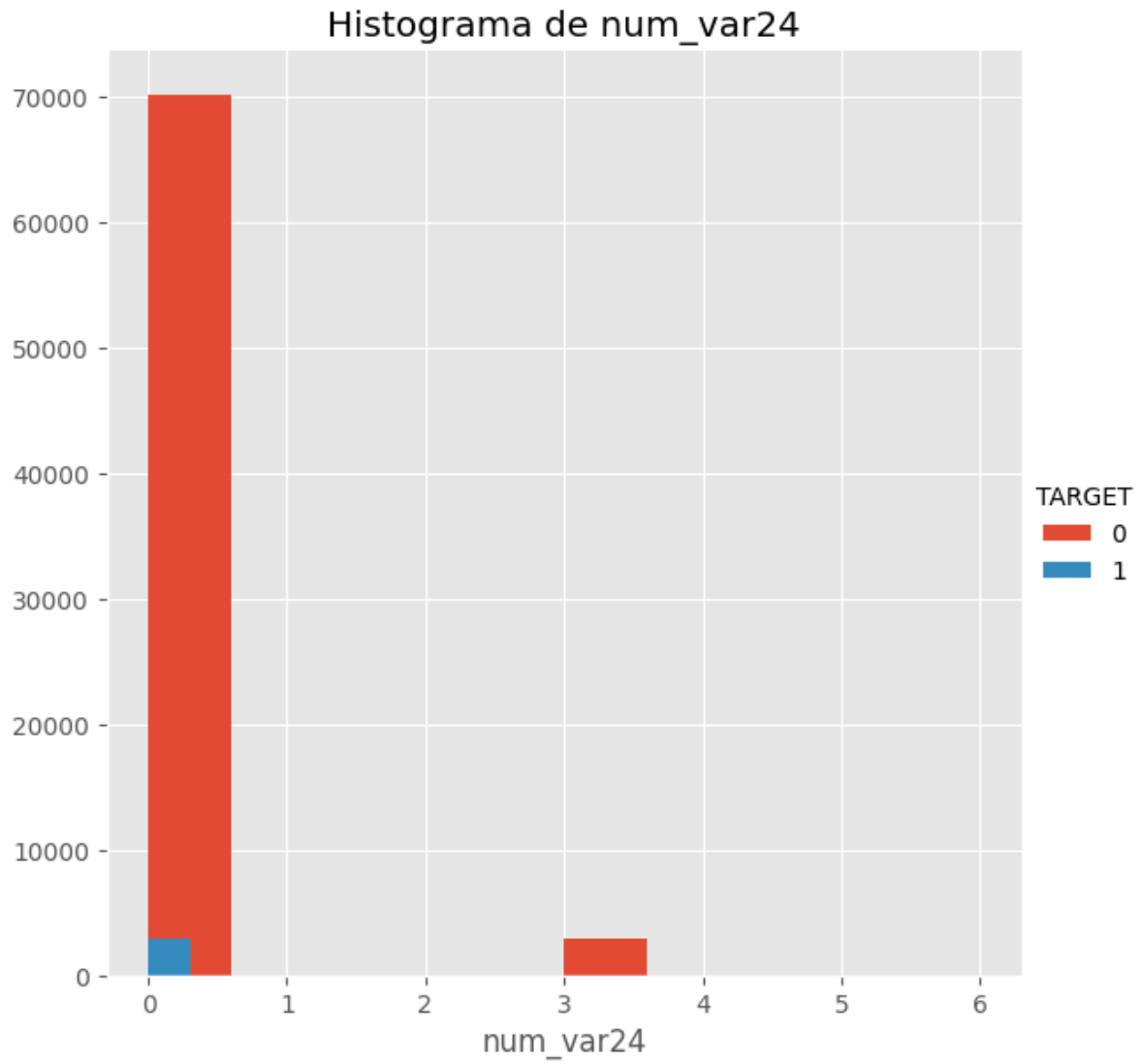
Se incluirmos o 0:

In [91]:
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```





In [95]:
```
y_col='TARGET'
sns.FacetGrid(data=df_train,hue='TARGET',height=6).map(plt.hist,col).add_legend()
plt.title('Histograma de %s'%(col))
plt.show()
for i in df_train[col].unique():
    print("O percentual de clientes insatisfeitos quando '%s'==%i é %.3f%%"%\
      (col,i,df_train[(df_train[y_col]==1)&(df_train[col]==i)][col].shape[0]*100/\
       df_train[df_train[col]==i].shape[0]))
```




```
O percentual de clientes insatisfeitos quando 'num_var24'==0 é 4.081%
O percentual de clientes insatisfeitos quando 'num_var24'==3 é 0.799%
O percentual de clientes insatisfeitos quando 'num_var24'==6 é 0.000%
```

Só para avaliar quantos valores únicos temos em cada variável com a palavra-chave 'num':

In [96]:
```
for feat in imp:
    print("%s \t Valores únicos:%i \t dtype:%s"%(feat,df_train[feat].nunique(),df_t
```

```
num_var1_0       Valores únicos:3        dtype:int64
num_var4         Valores únicos:8        dtype:int64
num_var5_0       Valores únicos:5        dtype:int64
num_var5         Valores únicos:5        dtype:int64
num_var8_0       Valores únicos:3        dtype:int64
num_var8         Valores únicos:2        dtype:int64
num_var12_0      Valores únicos:6        dtype:int64
num_var12        Valores únicos:4        dtype:int64
num_var13_0      Valores únicos:7        dtype:int64
num_var13_corto_0        Valores únicos:3        dtype:int64
num_var13_corto          Valores únicos:3        dtype:int64
num_var13_largo_0        Valores únicos:7        dtype:int64
num_var13        Valores únicos:7        dtype:int64
num_var14_0      Valores únicos:5        dtype:int64
num_var24_0      Valores únicos:4        dtype:int64
num_var24        Valores únicos:3        dtype:int64
num_var26_0      Valores únicos:9        dtype:int64
num_var25_0      Valores únicos:9        dtype:int64
num_op_var41_hace2       Valores únicos:51       dtype:int64
num_op_var41_hace3       Valores únicos:22       dtype:int64
num_op_var41_ult1        Valores únicos:68       dtype:int64
num_op_var41_ult3        Valores únicos:96       dtype:int64
num_op_var39_hace2       Valores únicos:50       dtype:int64
num_op_var39_hace3       Valores únicos:22       dtype:int64
num_op_var39_ult1        Valores únicos:71       dtype:int64
num_op_var39_ult3        Valores únicos:99       dtype:int64
num_var30_0      Valores únicos:11       dtype:int64
num_var30        Valores únicos:9        dtype:int64
num_var35        Valores únicos:13       dtype:int64
num_var37_med_ult2       Valores únicos:21       dtype:int64
num_var37_0      Valores únicos:22       dtype:int64
num_var39_0      Valores únicos:9        dtype:int64
num_var40_0      Valores únicos:3        dtype:int64
num_var41_0      Valores únicos:9        dtype:int64
num_var42_0      Valores únicos:8        dtype:int64
num_var42        Valores únicos:7        dtype:int64
num_aport_var13_hace3    Valores únicos:8        dtype:int64
num_ent_var16_ult1       Valores únicos:13       dtype:int64
num_var22_hace2          Valores únicos:22       dtype:int64
num_var22_hace3          Valores únicos:19       dtype:int64
num_var22_ult1   Valores únicos:18       dtype:int64
num_var22_ult3   Valores únicos:33       dtype:int64
num_med_var22_ult3       Valores únicos:15       dtype:int64
num_med_var45_ult3       Valores únicos:71       dtype:int64
num_meses_var5_ult3      Valores únicos:4        dtype:int64
num_meses_var8_ult3      Valores únicos:4        dtype:int64
num_meses_var12_ult3     Valores únicos:4        dtype:int64
num_meses_var13_corto_ult3       Valores únicos:4        dtype:int64
num_meses_var39_vig_ult3         Valores únicos:4        dtype:int64
num_op_var39_comer_ult1          Valores únicos:63       dtype:int64
num_op_var39_comer_ult3          Valores únicos:92       dtype:int64
num_op_var41_comer_ult1          Valores únicos:60       dtype:int64
num_op_var41_comer_ult3          Valores únicos:88       dtype:int64
num_op_var41_efect_ult1          Valores únicos:25       dtype:int64
num_op_var41_efect_ult3          Valores únicos:40       dtype:int64
num_op_var39_efect_ult1          Valores únicos:26       dtype:int64
```

```
num_op_var39_efect_ult3          Valores únicos:40       dtype:int64
num_var43_emit_ult1      Valores únicos:27       dtype:int64
num_var43_recib_ult1     Valores únicos:38       dtype:int64
num_trasp_var11_ult1     Valores únicos:18       dtype:int64
num_var45_hace2          Valores únicos:85       dtype:int64
num_var45_hace3          Valores únicos:66       dtype:int64
num_var45_ult1   Valores únicos:94       dtype:int64
num_var45_ult3   Valores únicos:172      dtype:int64
```

Palavra-chave ind

In [97]:
```
imp = [col for col in df_train.columns if 'ind' in col]
print("O número de colunas com a palavra-chave ind é: %i"%(len(imp)))
imp
```

```
O número de colunas com a palavra-chave ind é: 32
```

Out[97]:
```
['ind_var1_0',
 'ind_var5_0',
 'ind_var5',
 'ind_var8_0',
 'ind_var8',
 'ind_var12_0',
 'ind_var12',
 'ind_var13_0',
 'ind_var13_corto_0',
 'ind_var13_corto',
 'ind_var13_largo_0',
 'ind_var13',
 'ind_var14_0',
 'ind_var24_0',
 'ind_var24',
 'ind_var25_cte',
 'ind_var26_0',
 'ind_var26_cte',
 'ind_var25_0',
 'ind_var30_0',
 'ind_var30',
 'ind_var37_cte',
 'ind_var37_0',
 'ind_var39_0',
 'ind_var40_0',
 'ind_var41_0',
 'ind_var10_ult1',
 'ind_var10cte_ult1',
 'ind_var9_cte_ult1',
 'ind_var9_ult1',
 'ind_var43_emit_ult1',
 'ind_var43_recib_ult1']
```

In [98]:
```
random.seed()
imp_feat = random.sample(imp,2)
print("As variáveis escolhidas aleatoriamente são '%s' é '%s'"%(imp_feat[0],imp_fea
```

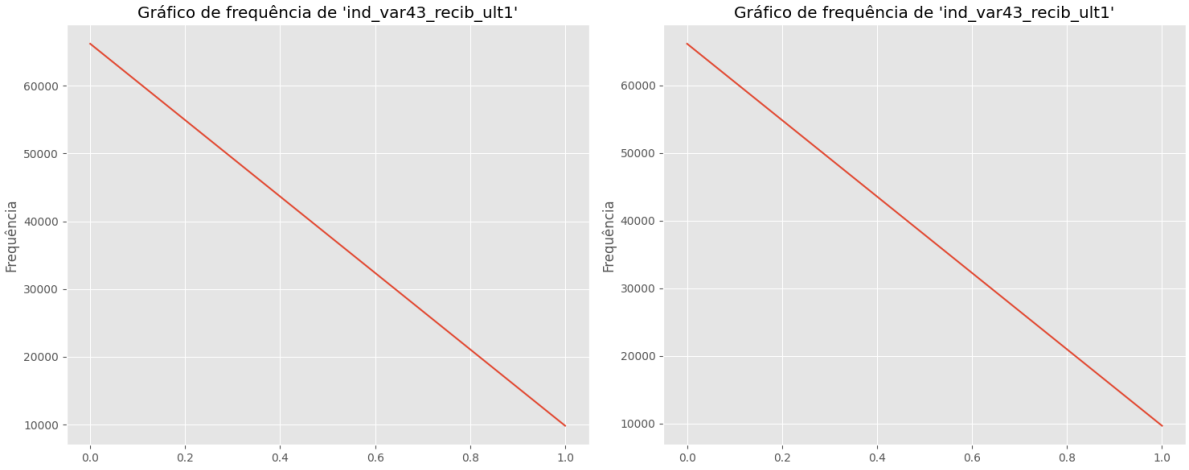
```
As variáveis escolhidas aleatoriamente são 'ind_var43_recib_ult1' é 'ind_var40_0'
```

'ind_var43_recib_ult1'

In [99]:
```
col = imp_feat[0]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'ind_var43_recib_ult1' é 0.0 e 1.0
O valor mínimo e máximo da base de teste para 'ind_var43_recib_ult1' é 0.0 e 1.0
```

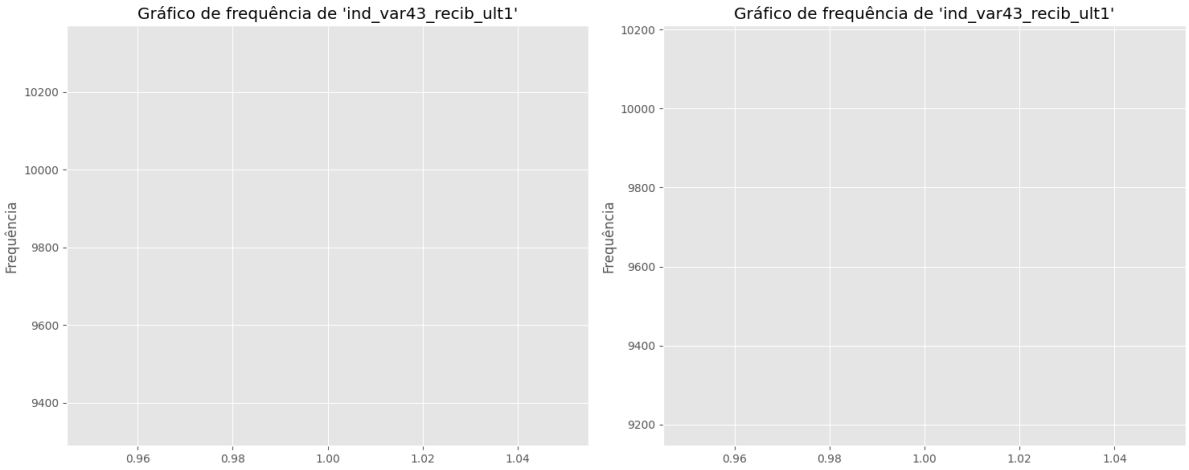
In [100…
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```




```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'ind_var43_recib_ult1':
Valor    Perc%
0    87.069192
1    12.930808
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'ind_var43_recib_ult1':
Valor    Perc%
0    87.069192
1    12.930808
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'ind_var43_recib_ult1':
Valor    Perc%
0    87.235221
1    12.764779
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'ind_var43_recib_ult1':
Valor    Perc%
0    87.235221
1    12.764779
Name: ind_var43_recib_ult1, dtype: float64
```

Aparentemente é uma coluna binária.

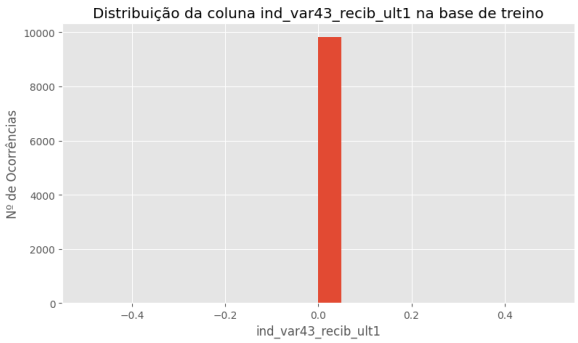
In [101…
```
col = imp_feat[0]
#plotting frequency plots with no zero (most occuring) values
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```

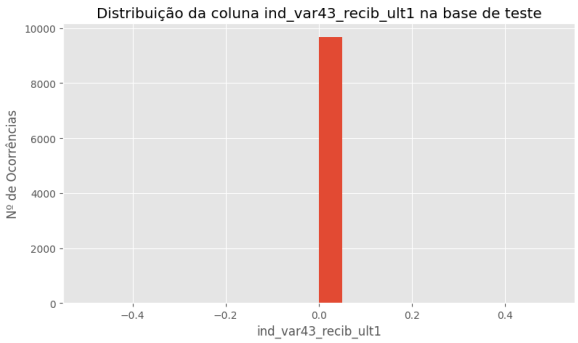



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'ind_var43_recib_ult1':
Valor    Perc%
1    100.0
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'ind_var43_recib_ult1':
Valor    Perc%
1    100.0
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'ind_var43_recib_ult1':
Valor    Perc%
1    100.0
Name: ind_var43_recib_ult1, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'ind_var43_recib_ult1':
Valor    Perc%
1    100.0
Name: ind_var43_recib_ult1, dtype: float64
```

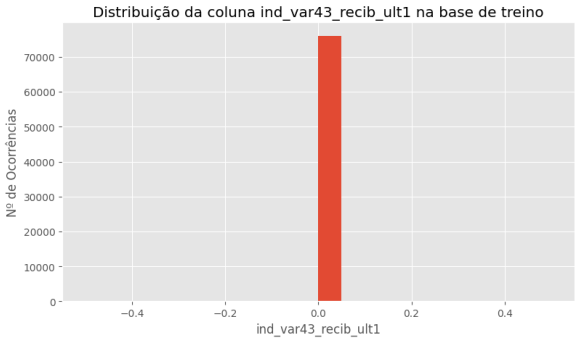
In [102…
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

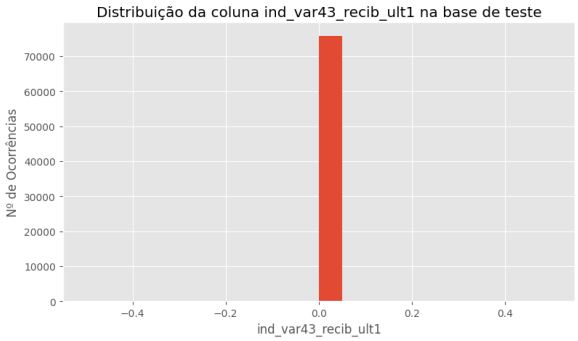
```
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\1883312384.py:2: FutureWarning:
In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpl
ace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newva
ls)`
  df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\1883312384.py:4: FutureWarning:
In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpl
ace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newva
ls)`
  df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
```





In [103…
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```



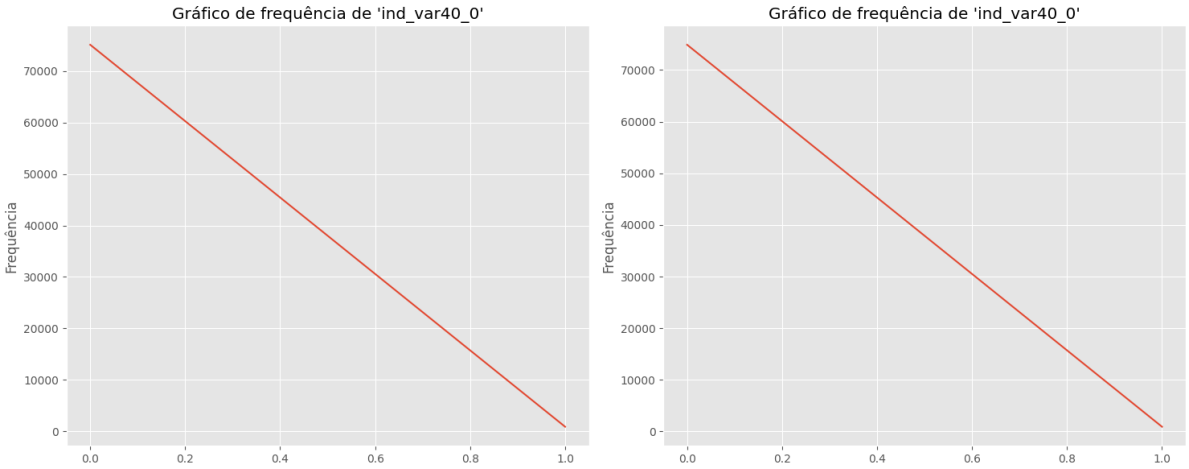


'ind_var40_0'

In [104…
```
col = imp_feat[1]
print("O valor mínimo e máximo da base de treino para '%s' é %.1f e %.1f"%(col,df_t
print("O valor mínimo e máximo da base de teste para '%s' é %.1f e %.1f"%(col,df_te
```

```
O valor mínimo e máximo da base de treino para 'ind_var40_0' é 0.0 e 1.0
O valor mínimo e máximo da base de teste para 'ind_var40_0' é 0.0 e 1.0
```

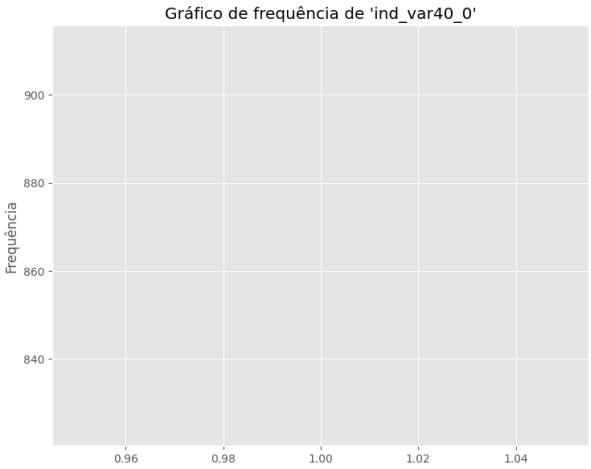
In [105…
```
valuecounts_plot(train=df_train,test=df_test,col=col)
```

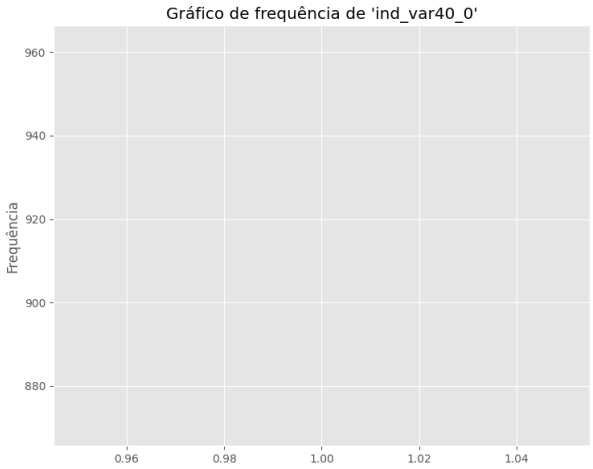



```
**********************************************************************************
******************
Valor percentual (5 maiores) na base de treino para 'ind_var40_0':
Valor    Perc%
0    98.858195
1     1.141805
Name: ind_var40_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de treino para 'ind_var40_0':
Valor    Perc%
0    98.858195
1     1.141805
Name: ind_var40_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 maiores) na base de teste para 'ind_var40_0':
Valor    Perc%
0    98.791844
1     1.208156
Name: ind_var40_0, dtype: float64
**********************************************************************************
******************
Valor percentual (5 menores) na base de teste para 'ind_var40_0':
Valor    Perc%
0    98.791844
1     1.208156
Name: ind_var40_0, dtype: float64
```

In [106…

```
col = imp_feat[1]
valuecounts_plot(train = df_train[df_train[col]!=0], test = df_test[df_test[col]!=0
```



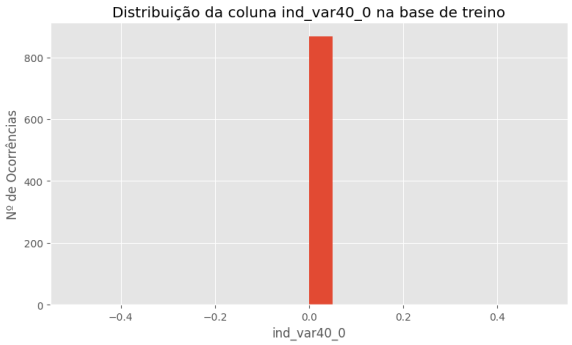


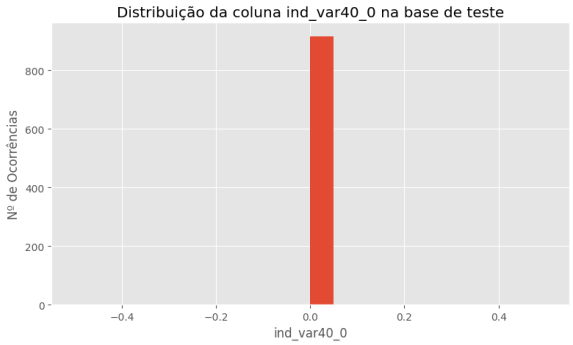

```
**********************************************************************************
*****************
Valor percentual (5 maiores) na base de treino para 'ind_var40_0':
Valor    Perc%
1    100.0
Name: ind_var40_0, dtype: float64
**********************************************************************************
*****************
Valor percentual (5 menores) na base de treino para 'ind_var40_0':
Valor    Perc%
1    100.0
Name: ind_var40_0, dtype: float64
**********************************************************************************
*****************
Valor percentual (5 maiores) na base de teste para 'ind_var40_0':
Valor    Perc%
1    100.0
Name: ind_var40_0, dtype: float64
**********************************************************************************
*****************
Valor percentual (5 menores) na base de teste para 'ind_var40_0':
Valor    Perc%
1    100.0
Name: ind_var40_0, dtype: float64
```

In [107…

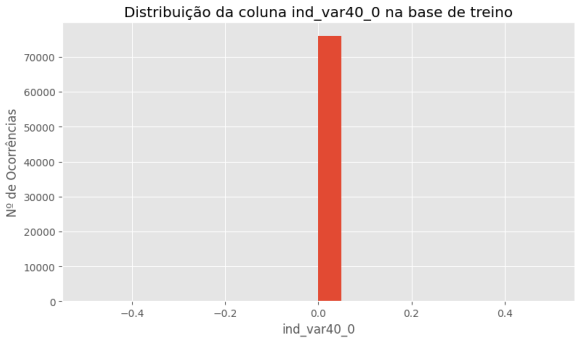
```
df = df_train[df_train[col]!=0].copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test[df_test[col]!=0].copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```

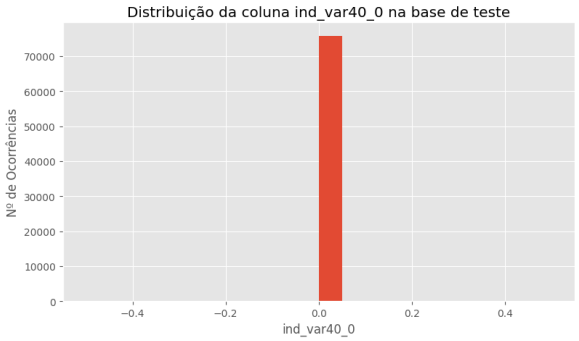
```
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\2973671972.py:2: FutureWarning:
In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpl
ace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newva
ls)`
  df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
C:\Users\gabri\AppData\Local\Temp\ipykernel_26304\2973671972.py:4: FutureWarning:
In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inpl
ace instead of always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newva
ls)`
  df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
```



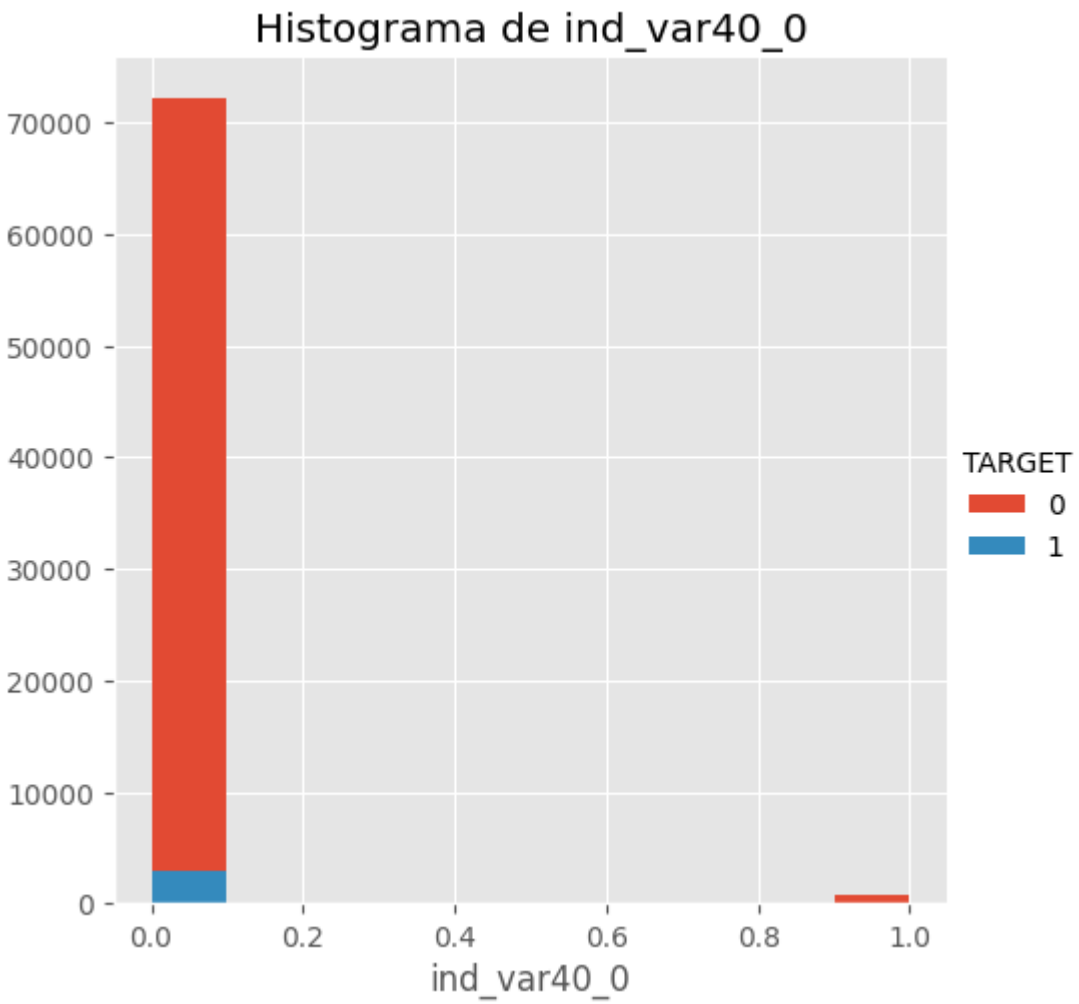


In [108…
```
df = df_train.copy()
df.loc[df[col]!=0,col] = np.log(df.loc[df[col]!=0,col])
df1 = df_test.copy()
df1.loc[df1[col]!=0,col] = np.log(df1.loc[df1[col]!=0,col])
histplot_comb(col,df,df1)
```





In [110…
```
print('')
sns.FacetGrid(data=df_train,hue=y_col,height=5).map(plt.hist,col).add_legend()
plt.title("Histograma de %s"%(col))
plt.show()
for i in df_train[col].unique():
    print("O percentual de clientes insatisfeitos quando '%s'==%i é %.3f%%"%\
      (col,i,df_train[(df_train[y_col]==1)&(df_train[col]==i)][col].shape[0]*100/\
       df_train[df_train[col]==i].shape[0]))
```




```
O percentual de clientes insatisfeitos quando 'ind_var40_0'==0 é 3.953%
O percentual de clientes insatisfeitos quando 'ind_var40_0'==1 é 4.263%
```

Vamos avaliar os valores únicos das colunas com a palavra-chsave 'ind':

In [111…

```
for feat in imp:
    print("%s \t Valores únicos:%i \t dtype:%s"%(feat,df_train[feat].nunique(),df_t
```

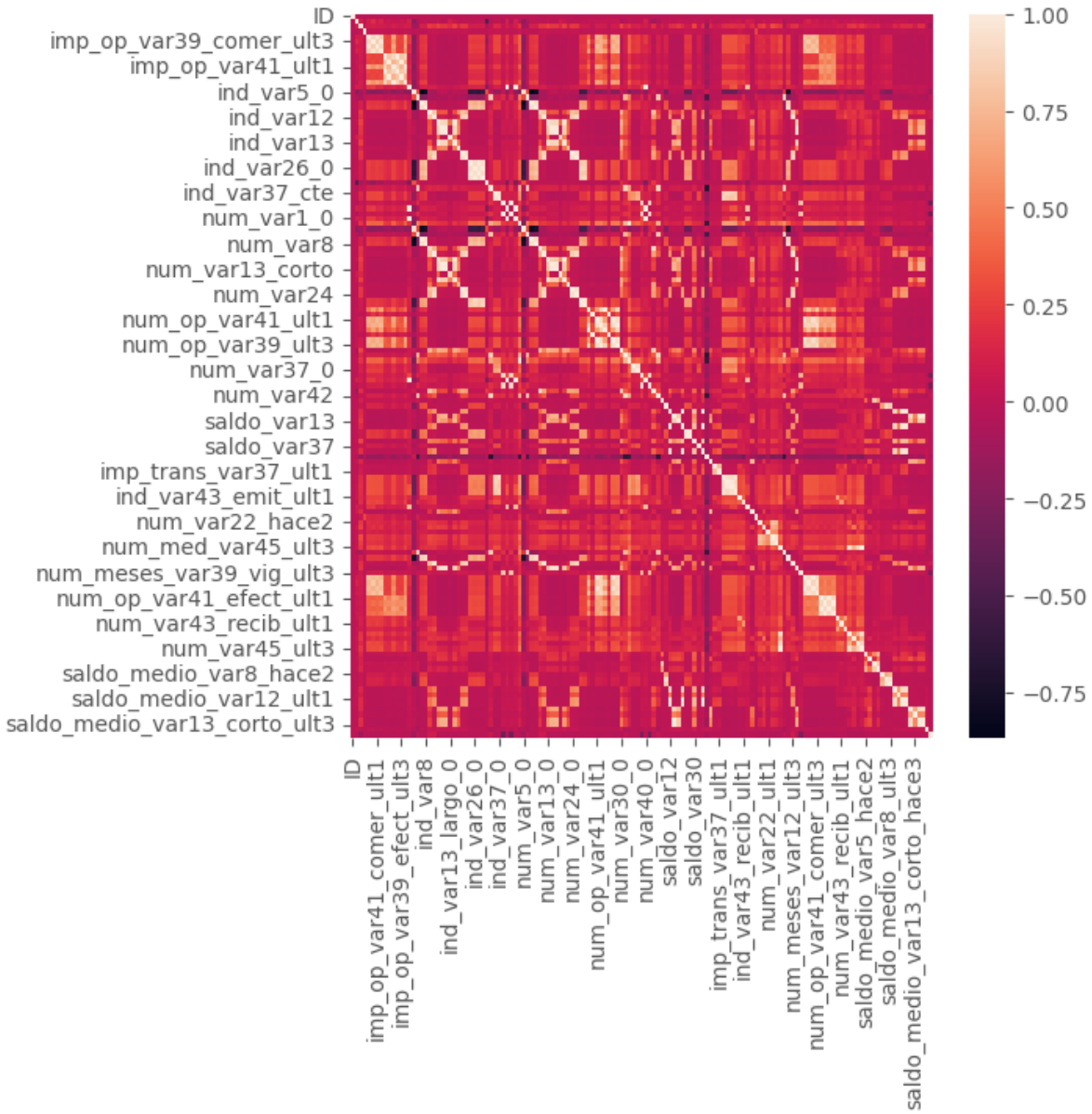
```
ind_var1_0       Valores únicos:2        dtype:int64
ind_var5_0       Valores únicos:2        dtype:int64
ind_var5         Valores únicos:2        dtype:int64
ind_var8_0       Valores únicos:2        dtype:int64
ind_var8         Valores únicos:2        dtype:int64
ind_var12_0      Valores únicos:2        dtype:int64
ind_var12        Valores únicos:2        dtype:int64
ind_var13_0      Valores únicos:2        dtype:int64
ind_var13_corto_0        Valores únicos:2        dtype:int64
ind_var13_corto          Valores únicos:2        dtype:int64
ind_var13_largo_0        Valores únicos:2        dtype:int64
ind_var13        Valores únicos:2        dtype:int64
ind_var14_0      Valores únicos:2        dtype:int64
ind_var24_0      Valores únicos:2        dtype:int64
ind_var24        Valores únicos:2        dtype:int64
ind_var25_cte    Valores únicos:2        dtype:int64
ind_var26_0      Valores únicos:2        dtype:int64
ind_var26_cte    Valores únicos:2        dtype:int64
ind_var25_0      Valores únicos:2        dtype:int64
ind_var30_0      Valores únicos:2        dtype:int64
ind_var30        Valores únicos:2        dtype:int64
ind_var37_cte    Valores únicos:2        dtype:int64
ind_var37_0      Valores únicos:2        dtype:int64
ind_var39_0      Valores únicos:2        dtype:int64
ind_var40_0      Valores únicos:2        dtype:int64
ind_var41_0      Valores únicos:2        dtype:int64
ind_var10_ult1   Valores únicos:2        dtype:int64
ind_var10cte_ult1        Valores únicos:2        dtype:int64
ind_var9_cte_ult1        Valores únicos:2        dtype:int64
ind_var9_ult1    Valores únicos:2        dtype:int64
ind_var43_emit_ult1      Valores únicos:2        dtype:int64
ind_var43_recib_ult1     Valores únicos:2        dtype:int64
```

Resultado interessante, todas as colunas com a palavra-chave 'ind' são binárias!

8. Matriz de Correlação

In [49]:
```
#plotting correlation matrix
corr = df_train.drop('TARGET',axis=1).corr()
plt.figure(figsize=(6,6))
sns.heatmap(corr)
```

Out[49]:
```
<AxesSubplot:>
```



Podemos ver que existem características altamente correlacionadas umas com as outras. Podemos criar um conjunto de dados que remove todos os recursos altamente correlacionados, mantendo um de lado para evitar o overfitting.

In [50]:
```
#correlation values between target
corr_df = pd.DataFrame(df_train.corr()['TARGET']).drop(['ID'])
corr_df.sort_values(by='TARGET',axis=0,ascending=0,inplace=True)
corr_df
```

Out[50]:

	TARGET
TARGET	1.000000
var15	0.111137
var36	0.102919
ind_var8_0	0.046665
num_var8_0	0.046622
...	...
ind_var5	-0.135349
num_var42	-0.135693
num_var30	-0.138289
num_meses_var5_ult3	-0.148253
ind_var30	-0.149811

143 rows × 1 columns

In [51]:
```
t = 10**-4
print(corr_df[abs(corr_df['TARGET'])<=t])
print("Existem %i variáveis que estão abaixo de %.5f de correlação com a coluna 'TA
                                                                                t
```

```
                      TARGET
imp_ent_var16_ult1 -0.000017
num_var37_med_ult2 -0.000029
Existem 2 variáveis que estão abaixo de 0.00010 de correlação com a coluna 'TARGE
T'.
```

Podemos remover essas colunas.

In [52]:
```
feat_to_remove = list(corr_df[abs(corr_df['TARGET'])<=t].index)
for df in [df_train,df_test]:
    df.drop(feat_to_remove,axis=1,inplace=True)
df_train.shape,df_test.shape
```

Out[52]:
```
((76020, 142), (75818, 141))
```

9. Salvando as alterações em arquivo .pkl

In [53]:
```
with open('train_1.pickle', 'wb') as handle:
    pickle.dump(df_train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [54]:
```
with open('test_1.pickle', 'wb') as handle:
    pickle.dump(df_test, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

10. Oversampling das bases

In [ ]:
```
# !pip install imblearn
```

Devido ao desbalanceamento das bases, vamos utilizar a técnica de Oversampling que consiste em ampliar o conjunto de registros que possui menor quantidade de ocorrências. No caso, vamos aumentar os dados de clientes insatisfeitos (TARGET = 1).

In [109…
```
X_train, X_test, y_train, y_test = train_test_split(df_train.drop(['TARGET','ID'],
```

In [115…
```
smote = SMOTE(sampling_strategy='minority')
X_over, y_over = smote.fit_resample(df_train.drop(['TARGET','ID'], axis = 1),df_tra
```

In [118…
```
X_over.shape, y_over.shape
```

Out[118]:
```
((76020, 142), (75818, 141))
```

In [117…
```
(y_over.value_counts()/y_over.count())*100
```

Out[117]:
```
0    50.0
1    50.0
Name: TARGET, dtype: float64
```

In [ ]:
```

```

SANTANDER DATA MASTER - CIENTISTA DE DADOS

Questão a) Maximizar o lucro a partir de uma ação de retenção aos clientes insatisfeitos.

Descrição: Um falso positivo ocorre quando classificamos um cliente como insatisfeito, mas ela não se comporta como tal. Neste caso, o custo de preparar e executar uma ação de retenção é um valor fixo de 10,00 reais por cliente. Nada é ganho pois a ação de retenção não é capaz de mudar o comportamento do cliente. Um falso negativo ocorre quando um cliente é previsto como satisfeito, mas na verdade ele estava insatisfeito. Neste caso, nenhum dinheiro foi gasto e nada foi ganho. Um verdadeiro positivo é um cliente que estava insatisfeito e foi alvo de uma ação de retenção. O benefício neste caso é o lucro da ação (100,00 reais) menos os custos relacionados à ação de retenção (10,00 reais). Por fim, um verdadeiro negativo é um cliente insatisfeito e que não é alvo de nenhuma ação. O benefício neste caso é zero, isto é, nenhum custo, mas nenhum lucro.

In [ ]:
```
# !pip install lightgbm
# !pip install bayesian-optimization
```

In [1]:
```
#Imports

#Manipulação dos Dados
import pandas as pd
import numpy as np

import time

#Visualização dos Dados
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
from scipy import stats

import pickle
import xgboost as xgb
import lightgbm as lgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score,GridSearchCV,
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report, p
from sklearn import ensemble,model_selection,linear_model,tree,calibration
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
```

```
from sklearn.ensemble import GradientBoostingClassifier
from bayes_opt import BayesianOptimization
from keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam,
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout, LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
import tensorflow as tf
```

In [2]:
```
df_train = pd.read_pickle('./train_1.pickle')
df_test = pd.read_pickle('./test_1.pickle')
```

In [3]:
```
df_train.shape,df_test.shape
```

Out[3]:
```
((76020, 142), (75818, 141))
```

1. Dividindo a base

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(df_train.drop(['TARGET','ID'],
```

2. Oversampling das bases

In [5]:
```
smote = SMOTE(sampling_strategy='minority')
X_over, y_over = smote.fit_resample(X_train,y_train)
```

In [6]:
```
X_over.shape, y_over.shape
```

Out[6]:
```
((116820, 140), (116820,))
```

In [7]:
```
(y_over.value_counts()/y_over.count())*100
```

Out[7]:
```
0    50.0
1    50.0
Name: TARGET, dtype: float64
```

Show! Agora temos a mesma quantidade de 0's e 1's.

Modeling

Vamos utilizar uma função que traga as principais informações que precisamos, que são: acurácia, curva ROC, tempo de execução, matriz de confusão e o lucro.

In [8]:
```
def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
    t0=time.time()
    if verbose == False:
        model.fit(X_train,y_train, verbose=0)
    else:
```

```
        model.fit(X_train,y_train)

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    time_taken = time.time()-t0
    print("Acurácia = {}".format(accuracy))
    print("ROC Característica de Operação do Receptor = {}".format(roc_auc))
    print("Tempo utilizado = {}".format(time_taken))
    print(classification_report(y_test,y_pred,digits=5))
    plot_confusion_matrix(model, X_test, y_test)
    plot_roc_curve(model, X_test, y_test)

    conf_matrix = confusion_matrix(y_test, y_pred)

    FP = conf_matrix[0][1] # Falsos positivos
    TP = conf_matrix[1][1] # Verdadeiros positivos

    lucro = 90*TP - 10*FP # Lucro da acao de retencao (os
                          # verdadeiros e falsos negativos
                          # não contribuem em
                          # nada no Lucro)

    print("Lucro obtido = {}".format(lucro))
    return model, accuracy, roc_auc, time_taken, lucro
```

Para identificar o melhor modelo, vamos testar os seguintes algoritmos:

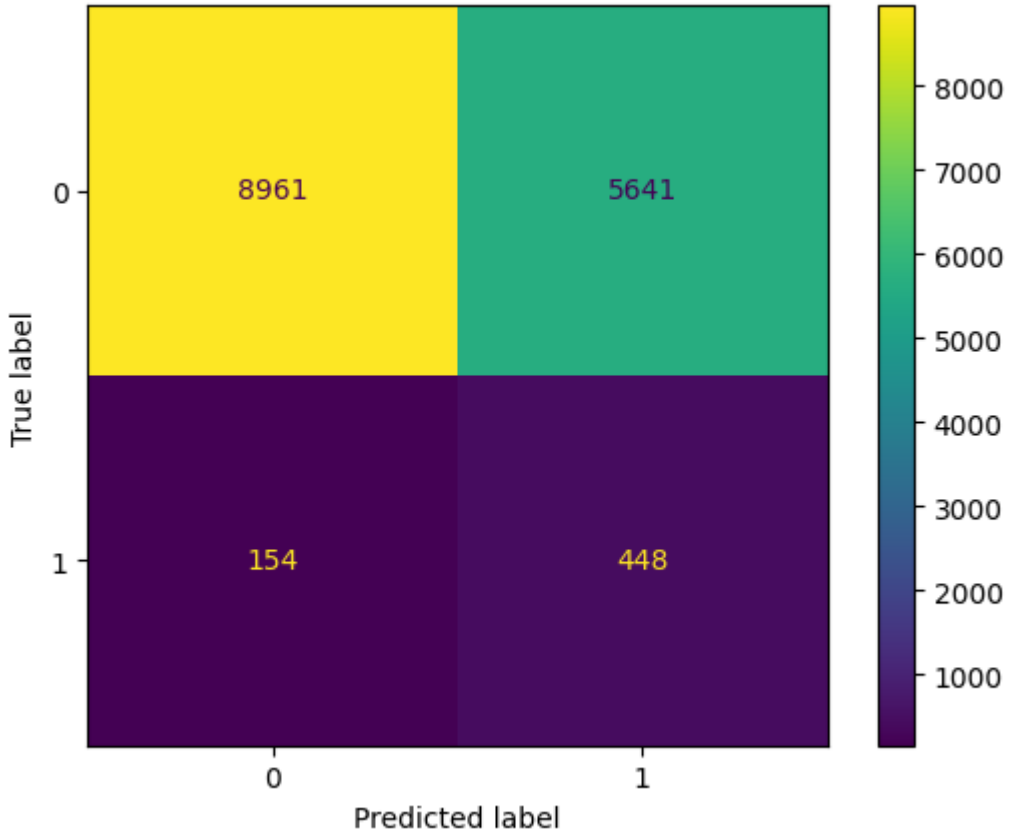
- Logistic Regression
- Decision Tree
- XGB Classifier
- LightGBM
- Naive Bayes
- KNN
- Neural Network
- AdaBoost
- Random Forest

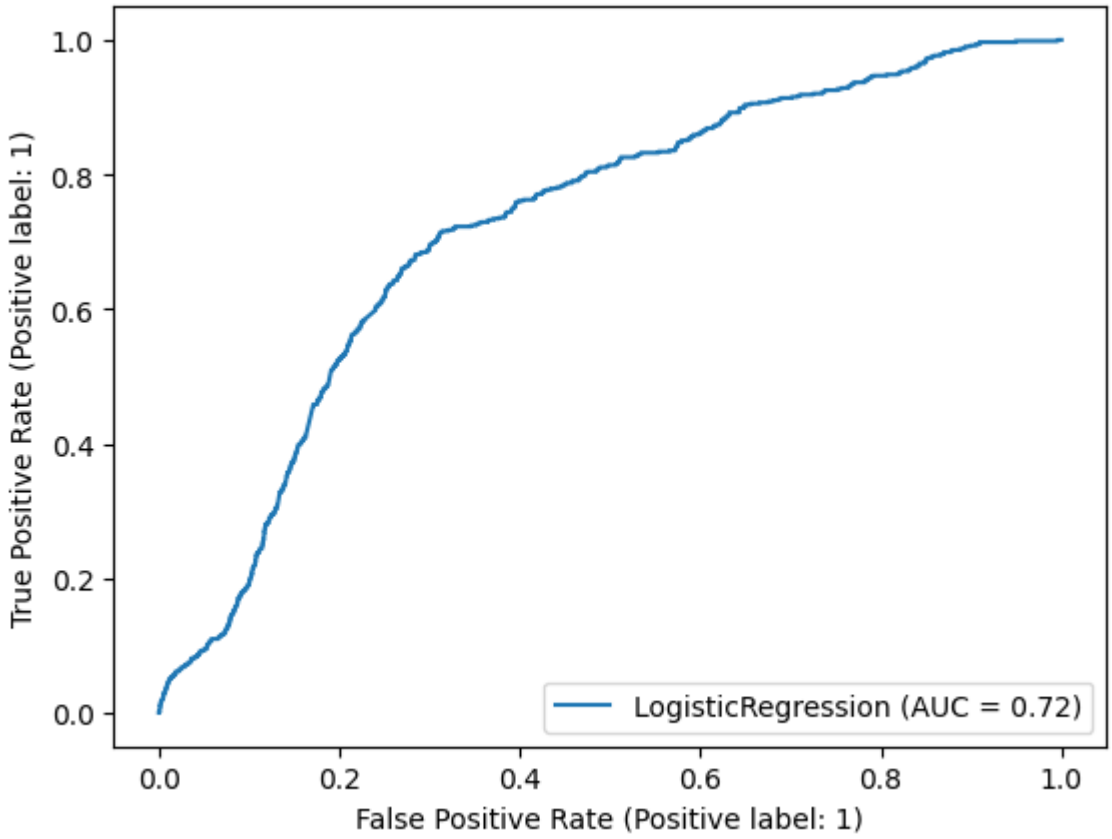
Logistic Regression

In [9]:
```
model = linear_model.LogisticRegression(C=100.0, class_weight='balanced')
```

In [10]:
```
params_lr = {'penalty': 'elasticnet', 'l1_ratio':0.5, 'solver': 'saga'}

model_lr = LogisticRegression(**params_lr)
model_lr, accuracy_lr, roc_auc_lr, tt_lr, lucro_lr = run_model(model_lr, X_over, y_
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Acurácia = 0.61885030255196
ROC Característica de Operação do Receptor = 0.6789345518135458
Tempo utilizado = 48.748918533325195
              precision    recall  f1-score   support

           0    0.98310   0.61368   0.75566     14602
           1    0.07358   0.74419   0.13391       602

    accuracy                        0.61885     15204
   macro avg    0.52834   0.67893   0.44479     15204
weighted avg    0.94709   0.61885   0.73104     15204

Lucro obtido = -16090
```



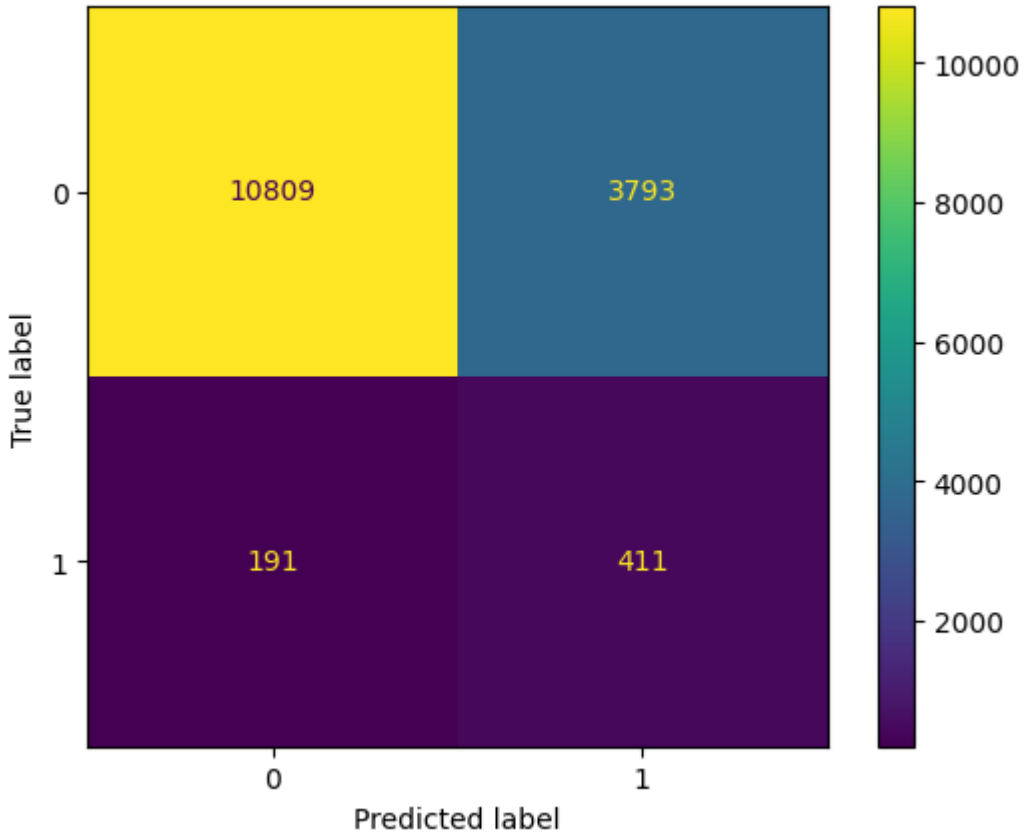


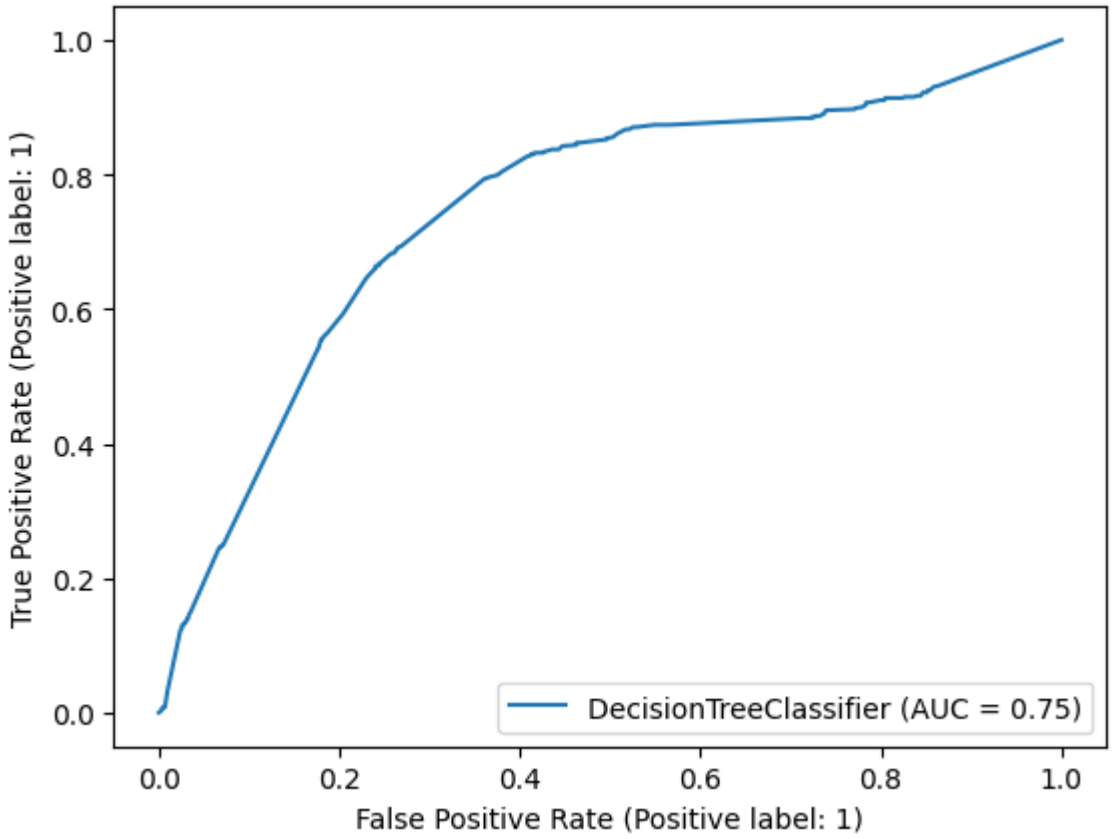
Decision Tree

In [11]:
```
params_dt = {'max_depth': 12,
             'max_features': "sqrt"}

model_dt = DecisionTreeClassifier(**params_dt)
model_dt, accuracy_dt, roc_auc_dt, tt_dt, lucro_dt = run_model(model_dt, X_over, y_
```

```
Acurácia = 0.7379636937647988
ROC Característica de Operação do Receptor = 0.7114826576798973
Tempo utilizado = 0.8419678211212158
              precision    recall  f1-score   support

           0    0.98264   0.74024   0.84439     14602
           1    0.09776   0.68272   0.17104       602

    accuracy                        0.73796     15204
   macro avg    0.54020   0.71148   0.50771     15204
weighted avg    0.94760   0.73796   0.81773     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = -940
```



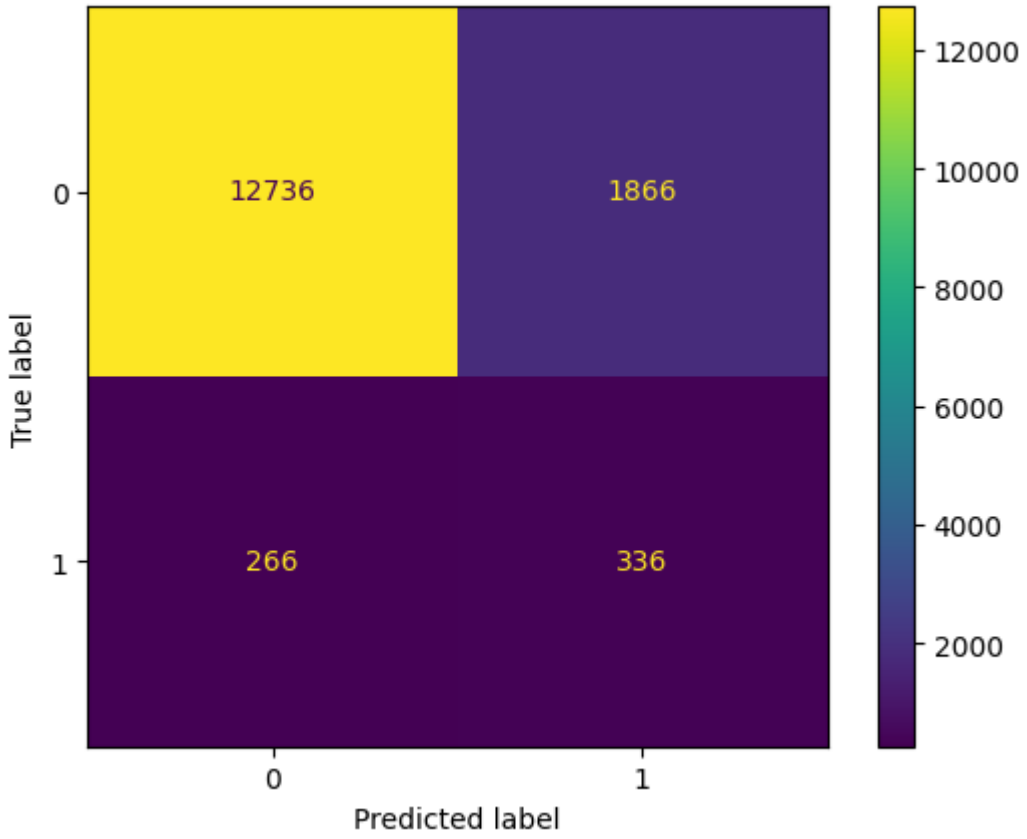


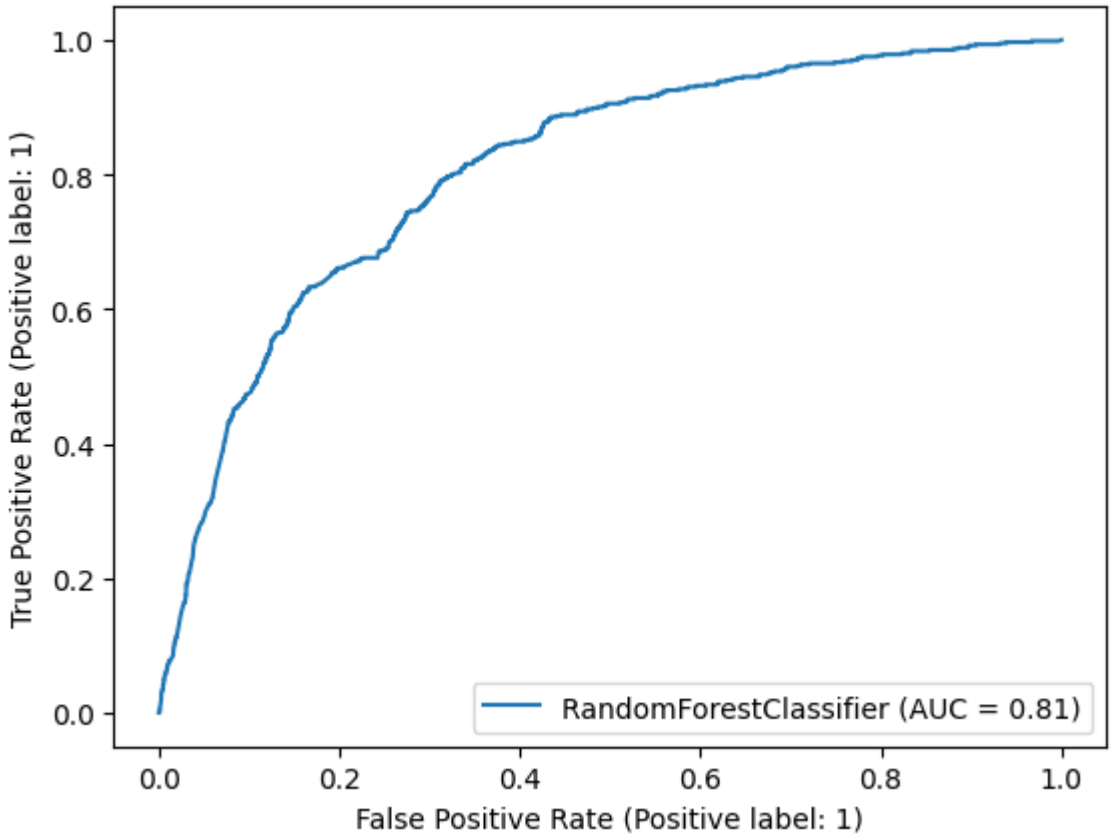
Random Forest

In [12]:
```
params_rf = {'max_depth': 16,
             'min_samples_leaf': 1,
             'min_samples_split': 2,
             'n_estimators': 100,
             'random_state': 12345}

model_rf = RandomForestClassifier(**params_rf)
model_rf, accuracy_rf, roc_auc_rf, tt_rf, lucro_rf = run_model(model_rf, X_over, y_
```

```
Acurácia = 0.8597737437516443
ROC Característica de Operação do Receptor = 0.7151744106414222
Tempo utilizado = 36.16036605834961
              precision    recall  f1-score   support

           0    0.97954   0.87221   0.92276     14602
           1    0.15259   0.55814   0.23966       602

    accuracy                        0.85977     15204
   macro avg    0.56607   0.71517   0.58121     15204
weighted avg    0.94680   0.85977   0.89572     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = 11580
```



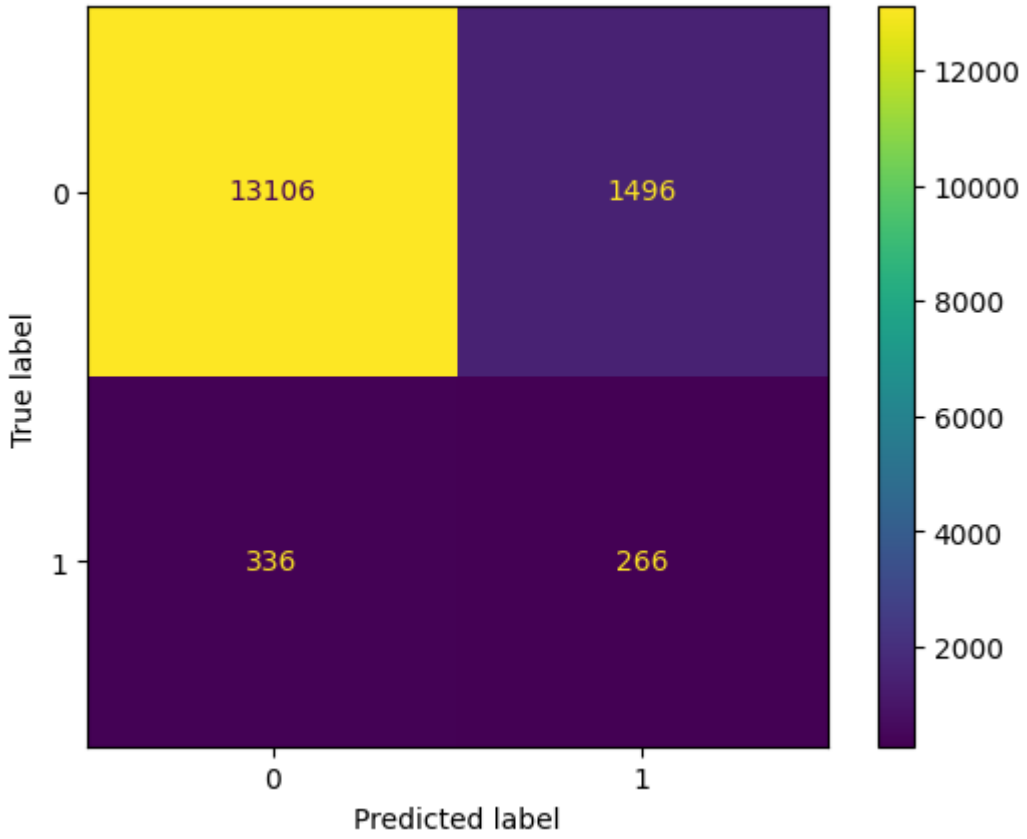


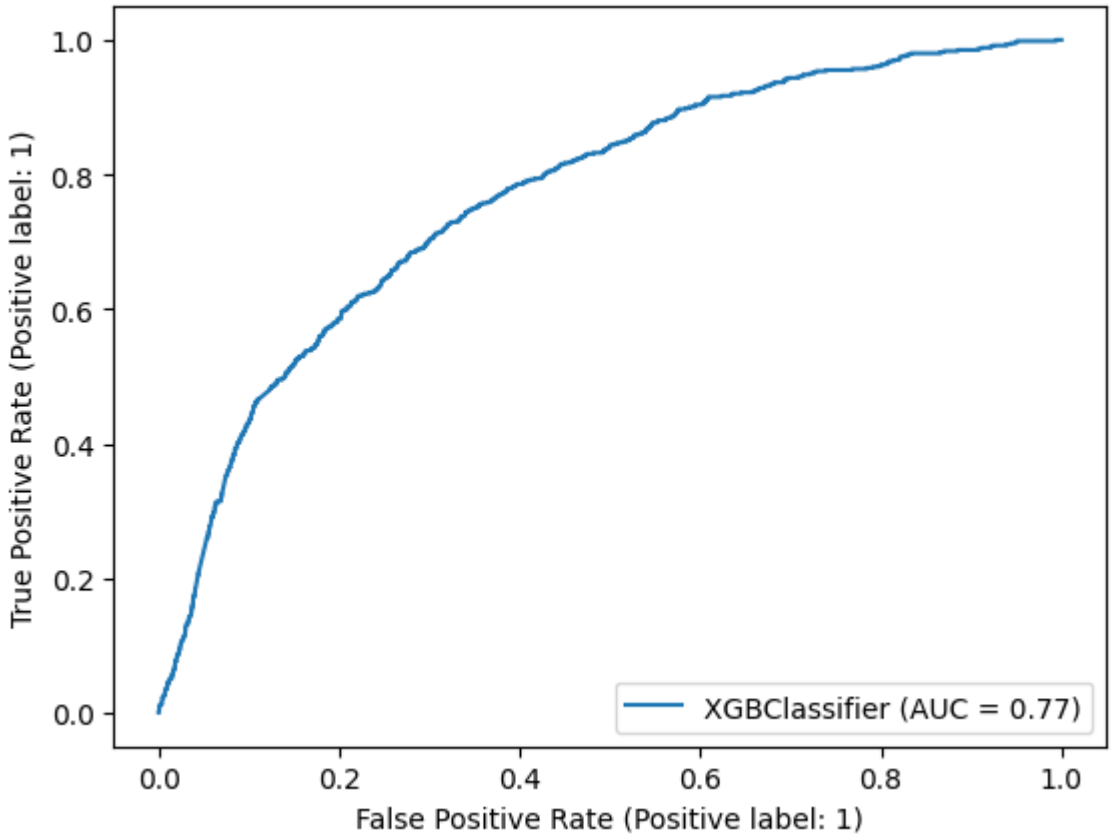
XGB Classifier

In [13]:
```
params_xgb ={'n_estimators': 500,
             'max_depth': 16}

model_xgb = xgb.XGBClassifier(**params_xgb)
model_xgb, accuracy_xgb, roc_auc_xgb, tt_xgb, lucro_xgb = run_model(model_xgb, X_ov
```

```
Acurácia = 0.8795053933175481
ROC Característica de Operação do Receptor = 0.6697043730868342
Tempo utilizado = 367.7983675003052
              precision    recall  f1-score   support

           0    0.97500   0.89755   0.93467     14602
           1    0.15096   0.44186   0.22504       602

    accuracy                        0.87951     15204
   macro avg    0.56298   0.66970   0.57986     15204
weighted avg    0.94238   0.87951   0.90658     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = 8980
```





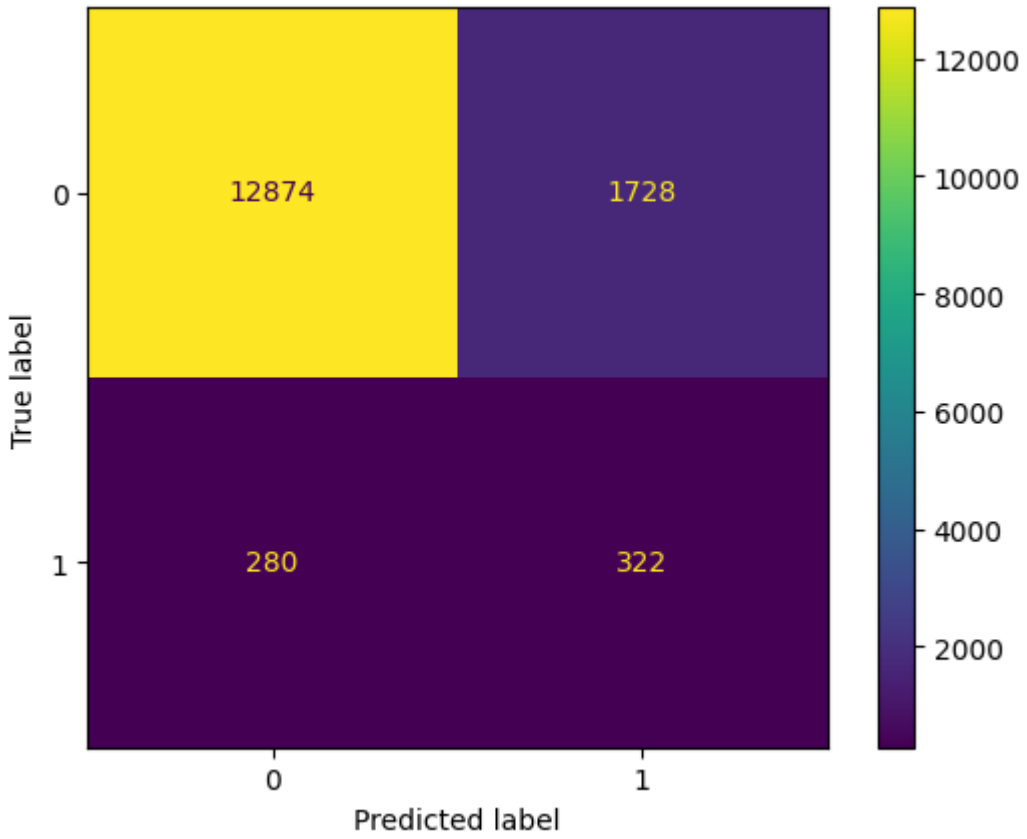
LightGBM Classifier

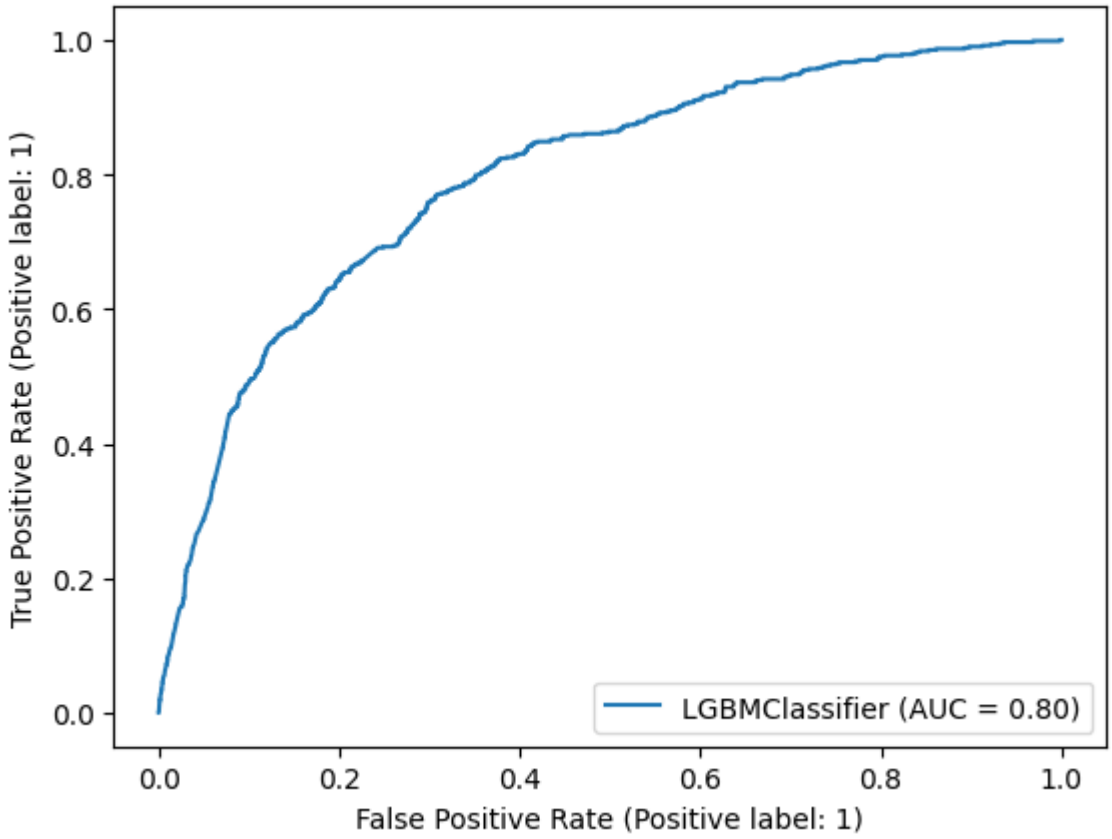
In [14]:
```
model_lgb = lgb.LGBMClassifier(n_jobs=-1,
                       nthread=-1,
                       scale_pos_weight=1.,
                       learning_rate=0.01,
                       colsample_bytree = 0.5,
                       subsample = 0.8,
                       objective='binary',
                       n_estimators=1000,
                       reg_alpha = 0.3,
                       max_depth=7,
                       random_state=42
                      )
model_lgb, accuracy_lgb, roc_auc_lgb, tt_lgb, lucro_lgb = run_model(model_lgb, X_ov
```

```
[LightGBM] [Warning] num_threads is set with n_jobs=-1, nthread=-1 will be ignore
d. Current value: num_threads=-1
Acurácia = 0.8679294922388845
ROC Característica de Operação do Receptor = 0.7082718837495979
Tempo utilizado = 41.23930263519287
              precision    recall  f1-score   support

           0    0.97871   0.88166   0.92766     14602
           1    0.15707   0.53488   0.24284       602

    accuracy                        0.86793     15204
   macro avg    0.56789   0.70827   0.58525     15204
weighted avg    0.94618   0.86793   0.90054     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecate
d; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confus
ionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Funct
ion :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` o
r :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

```
Lucro obtido = 11700
```



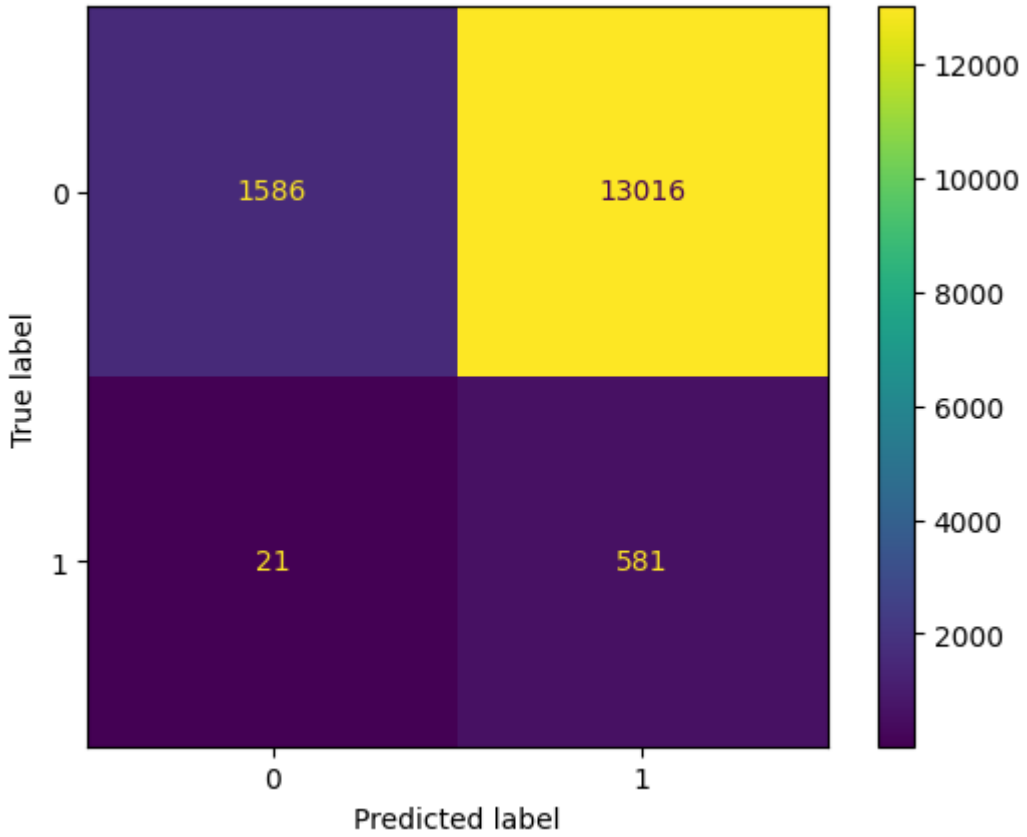


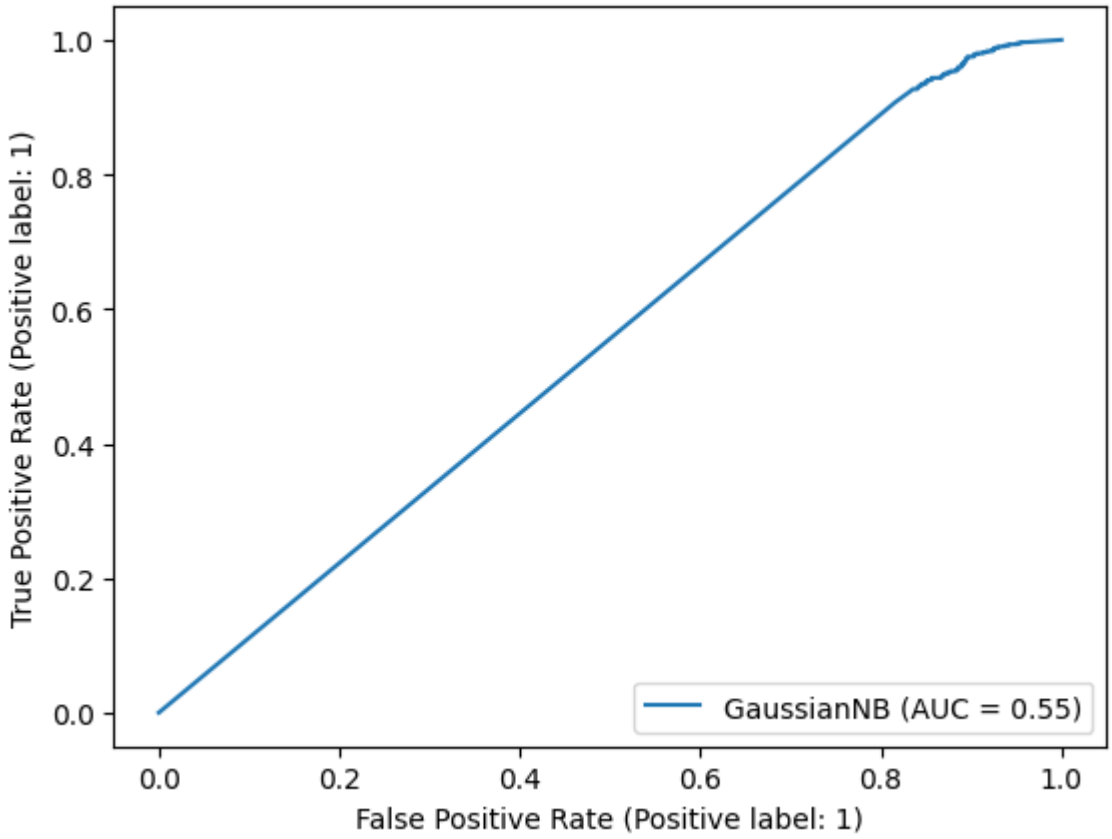
Naive Bayes

In [15]:
```
params_nb = {}

model_nb = GaussianNB(**params_nb)
model_nb, accuracy_nb, roc_auc_nb, tt_nb, lucro_nb = run_model(model_nb, X_over, y_
```

```
Acurácia = 0.14252828203104445
ROC Característica de Operação do Receptor = 0.5368657686267889
Tempo utilizado = 0.42981481552124023
              precision    recall  f1-score   support

           0    0.98693   0.10862   0.19569     14602
           1    0.04273   0.96512   0.08184       602

    accuracy                        0.14253     15204
   macro avg    0.51483   0.53687   0.13877     15204
weighted avg    0.94955   0.14253   0.19119     15204

Lucro obtido = -77870
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



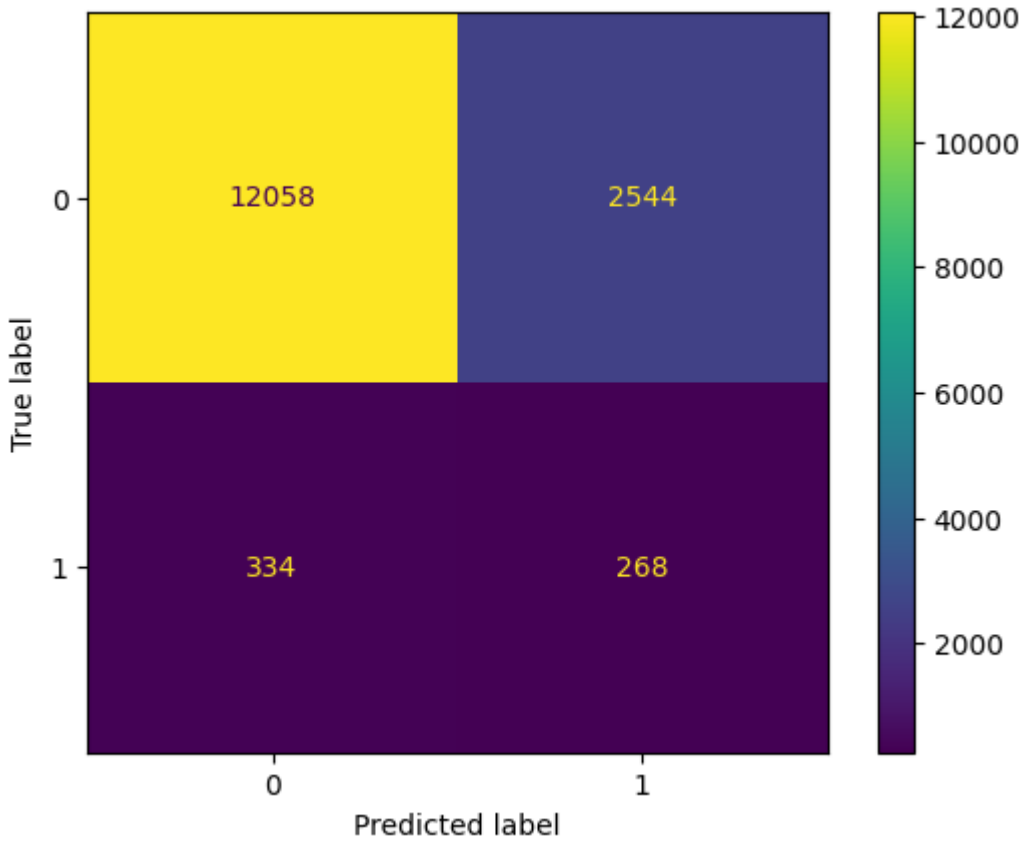


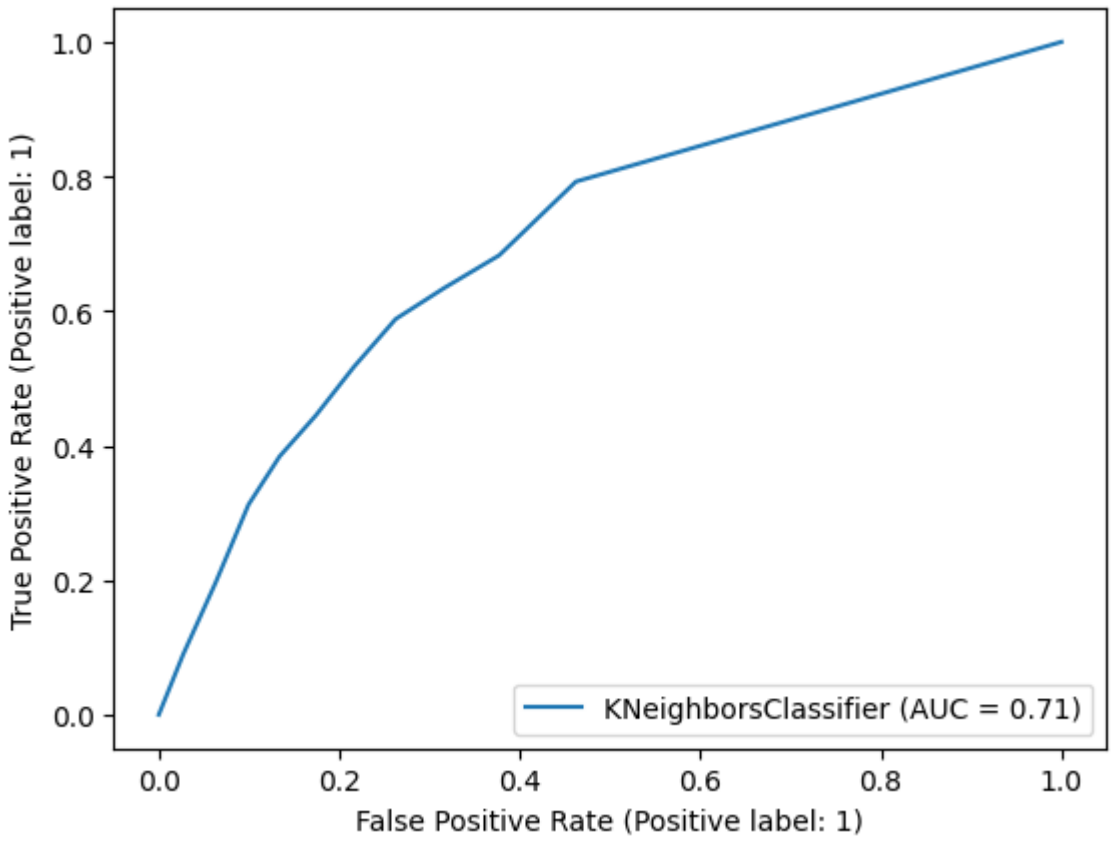
KNN

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
params_kn = {'n_neighbors':10, 'algorithm': 'kd_tree', 'n_jobs':4}

model_kn = KNeighborsClassifier(**params_kn)
model_kn, accuracy_kn, roc_auc_kn, tt_kn, lucro_kn = run_model(model_kn, X_over, y_
```

```
Acurácia = 0.8107077084977637
ROC Característica de Operação do Receptor = 0.6354800075172882
Tempo utilizado = 33.509093046188354
              precision    recall  f1-score   support

           0    0.97305   0.82578   0.89338     14602
           1    0.09531   0.44518   0.15700       602

    accuracy                        0.81071     15204
   macro avg    0.53418   0.63548   0.52519     15204
weighted avg    0.93829   0.81071   0.86423     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = -1320
```





Neural Network

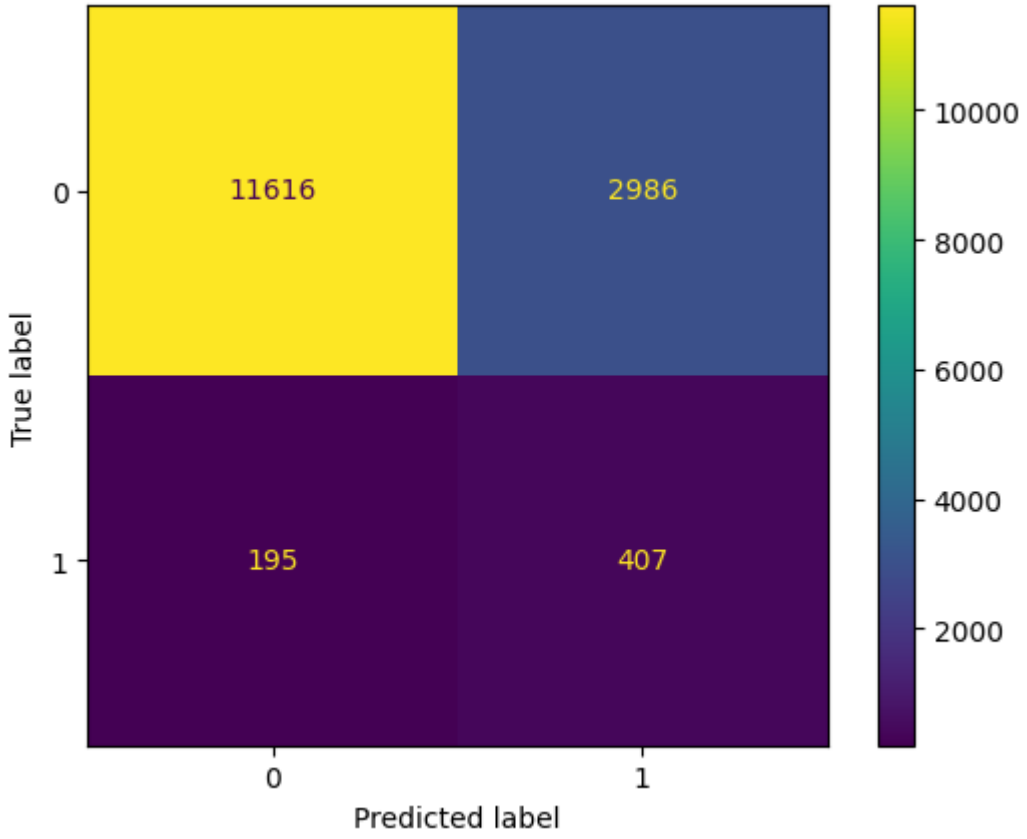
In [17]:
```
# Neural Network (Multilayer Perceptron)
params_nn = {'hidden_layer_sizes': (30,30,30),
             'activation': 'logistic',
             'solver': 'lbfgs',
             'max_iter': 100}

model_nn = MLPClassifier(**params_nn)
model_nn, accuracy_nn, roc_auc_nn, tt_nn, lucro_nn = run_model(model_nn, X_over, y_
```

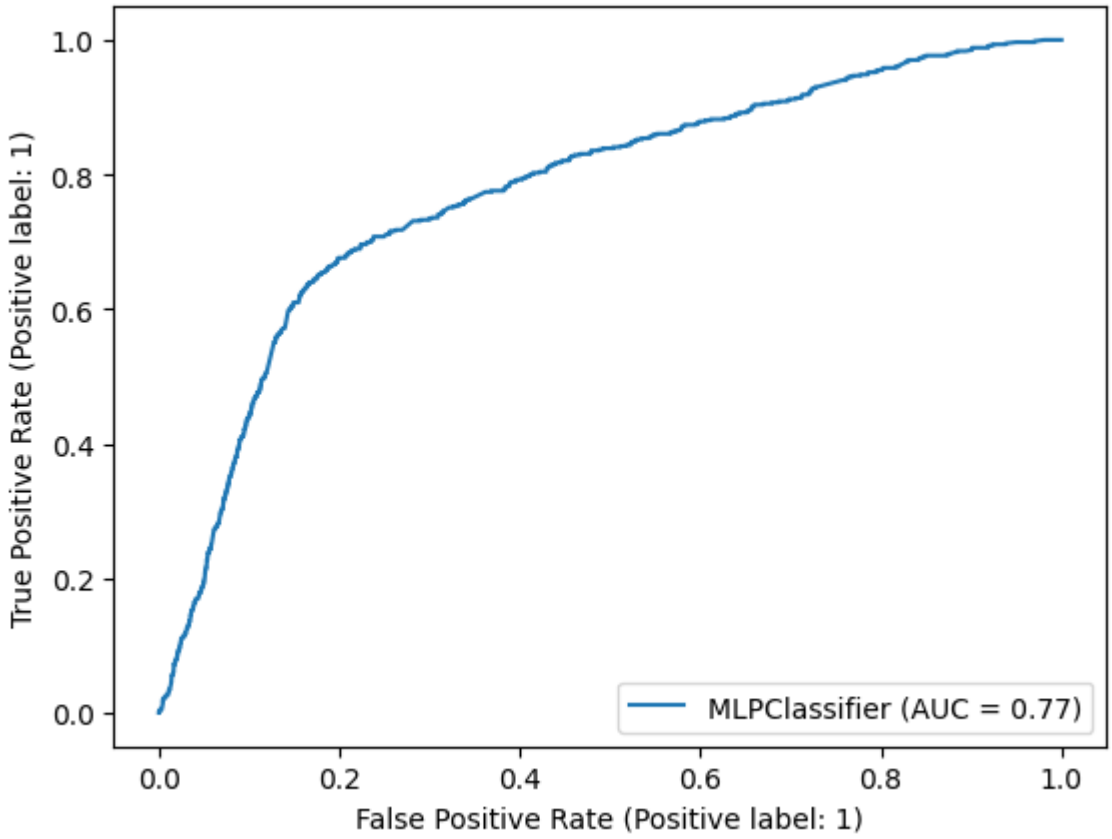
```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ne
ural_network\_multilayer_perceptron.py:559: ConvergenceWarning: lbfgs failed to co
nverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecate
d; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confus
ionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
Acurácia = 0.790778742436201
ROC Característica de Operação do Receptor = 0.7357935994750638
Tempo utilizado = 121.93976879119873
              precision    recall  f1-score   support

           0    0.98349   0.79551   0.87957     14602
           1    0.11995   0.67608   0.20375       602

    accuracy                        0.79078     15204
   macro avg    0.55172   0.73579   0.54166     15204
weighted avg    0.94930   0.79078   0.85281     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

```
Lucro obtido = 6770
```



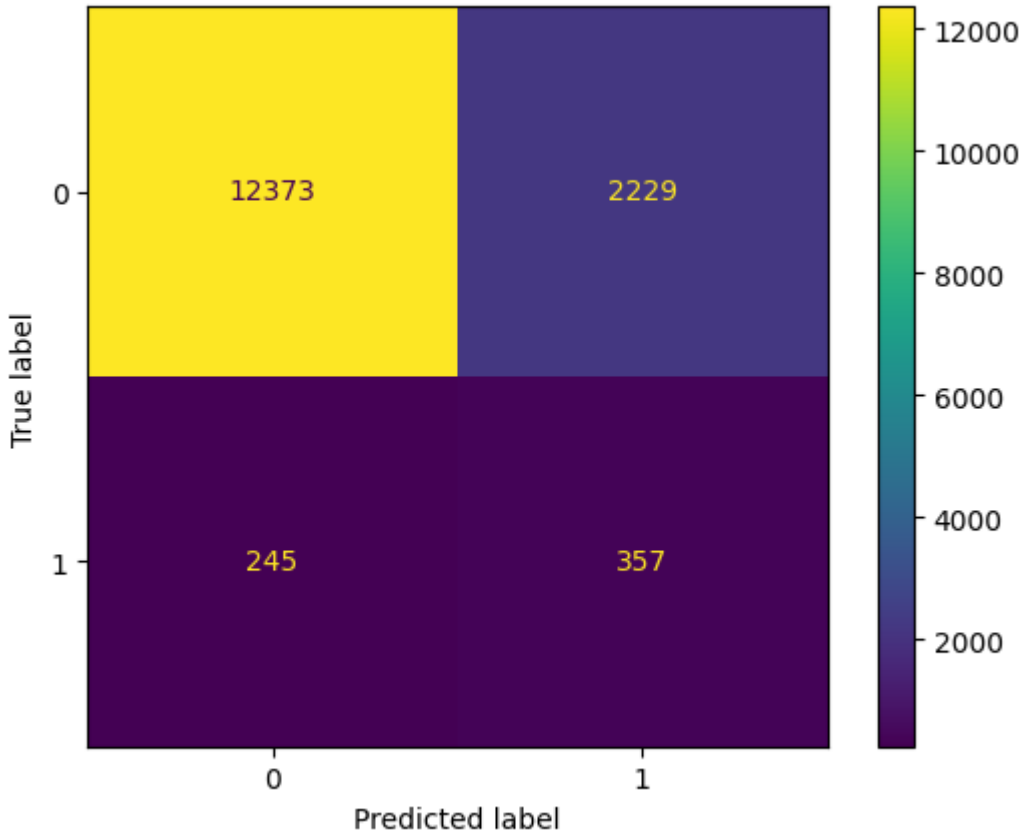


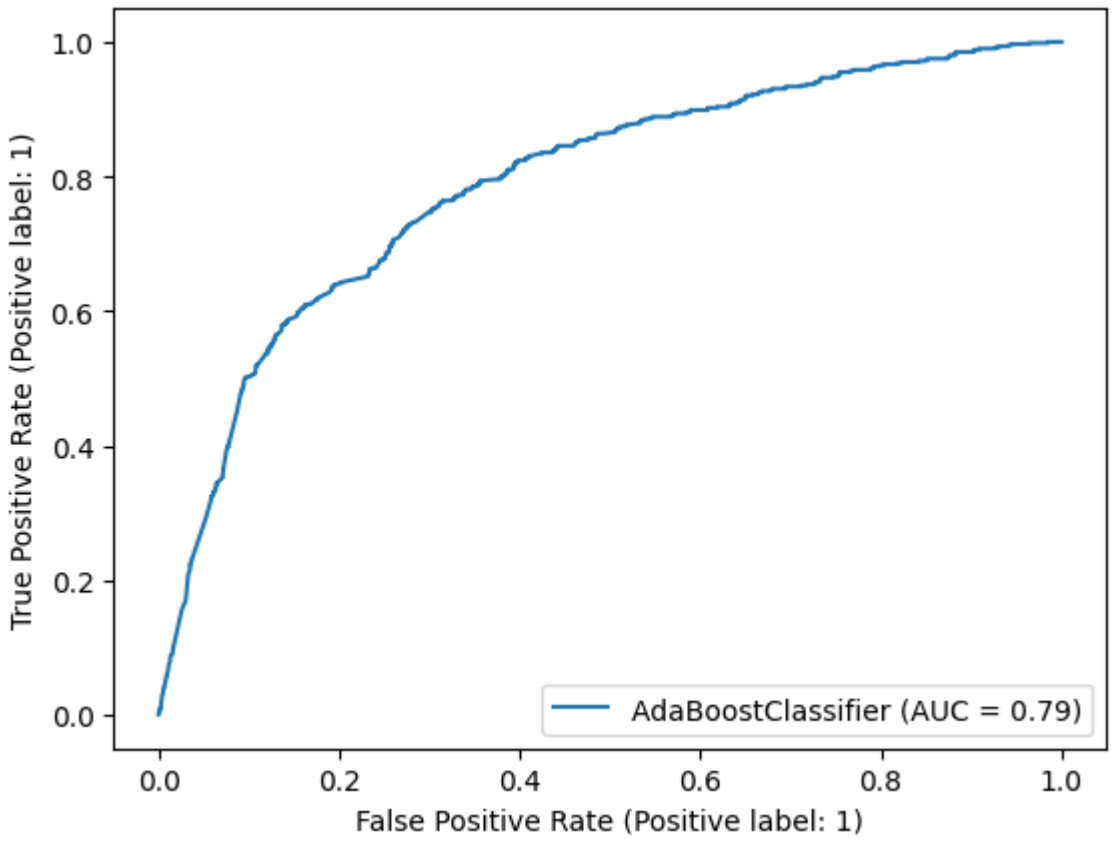
AdaBoost

In [18]:
```
params_adab ={'n_estimators': 500,
              'random_state': 12345}

model_adab = AdaBoostClassifier(**params_adab)
model_adab, accuracy_adab, roc_auc_adab, tt_adab, lucro_adab = run_model(model_adab
```

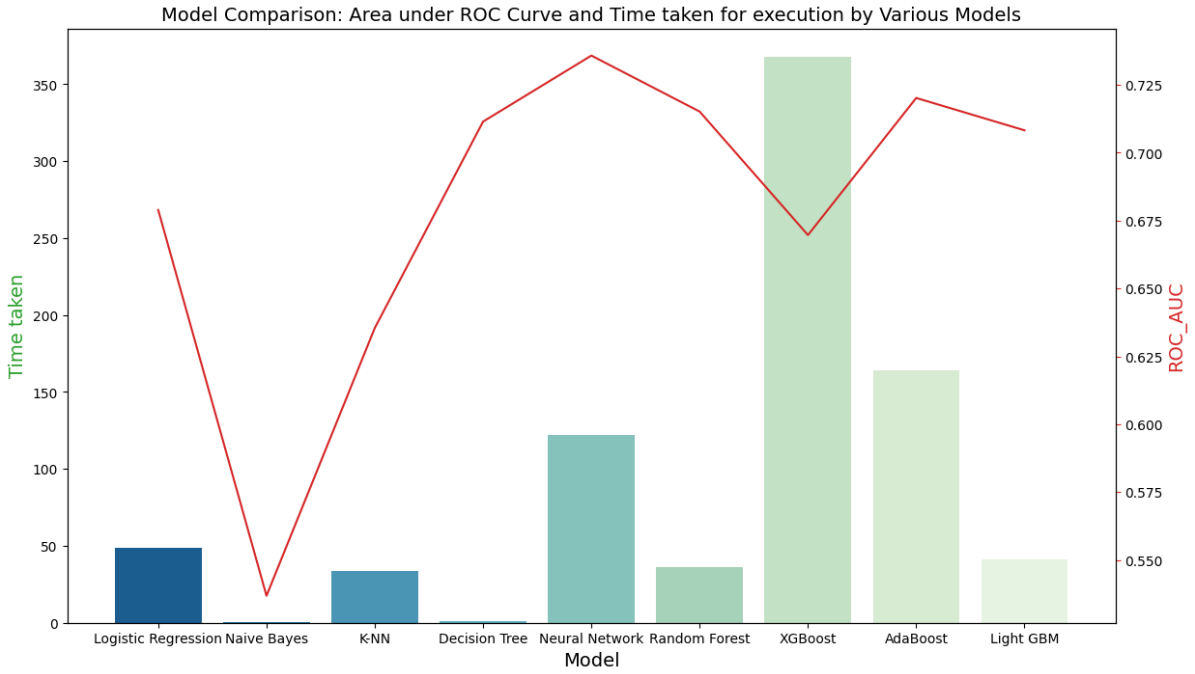
```
Acurácia = 0.837279663246514
ROC Característica de Operação do Receptor = 0.7201864669701188
Tempo utilizado = 164.10076141357422
              precision    recall  f1-score   support

           0    0.98058   0.84735   0.90911     14602
           1    0.13805   0.59302   0.22396       602

    accuracy                        0.83728     15204
   macro avg    0.55932   0.72019   0.56654     15204
weighted avg    0.94722   0.83728   0.88198     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = 9840
```



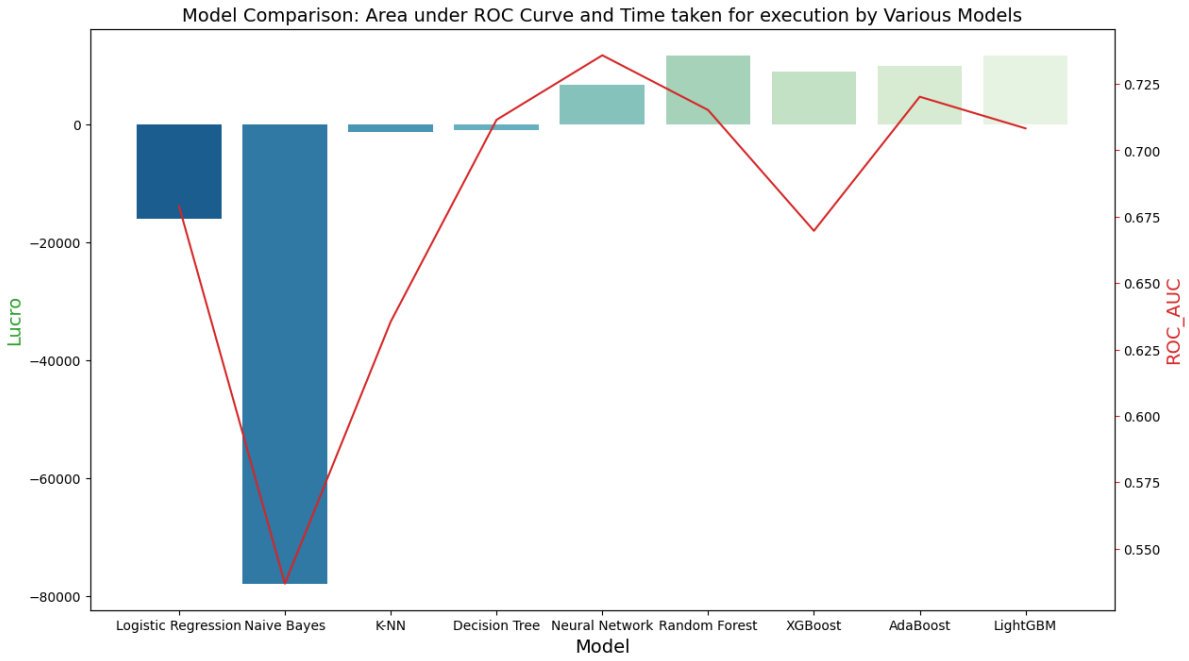


In [19]:
```
roc_auc_scores = [roc_auc_lr, roc_auc_nb, roc_auc_kn, roc_auc_dt, roc_auc_nn, roc_a
tt = [tt_lr, tt_nb, tt_kn, tt_dt, tt_nn, tt_rf, tt_xgb, tt_adab, tt_lgb]

model_data = {'Model': ['Logistic Regression','Naive Bayes','K-NN','Decision Tree',
              'ROC_AUC': roc_auc_scores,
              'Time taken': tt}
data = pd.DataFrame(model_data)

fig, ax1 = plt.subplots(figsize=(14,8))
ax1.set_title('Model Comparison: Area under ROC Curve and Time taken for execution
color = 'tab:green'
ax1.set_xlabel('Model', fontsize=14)
ax1.set_ylabel('Time taken', fontsize=14, color=color)
ax2 = sns.barplot(x='Model', y='Time taken', data = data, palette='GnBu_r')
ax1.tick_params(axis='y')
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('ROC_AUC', fontsize=14, color=color)
ax2 = sns.lineplot(x='Model', y='ROC_AUC', data = data, sort=False, color=color)
ax2.tick_params(axis='y', color=color)
```



In [20]:

```
roc_auc_scores = [roc_auc_lr, roc_auc_nb, roc_auc_kn, roc_auc_dt, roc_auc_nn, roc_a
lucro = [lucro_lr, lucro_nb, lucro_kn, lucro_dt, lucro_nn, lucro_rf, lucro_xgb, luc

model_data = {'Model': ['Logistic Regression','Naive Bayes','K-NN','Decision Tree',
              'ROC_AUC': roc_auc_scores,
              'Lucro': lucro}
data = pd.DataFrame(model_data)

fig, ax1 = plt.subplots(figsize=(14,8))
ax1.set_title('Model Comparison: Area under ROC Curve and Time taken for execution
color = 'tab:green'
ax1.set_xlabel('Model', fontsize=14)
ax1.set_ylabel('Lucro', fontsize=14, color=color)
ax2 = sns.barplot(x='Model', y='Lucro', data = data, palette='GnBu_r')
ax1.tick_params(axis='y')
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('ROC_AUC', fontsize=14, color=color)
ax2 = sns.lineplot(x='Model', y='ROC_AUC', data = data, sort=False, color=color)
ax2.tick_params(axis='y', color=color)
```



Pensando em acurácia, o melhor foi a rede neural (Neural Network). Porém, pensando em maximixar o lucro, Random Forest, XGBoost, AdaBoost e LightGBM tiveram bons resultados.

Hyperparameters Tunning

Vamos otimizar os parâmetros de alguns modelos para ver se a acurácia e o lucro se modificam. Começando pelo Random Forest.

In [21]:
```
def find_best_params(model,params,cv=10,n_jobs=-1,X_train=X_over):
  random_cv = model_selection.RandomizedSearchCV(model,param_distributions=params,s
  random_cv.fit(X_train,y_over)
  print("The best auc score was %.3f"%(random_cv.best_score_))
  print("The best params were: %s"%(random_cv.best_params_))
  return random_cv.best_estimator_
```

In [22]:
```
model = ensemble.RandomForestClassifier(class_weight='balanced')
params = { 'n_estimators':[1000,2000],
           'max_depth':[1000,2000],
          'min_samples_split':[100,500],
           'min_samples_leaf':[3,5],
           'max_leaf_nodes':[100,250]
           }
```

In [23]:
```
find_best_params(model,params,cv=3)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
The best auc score was 0.911
The best params were: {'n_estimators': 2000, 'min_samples_split': 100, 'min_sample
s_leaf': 5, 'max_leaf_nodes': 250, 'max_depth': 1000}
```

Out[23]:

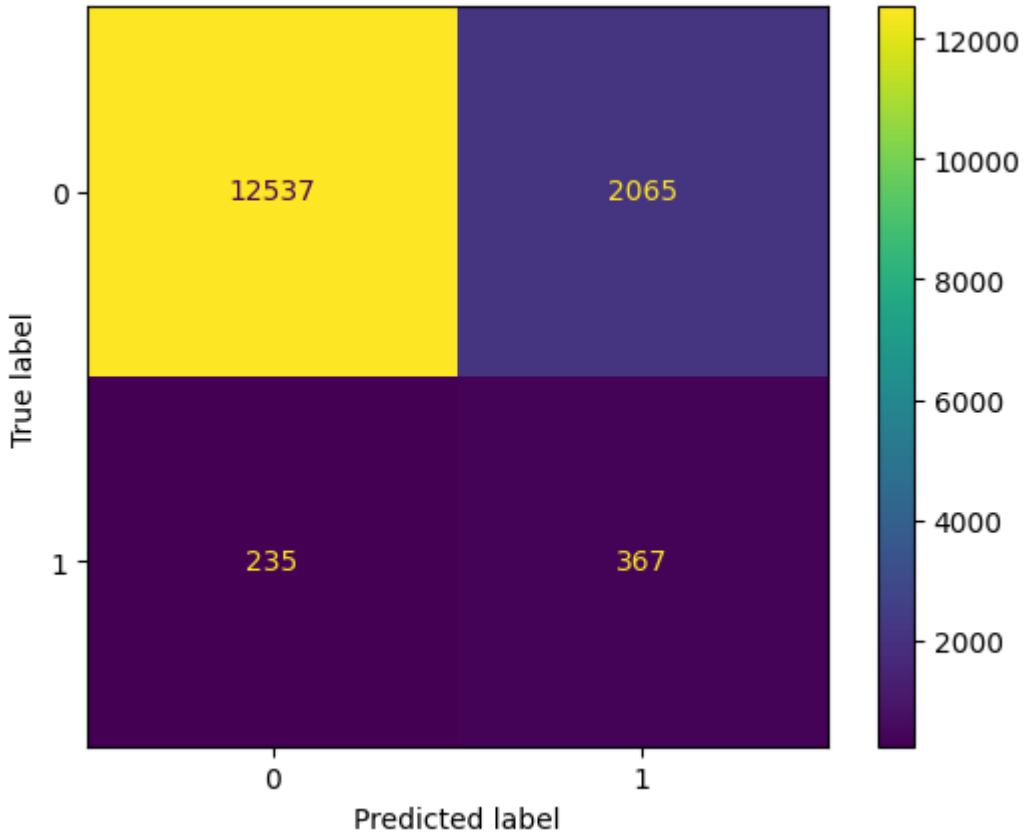
```
RandomForestClassifier(class_weight='balanced', max_depth=1000,
                       max_leaf_nodes=250, min_samples_leaf=5,
                       min_samples_split=100, n_estimators=2000)
```

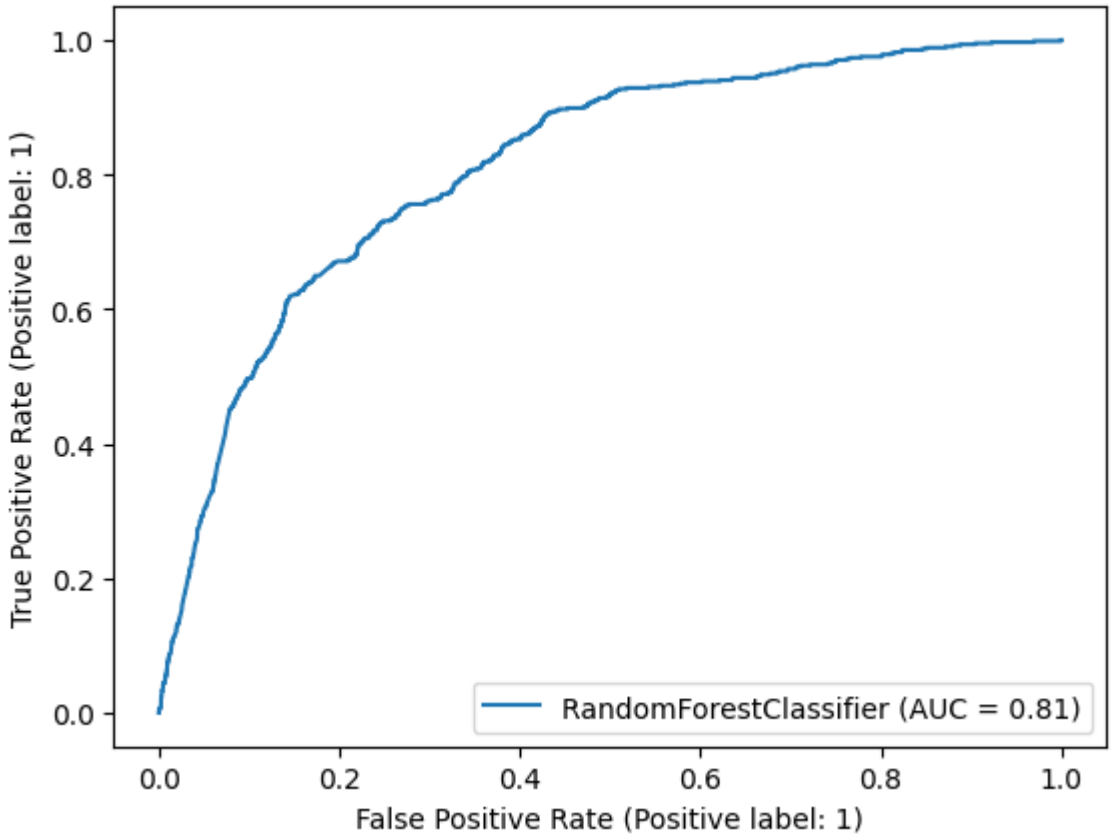
In [24]:

```
params_rf = {'class_weight':'balanced',
             'max_leaf_nodes':250,
             'max_depth': 1000,
             'min_samples_leaf': 5,
             'min_samples_split': 100,
             'n_estimators': 2000,
             'random_state': 12345}

model_rf = RandomForestClassifier(**params_rf)
model_rf, accuracy_rf, roc_auc_rf, tt_rf, lucro_rf = run_model(model_rf, X_over, y_
```

```
Acurácia = 0.8487240199947382
ROC Característica de Operação do Receptor = 0.7341077838970768
Tempo utilizado = 185.46787428855896
              precision    recall  f1-score   support

           0    0.98160   0.85858   0.91598     14602
           1    0.15090   0.60963   0.24192       602

    accuracy                        0.84872     15204
   macro avg    0.56625   0.73411   0.57895     15204
weighted avg    0.94871   0.84872   0.88929     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecate
d; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confus
ionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Funct
ion :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` o
r :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
Lucro obtido = 12380
```





Interessante! A acurácia foi para 84%, AUC em 81% e o lucro para 12830.

In [29]:
```
importance = model_rf.feature_importances_
```

In [30]:
```
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.00206
Feature: 1, Score: 0.13463
Feature: 2, Score: 0.00230
Feature: 3, Score: 0.00343
Feature: 4, Score: 0.00251
Feature: 5, Score: 0.00329
Feature: 6, Score: 0.00503
Feature: 7, Score: 0.00769
Feature: 8, Score: 0.00366
Feature: 9, Score: 0.00474
Feature: 10, Score: 0.00709
Feature: 11, Score: 0.00363
Feature: 12, Score: 0.00026
Feature: 13, Score: 0.00278
Feature: 14, Score: 0.02653
Feature: 15, Score: 0.00069
Feature: 16, Score: 0.00051
Feature: 17, Score: 0.00589
Feature: 18, Score: 0.00146
Feature: 19, Score: 0.00300
Feature: 20, Score: 0.00081
Feature: 21, Score: 0.00067
Feature: 22, Score: 0.00021
Feature: 23, Score: 0.00310
Feature: 24, Score: 0.00019
Feature: 25, Score: 0.00073
Feature: 26, Score: 0.00027
Feature: 27, Score: 0.00019
Feature: 28, Score: 0.00009
Feature: 29, Score: 0.00017
Feature: 30, Score: 0.00013
Feature: 31, Score: 0.00083
Feature: 32, Score: 0.03168
Feature: 33, Score: 0.00296
Feature: 34, Score: 0.00276
Feature: 35, Score: 0.00092
Feature: 36, Score: 0.00026
Feature: 37, Score: 0.00090
Feature: 38, Score: 0.00032
Feature: 39, Score: 0.02394
Feature: 40, Score: 0.00308
Feature: 41, Score: 0.03083
Feature: 42, Score: 0.00159
Feature: 43, Score: 0.00074
Feature: 44, Score: 0.00563
Feature: 45, Score: 0.00203
Feature: 46, Score: 0.00299
Feature: 47, Score: 0.00082
Feature: 48, Score: 0.00083
Feature: 49, Score: 0.00020
Feature: 50, Score: 0.00268
Feature: 51, Score: 0.00045
Feature: 52, Score: 0.00108
Feature: 53, Score: 0.00051
Feature: 54, Score: 0.00032
Feature: 55, Score: 0.00033
```

```
Feature: 56, Score: 0.00193
Feature: 57, Score: 0.00016
Feature: 58, Score: 0.00213
Feature: 59, Score: 0.00284
Feature: 60, Score: 0.00197
Feature: 61, Score: 0.00016
Feature: 62, Score: 0.00228
Feature: 63, Score: 0.00279
Feature: 64, Score: 0.00959
Feature: 65, Score: 0.03091
Feature: 66, Score: 0.02229
Feature: 67, Score: 0.00571
Feature: 68, Score: 0.00203
Feature: 69, Score: 0.00033
Feature: 70, Score: 0.00204
Feature: 71, Score: 0.00606
Feature: 72, Score: 0.02206
Feature: 73, Score: 0.02593
Feature: 74, Score: 0.00319
Feature: 75, Score: 0.00113
Feature: 76, Score: 0.00067
Feature: 77, Score: 0.00283
Feature: 78, Score: 0.00056
Feature: 79, Score: 0.00049
Feature: 80, Score: 0.00050
Feature: 81, Score: 0.04267
Feature: 82, Score: 0.01230
Feature: 83, Score: 0.02812
Feature: 84, Score: 0.01248
Feature: 85, Score: 0.00029
Feature: 86, Score: 0.00251
Feature: 87, Score: 0.00586
Feature: 88, Score: 0.00216
Feature: 89, Score: 0.00223
Feature: 90, Score: 0.00247
Feature: 91, Score: 0.00253
Feature: 92, Score: 0.00226
Feature: 93, Score: 0.00597
Feature: 94, Score: 0.00034
Feature: 95, Score: 0.00050
Feature: 96, Score: 0.00243
Feature: 97, Score: 0.01939
Feature: 98, Score: 0.01439
Feature: 99, Score: 0.01765
Feature: 100, Score: 0.02603
Feature: 101, Score: 0.01485
Feature: 102, Score: 0.00693
Feature: 103, Score: 0.05827
Feature: 104, Score: 0.00165
Feature: 105, Score: 0.00084
Feature: 106, Score: 0.00118
Feature: 107, Score: 0.00531
Feature: 108, Score: 0.00192
Feature: 109, Score: 0.00246
Feature: 110, Score: 0.00184
Feature: 111, Score: 0.00245
```

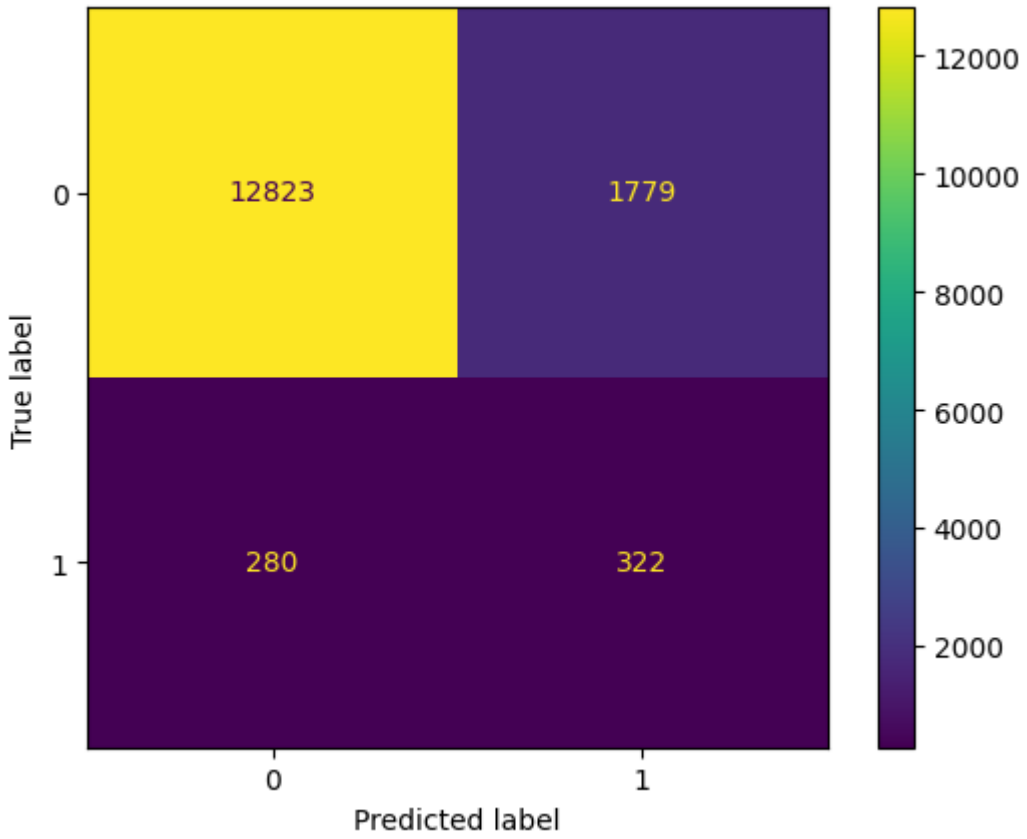
```
Feature: 112, Score: 0.00238
Feature: 113, Score: 0.00454
Feature: 114, Score: 0.00256
Feature: 115, Score: 0.00401
Feature: 116, Score: 0.00182
Feature: 117, Score: 0.00522
Feature: 118, Score: 0.00037
Feature: 119, Score: 0.00674
Feature: 120, Score: 0.01212
Feature: 121, Score: 0.00720
Feature: 122, Score: 0.00899
Feature: 123, Score: 0.02173
Feature: 124, Score: 0.03235
Feature: 125, Score: 0.02277
Feature: 126, Score: 0.02538
Feature: 127, Score: 0.00162
Feature: 128, Score: 0.00244
Feature: 129, Score: 0.00301
Feature: 130, Score: 0.00094
Feature: 131, Score: 0.00025
Feature: 132, Score: 0.00155
Feature: 133, Score: 0.00138
Feature: 134, Score: 0.00066
Feature: 135, Score: 0.00009
Feature: 136, Score: 0.00075
Feature: 137, Score: 0.00067
Feature: 138, Score: 0.03015
Feature: 139, Score: 0.00106
```

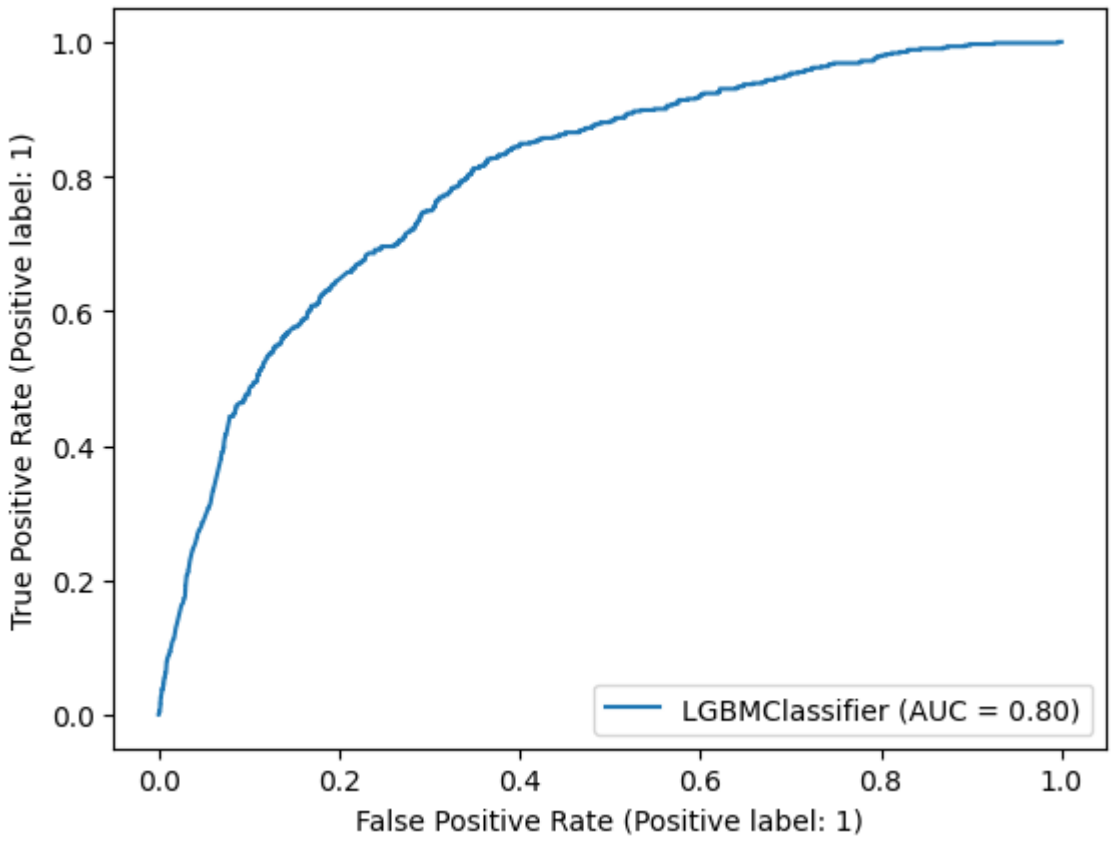
In [25]:
```
model_lgb_opt = lgb.LGBMClassifier(subsample= 0.7,
                    random_state=501,
                    objective='binary',
                    num_leaves=200,
                    min_split_gain= 0.01,
                    min_data_in_leaf= 10,
                    metric='auc',
                    max_depth=8,
                    learning_rate=0.05,
                    colsample_bytree= 0.5,
                    boosting_type= 'gbdt'
                    )
```

In [26]:
```
model_lgb_opt, accuracy_lgb_opt, roc_auc_lgb_opt, tt_lgb_opt, lucro_lgb_opt = run_m
```

```
[LightGBM] [Warning] min_data_in_leaf is set=10, min_child_samples=20 will be igno
red. Current value: min_data_in_leaf=10
Acurácia = 0.8645751118126809
ROC Característica de Operação do Receptor = 0.706525547631258
Tempo utilizado = 1.6850323677062988
              precision    recall  f1-score   support

           0    0.97863   0.87817   0.92568     14602
           1    0.15326   0.53488   0.23825       602

    accuracy                        0.86458     15204
   macro avg    0.56595   0.70653   0.58197     15204
weighted avg    0.94595   0.86458   0.89846     15204

Lucro obtido = 11190
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecate
d; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confus
ionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Funct
ion :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` o
r :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```





In [27]:
```
importance = model_lgb_opt.feature_importances_
```

In [28]:
```
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
```

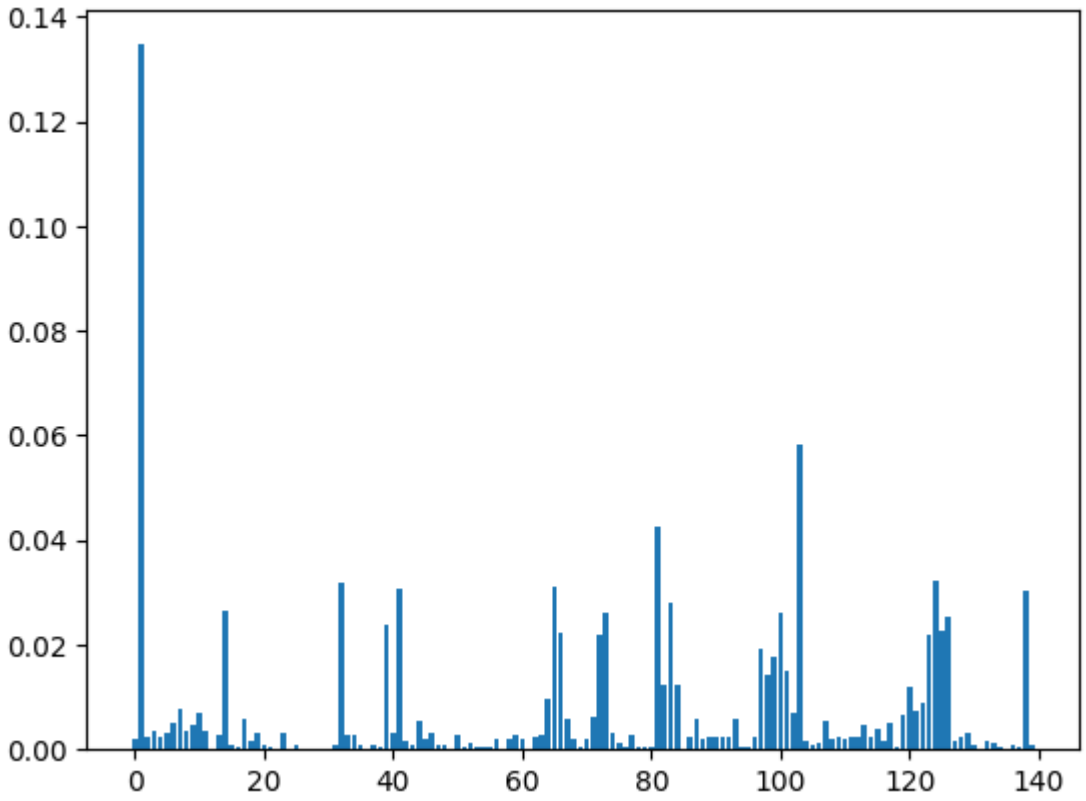
```
Feature: 0, Score: 102.00000
Feature: 1, Score: 183.00000
Feature: 2, Score: 115.00000
Feature: 3, Score: 174.00000
Feature: 4, Score: 65.00000
Feature: 5, Score: 77.00000
Feature: 6, Score: 163.00000
Feature: 7, Score: 214.00000
Feature: 8, Score: 151.00000
Feature: 9, Score: 89.00000
Feature: 10, Score: 65.00000
Feature: 11, Score: 126.00000
Feature: 12, Score: 5.00000
Feature: 13, Score: 76.00000
Feature: 14, Score: 121.00000
Feature: 15, Score: 20.00000
Feature: 16, Score: 7.00000
Feature: 17, Score: 29.00000
Feature: 18, Score: 0.00000
Feature: 19, Score: 7.00000
Feature: 20, Score: 2.00000
Feature: 21, Score: 1.00000
Feature: 22, Score: 2.00000
Feature: 23, Score: 5.00000
Feature: 24, Score: 2.00000
Feature: 25, Score: 1.00000
Feature: 26, Score: 2.00000
Feature: 27, Score: 4.00000
Feature: 28, Score: 1.00000
Feature: 29, Score: 7.00000
Feature: 30, Score: 1.00000
Feature: 31, Score: 7.00000
Feature: 32, Score: 39.00000
Feature: 33, Score: 60.00000
Feature: 34, Score: 63.00000
Feature: 35, Score: 72.00000
Feature: 36, Score: 5.00000
Feature: 37, Score: 53.00000
Feature: 38, Score: 5.00000
Feature: 39, Score: 155.00000
Feature: 40, Score: 65.00000
Feature: 41, Score: 103.00000
Feature: 42, Score: 24.00000
Feature: 43, Score: 8.00000
Feature: 44, Score: 40.00000
Feature: 45, Score: 3.00000
Feature: 46, Score: 3.00000
Feature: 47, Score: 6.00000
Feature: 48, Score: 1.00000
Feature: 49, Score: 0.00000
Feature: 50, Score: 8.00000
Feature: 51, Score: 11.00000
Feature: 52, Score: 4.00000
Feature: 53, Score: 0.00000
Feature: 54, Score: 11.00000
Feature: 55, Score: 19.00000
```

```
Feature: 56, Score: 116.00000
Feature: 57, Score: 33.00000
Feature: 58, Score: 49.00000
Feature: 59, Score: 115.00000
Feature: 60, Score: 51.00000
Feature: 61, Score: 20.00000
Feature: 62, Score: 60.00000
Feature: 63, Score: 52.00000
Feature: 64, Score: 117.00000
Feature: 65, Score: 62.00000
Feature: 66, Score: 232.00000
Feature: 67, Score: 99.00000
Feature: 68, Score: 127.00000
Feature: 69, Score: 1.00000
Feature: 70, Score: 104.00000
Feature: 71, Score: 51.00000
Feature: 72, Score: 81.00000
Feature: 73, Score: 508.00000
Feature: 74, Score: 70.00000
Feature: 75, Score: 37.00000
Feature: 76, Score: 33.00000
Feature: 77, Score: 28.00000
Feature: 78, Score: 38.00000
Feature: 79, Score: 28.00000
Feature: 80, Score: 28.00000
Feature: 81, Score: 532.00000
Feature: 82, Score: 289.00000
Feature: 83, Score: 353.00000
Feature: 84, Score: 321.00000
Feature: 85, Score: 4.00000
Feature: 86, Score: 147.00000
Feature: 87, Score: 221.00000
Feature: 88, Score: 20.00000
Feature: 89, Score: 7.00000
Feature: 90, Score: 20.00000
Feature: 91, Score: 17.00000
Feature: 92, Score: 78.00000
Feature: 93, Score: 76.00000
Feature: 94, Score: 22.00000
Feature: 95, Score: 8.00000
Feature: 96, Score: 174.00000
Feature: 97, Score: 493.00000
Feature: 98, Score: 530.00000
Feature: 99, Score: 388.00000
Feature: 100, Score: 571.00000
Feature: 101, Score: 303.00000
Feature: 102, Score: 260.00000
Feature: 103, Score: 380.00000
Feature: 104, Score: 47.00000
Feature: 105, Score: 16.00000
Feature: 106, Score: 8.00000
Feature: 107, Score: 269.00000
Feature: 108, Score: 63.00000
Feature: 109, Score: 55.00000
Feature: 110, Score: 42.00000
Feature: 111, Score: 36.00000
```

```
Feature: 112, Score: 76.00000
Feature: 113, Score: 94.00000
Feature: 114, Score: 40.00000
Feature: 115, Score: 44.00000
Feature: 116, Score: 84.00000
Feature: 117, Score: 113.00000
Feature: 118, Score: 20.00000
Feature: 119, Score: 454.00000
Feature: 120, Score: 509.00000
Feature: 121, Score: 365.00000
Feature: 122, Score: 473.00000
Feature: 123, Score: 521.00000
Feature: 124, Score: 880.00000
Feature: 125, Score: 467.00000
Feature: 126, Score: 811.00000
Feature: 127, Score: 50.00000
Feature: 128, Score: 62.00000
Feature: 129, Score: 35.00000
Feature: 130, Score: 51.00000
Feature: 131, Score: 40.00000
Feature: 132, Score: 38.00000
Feature: 133, Score: 45.00000
Feature: 134, Score: 55.00000
Feature: 135, Score: 10.00000
Feature: 136, Score: 14.00000
Feature: 137, Score: 22.00000
Feature: 138, Score: 1197.00000
Feature: 139, Score: 46.00000
```

In [31]:

```
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```



In [32]:
```
ft_imp = pd.DataFrame(zip(X_train.columns, model_lgb_opt.feature_importances_),
                      columns=['feature','importance']).\
                      set_index('feature')
ft_imp
```

Out[32]:

feature	importance
var3	102
var15	183
imp_op_var39_comer_ult1	115
imp_op_var39_comer_ult3	174
imp_op_var41_comer_ult1	65
...	...
saldo_medio_var13_corto_hace3	10
saldo_medio_var13_corto_ult1	14
saldo_medio_var13_corto_ult3	22
var38	1197
var15_below_23	46

140 rows × 1 columns

Vamos considerar as 20 features mais importantes no modelo Random Forest. A agora

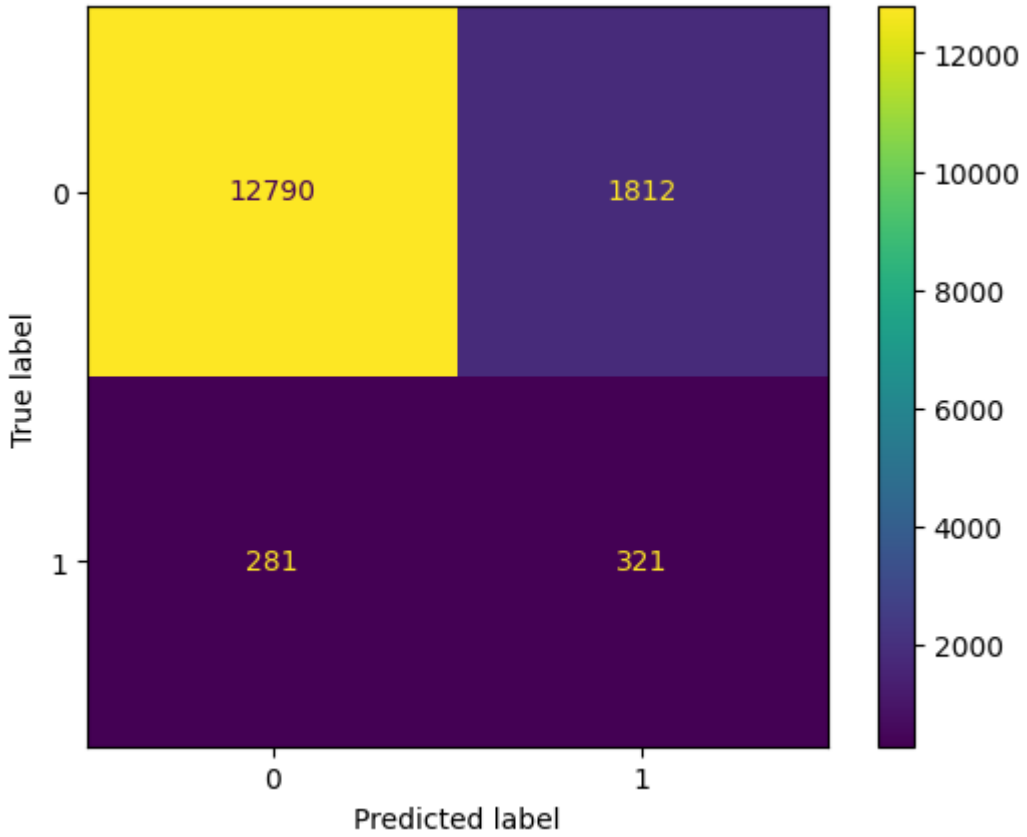
testamos o LightGBM e RF para ver o resultado

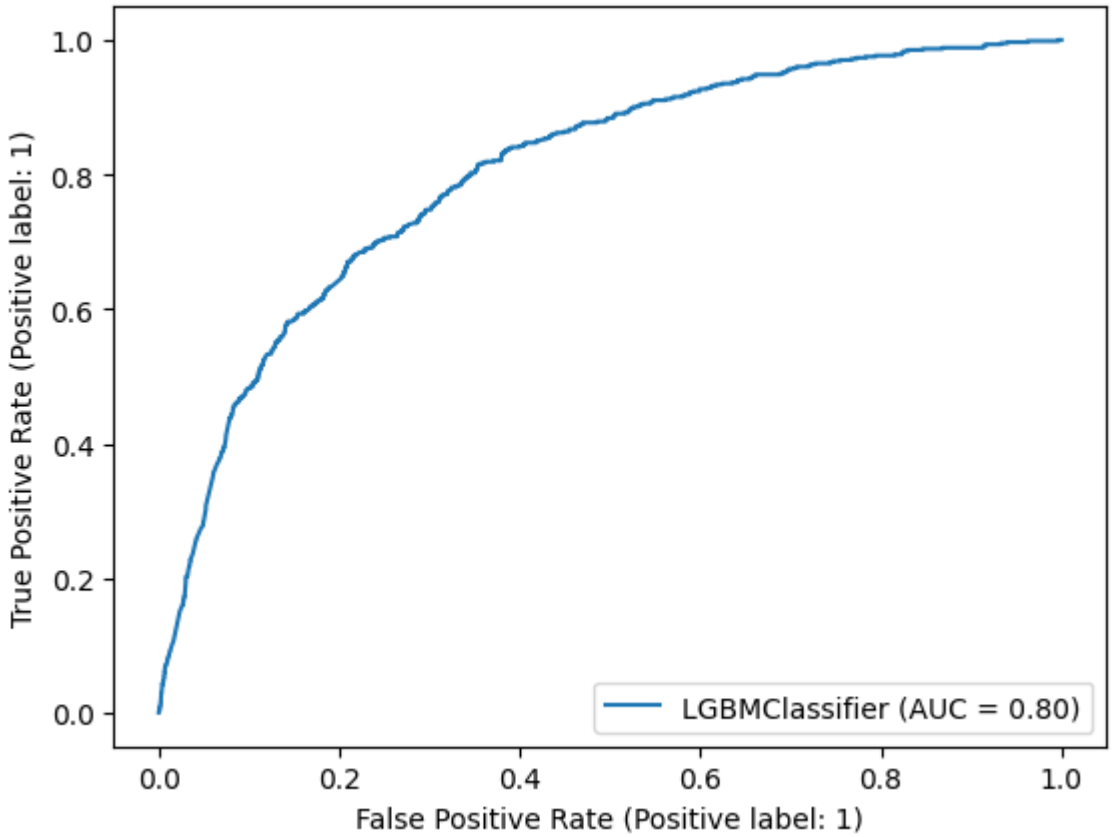
In [33]:
```
rand_imp = 20

better_than_random = ft_imp.sort_values(by='importance', ascending=False).query("im
```

In [34]:
```
X_train_better = X_over[better_than_random]
X_test_better = X_test[better_than_random]
```

In [35]:
```
model_lgb_opt, accuracy_lgb_opt, roc_auc_lgb_opt, tt_lgb_opt, lucro_lgb_opt = run_m
```

```
[LightGBM] [Warning] min_data_in_leaf is set=10, min_child_samples=20 will be igno
red. Current value: min_data_in_leaf=10
Acurácia = 0.8623388581952118
ROC Característica de Operação do Receptor = 0.7045650006529848
Tempo utilizado = 1.3964464664459229
              precision    recall  f1-score   support

           0    0.97850   0.87591   0.92437     14602
           1    0.15049   0.53322   0.23473       602

    accuracy                        0.86234     15204
   macro avg    0.56450   0.70457   0.57955     15204
weighted avg    0.94572   0.86234   0.89706     15204

Lucro obtido = 10770
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecate
d; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.
2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confus
ionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\ut
ils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Funct
ion :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use on
e of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` o
r :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



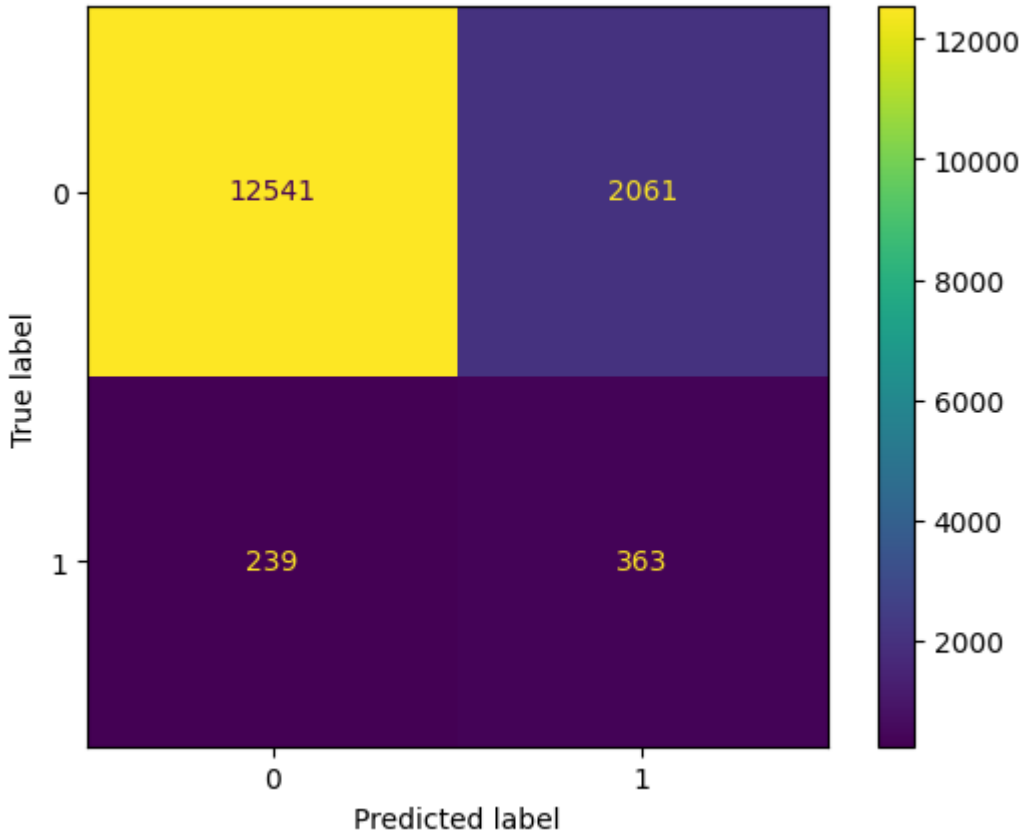


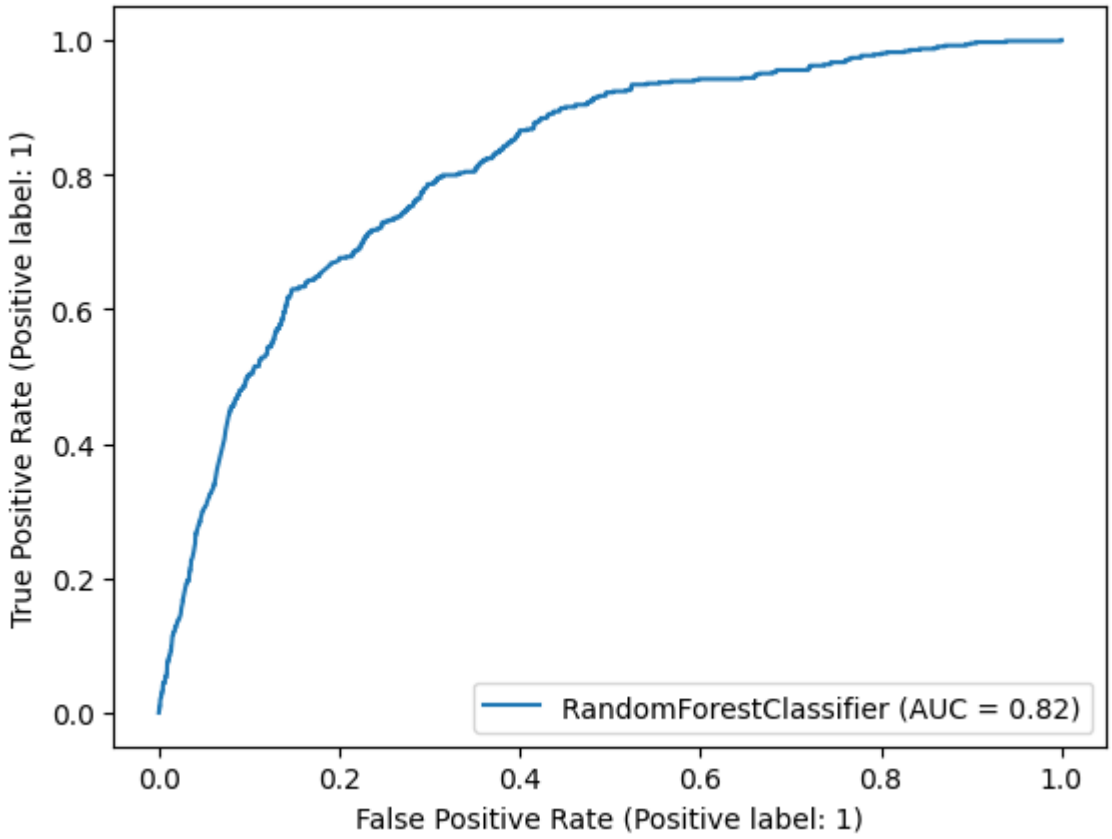
In [36]:
```
model_rf_fi, accuracy_rf_fi, roc_auc_rf_fi, tt_rf_fi, lucro_rf_fi = run_model(model
```

```
Acurácia = 0.8487240199947382
ROC Característica de Operação do Receptor = 0.7309224922995576
Tempo utilizado = 196.28951621055603
              precision    recall  f1-score   support

           0    0.98130   0.85885   0.91600     14602
           1    0.14975   0.60299   0.23992       602

    accuracy                        0.84872     15204
   macro avg    0.56553   0.73092   0.57796     15204
weighted avg    0.94837   0.84872   0.88923     15204
```

```
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
c:\users\gabri\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

```
Lucro obtido = 12060
```





O resultado manteve uma boa qualidade. A acurácia continua nos 84% e o AUC em 82%!

In [37]:
```
# Import dos módulos
from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Separando o array em componentes de input e output
X = X_over
Y = y_over
# X = X_train
# Y = y_train

# Aplicando a mesma escala nos dados
X = MinMaxScaler().fit_transform(X)

# Padronizando os dados (0 para a média, 1 para o desvio padrão)
X = StandardScaler().fit_transform(X)

# Definindo o tamanho dos dados de treino e de teste
teste_size = 0.3
seed = 10
```

In [38]:
```
# Criando o dataset de treino e de teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, Y, test_size = teste_siz
```

Vamos testar agora o Random Forest com valor padrão (100 estimators) mas, com os dados

na mesma escala e padronizados entre 0 e 1.

In [49]:
```
# Criação do modelo
modeloRF = RandomForestClassifier(n_estimators = 100)
modeloRF.fit(X_treino, y_treino)
```

Out[49]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [50]:
```
# Fazendo previsões
y_pred = modeloRF.predict(X_teste)
previsoes = [round(value) for value in y_pred]

# Avaliando as previsões
accuracy = accuracy_score(y_teste, previsoes)
print("Acurácia: %.2f%%" % (accuracy * 100.0))
```

```
Acurácia: 89.74%
```

Que interessante, tivemos uma acurácia acima da anterior (89%).

In [51]:
```
# Relatório de Classificação

# Import dos módulos
from sklearn.metrics import classification_report

# Fazendo as previsões e construindo o relatório
report = classification_report(y_teste, previsoes)

# Imprimindo o relatório
print(report)
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90     17462
           1       0.90      0.90      0.90     17584

    accuracy                           0.90     35046
   macro avg       0.90      0.90      0.90     35046
weighted avg       0.90      0.90      0.90     35046
```

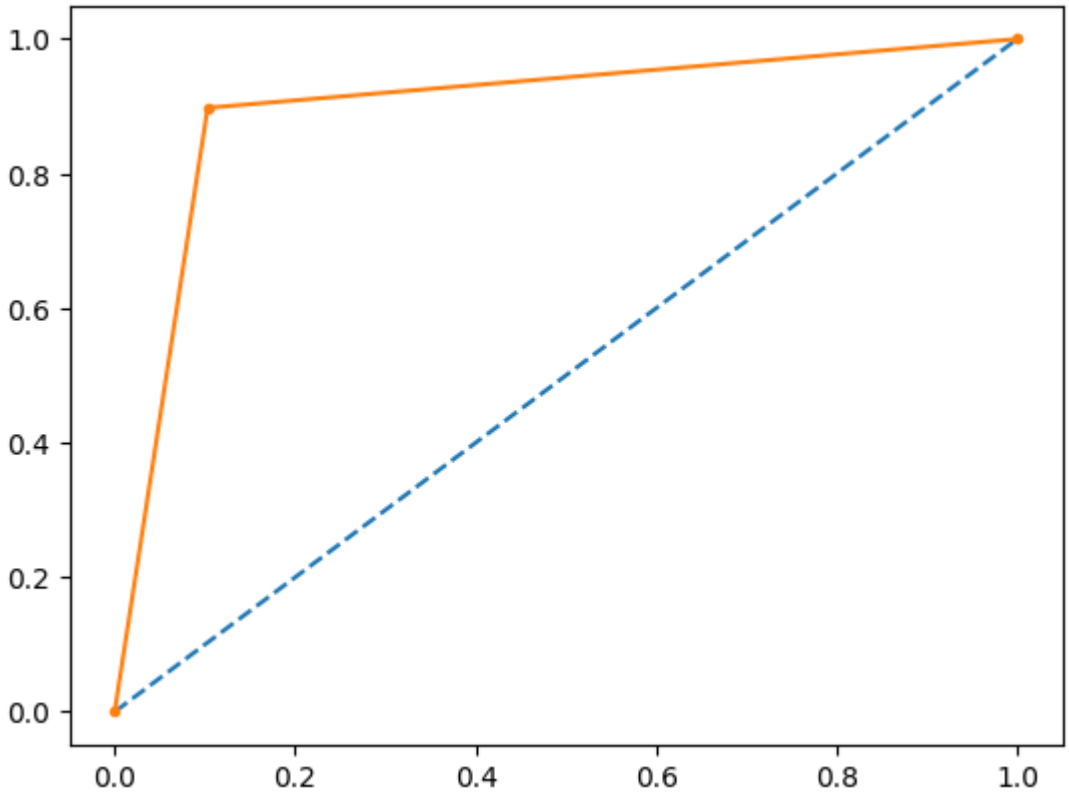
In [52]:
```
# Confusion Matrix
# Permite verificar a acurácia em um formato de tabela
matrix = confusion_matrix(y_teste, previsoes)
print("Confusion Matrix")
print(matrix)
```

```
Confusion Matrix
[[15663  1799]
 [ 1798 15786]]
```

In [53]:
```
# Visualizando 'ROC Curve and AUC'
```

```
# Import dos módulos
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# Calculando AUC
auc = roc_auc_score(y_teste, previsoes)
print('AUC: %.3f' % auc)

# Calculando ROC Curve
fpr, tpr, thresholds = roc_curve(y_teste, previsoes)

# Exibir a ROC curve do modelo
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.show()
```

```
AUC: 0.897
```



Show! Vamos seguir com o modelo Random Forest com parâmetros otimizados para a próxima etapa.

Próximos passos:

- Utilizar mais Feature Engineering (feature tools);
- Melhorar a busca no espaço de hiperparâmetros;
- Utilizar SelectKBest para feature selection;
- Testar TPOT;

- Explorar mais variáveis binárias como as que possuem a palavra-chave 'ind' e compará-las com a coluna 'TARGET'.

SANTANDER DATA MASTER - CIENTISTA DE DADOS

Questão b) Classificar clientes de 1 a 5

Descrição: A segunda tarefa consiste em dar uma nota de 1 a 5 para cada cliente da base teste, respeitando a variável 'TARGET', isto é, o seu nível de satisfação, sendo 1 o mais insatisfeito e 5 o mais satisfeito. Ao dar essa nota deve-se ter em mente que somente os clientes com nota 1 serão alvos de uma ação de retenção e que o objetivo dessa ação é maximizar o lucro esperado por cliente (usando os mesmos valores da primeira questão).

In [ ]:
```
!pip install scikit-learn
!pip install imblearn
!pip install xgboost
```

In [2]:
```
#Imports

#Manipulação dos Dados
import pandas as pd
import numpy as np

#Visualização dos Dados
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score as auc
from sklearn.metrics import (make_scorer, confusion_matrix,
                             silhouette_score, roc_curve,
                             precision_recall_curve, classification_report)
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
import xgboost as xgb
```

In [3]:
```
df_train = pd.read_pickle('./train_1.pickle')
df_test = pd.read_pickle('./test_1.pickle')
```

In [4]:
```
df_train.shape,df_test.shape
```

Out[4]:
```
((76020, 142), (75818, 141))
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(df_train.drop(['TARGET','ID'],
```

Para realizar a atividade de dar nota de 1 a 5, usaremos como base de ordenação o modelo otimizado de Random Forest. O método predict_proba trará a probabilidade até 1 da coluna

TARGET, ou seja, quanto maior o valor, maior a chance do cliente ser instatisfeito.

In [6]:
```
params_rf = {'class_weight':'balanced',
             'max_leaf_nodes':250,
             'max_depth': 1000,
             'min_samples_leaf': 3,
             'min_samples_split': 100,
             'n_estimators': 2000,
             'random_state': 12345}

rf_optimized = RandomForestClassifier(**params_rf)
rf_optimized.fit(X_train, y_train)
```

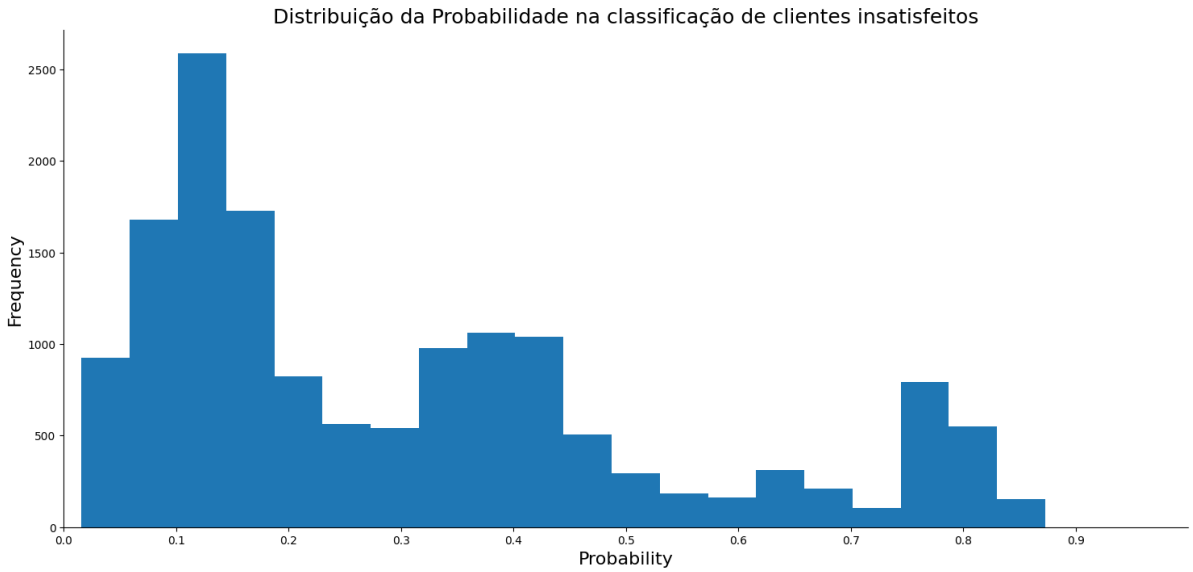
Out[6]:
```
RandomForestClassifier(class_weight='balanced', max_depth=1000,
                       max_leaf_nodes=250, min_samples_leaf=3,
                       min_samples_split=100, n_estimators=2000,
                       random_state=12345)
```

In [7]:
```
y_predicted = rf_optimized.predict_proba(X_test)[:,1]
auc(y_test, y_predicted)
```

Out[7]:
```
0.8290460825236246
```

Mantendo o modelo com aproximadamente 83% de acurácia e ajustando a probabilidade do output ser de 0 a 1, vamos verificar a distribuição da classicação dos clientes insatisfeitos.

In [9]:
```
fig, ax = plt.subplots(figsize = (18, 8))
ax.hist(rf_optimized.predict_proba(X_test)[:,1], bins = 20);
ax.set_xlim(0, 1);
plt.xticks(np.arange(0, 1, 0.1))
plt.title('Distribuição da Probabilidade na classificação de clientes insatisfeitos
plt.ylabel('Frequency', fontsize=16);
plt.xlabel('Probability', fontsize=16);
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
```



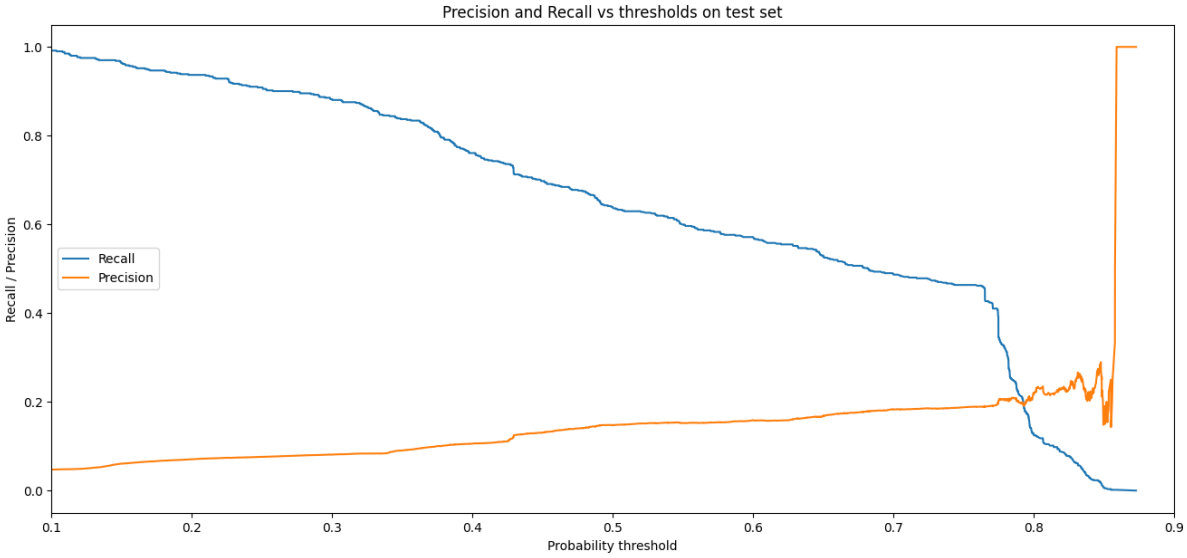
Conforme o gráfico acima, a probabilidades tem maior ocorrência até 0,8. Vamos testar de uma outra maneira. Agora utilizando o predict_proba para as bases de treino e teste.

In [10]:
```
y_pred_train = rf_optimized.predict(X_train)
y_pred_proba_train = rf_optimized.predict_proba(X_train)[:,1]
y_pred = rf_optimized.predict(X_test)
y_pred_proba = rf_optimized.predict_proba(X_test)[:,1]
```

A ideia é identificar a relação de precisão e sensibilidade para analisar o limiar (threshold) de probabilidade.

In [11]:
```
pr, recall, thr = precision_recall_curve(y_test, y_pred_proba)
pr_train, recall_train, thr_train = precision_recall_curve(y_train, y_pred_proba_tr
```

In [12]:
```
_ = plt.figure(figsize=(16,7))
_ = plt.plot(thr, recall[1:],label='Recall')
_ = plt.plot(thr, pr[1:],label='Precision')
_ = plt.xlabel('Probability threshold')
_ = plt.ylabel('Recall / Precision')
_ = plt.title('Precision and Recall vs thresholds on test set')
_ = plt.legend()
_ = plt.xlim([0.1,0.9])
plt.show()
```



A intersecção parece estar bem próxima do gráfico de distribuição anterior (0,8).

Como a ideia é dar uma nota de 1 a 5 mas pensando em maximizar o lucro, utilizo uma função criada pelo Caio Martins (https://github.com/CaioMar/certificao_data_masters/blob/master/Certificacao%20Data%20Mast que retorna o lucro conforme o y de teste e a previsão.

In [17]:
```
def max_profit_score(y_true, y_pred):
    """
    Função que avalia o modelo computando o lucro esperado
    da ação de retenção dos clientes insatisfeitos.
    """
    conf_matrix = confusion_matrix(y_true, y_pred)

    FP = conf_matrix[0][1] # Falsos positivos
    TP = conf_matrix[1][1] # Verdadeiros positivos

    lucro = 90*TP - 10*FP # lucro da acao de retencao (os
                          # verdadeiros e falsos negativos
                          # não contribuem em
                          # nada no lucro)
#     print(FP, TP)
    return lucro
```

Vamos testar o intervalo de 0.2 a 0.9 para avaliar o valor do lucro na base de treino e teste. Percebe-se um certo overfitting, mas seguimos com o estudo.

In [18]:
```
thresholds = np.linspace(0.2,0.9,250)
total_profit_percentage = [max_profit_score(y_train, (y_pred_proba_train > i)*1)/(y
total_profit_percentage_test = [max_profit_score(y_test, (y_pred_proba > i)*1)/(y_t
_ = plt.figure(figsize=(15,7))
_ = plt.plot(thresholds, total_profit_percentage, label='training')
_ = plt.plot(thresholds, total_profit_percentage_test, label='test')
_ = plt.ylabel('Max profit percentage')
_ = plt.xlabel('Probability threshold')
_ = plt.legend()
```

29433 2362
29208 2361
29000 2361
28791 2360
28583 2359
28408 2358
28215 2355
28032 2355
27871 2353
27678 2352
27481 2352
27290 2352
27112 2351
26962 2351
26816 2350
26693 2350
26515 2349
26358 2347
26194 2345
26053 2344
25913 2343
25754 2341
25640 2341
25510 2341
25375 2340
25259 2338
25132 2336
24997 2335
24887 2335
24778 2333
24656 2333
24537 2332
24433 2332
24304 2331
24115 2331
23966 2330
23833 2329
23727 2329
23579 2327
23456 2326
23313 2325
23194 2323
23066 2323
22925 2323
22799 2319
22648 2318
22507 2317
22363 2317
22203 2317
22055 2313
21612 2310
20970 2304
20334 2296
20074 2295
19817 2293
19517 2287

19252 2282
18957 2278
18550 2271
18192 2265
17852 2260
17575 2253
17227 2245
16918 2232
16546 2221
16231 2212
15982 2207
15742 2201
15563 2198
15406 2187
15250 2183
15107 2176
14973 2167
14841 2161
14712 2154
14571 2151
14423 2145
14268 2134
14113 2124
13921 2118
13773 2113
13053 2104
11589 2066
11484 2062
11392 2054
11232 2046
11141 2043
11040 2033
10908 2025
10792 2018
10517 2009
10435 2006
10267 2000
10172 1991
10086 1981
9994 1972
9877 1962
9811 1958
9762 1953
9705 1949
9580 1944
9374 1936
9289 1927
9051 1915
8789 1898
8734 1889
8693 1886
8642 1881
8583 1873
8543 1869
8499 1868
8454 1861

8418 1855
8373 1846
8328 1842
8296 1837
8264 1832
8235 1827
8193 1822
8154 1814
8109 1808
8075 1802
8047 1798
8007 1787
7972 1782
7934 1770
7911 1763
7882 1754
7838 1745
7804 1737
7767 1729
7723 1724
7686 1715
7645 1702
7579 1695
7532 1686
7486 1678
7433 1670
7393 1659
7364 1653
7323 1642
7287 1634
7236 1626
7191 1620
7148 1613
7111 1606
7081 1601
7054 1600
7025 1591
7000 1584
6971 1573
6938 1569
6861 1565
6723 1555
6639 1549
6565 1538
6509 1528
6472 1518
6386 1511
6283 1498
6113 1481
5957 1469
5860 1457
5809 1446
5773 1434
5748 1424
5666 1413
5602 1401

5575 1395
5554 1388
5524 1377
5452 1364
5364 1349
5327 1334
5301 1329
5270 1326
5229 1312
5178 1299
5121 1291
5092 1285
5074 1275
5034 1268
5002 1260
4976 1252
4960 1244
4940 1232
4916 1223
4898 1211
4868 1204
4846 1200
4829 1191
4800 1185
4781 1180
4760 1172
4741 1161
4720 1154
4694 1148
4686 1141
4672 1134
4667 1129
4652 1121
4518 1110
4182 1070
4123 1056
3947 1031
2971 871
2815 849
2403 765
2139 694
2074 684
1948 649
1703 599
1455 541
1133 445
1002 385
921 357
876 341
855 334
834 326
796 313
703 290
670 280
627 268
529 249

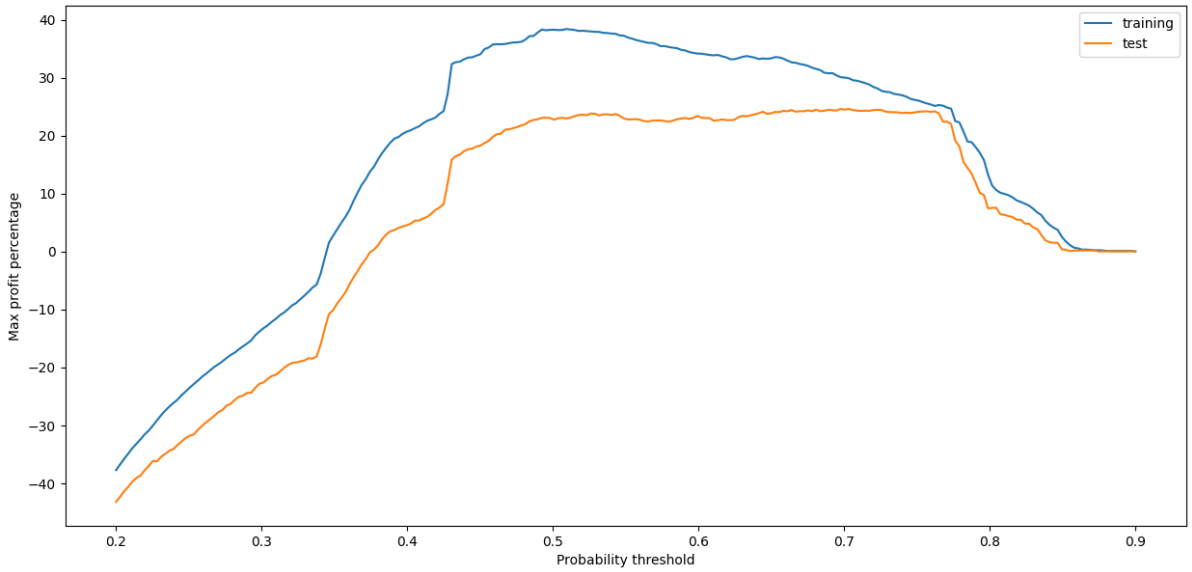
454 228
386 205
331 188
284 159
208 134
174 118
148 106
78 70
52 47
32 30
20 18
5 13
4 8
3 8
3 7
2 5
0 5
0 4
0 2
0 2
0 2
0 2
0 2
0 2
0 2
0 0
7419 564
7370 564
7318 564
7277 564
7219 563
7176 562
7127 559
7077 559
7037 559
6991 559
6940 553
6889 552
6853 551
6814 550
6790 549
6743 548
6710 548
6668 547
6629 545
6599 543
6555 543
6509 542
6476 542
6447 542
6415 542
6381 542
6352 541
6311 541
6277 539
6240 539

6210 539
6182 537
6156 537
6128 534
6073 533
6035 533
5996 530
5963 530
5927 529
5897 527
5868 527
5832 527
5797 526
5759 524
5727 521
5700 519
5664 516
5634 515
5593 510
5563 509
5449 509
5292 508
5141 506
5086 504
5024 504
4964 503
4903 502
4827 502
4744 500
4648 496
4558 493
4481 490
4391 487
4312 481
4236 477
4168 476
4099 473
4023 468
3985 465
3947 463
3906 460
3878 458
3835 455
3799 454
3770 451
3734 449
3709 448
3670 447
3626 446
3586 444
3545 443
3341 442
3002 429
2961 428
2928 426
2896 426

2869 425
2846 423
2817 422
2790 420
2730 416
2710 416
2664 415
2630 414
2607 412
2569 412
2539 409
2519 408
2499 407
2484 407
2445 404
2392 401
2370 400
2310 394
2243 388
2233 387
2226 386
2215 383
2192 382
2180 381
2168 379
2160 379
2146 379
2135 379
2123 378
2119 377
2103 377
2096 376
2086 373
2075 373
2066 372
2053 370
2045 370
2024 366
2014 362
2005 360
1996 359
1985 358
1973 356
1967 354
1962 353
1953 353
1944 352
1934 351
1921 349
1909 347
1901 347
1886 347
1871 346
1866 346
1856 344
1848 344

1829 344
1818 341
1811 340
1805 339
1801 336
1797 336
1790 336
1786 335
1779 334
1776 334
1751 334
1720 332
1698 329
1688 329
1668 328
1660 328
1637 327
1620 323
1579 319
1539 316
1524 314
1502 313
1488 311
1477 311
1456 307
1442 306
1435 305
1426 305
1415 303
1392 302
1371 298
1357 297
1349 297
1344 296
1338 295
1322 295
1311 293
1304 293
1297 291
1293 290
1287 289
1284 289
1277 288
1270 288
1267 288
1261 287
1258 285
1253 284
1244 283
1238 282
1234 281
1222 280
1217 279
1211 279
1205 279
1203 279

1200 279
1196 278
1192 278
1166 273
1090 256
1072 254
1033 247
775 201
755 193
659 166
570 150
550 142
522 129
461 112
391 102
299 78
259 74
239 72
229 64
224 63
219 61
209 59
181 53
171 52
163 47
137 44
116 38
96 34
88 27
76 20
57 16
45 14
36 13
24 5
12 3
6 1
1 1
0 1
0 1
0 1
0 1
0 1
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0



In [19]:
```
threshold_for_max_profit = thresholds[np.argmax(total_profit_percentage)]
print(threshold_for_max_profit)
```

```
0.5092369477911647
```

O limiar ideal que maximiza o lucro é de aproximadamente 0,51.

In [20]:
```
print(classification_report(y_test, (y_pred_proba > threshold_for_max_profit)*1) )
```

```
              precision    recall  f1-score   support

           0       0.98      0.85      0.91     14602
           1       0.15      0.63      0.24       602

    accuracy                           0.84     15204
   macro avg       0.57      0.74      0.58     15204
weighted avg       0.95      0.84      0.89     15204
```

Agora vamos dividir o valor predito conforme o limiar (0,51) para termos as 5 notas.

In [21]:
```
notas = pd.cut(y_pred_proba, bins=list(np.linspace(0,threshold_for_max_profit,5))+[
```

In [22]:
```
_ = pd.concat((pd.Series(notas, name='notas'), y_test.reset_index(drop=True)), axis
    groupby(['TARGET','notas']).size().unstack('TARGET').plot.bar(stacked=True,figs
_ = plt.xticks(rotation=0)
_ = plt.ylabel('Volume de clientes no conjunto teste')
_ = plt.xlabel('Nota')
_ = plt.title('Volume de clientes por nota com volume real de clientes insatisfeito
```

No gráfico, vemos uma concentração maior de clientes insatisfeitos (conforme solicitado), o que auxilia na ação de retenção.

In [23]:
```
print("Lucro total das notas 1 no conjunto de treinamento: R$ "+str(max_profit_scor
print("Representando "+str(round(max_profit_score(y_test, (notas == 1)*1)/(y_test.s
```

```
2168 379
Lucro total das notas 1 no conjunto de treinamento: R$ 12430
2168 379
Representando 22.9% do máximo lucro possível no conjunto de teste.
```

Então, chegamos em um valor de quase 23% do lucro total que podíamos adquirir. Vale ressaltar que um valor maior que o da questão anterior (11930).

Próximos passos de testes:

- Evoluir na precisão da nota 1 para termos uma concentração maior de clientes insatisfeitos no 1º quesito e conseguirmos realizar as ações de retenção.

In [ ]:

SANTANDER DATA MASTER - CIENTISTA DE DADOS

Questão c) Três grupos naturais com maior lucro

Descrição: Todo conjunto de dados é passível de ser dividido em grupos coesos, conhecidos como agrupamentos naturais. A terceira tarefa é encontrar os três grupos naturais que possuem os maiores lucros esperados por cliente (usando os mesmos valores da primeira questão).

In [1]:
```
#Imports

#Manipulação dos Dados
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import xgboost as xgb
from scipy.cluster.hierarchy import linkage, dendrogram

from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
```

In [2]:
```
df_train = pd.read_pickle('./train_1.pickle')
df_test = pd.read_pickle('./test_1.pickle')
```

In [3]:
```
df_train.shape,df_test.shape
```

Out[3]:
```
((76020, 142), (75818, 141))
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(df_train.drop(['TARGET','ID'],
```

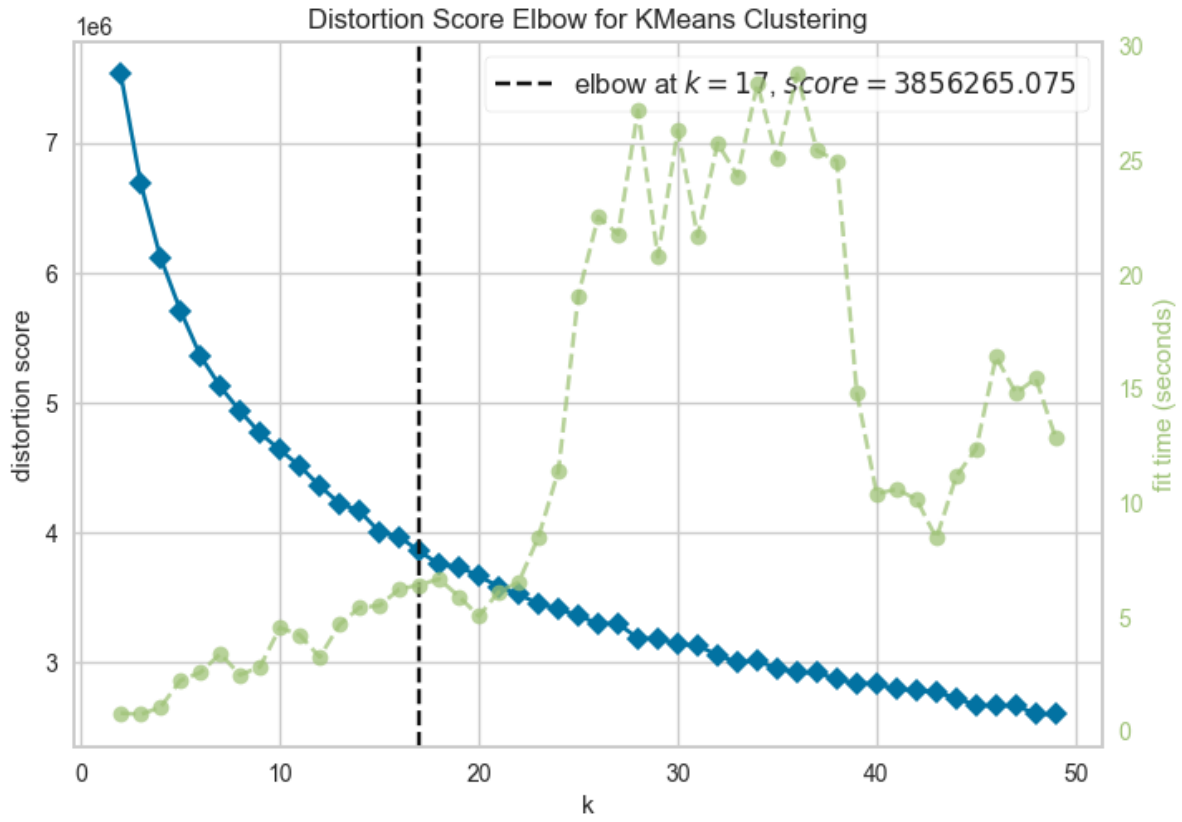
Para realizar essa questão, vamos utilizar o método KMeans. Para que o resultado seja mais assertivo, o StandardScaler será essencial, pois ele padroniza os recursos removendo a média e dimensionando para a variação da unidade. Como o algoritmo é baseado no cálculo da distância e, portanto, os dados devem ser transformados em uma escala padronizada.

In [5]:
```
X_train_scaler = StandardScaler().fit_transform(X_train)
```

Para usar o KMeans com maestria, identificar o K ideal é primordial para chegarmos em uma conclusão factível. K seria o número de cluster naturais da base.

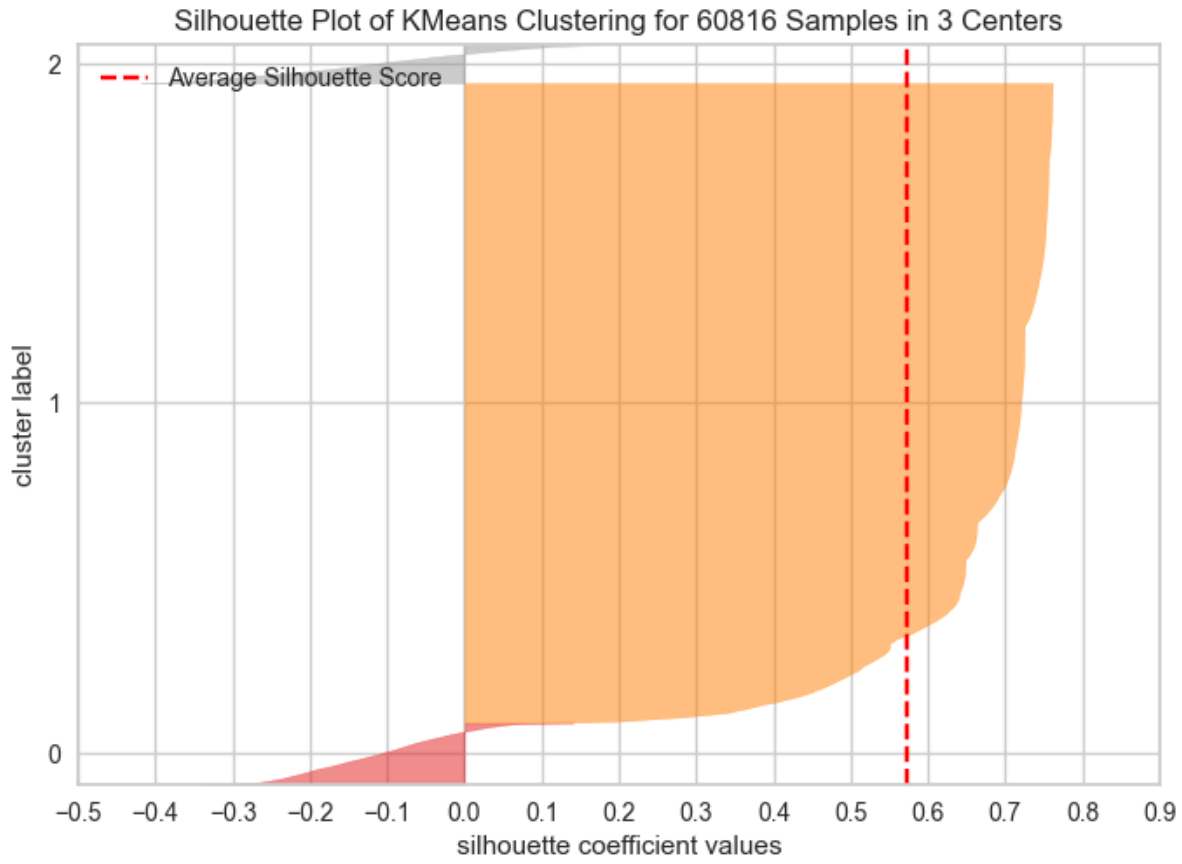
In [6]:
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,50))
```

In [7]:
```
visualizer.fit(X_train_scaler)        # Fit the data to the visualizer
visualizer.show();        # Finalize and render the figure
```



Identificamos que o K ideal é 17.

In [23]:
```
model_3clust = KMeans(n_clusters = 3, random_state=42)

sil_visualizer = SilhouetteVisualizer(model_3clust)
sil_visualizer.fit(X_train_scaler)
sil_visualizer.show()
```



Out[23]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 60816 Samples in 3 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

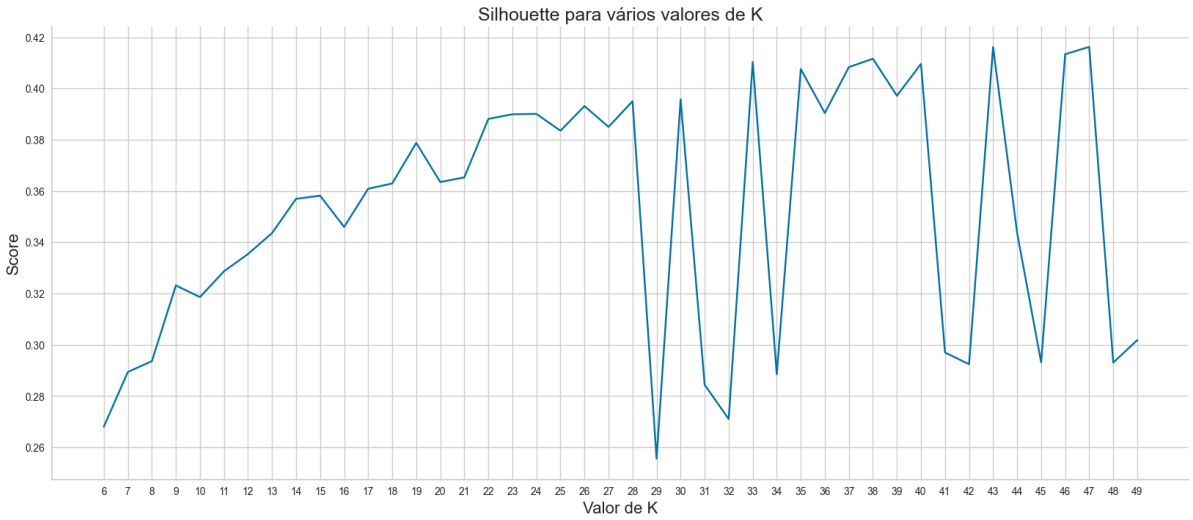
Uma vez definido que deve haver 17 clusters, os 3 grupos que geram maior lucro por cliente são aqueles que possuem a maior proporção de clientes insatisfeitos para que o programa de retenção seja aplicado a um maior número de clientes verdadeiramente insatisfeitos e gere lucro.

In [8]:
```
from sklearn.metrics import silhouette_score

coefficients = []

for i in range(6, 50):
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(X_train_scaler)
    score = silhouette_score(X_train_scaler, kmeans.labels_)
    coefficients.append(score)

    print("For K = {}, silhouette score is {})".format(i, score))
```

```
For K = 6, silhouette score is 0.2680334831961308)
For K = 7, silhouette score is 0.2894062910008122)
For K = 8, silhouette score is 0.29363177180375866)
For K = 9, silhouette score is 0.3231632549919612)
For K = 10, silhouette score is 0.31862384927411286)
For K = 11, silhouette score is 0.32867632167657596)
For K = 12, silhouette score is 0.33541456286524607)
For K = 13, silhouette score is 0.3435886044045092)
For K = 14, silhouette score is 0.3569429514701501)
For K = 15, silhouette score is 0.3581677886835501)
For K = 16, silhouette score is 0.3459838348249535)
For K = 17, silhouette score is 0.36087933271364847)
For K = 18, silhouette score is 0.3629449330606034)
For K = 19, silhouette score is 0.37876814709525425)
For K = 20, silhouette score is 0.36349312631555586)
For K = 21, silhouette score is 0.36531070942469707)
For K = 22, silhouette score is 0.38811937207578895)
For K = 23, silhouette score is 0.3899275030981974)
For K = 24, silhouette score is 0.3900753358511592)
For K = 25, silhouette score is 0.3835096887079895)
For K = 26, silhouette score is 0.39310810691918435)
For K = 27, silhouette score is 0.3850156566099473)
For K = 28, silhouette score is 0.39501236564851344)
For K = 29, silhouette score is 0.25554694270964273)
For K = 30, silhouette score is 0.3957589756368894)
For K = 31, silhouette score is 0.28442276137878963)
For K = 32, silhouette score is 0.2710587555941658)
For K = 33, silhouette score is 0.41031203326529164)
For K = 34, silhouette score is 0.2885189472014106)
For K = 35, silhouette score is 0.40755580051382617)
For K = 36, silhouette score is 0.3904145571766379)
For K = 37, silhouette score is 0.40831716829529124)
For K = 38, silhouette score is 0.4115935682823781)
For K = 39, silhouette score is 0.3971656914429141)
For K = 40, silhouette score is 0.4095697376084423)
For K = 41, silhouette score is 0.29700753086097176)
For K = 42, silhouette score is 0.29243698035209564)
For K = 43, silhouette score is 0.416177934225052)
For K = 44, silhouette score is 0.3437102090451282)
For K = 45, silhouette score is 0.2931429526517089)
For K = 46, silhouette score is 0.4133523789123446)
For K = 47, silhouette score is 0.41621496608034747)
For K = 48, silhouette score is 0.293071142610572)
For K = 49, silhouette score is 0.3018565556266377)
```

In [9]:

```
# Plotando a Silhouette
fig, ax = plt.subplots(figsize = (20, 8))
plt.plot(range(6, 50), coefficients)
plt.title('Silhouette para vários valores de K', fontsize = 18)
plt.xlabel('Valor de K', fontsize = 16)
plt.xticks(np.arange(6, 50, 1))
plt.ylabel('Score', fontsize = 16)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```



Melhor valor de K considerando a Silhueta foi 47.

Vamos testar os dois resultados, sendo K=17 (Elbow) e K=47 (Silhouette)

K=17

In [9]:
```
kmeans = KMeans(n_clusters = 17).fit(X_train_scaler)
kmeans.labels_
```

Out[9]:
```
array([ 0,  8,  1, ..., 16,  0,  1])
```

In [10]:
```
result_train = pd.DataFrame({'target': y_train, 'labels': kmeans.labels_})
result_train.head()
```

Out[10]:

	target	labels
56035	0	12
15449	0	1
68443	0	2
50258	0	12
4858	0	2

In [11]:
```
unsatisfied_dist = result_train[result_train['target'] == 1].labels.value_counts().
```

In [12]:
```
fig, ax = plt.subplots(figsize = (20, 8))
plt.bar(unsatisfied_dist.index + 1, unsatisfied_dist.values);
plt.title('Unsatisfied customer amount for each cluster', fontsize = 18);
plt.xlabel('Cluster', fontsize = 16);
plt.ylabel('Amount', fontsize = 16)
plt.xticks(range(1, 26, 1))

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
```

Agrupamento 1, 3 e 13.

In [10]:
```
scaled_features = StandardScaler().fit_transform(X_test)
result_test = pd.DataFrame({'target': y_test, 'labels': kmeans.predict(scaled_featu
result_test.head()
```

Out[10]:

	target	labels
63089	0	5
27056	0	1
5923	0	1
4865	0	1
6344	0	0

In [14]:
```
unsatisfied_dist_test = result_test[result_test['target'] == 1].labels.value_counts
```

In [15]:
```
fig, ax = plt.subplots(figsize = (20, 8))
plt.bar(unsatisfied_dist_test.index + 1, unsatisfied_dist_test.values);
plt.title('Unsatisfied customer amount for each cluster', fontsize = 18);
plt.xlabel('Cluster', fontsize = 16);
plt.ylabel('Amount', fontsize = 16)
plt.xticks(range(1, 26, 1))

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
```

Como esperado, os clusters 1, 3 e 13 possuem a maior parte de clientes insatisfeitos. É uma evidência clara de que nosso modelo funciona bem para novas instâncias!

K=47

In [ ]:
```
kmeans = KMeans(n_clusters = 47).fit(X_train_scaler)
kmeans.labels_
```

In [11]:
```
result_train = pd.DataFrame({'target': y_train, 'labels': kmeans.labels_})
result_train.head()
```

Out[11]:

	target	labels
56035	0	20
15449	0	40
68443	0	2
50258	0	27
4858	0	2

In [12]:
```
unsatisfied_dist = result_train[result_train['target'] == 1].labels.value_counts().
```

In [15]:
```
fig, ax = plt.subplots(figsize = (20, 8))
plt.bar(unsatisfied_dist.index + 1, unsatisfied_dist.values);
plt.title('Unsatisfied customer amount for each cluster', fontsize = 18);
plt.xlabel('Cluster', fontsize = 16);
plt.ylabel('Amount', fontsize = 16)
plt.xticks(range(1, 48, 1))

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
```

Os maiores grupos foram 1, 3 e 7.

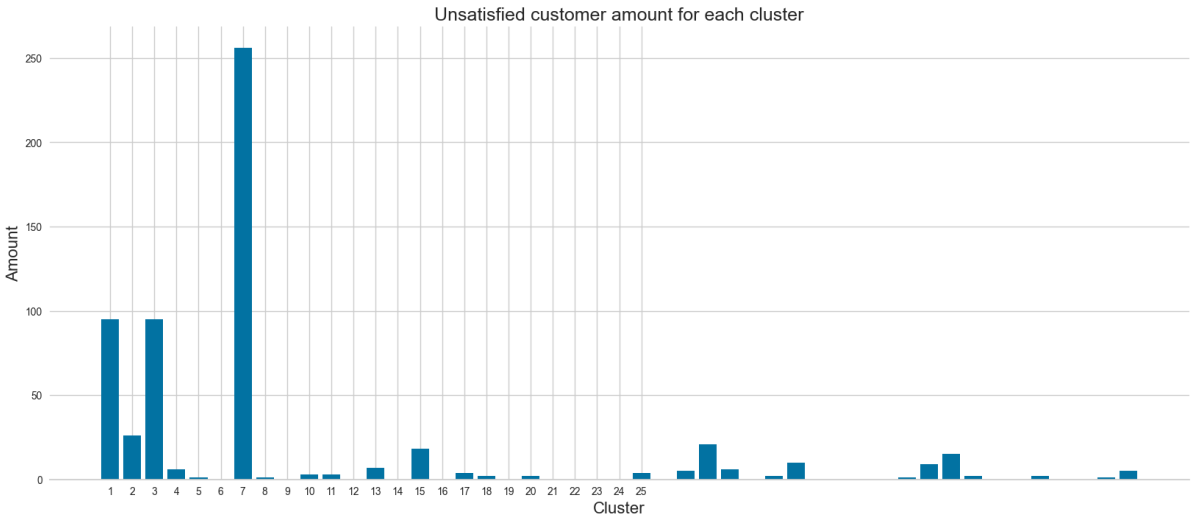
In [16]:
```
scaled_features = StandardScaler().fit_transform(X_test)
result_test = pd.DataFrame({'target': y_test, 'labels': kmeans.predict(scaled_featu
result_test.head()
```

Out[16]:

	target	labels
63089	0	6
27056	0	2
5923	0	2
4865	0	2
6344	0	27

In [17]:
```
unsatisfied_dist_test = result_test[result_test['target'] == 1].labels.value_counts
```

In [18]:
```
fig, ax = plt.subplots(figsize = (20, 8))
plt.bar(unsatisfied_dist_test.index + 1, unsatisfied_dist_test.values);
plt.title('Unsatisfied customer amount for each cluster', fontsize = 18);
plt.xlabel('Cluster', fontsize = 16);
plt.ylabel('Amount', fontsize = 16)
plt.xticks(range(1, 26, 1))

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
```



Como esperado, os clusters 1, 3 e 7 possuem a maior parte de clientes insatisfeitos.

Próximos passos:

- Testar Agglomerative Hierarchical Clustering;
- Testar o GMM;
- Calcular o lucro a partir desses grupos.

Referências

[1] Dwivedi. Rohit. How Does K-nearest Neighbor Works In Machine Learning Classification Problem?. https://www.analyticssteps.com/blogs/how-does-k-nearest-neighbor-works-machine-learning-classification-problem

[2] Dwivedi. Rohit. How Does Support Vector Machine (SVM) Algorithm Works In Machine Learning?. https://www.analyticssteps.com/blogs/how-does-support-vector-machine-algorithm-works-machine-learning

[3] Dutta. Bhumika. Top Classification Algorithms Using Python. https://www.analyticssteps.com/blogs/top-classification-algorithms-using-python

[4] Chauhan. Ankit. Random Forest Classifier and its Hyperparameters. https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6

[5] Myrianthous. Giorgos. What Is The Difference Between predict() and predict_proba() in scikit-learn?. https://towardsdatascience.com/predict-vs-predict-proba-scikit-learn-bdc45daa5972

[6] Malli. How to Drop Duplicate Columns in pandas DataFrame. https://sparkbyexamples.com/pandas/pandas-remove-duplicate-columns-from-dataframe/

[7] Shanmuganathan. Yogeshwaran. Airline-Passenger-Satisfaction. https://github.com/yogeshwaran-shanmuganathan/Airline-Passenger-Satisfaction/blob/master/Code/airline_passenger_satisfaction.ipynb

[8] Vickery. Rebecca. Optimising a Machine Learning Model with the Confusion Matrix. https://towardsdatascience.com/understanding-the-confusion-matrix-and-its-business-applications-c4e8aaf37f42

[9] Hashmi. Farukh. How to use Artificial Neural Networks for classification in python?. https://thinkingneuron.com/how-to-use-artificial-neural-networks-for-classification-in-python/

[10] Wu. Songhao. Hyper-Parameter Tuning in Python. https://towardsdatascience.com/hyper-parameter-tuning-in-python-1923797f124f

[11] Koehrsen. Will. Hyperparameter Tuning the Random Forest in Python. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

[12] Perez. Oscar. case_santander_data_master. https://github.com/obabilonia/case_santander_data_master/blob/main/Case_Data_Master_Santa

[13] Martins. Caio. certificao_data_masters. https://github.com/CaioMar/certificao_data_masters/blob/master/Certificacao%20Data%20Mast

[14] Couto. Pedro. Santander-Case. https://github.com/PedroHCouto/Santander-Case

Contatos

LinkedIn: https://www.linkedin.com/in/gabriel-barbosa-cardoso-98b479a7/

GitHub: https://github.com/Gcardoso1

Repositório do Case: https://github.com/Gcardoso1/Case-Santander-Data-Master

Email: gabrielbcardoso1@gmail.com