In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from IPython.display import display
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,roc_curve, roc_auc_sc
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import scikitplot as skplt
from IPython.display import display
from pycaret.classification import *
```

In [2]:

```
dtypes = {  'Regiao': 'object',
            'UF': 'object',
            'CNAE': 'object',
            'Atendida':'bool',
            'CodAssunto': 'object',
            'SexoConsumidor': 'object',
            'FaixaEtaria': 'object',
            'CEP': 'object',
            'InscritoDAU':'bool'}
```

In [3]:

```
df_ml1 = pd.read_csv(r'C:\Users\73594253368\Desktop\Curso\Datasets\Procon\dataset_trata
```

In [4]:

```
df_ml1 = df_ml1[['Regiao','UF','CNAE','Atendida','CodAssunto','SexoConsumidor','FaixaEt
```

In [5]:

```
# Este df_ml1 foi a primeira tentativa. Manteremos esse data frame para testes demonstr
# Para o ML "oficial", copiaremos esse df_ml1 para o df_ml
df_ml = df_ml1
```

In [6]:

```
1 df_ml
```

Out[6]:

	Regiao	UF	CNAE	Atendida	CodAssunto	SexoConsumidor	FaixaEtaria	CE
0	Norte	RO	6120501.0	False	187.0	M	5	76824042
1	Norte	RO	3514000.0	False	185.0	M	4	76824322
2	Norte	RO	8599604.0	True	236.0	M	3	78932000
3	Norte	RO	6120501.0	True	187.0	M	5	78932000
4	Norte	RO	6493000.0	False	57.0	M	6	76821331
...	...	...	...	...	...	...	...	
10514	Sudeste	SP	6110801.0	True	187.0	F	4	9617000
10515	Norte	RO	6143400.0	True	259.0	M	2	76940000
10516	Norte	RO	6422100.0	False	63.0	F	6	76990000
10517	Norte	RO	3514000.0	False	185.0	F	4	76807400
10518	Norte	RO	6423900.0	True	53.0	F	3	76806420

10519 rows × 9 columns

In [7]:

```
1 # Demonstração do desbalanceamento na variável "target"
2 df_ml['Atendida'].value_counts()
```

Out[7]:

```
True     6306
False    4213
Name: Atendida, dtype: int64
```

Aplicando SMOTE

Data Preparation

As variáveis preditoras mais importantes do nosso dataset são as categóricas. Assim, como etapa preparatória do SMOTE, temos que criar variáveis dummies. Testamos, antes, com SMOTE e sem dummies e, também, o tradicional dummies sem SMOTE. Igualmente testamos LabelEncoder + dummies + SMOTE. Os melhores resultados de acurária e recall foram com o procedimento a seguir.

In [8]:

```python
df_ml.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10519 entries, 0 to 10518
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Regiao          10519 non-null  object
 1   UF              10519 non-null  object
 2   CNAE            10519 non-null  object
 3   Atendida        10519 non-null  bool  
 4   CodAssunto      10519 non-null  object
 5   SexoConsumidor  10519 non-null  object
 6   FaixaEtaria     10519 non-null  object
 7   CEP             10519 non-null  object
 8   InscritoDAU     10519 non-null  bool  
dtypes: bool(2), object(7)
memory usage: 595.9+ KB
```

In [9]:

```python
df_ml = pd.get_dummies(df_ml[['Regiao',
                             'UF',
                             'CNAE',
                             'Atendida',
                             'CodAssunto',
                             'SexoConsumidor',
                             'FaixaEtaria',
                             'CEP','InscritoDAU']])
```

In [10]:

```python
df_ml.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10519 entries, 0 to 10518
Columns: 6928 entries, Atendida to CEP_9990244.0
dtypes: bool(2), uint8(6926)
memory usage: 69.5 MB
```

In [11]:

```python
df_ml.shape
```

Out[11]:

```
(10519, 6928)
```

SMOTE após dummies

In [12]:

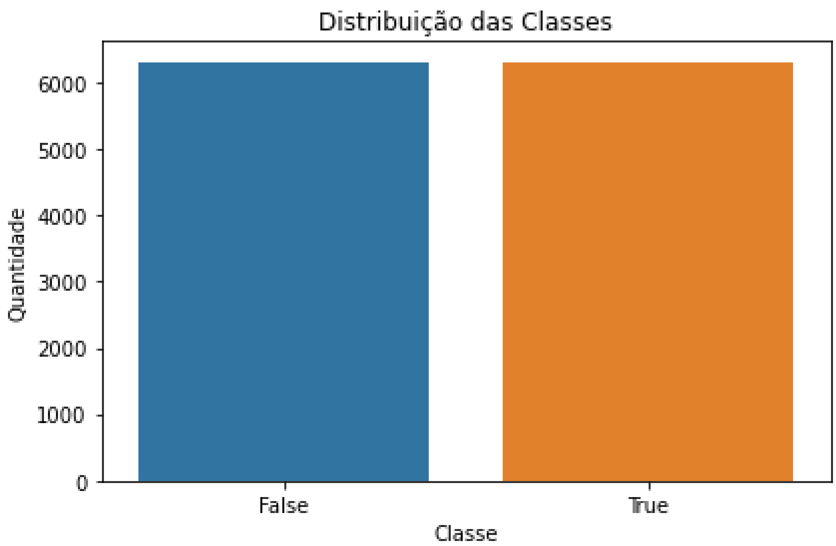
```
X = df_ml.drop(['Atendida'],axis=1)
y = df_ml.Atendida
smt = SMOTE()
X_os,y_os = smt.fit_sample(X,y) #os de oversampled
counter = Counter(y_os)
print(counter)
```

```
Counter({False: 6306, True: 6306})
```

In [13]:

```
#grafico da nova distribuição de classes
fig, ax = plt.subplots()
sns.countplot(y_os, ax=ax)
ax.set_title('Distribuição das Classes')
plt.xlabel('Classe')
plt.ylabel('Quantidade')
plt.tight_layout();

#print do balanceamento
print(pd.Series(y_os).value_counts())
```

```
True     6306
False    6306
Name: Atendida, dtype: int64
```



In [14]:

```
#Train_test_split nessa oversampled
#Especificamos o tamanho do test_size = 0.3 pq assim as True/False do ytreinamento e as
xtreinamento, xteste, ytreinamento, yteste = train_test_split(X_os, y_os, test_size = 
```

In [15]:

```
1 xtreinamento
```

Out[15]:

	InscritoDAU	Regiao_Centro-oeste	Regiao_Nordeste	Regiao_Norte	Regiao_Sudeste	Regiao_S
11454	True	0	0	0	1	
8765	False	0	0	0	1	
3191	False	0	0	0	1	
2792	False	0	0	0	1	
9375	True	0	0	0	1	
...	...	...	...	...	...	
12177	True	0	0	0	1	
8964	True	0	0	0	1	
4682	False	0	0	0	1	
5278	False	1	0	0	0	
7598	False	0	0	0	1	

8828 rows × 6927 columns

In [16]:

```
xteste
```

Out[16]:

	InscritoDAU	Regiao_Centro-oeste	Regiao_Nordeste	Regiao_Norte	Regiao_Sudeste	Regiao_Su
6594	False	0	0	0	0	
6068	False	0	0	0	1	(
3782	False	0	0	1	0	(
1772	False	0	0	0	0	
289	False	0	0	0	1	(
...	...	...	...	...	...	.
9219	True	0	0	1	0	(
43	False	0	0	0	1	(
4719	False	0	0	0	1	(
225	False	0	0	0	1	(
9295	False	0	1	0	0	(

3784 rows × 6927 columns

In [17]:

```
ytreinamento.value_counts()
```

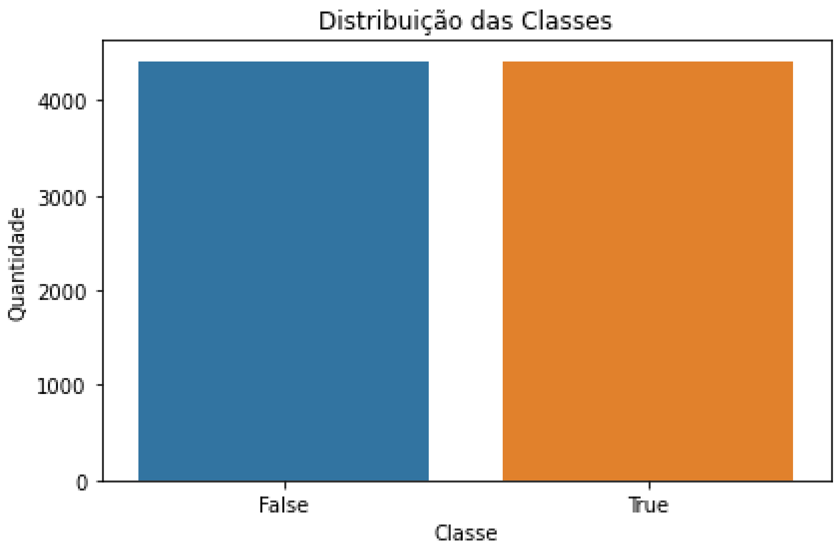
Out[17]:

```
True     4414
False    4414
Name: Atendida, dtype: int64
```

In [18]:

```
fig, ax = plt.subplots()
sns.countplot(ytreinamento, ax=ax)
ax.set_title('Distribuição das Classes')
plt.xlabel('Classe')
plt.ylabel('Quantidade')
plt.tight_layout();

#print do balanceamento
print(pd.Series(ytreinamento).value_counts())
```

```
True     4414
False    4414
Name: Atendida, dtype: int64
```



In [19]:

```
yteste.value_counts()
```

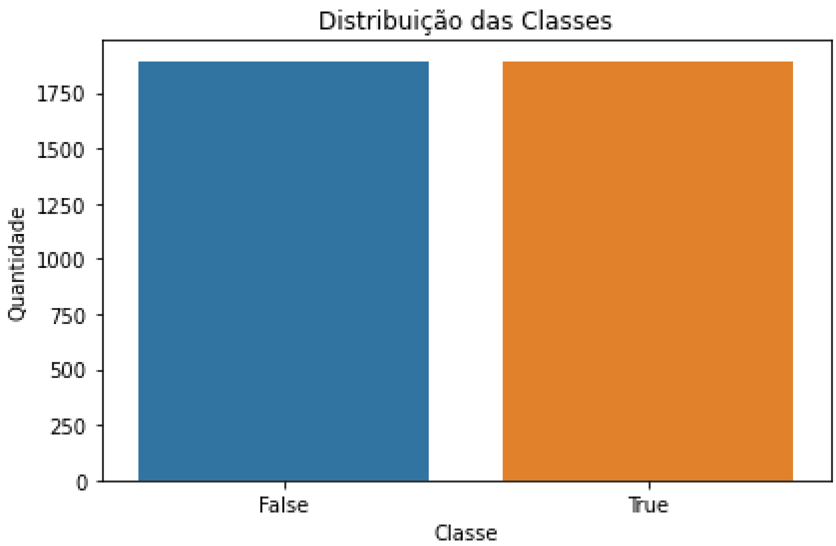
Out[19]:

```
True     1892
False    1892
Name: Atendida, dtype: int64
```

In [20]:

```
fig, ax = plt.subplots()
sns.countplot(yteste, ax=ax)
ax.set_title('Distribuição das Classes')
plt.xlabel('Classe')
plt.ylabel('Quantidade')
plt.tight_layout();

#print do balanceamento
print(pd.Series(ytreinamento).value_counts())
```

```
True     4414
False    4414
Name: Atendida, dtype: int64
```



In [21]:

```
#Neste ponto do notebook, as bases do "df_ml" estão balanceadas pelo SMOTE e separadas
```

PyCaret preparatório

Temos as bases balanceadas e já separadas no train_test_split. Todavia, não sabemos qual modelo de machine learning aplicar. Utilizaremos a ferramenta de automação de machine learning Pycaret apenas como guia para escolher os melhores algoritmos para implementação manual.

A documentação do PyCaret dispõe que, ao utilizar o parâmetro fix_imbalance=True, a biblioteca aplica, automaticamente, a técnica SMOTE. Dessa forma, não há necessidade de aplicar as bases balanceadas por SMOTE, às quais preparamos para o ML manual. Assim, utilizaremos, no PyCaret, a base "df_ml1". Como haverá SMOTE automático, a ml1 será semelhante à "df_ml" submetida ao SMOTE

In [22]:

```
#PyCaret no automático mas com fix_imbalance=True
#pycaret_df_ml = setup(data = ml1, target='Atendida',fix_imbalance=True)
#modelsml1 = compare_models()
#resultsml1 = pull()
```

#Resultado do PyCaret:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.7333	0.0000	0.7642	0.7873	0.7754	0.4471	0.4477	16.6980
et	Extra Trees Classifier	0.7308	0.7868	0.7998	0.7648	0.7816	0.4311	0.4325	33.3220
rf	Random Forest Classifier	0.7291	0.7867	0.8129	0.7560	0.7833	0.4231	0.4253	23.5580
lr	Logistic Regression	0.7250	0.7892	0.7380	0.7918	0.7638	0.4355	0.4374	31.5750
svm	SVM - Linear Kernel	0.7229	0.0000	0.7315	0.7974	0.7593	0.4327	0.4393	16.7670
dt	Decision Tree Classifier	0.7206	0.7016	0.7967	0.7538	0.7745	0.4081	0.4094	15.5360
lightgbm	Light Gradient Boosting Machine	0.7201	0.7718	0.7649	0.7696	0.7671	0.4163	0.4166	16.7510
gbc	Gradient Boosting Classifier	0.7027	0.7645	0.7344	0.7637	0.7486	0.3852	0.3859	27.2430
ada	Ada Boost Classifier	0.6925	0.7554	0.7000	0.7692	0.7327	0.3726	0.3752	17.9490
knn	K Neighbors Classifier	0.6345	0.7270	0.5246	0.7999	0.6333	0.2993	0.3262	32.0240
nb	Naive Bayes	0.5263	0.5962	0.2563	0.8570	0.3943	0.1625	0.2440	14.8830
lda	Linear Discriminant Analysis	0.4105	0.4492	0.4043	0.4638	0.4320	0.2171	0.2199	181.5890
qda	Quadratic Discriminant Analysis	0.3917	0.4549	0.1470	0.6861	0.2419	0.0912	0.1603	110.5300

ML manual a partir dos melhores modelos que prospectamos com o PyCaret: et, rf e lr. Privilegiando o recall e, também, para variar dos modelos de árvore, colocamos, também, o lightgbm. Plotamos Matriz de Confusão e grafico de ROC e AUC.

In [23]:

```
#Retomamos os train_test_split a partir do oversample ("_os") que já tínhamos feito
```

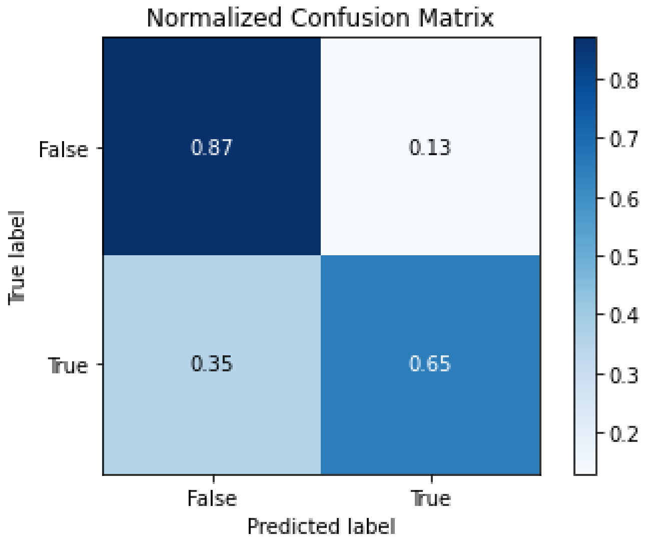
In [24]:

```
# Modelo Extra Trees Classifier
et = ExtraTreesClassifier(random_state=0)
et = et.fit(xtreinamento, ytreinamento)
Train_predict_et = et.predict(xteste)
print("Accuracy Score:", accuracy_score(yteste, Train_predict_et))
print(classification_report(yteste, Train_predict_et))
```

```
Accuracy Score: 0.7568710359408034
              precision    recall  f1-score   support

       False       0.71      0.87      0.78      1892
        True       0.83      0.65      0.73      1892

    accuracy                           0.76      3784
   macro avg       0.77      0.76      0.75      3784
weighted avg       0.77      0.76      0.75      3784
```

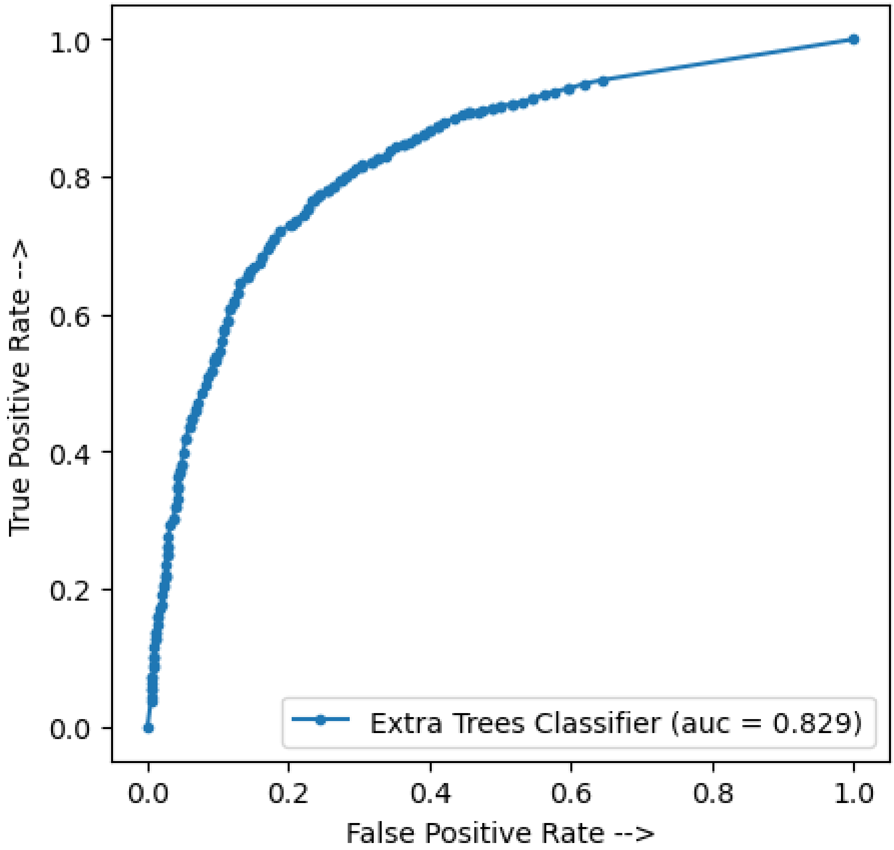
In [25]:

```
#Matriz de Confusão
skplt.metrics.plot_confusion_matrix(yteste, Train_predict_et, normalize=True)
plt.show()
```



In [26]:

```
# Curva ROC e área abaixo da curva (AUC)
y_pred_et = et.predict_proba(xteste)
et_fpr,et_tpr,thereshold = roc_curve(yteste,y_pred_et[:,1])
auc_et = auc(et_fpr,et_tpr)
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(et_fpr,et_tpr, marker='.', label='Extra Trees Classifier (auc = %0.3f)' % auc_
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')
plt.legend()
plt.show()
```



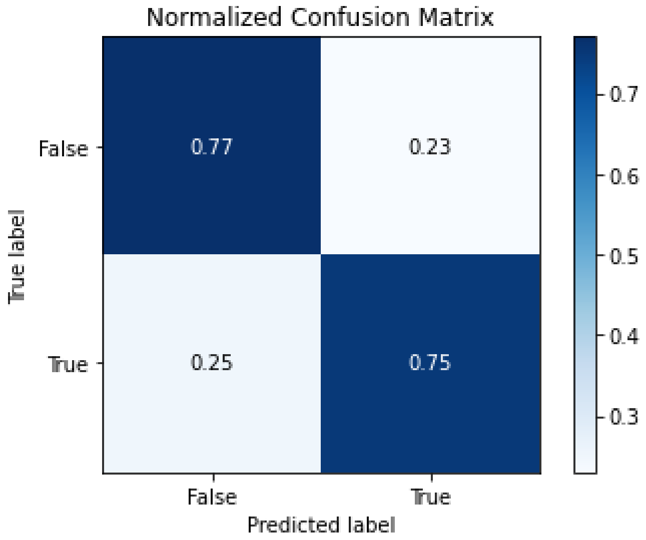
In [27]:

```
# Modelo Logistic Regression
lr = LogisticRegression(random_state=0)
lr = lr.fit(xtreinamento, ytreinamento)
Train_predict_lr = lr.predict(xteste)
print("Accuracy Score:", accuracy_score(yteste, Train_predict_lr))
print(classification_report(yteste, Train_predict_lr))
```

```
Accuracy Score: 0.7608350951374208
              precision    recall  f1-score   support

       False       0.76      0.77      0.76      1892
        True       0.77      0.75      0.76      1892

    accuracy                           0.76      3784
   macro avg       0.76      0.76      0.76      3784
weighted avg       0.76      0.76      0.76      3784
```

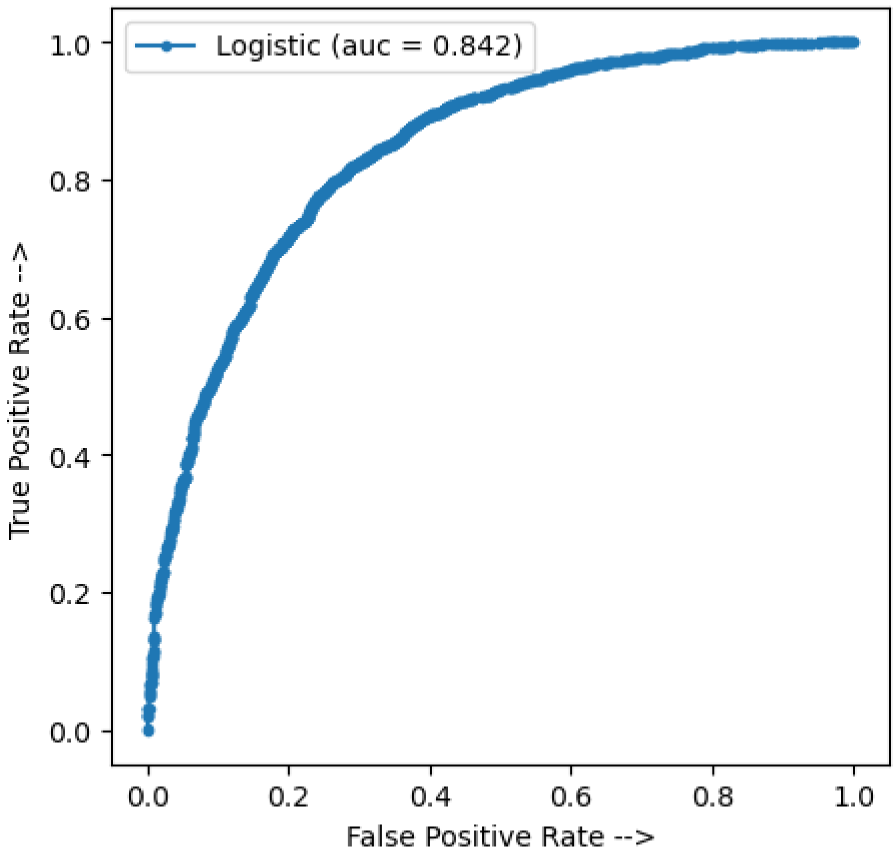
In [28]:

```
skplt.metrics.plot_confusion_matrix(yteste, Train_predict_lr, normalize=True)
plt.show()
```



In [29]:

```
Y_pred_lr=lr.decision_function(xteste)
logistic_fpr,logistic_tpr,thereshold = roc_curve(yteste,Y_pred_lr) # Y_pred_lr do decis
auc_logistic = auc(logistic_fpr, logistic_tpr)
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(logistic_fpr, logistic_tpr, marker='.', label='Logistic (auc = %0.3f)' % auc_l
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')
plt.legend()
plt.show()
```



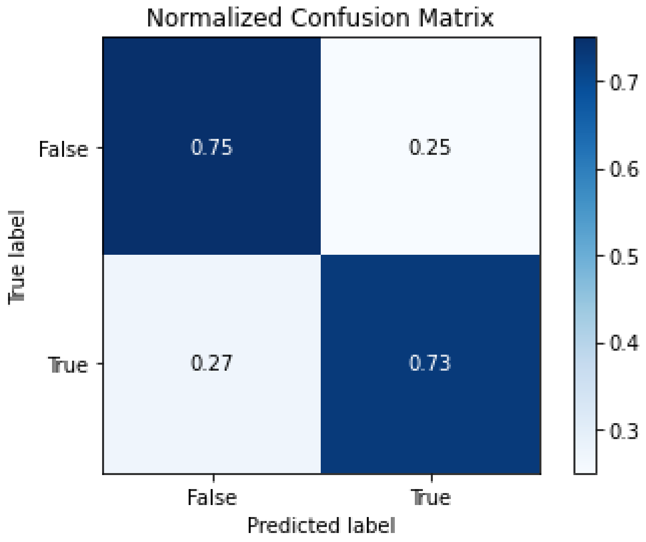
In [30]:

```
# Modelo Light Gradient Boosting Machine
import lightgbm
from lightgbm import LGBMClassifier
lightgbm = LGBMClassifier(random_state=0)
lightgbm = lightgbm.fit(xtreinamento, ytreinamento)
Train_predict_lightgbm = lightgbm.predict(xteste)
print("Accuracy Score:", accuracy_score(yteste, Train_predict_lightgbm))
print(classification_report(yteste, Train_predict_lightgbm))
```

```
Accuracy Score: 0.742600422832981
              precision    recall  f1-score   support

       False       0.74      0.75      0.74      1892
        True       0.75      0.73      0.74      1892

    accuracy                           0.74      3784
   macro avg       0.74      0.74      0.74      3784
weighted avg       0.74      0.74      0.74      3784
```

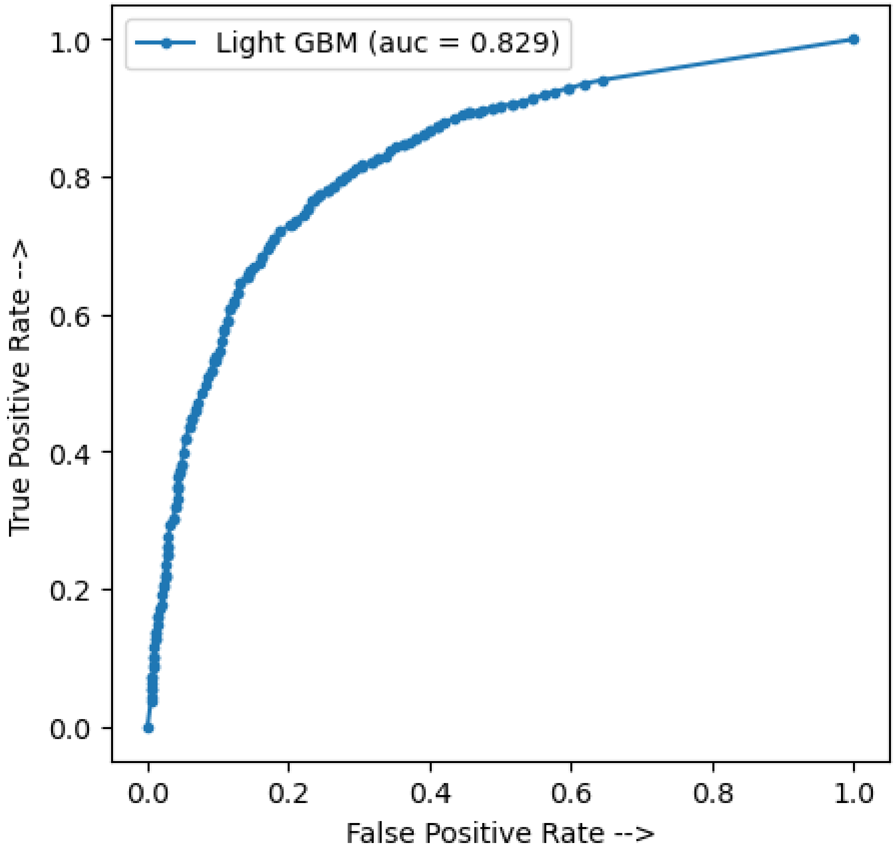
In [31]:

```
skplt.metrics.plot_confusion_matrix(yteste, Train_predict_lightgbm, normalize=True)
plt.show()
```



In [32]:

```
y_pred_lightgbm = lightgbm.predict_proba(xteste)
lightgbm_fpr,lightgbm_tpr,thereshold = roc_curve(yteste,y_pred_et[:,1])
auc_lightgbm = auc(lightgbm_fpr,lightgbm_tpr)
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(lightgbm_fpr,lightgbm_tpr, marker='.', label='Light GBM (auc = %0.3f)' % auc_l
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')
plt.legend()
plt.show()
```



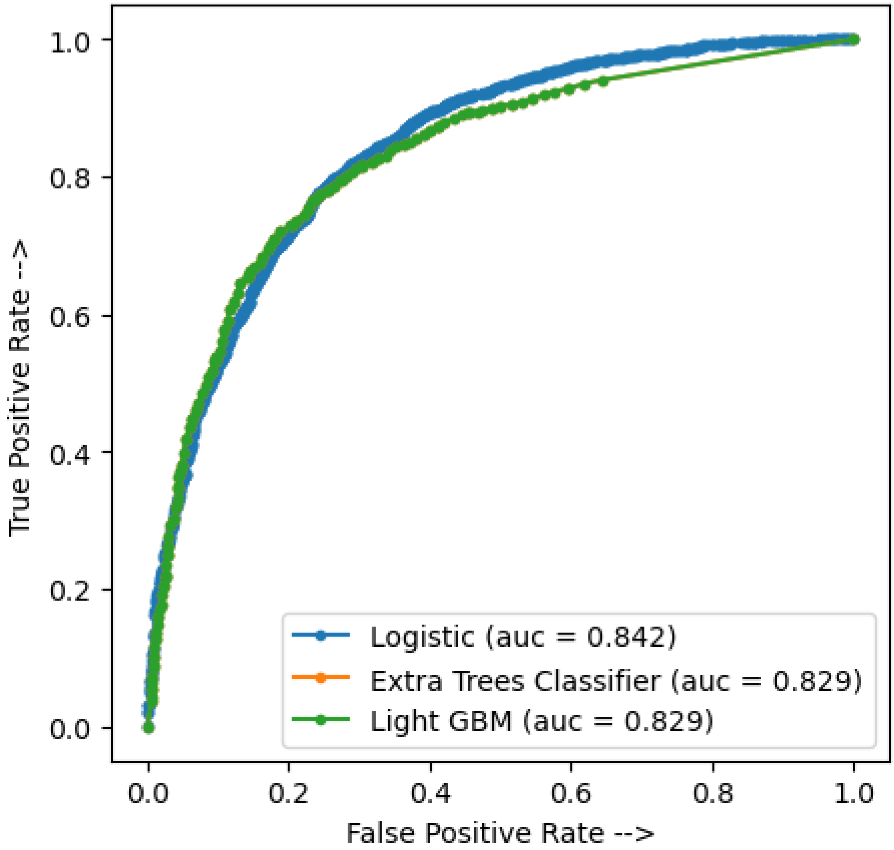
In [33]:

```
# Comparativo dos três modelos
print("EXTRA TREES CLASSIFIER:")
print("Accuracy Score:", accuracy_score(yteste, Train_predict_et))
print(classification_report(yteste, Train_predict_et))
print("REGRESSÃO LOGÍSTICA:")
print("Accuracy Score:", accuracy_score(yteste, Train_predict_lr))
print(classification_report(yteste, Train_predict_lr))
print("LIGHTGBM:")
print("Accuracy Score:", accuracy_score(yteste, Train_predict_lightgbm))
print(classification_report(yteste, Train_predict_lightgbm))
```

```
EXTRA TREES CLASSIFIER:
Accuracy Score: 0.7568710359408034
              precision    recall  f1-score   support

       False       0.71      0.87      0.78      1892
        True       0.83      0.65      0.73      1892

    accuracy                           0.76      3784
   macro avg       0.77      0.76      0.75      3784
weighted avg       0.77      0.76      0.75      3784

REGRESSÃO LOGÍSTICA:
Accuracy Score: 0.7608350951374208
              precision    recall  f1-score   support

       False       0.76      0.77      0.76      1892
        True       0.77      0.75      0.76      1892

    accuracy                           0.76      3784
   macro avg       0.76      0.76      0.76      3784
weighted avg       0.76      0.76      0.76      3784

LIGHTGBM:
Accuracy Score: 0.742600422832981
              precision    recall  f1-score   support

       False       0.74      0.75      0.74      1892
        True       0.75      0.73      0.74      1892

    accuracy                           0.74      3784
   macro avg       0.74      0.74      0.74      3784
weighted avg       0.74      0.74      0.74      3784
```

In [34]:

```
# Comparando AUC dos modelos ExtraTrees, LogisticRegression e LightGBM
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(logistic_fpr, logistic_tpr, marker='.', label='Logistic (auc = %0.3f)' % auc_l
plt.plot(et_fpr,et_tpr, marker='.', label='Extra Trees Classifier (auc = %0.3f)' % auc_
plt.plot(lightgbm_fpr,lightgbm_tpr, marker='.', label='Light GBM (auc = %0.3f)' % auc_l
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')
plt.legend()
plt.show()
```



A exigência da PUC Minas é de, no mínimo, três modelos de ML. Conforme demonstramos abaixo, o RandomForest gera basicamente o mesmo resultado do ExtraTrees. Dessa forma, utilizaremos, na versão a ser apresentada, os modelos ExtraTrees, LogisticRegression e LightGBM.

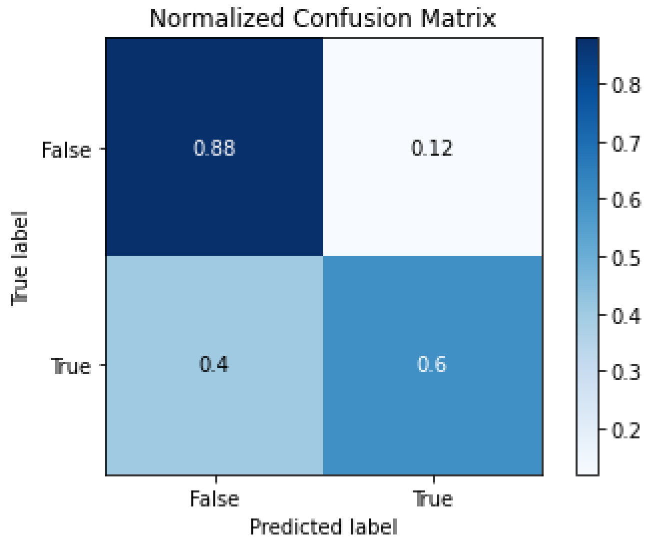
In [35]:

```
#Modelo RandomForest
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
tp_rfm = rfm.predict(xteste)
print("Accuracy Score:", accuracy_score(yteste, tp_rfm))
print(classification_report(yteste, tp_rfm))
```

```
Accuracy Score: 0.7415433403805497
              precision    recall  f1-score   support

       False       0.69      0.88      0.77      1892
        True       0.84      0.60      0.70      1892

    accuracy                           0.74      3784
   macro avg       0.76      0.74      0.74      3784
weighted avg       0.76      0.74      0.74      3784
```

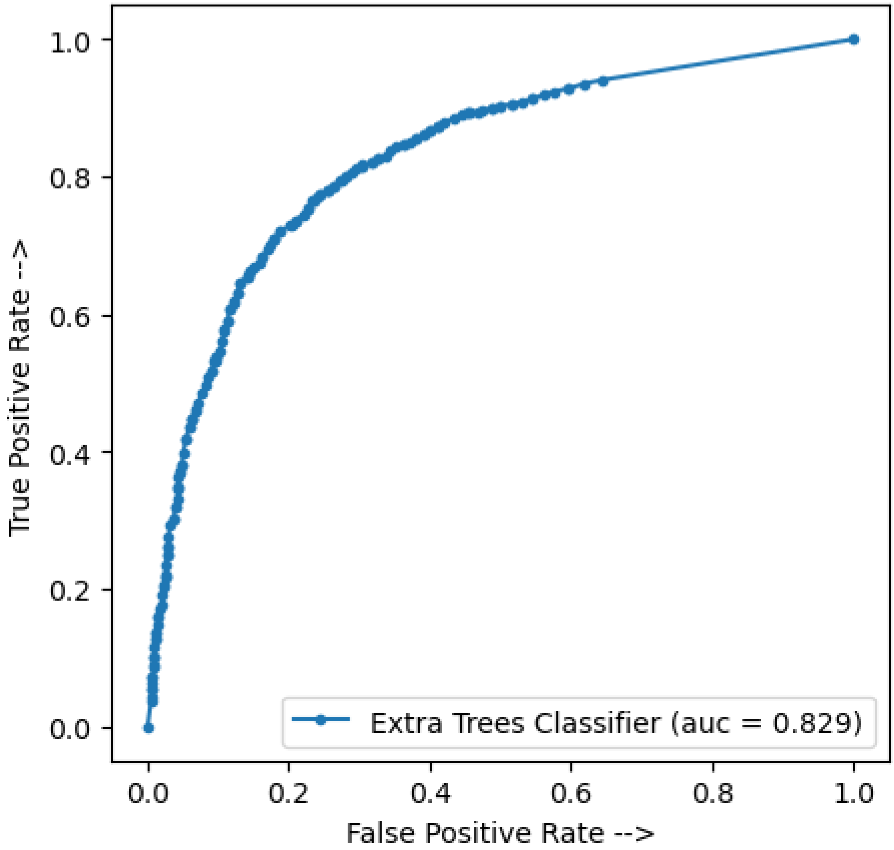
In [36]:

```
skplt.metrics.plot_confusion_matrix(yteste, tp_rfm, normalize=True)
plt.show()
```



In [37]:

```
y_pred_rfm = rfm.predict_proba(xteste)
rfm_fpr,rfm_tpr,thereshold = roc_curve(yteste,y_pred_et[:,1])
auc_rfm = auc(rfm_fpr,rfm_tpr)
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(rfm_fpr,rfm_tpr, marker='.', label='Extra Trees Classifier (auc = %0.3f)' % au
plt.xlabel('False Positive Rate -->')
plt.ylabel('True Positive Rate -->')
plt.legend()
plt.show()
```



Apêndice: testes mostrando variações que não foram aproveitadas no ML acima

1) Tentativa sem a coluna CEP

In [38]:

```
df_sem_cep = df_ml1[['Regiao','UF','CNAE','Atendida','CodAssunto','SexoConsumidor','Fa
```

In [39]:

```
df_sem_cep
```

Out[39]:

	Regiao	UF	CNAE	Atendida	CodAssunto	SexoConsumidor	FaixaEtaria	InscritoD
0	Norte	RO	6120501.0	False	187.0	M	5	T
1	Norte	RO	3514000.0	False	185.0	M	4	Fa
2	Norte	RO	8599604.0	True	236.0	M	3	Fa
3	Norte	RO	6120501.0	True	187.0	M	5	T
4	Norte	RO	6493000.0	False	57.0	M	6	T
...	...	...	...	...	...	...	...	
10514	Sudeste	SP	6110801.0	True	187.0	F	4	T
10515	Norte	RO	6143400.0	True	259.0	M	2	Fa
10516	Norte	RO	6422100.0	False	63.0	F	6	T
10517	Norte	RO	3514000.0	False	185.0	F	4	Fa
10518	Norte	RO	6423900.0	True	53.0	F	3	T

10519 rows × 8 columns

In [40]:

```
df_sem_cep = pd.get_dummies(df_sem_cep[['Regiao',
                            'UF',
                            'CNAE',
                            'Atendida',
                            'CodAssunto',
                            'SexoConsumidor',
                            'FaixaEtaria','InscritoDAU']])
```

In [41]:

```
df_sem_cep.shape # a do ml1 tinha 6922 colunas
```

Out[41]:

```
(10519, 574)
```

In [42]:

```
Xdf_sem_cep = df_sem_cep.drop(['Atendida'],axis=1)
ydf_sem_cep = df_sem_cep.Atendida
smt = SMOTE()
Xdf_sem_cep_os,ydf_sem_cep_os = smt.fit_sample(Xdf_sem_cep,ydf_sem_cep) #os de oversamp
counter = Counter(ydf_sem_cep_os)
print(counter)
```

```
Counter({False: 6306, True: 6306})
```

In [43]:

```
#Train_test_split nessa oversampled
# especificamos o tamanho do test_size = 0.3 pq assim as True/False do ytreinamento e 1
xtreinamentodf_sem_cep, xtestedf_sem_cep, ytreinamentodf_sem_cep, ytestedf_sem_cep = tr
```

ML do df_sem_cep

In [44]:

```
#Retomamos os train_test_split a partir do oversample ("_os") que já tínhamos feito
#xtreinamentodf_sem_cep, xtestedf_sem_cep, ytreinamentodf_sem_cep, ytestedf_sem_cep= tr
```

In [45]:

```
# Modelo Extra Trees Classifier
etdf_sem_cep = ExtraTreesClassifier(random_state=0)
etdf_sem_cep = etdf_sem_cep.fit(xtreinamentodf_sem_cep, ytreinamentodf_sem_cep)
print("Acurácia de treinamento: ", etdf_sem_cep.score(xtreinamentodf_sem_cep, ytreiname
Train_predict_etdf_sem_cep = etdf_sem_cep.predict(xtestedf_sem_cep)
print("Acurácia de previsão: ", accuracy_score(ytestedf_sem_cep, Train_predict_etdf_ser
print(classification_report(ytestedf_sem_cep, Train_predict_etdf_sem_cep))
```

```
Acurácia de treinamento:  0.9335070231082918
Acurácia de previsão:  0.7296511627906976
              precision    recall  f1-score   support

       False       0.70      0.81      0.75      1892
        True       0.77      0.65      0.71      1892

    accuracy                           0.73      3784
   macro avg       0.74      0.73      0.73      3784
weighted avg       0.74      0.73      0.73      3784
```

In [46]:

```
# Modelo Logistic Regression
lrdf_sem_cep = LogisticRegression(random_state=0)
lrdf_sem_cep = lrdf_sem_cep.fit(xtreinamentodf_sem_cep, ytreinamentodf_sem_cep)
print("Acurácia de treinamento: ", lrdf_sem_cep.score(xtreinamentodf_sem_cep, ytreiname
Train_predict_lrdf_sem_cep = lrdf_sem_cep.predict(xtestedf_sem_cep)
print("Acurácia de previsão: ", accuracy_score(ytestedf_sem_cep, Train_predict_lrdf_ser
print(classification_report(ytestedf_sem_cep, Train_predict_lrdf_sem_cep))
```

```
Acurácia de treinamento:  0.7642727684639783
Acurácia de previsão:  0.7431289640591966
              precision    recall  f1-score   support

       False       0.75      0.74      0.74      1892
        True       0.74      0.75      0.74      1892

    accuracy                           0.74      3784
   macro avg       0.74      0.74      0.74      3784
weighted avg       0.74      0.74      0.74      3784
```

In [47]:

```
# Modelo Light Gradient Boosting Machine
import lightgbm
from lightgbm import LGBMClassifier
lightgbmdf_sem_cep = LGBMClassifier(random_state=0)
lightgbmdf_sem_cep = lightgbmdf_sem_cep.fit(xtreinamentodf_sem_cep, ytreinamentodf_sem_
print("Acurácia de treinamento: ", lightgbmdf_sem_cep.score(xtreinamentodf_sem_cep, ytr
Train_predict_lightgbmdf_sem_cep = lightgbmdf_sem_cep.predict(xtestedf_sem_cep)
print("Acurácia de previsão: ", accuracy_score(ytestedf_sem_cep, Train_predict_lightgbr
print(classification_report(ytestedf_sem_cep, Train_predict_lightgbmdf_sem_cep))
```

```
Acurácia de treinamento:  0.7803579519710013
Acurácia de previsão:  0.7325581395348837
              precision    recall  f1-score   support

       False       0.73      0.74      0.73      1892
        True       0.74      0.72      0.73      1892

    accuracy                           0.73      3784
   macro avg       0.73      0.73      0.73      3784
weighted avg       0.73      0.73      0.73      3784
```

2) Tentativa com LabelEncoder para as de alta cardinalidade

A melhor técnica dispõe que devemos criar label enconders para as variáveis de alta cardinalidade. Não obstante, as nossas variáveis categóricas não podem sofrer o enviesamento: um CNAE 8630502 não é 8630502 vezes melhor ou pior do que o CNAE 0000001. Adotar os label encoders para as varáveis verdadeiramente explicativas - CNAE e CodAssunto - implicaria em um descolamento com a realidade dos problemas dessa natureza.

In [48]:

```
df_le = df_ml1 #leia-se "dataframe label encoder"
```

In [49]:

```
from sklearn.preprocessing import LabelEncoder
```

In [50]:

```
df_le.nunique()
```

Out[50]:

```
Regiao              5
UF                 15
CNAE              368
Atendida            2
CodAssunto        175
SexoConsumidor      2
FaixaEtaria         7
CEP              6354
InscritoDAU         2
dtype: int64
```

In [51]:

```
colunascategoricas = df_le.select_dtypes('object').columns
colunascategoricas
```

Out[51]:

```
Index(['Regiao', 'UF', 'CNAE', 'CodAssunto', 'SexoConsumidor', 'FaixaEtari
a',
       'CEP'],
      dtype='object')
```

In [52]:

```
for col in colunascategoricas:
    df_le[col+'_encoded'] = LabelEncoder().fit_transform(df_le[col])
```

In [53]:

```
df_le
```

Out[53]:

	Regiao	UF	CNAE	Atendida	CodAssunto	SexoConsumidor	FaixaEtaria	CE
0	Norte	RO	6120501.0	False	187.0	M	5	76824042
1	Norte	RO	3514000.0	False	185.0	M	4	76824322
2	Norte	RO	8599604.0	True	236.0	M	3	78932000
3	Norte	RO	6120501.0	True	187.0	M	5	78932000
4	Norte	RO	6493000.0	False	57.0	M	6	76821331
...	...	...	...	...	...	...	...	
10514	Sudeste	SP	6110801.0	True	187.0	F	4	9617000
10515	Norte	RO	6143400.0	True	259.0	M	2	76940000
10516	Norte	RO	6422100.0	False	63.0	F	6	76990000
10517	Norte	RO	3514000.0	False	185.0	F	4	76807400
10518	Norte	RO	6423900.0	True	53.0	F	3	76806420

10519 rows × 16 columns

In [54]:

```
# Para definir as colunas que permanecerem no nosso nosso dataframe selecionaremos:
# para de alta cardinalidade, mantemos LabelEncoder;
# para as de baixa e média, criaremos dummies
```

In [55]:

```
df_le = df_le[['Regiao','UF','CNAE_encoded','Atendida','CodAssunto_encoded','SexoConsur
```

In [56]:

```
df_le
```

Out[56]:

	Regiao	UF	CNAE_encoded	Atendida	CodAssunto_encoded	SexoConsumidor	FaixaE
0	Norte	RO	231	False	54	M	
1	Norte	RO	80	False	52	M	
2	Norte	RO	330	True	80	M	
3	Norte	RO	231	True	54	M	
4	Norte	RO	262	False	132	M	
...	...	...	...	...	...	...	
10514	Sudeste	SP	228	True	54	F	
10515	Norte	RO	235	True	98	M	
10516	Norte	RO	248	False	138	F	
10517	Norte	RO	80	False	52	F	
10518	Norte	RO	249	True	128	F	

10519 rows × 9 columns

In [57]:

```
# Pois bem, vamos agora criar dummies para 'Regiao' (5 unique), 'UF' (15), SexoConsumic
```

In [58]:

```
df_le_du = pd.get_dummies(df_le,
                         columns = ['Regiao',
                                    'UF',
                                    'SexoConsumidor',
                                    'FaixaEtaria'],
                          prefix = ['Regiao',
                                    'UF',
                                    'SexoConsumidor',
                                    'FaixaEtaria'],
                          prefix_sep = '_' )

```

In [59]:

```
df_le_du.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10519 entries, 0 to 10518
Data columns (total 34 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CNAE_encoded         10519 non-null  int32
 1   Atendida             10519 non-null  bool
 2   CodAssunto_encoded   10519 non-null  int32
 3   CEP_encoded          10519 non-null  int32
 4   InscritoDAU          10519 non-null  bool
 5   Regiao_Centro-oeste  10519 non-null  uint8
 6   Regiao_Nordeste      10519 non-null  uint8
 7   Regiao_Norte         10519 non-null  uint8
 8   Regiao_Sudeste       10519 non-null  uint8
 9   Regiao_Sul           10519 non-null  uint8
 10  UF_CE                10519 non-null  uint8
 11  UF_ES                10519 non-null  uint8
 12  UF_GO                10519 non-null  uint8
 13  UF_MG                10519 non-null  uint8
 14  UF_MS                10519 non-null  uint8
 15  UF_MT                10519 non-null  uint8
 16  UF_PA                10519 non-null  uint8
 17  UF_PB                10519 non-null  uint8
 18  UF_PR                10519 non-null  uint8
 19  UF_RJ                10519 non-null  uint8
 20  UF_RN                10519 non-null  uint8
 21  UF_RO                10519 non-null  uint8
 22  UF_RS                10519 non-null  uint8
 23  UF_SC                10519 non-null  uint8
 24  UF_SP                10519 non-null  uint8
 25  SexoConsumidor_F     10519 non-null  uint8
 26  SexoConsumidor_M     10519 non-null  uint8
 27  FaixaEtaria_1        10519 non-null  uint8
 28  FaixaEtaria_2        10519 non-null  uint8
 29  FaixaEtaria_3        10519 non-null  uint8
 30  FaixaEtaria_4        10519 non-null  uint8
 31  FaixaEtaria_5        10519 non-null  uint8
 32  FaixaEtaria_6        10519 non-null  uint8
 33  FaixaEtaria_7        10519 non-null  uint8
dtypes: bool(2), int32(3), uint8(29)
memory usage: 441.8 KB
```

In [60]:

```
df_le_du # leia-se df label encoder com dummies
```

Out[60]:

	CNAE_encoded	Atendida	CodAssunto_encoded	CEP_encoded	InscritoDAU	Regiao_Ce c
0	231	False	54	5222	True	
1	80	False	52	5252	False	
2	330	True	80	5673	False	
3	231	True	54	5673	True	
4	262	False	132	5202	True	
...	...	...	...	...	...	
10514	228	True	54	6321	True	
10515	235	True	98	5494	False	
10516	248	False	138	5519	True	
10517	80	False	52	4978	False	
10518	249	True	128	4965	True	

10519 rows × 34 columns

In [61]:

```
df_le_du.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10519 entries, 0 to 10518
Data columns (total 34 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CNAE_encoded         10519 non-null  int32
 1   Atendida             10519 non-null  bool
 2   CodAssunto_encoded   10519 non-null  int32
 3   CEP_encoded          10519 non-null  int32
 4   InscritoDAU          10519 non-null  bool
 5   Regiao_Centro-oeste  10519 non-null  uint8
 6   Regiao_Nordeste      10519 non-null  uint8
 7   Regiao_Norte         10519 non-null  uint8
 8   Regiao_Sudeste       10519 non-null  uint8
 9   Regiao_Sul           10519 non-null  uint8
 10  UF_CE                10519 non-null  uint8
 11  UF_ES                10519 non-null  uint8
 12  UF_GO                10519 non-null  uint8
 13  UF_MG                10519 non-null  uint8
 14  UF_MS                10519 non-null  uint8
 15  UF_MT                10519 non-null  uint8
 16  UF_PA                10519 non-null  uint8
 17  UF_PB                10519 non-null  uint8
 18  UF_PR                10519 non-null  uint8
 19  UF_RJ                10519 non-null  uint8
 20  UF_RN                10519 non-null  uint8
 21  UF_RO                10519 non-null  uint8
 22  UF_RS                10519 non-null  uint8
 23  UF_SC                10519 non-null  uint8
 24  UF_SP                10519 non-null  uint8
 25  SexoConsumidor_F     10519 non-null  uint8
 26  SexoConsumidor_M     10519 non-null  uint8
 27  FaixaEtaria_1        10519 non-null  uint8
 28  FaixaEtaria_2        10519 non-null  uint8
 29  FaixaEtaria_3        10519 non-null  uint8
 30  FaixaEtaria_4        10519 non-null  uint8
 31  FaixaEtaria_5        10519 non-null  uint8
 32  FaixaEtaria_6        10519 non-null  uint8
 33  FaixaEtaria_7        10519 non-null  uint8
dtypes: bool(2), int32(3), uint8(29)
memory usage: 441.8 KB
```

Agora, então, temos dummies para as de baixa e média cardinalidade: 'Regiao' (5 unique), 'UF' (15 unique), SexoConsumidor' (2 unique) e 'FaixaEtaria' (7); e label encoder para as de alta cardinalidade 'CNAE_encoded' (368 unique), 'CodAssunto_encoded'(175 unique) e CEP (6354 unique)

In [62]:

```
Xdf_le_du = df_le_du.drop(['Atendida'],axis=1)
ydf_le_du = df_le_du.Atendida
smt = SMOTE()
Xdf_le_du_os,ydf_le_du_os = smt.fit_sample(Xdf_le_du,ydf_le_du) #os de oversampled
counter = Counter(ydf_le_du_os)
print(counter)
```

```
Counter({False: 6306, True: 6306})
```

In [63]:

```
#Train_test_split nessa oversampled
# especificamos o tamanho do test_size = 0.3 pq assim as True/False do ytreinamento fic
#True/False do yteste também ficam iguais
xtreinamentodf_le_du, xtestedf_le_du, ytreinamentodf_le_du, ytestedf_le_du = train_test
```

In [64]:

```
#Retomamos os train_test_split a partir do oversample ("_os") que já tínhamos feito
#xtreinamentodf_le_du, xtestedf_le_du, ytreinamentodf_le_du, ytestedf_le_du = train_tes
```

ML do df_le_du ("df label encoder dummies")

In [65]:

```
# Modelo Extra Trees Classifier
etdf_le_du = ExtraTreesClassifier(random_state=0)
etdf_le_du = etdf_le_du.fit(xtreinamentodf_le_du, ytreinamentodf_le_du)
print("Acurácia de treinamento: ", etdf_le_du.score(xtreinamentodf_le_du, ytreinamentod
Train_predict_etdf_le_du = etdf_le_du.predict(xtestedf_le_du)
print("Acurácia de previsão: ", accuracy_score(ytestedf_le_du, Train_predict_etdf_le_du
print(classification_report(ytestedf_le_du, Train_predict_etdf_le_du))
```

```
Acurácia de treinamento:  0.9971681014952424
Acurácia de previsão:  0.7270084566596194
              precision    recall  f1-score   support

       False       0.73      0.72      0.72      1892
        True       0.72      0.74      0.73      1892

    accuracy                           0.73      3784
   macro avg       0.73      0.73      0.73      3784
weighted avg       0.73      0.73      0.73      3784
```

In [66]:

```
# Modelo Logistic Regression
lrdf_le_du = LogisticRegression(random_state=0)
lrdf_le_du = lrdf_le_du.fit(xtreinamentodf_le_du, ytreinamentodf_le_du)
print("Acurácia de treinamento: ", lrdf_le_du.score(xtreinamentodf_le_du, ytreinamentod
Train_predict_lrdf_le_du = lrdf_le_du.predict(xtestedf_le_du)
print("Acurácia de previsão: ", accuracy_score(ytestedf_le_du, Train_predict_lrdf_le_du
print(classification_report(ytestedf_le_du, Train_predict_lrdf_le_du))
```

```
Acurácia de treinamento:  0.6746714997734481
Acurácia de previsão:  0.653276955602537
              precision    recall  f1-score   support

       False       0.66      0.64      0.65      1892
        True       0.65      0.67      0.66      1892

    accuracy                           0.65      3784
   macro avg       0.65      0.65      0.65      3784
weighted avg       0.65      0.65      0.65      3784
```

In [67]:

```
#Modelo Light Gradient Boosting Machine
import lightgbm
from lightgbm import LGBMClassifier
lightgbmdf_le_du = LGBMClassifier(random_state=0)
lightgbmdf_le_du = lightgbmdf_le_du.fit(xtreinamentodf_le_du, ytreinamentodf_le_du)
print("Acurácia de treinamento: ", lightgbmdf_le_du.score(xtreinamentodf_le_du, ytreina
Train_predict_lightgbmdf_le_du = lightgbmdf_le_du.predict(xtestedf_le_du)
print("Acurácia de previsão: ", accuracy_score(ytestedf_le_du, Train_predict_lightgbmdf
print(classification_report(ytestedf_le_du, Train_predict_lightgbmdf_le_du))
```

```
Acurácia de treinamento:  0.8333710919800634
Acurácia de previsão:  0.7640063424947146
              precision    recall  f1-score   support

       False       0.77      0.76      0.76      1892
        True       0.76      0.77      0.77      1892

    accuracy                           0.76      3784
   macro avg       0.76      0.76      0.76      3784
weighted avg       0.76      0.76      0.76      3784
```

Não obstante melhorar o desempenho dos modelos Extra Trees e LightGBM, houve piora no modelo escolhido, o de Regressão Logística.