In [ ]:

```python
# Essentials
import numpy as np
import pandas as pd

# Plots
import seaborn as sns
import matplotlib.pyplot as plt

# Models Regression
import lightgbm as lgb

# Models Classification
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

# Misc
import math
import shap
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from hyperopt import fmin, hp, tpe, Trials, space_eval, STATUS_OK
from hyperopt.pyll import scope
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Feature Selection
from sklearn.feature_selection import VarianceThreshold
from sklearn.base import BaseEstimator, TransformerMixin


import warnings
warnings.filterwarnings("ignore")
```

Utils

In [2]:

```python
def scoringProfit(yTrue, pred, doubleAnalysis = False):
    '''
        Custrom metric to compute profit where TP = 90 and
        FP = -10 in generating the metric

        Args:
            yTrue: Array with ground truth
            pred: Array with predict from model
            doubleAnalysis: Boolean to say if return one metric or two

        Returns:
            Scorer: Scorer of  profit by customer or
                    profit by customer and all profit
    '''
```

```python
    # ### Matriz de confusão com (tn,fp,fn,tp)
    crosstab = metrics.confusion_matrix(yTrue, pred)

    if len(pred) < 4:
        tn, fp, fn, tp = 0, 0, 0, 0
    else:
        tn, fp, fn, tp = crosstab.ravel()

    # Compute profit
    profit = (tp*90 - fp*10)

    # Compute profit per customer
    profitCustomer = profit/len(pred)

    if doubleAnalysis:
        return profitCustomer, profit

    return profitCustomer
```

In [3]:

```python
def thresholdsCurve(y_true = None, y_proba = None):
  '''
    Calculate the best threshold to maximize profit

    Args:
      y_true (np.array):
      y_proba (np.array):
  '''

  thresholds = np.arange(0.0, 1.0, 0.01)
  threshold_gain_max = 0.0

  gain_score_eval = list()
  gain_score_max = 0.0

  gain_tp = 90
  gain_fp = -10

  for threshold in thresholds:
    y_pred = np.array(list(map(lambda proba: 1 if proba >= threshold else 0,
    tn, fp, fn, tp = metrics.confusion_matrix(y_true, y_pred).ravel()
    total_clientes = int(tp + tn + fp + fn)
    gain = round((tp * gain_tp + fp * gain_fp) / total_clientes, 2)

    gain_score_eval.append(gain)

    if gain >= gain_score_max and threshold >= threshold_gain_max:
      gain_score_max = gain
      threshold_gain_max = threshold

  plt.figure(figsize= (5, 5))
  sns.lineplot(x = thresholds, y = gain_score_eval, color= "red")

  plt.text(x = 0.45,
           y = gain_score_max*1.1,
           s = f"THRESHOLD: {threshold_gain_max:.2f},\nLucro Per. Cliente (R
```

```
           bbox = {'facecolor': 'gray','alpha': 0.4, 'pad': 6}
  )

  plt.title("Curva Lucro Per. Cliente(R$) x THRESHOLD")
  plt.legend(["Classe 1 - Insatisfeito"])
  plt.ylim(0, gain_score_max*1.35)
  plt.ylabel("Lucro Per. Cliente(R$)")
  plt.xlabel("Thresholds")
  plt.show()
```

Carregamento dos Dados

In [4]:
```
dfTrain = pd.read_csv('train_features.csv')
yTrain = dfTrain.TARGET
dfTrain = dfTrain.drop(labels=['TARGET'], axis=1)

dfVal = pd.read_csv('val_features.csv')
yVal = dfVal.TARGET
dfVal = dfVal.drop(labels=['TARGET'], axis=1)

dfTest = pd.read_csv('test_features.csv')
```

Solução

1.0 - Classificação (Case A)

1.1 - Preparação do Dataset

In [78]:
```
modelTransform = Pipeline(
                 [
                    ('scale', PowerTransformer(method='yeo-johnson'))
                ]
            )
modelTransform.fit(dfTrain, yTrain)
```

Out[78]:
```
▸ Pipeline
  ▸ PowerTransformer
```

In [79]:
```
dfTrainCLF = modelTransform.transform(dfTrain)
dfValCLF = modelTransform.transform(dfVal)
dfTestCLF = modelTransform.transform(dfTest)
```

1.2 - Ponto de Partida

Como base line vamos treinar uma simples árvore de decisão sem fazer nenhuma seleção de atribuitos ou tunning de hyperparametros.

In [11]:
```
# Model base
clRF = RandomForestClassifier(n_jobs = -1, class_weight='balanced')
clRF.fit(dfTrainCLF, yTrain)
preds = clRF.predict(dfValCLF)
rfPerCustomer, rfProfit = scoringProfit(yVal, preds, True)

# Lucro caso a campanha impacte todos os clientes
allCustomerProfitPerCustomer, allCustomerProfit = scoringProfit(yVal, [1 for

# Verificação do score máximo possível no treino
maxProfitPerCustomer, maxProfit = (scoringProfit(yVal, yVal, True))

print(f"Lucro no modelo base R${rfProfit:,.2f}")
print(f"Lucro por cliente no modelo base R$ {rfPerCustomer:,.2f}\n")

print(f"Lucro caso a capanha seja aplicada a todos os clientes R$ {allCustom
print(f"Lucro por cliente caso a capanha seja aplicada a todos os clientes R

print(f"Lucro caso a capanha seja aplicada apenas aos clientes insatisfeitos
print(f"Lucro por cliente caso a capanha seja aplicada apenas aos clientes i
```

```
Lucro no modelo base R$1,040.00
Lucro por cliente no modelo base R$ 0.07

Lucro caso a capanha seja aplicada a todos os clientes R$ -88,300.00
Lucro por cliente caso a capanha seja aplicada a todos os clientes R$ -6.22

Lucro caso a capanha seja aplicada apenas aos clientes insatisfeitos R$ 48,2
40.00
Lucro por cliente caso a capanha seja aplicada apenas aos clientes insatisfe
itos R$ 3.40
```

1.3 - Busca dos Melhores Parâmetros

1.3.0 - LGBM

In [10]:
```
def objective(params):
    classifier = lgb.LGBMClassifier(verbose = -1,
                                    n_jobs = -1,
                                    class_weight='balanced',
                                    objective='binary',
                                    random_state=424, **params)

    model = Pipeline(
                [
                   ('classifier', classifier)
                 ]
            )

    model.fit(dfTrainCLF, yTrain)
```

```
    yPred = classifier.predict(dfValCLF)

    score = scoringProfit(yVal, yPred)

    return {
        'loss': -score,
        'status': STATUS_OK,
    }
```

In [11]:
```
spaceLGBM = {
    'n_estimators': scope.int(hp.uniform('n_estimators', 50, 5000)),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.01, 1.0),
    'learning_rate': hp.choice('learning_rate', np.arange(0.05, 0.35, 0.02))
    'max_depth': scope.int(hp.quniform('max_depth', 3, 20, 1)),
    'min_child_weight': scope.int(hp.quniform('min_child_weight', 3, 20, 1))
    'subsample': hp.uniform('subsample', 0.01, 1.0),
    'reg_lambda': hp.lognormal('reg_lambda', 0.0, 1.0),
    'reg_alpha': hp.lognormal('reg_alpha', 0.0, 1.0),

}
```

In [12]:
```
trials = Trials ()

bestParamsLGBM = fmin(
    fn=objective,
    space=spaceLGBM,
    algo=tpe.suggest,
    trials=trials,
    max_evals=1000
)
```

```
100%|██| 1000/1000 [1:40:31<00:00,  6.03s/trial, best loss: -0.8421423537702
607]
```

In [13]:
```
print(bestParamsLGBM)
```

```
{'colsample_bytree': 0.33833779547527354, 'learning_rate': 5, 'max_depth': 1
6.0, 'min_child_weight': 6.0, 'n_estimators': 2482.3448603876004, 'reg_alph
a': 7.20937108380558, 'reg_lambda': 1.985448182110237, 'subsample': 0.075906
31124333537}
```

1.3.1 - XGBoost

In [14]:
```
def objective(params):
    classifier = XGBClassifier(n_jobs = -1, objective='binary:logistic', rar

    model = Pipeline(
                [
                    ('classifier', classifier)
                ]
            )
    model.fit(dfTrainCLF, yTrain)

    yPred = classifier.predict(dfValCLF)
```

```
    score = scoringProfit(yVal, yPred)

    return {
        'loss': -score,
        'status': STATUS_OK,
        'booster': model.named_steps['classifier'].get_booster().attributes(
    }
```

In [15]:
```
spaceXGB = {
    'n_estimators': scope.int(hp.uniform('n_estimators', 50, 5000)),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.01, 1.0),
    'learning_rate': hp.choice('learning_rate', np.arange(0.05, 0.35, 0.02))
    'max_depth': scope.int(hp.quniform('max_depth', 3, 20, 1)),
    'scale_pos_weight': scope.int(hp.quniform('scale_pos_weight', 20, 30, 1)
    'min_child_weight': scope.int(hp.quniform('min_child_weight', 3, 20, 1))
    'subsample': hp.uniform('subsample', 0.01, 1.0),
    'reg_lambda': hp.lognormal('reg_lambda', 0.0, 1.0),
    'reg_alpha': hp.lognormal('reg_alpha', 0.0, 1.0)
}
```

In [16]:
```
trials = Trials ()

bestParamsXGB = fmin(
    fn=objective,
    space=spaceXGB,
    algo=tpe.suggest,
    trials=trials,
    max_evals=1000
)
```

```
100%|████| 1000/1000 [54:26<00:00,  3.27s/trial, best loss: -0.9344608879492
601]
```

In [17]:
```
print(bestParamsXGB)
```

```
{'colsample_bytree': 0.3126403997344753, 'learning_rate': 0, 'max_depth': 9.
0, 'min_child_weight': 4.0, 'n_estimators': 469.3250670352471, 'reg_alpha':
0.6860970446487238, 'reg_lambda': 3.6398478765329894, 'scale_pos_weight': 2
0.0, 'subsample': 0.9817877351312972}
```

In [18]:
```
print(trials.best_trial['result'])
```

```
{'loss': -0.9344608879492601, 'status': 'ok', 'booster': {}}
```

1.4 - Avaliar Melhor Modelo

1.4.0 - XGBoost

In [19]:
```
bestParamsXGB['learning_rate'] = np.arange(0.05, 0.35, 0.02)[0]
bestParamsXGB['n_estimators'] = int(bestParamsXGB['n_estimators'])
bestParamsXGB['max_depth'] = int(bestParamsXGB['max_depth'])
```

In [86]:
```
classifierXGB = XGBClassifier(n_jobs = -1, objective='binary:logistic', ranc
classifierXGB.fit(dfTrainCLF, yTrain)
```

Out[86]:

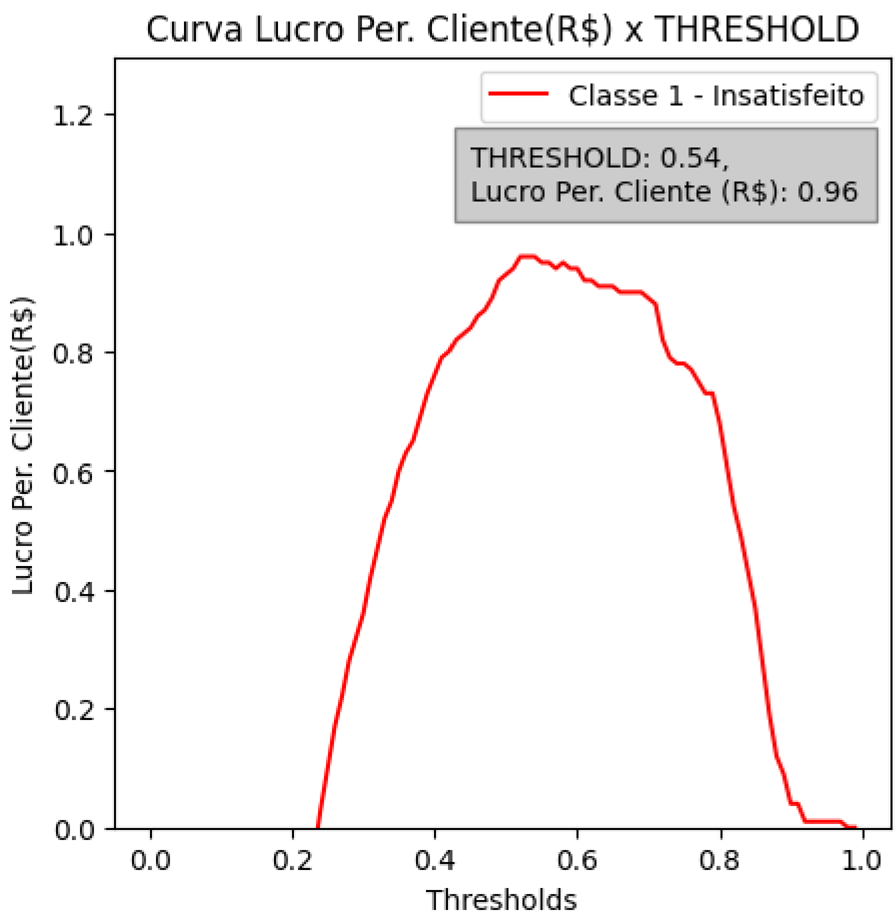
```
▾ XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.3126403997344753, device=None,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max
_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
```

In [40]:

```
yPred_prob = classifierXGB.predict_proba(dfValCLF)
```

In [41]:

```
thresholdsCurve(yVal, yPred_prob)
```



1.4.1 - LGBM

In [95]:

```
bestParamsLGBM['learning_rate'] = np.arange(0.05, 0.35, 0.02)[5]
bestParamsLGBM['n_estimators'] = int(bestParamsLGBM['n_estimators'])
bestParamsLGBM['max_depth'] = int(bestParamsLGBM['max_depth'])
```

In [96]:
```
classifierLGBM = lgb.LGBMClassifier(verbose = -1,
                                    n_jobs = -1,
                                    objective='binary',
                                    random_state=424,
                                    **bestParamsLGBM)
```

In [97]:
```
classifierLGBM.fit(dfTrainCLF, yTrain)
```

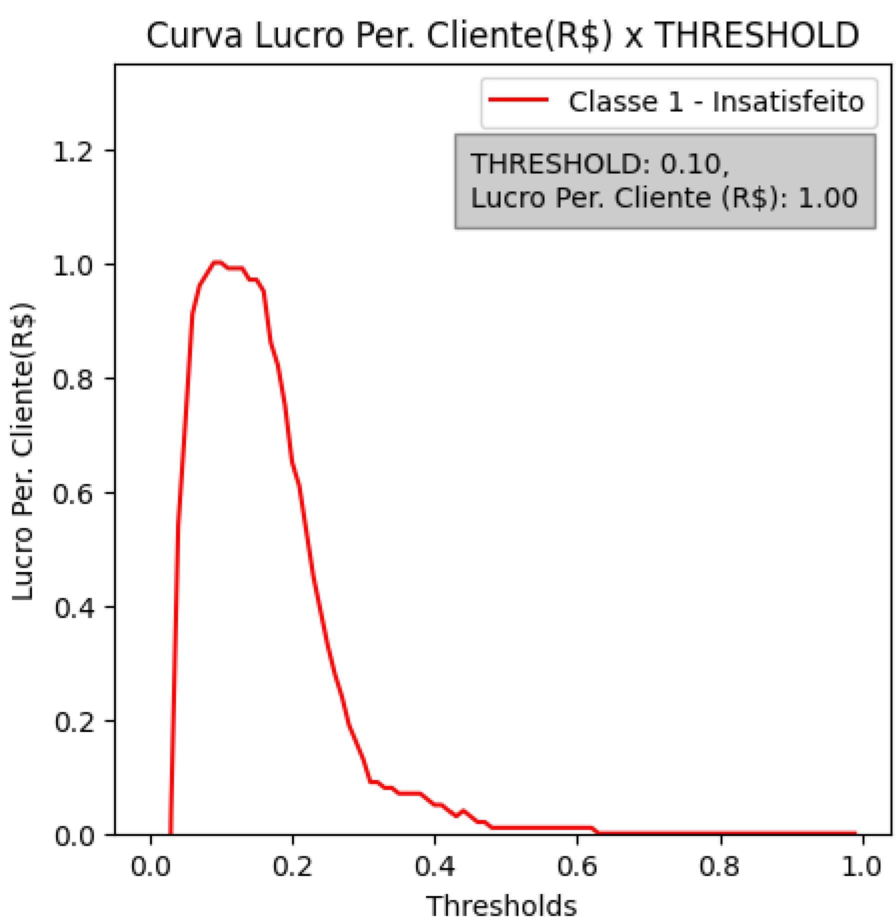
Out[97]:

LGBMClassifier

```
LGBMClassifier(colsample_bytree=0.33833779547527354,
               learning_rate=0.15000000000000002, max_depth=16,
               min_child_weight=6.0, n_estimators=2482, n_jobs=-1,
               objective='binary', random_state=424, reg_alpha=7.20
937108380558,
               reg_lambda=1.985448182110237, subsample=0.0759063112
4333537,
               verbose=-1)
```

In [98]:
```
yPred_prob = classifierLGBM.predict_proba(dfValCLF)
```

In [54]:
```
thresholdsCurve(yVal, yPred_prob)
```



Entre os dois melhores modelos o modelo que melhor performou em relação a métrica de avaliação foi o LGBM. Importante destacar que avaliamos sem o class_weight e o resultado foi infinitamente inferior.

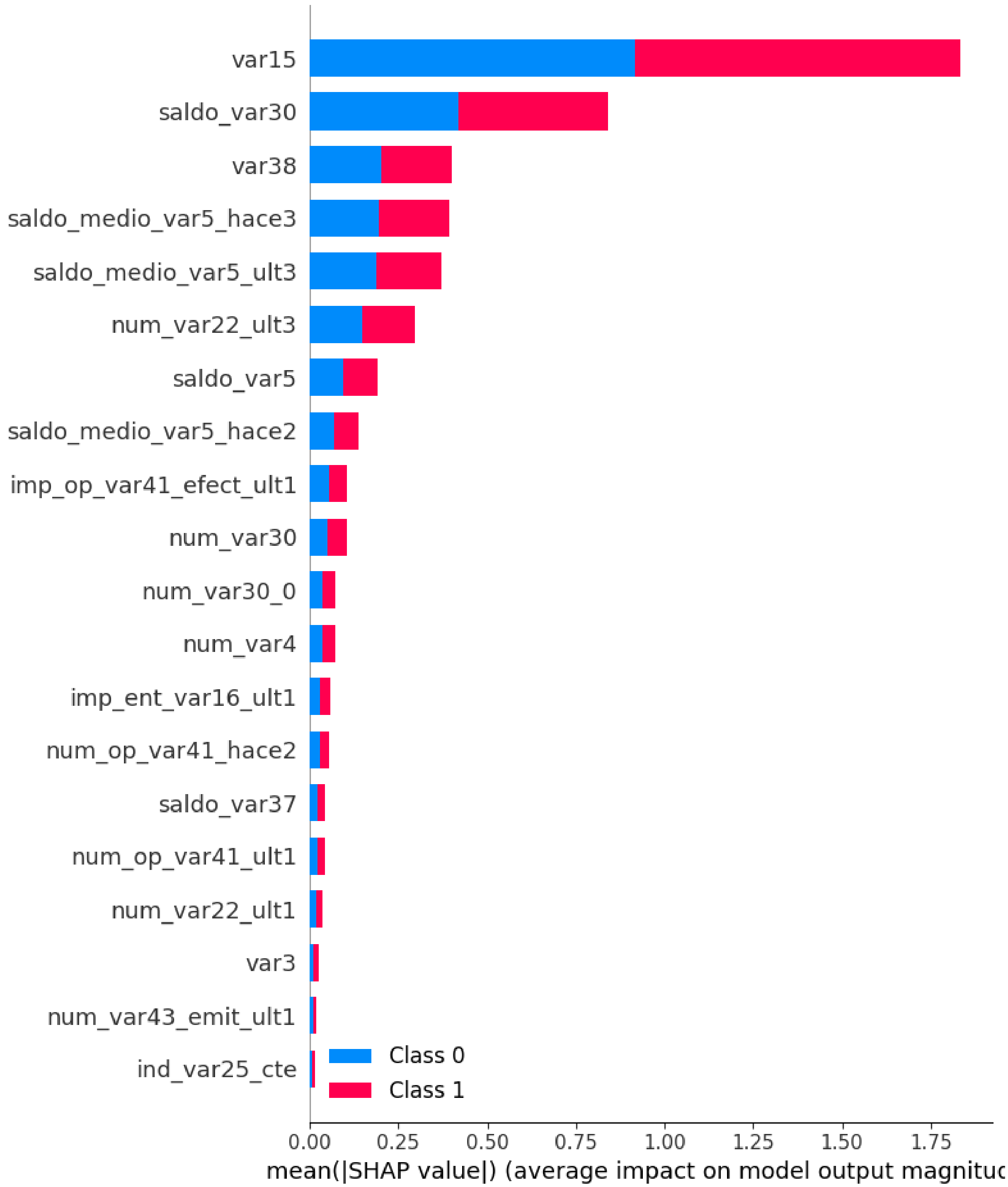
In [55]:
```
yPred = yPred_prob[:,1] > 0.10
profitClient, profit = scoringProfit(yVal, yPred, True)
profitClientMax, profitMax = scoringProfit(yVal, yVal, True)

print(f'Lucro máximo por cliente {profitClient:,.2f}')
print(f'Lucro máximo da campanha: R$ {profit:,.2f}')
print(f'Lucro máximo que poderia ser atingindo na campanha: R$ {profitMax:,.
print(f'Lucro máximo em %: {profit/profitMax:.2%}')
```

```
Lucro máximo por cliente 1.00
Lucro máximo da campanha: R$ 14,220.00
Lucro máximo que poderia ser atingindo na campanha: R$ 48,240.00
Lucro máximo em %: 29.48%
```

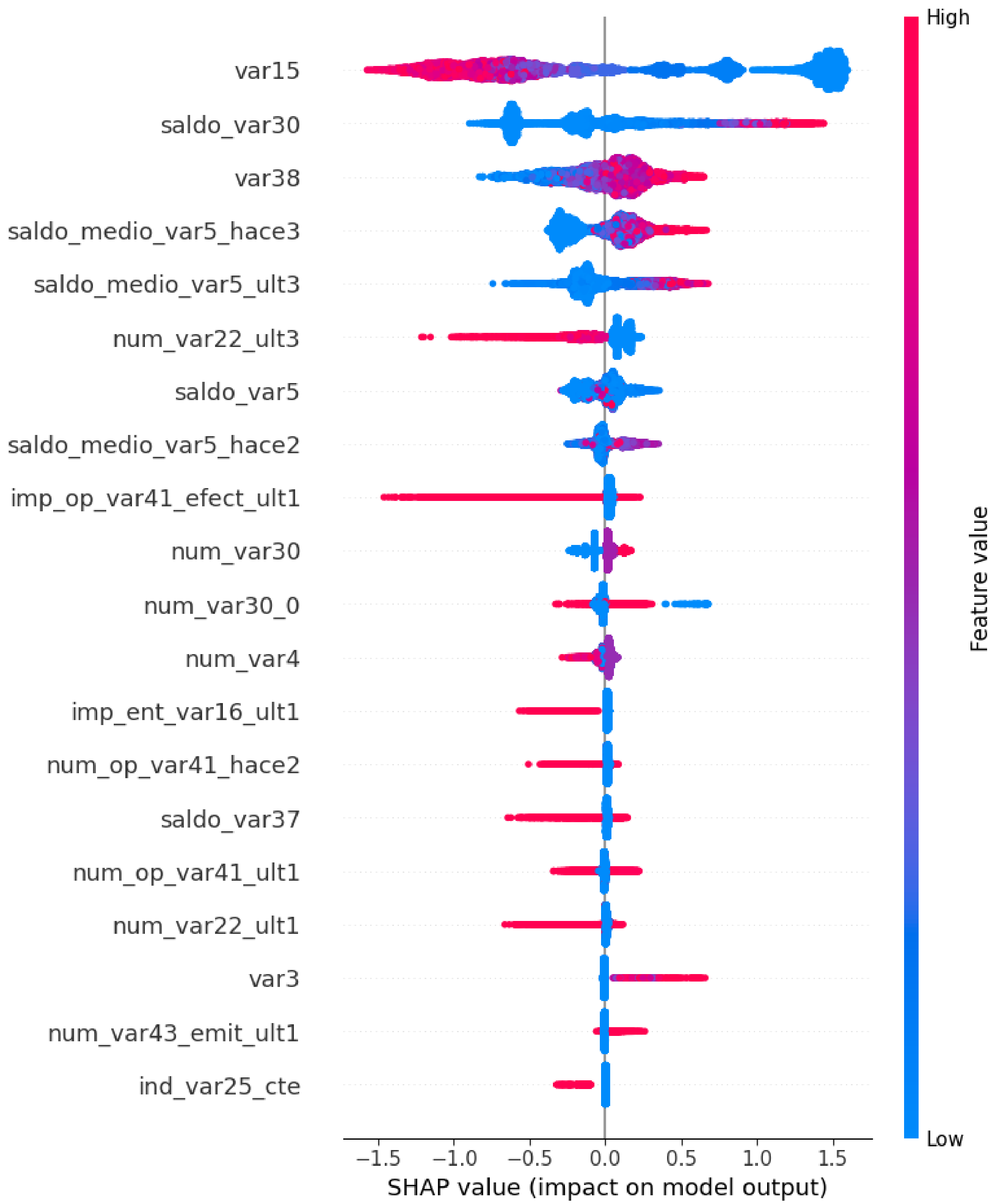
1.5 - Explicabilidade do Melhor Modelo (SHAP Values)

In [65]:
```
shap.initjs()
explainer = shap.TreeExplainer(classifierLGBM)
shap_values = explainer.shap_values(dfValCLF)
```

In [66]:
```
shap.summary_plot(shap_values, dfValCLF, feature_names=model.get_feature_nam
```



In [67]:
```
shap.summary_plot(shap_values[0], dfValCLF, feature_names=model.get_feature_
```



Como podemos observar todas as variáveis são importantes para o modelo. Sendo as variáveis var15, saldo_var30, var38... as mais importantes

2.0 - Notas (Case B)

In [100…

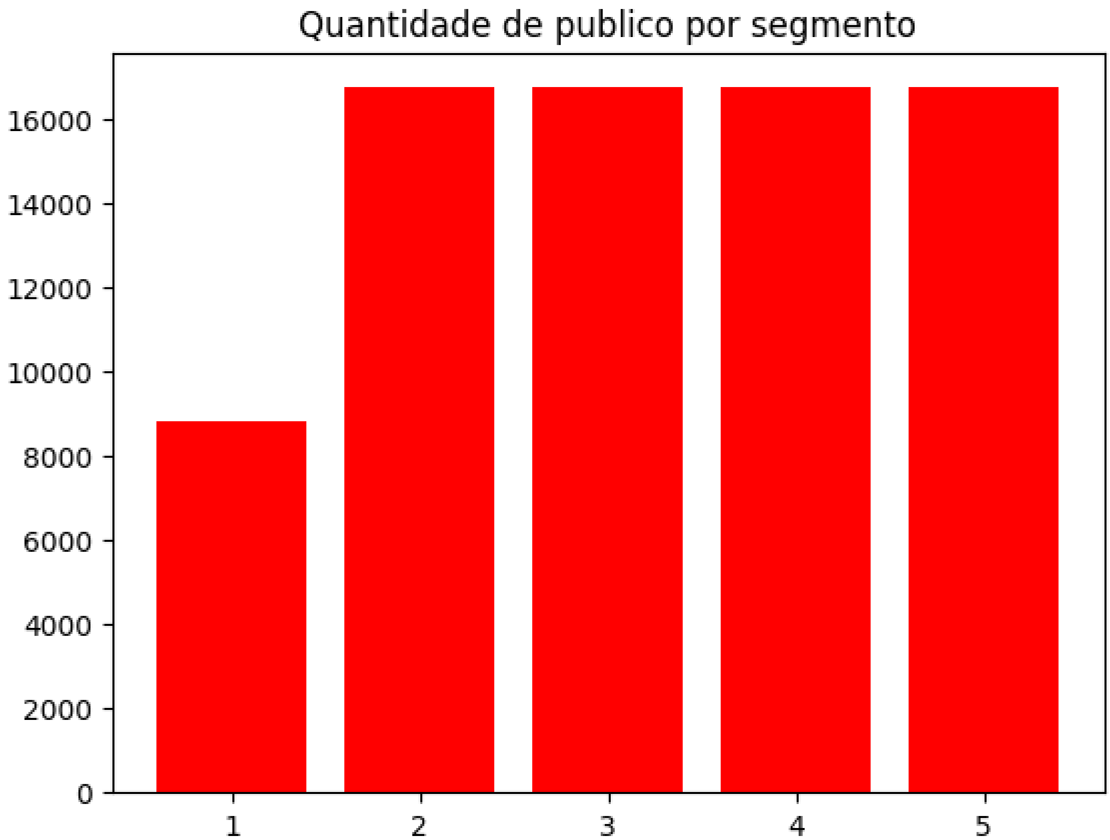
```
yPred_prob = classifierLGBM.predict_proba(dfTestCLF)
```

In [101…

```
# ### Separar os clientes classificados como insatisfeitos pelo modelo
df_preds_1 =  pd.DataFrame(yPred_prob_test[yPred_prob_test[:,1] > 0.10][:,1]
df_preds_0 =  pd.DataFrame(yPred_prob_test[yPred_prob_test[:,1] <= 0.10][:,1
```

```
# ### Dividir os registros classificados como satisfeitos em quartis baseado
decil_preds_0 = pd.qcut(df_preds_0[0].rank(method='first'), q = 4, labels =
pub_segmentado = np.concatenate((df_preds_1, decil_preds_0), axis=None)
segmentos, quantidade = np.unique(pub_segmentado, return_counts=True)

# ### Plot
plt.bar(segmentos, quantidade, color='red')
plt.title('Quantidade de publico por segmento');
print('A quantidade de publico no segmento 1 é menor pois trata-se do públic
```

```
A quantidade de publico no segmento 1 é menor pois trata-se do público categ
orizado como insatisfeito pelo modelo
```



3.0 - Clustering (Agrupamento)

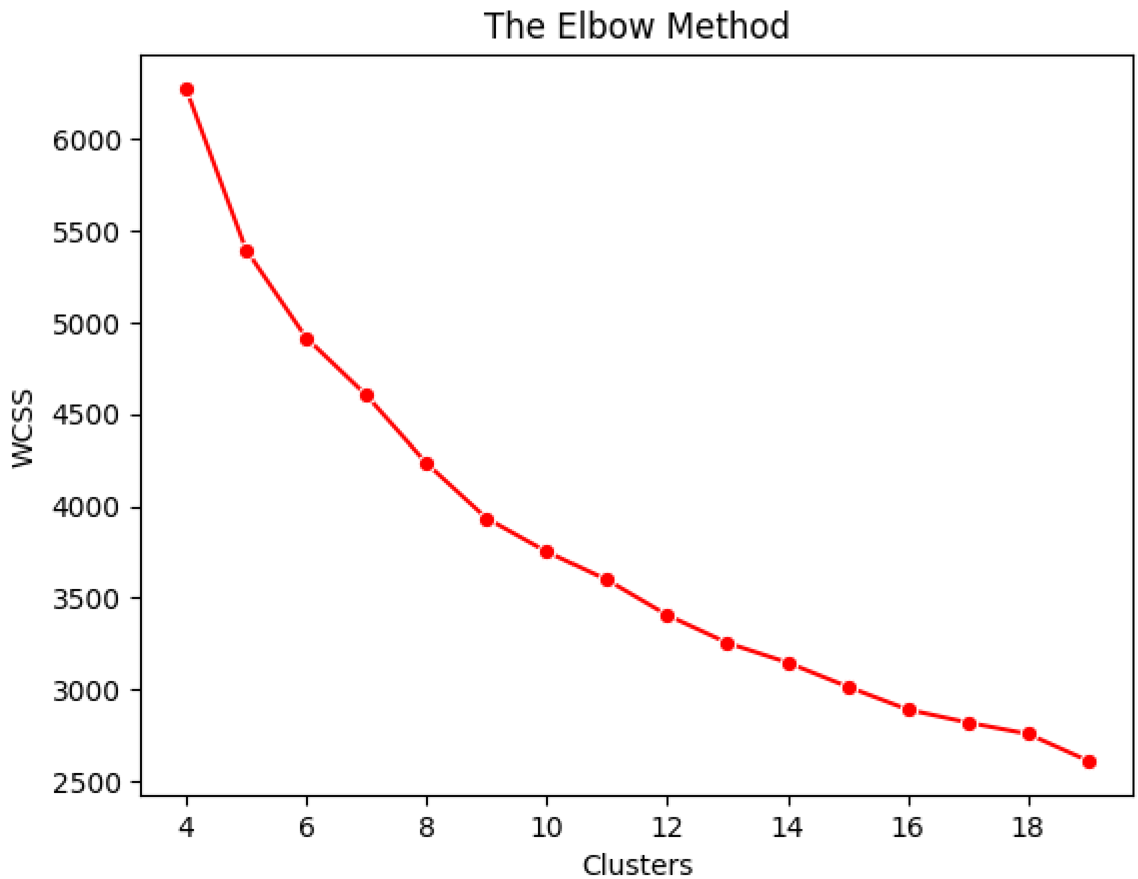
3.1 - Testar usando a normalização MinMaxScaler e todas as features

In [11]:
```
# ### Scaler feature for clustering
scaler = MinMaxScaler()
scaler.fit(dfTrain)

# ### Apply in train and val
xTrainScaledcaled = scaler.transform(dfTrain)
xValScaled = scaler.transform(dfVal)
```

In [12]:
```
minCluster = 4
maxCluster = 20
wcss = []

for nClusters in range(minCluster, maxCluster):

    # ### Train kmeans model
    kmeanModel = KMeans(n_clusters = nClusters, random_state = 424)
    kmeanModel.fit(xTrainScaledcaled)

    # ### Predict
    pred = kmeanModel.predict(xTrainScaledcaled)

    # ### Compute WCSS (Elbow)
    wcss.append(kmeanModel.inertia_)

    # ### Compute silhouette score
    score = silhouette_score(xTrainScaledcaled, pred)

    print(f'Silhouette Score for {nClusters} clusters: {score:.5f}')
```

```
Silhouette Score for 4 clusters: 0.41712
Silhouette Score for 5 clusters: 0.45717
Silhouette Score for 6 clusters: 0.46655
Silhouette Score for 7 clusters: 0.46847
Silhouette Score for 8 clusters: 0.47283
Silhouette Score for 9 clusters: 0.48264
Silhouette Score for 10 clusters: 0.48210
Silhouette Score for 11 clusters: 0.48380
Silhouette Score for 12 clusters: 0.48181
Silhouette Score for 13 clusters: 0.48411
Silhouette Score for 14 clusters: 0.48820
Silhouette Score for 15 clusters: 0.49153
Silhouette Score for 16 clusters: 0.48568
Silhouette Score for 17 clusters: 0.47026
Silhouette Score for 18 clusters: 0.47893
Silhouette Score for 19 clusters: 0.47660
```

In [13]:
```
mycenters = pd.DataFrame({'Clusters' : range(minCluster, maxCluster), 'WCSS'
sns.lineplot(x = 'Clusters', y = 'WCSS', data = mycenters, marker="o", color
plt.title('The Elbow Method')
plt.show()
```



In [14]:
```
def optimalNumberClusters(wcss):
    x1, y1 = 4, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]

    distances = []
    for i in range(len(wcss)):
        x0 = i+4
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = math.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 4

print(f'Melhor número de clusters de acordo com o método Elbow: {optimalNumb
```

```
Melhor número de clusters de acordo com o método Elbow: 9
```

3.1.0 - Com base no Elbow

In [23]:
```
# ### Avaliar com base no método Elbow
best_k = 9

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)
```

```
df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0   1  Ganho  Perda  Lucro
labels
5            129   4    360   1290   -930
3            359  28   2520   3590  -1070
7            190   0      0   1900  -1900
```

3.1.1 - Com base na Silhueta Score

In [21]:
```
best_k = 15

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0    1  Ganho  Perda  Lucro
labels
0            705  143  12870   7050   5820
14            72    7    630    720    -90
4             36    2    180    360   -180
```

3.2 - Testar usando a normalização MinMaxScaler e removendo as variáveis binárias

In [29]:
```
dfTrain.nunique()
```

```
Out[29]: var3                        195
         var15                        81
         imp_ent_var16_ult1          513
         imp_op_var41_efect_ult1     289
         ind_var1_0                    2
         ind_var25_cte                 2
         num_var4                      7
         num_op_var41_hace2           48
         num_op_var41_ult1            58
         num_var30_0                  11
         num_var30                     5
         num_var37_med_ult2           15
         saldo_var5                 9592
         saldo_var30               13848
         saldo_var37                3301
         imp_aport_var13_hace3       187
         num_var22_ult1               15
         num_var22_ult3               29
         num_meses_var8_ult3           4
         num_var43_emit_ult1          10
         saldo_medio_var5_hace2    12021
         saldo_medio_var5_hace3     5801
         saldo_medio_var5_ult3     14479
         saldo_medio_var8_hace3      304
         var38                     46609
         dtype: int64
```

```
In [57]: dfTrainDrop = dfTrain.drop(labels=['ind_var1_0', 'ind_var25_cte'] , axis=1,
         dfValDrop = dfVal.drop(labels=['ind_var1_0', 'ind_var25_cte'] , axis=1, inpl
```

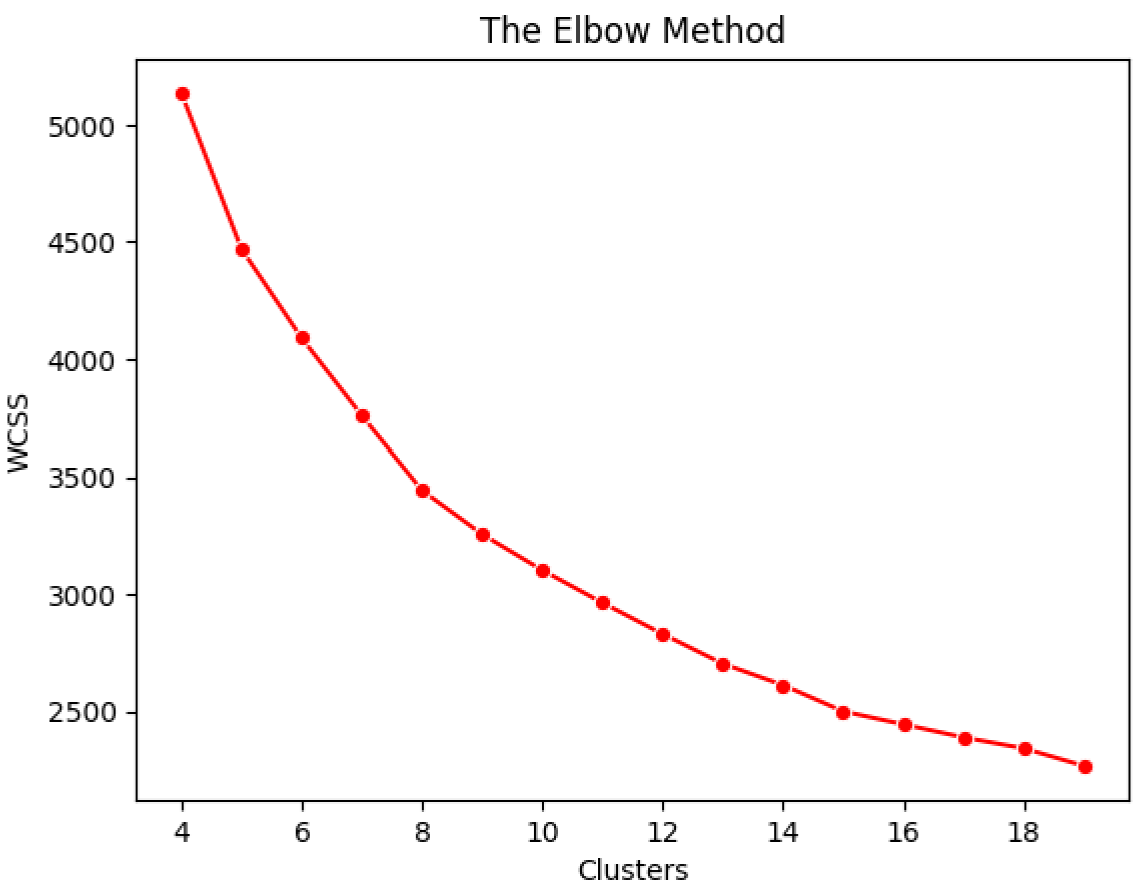
```
In [58]: # ### Scaler feature for clustering
         scaler = MinMaxScaler()
         scaler.fit(dfTrainDrop)

         # ### Apply in train and val
         xTrainScaledcaled = scaler.transform(dfTrainDrop)
         xValScaled = scaler.transform(dfValDrop)
```

```
In [59]: minCluster = 4
         maxCluster = 20
         wcss = []

         for nClusters in range(minCluster, maxCluster):

             # ### Train kmeans model
             kmeanModel = KMeans(n_clusters = nClusters, random_state = 424)
             kmeanModel.fit(xTrainScaledcaled)

             # ### Predict
             pred = kmeanModel.predict(xTrainScaledcaled)

             # ### Compute WCSS (Elbow)
             wcss.append(kmeanModel.inertia_)

             # ### Compute silhouette score
```

```
    score = silhouette_score(xTrainScaledcaled, pred)

    print(f'Silhouette Score for {nClusters} clusters: {score:.5f}')
```

```
Silhouette Score for 4 clusters: 0.44628
Silhouette Score for 5 clusters: 0.46062
Silhouette Score for 6 clusters: 0.46468
Silhouette Score for 7 clusters: 0.47107
Silhouette Score for 8 clusters: 0.48019
Silhouette Score for 9 clusters: 0.48148
Silhouette Score for 10 clusters: 0.48342
Silhouette Score for 11 clusters: 0.47561
Silhouette Score for 12 clusters: 0.47724
Silhouette Score for 13 clusters: 0.47215
Silhouette Score for 14 clusters: 0.46838
Silhouette Score for 15 clusters: 0.46909
Silhouette Score for 16 clusters: 0.46981
Silhouette Score for 17 clusters: 0.46919
Silhouette Score for 18 clusters: 0.45965
Silhouette Score for 19 clusters: 0.47256
```

In [60]:
```
mycenters = pd.DataFrame({'Clusters' : range(minCluster, maxCluster), 'WCSS'
sns.lineplot(x = 'Clusters', y = 'WCSS', data = mycenters, marker="o", color
plt.title('The Elbow Method')
plt.show()
```



In [61]:
```
def optimalNumberClusters(wcss):
    x1, y1 = 4, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]
```

```
    distances = []
    for i in range(len(wcss)):
        x0 = i+4
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = math.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 4

print(f'Melhor número de clusters de acordo com o método Elbow: {optimalNumb
```

```
Melhor número de clusters de acordo com o método Elbow: 9
```

3.2.0 - Com base no Elbow

In [62]:
```
best_k = 9

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0    1  Ganho  Perda  Lucro
labels
1            764  153  13770   7640   6130
6            266   25   2250   2660   -410
8            196    0      0   1960  -1960
```

3.2.1 - Com base na Silhueta Score

In [63]:
```
best_k = 10

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])
```

```
data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0    1  Ganho  Perda  Lucro
labels
9            765  152  13680   7650   6030
7            268   25   2250   2680   -430
8             48    0      0    480   -480
```

3.3 - Testar usando a normalização MinMaxScaler e removendo as variáveis com menos de 10 valores distitos

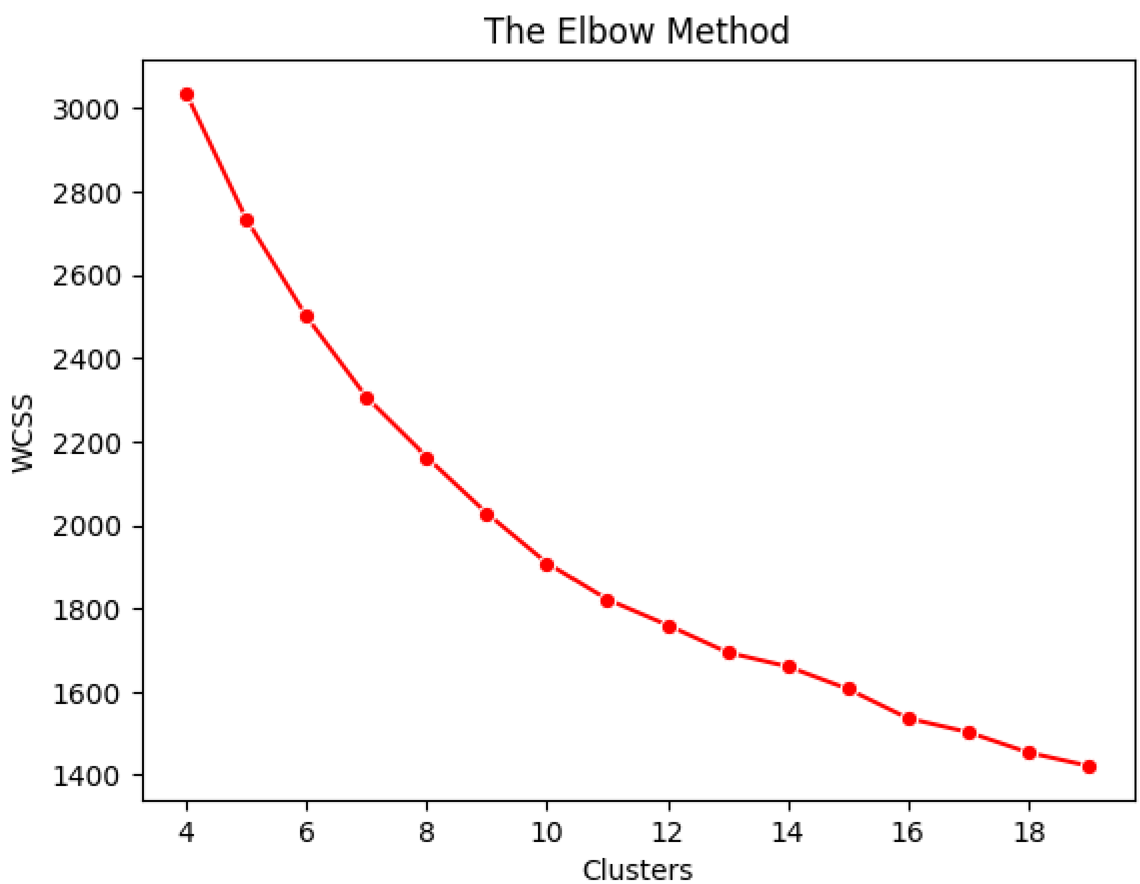
In [31]:
```
dfTrainDrop = dfTrain.drop(labels=['ind_var1_0', 'ind_var25_cte', 'num_var4'
dfValDrop = dfVal.drop(labels=['ind_var1_0', 'ind_var25_cte', 'num_var4', 'r
```

In [35]:
```
# ### Scaler feature for clustering
scaler = MinMaxScaler()
scaler.fit(dfTrainDrop)

# ### Apply in train and val
xTrainScaledcaled = scaler.transform(dfTrainDrop)
xValScaled = scaler.transform(dfValDrop)
```

In [36]:
```
minCluster = 4
maxCluster = 20
wcss = []

for nClusters in range(minCluster, maxCluster):

    # ### Train kmeans model
    kmeanModel = KMeans(n_clusters = nClusters, random_state = 424)
    kmeanModel.fit(xTrainScaledcaled)

    # ### Predict
    pred = kmeanModel.predict(xTrainScaledcaled)

    # ### Compute WCSS (Elbow)
    wcss.append(kmeanModel.inertia_)

    # ### Compute silhouette score
    score = silhouette_score(xTrainScaledcaled, pred)

    print(f'Silhouette Score for {nClusters} clusters: {score:.5f}')
```

```
Silhouette Score for 4 clusters: 0.51940
Silhouette Score for 5 clusters: 0.51334
Silhouette Score for 6 clusters: 0.52355
Silhouette Score for 7 clusters: 0.45830
Silhouette Score for 8 clusters: 0.45901
Silhouette Score for 9 clusters: 0.46099
Silhouette Score for 10 clusters: 0.47009
Silhouette Score for 11 clusters: 0.47650
Silhouette Score for 12 clusters: 0.42342
Silhouette Score for 13 clusters: 0.47931
Silhouette Score for 14 clusters: 0.42142
Silhouette Score for 15 clusters: 0.34102
Silhouette Score for 16 clusters: 0.42897
Silhouette Score for 17 clusters: 0.41503
Silhouette Score for 18 clusters: 0.41653
Silhouette Score for 19 clusters: 0.41708
```

In [37]:
```
mycenters = pd.DataFrame({'Clusters' : range(minCluster, maxCluster), 'WCSS'
sns.lineplot(x = 'Clusters', y = 'WCSS', data = mycenters, marker="o", color
plt.title('The Elbow Method')
plt.show()
```



In [38]:
```
def optimalNumberClusters(wcss):
    x1, y1 = 4, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]

    distances = []
    for i in range(len(wcss)):
```

```
        x0 = i+4
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = math.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 4

print(f'Melhor número de clusters de acordo com o método Elbow: {optimalNumb
```

```
Melhor número de clusters de acordo com o método Elbow: 10
```

3.3.0 - Com base no Elbow

In [39]:
```
best_k = 10

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0   1  Ganho  Perda  Lucro
labels
9             47   0      0    470   -470
8            191  11    990   1910   -920
3            112   1     90   1120  -1030
```

3.3.1 - Com base na Silhueta Score

In [40]:
```
best_k = 6

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
```

```
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0   1  Ganho  Perda  Lucro
labels
2            205   0      0   2050  -2050
1            227   2    180   2270  -2090
5            714  51   4590   7140  -2550
```

3.4 - Testar usando a normalização StandardScaler e todas as features

In [41]:

```
# ### Scaler feature for clustering
scaler = StandardScaler()
scaler.fit(dfTrain)

# ### Apply in train and val
xTrainScaledcaled = scaler.transform(dfTrain)
xValScaled = scaler.transform(dfVal)
```

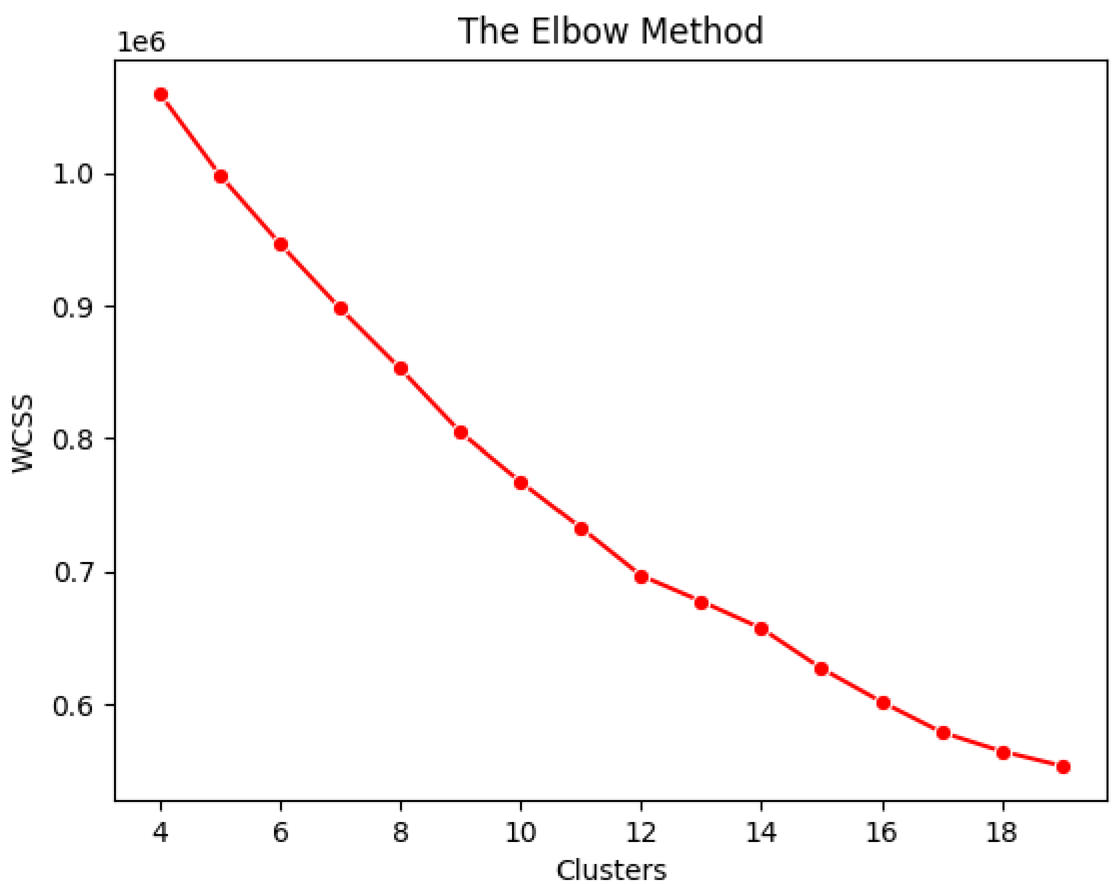
In [44]:

```
minCluster = 4
maxCluster = 20
wcss = []

for nClusters in range(minCluster, maxCluster):

    # ### Train Kmeans
    kmeanModel = KMeans(n_clusters = nClusters, random_state = 424)
    kmeanModel.fit(xTrainScaledcaled)

    # ### Predict
    pred = kmeanModel.predict(xTrainScaledcaled)

    # ### Compute WCSS (Elbow)
    wcss.append(kmeanModel.inertia_)

    # ### Compute silhouette score
    score = silhouette_score(xTrainScaledcaled, pred)

    print(f'Silhouette Score for {nClusters} clusters: {score:.5f}')
```

```
Silhouette Score for 4 clusters: 0.56280
Silhouette Score for 5 clusters: 0.44995
Silhouette Score for 6 clusters: 0.29691
Silhouette Score for 7 clusters: 0.30027
Silhouette Score for 8 clusters: 0.31074
Silhouette Score for 9 clusters: 0.31307
Silhouette Score for 10 clusters: 0.31758
Silhouette Score for 11 clusters: 0.32416
Silhouette Score for 12 clusters: 0.33168
Silhouette Score for 13 clusters: 0.33153
Silhouette Score for 14 clusters: 0.33531
Silhouette Score for 15 clusters: 0.34208
Silhouette Score for 16 clusters: 0.34364
Silhouette Score for 17 clusters: 0.35333
Silhouette Score for 18 clusters: 0.36269
Silhouette Score for 19 clusters: 0.35715
```

In [45]:

```
mycenters = pd.DataFrame({'Clusters' : range(minCluster, maxCluster), 'WCSS'
sns.lineplot(x = 'Clusters', y = 'WCSS', data = mycenters, marker="o", color
plt.title('The Elbow Method')
plt.show()
```



In [46]:

```
def optimalNumberClusters(wcss):
    x1, y1 = 4, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]

    distances = []
    for i in range(len(wcss)):
```

```
        x0 = i+4
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = math.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 4

print(f'Melhor número de clusters de acordo com o método Elbow: {optimalNumb
```

```
Melhor número de clusters de acordo com o método Elbow: 12
```

3.4.0 - Com base no Elbow

In [47]:

```
best_k = 12

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target   0  1  Ganho  Perda  Lucro
labels
9            14  1     90    140    -50
3            29  0      0    290   -290
8            39  1     90    390   -300
```

3.4.1 - Com base na Silhueta Score

In [48]:

```
best_k = 4

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
```

```
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target     0   1  Ganho  Perda  Lucro
labels
3              86   0      0    860   -860
2             731  53   4770   7310  -2540
1            1101  12   1080  11010  -9930
```

3.5 - Testar usando a normalização yeo-johnson e todas as features

In [49]:

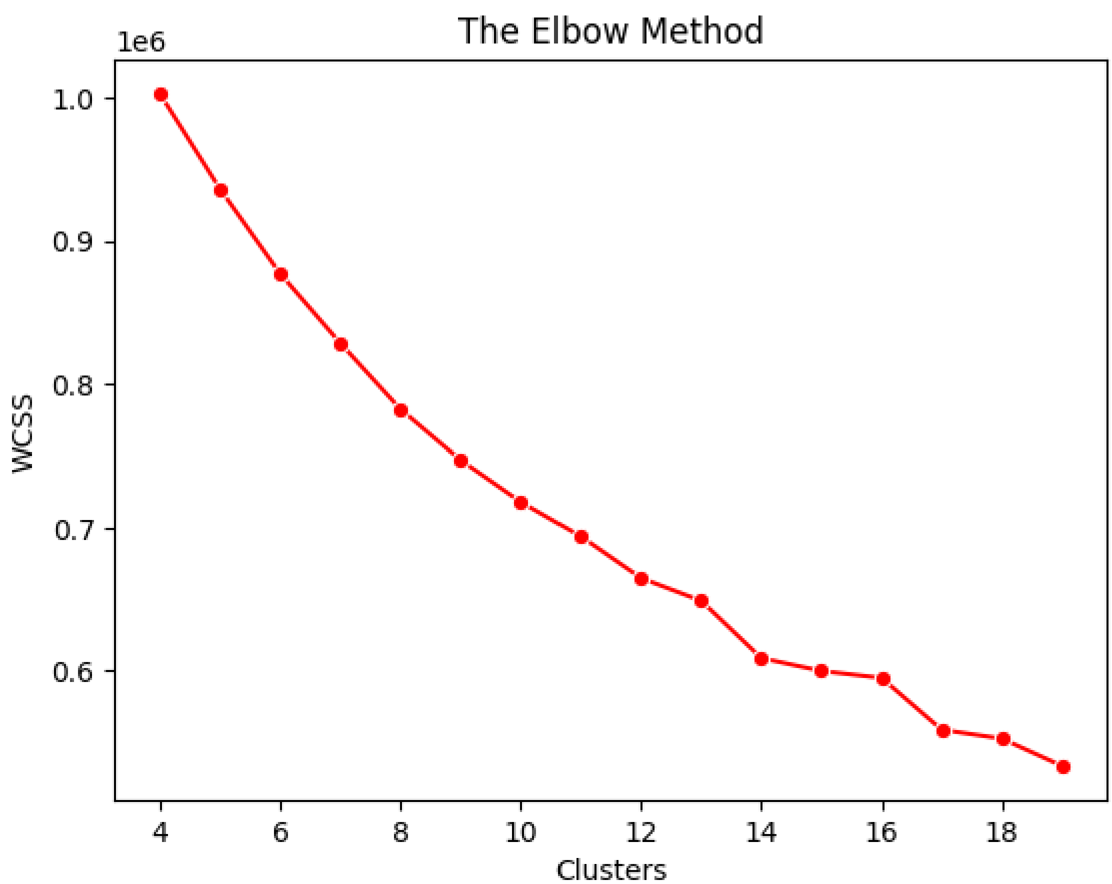
```
# ### Scaler feature for clustering
scaler = PowerTransformer(method='yeo-johnson')
scaler.fit(dfTrain)

# ### Apply in train and val
xTrainScaledcaled = scaler.transform(dfTrain)
xValScaled = scaler.transform(dfVal)
```

In [50]:

```
minCluster = 4
maxCluster = 20
wcss = []

for nClusters in range(minCluster, maxCluster):

    # ### Train kmeans model
    kmeanModel = KMeans(n_clusters = nClusters, random_state = 424)
    kmeanModel.fit(xTrainScaledcaled)

    # ### Predict
    pred = kmeanModel.predict(xTrainScaledcaled)

    # ### Compute WCSS (Elbow)
    wcss.append(kmeanModel.inertia_)

    # ### Compute silhouette score
    score = silhouette_score(xTrainScaledcaled, pred)

    print(f'Silhouette Score for {nClusters} clusters: {score:.5f}')
```

```
Silhouette Score for 4 clusters: 0.39075
Silhouette Score for 5 clusters: 0.21144
Silhouette Score for 6 clusters: 0.22606
Silhouette Score for 7 clusters: 0.24022
Silhouette Score for 8 clusters: 0.24570
Silhouette Score for 9 clusters: 0.26244
Silhouette Score for 10 clusters: 0.25647
Silhouette Score for 11 clusters: 0.24959
Silhouette Score for 12 clusters: 0.27540
Silhouette Score for 13 clusters: 0.25305
Silhouette Score for 14 clusters: 0.28694
Silhouette Score for 15 clusters: 0.20484
Silhouette Score for 16 clusters: 0.26093
Silhouette Score for 17 clusters: 0.26627
Silhouette Score for 18 clusters: 0.21308
Silhouette Score for 19 clusters: 0.26636
```

In [51]:
```
mycenters = pd.DataFrame({'Clusters' : range(minCluster, maxCluster), 'WCSS'
sns.lineplot(x = 'Clusters', y = 'WCSS', data = mycenters, marker="o", color
plt.title('The Elbow Method')
plt.show()
```



In [52]:
```
def optimalNumberClusters(wcss):
    x1, y1 = 4, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]

    distances = []
    for i in range(len(wcss)):
```

```
        x0 = i+4
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = math.sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 4

print(f'Melhor número de clusters de acordo com o método Elbow: {optimalNumb
```

```
Melhor número de clusters de acordo com o método Elbow: 9
```

3.5.0 - Com base no Elbow

In [53]:

```
best_k = 9

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target    0   1  Ganho  Perda  Lucro
labels
3             93   7    630    930   -300
6            276  25   2250   2760   -510
5            357   2    180   3570  -3390
```

3.5.1 - Com base na Silhueta Score

In [54]:

```
best_k = 4

model = KMeans(n_clusters = best_k, random_state = 424)
model.fit(xTrainScaledcaled)

cluster = model.predict(xValScaled)

df_cluster = pd.DataFrame({'cluster': cluster}).join(yVal)
df_cluster.columns = ['labels','true_target']

data = pd.crosstab(df_cluster['labels'],df_cluster['true_target'])

data ['Ganho'] = data [1]*90
data ['Perda'] = data[0]*10
```

```python
data ['Lucro'] = data [1]*90 - data[0]*10
print('top 3 clusters para o kmeans')
print(data.sort_values(by='Lucro', ascending=False).head(3))
```

```
top 3 clusters para o kmeans
true_target     0   1  Ganho  Perda  Lucro
labels
1             355  31   2790   3550   -760
0            1430  10    900  14300 -13400
3            2258  71   6390  22580 -16190
```

O melhor resultado encontrado foi usando a normalização MinMaxScaler e removendo as variáveis binárias, usando o K com base na silhuete score (3.2.1). Onde encontramos um cluster com um bom lucro e os demais com pouca perda. Sendo os labels 7,8 e 9 os principais clusters.Para visualizar toda a solução basta acessar: https://github.com/vitorgodeiro/DataMaster