Criar um modelo para classificar clientes do Banco União, a partir das variáveis de entrada, classificando-os como possíveis bons ou maus pagadores, a fim de uma aprovação ou não para pedidos de empréstimo.

In [1]:
```
# Findspark
import findspark
findspark.init()
```

In [2]:
```
# Imports
import pyspark
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.ml.linalg import Vectors
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import PCA
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

Carga dos Dados

Carregaremos os dados em um RDD e removeremos o cabeçalho.

In [3]:
```
# Spark Context
sc = SparkContext(appName = "EmprestimoBancario")
sc.setLogLevel("ERROR")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/01/30 14:31:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
```

In [4]:
```
# Spark Session
spSession = SparkSession.builder.master("local").getOrCreate()
```

In [5]:
```
# Carga dos dados em RDD
bankRDD = sc.textFile("dados/dataset3.csv")
bankRDD.cache()
```

Out[5]:
```
dados/dataset3.csv MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0
```

In [6]:
```
bankRDD.count()
```

Out[6]:
```
542
```

In [7]:
```
bankRDD.take(5)
```

Out[7]:
```
['"age";"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"durati
on";"campaign";"pdays";"previous";"poutcome";"y"',
 '30;"unemployed";"married";"primary";"no";1787;"no";"no";"cellular";19;"oct";79;1;-1;0;"unknown";"no"',
 '33;"services";"married";"secondary";"no";4789;"yes";"yes";"cellular";11;"may";220;1;339;4;"failure";"y
es"',
 '35;"management";"single";"tertiary";"no";1350;"yes";"no";"cellular";16;"apr";185;1;330;1;"failure";"ye
s"',
 '30;"management";"married";"tertiary";"no";1476;"yes";"yes";"unknown";3;"jun";199;4;-1;0;"unknown";"ye
s"']
```

In [8]:
```
# Remove cabeçalho
firstline = bankRDD.first()
bankRDD2 = bankRDD.filter(lambda x: x != firstline)
bankRDD2.count()
```

Out[8]:
```
541
```

Limpeza e Transformação

Vamos criar uma função para dividir as colunas, converter para float para aumentar a precisão dos cálculos, realizar o encoding com one-hot para "estatos civil e escolaridade", e label encoding para as demais variáveis categóricas, e então converter para row com os atributos transformados. Então aplicaremos a função ao conjunto de dados.

In [9]:
```
# Encoding
def transformToNumeric(inputStr) :

    #substituição do enter e split
    attList = inputStr.replace("\"","").split(";")

    # Int. para Float
    age = float(attList[0])
    balance = float(attList[5])

    # One-Hot Enconding com var. dummy
    single = 1.0 if attList[2] == 'single' else 0.0
    married = 1.0 if attList[2] == 'married' else 0.0
    divorced = 1.0 if attList[2] == 'divorced' else 0.0
    primary = 1.0 if attList[3] == 'primary' else 0.0
    secondary = 1.0 if attList[3] == 'secondary' else 0.0
    tertiary = 1.0 if attList[3] == 'tertiary' else 0.0

    # Label Encoding
    default = 0.0 if attList[4] == "no" else 1.0
    loan = 0.0 if attList[7] == "no" else 1.0
    outcome = 0.0 if attList[16] == "no" else 1.0

    # Linhas com os att. transformados
    linhas = Row(OUTCOME = outcome,
                 AGE = age,
                 SINGLE = single,
                 MARRIED = married,
                 DIVORCED = divorced,
                 PRIMARY = primary,
                 SECONDARY = secondary,
                 TERTIARY = tertiary,
                 DEFAULT = default,
                 BALANCE = balance,
                 LOAN = loan)
    return linhas
```

In [10]:
```
# Aplicando a função aos dados
bankRDD3 = bankRDD2.map(transformToNumeric)
```

In [11]:
```
bankRDD3.collect()[:5]
```

Out[11]:
```
[Row(OUTCOME=0.0, AGE=30.0, SINGLE=0.0, MARRIED=1.0, DIVORCED=0.0, PRIMARY=1.0, SECONDARY=0.0, TERTIARY=
0.0, DEFAULT=0.0, BALANCE=1787.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=33.0, SINGLE=0.0, MARRIED=1.0, DIVORCED=0.0, PRIMARY=0.0, SECONDARY=1.0, TERTIARY=
0.0, DEFAULT=0.0, BALANCE=4789.0, LOAN=1.0),
 Row(OUTCOME=1.0, AGE=35.0, SINGLE=1.0, MARRIED=0.0, DIVORCED=0.0, PRIMARY=0.0, SECONDARY=0.0, TERTIARY=
1.0, DEFAULT=0.0, BALANCE=1350.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=30.0, SINGLE=0.0, MARRIED=1.0, DIVORCED=0.0, PRIMARY=0.0, SECONDARY=0.0, TERTIARY=
1.0, DEFAULT=0.0, BALANCE=1476.0, LOAN=1.0),
 Row(OUTCOME=0.0, AGE=59.0, SINGLE=0.0, MARRIED=1.0, DIVORCED=0.0, PRIMARY=0.0, SECONDARY=1.0, TERTIARY=
0.0, DEFAULT=0.0, BALANCE=0.0, LOAN=0.0)]
```

Análise Exploratória

```
    Vamos verificar a correlação da variável alvo com as variáveis de entrada buscando a possível
relação entre tais, para a seleção das variáveis de entrada do modelo.
```

In [12]:
```
# Transforma em DF
bankDF = spSession.createDataFrame(bankRDD3)
```

In [13]:
```
# Correlação da variável OUTCOME as demais
for i in bankDF.columns:
    if not ( isinstance(bankDF.select(i).take(1)[0][0], str)) :
        print("Correlação da variável OUTCOME com:", i, bankDF.stat.corr('OUTCOME',i))
```

```
Correlação da variável OUTCOME com: OUTCOME 1.0
Correlação da variável OUTCOME com: AGE -0.18232104327365253
Correlação da variável OUTCOME com: SINGLE 0.46323284934360515
Correlação da variável OUTCOME com: MARRIED -0.3753241299133561
Correlação da variável OUTCOME com: DIVORCED -0.07812659940926987
Correlação da variável OUTCOME com: PRIMARY -0.12561548832677985
Correlação da variável OUTCOME com: SECONDARY 0.026392774894072976
Correlação da variável OUTCOME com: TERTIARY 0.08494840766635618
Correlação da variável OUTCOME com: DEFAULT -0.04536965206737378
Correlação da variável OUTCOME com: BALANCE 0.03657486611997681
Correlação da variável OUTCOME com: LOAN -0.03042058611271732
```

Pré-Processamento dos Dados

Foi recomendado seguir com todas as variáveis de entrada para a criação do modelo, logo será criada e aplicada uma função que irá pegar os dados e criar um RDD com um vetor denso composto por a variável alvo e as variáveis preditoras.

In [14]:
```
# Criando LabelPoint
def transformaVar(row):
    obj = (row["OUTCOME"], Vectors.dense([row["AGE"],
                                          row["BALANCE"],
                                          row["DEFAULT"],
                                          row["DIVORCED"],
                                          row["LOAN"],
                                          row["MARRIED"],
                                          row["PRIMARY"],
                                          row["SECONDARY"],
                                          row["SINGLE"],
                                          row["TERTIARY"]]))
    return obj
```

In [15]:
```
# Aplica função
bankRDD4 = bankDF.rdd.map(transformaVar)
```

In [16]:
```
bankRDD4.collect()
```

```
(1.0, DenseVector([67.0, 696.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0])),
(0.0, DenseVector([56.0, 784.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(0.0, DenseVector([53.0, 105.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(1.0, DenseVector([68.0, 4189.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0])),
(0.0, DenseVector([31.0, 171.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(0.0, DenseVector([59.0, 42.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(1.0, DenseVector([32.0, 2536.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(1.0, DenseVector([49.0, 1235.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])),
(0.0, DenseVector([42.0, 1811.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0])),
(1.0, DenseVector([78.0, 229.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0])),
(1.0, DenseVector([32.0, 2089.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(1.0, DenseVector([33.0, 3935.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(1.0, DenseVector([23.0, 363.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(1.0, DenseVector([38.0, 11971.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(0.0, DenseVector([36.0, 553.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(0.0, DenseVector([52.0, 1117.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0])),
(0.0, DenseVector([32.0, 396.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])),
(1.0, DenseVector([32.0, 2204.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(1.0, DenseVector([34.0, 872.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0])),
(1.0, DenseVector([55.0, 145.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0])),
```

In [17]:
```
# Converte RDD em DF
bankDF = spSession.createDataFrame(bankRDD4, ["label", "features"])
```

In [18]:
```
bankDF.select('features', 'label').show(5)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[30.0,1787.0,0.0,...|  0.0|
|[33.0,4789.0,0.0,...|  1.0|
|[35.0,1350.0,0.0,...|  1.0|
|[30.0,1476.0,0.0,...|  1.0|
|[59.0,0.0,0.0,0.0...|  0.0|
+--------------------+-----+
only showing top 5 rows
```

Redução de dimensionalidade

Como são muitas colunas de variáveis preditoras, vamos reduzir a dimensionalidade com o PCA. O objetivo com o PCA é simplificar conjuntos de dados complexos, preservando suas propriedades essenciais, ou seja, alteramos os dados para que fique melhor para a interpretação do algorítmo, mas mantendo a informação intacta. E então indexar a variável alvo.

In [19]:
```
# Cria PCA de 3 componentes
bankPCA = PCA(k = 3, inputCol = "features", outputCol = "pcaFeatures")
```

In [20]:
```
#Treina
pcaModel = bankPCA.fit(bankDF)
```

In [21]:
```
# Redução
pcaResult = pcaModel.transform(bankDF).select("label", "pcaFeatures")
```

In [22]:
```
pcaResult.show(5, truncate = False)
```

```
+-----+-----------------------------------------------------------+
|label|pcaFeatures                                                 |
+-----+-----------------------------------------------------------+
|0.0  |[-1787.018897197381,28.86209683775529,-0.06459982604876241] |
|1.0  |[-4789.020177138492,29.922562636341947,-0.9830243513096373] |
|1.0  |[-1350.022213163262,34.10110809796688,0.8951427168301704]   |
|1.0  |[-1476.0189517184556,29.051333993596703,0.3952723868021948] |
|0.0  |[-0.037889185366442445,58.9897182000177,-0.7290792383661886]|
+-----+-----------------------------------------------------------+
only showing top 5 rows
```

In [23]:
```
# Indexação do label para Decision Trees
stringIndexer = StringIndexer(inputCol = "label", outputCol = "label_indexed")
si_model = stringIndexer.fit(pcaResult)
obj_final = si_model.transform(pcaResult)
obj_final.collect()
```

```
 Row(label=1.0, pcaFeatures=DenseVector([-307.0231, 35.7999, 0.5171]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-147.025, 38.9011, -0.807]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-221.0263, 40.8536, 0.5373]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([87.9724, 43.0627, -0.067]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-9374.0231, 32.9765, -0.9511]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-264.0276, 42.8248, -0.7937]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-1109.0229, 35.2849, 0.5045]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-502.0127, 19.6493, -0.4862]), label_indexed=1.0),
 Row(label=1.0, pcaFeatures=DenseVector([-360.0198, 30.767, -0.9214]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-194.0256, 39.8716, 0.4531]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-4073.0351, 53.3753, -0.8041]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-2317.0233, 35.4796, 0.8876]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([220.9839, 25.1235, 0.346]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-132.0199, 30.913, -0.837]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-0.0244, 37.9953, 0.2124]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-16.027, 41.9797, 0.8465]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-106.0282, 43.8966, -0.3893]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-93.0282, 43.9343, -0.7872]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-543.0166, 25.6512, 0.4756]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-5883.0251, 37.2181, 0.4488]), label_indexed=0.0),
```

Machine Learning

Vamos criar nossa primeira versão do modelo a partir do algorítimo Random Forest. Dividindo os dados em 75% para treinamento e 25% para testar o modelo. Criamos o objeto, treinamos e testamos, e então apresentaremos a acurácia e a matrix de confusão dos resultados.

In [24]:
```
# Split em treino e teste (75% para treino e 25% para teste)
(dados_treino, dados_teste) = obj_final.randomSplit([0.75, 0.25])
```

In [25]:
```
dados_treino.count()
```

Out[25]: 414

In [26]:
```
dados_teste.count()
```

Out[26]: 127

In [27]:
```
# Cria objeto
rfClassifer = RandomForestClassifier(labelCol = "label_indexed", featuresCol = "pcaFeatures")
```

In [28]:
```
# Treina objeto e cria modelo
modelo = rfClassifer.fit(dados_treino)
```

In [29]:
```
# Previsões com teste
predictions = modelo.transform(dados_teste)
```

In [30]:
```
predictions
```

Out[30]:
```
DataFrame[label: double, pcaFeatures: vector, label_indexed: double, rawPrediction: vector, probability:
vector, prediction: double]
```

In [31]:
```
predictions.select("label", "label_indexed", "pcaFeatures", "prediction").collect()
```

Out[31]:
```
[Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-8104.0336, 49.7873, -0.8708]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-7190.0255, 37.3733, 0.7344]), predictio
n=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-7082.0351, 52.4544, -0.0453]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-5996.0302, 45.1426, -0.8606]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-5426.0252, 37.5115, 0.456]), prediction
=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-4073.0351, 53.3753, -0.8041]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-3762.0275, 41.5791, 0.4933]), predictio
n=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-1831.0215, 32.8212, -0.8522]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-1787.0189, 28.8621, -0.0646]), predicti
on=0.0),
 Row(label=0.0, label_indexed=0.0, pcaFeatures=DenseVector([-1699.0234, 35.91, -0.1202]), prediction
=0.0),
```

In [32]:
```
# Avaliando acurácia
evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction",
                                              labelCol = "label_indexed",
                                              metricName = "accuracy")
```

In [33]:
```
evaluator.evaluate(predictions)
```

Out[33]:
```
0.6299212598425197
```

In [34]:
```
# Confusion Matrix
predictions.groupBy("label_indexed", "prediction").count().show()
```

```
+-------------+----------+-----+
|label_indexed|prediction|count|
+-------------+----------+-----+
|          1.0|       1.0|   18|
|          0.0|       1.0|   17|
|          1.0|       0.0|   30|
|          0.0|       0.0|   62|
+-------------+----------+-----+
```

O modelo BenchMark não apresentou um resultado tão satisfatório aos líderes da equipe, onde foi solicitado realizar ajustes buscando-o melhorar. Será criado então uma segunda versão modificando a parte de pré-processamento dos dados, alterando o encoding das variáveis categóricas e aumentando em 1 grupo a mais na redução da dimensionalidade dos dados com o PCA.

Modelo 2 - PCA k = 4, e utilizando somente Label Encoding

Para a segunda verão, trataremos os dados criando outra função para separar as colunas, converter os dados numericos para float e realizar o label encoding em todas as variáveis categóricas, e então retornar as linhas com o tratamento aplicado. E então aplicaremos a função no conjunto de dados.

In [35]:
```
# Dicionário para o encoding das variáveis.
marital_status_mapping = {"single": 0, "married": 1, "divorced": 2}
education_mapping = {"primary": 0, "secondary": 1, "tertiary": 2}
```

In [36]:
```
# Função para transformação dos dados
def transformToNumeric2(inputStr) :

    #substituição do enter e split
    attList = inputStr.replace("\"","").split(";")

    # Inteiro para Float
    age = float(attList[0])
    balance = float(attList[5])


    # Label Encoding
    marital_status = marital_status_mapping.get(attList[2], -1)
    default = 0.0 if attList[4] == "no" else 1.0
    education = education_mapping.get(attList[3], -1)
    loan = 1.0 if attList[7] == "yes" else 0.0
    outcome = 1.0 if attList[16] == "yes" else 0.0

    # Linhas com os att. transformados
    linhas = Row(
        OUTCOME=outcome,
        AGE=age,
        MARITAL_STATUS=marital_status,
        EDUCATION=education,
        DEFAULT=default,
        BALANCE=balance,
        LOAN=loan
    )
    return linhas
```

In [37]:
```
# Aplicando a função aos dados
bank2RDD = bankRDD2.map(transformToNumeric2)
```

In [38]:
```
bank2RDD.collect()[:10]
```

Out[38]:
```
[Row(OUTCOME=0.0, AGE=30.0, MARITAL_STATUS=1, EDUCATION=0, DEFAULT=0.0, BALANCE=1787.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=33.0, MARITAL_STATUS=1, EDUCATION=1, DEFAULT=0.0, BALANCE=4789.0, LOAN=1.0),
 Row(OUTCOME=1.0, AGE=35.0, MARITAL_STATUS=0, EDUCATION=2, DEFAULT=0.0, BALANCE=1350.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=30.0, MARITAL_STATUS=1, EDUCATION=2, DEFAULT=0.0, BALANCE=1476.0, LOAN=1.0),
 Row(OUTCOME=0.0, AGE=59.0, MARITAL_STATUS=1, EDUCATION=1, DEFAULT=0.0, BALANCE=0.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=35.0, MARITAL_STATUS=0, EDUCATION=2, DEFAULT=0.0, BALANCE=747.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=36.0, MARITAL_STATUS=1, EDUCATION=2, DEFAULT=0.0, BALANCE=307.0, LOAN=0.0),
 Row(OUTCOME=0.0, AGE=39.0, MARITAL_STATUS=1, EDUCATION=1, DEFAULT=0.0, BALANCE=147.0, LOAN=0.0),
 Row(OUTCOME=0.0, AGE=41.0, MARITAL_STATUS=1, EDUCATION=2, DEFAULT=0.0, BALANCE=221.0, LOAN=0.0),
 Row(OUTCOME=1.0, AGE=43.0, MARITAL_STATUS=1, EDUCATION=0, DEFAULT=0.0, BALANCE=-88.0, LOAN=1.0)]
```

In [39]:
```
# Transforma em DF
bank2DF = spSession.createDataFrame(bank2RDD)
```

Pré-Processamento dos Dados

Vamos criar e aplicar nos dados a função para criar o vetor denso das linhas, e então converter o RDD em data Frame. Então aplicaremos a redução de dimensionalidade com o PCA, porém agora PCA= 4, onde será criado 4 grupos e não 3 como no modelo anterior, isso possivelmente agrupara com mais fidedignidade os dados. E indexaremos a variável target.

In [40]:
```
# Criando LabelPoint
def transformaVar2(row):
    obj = (row["OUTCOME"], Vectors.dense([row["AGE"],
                                          row["BALANCE"],
                                          row["DEFAULT"],
                                          row["LOAN"],
                                          row["MARITAL_STATUS"],
                                          row["EDUCATION"]]))
    return obj
```

In [41]:
```
# Aplica função transformaVar
bank2RDD2 = bank2DF.rdd.map(transformaVar2)
```

In [42]:
```
bank2RDD2.collect()
```

```
 (0.0, DenseVector([52.0, 1117.0, 0.0, 0.0, 1.0, 1.0])),
 (0.0, DenseVector([32.0, 396.0, 0.0, 0.0, 1.0, 2.0])),
 (1.0, DenseVector([32.0, 2204.0, 0.0, 0.0, 0.0, 2.0])),
 (1.0, DenseVector([34.0, 872.0, 0.0, 0.0, 0.0, 2.0])),
 (1.0, DenseVector([55.0, 145.0, 0.0, 0.0, 1.0, 0.0])),
 (0.0, DenseVector([26.0, 0.0, 0.0, 0.0, 1.0, 0.0])),
 (1.0, DenseVector([32.0, -849.0, 1.0, 1.0, 0.0, 0.0])),
 (1.0, DenseVector([61.0, 4629.0, 0.0, 0.0, 1.0, -1.0])),
 (1.0, DenseVector([45.0, 844.0, 0.0, 0.0, 2.0, 0.0])),
 (1.0, DenseVector([37.0, 228.0, 0.0, 0.0, 0.0, 1.0])),
 (1.0, DenseVector([38.0, 50.0, 0.0, 0.0, 0.0, 1.0])),
 (1.0, DenseVector([34.0, 1539.0, 0.0, 0.0, 1.0, 2.0])),
 (0.0, DenseVector([53.0, 2231.0, 0.0, 0.0, 1.0, 1.0])),
 (0.0, DenseVector([48.0, 3064.0, 0.0, 0.0, 1.0, 1.0])),
 (0.0, DenseVector([57.0, 82.0, 0.0, 1.0, 1.0, 1.0])),
 (1.0, DenseVector([33.0, 2155.0, 0.0, 0.0, 0.0, 2.0])),
 (1.0, DenseVector([36.0, 101.0, 0.0, 1.0, 0.0, 1.0])),
 (0.0, DenseVector([54.0, 784.0, 0.0, 1.0, 2.0, 1.0])),
 (0.0, DenseVector([41.0, -516.0, 0.0, 1.0, 1.0, 0.0])),
```

In [43]:
```
# Converte RDD em DF
bank2DF = spSession.createDataFrame(bank2RDD2, ["label", "features"])
```

In [44]:
```
bank2DF.select('features', 'label').show(3)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[30.0,1787.0,0.0,...|  0.0|
|[33.0,4789.0,0.0,...|  1.0|
|[35.0,1350.0,0.0,...|  1.0|
+--------------------+-----+
only showing top 3 rows
```

Redução de dimensionalidade

In [45]:
```
# Cria PCA de 4 componentes
bank2PCA = PCA(k = 4, inputCol = "features", outputCol = "pcaFeatures")
```

In [46]:
```
#Treina
pcaModel2 = bank2PCA.fit(bank2DF)
```

In [47]:
```
# Redução
pcaResult2 = pcaModel2.transform(bank2DF).select("label", "pcaFeatures")
```

In [48]:
```
pcaResult2.show(5, truncate = False)
```

```
+-----+-----------------------------------------------------------------------------------+
|label|pcaFeatures                                                                         |
+-----+-----------------------------------------------------------------------------------+
|0.0  |[-1787.0189013456968,28.86465735936177,0.42720870292758745,0.3016100110286005]    |
|1.0  |[-4789.020200453407,29.921189981776095,1.368325854165402,0.19434042149159386]     |
|1.0  |[-1350.0222313116572,34.087439977346605,2.521215563988013,-0.8139875298027369]    |
|1.0  |[-1476.0189724478373,29.033089082983494,2.4036636564627702,0.2256740119710345]    |
|0.0  |[-0.03791261698648324,58.98382682694944,1.9329018663001152,-0.4130336855783044]   |
+-----+-----------------------------------------------------------------------------------+
only showing top 5 rows
```

In [49]:
```
# Indexação do label para Decision Trees
stringIndexer2 = StringIndexer(inputCol = "label", outputCol = "label_indexed")
si_model2 = stringIndexer2.fit(pcaResult2)
obj_final2 = si_model2.transform(pcaResult2)
obj_final2.collect()
```

Out[49]:
```
[Row(label=0.0, pcaFeatures=DenseVector([-1787.0189, 28.8647, 0.4272, 0.3016]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-4789.0202, 29.9212, 1.3683, 0.1943]), label_indexed=1.0),
 Row(label=1.0, pcaFeatures=DenseVector([-1350.0222, 34.0874, 2.5212, -0.814]), label_indexed=1.0),
 Row(label=1.0, pcaFeatures=DenseVector([-1476.019, 29.0331, 2.4037, 0.2257]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-0.0379, 58.9838, 1.9329, -0.413]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-747.0224, 34.4745, 2.5364, -0.8219]), label_indexed=1.0),
 Row(label=1.0, pcaFeatures=DenseVector([-307.0231, 35.7805, 2.5587, 0.1446]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-147.025, 38.8978, 1.6111, 0.0677]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-221.0263, 40.8337, 2.6404, 0.0238]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([87.9724, 43.0631, 0.6512, -0.1122]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-9374.0231, 32.9755, 1.378, 0.1889]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-264.0276, 42.821, 1.6718, -0.0265]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-1109.0229, 35.2658, 2.5385, 0.1552]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-502.0128, 19.6537, 1.3047, -0.4691]), label_indexed=1.0),
 Row(label=1.0, pcaFeatures=DenseVector([-360.0198, 30.7647, 1.4484, 0.184]), label_indexed=1.0),
 Row(label=0.0, pcaFeatures=DenseVector([-194.0257, 39.8518, 2.5951, -0.0306]), label_indexed=0.0),
 Row(label=0.0, pcaFeatures=DenseVector([-4073.0351, 53.3708, 1.7823, -0.2877]), label_indexed=0.0),
 Row(label=1.0, pcaFeatures=DenseVector([-2317.0233, 35.4659, 2.5286, -0.8492]), label_indexed=1.0),
```

Machine Learning

Aplicaremos o mesmo procedimento de criação do algorítmo anterior, porém com a nova formatação dos dados. Separaremos os dados em treino e teste, criaremos o modelo, treinaremos com dados de treino, testaremos com dados de teste, calcularemos a acurácia e a matrix de confusão.

In [50]:
```
# Split em treino e teste
(dados_treino, dados_teste) = obj_final2.randomSplit([0.75, 0.25])
```

In [51]:
```
dados_treino.count()
```

Out[51]: 388

In [52]:
```
dados_teste.count()
```

Out[52]: 153

In [53]:
```
# Cria objeto
rfClassifer = RandomForestClassifier(labelCol = "label_indexed", featuresCol = "pcaFeatures")
```

In [54]:
```
# Treina objeto e criar modelo
modelo2 = rfClassifer.fit(dados_treino)
```

In [55]:
```
# Previsões com teste
predictions2 = modelo2.transform(dados_teste)
```

In [56]:
```
predictions2
```

Out[56]:
```
DataFrame[label: double, pcaFeatures: vector, label_indexed: double, rawPrediction: vector, probability:
vector, prediction: double]
```

In [57]:
```
predictions2.select("label", "label_indexed", "pcaFeatures", "prediction").collect()
```

```
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-4590.0151, 22.0278, 1.281, -0.5351]), p
rediction=1.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-3935.0204, 30.469, 1.42, 0.2611]), pred
iction=0.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-3777.0358, 54.5521, 0.8111, -1.3149]),
prediction=1.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-2536.0201, 30.3275, 2.4435, -0.7266]),
prediction=1.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-2317.0233, 35.4659, 2.5286, -0.8492]),
prediction=1.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-1567.026, 39.9457, 2.6112, -0.9548]), p
rediction=0.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-1539.0216, 32.9906, 2.4958, 0.2087]), p
rediction=0.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-1350.0222, 34.0874, 2.5212, -0.814]), p
rediction=1.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-1235.0312, 48.1795, 2.742, -0.1544]), p
rediction=0.0),
 Row(label=1.0, label_indexed=1.0, pcaFeatures=DenseVector([-902.0242, 37.3895, 1.5809, -0.8948]), p
rediction=1.0),
```

In [58]:
```
# Avaliando acurácia
evaluator.evaluate(predictions2)
```

Out[58]:
```
0.7581699346405228
```

In [59]:
```
# Confusion Matrix
predictions2.groupBy("label_indexed", "prediction").count().show()
```

```
+-------------+----------+-----+
|label_indexed|prediction|count|
+-------------+----------+-----+
|          1.0|       1.0|   29|
|          0.0|       1.0|   14|
|          1.0|       0.0|   23|
|          0.0|       0.0|   87|
+-------------+----------+-----+
```

Realizando as modificações no tratamento e pré-processamento dos dados, conseguimos aumentar a acurácia do modelo. A partir do modelo final, quando o resultado era 1 nosso modelo previo 1 em 29 vezes, quando era 1 e nosso modelo previo 0 foram 23 vezes. Quando o resultado era 0 nosso modelo previu 0 87 vezes, quando era 0 e nosso modelo previo 1 foram 14 vezes. Aumentando a acurácia de 0.629 para 0.758.

Apesar de ser uma melhora considerável, ainda segue uma taxa de acertos fraca/mediana, tendo espaços para melhorias.Ficam aqui minhas recomendações aos líderes da equipe, para que continuamos este projeto testando novas versões, implementando outros algorítmos bem como ajustes de parâmetros, a fim de aumentar a probabilidade de acerto das previsões realizadas pelo modelo, com intúito de aumentar o êxito na tomada de decisão pretendida para este problema de negócio.