Prevendo Customer Churn em Operadoras de Telecom

In [1]:

```
#Biblioteca Python
import numpy as np
import pandas as pd
```

In [2]:

```
# Biblioteca Pyspark
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [3]:

```
#Sessão Spark
spSession = SparkSession.builder.master('local').appName('DSA-OperadorasTelecom').getOr
Create()
```

Transformação dos dados de treino

In [4]:

```
def Transformacao(rdd):
    header = rdd.first()
    rdd2 = rdd.filter(lambda x : x not in header)
    rddInter = rdd2.map(lambda x: x.replace('"yes"', '1').replace('"no"','0'))
    rdd3 = rddInter.map(lambda x : x.split(","))
    rdd4 = rdd3.map(lambda p : Row(state = p[1],
                                   account_length = int(p[2]),
                                   area_code = p[3],
                                   international_plan = int(p[4]),
                                   voice_mail_plan = int(p[5]),
                                   number_vmail_messages = int(p[6]),
                                   total_day_minutes = float(p[7]),
                                   total_day_calls = int(p[8]),
                                   total_day_charge = float(p[9]),
                                   total_eve_minutes = float(p[10]),
                                   total_eve_calls = int(p[11]),
                                   total_eve_charge = float(p[12]),
                                   total_night_minutes = float(p[13]),
                                   total_night_calls = int(p[14]),
                                   total_night_charge = float(p[15]),
                                   total_intl_minutes = float(p[16]),
                                   total_intl_calls = int(p[17]),
                                   total_intl_charge = float(p[18]),
                                   number_customer_service_calls = int(p[19]),
                                   churn = int(p[20]) ))
    dataDF = spSession.createDataFrame(rdd4)
    dataDF_s = indexString(dataDF)
    return dataDF_s
```

In [5]:

```
def indexString(dataDF):
    #String Index para indexar as variaveis que são string
    area_s = StringIndexer(inputCol='area_code', outputCol= 'area_code_string').fit(dat
aDF)
    #String Index para indexar as variaveis que são string
    state_s = StringIndexer(inputCol='state', outputCol= 'state_string').fit(dataDF)
    dataDF_s = area_s.transform(dataDF)
    dataDF_s = state_s.transform(dataDF_s)
    dataDF_s = dataDF_s.drop(*['area_code','state'])
    return dataDF_s
```

In [6]:

```
dataDF_s = Transformacao(sc.textFile('projeto4_telecom_treino.csv'))
```

In [7]:

```
dataDF_s.toPandas().describe()
```

Out[7]:

	account_length	churn	international_plan	number_customer_service_calls	numb
count	3333.000000	3333.000000	3333.000000	3333.000000	
mean	101.064806	0.144914	0.096910	1.562856	
std	39.822106	0.352067	0.295879	1.315491	
min	1.000000	0.000000	0.000000	0.000000	
25%	74.000000	0.000000	0.000000	1.000000	
50%	101.000000	0.000000	0.000000	1.000000	
75%	127.000000	0.000000	0.000000	2.000000	
max	243.000000	1.000000	1.000000	9.000000	

Analise exploratoria

In [8]:

```
# Correlação entre as variáveis
for i in dataDF_s.columns:
    if not(isinstance(dataDF_s.select(i).take(1)[0][0], str)) :
        print("Correlação da variável CHURN com", i, dataDF_s.stat.corr('churn', i))
```

```
Correlação da variável CHURN com account_length 0.016540742243674286
Correlação da variável CHURN com churn 1.0
Correlação da variável CHURN com international_plan 0.2598518473454819
Correlação da variável CHURN com number_customer_service_calls 0.208749998
78379408
Correlação da variável CHURN com number_vmail_messages -0.0897279698350641
8
Correlação da variável CHURN com total_day_calls 0.018459311608577066
Correlação da variável CHURN com total_day_charge 0.20515074317015397
Correlação da variável CHURN com total_day_minutes 0.2051508292613899
Correlação da variável CHURN com total_eve_calls 0.009233131913077921
Correlação da variável CHURN com total_eve_charge 0.09278603942871391
Correlação da variável CHURN com total_eve_minutes 0.09279579031259168
Correlação da variável CHURN com total_intl_calls -0.052844335774137816
Correlação da variável CHURN com total_intl_charge 0.06825863150391472
Correlação da variável CHURN com total_intl_minutes 0.06823877562717737
Correlação da variável CHURN com total_night_calls 0.006141203007399843
Correlação da variável CHURN com total_night_charge 0.0354955562405066
Correlação da variável CHURN com total_night_minutes 0.03549285342127406
Correlação da variável CHURN com voice_mail_plan -0.1021481406701469
Correlação da variável CHURN com area_code_string 0.004516661668833458
Correlação da variável CHURN com state_string -0.01471772533561556
```

In [9]:

```
#Correlaçao dos dados por ordem
lista =[]
for i in dataDF_s.columns:
    lista.append((i, dataDF_s.stat.corr('churn', i)))
sorted(lista, key=lambda p: abs(p[1]), reverse=True)
```

Out[9]:

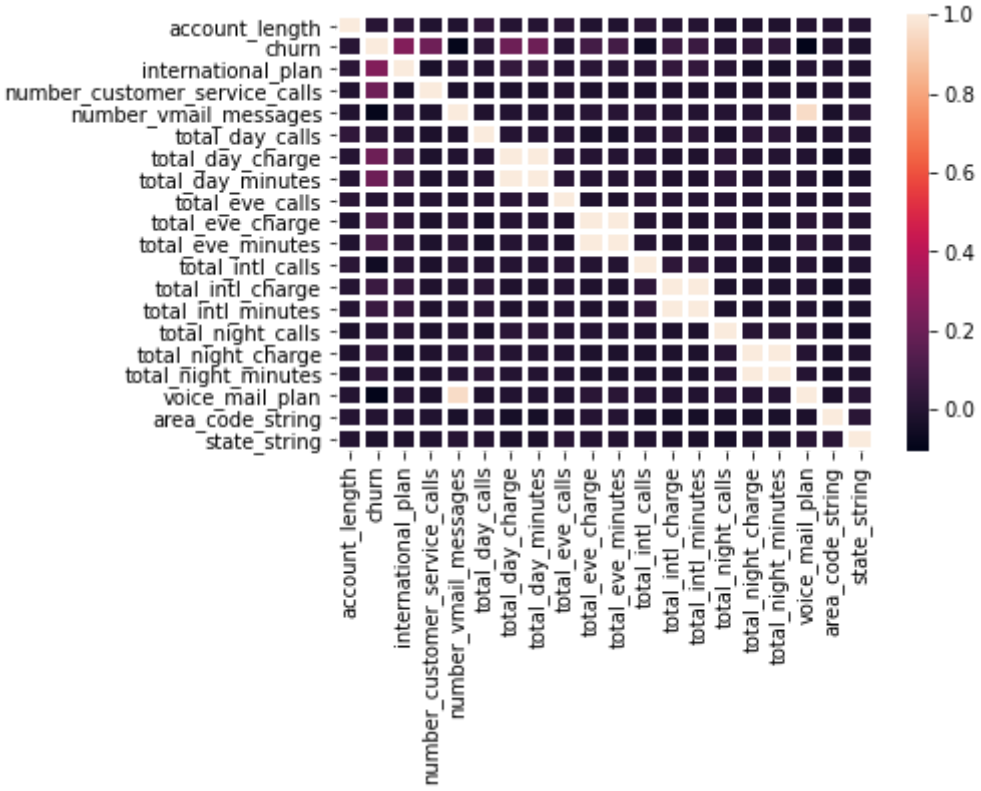
```
[('churn', 1.0),
 ('international_plan', 0.2598518473454819),
 ('number_customer_service_calls', 0.20874999878379408),
 ('total_day_minutes', 0.2051508292613899),
 ('total_day_charge', 0.20515074317015397),
 ('voice_mail_plan', -0.1021481406701469),
 ('total_eve_minutes', 0.09279579031259168),
 ('total_eve_charge', 0.09278603942871391),
 ('number_vmail_messages', -0.08972796983506418),
 ('total_intl_charge', 0.06825863150391472),
 ('total_intl_minutes', 0.06823877562717737),
 ('total_intl_calls', -0.052844335774137816),
 ('total_night_charge', 0.0354955562405066),
 ('total_night_minutes', 0.03549285342127406),
 ('total_day_calls', 0.018459311608577066),
 ('account_length', 0.016540742243674286),
 ('state_string', -0.01471772533561556),
 ('total_eve_calls', 0.009233131913077921),
 ('total_night_calls', 0.006141203007399843),
 ('area_code_string', 0.004516661668833458)]
```

In [10]:

```
import seaborn as sns
%matplotlib inline
sns.heatmap(dataDF_s.toPandas().corr(), linewidths=2)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2c32a2c05c0>
```



In [11]:

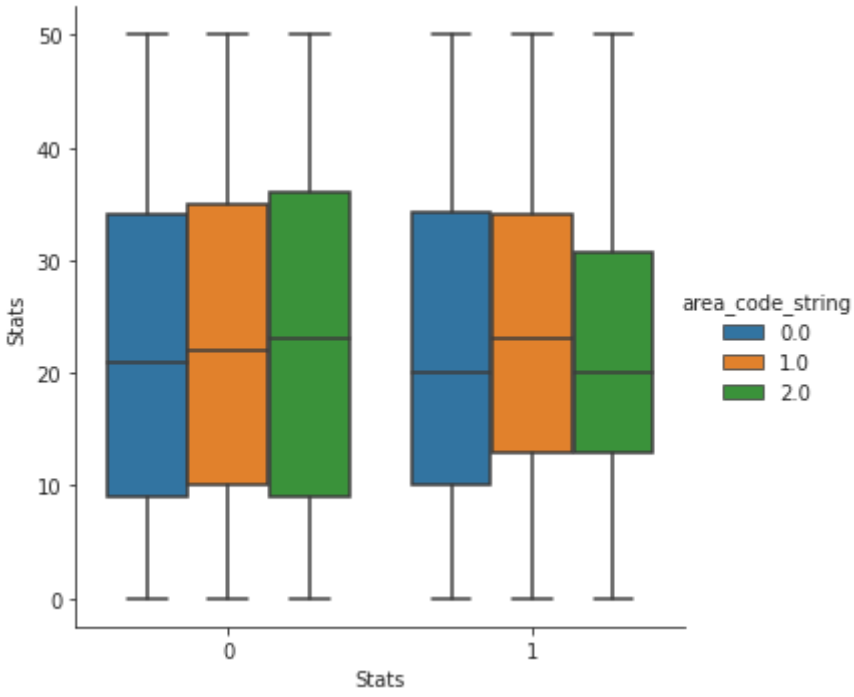
```
import matplotlib.pyplot as plt
```

Analise Grafica

In [12]:

```
with sns.axes_style(style='ticks'):
    g = sns.factorplot("churn", "state_string", "area_code_string",
                       data=dataDF_s.select(['churn','international_plan',
                                             'total_day_minutes','total_day_charge',
'state_string',
                                             'voice_mail_plan','number_vmail_message
s', 'area_code_string']).toPandas(),
                       kind="box")
    g.set_axis_labels("Stats", "Stats");
```

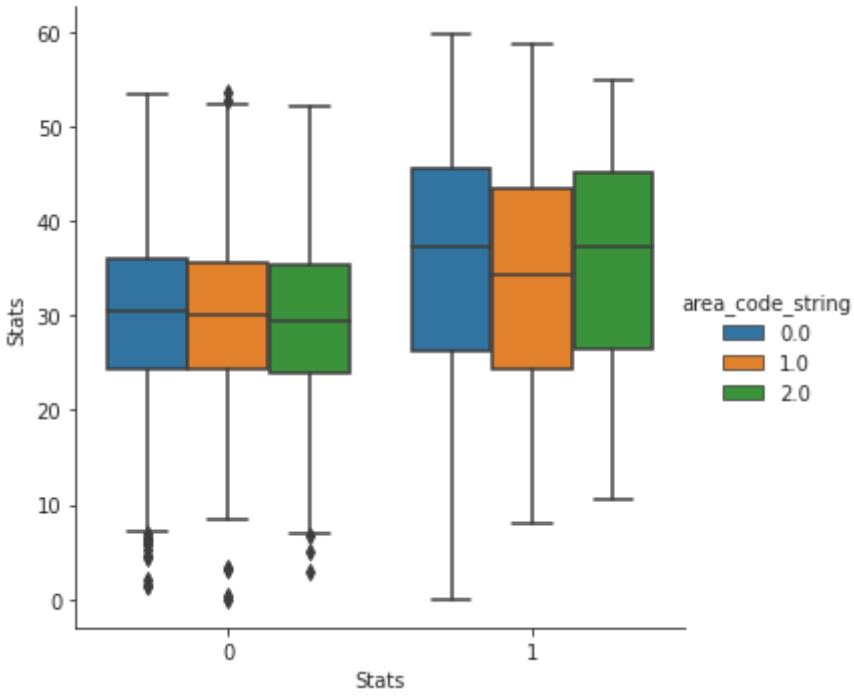
```
C:\Users\bruno\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: Us
erWarning: The `factorplot` function has been renamed to `catplot`. The or
iginal name will be removed in a future release. Please update your code.
Note that the default `kind` in `factorplot` (`'point'`) has changed `'str
ip'` in `catplot`.
  warnings.warn(msg)
```



In [13]:

```
with sns.axes_style(style='ticks'):
    g = sns.factorplot("churn", "total_day_charge", "area_code_string",
                       data=dataDF_s.select(['churn','international_plan',
                                             'total_day_minutes','total_day_charge',
'state_string',
                                             'voice_mail_plan','number_vmail_message
s', 'area_code_string']).toPandas(),
                       kind="box")
    g.set_axis_labels("Stats", "Stats");
```

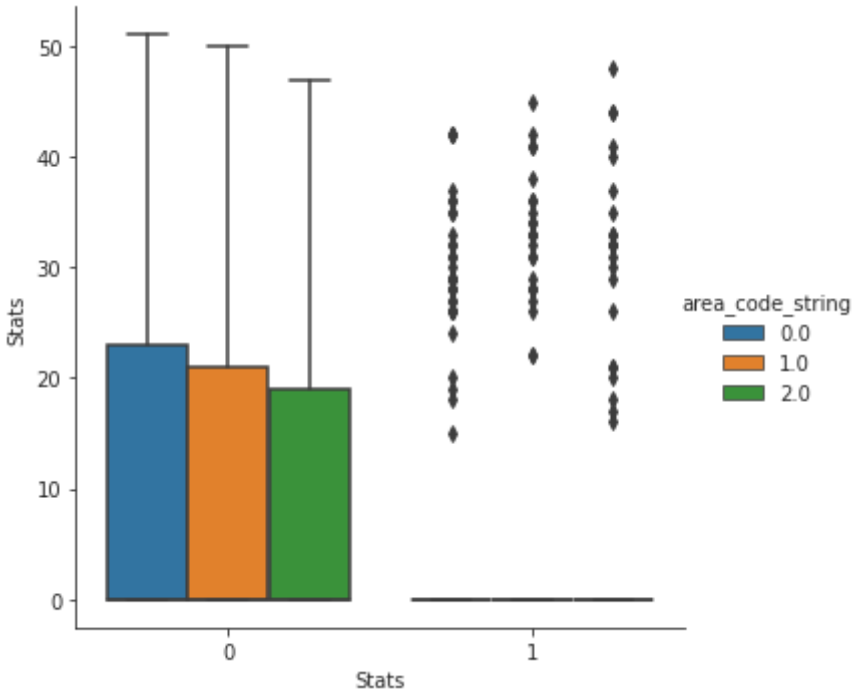
```
C:\Users\bruno\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: Us
erWarning: The `factorplot` function has been renamed to `catplot`. The or
iginal name will be removed in a future release. Please update your code.
Note that the default `kind` in `factorplot` (`'point'`) has changed `'str
ip'` in `catplot`.
  warnings.warn(msg)
```



In [14]:

```
with sns.axes_style(style='ticks'):
    g = sns.factorplot("churn", "number_vmail_messages", "area_code_string",
                       data=dataDF_s.select(['churn','international_plan',
                                             'total_day_minutes','total_day_charge',
'state_string',
                                             'voice_mail_plan','number_vmail_message
s', 'area_code_string']).toPandas(),
                       kind="box")
    g.set_axis_labels("Stats", "Stats");
```

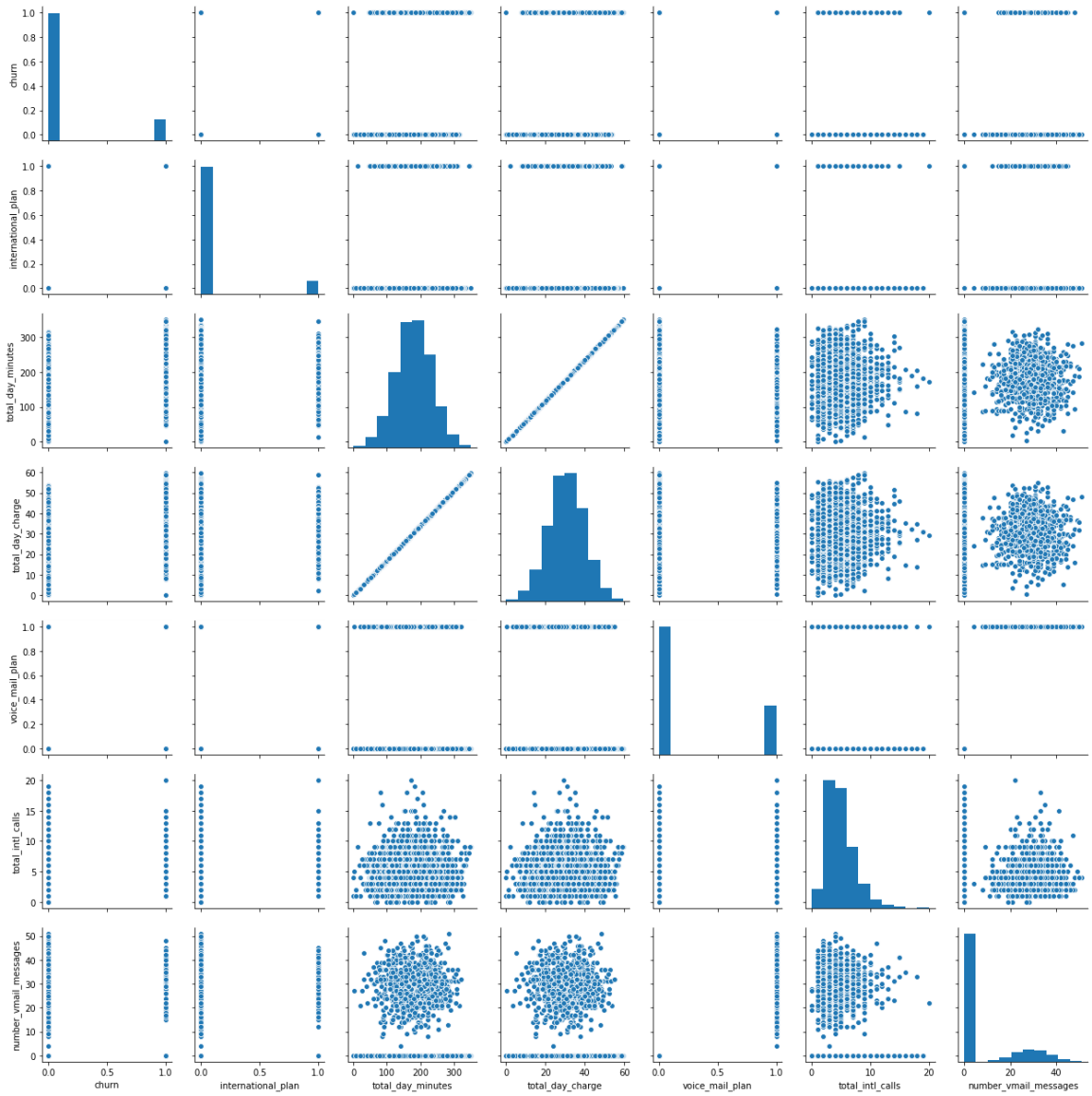
```
C:\Users\bruno\Anaconda3\lib\site-packages\seaborn\categorical.py:3666: Us
erWarning: The `factorplot` function has been renamed to `catplot`. The or
iginal name will be removed in a future release. Please update your code.
Note that the default `kind` in `factorplot` (`'point'`) has changed `'str
ip'` in `catplot`.
  warnings.warn(msg)
```



In [15]:

```
sns.pairplot(dataDF_s.select(['churn','international_plan',
                              'total_day_minutes','total_day_charge',
                              'voice_mail_plan',
                              'total_intl_calls','number_vmail_messages']).toPandas
(), size=2.5);
```

```
C:\Users\bruno\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: UserW
arning: The `size` parameter has been renamed to `height`; pleaes update y
our code.
  warnings.warn(msg, UserWarning)
```



In [16]:

```
dataDF_s.toPandas().head(3)
```

Out[16]:

	account_length	churn	international_plan	number_customer_service_calls	number_vmail_n
0	128	0	0	1	
1	107	0	0	1	
2	137	0	0	0	

In [17]:

```
type(dataDF_s)
```

Out[17]:

```
pyspark.sql.dataframe.DataFrame
```

Feature Selection

In [18]:

```
def featureSelection(quant, telecomDF_s):
        #Correlaçao dos dados por ordem
    lista =[]
    for i in telecomDF_s.columns:
        lista.append((i, telecomDF_s.stat.corr('churn', i)))
    lista = sorted(lista, key=lambda p: abs(p[1]), reverse=True)

    selection_feature = []
    for i in range(quant):
        selection_feature.append(lista[i][0])

    df = telecomDF_s.select(selection_feature)

    return selection_feature, df
```

In [19]:

```
selection_feature , df = featureSelection(7,dataDF_s )
selection_feature
```

Out[19]:

```
['churn',
 'international_plan',
 'number_customer_service_calls',
 'total_day_minutes',
 'total_day_charge',
 'voice_mail_plan',
 'total_eve_minutes']
```

In [20]:

```
type(df), type(selection_feature)
```

Out[20]:

```
(pyspark.sql.dataframe.DataFrame, list)
```

Vetorizando

In [21]:

```
from pyspark.ml.feature import VectorAssembler
```

In [22]:

```
def Vetoriza(df):
    assemble = VectorAssembler().setInputCols(df.columns[1:len(df.columns)]).setOutputC
ol("features")
    transformed = assemble.transform(df)
    return transformed
```

In [23]:

```
transformed = Vetoriza(df)
type(transformed)
```

Out[23]:

```
pyspark.sql.dataframe.DataFrame
```

Normalizando

In [24]:

```
from pyspark.ml.feature import MinMaxScaler

def Normaliza(transformed):
    scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
    scalerModel =  scaler.fit(transformed.select("features"))
    scaledData = scalerModel.transform(transformed)
    scaledDF = scaledData.select("churn", "scaledFeatures")
    return scaledDF
```

In [25]:

```
scaledDF = Normaliza(transformed)
type(scaledDF)
```

Out[25]:

```
pyspark.sql.dataframe.DataFrame
```

In [26]:

```
scaledDF.show(5)
```

```
+-----+--------------------+
|churn|      scaledFeatures|
+-----+--------------------+
|    0|[0.0,0.1111111111...|
|    0|[0.0,0.1111111111...|
|    0|[0.0,0.0,0.693842...|
|    0|[1.0,0.2222222222...|
|    0|[1.0,0.3333333333...|
+-----+--------------------+
only showing top 5 rows
```

Machine Learning

Treino

In [27]:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import LogisticRegression

logit_clf = LogisticRegression(labelCol = "churn", featuresCol = "scaledFeatures").fit(
scaledDF)
```

Teste

In [28]:

```
testeRDD = sc.textFile('projeto4_telecom_teste.csv')
```

In [29]:

```
df_teste = Transformacao(testeRDD)
feature, df_teste = featureSelection(7,df_teste)
```

In [30]:

```
transformed_teste = Vetoriza(df_teste)
scaledDF_teste = Normaliza(transformed_teste)
```

In [31]:

```
scaledDF_teste.show(5)
```

```
+-----+--------------------+
|churn|      scaledFeatures|
+-----+--------------------+
|    0|[0.0,0.4285714285...|
|    0|[0.0,0.0,0.629167...|
|    0|[0.0,0.1428571428...|
|    0|[0.0,0.1428571428...|
|    0|[0.0,0.2857142857...|
+-----+--------------------+
only showing top 5 rows
```

Predições

In [32]:

```
preds = logit_clf.transform(scaledDF_teste)
```

In [33]:

```
avaliador = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol =
"churn", metricName = "accuracy")
```

In [34]:

```
avaliador.evaluate(preds)
preds.select('churn', 'prediction','probability').toPandas().prediction.value_counts()
```

Out[34]:

```
0.0    1630
1.0      37
Name: prediction, dtype: int64
```

Prediction com diferentes variaveis - Resumo de todo o projeto em poucas linhas

In [35]:

```
def avaliacoes(quant, df_train_, df_test_):
    lista =[]
    for i in range(quant+1) :
        #Feature Selection
        feature_train, df_train = featureSelection(i+3,df_train_)
        feature_teste, df_test = featureSelection(i+3,df_test_)

        #Vetorização
        transformed_train = Vetoriza(df_train)
        transformed_test = Vetoriza(df_test)

        #Normalização
        scaledDF_train = Normaliza(transformed_train)
        scaledDF_test = Normaliza(transformed_test)

        #Modelo Preditivo e avaliação
        logit_clf = LogisticRegression(labelCol = "churn", featuresCol = "scaledFeature
s").fit(scaledDF_train)
        preds = logit_clf.transform(scaledDF_test)

        avaliador = MulticlassClassificationEvaluator(predictionCol = "prediction", lab
elCol = "churn", metricName = "accuracy")
        lista.append((avaliador.evaluate(preds),feature_train))

    return lista
```

In [36]:

```
df_train = Transformacao(sc.textFile('projeto4_telecom_treino.csv'))
df_teste = Transformacao(sc.textFile('projeto4_telecom_teste.csv'))
```

In [37]:

```
#A funcao retorna no minimo as 2 variaveis mais importantes, o valor enviado é a quanti
dade adicional
avalia = avaliacoes(5,df_train,df_teste)
avalia
```

Out[37]:

```
[(0.8662267546490702,
  ['churn', 'international_plan', 'number_customer_service_calls']),
 (0.8584283143371326,
  ['churn',
   'international_plan',
   'number_customer_service_calls',
   'total_day_minutes']),
 (0.8584283143371326,
  ['churn',
   'international_plan',
   'number_customer_service_calls',
   'total_day_minutes',
   'total_day_charge']),
 (0.8578284343131374,
  ['churn',
   'international_plan',
   'number_customer_service_calls',
   'total_day_minutes',
   'total_day_charge',
   'voice_mail_plan']),
 (0.8710257948410318,
  ['churn',
   'international_plan',
   'number_customer_service_calls',
   'total_day_minutes',
   'total_day_charge',
   'voice_mail_plan',
   'total_eve_minutes']),
 (0.7576484703059388,
  ['churn',
   'international_plan',
   'number_customer_service_calls',
   'total_day_minutes',
   'total_day_charge',
   'voice_mail_plan',
   'total_eve_minutes',
   'total_eve_charge'])]
```

No nosso modelo, utilizar 7 variaveis apresenta a melhor performance

Codigo utilizado durante o projeto

In [38]:

```
# Carregando os dados e gerando um RDD
    #telecomRDD = sc.textFile('projeto4_telecom_treino.csv')
#Salvando em cache para elevar a performance
    #telecomRDD.cache()
    #telecomRDD.count()
    #telecomRDD.take(3)
    #header = telecomRDD.first()
    #telecomRDD2 = telecomRDD.filter(lambda x : x not in header)
    #telecomRDD2.count()
    #telecomInter = telecomRDD2.map(lambda x: x.replace('"yes"', '1').replace('"n
o"','0'))
    #telecomRDD3 = telecomInter.map(lambda x : x.split(","))
    #telecomRDD3.take(1)
    #telecomRDD4 = telecomRDD3.map(lambda p : Row(state = p[1],
#                                                 account_length = int(p[2]),
#                                                 area_code = p[3],
#                                                 international_plan = int(p[4]),
#                                                 voice_mail_plan = int(p[5]),
#                                                 number_vmail_messages = int(p[6]),
#                                                 total_day_minutes = float(p[7]),
#                                                 total_day_calls = int(p[8]),
#                                                 total_day_charge = float(p[9]),
#                                                 total_eve_minutes = float(p[10]),
#                                                 total_eve_calls = int(p[11]),
#                                                 total_eve_charge = float(p[12]),
#                                                 total_night_minutes = float(p[13]),
#                                                 total_night_calls = int(p[14]),
#                                                 total_night_charge = float(p[15]),
#                                                 total_intl_minutes = float(p[16]),
#                                                 total_intl_calls = int(p[17]),
#                                                 total_intl_charge = float(p[18]),
#                                                 number_customer_service_calls = int(p[1
9]),
#                                                 churn = int(p[20]) ))

# Criando um Dataframe
    #telecomRDD4.take(3)

# Criando um Dataframe
    #telecomDF = spSession.createDataFrame(telecomRDD4)
    #telecomDF.cache()
    #telecomDF.toPandas().describe()

#String Index para indexar as variaveis que são string
    #area_s = StringIndexer(inputCol='area_code', outputCol= 'area_code_string').fit(te
lecomDF)
#String Index para indexar as variaveis que são string
    #state_s = StringIndexer(inputCol='state', outputCol= 'state_string').fit(telecomD
F)

    #telecomDF_s = area_s.transform(telecomDF)
    #telecomDF_s = state_s.transform(telecomDF_s)
    #telecomDF_s.take(1)
    #telecomDF_s = telecomDF_s.drop(*['area_code','state'])
    #telecomDF_s.take(1)
```