In [1]:
```
import pyspark
import findspark
from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression

import pandas as pd

from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

In [2]:
```
spark = SparkSession.builder.master('local[*]').getOrCreate()
```

```
23/04/20 01:22:39 WARN Utils: Your hostname, MacBook-Air-Evgenij.local resolv
es to a loopback address: 127.0.0.1; using 192.168.0.102 instead (on interfac
e en0)
23/04/20 01:22:39 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to anoth
er address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogL
evel(newLevel).
23/04/20 01:22:39 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
```

In [3]:
```
spark
```

Out[3]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.4.0`
Master	`local[*]`
AppName	`pyspark-shell`

In [4]:
```
df = pd.read_csv('./santander-customer-transaction-prediction/train.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	va
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.62
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.53
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.92
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2

5 rows × 202 columns

In [6]:
```
df.dtypes
```

Out[6]:
```
ID_code     object
target       int64
var_0      float64
var_1      float64
var_2      float64
            ...
var_195    float64
var_196    float64
var_197    float64
var_198    float64
var_199    float64
Length: 202, dtype: object
```

In [7]:
```
input_columns = df.columns.tolist()[2:]
```

In [8]:
```
dataset = spark.read.csv('./santander-customer-transaction-prediction/train.
```

In [9]:
```
input_columns = dataset.columns[2:]
```

In [10]:
```
assembler = VectorAssembler(inputCols=input_columns, outputCol='Attributes')

output = assembler.transform(dataset)

finalized_data = output.select('Attributes', 'target')

finalized_data.show()
```

```
23/04/20 01:22:46 WARN package: Truncated the string representation of a plan
since it was too large. This behavior can be adjusted by setting 'spark.sql.d
ebug.maxToStringFields'.
```

```
+--------------------+------+
|          Attributes|target|
+--------------------+------+
|[8.9255,-6.7863,1...|     0|
|[11.5006,-4.1473,...|     0|
|[8.6093,-2.7457,1...|     0|
|[11.0604,-2.1518,...|     0|
|[9.8369,-1.4834,1...|     0|
|[11.4763,-2.3182,...|     0|
|[11.8091,-0.0832,...|     0|
|[13.558,-7.9881,1...|     0|
|[16.1071,2.4426,1...|     0|
|[12.5088,1.9743,8...|     0|
|[5.0702,-0.5447,9...|     0|
|[12.7188,-7.975,1...|     0|
|[8.7671,-4.6154,9...|     0|
|[16.3699,1.5934,1...|     1|
|[13.808,5.0514,17...|     0|
|[3.9416,2.6562,13...|     0|
|[5.0615,0.2689,15...|     0|
|[8.4199,-1.8128,8...|     0|
|[4.875,1.2646,11....|     0|
|[4.409,-0.7863,15...|     0|
+--------------------+------+
only showing top 20 rows
```

In [11]:

```
train_data, test_data = finalized_data.randomSplit([0.8, 0.2])

regressor = LogisticRegression(featuresCol='Attributes', labelCol='target')

regressor = regressor.fit(train_data)

pred = regressor.evaluate(test_data)

pred.predictions.show()
```

```
23/04/20 01:22:51 WARN InstanceBuilder: Failed to load implementation from:de
v.ludovic.netlib.blas.JNIBLAS
23/04/20 01:22:51 WARN InstanceBuilder: Failed to load implementation from:de
v.ludovic.netlib.blas.VectorBLAS
23/04/20 01:22:52 WARN GarbageCollectionMetrics: To enable non-built-in garba
ge collector(s) List(G1 Concurrent GC), users should configure it(them) to sp
ark.eventLog.gcMetrics.youngGenerationGarbageCollectors or spark.eventLog.gcM
etrics.oldGenerationGarbageCollectors
```

```
+--------------------+------+--------------------+--------------------+------
----+
|          Attributes|target|       rawPrediction|         probability|predic
tion|
+--------------------+------+--------------------+--------------------+------
```

```
----+
|[1.6044,-0.6302,1...|     0|[2.81577733270635...|[0.94352247066797...|
0.0|
|[1.922,-3.31,8.37...|     0|[2.97425341712561...|[0.95139733495655...|
0.0|
|[2.5041,-4.9081,8...|     0|[5.21681693215687...|[0.99460469873667...|
0.0|
|[2.6501,-2.6365,1...|     0|[3.24585880260540...|[0.96252401910992...|
0.0|
|[2.8253,4.2618,15...|     0|[5.54884902547382...|[0.99612315424485...|
0.0|
|[3.6355,4.1567,14...|     1|[1.94706342408893...|[0.87512608491385...|
0.0|
|[3.7917,-1.0115,8...|     0|[4.41978174446677...|[0.98810630363823...|
0.0|
|[3.8296,-7.5727,8...|     0|[3.69750919841062...|[0.97581426332228...|
0.0|
|[3.8649,2.1927,10...|     0|[2.66281427774470...|[0.93479641269697...|
0.0|
|[3.9177,1.2849,5....|     0|[6.25206108581090...|[0.99807722478470...|
0.0|
|[3.9977,2.2683,8....|     0|[3.19744450777572...|[0.96073799617652...|
0.0|
|[4.0771,-7.9056,7...|     0|[5.50612310511931...|[0.99595460751027...|
0.0|
|[4.1896,-2.8807,1...|     1|[0.61723846028127...|[0.64959021922551...|
0.0|
|[4.203,-7.5025,14...|     0|[4.14977376339386...|[0.98447678639762...|
0.0|
|[4.2341,-8.5367,1...|     0|[5.33895359777056...|[0.99522204668995...|
0.0|
|[4.2485,1.1863,6....|     0|[4.03553113542261...|[0.98263073503312...|
0.0|
|[4.3542,2.359,9.6...|     0|[6.02678492172501...|[0.99759256943255...|
0.0|
|[4.3836,-1.4746,9...|     0|[2.28617095563942...|[0.90772522910623...|
0.0|
|[4.3854,-3.0492,7...|     0|[1.47928493562790...|[0.81446455016284...|
0.0|
|[4.409,-0.7863,15...|     0|[3.93850733776943...|[0.98089485005077...|
0.0|
+--------------------+------+--------------------+--------------------+------
----+
only showing top 20 rows
```

In [12]:
```
coef = regressor.coefficients

intr = regressor.intercept

print('coef: %a' % coef)
print('intr: %a' % intr)
```

```
coef: DenseVector([0.0531, 0.0386, 0.0591, 0.0221, 0.0224, 0.0147, 0.2342, -0
.0038, 0.0217, -0.084, 0.0019, 0.0161, -1.1831, -0.0341, -0.0037, 0.167, 0.00
77, -0.0016, 0.0196, 0.0072, -0.011, -0.0235, 0.0629, -0.1602, 0.0346, 0.1419
, 0.0388, -0.0081, -0.1076, 0.0057, -0.0006, -0.0439, 0.0314, -0.0396, -0.311
8, 0.0262, -0.0395, 0.008, -0.0032, -0.0026, 0.0187, -0.0011, -0.0406, -0.351
4, -0.0319, -0.0022, 0.002, 0.0031, 0.0093, 0.0109, -0.0538, 0.0048, 0.0153,
0.2585, -0.0067, 0.0129, -0.0269, -0.0896, -0.0188, -0.0352, 0.0098, 0.0038,
0.0228, -0.0182, -0.0302, 0.01, 0.0784, 0.0206, -5.3815, 0.0062, 0.0056, 0.37
75, -0.0163, -0.0008, 0.0059, -0.0266, -0.0236, -0.0115, 0.0718, 0.0341, -0.0
283, -0.1025, 0.0099, -0.0064, 0.0099, -0.0256, -0.0193, -0.0147, -0.0225, 0.
0341, 0.0061, 0.8587, -0.0326, -0.2586, 0.0663, 0.2246, 0.0026, 0.0058, -0.01
57, 0.1014, -0.0009, -0.0107, -0.0068, 0.113, -0.039, 0.1175, 0.065, -0.0161,
-0.8179, -0.035, 0.0521, 0.0885, 0.0538, -0.0201, -0.109, -0.0605, -0.0562, 0
.0009, 0.015, 0.0238, -0.004, -0.0736, -0.03, -0.0208, -0.0065, 0.245, 0.029,
-0.0402, 0.0269, -0.0061, 0.1545, -0.2222, -0.0642, 0.422, 0.0073, 0.0116, -0
.0027, 0.0089, 0.0093, -0.0278, 0.0134, -0.0128, -0.0164, -0.0207, 0.1115, 0.
0295, -0.0843, 0.0185, -0.8814, -0.0132, -0.0298, 0.026, -0.0128, -0.0105, -0
.0268, 0.0219, -0.0733, 0.0127, -0.004, 0.0157, -0.0007, 0.1049, 0.0831, 0.01
96, 0.0249, -0.0376, -0.5367, 0.011, 0.013, -0.4508, 0.0379, 0.0116, -0.0147,
0.0286, -0.0261, 0.0355, 0.0054, -0.0408, -0.0043, 0.0594, 0.0194, 0.0416, -0
.004, 0.0009, 0.017, 0.0011, -0.0309, 0.004, -0.0308, 0.0435, 0.0397, 0.0429,
-0.0814, -0.016, -0.0215, 0.0638, 0.0154, -0.1337, -0.0572, 0.0097])
intr: 60.6480039546467
```

In [13]:
```
eval = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction', label
```

In [14]:
```
roc = eval.evaluate(pred.predictions)
print('ROC: %.3f' % roc)
```

```
ROC: 0.853
```