BIGDATA
PROJECT

GROUP 2
CABIBIHAN | CABRERA | CORTEZ | GAITE | JIMENEZ | MAGBOO | MARANION

CONTENT

01 ABOUT THE DATA

02 SQL QUERIES & VISUALIZATION

03 MACHINE LEARNING IMPLEMENTATION

04 ROC IMPLEMENTATION

05 LOGISTIC REGRESSION

06 CONCLUSION

DATA INFO

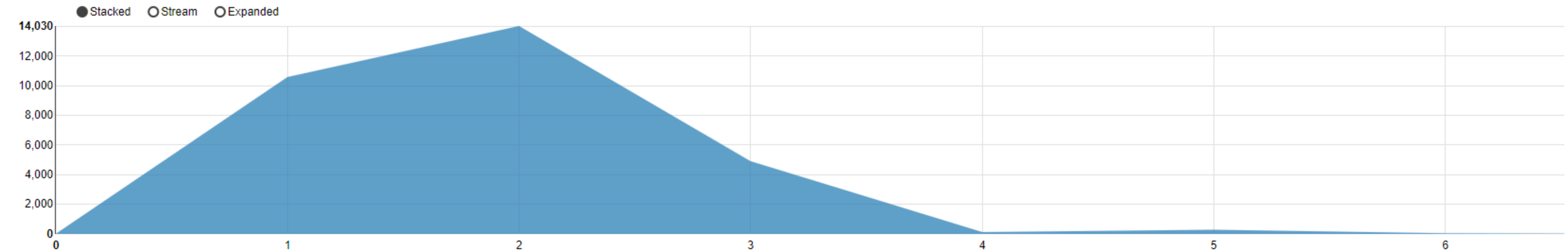
This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

CONSISTENCY OF PAYMENTS FOR MEMBERS WITH DIFFERENT EDUCATION LEVELS

```
%sql
SELECT X3 AS Education, COUNT(*) AS count, SUM(Y) AS total_payment
FROM credit_card
GROUP BY X3
ORDER BY total_payment DESC
```

SPARK JOB

settings



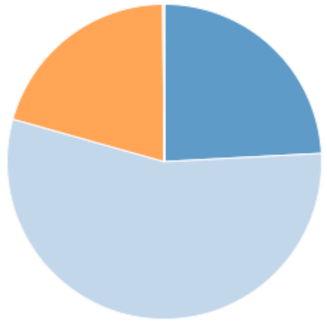
CONSISTENCY OF PAYMENTS FOR MEMBERS WITH DIFFERENT EDUCATION LEVELS

```
%sql
SELECT X3 AS Education, COUNT(*) AS count, SUM(Y) AS total_payment
FROM credit_card
WHERE (X6 + X7 + X8 + X9 + X10 + X11) >= 6
GROUP BY X3
ORDER BY total_payment DESC
```

SPARK JOB FINISHED

settings

1 2 3 5 6

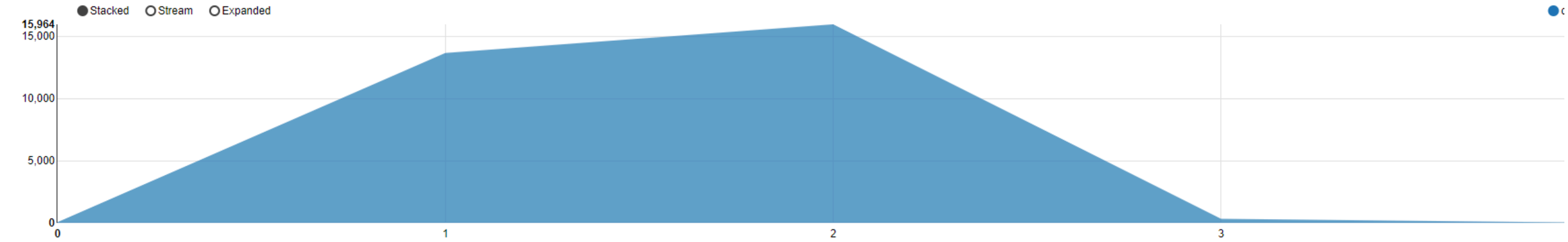


CONSISTENCY OF PAYMENTS FOR MARRIED AND UNMARRIED MEMBERS

```
%sql
SELECT X4 AS Marriage, COUNT(*) AS count, SUM(Y) AS total_payment
FROM credit_card
GROUP BY X4
ORDER BY total_payment DESC
```

SPARK JOB FINISHED

settings

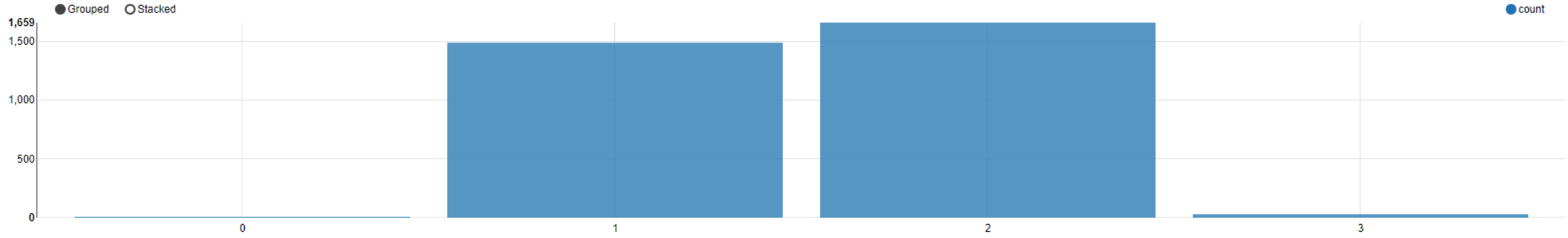


CONSISTENCY OF PAYMENTS FOR MARRIED AND UNMARRIED MEMBERS, AND ONLY INCLUDE MEMBERS WHO HAVE MADE AT LEAST 6 PAYMENTS

```
%sql
SELECT X4 AS Marriage, COUNT(*) AS count, SUM(Y) AS total_payment
FROM credit_card
WHERE (X6 + X7 + X8 + X9 + X10 + X11) >= 6
GROUP BY X4
ORDER BY total_payment DESC
```

SPARK JOB FINISHED

settings



Took 4 sec. Last updated by anonymous at July 29 2023, 2:49:31 AM. (outdated)

MACHINE LEARNING IMPLEMENTATION

LOGISTIC REGRESSION

is a Machine Learning classification algorithm that is used to predict the probability of certain classes based on some dependent variables. In short, the logistic regression model computes a sum of the input features and calculates the logistic of the result.

ROC CURVE

also known as Receiver Operating Characteristics Curve, is a metric used to measure the performance of a classifier model.

The ROC curve depicts the rate of true positives with respect to the rate of false positives, therefore highlighting the sensitivity of the classifier model.

WHY IS IT IMPORTANT?

DATA WRANGLING

```
%spark2
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.linalg.DenseVector
import org.apache.spark.ml.stat.Summarizer
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.sql.functions.col
import org.apache.spark.ml.Pipeline

val df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("hdfs:///tmp/data/credit_cards.csv")
credit_card_df.createOrReplaceTempView("credit_card")
```

```
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.linalg.DenseVector
import org.apache.spark.ml.stat.Summarizer
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.sql.functions.col
import org.apache.spark.ml.Pipeline
df: org.apache.spark.sql.DataFrame = [ID: int, LIMIT_BAL: decimal(7,0) ... 23 more fields]
```

Took 3 sec. Last updated by anonymous at July 25 2023, 10:13:52 AM.

```
%spark2
df.show(10)
```

```
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+---------+---------+---------+--------+--------+--------+--------+--------+--------+--------------------------+
| ID|LIMIT_BAL|SEX|EDUCATION|MARRIAGE|AGE|PAY_0|PAY_2|PAY_3|PAY_4|PAY_5|PAY_6|BILL_AMT1|BILL_AMT2|BILL_AMT3|BILL_AMT4|BILL_AMT5|BILL_AMT6|PAY_AMT1|PAY_AMT2|PAY_AMT3|PAY_AMT4|PAY_AMT5|PAY_AMT6|default_payment_next_month|
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+---------+---------+---------+--------+--------+--------+--------+--------+--------+--------------------------+
|  1|    20000|  2|        2|       1| 24|    2|    2|   -1|   -1|   -2|   -2|     3913|     3102|      689|        0|        0|        0|       0|     689|       0|       0|       0|       0|                         1|
|  2|   120000|  2|        2|       2| 26|   -1|    2|    0|    0|    0|    2|     2682|     1725|     2682|     3272|     3455|     3261|       0|    1000|    1000|    1000|       0|    2000|                         1|
|  3|    90000|  2|        2|       2| 34|    0|    0|    0|    0|    0|    0|    29239|    14027|    13559|    14331|    14948|    15549|    1518|    1500|    1000|    1000|    1000|    5000|                         0|
|  4|    50000|  2|        2|       1| 37|    0|    0|    0|    0|    0|    0|    46990|    48233|    49291|    28314|    28959|    29547|    2000|    2019|    1200|    1100|    1069|    1000|                         0|
|  5|    50000|  1|        2|       1| 57|   -1|    0|   -1|    0|    0|    0|     8617|     5670|    35835|    20940|    19146|    19131|    2000|   36681|   10000|    9000|     689|     679|                         0|
|  6|    50000|  1|        1|       2| 37|    0|    0|    0|    0|    0|    0|    64400|    57069|    57608|    19394|    19619|    20024|    2500|    1815|     657|    1000|    1000|     800|                         0|
|  7|   500000|  1|        1|       2| 29|    0|    0|    0|    0|    0|    0|   367965|   412023|   445007|   542653|   483003|   473944|   55000|   40000|   38000|   20239|   13750|   13770|                         0|
|  8|   100000|  2|        2|       2| 23|    0|   -1|   -1|    0|    0|   -1|    11876|      380|      601|      221|     -159|      567|     380|     601|       0|     581|    1687|    1542|                         0|
|  9|   140000|  2|        3|       1| 28|    0|    0|    2|    0|    0|    0|    11285|    14096|    12108|    12211|    11793|     3719|    3329|       0|     432|    1000|    1000|    1000|                         0|
| 10|    20000|  1|        3|       2| 35|   -2|   -2|   -2|   -2|   -1|   -1|        0|        0|        0|        0|    13007|    13912|       0|       0|       0|   13007|    1122|       0|                         0|
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+---------+---------+---------+--------+--------+--------+--------+--------+--------+--------------------------+
only showing top 10 rows
```

DATA WRANGLING

```
%spark2
import org.apache.spark.sql.functions._


val totalRows = df.count()
val nullCounts = df.select(df.columns.map(c => sum(when(col(c).isNull, 1).otherwise(0)).alias(s"${c}nullcount")): _*)

val NullValues = nullCounts.columns.forall(c => nullCounts.select(c).collect()(0) == 0)

if (NullValues) {
  println("There are null values in the DataFrame.")
    nullCounts.show()
} else {
  println(s"There are no null values in the DataFrame.")
}
```

```
There are no null values in the DataFrame.
import org.apache.spark.sql.functions._
totalRows: Long = 30000
nullCounts: org.apache.spark.sql.DataFrame = [IDnullcount: bigint, LIMIT_BALnullcount: bigint ... 23 more fields]
NullValues: Boolean = false
```

Took 2 sec. Last updated by anonymous at July 25 2023, 10:13:55 AM.

```
%spark2

val numRows = df.count()
val numCol = df.columns.length

println(s"($numRows,$numCol)")
```

```
(30000,25)
numRows: Long = 30000
numCol: Int = 25
```

Took 1 sec. Last updated by anonymous at July 25 2023, 10:13:56 AM.

DATA WRANGLING

```
%spark2                                                                                     SPARK JOB FINISHED

val summaryDF = df.describe()

summaryDF.show()
```

```
+-------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-------------------+-----------------+--------------------+-----------------+-----------------+-----------------+----
------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+--------------------------+
|summary|               ID|        LIMIT_BAL|              SEX|        EDUCATION|         MARRIAGE|              AGE|            PAY_0|              PAY_2|            PAY_3|               PAY_4|            PAY_5|            PAY_6|        BILL_AMT1|
BILL_AMT2|        BILL_AMT3|        BILL_AMT4|        BILL_AMT5|        BILL_AMT6|         PAY_AMT1|         PAY_AMT2|         PAY_AMT3|         PAY_AMT4|         PAY_AMT5|         PAY_AMT6|default_payment_next_month|
+-------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-------------------+-----------------+--------------------+-----------------+-----------------+-----------------+----
------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+--------------------------+
|  count|            30000|            30000|            30000|            30000|            30000|            30000|            30000|              30000|            30000|               30000|            30000|            30000|            30000|
30000|            30000|            30000|            30000|            30000|            30000|            30000|            30000|            30000|            30000|            30000|                     30000|
|   mean|          15000.5|      167484.3227|1.6037333333333332|1.8531333333333333|1.5518666666666667|          35.4855|          -0.0167|-0.13376666666666667|          -0.1662|-0.22066666666666668|          -0.2662|          -0.2911|       51223.3309|
49179.0752|       47013.1548|       43262.9490|       40311.4010|       38871.7604|        5663.5805|        5921.1635|        5225.6815|        4826.0769|        4799.3876|        5215.5026|                    0.2212|
| stddev|8660.398374208891|129747.66156720246|0.4891291960902602|0.7903486597207269|0.5219696006132467|9.217904068090155|1.1238015279973335|  1.1971859730345495|1.1968675684465686|  1.1691386224023357|1.1331874060027525|1.149987625607897|73635.86057552966|7117
3.76878252832|69349.38742703677|64332.856133916444|60797.15577026471|59554.1075367459|16563.28035402577|23040.870402057186|17606.96146980311|15666.159744032062|15278.305679144742|17777.46577543531|       0.41506180569093254|
|    min|                1|            10000|                1|                0|                0|               21|               -2|                 -2|               -2|                  -2|               -2|               -2|          -165580|
-69777|          -157264|          -170000|           -81334|          -339603|                0|                0|                0|                0|                0|                0|                         0|
|    max|            30000|          1000000|                2|                6|                3|               79|                8|                  8|                8|                   8|                8|                8|           964511|
983931|          1664089|           891586|           927171|           961664|           873552|          1684259|           896040|           621000|           426529|           528666|                         1|
+-------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-------------------+-----------------+--------------------+-----------------+-----------------+-----------------+----
```

Took 2 sec. Last updated by anonymous at July 25 2023, 10:13:59 AM.

DATA PREPROCESSING

```
%spark2
//data prep
val selectedCols = Seq("LIMIT_BAL", "SEX", "EDUCATION", "MARRIAGE", "AGE", "PAY_0", "PAY_2",
"PAY_3", "PAY_4", "PAY_5", "PAY_6", "BILL_AMT1", "BILL_AMT2",
"BILL_AMT3", "BILL_AMT4", "BILL_AMT5", "BILL_AMT6", "PAY_AMT1",
"PAY_AMT2", "PAY_AMT3", "PAY_AMT4", "PAY_AMT5", "PAY_AMT6")
val featureCols = selectedCols.dropRight(1)
val assembler = new VectorAssembler().setInputCols(featureCols.toArray).setOutputCol("features")
val dataset = assembler.transform(df).select(col("features"), col("default_payment_next_month").alias("label"))
```

```
selectedCols: Seq[String] = List(LIMIT_BAL, SEX, EDUCATION, MARRIAGE, AGE, PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6, BILL_AMT1, BILL_AMT2, BILL_AMT3,
featureCols: Seq[String] = List(LIMIT_BAL, SEX, EDUCATION, MARRIAGE, AGE, PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6, BILL_AMT1, BILL_AMT2, BILL_AMT3, B
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_67c6bb6c83af
dataset: org.apache.spark.sql.DataFrame = [features: vector, label: int]
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:07 AM.

```
%spark2
//Split data
val Array(trainData, testData) = dataset.randomSplit(Array(0.7, 0.3), seed = 1234L)
```

```
trainData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: int]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: int]
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:08 AM.

MODEL TRAINING/PREDICTING

```
%spark2
//Training the classifier (logistic regression)
val lr = new LogisticRegression().setLabelCol("label").setFeaturesCol("features")
val model = lr.fit(trainData)
```

```
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_e0b15ac89777
model: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_e0b15ac89777
```

Took 14 sec. Last updated by anonymous at July 25 2023, 9:17:22 AM.

```
%spark2
//predictions
val predictions = model.transform(testData)
```

```
predictions: org.apache.spark.sql.DataFrame = [features: vector, label: int ... 3 more fields]
```

Took 0 sec. Last updated by anonymous at July 25 2023, 9:17:23 AM.

ROC COMPUTING

```
%spark2
//converted predictions to  RDD format
val scoreAndLabels = predictions.select(col("probability"), col("label"))
  .rdd
  .map { row =>
    val probability = row.getAs[org.apache.spark.ml.linalg.Vector]("probability")(1)
    (probability, row.getInt(1).toDouble)
  }
```

```
scoreAndLabels: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[2170] at map at <console>:165
```

```
%spark2
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
```

```
metrics: org.apache.spark.mllib.evaluation.BinaryClassificationMetrics = org.apache.spark.mllib.evaluation.BinaryClassificationMetrics@4ef8e504
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:25 AM.

```
%spark2
// Compute the ROC curve
val roc = metrics.roc()
```

```
roc: org.apache.spark.rdd.RDD[(Double, Double)] = UnionRDD[2180] at UnionRDD at BinaryClassificationMetrics.scala:90
```

Took 2 sec. Last updated by anonymous at July 25 2023, 9:17:27 AM.

```
%spark2
// Extract the ROC curve points
val rocPoints = roc.collect()
```

```
rocPoints: Array[(Double, Double)] = Array((0.0,0.0), (1.412030499858797E-4,0.0), (1.412030499858797E-4,5.08646998982706E-4), (1.412030499858797E-4,0.0(
(1.412030499858797E-4,0.00254323499491353), (2.824060999717594E-4,0.00254323499491353), (4.236091499576391E-4,0.00254323499491353), (5.648121999435188E-
2), (7.060152499293985E-4,0.003560528992878942), (8.472182999152782E-4,0.003560528992878942), (9.884213499011581E-4,0.003560528992878942), (0.0011296243(
5991861648)...
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:28 AM.

FPR AND TPR

```
%spark2
val rocDF = rocPoints.toSeq.toDF("False Positive Rate (FPR)", "True Positive Rate (TPR)")
z.show(rocDF)
```

settings

False Positive Rate (FPR)	True Positive Rate (TPR)
0.04617339734538266	0.3402848423194303
0.04631460039536854	0.3402848423194303
0.04645580344535442	0.3402848423194303
0.0465970064953403	0.3402848423194303
0.04603219429539678	0.3397761953204476
0.04617339734538266	0.3397761953204476
0.04603219429539678	0.3392675483214649
0.04560858514543914	0.3387589013224822

LOGISTIC REGRESSION

MODEL TRAINING

```
%spark2
//LOGISTIC REGRESSION
val newDat = df.drop("ID")
val featureColumns = newDat.columns.filter(_ != "default_payment_next_month")

val assembler = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")
val assembledData = assembler.transform(newDat).select("features", "default_payment_next_month").withColumnRenamed("default_payment_next_month", "label")
```

```
newDat: org.apache.spark.sql.DataFrame = [LIMIT_BAL: decimal(7,0), SEX: int ... 22 more fields]
featureColumns: Array[String] = Array(LIMIT_BAL, SEX, EDUCATION, MARRIAGE, AGE, PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6, BILL_AMT1, BILL_AMT2, BILL_AMT3, BI
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_d8d5851be3ce
assembledData: org.apache.spark.sql.DataFrame = [features: vector, label: int]
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:31 AM.

```
%spark2
val Array(trainingData, testData) = assembledData.randomSplit(Array(0.7, 0.3), seed = 1234)
```

```
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: int]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: int]
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:33 AM.

```
%spark2
val lr = new LogisticRegression()
  .setLabelCol("label")
  .setFeaturesCol("features")
  .setMaxIter(10) // Set the maximum number of iterations
  .setRegParam(0.3) // Set the regularization parameter (adjustable)
```

```
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_69161a9002fe
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:34 AM.

MODEL PREDICTING

```
%spark2
val pipeline = new Pipeline().setStages(Array(lr))
val model = pipeline.fit(trainingData)
```

```
pipeline: org.apache.spark.ml.Pipeline = pipeline_5626ae2ae154
model: org.apache.spark.ml.PipelineModel = pipeline_5626ae2ae154
```

Took 6 sec. Last updated by anonymous at July 25 2023, 9:17:40 AM.

```
%spark2
val predictions = model.transform(testData)
```

```
predictions: org.apache.spark.sql.DataFrame = [features: vector, label: int ... 3 more fields]
```

Took 1 sec. Last updated by anonymous at July 25 2023, 9:17:41 AM.

PERFORMANCE METRICS

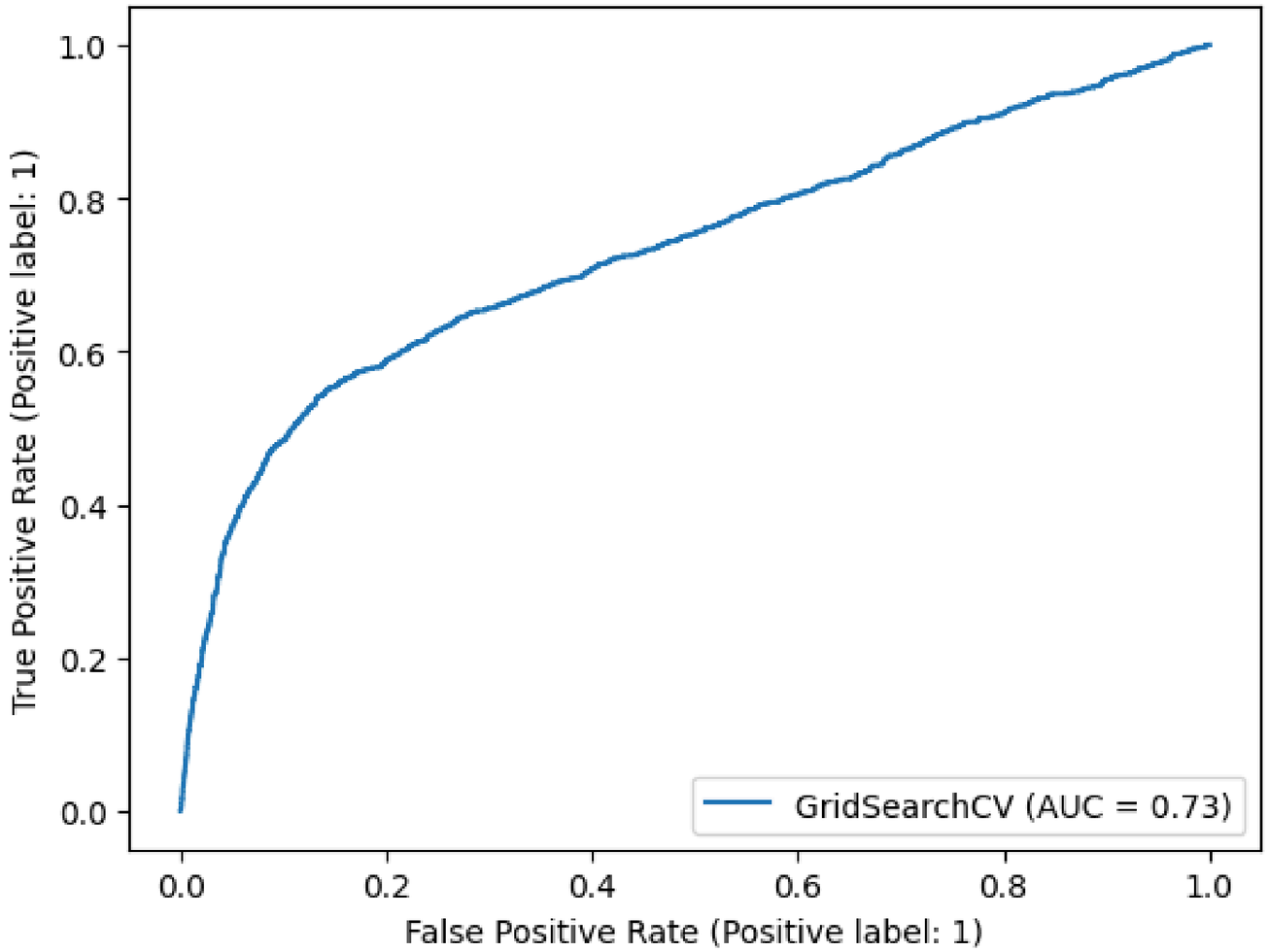
```
%spark2
val evaluator = new BinaryClassificationEvaluator()
  .setLabelCol("label")
  .setRawPredictionCol("rawPrediction")
  .setMetricName("areaUnderROC")

val areaUnderROC = evaluator.evaluate(predictions)
println(s"Area under ROC curve: $areaUnderROC")
```

```
Area under ROC curve: 0.7120467142448056
evaluator: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = binEval_6f3d04b7d1d1
areaUnderROC: Double = 0.7120467142448056
```

Took 3 sec. Last updated by anonymous at July 25 2023, 9:17:44 AM.

ROC CURVE


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
GridSearchCV (AUC = 0.73)