In [3]:

```
import pyspark
```

In [4]:

```
from pyspark import SparkConf
from pyspark import SparkContext

HDFS_MASTER = 'hadoop-master'

conf = SparkConf()
conf.setMaster('yarn')
conf.setAppName('spark-test')
#sc = SparkContext(conf=conf)
```

Out[4]:

```
<pyspark.conf.SparkConf at 0x7f484b70e748>
```

In [5]:

```
def test_function():
    for x in range(0,99999999999999999999999999999999):
        print(x)
    print("Done")


#appName = "Python Example - PySpark Row List to Pandas Data Frame"

# Create Spark session
#spark = SparkSession.builder.appName("sample-one").getOrCreate()

# Call the function
test_function()
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [ ]:

```
def test_function():
    while(1):
        print()
    print("Done")


#appName = "Python Example - PySpark Row List to Pandas Data Frame"

# Create Spark session
#spark = SparkSession.builder.appName("sample-one").getOrCreate()

# Call the function
test_function()
```

In [6]:

```
def test_function():
    x=0
    while(x<100):
        x+=1
    print("Done")


#appName = "Python Example - PySpark Row List to Pandas Data Frame"

# Create Spark session
#spark = SparkSession.builder.appName("sample-one").getOrCreate()

# Call the function
test_function()
```

```
Done
```

In [36]:

```
from pyspark import SQLContext
from pyspark.sql.types import StructField,StructType,IntegerType,StringType

conf = pyspark.SparkConf()

sc = pyspark.SparkContext.getOrCreate(conf=conf)
sqlcontext = SQLContext(sc)

schema = StructType([
    StructField("sales", IntegerType(),True),
    StructField("sales person", StringType(),True)
])

data = ([(10, 'Walker'),
        ( 20, 'Stepher')
        ])

df=sqlcontext.createDataFrame(data,schema=schema)
```

In [37]:

```
df.show()
```

```
+-----+------------+
|sales|sales person|
+-----+------------+
|   10|      Walker|
|   20|     Stepher|
+-----+------------+
```

In [38]:

```
from pyspark.sql.functions import udf
from pyspark.rdd import portable_hash
from pyspark import Row
```

In [58]:

```
print("Spark Version ::",spark.version)
spark.sparkContext.setLogLevel("ERROR")

# Populate sample data
countries = ("CN", "AU", "US")
data = []
for i in range(1, 13):
    data.append({"ID": i, "Country": countries[i % 3],  "Amount": 10+i})

def print_partitions(df):
    numPartitions = df.rdd.getNumPartitions()
    print("Total partitions: {}".format(numPartitions))
    print("Partitioner: {}".format(df.rdd.partitioner))
    df.explain()
    parts = df.rdd.glom().collect()
    i = 0
    j = 0
    for p in parts:
        print("Partition {}:".format(i))
        for r in p:
            print("Row {}:{}".format(j, r))
            j = j+1
        i = i+1


ddf = spark.createDataFrame(data)
ddf.show()
print_partitions(ddf)
```

```
Spark Version :: 2.4.4
+------+-------+---+
|Amount|Country| ID|
+------+-------+---+
|    11|     AU|  1|
|    12|     US|  2|
|    13|     CN|  3|
|    14|     AU|  4|
|    15|     US|  5|
|    16|     CN|  6|
|    17|     AU|  7|
|    18|     US|  8|
|    19|     CN|  9|
|    20|     AU| 10|
|    21|     US| 11|
|    22|     CN| 12|
+------+-------+---+

Total partitions: 2
Partitioner: None
== Physical Plan ==
Scan ExistingRDD[Amount#1469L,Country#1470,ID#1471L]

/home/scipyuser/spark/python/pyspark/sql/session.py:346: UserWarning:
inferring schema from dict is deprecated,please use pyspark.sql.Row in
stead
  warnings.warn("inferring schema from dict is deprecated,"

Partition 0:
Row 0:Row(Amount=11, Country='AU', ID=1)
```

```
Row 1:Row(Amount=12, Country='US', ID=2)
Row 2:Row(Amount=13, Country='CN', ID=3)
Row 3:Row(Amount=14, Country='AU', ID=4)
Row 4:Row(Amount=15, Country='US', ID=5)
Row 5:Row(Amount=16, Country='CN', ID=6)
Partition 1:
Row 6:Row(Amount=17, Country='AU', ID=7)
Row 7:Row(Amount=18, Country='US', ID=8)
Row 8:Row(Amount=19, Country='CN', ID=9)
Row 9:Row(Amount=20, Country='AU', ID=10)
Row 10:Row(Amount=21, Country='US', ID=11)
Row 11:Row(Amount=22, Country='CN', ID=12)
```

In [59]:

```
    numPartitions = 5

    df = df.repartition(numPartitions, "Country")

    print_partitions(df)

    udf_portable_hash = udf(lambda str: portable_hash(str))

    df = df.withColumn("Hash#", udf_portable_hash(df.Country))

    df = df.withColumn("Partition#", df["Hash#"] % numPartitions)

    df.show()
```

```
---------------------------------------------------------------------------
-------
Py4JJavaError                             Traceback (most recent cal
l last)
~/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:

~/spark/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_retur
n_value(answer, gateway_client, target_id, name)
    327                     "An error occurred while calling {0}{1}
{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:

Py4JJavaError: An error occurred while calling o1080.repartition.
: org.apache.spark.sql.AnalysisException: cannot resolve ``Country``
given input columns: [features];;
'RepartitionByExpression ['Country], 5
```

In [60]:

```
df.printSchema()
```

```
root
 |-- features: vector (nullable = true)
```

In [61]:

```
df.describe()
```

Out[61]:

```
DataFrame[summary: string]
```

In [62]:

```
# Customised Paritioning Function
```

In [63]:

```
def country_partitioning(k):
    return countries.index(k)

udf_country_hash = udf(lambda str: country_partitioning(str))
# Udf are most important in the implementation
# using udf one can design the customised functions to process the
# rdd or the data frame.

df = df.rdd \
    .map(lambda el: (el["Country"], el)) \
    .partitionBy(numPartitions, country_partitioning) \
    .toDF()
print_partitions(df)

df = df.withColumn("Hash#", udf_country_hash(df[0]))
df = df.withColumn("Partition#", df["Hash#"] % numPartitions)
df.show()
```

```
---------------------------------------------------------------------------
-------
Py4JJavaError                             Traceback (most recent cal
l last)
<ipython-input-63-aaa03b423777> in <module>
      9 df = df.rdd \
     10     .map(lambda el: (el["Country"], el)) \
---> 11     .partitionBy(numPartitions, country_partitioning) \
     12     .toDF()
     13 print_partitions(df)

~/spark/python/pyspark/sql/session.py in toDF(self, schema, sampleRa
tio)
     56         [Row(name=u'Alice', age=1)]
     57         """
---> 58         return sparkSession.createDataFrame(self, schema, sa
mpleRatio)
     59
     60     RDD.toDF = toDF
```

In [ ]:

```
import os

def myfun(x):
    os.system("pip install shapely")
    return x
rdd = sc.parallelize([1,2,3,4]) ## assuming 4 worker nodes
rdd.map(lambda x: myfun(x)).collect()
```

In [64]:

```
import matplotlib as plt
```

In [65]:

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation

data = [(Vectors.sparse(4, [(0, 1.0), (3, -2.0)]),),
        (Vectors.dense([4.0, 5.0, 0.0, 3.0]),),
        (Vectors.dense([6.0, 7.0, 0.0, 8.0]),),
        (Vectors.sparse(4, [(0, 9.0), (3, 1.0)]),)]
df = spark.createDataFrame(data, ["features"])

r1 = Correlation.corr(df, "features").head()
print("Pearson correlation matrix:\n" + str(r1[0]))

r2 = Correlation.corr(df, "features", "spearman").head()
print("Spearman correlation matrix:\n" + str(r2[0]))
```

```
Pearson correlation matrix:
DenseMatrix([[1.        , 0.05564149,        nan, 0.40047142],
             [0.05564149, 1.        ,        nan, 0.91359586],
             [       nan,        nan, 1.        ,        nan],
             [0.40047142, 0.91359586,        nan, 1.        ]])
Spearman correlation matrix:
DenseMatrix([[1.        , 0.10540926,        nan, 0.4       ],
             [0.10540926, 1.        ,        nan, 0.9486833 ],
             [       nan,        nan, 1.        ,        nan],
             [0.4       , 0.9486833 ,        nan, 1.        ]])
```

In [66]:

```
df.show()
```

```
+--------------------+
|            features|
+--------------------+
|(4,[0,3],[1.0,-2.0])|
|   [4.0,5.0,0.0,3.0]|
|   [6.0,7.0,0.0,8.0]|
| (4,[0,3],[9.0,1.0])|
+--------------------+
```

In [67]:

```
ddf.show()
```

```
+------+-------+---+
|Amount|Country| ID|
+------+-------+---+
|    11|     AU|  1|
|    12|     US|  2|
|    13|     CN|  3|
|    14|     AU|  4|
|    15|     US|  5|
|    16|     CN|  6|
|    17|     AU|  7|
|    18|     US|  8|
|    19|     CN|  9|
|    20|     AU| 10|
|    21|     US| 11|
|    22|     CN| 12|
+------+-------+---+
```

In [81]:

```
from pyspark.sql.functions import col
ddf.select().show()
ddf.where(col("Country").between(1,2)).show()
```

```
++
||
++
||
||
||
||
||
||
||
||
||
||
||
++

+------+-------+---+
|Amount|Country| ID|
+------+-------+---+
+------+-------+---+
```

In [93]:

```
from pyspark.mllib.linalg.distributed import RowMatrix

# Create an RDD of vectors.
rows = sc.parallelize([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

# Create a RowMatrix from an RDD of vectors.
mat = RowMatrix(rows)

# Get its size.
m = mat.numRows()  # 4
n = mat.numCols()  # 3

# Get the rows as an RDD of vectors again.
rowsRDD = mat.rows
print(m,n)
```

```
4 3
```

In [94]:

```
from pyspark.mllib.linalg import Matrix, Matrices

# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
dm2 = Matrices.dense(3, 2, [1, 3, 5, 2, 4, 6])

# Create a sparse matrix ((9.0, 0.0), (0.0, 8.0), (0.0, 6.0))
sm = Matrices.sparse(3, 2, [0, 1, 3], [0, 2, 1], [9, 6, 8])
print(dm2)
```

```
DenseMatrix([[1., 2.],
             [3., 4.],
             [5., 6.]])
```

In [95]:

```
from pyspark.mllib.linalg.distributed import IndexedRow, IndexedRowMatrix

# Create an RDD of indexed rows.
#   - This can be done explicitly with the IndexedRow class:
indexedRows = sc.parallelize([IndexedRow(0, [1, 2, 3]),
                              IndexedRow(1, [4, 5, 6]),
                              IndexedRow(2, [7, 8, 9]),
                              IndexedRow(3, [10, 11, 12])])
#   - or by using (long, vector) tuples:
indexedRows = sc.parallelize([(0, [1, 2, 3]), (1, [4, 5, 6]),
                              (2, [7, 8, 9]), (3, [10, 11, 12])])

# Create an IndexedRowMatrix from an RDD of IndexedRows.
mat = IndexedRowMatrix(indexedRows)

# Get its size.
m = mat.numRows()  # 4
n = mat.numCols()  # 3

# Get the rows as an RDD of IndexedRows.
rowsRDD = mat.rows

# Convert to a RowMatrix by dropping the row indices.
rowMat = mat.toRowMatrix()
```

In [96]:

```
from pyspark.mllib.linalg import Matrices
from pyspark.mllib.linalg.distributed import BlockMatrix

# Create an RDD of sub-matrix blocks.
blocks = sc.parallelize([((0, 0), Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])),
                         ((1, 0), Matrices.dense(3, 2, [7, 8, 9, 10, 11, 12]))])

# Create a BlockMatrix from an RDD of sub-matrix blocks.
mat = BlockMatrix(blocks, 3, 2)

# Get its size.
m = mat.numRows()  # 6
n = mat.numCols()  # 2

# Get the blocks as an RDD of sub-matrix blocks.
blocksRDD = mat.blocks

# Convert to a LocalMatrix.
localMat = mat.toLocalMatrix()

# Convert to an IndexedRowMatrix.
indexedRowMat = mat.toIndexedRowMatrix()

# Convert to a CoordinateMatrix.
coordinateMat = mat.toCoordinateMatrix
```

In [99]:

```
from pyspark.sql.functions import add_months
df = spark.createDataFrame([('2015-04-08',)], ['dt'])
df.select(add_months(df.dt, 1).alias('next_month')).collect()
```

Out[99]:

```
[Row(next_month=datetime.date(2015, 5, 8))]
```

In [101]:

```
from pyspark.sql.functions import approx_count_distinct
df.agg(approx_count_distinct(df.age).alias('distinct_ages')).collect()
```

```
---------------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
 last)
<ipython-input-101-0b15572343d0> in <module>
      1 from pyspark.sql.functions import approx_count_distinct
----> 2 df.agg(approx_count_distinct(df.age).alias('distinct_ages')).c
ollect()

~/spark/python/pyspark/sql/dataframe.py in __getattr__(self, name)
   1299         if name not in self.columns:
   1300             raise AttributeError(
-> 1301                 "'%s' object has no attribute '%s'" % (self.__
class__.__name__, name))
   1302         jc = self._jdf.apply(name)
   1303         return Column(jc)

AttributeError: 'DataFrame' object has no attribute 'age'
```

In [103]:

```
from pyspark.sql.functions import array_join
df = spark.createDataFrame([(["a", "b", "c"],), (["a", None],)], ['data'])
df.select(array_join(df.data, ",").alias("joined")).collect()
```

Out[103]:

```
[Row(joined='a,b,c'), Row(joined='a')]
```

In [108]:

```
from pyspark.mllib.evaluation import MultilabelMetrics

scoreAndLabels = sc.parallelize([
    ([0.0, 1.0], [0.0, 2.0]),
    ([0.0, 2.0], [0.0, 1.0]),
    ([], [0.0]),
    ([2.0], [2.0]),
    ([2.0, 0.0], [2.0, 0.0]),
    ([0.0, 1.0, 2.0], [0.0, 1.0]),
    ([1.0], [1.0, 2.0])])

# Instantiate metrics object
metrics = MultilabelMetrics(scoreAndLabels)

# Summary stats
print("Recall = %s" % metrics.recall())
print("Precision = %s" % metrics.precision())
print("F1 measure = %s" % metrics.f1Measure())
print("Accuracy = %s" % metrics.accuracy)

# Individual label stats
labels = scoreAndLabels.flatMap(lambda x: x[1]).distinct().collect()
for label in labels:
    print("Class %s precision = %s" % (label, metrics.precision(label)))
    print("Class %s recall = %s" % (label, metrics.recall(label)))
    print("Class %s F1 Measure = %s" % (label, metrics.f1Measure(label)))

# Micro stats
print("Micro precision = %s" % metrics.microPrecision)
print("Micro recall = %s" % metrics.microRecall)
print("Micro F1 measure = %s" % metrics.microF1Measure)

# Hamming loss
print("Hamming loss = %s" % metrics.hammingLoss)

# Subset accuracy
print("Subset accuracy = %s" % metrics.subsetAccuracy)
```

```
Recall = 0.6428571428571429
Precision = 0.6666666666666666
F1 measure = 0.6380952380952382
Accuracy = 0.5476190476190476
Class 2.0 precision = 0.5
Class 2.0 recall = 0.5
Class 2.0 F1 Measure = 0.5
Class 0.0 precision = 1.0
Class 0.0 recall = 0.8
Class 0.0 F1 Measure = 0.888888888888889
Class 1.0 precision = 0.6666666666666666
Class 1.0 recall = 0.6666666666666666
Class 1.0 F1 Measure = 0.6666666666666666
Micro precision = 0.7272727272727273
Micro recall = 0.6666666666666666
Micro F1 measure = 0.6956521739130435
Hamming loss = 0.3333333333333333
Subset accuracy = 0.2857142857142857
```

In [113]:

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.util import MLUtils

# Several of the methods available in scala are currently missing from pyspark
# Load training data in LIBSVM format
data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_binary_classification_data.txt

# Split data into training (60%) and test (40%)
training, test = data.randomSplit([0.6, 0.4], seed=11)
training.cache()

# Run training algorithm to build the model
model = LogisticRegressionWithLBFGS.train(training)

# Compute raw scores on the test set
predictionAndLabels = test.map(lambda lp: (float(model.predict(lp.features)), lp.la

# Instantiate metrics object
metrics = BinaryClassificationMetrics(predictionAndLabels)

# Area under precision-recall curve
print("Area under PR = %s" % metrics.areaUnderPR)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
---------------------------------------------------------------------------
-----
Py4JJavaError                             Traceback (most recent call
 last)
<ipython-input-113-db42bcf27683> in <module>
      5 # Several of the methods available in scala are currently miss
ing from pyspark
      6 # Load training data in LIBSVM format
----> 7 data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_binary_cl
assification_data.txt")
      8
      9 # Split data into training (60%) and test (40%)

~/spark/python/pyspark/mllib/util.py in loadLibSVMFile(sc, path, numFe
atures, minPartitions, multiclass)
    124         if numFeatures <= 0:
    125             parsed.cache()
--> 126             numFeatures = parsed.map(lambda x: -1 if x[1].size
== 0 else x[1][-1]).reduce(max) + 1
    127         return parsed.map(lambda x: LabeledPoint(x[0], Vectors
.sparse(numFeatures, x[1], x[2])))
    128

~/spark/python/pyspark/rdd.py in reduce(self, f)
    842             yield reduce(f, iterator, initial)
    843
--> 844         vals = self.mapPartitions(func).collect()
    845         if vals:
    846             return reduce(f, vals)

~/spark/python/pyspark/rdd.py in collect(self)
    814         """
```

```
    815         with SCCallSiteSync(self.context) as css:
--> 816             sock_info = self.ctx._jvm.PythonRDD.collectAndServ
e(self._jrdd.rdd())
    817         return list(_load_from_socket(sock_info, self._jrdd_de
serializer))
    818

~/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in __call_
_(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
-> 1257             answer, self.gateway_client, self.target_id, self.
name)
   1258
   1259         for temp_arg in temp_args:

~/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
     61     def deco(*a, **kw):
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:
     65             s = e.java_exception.toString()

~/spark/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_return_
value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}
{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling z:org.apache.spark.api.
python.PythonRDD.collectAndServe.
: org.apache.hadoop.mapred.InvalidInputException: Input path does not
 exist: hdfs://172.27.35.73:9000/user/scipyuser/data/mllib/sample_bina
ry_classification_data.txt
        at org.apache.hadoop.mapred.FileInputFormat.singleThreadedList
Status(FileInputFormat.java:287)
        at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInp
utFormat.java:229)
        at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInpu
tFormat.java:315)
        at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scal
a:204)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPart
itionsRDD.scala:49)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.api.python.PythonRDD.getPartitions(PythonR
DD.scala:55)
```

```
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.api.python.PythonRDD.getPartitions(PythonR
DD.scala:55)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.sc
ala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.SparkContext.runJob(SparkContext.scala:212
6)
        at org.apache.spark.rdd.RDD$$anonfun$collect$1.apply(RDD.scal
a:945)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperat
ionScope.scala:151)
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperat
ionScope.scala:112)
        at org.apache.spark.rdd.RDD.withScope(RDD.scala:363)
        at org.apache.spark.rdd.RDD.collect(RDD.scala:944)
        at org.apache.spark.api.python.PythonRDD$.collectAndServe(Pyth
onRDD.scala:166)
        at org.apache.spark.api.python.PythonRDD.collectAndServe(Pytho
nRDD.scala)
        at sun.reflect.GeneratedMethodAccessor62.invoke(Unknown Sourc
e)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingM
ethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:24
4)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.ja
va:357)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.
java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:748)
```

In [136]:

```
data=[(0.0, 1.0, 0.0, 2.0),
    (0.0, 2.0, 0.0, 1.0),
    (1.0,8.0,7.7,0.0),
    (2.0, 2.0,7,8),
    (2.0, 0.0, 2.0, 0.0),
    (0.0, 1.0, 2.0, 0.0),
    (1.0, 1.0,5.9,2.0)]
scoreAndLabels = sc.parallelize(data)
xx=scoreAndLabels.toDF()
xx.show()
f=['_1','_3']
xx.select(f).show()
```

```
+---+---+----+----+
| _1| _2|  _3|  _4|
+---+---+----+----+
|0.0|1.0| 0.0| 2.0|
|0.0|2.0| 0.0| 1.0|
|1.0|8.0| 7.7| 0.0|
|2.0|2.0|null|null|
|2.0|0.0| 2.0| 0.0|
|0.0|1.0| 2.0| 0.0|
|1.0|1.0| 5.9| 2.0|
+---+---+----+----+

+---+----+
| _1|  _3|
+---+----+
|0.0| 0.0|
|0.0| 0.0|
|1.0| 7.7|
|2.0|null|
|2.0| 2.0|
|0.0| 2.0|
|1.0| 5.9|
+---+----+
```

In [ ]: