# Count number of non-NaN entries in each column of Spark dataframe with Pyspark


I have a very large dataset that is loaded in Hive. It consists of about 1.9 million rows and 1450 columns. I need to determine the "coverage" of each of the columns, meaning, the fraction of rows that have non-NaN values for each column.

Here is my code:

```
from pyspark import SparkContext
from pyspark.sql import HiveContext
import string as string

sc = SparkContext(appName="compute_coverages") ## Create the context
sqlContext = HiveContext(sc)

df = sqlContext.sql("select * from data_table")
nrows_tot = df.count()

covgs=sc.parallelize(df.columns)
        .map(lambda x: str(x))
```

```
        .map(lambda x: (x, float(df.select(x).dropna().count()) / float(nrows_tot) *
100.))
```

Trying this out in the pyspark shell, if I then do covgs.take(10), it returns a rather large error stack. It says that there's a problem in save in the file `/usr/lib64/python2.6/pickle.py` . This is the final part of the error:

```
py4j.protocol.Py4JError: An error occurred while calling o37.__getnewargs__. Trace:
py4j.Py4JException: Method __getnewargs__([]) does not exist
        at py4j.reflection.ReflectionEngine.getMethod(ReflectionEngine.java:333)
        at py4j.reflection.ReflectionEngine.getMethod(ReflectionEngine.java:342)
        at py4j.Gateway.invoke(Gateway.java:252)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:133)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:207)
        at java.lang.Thread.run(Thread.java:745)
```

If there is a better way to accomplish this than the way I'm trying, I'm open to suggestions. I can't use pandas, though, as it's not currently available on the cluster I work on and I don't have rights to install it.

python apache-spark apache-spark-sql pyspark

edited Nov 24 '15 at 23:31

zero323
**69.1k** 16 82 148

asked Nov 24 '15 at 18:03

RKD314
**140** 1 8

## 1 Answer

Let's start with a dummy data:

```
from pyspark.sql import Row

row = Row("x", "y", "z")
df = sc.parallelize([
    row(0, 1, 2), row(None, 3, 4), row(None, None, 5)]).toDF()

## +----+----+---+
## |   x|   y|  z|
## +----+----+---+
## |   0|   1|  2|
## |null|   3|  4|
```

```
## |null|null|  5|
## +----+----+---+
```

All you need is a simple aggregation:

```
from pyspark.sql.functions import col, count, sum

def count_not_null(c):
    """Use conversion between boolean and integer
    - False -> 0
    - True ->  1
    """
    return sum(col(c).isNotNull().cast("integer")).alias(c)

exprs = [count_not_null(c) for c in df.columns]
df.agg(*exprs).show()

## +---+---+---+
## |  x|  y|  z|
## +---+---+---+
## |  1|  2|  3|
## +---+---+---+
```

You can also leverage SQL `NULL` semantics to achieve the same result without creating a custom function:

```
df.agg(*[
    count(c).alias(c)    # vertical (column-wise) operations in SQL ignore NULLs
    for c in df.columns
]).show()

## +---+---+---+
## |  x|  y|  z|
## +---+---+---+
## |  1|  2|  3|
## +---+---+---+
```

If you prefer fractions:

```
exprs = [(count_not_null(c) / count("*")).alias(c) for c in df.columns]
df.agg(*exprs).show()

## +------------------+------------------+---+
## |                 x|                 y|  z|
## +------------------+------------------+---+
```

```
## |0.3333333333333333|0.6666666666666666|1.0|
## +------------------+------------------+---+
```

or

```
# COUNT(*) is equivalent to COUNT(1) so NULLs won't be an issue
df.select(*[(count(c) / count("*")).alias(c) for c in df.columns]).show()

## +------------------+------------------+---+
## |                 x|                 y|  z|
## +------------------+------------------+---+
## |0.3333333333333333|0.6666666666666666|1.0|
## +------------------+------------------+---+
```

edited Jul 7 at 14:50

eliasah
**9,173** 3 21 48

answered Nov 24 '15 at 18:38

zero323
**69.1k** 16 82 148