How take a random row from a PySpark DataFrame?

Asked 7 years, 1 month ago   Modified 11 months ago   Viewed 86k times

47

How can I get a random row from a PySpark DataFrame? I only see the method `sample()` which takes a fraction as parameter. Setting this fraction to `1/numberOfRows` leads to random results, where sometimes I won't get any row.

On `RDD` there is a method `takeSample()` that takes as a parameter the number of elements you want the sample to contain. I understand that this might be slow, as you have to count each partition, but is there a way to get something like this on a DataFrame?

python   apache-spark   dataframe   pyspark   apache-spark-sql

Share Edit Follow

edited Mar 8, 2021 at 11:00
mck
39.6k 13 34 49

asked Nov 30, 2015 at 16:29
DanT
3,800 5 28 33

3 Answers

Sorted by: Highest score (default)

84

You can simply call `takeSample` on a `RDD` :

```
df = sqlContext.createDataFrame(
    [(1, "a"), (2, "b"), (3, "c"), (4, "d")], ("k", "v"))
df.rdd.takeSample(False, 1, seed=0)
## [Row(k=3, v='c')]
```

If you don't want to collect you can simply take a higher fraction and limit:

```
df.sample(False, 0.1, seed=0).limit(1)
```

Don't pass a `seed` , and you should get a different DataFrame each time.

Share Edit Follow

edited Feb 18, 2022 at 7:03
Jacek Laskowski
71.1k 26 235 411

answered Dec 1, 2015 at 2:06
zero323
316k 97 948 930

2 Is there a way of getting random values. In the above case the same dataframe in produced each time I run the query. – Nikhil Baby Dec 20, 2017 at 9:50

1 Nice tip, @LateCoder! (On Spark 2.3.1, keeping seed=None only seems to work for df.rdd.takeSample, not df.sample.) – Quentin Pradet Jul 17, 2018 at 11:00

1 Why might one not want to `collect` ? – ijoseph Jan 14, 2020 at 0:41

5 Oh because `collect` returns it to the driver program and it might not fit in the driver's memory. – ijoseph Jan 14, 2020 at 0:47

1 I don't think the second sample -> limit solution is quite random. The sample() part is good and random, but then the results seem to be somewhat sorted before the limit is taken. This is especially obvious if you do limit(10) instead of 1 and your fraction is too big. the results may look similar. – Paul Fornia Dec 10, 2020 at 21:58

15

Different Types of Sample

Randomly sample % of the data with and without replacement

```
import pyspark.sql.functions as F
#Randomly sample 50% of the data without replacement
sample1 = df.sample(False, 0.5, seed=0)

#Randomly sample 50% of the data with replacement
sample1 = df.sample(True, 0.5, seed=0)

#Take another sample exlcuding records from previous sample using Anti Join
sample2 = df.join(sample1, on='ID', how='left_anti').sample(False, 0.5, seed=0)

#Take another sample exlcuding records from previous sample using Where
sample1_ids = [row['ID'] for row in sample1.ID]
sample2 = df.where(~F.col('ID').isin(sample1_ids)).sample(False, 0.5, seed=0)

#Generate a startfied sample of the data across column(s)
#Sampling is probabilistic and thus cannot guarantee an exact number of rows
fractions = {
        'NJ': 0.5, #Take about 50% of records where state = NJ
    'NY': 0.25, #Take about 25% of records where state = NY
    'VA': 0.1, #Take about 10% of records where state = VA
}
stratified_sample = df.sampleBy(F.col('state'), fractions, seed=0)
```

Share Edit Follow

answered Dec 11, 2020 at 18:32
Yash M
326 3 7

0

Here's an alternative using Pandas DataFrame.Sample method. This uses the spark `applyInPandas` method to distribute the groups, available from Spark 3.0.0. This allows you to select an exact number of rows per group.

I've added `args` and `kwargs` to the function so you can access the other arguments of `DataFrame.Sample`.

```
def sample_n_per_group(n, *args, **kwargs):
    def sample_per_group(pdf):
        return pdf.sample(n, *args, **kwargs)
    return sample_per_group

df = spark.createDataFrame(
    [
        (1, 1.0),
        (1, 2.0),
        (2, 3.0),
        (2, 5.0),
        (2, 10.0)
        ],
        ("id", "v")
    )

    (df.groupBy("id")
       .applyInPandas(
            sample_n_per_group(1, random_state=2),
            schema=df.schema
       )
    )
```

To be aware of the limitations for very large groups, from the documentation:

> This function requires a full shuffle. All the data of a group will be loaded into memory, so the user should be aware of the potential OOM risk if data is skewed and certain groups are too large to fit in memory.

Share Edit Follow

edited Oct 12, 2021 at 11:59

answered Oct 12, 2021 at 11:38

s_pike
1,346 10 21