Handling missing values and Imputer class in Pyspark MLlib library

All About The Data — Read time: 4 minutes

When you create machine learning models, one of the enemies of making a good model is missing data. It can impact the model results or sometimes make it even impossible to create any good model at all.

Missing data means basically that we lack some information about an observation in the dataset.

How in general we can deal with that?

First thing is to check what actually missing value means. It will always depend on specific dataset.

Sometimes it can mean that we really miss some information, it wasn't collected for some reason.

However, sometimes it can in fact be meaningful, just the representation in the data is not the best in terms of data modelling. For example, if we have a dataset about houses and column indicating whether a house has a pool contains distinct values of 1 or NaN values, the NaN value can actually mean that a house doesn't have a pool and then we can simply replace NaN with 0.

So it can be that using domain knowledge and dataset documentation, we can add meaning to these NaN values.

But what about if we cannot do that — we have NaN values and it seems like it's really something that is unknown and we don't have a knowledge of how to replace it?

There are few common ways of doing that.

1. **Delete the whole column.** This option can work if we have a column that misses a lot of values. If for example 90% of records doesn't have value in this column, the best way may be to simply delete it completely as probably it won't bring much information to the model
2. **Delete the row.** This option can be the proper one if we miss information just on very small percent of records in the dataset (especially if these few records miss information in many columns). If we can afford deleting these rows, without loosing a lot of data, then it can be the easiest approach — we can be sure we don't replace missing data with values that actually can be not truth
3. **Replace the missing values with calculated values**. If we cannot afford to delete rows with missing values, we don't want to drop columns completely and we don't have knowledge to assign values in a more reliable way, we can use statistics to assign values artificially. We can do that by:
   – using simple rules like assigning mean, median or mode to missing values (these metrics are calculated on rows that contain the data)
   – building some statistical / machine learning model that would predict the value

In this article we will focus on the first option for replacing the values — so using mean, median or mode to that — and we will see how to do that in PySpark using Imputer class.

Imputer class — MLlib library

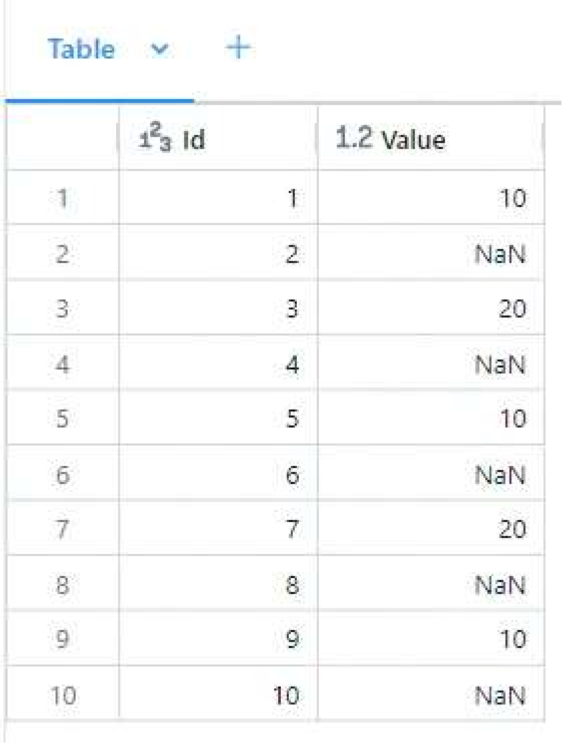
Although we could calculate basic statistics in quite a simple way using standard aggregation functions, *Imputer* class is simplifying things a little for us.

Imputer class follows Spark ML API standards, meaning it consists of estimator and transformer objects. So first we will fit our model of replacing the missing data into our frame and then we call the transform method to do the actual transformation.

The important thing to remember is that *Imputer* can accept only numeric values.

Let’s see how the syntax looks like and how to use it in an example. First we create a very simple Spark dataframe with Id and Value columns, where Value column have 50% of missing values.

```
df = spark.createDataFrame([(1, 10.0),
                             (2, float("nan")),
                             (3, 20.0),
                             (4, float("nan")),
                             (5, 10.0),
                             (6, float("nan")),
                             (7, 20.0),
                             (8, float("nan")),
                             (9, 10.0),
                             (10, float("nan")),
                             ]
                            ,
                            ["Id", "Value"])
```



Table

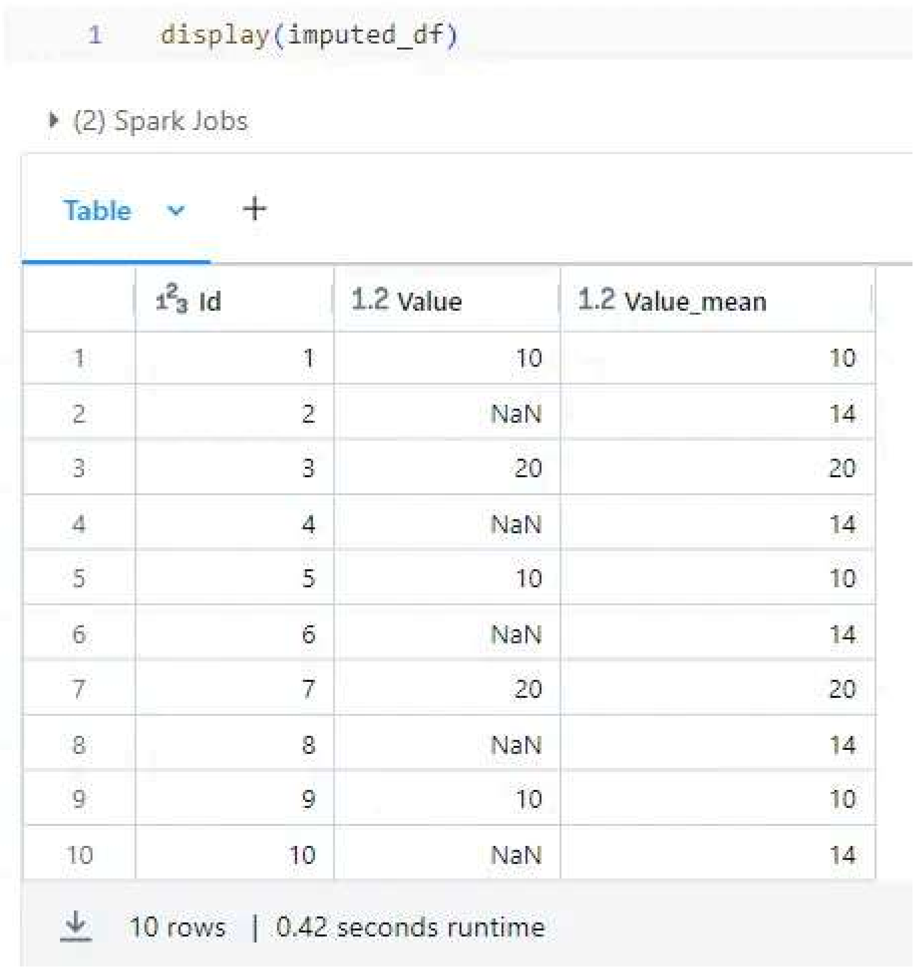
	Id	Value
1	1	10
2	2	NaN
3	3	20
4	4	NaN
5	5	10
6	6	NaN
7	7	20
8	8	NaN
9	9	10
10	10	NaN

```
from pyspark.ml.feature import Imputer

input_cols=['Value']
output_cols_mean=["Value_mean"]

i_mean = Imputer(strategy='mean', inputCols=input_cols,
outputCols=output_cols_mean)

imputer_model_mean = i_mean.fit(df)

imputed_df=imputer_model_mean.transform(df)
```

Our basic call for Imputer takes three input parameters:

- *strategy* — that defines whether we would like to replace missing values with mean, median and mode. The default is mean.
- *inputCols*— list of input columns for which we want to fill out missing values
- *outputCols* — list of names of output columns that will contain filled data

In our case, we want to fill out the *Value* column so we define 'Value' on the list for *input_cols* variable. The imputer will add new column with filled values which we decide to call *Value_mean*



```
display(imputed_df)
```

▸ (2) Spark Jobs

Table

	Id	Value	Value_mean
1	1	10	10
2	2	NaN	14
3	3	20	20
4	4	NaN	14
5	5	10	10
6	6	NaN	14
7	7	20	20
8	8	NaN	14
9	9	10	10
10	10	NaN	14

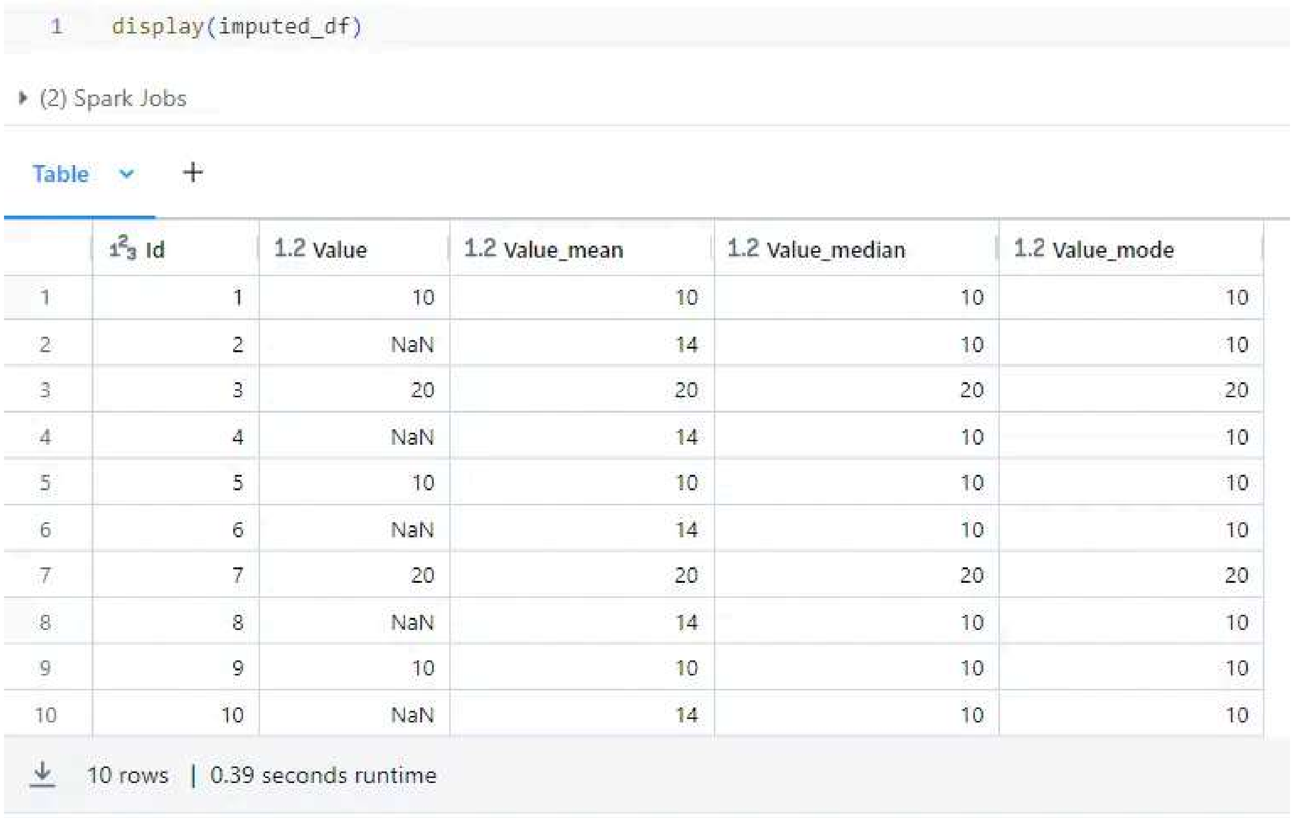
10 rows | 0.42 seconds runtime

As you can see, the missing values were replaced with value 14 as this the average of 10 + 20 + 10 + 20 + 10 / 5 =14

We can create other Imputer() instances with different strategies and add them to the dataframe to compare:

```
from pyspark.ml.feature import Imputer

input_cols=['Value']
output_cols_mean=["Value_mean"]
output_cols_median=["Value_median"]
output_cols_mode=["Value_mode"]
```

```
i_mean = Imputer(strategy='mean', inputCols=input_cols,
outputCols=output_cols_mean)
i_median = Imputer(strategy='median', inputCols=input_cols,
outputCols=output_cols_median)
i_mode = Imputer(strategy='mode', inputCols=input_cols,
outputCols=output_cols_mode)

imputer_model_mean = i_mean.fit(df)
imputer_model_median = i_median.fit(df)
imputer_model_mode = i_mode.fit(df)

imputed_df=imputer_model_mean.transform(df)
imputed_df=imputer_model_median.transform(imputed_df)
imputed_df=imputer_model_mode.transform(imputed_df)
```



```
display(imputed_df)
```

▸ (2) Spark Jobs

Table

	Id	Value	Value_mean	Value_median	Value_mode
1	1	10	10	10	10
2	2	NaN	14	10	10
3	3	20	20	20	20
4	4	NaN	14	10	10
5	5	10	10	10	10
6	6	NaN	14	10	10
7	7	20	20	20	20
8	8	NaN	14	10	10
9	9	10	10	10	10
10	10	NaN	14	10	10

10 rows | 0.39 seconds runtime

Both median and mode have value of 10 (calculated based on non-missing values: 10, 20, 10, 20, 10) so we get the same result for *Value_median* and *Value_mode* columns.

One thing that may come to your mind is that *Imputer* takes the whole data (all rows) into mean / median / mode calculations.

In practice, we may want to calculate these statistics within some groups. For example, if we had dataset that contains geography related data, we may want to replace missing values with statistics for corresponding country / region, not for the whole dataset, because even without deep domain knowledge, we may know that it will be more accurate.

In this case, the way to go would be to simply iterate through the dataset, creating subsets in a loop and creating instances of *Imputer* for each of them.