Get unlimited access

Published in Towards Data Science

Emma Grimaldi Follow

Oct 15, 2018 · 7 min read · Listen

Pandas vs. Spark: how to handle dataframes (Part II)

Get unlimited access

“Panda statues on gray concrete stairs during daytime” by chuttersnap on Unsplash. “Scala” means “stairway” in Italian, my native language: hence the choice of the picture. It just seemed appropriate.

with a second part, comparing how to handle dataframes in the two programming languages, in order to get the data ready before the modeling process. In Python, we will do all this by using Pandas library, while in Scala we will use Spark.

> *For this exercise, I will use the Titanic train dataset that can be easily downloaded at this link. Also, I do my Scala practices in Databricks: if you do so as well, remember to import your dataset first by clicking on Data and then Add Data.*

1. Read the dataframe

I will import and name my dataframe *df*, in **Python** this will be just two lines of code. This will work if you saved your *train.csv* in the same folder where your notebook is.

```
import pandas as pd
```

Scala will require more typing.

```
var df = sqlContext
    .read
    .format("csv")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("Filestore/tables/train.csv")
```

Let's see what's going on up here. Scala does not assume your dataset has a header, so we need to specify that. Also, Python will assign automatically a dtype to the dataframe columns, while Scala doesn't do so, unless we specify `.option("inferSchema", "true")`. Also notice that I did not import Spark Dataframe, because I practice Scala in Databricks, and it is preloaded. Otherwise we will need to do so.

> *Notice: booleans are capitalized in Python, while they are all lower-case in Scala!*

In **Python**, `df.head()` will show the first five rows by default: the output will look like this.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

df.head() output in Python.

If you want to see a number of rows different than five, you can just pass a different number in the parenthesis. **Scala**, with its `df.show()` ,will display the first 20 rows by default.

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22.0	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. ...	female	26.0	0	0	STON/O2. 3101282	7.925	null	S
4	1	1	Futrelle, Mrs. Ja...	female	35.0	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. Willia...	male	35.0	0	0	373450	8.05	null	S
6	0	3	Moran, Mr. James	male	null	0	0	330877	8.4583	null	Q
7	0	1	McCarthy, Mr. Tim...	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. ...	male	2.0	3	1	349909	21.075	null	S
9	1	3	Johnson, Mrs. Osc...	female	27.0	0	2	347742	11.1333	null	S
10	1	2	Nasser, Mrs. Nich...	female	14.0	1	0	237736	30.0708	null	C
11	1	3	Sandstrom, Miss. ...	female	4.0	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss. El...	female	58.0	0	0	113783	26.55	C103	S
13	0	3	Saundercock, Mr. ...	male	20.0	0	0	A/5. 2151	8.05	null	S
14	0	3	Andersson, Mr. An...	male	39.0	1	5	347082	31.275	null	S
15	0	3	Vestrom, Miss. Hu...	female	14.0	0	0	350406	7.8542	null	S
16	1	2	Hewlett, Mrs. (Ma...	female	55.0	0	0	248706	16.0	null	S
17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.125	null	Q
18	1	2	Williams, Mr. Cha...	male	null	0	0	244373	13.0	null	S
19	0	3	Vander Planke, Mr...	female	31.0	1	0	345763	18.0	null	S
20	1	3	Masselmani, Mrs. ...	female	null	0	0	2649	7.225	null	C

df.show() in Scala.

If we want to keep it shorter, and also get rid of the ellipsis in order to read the entire content of the columns, we can run `df.show(5, false)` .

3. Dataframe Columns and Dtypes

To retrieve the column names, in both cases we can just type `df.columns` :

Get unlimited access

If we want to check the dtypes, the command is again the same for both languages: `df.dtypes` . Pandas will return a Series object, while Scala will return an Array of tuples, each tuple containing respectively the name of the

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

column and the dtype. So, if we are in Python and we want to check what type is the *Age* column, we run `df.dtypes['Age']` , while in Scala we will need to filter and use the Tuple indexing: `df.dtypes.filter(colTup => colTup._1 == "Age")`.

4. Summary Statistics

This is another thing that every Data Scientist does while exploring his/her data: summary statistics. For every numerical column, we can see information such as count, mean, median, deviation, so on and so forth, to see immediately if there is something that doesn't look right. In both cases this will return a dataframe, where the columns are the numerical columns of the original dataframe, and the rows are the statistical values.

In **Python**, we type `df.describe()` , while in **Scala** `df.describe().show()` . The reason we have to add the `.show()` in the latter case, is because Scala doesn't output the resulting dataframe automatically, while Python does so (as long as we don't assign it to a new variable).

In **Python** we can use either `df[['Name','Survived]]` or `df.loc[:, ['Name','Survived]]` indistinctly. Remember that the `:` in this case means "all the rows".

In **Scala**, we will type `df.select("Name","Survived").show()`. If you want to assign the subset to a new variable, remember to omit the `.show()`.

6. Filtering

Let's say we want to have a look at the *Name* and *Pclass* of the passengers who survived. We will need to filter a condition on the *Survived* column and then select the the other ones.

In **Python**, we will use `.loc` again, by passing the filter in the rows place and then selecting the columns with a list. Basically like the example above but substituting the `:` with a filter, which means `df.loc[df['Survived'] == 1, ['Name','Pclass']]`.

In **Scala** we will use `filter` followed by `select` which will be

6.1. Filtering null values

If we want to check the null values, for example in the *Embarked* column, it will work like a normal filter, just with a different condition.

In **Python**, we apply the `.isnull()` when passing the condition, in this case `df[df['Embarked'].isnull()]`. Since we didn't specify any columns, this will return a dataframe will all the original columns, but only the rows where the *Embarked* values are empty.

In **Scala**, we will use `.filter` again: `df.filter("Embarked IS NULL").show()`. Notice that the boolean filters we pass in Scala, kind of look like SQL queries.

7. Imputing Null Values

We should always give some thought before imputing null values in a dataset, because it is something that will influence our final model and we want to be careful with that. However, just for demonstrative purposes, let's say we want to impute the string "N/A" to the null values in our dataframe.

In **Scala**, quite similarly, this would be achieved with `df = df.na.fill("N/A")`. Remember to not use the `.show()` in this case, because we are assigning the revised dataframe to a variable.

8. Renaming Columns

This is something that you will need to for sure in Scala, since the machine learning models will need two columns named *features* and *label* in order to be trained. However, this is something you might want to do also in Pandas if you don't like how a column has been named, for example. For this purpose, we want to change the *Survived* column into *label*.

In **Python** we will pass a dictionary, where the key and the value are respectively the old and the new name of the column. In this case, it will be `df.rename(columns = {"Survived": "label"}, inplace = True)`.

In **Scala**, this equals to `df = df.withColumnRenamed("Survived", "label")`.

maximum value; after `.groupby` we can use all sorts of aggregation functions: mean, count, median, so on and so forth. We stick with `.max()` for this example.

In **Python** this will be `df.groupby('Sex').mean()['Age']`. If we don't specify `['Age']` after `.mean()`, this will return a dataframe with the maximum values for all numerical columns, grouped by *Sex*.

In **Scala**, we will need to import the aggregation function we want to use, first.

```
import org.apache.spark.sql.functions.max
df.groupBy("Sex").agg(max("Age")).show()
```


10. Create a New Column

This is really useful for feature engineering, we might want to combine two variables to see how their interaction is related to the target. For purely demonstrative purpose, let's see how to create a column containing the

Get unlimited access

```
df['Age_times_Fare'] = df['Age'] * df['Fare']
```

In **Scala**, we will need to put `$` before the names of the columns we want to use, so that the column object with the corresponding name will be considered.

```
df = df.withColumn("AgeTimesFare", $"Age" * $"Fare")
```

11. Correlation

Exploring correlation among numerical variables and target is always convenient, and obtaining a matrix of correlation coefficients among all numeric variables is pretty easy in **Python**, just by running `df.corr()`. If you want to look at the correlation, let's say between *Age* and *Fare*, we will just need to specify the columns: `df[['Age','Fare']].corr()`.

In Scala, we will need to import first, and then run the command by specifying the columns.

Get unlimited access

This is it! I hope you found this post useful as much as it has been useful for me writing it. I intend to publish a Part III where I can walk through a machine learning model example to kind of complete the circle!

Feel free to check out:

my LinkedIn profile.

Thank you for reading!
