Lab 4.3: DataFrame Transformations

Overview

In this lab, we will perform a number of transformations on dataframes. This will provide exposure to the API, and some experience in using it.

We will cover a number of common techniques, but will not attempt to cover all of the API, or try to illustrate every possible technique. The API is too large for that.

We'll be working at the Zeppelin shell for all of these.

Builds on

Lab 4.2

Run time

30-40 minutes

Python-Based Column Access

Python has convenient syntax for accessing a column in a DataFrame (di**!**erent from Scala syntax). Assuming you have a DataFrame named `folksDF` , that has a column named `age` , `folksDF.age` will access this column. So to filter on age, you can use the Column expression below

```
# Python
> folksDF.filter(folksDF.age > 25)
```

Note that if using an SQL expression, you can pass it in string form exactly as for Scala, e.g.

```
# Python
> folksDF.filter("age > 25")
```

Note also, that if combining boolean expressions, you may need to explicitly parenthesize due to Python syntax requirements. For example, to filter for `age>25` and `age<50` , you could use the below. Note the parentheses around each boolean expression, without which you would get syntax errors.

```
# Python
folksDF.filter((folksDF.age>25) & (folksDF.age<50)).show()
```

Lastly, note that you use `==` for equality of a Column value in Python (as opposed to `===` in Scala)

Some Simple Transformations

Tasks

- Read *people.json* if you haven't already, then perform the transformations listed below.

```
// Scala
> val folksDF=spark.read.json("spark-labs/data/people.json")
```

-

```
# Python
> folksDF=spark.read.json("spark-labs/data/people.json")
```

  Display the data.

- Filter on age and display the data. Filter
- on gender and display the data.
- Count how many "F" and "M" items there are.
- **[Optional]** Find the oldest person with gender "F".
  - Use SQL to write this query - e.g. `spark.sql(" ... ")`
  - Remember to register a temporaty table. Use a
  - subquery to find the maximum age.

Working with More Complex Data

Tasks

- Read *github.json* if you haven't already

```
// Scala
> val githubDF=spark.read.json("spark-labs/data/github.json")
```

```
# Python
> githubDF=spark.read.json("spark-labs/data/github.json")
```

- Look at the schema and 5 sample rows (`limit(5).show`) again.

    - It's hard to understand and view because there is so much data in a single row.
- Select the actor column, Store this dataframe in a separate variable.

```
// Scala
> val actorDF = // ...
```

```
# Python
 > actorDF = // ...
```

- View this dataframe's schema, then display 5 sample rows.
- When using show, pass in a false, so the display is not truncated e.g.
    - **Scala:** `show(false)`
    - **Python:** `show(truncate=False)`
- There is less data to view, so it's easier to see the details of the actor column now, along with its nested structure .
    - i.e. the avatar_url, login, and other data. Select
- the login value of the actor and display it.
    - Use dot notation for nested elements. Again,
    - only display a few rows.
- Find out how many unique logins there are in the data.
    - Not sure how to do this? **Look at the documentation!**
- Each row in this data, contains a `type` column.
    - Display all the unique values for the 'type' column.
    - Not sure how to do this? **Look at the documentation!**
- Determine how may rows have a type of `CreateEvent`
    - Retrieve five of these rows

Working with Text Data

Tasks

- Read *wiki-pageviews.txt* if you haven't already Show 5
- rows from this data so you can examine it

```
// Scala
> val viewsDF=spark.read.text("spark-labs/data/wiki-pageviews.txt")
viewsDF: org.apache.spark.sql.DataFrame = [value: string]
> viewsDF.limit(5).show

+------------------+
|             value|
+------------------+
|  aa Main_Page 3 0|
|  aa Main_page 1 0|
|  aa User:Savh 1 0|
|  aa Wikipedia 1 0|
|aa.b User:Savh 1 0|
+------------------+
```

```
# Python
> viewsDF=spark.read.text("spark-labs/data/wiki-pageviews.txt")
> viewsDF.limit(5).show()

+------------------+
|             value|
+------------------+
|  aa Main_Page 3 0|
|  aa Main_page 1 0|
|  aa User:Savh 1 0|
|  aa Wikipedia 1 0|
|aa.b User:Savh 1 0|
+------------------+
```

- We can see that each row contains a single string.
    - Unlike JSON data, there is no automatic application of a schema.
    - This is a cumbersome format to work with, and we'll see ways to transform it into an easier to use format later.

Summary

We've practiced using some of the common transformations that are available for dataframes. We'll continue using this data to delve into Spark's capabilities.


STOP