Spark Project - 4

Spark SQL

In this project, we will use Spark's SQL interface to query and interact with the data.

About the dataset:

We will still be using the JSON file census_2010.json containing the data from the 2010 U.S. Census. It has been taken from the Machine Learning Library of UC Irvine: https://archive.ics.uci.edu/ml/datasets/Census+Income

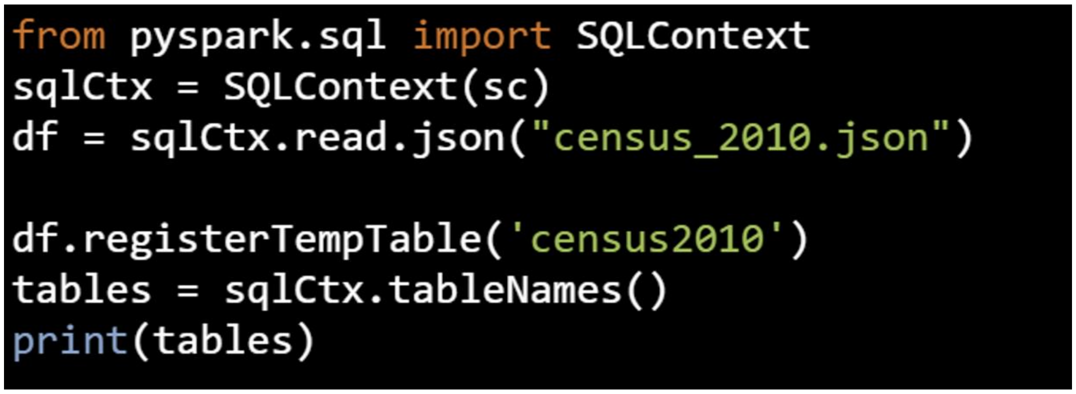
It has the following columns:

- age - Age (year)
- females - Number of females
- males - Number of males
- total - Total number of individuals
- year - Year column (2010 for all rows)

Later, we will be adding other census data json files like census_1980.json, census_1990.json and census_2000.json, just to see the power of this particular interface of Spark.

1. Before we can write and run SQL queries, we need to tell Spark to treat the DataFrame as a SQL table. Spark internally maintains a virtual database within the SQLContext object. This object, which we enter as sqlCtx, has methods for registering temporary tables.

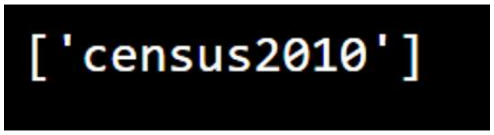
To register a DataFrame as a table, we will call the registerTempTable() method on that DataFrame object. This method requires one string parameter, name, that we use to set the table name for reference in our SQL queries.



```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
df = sqlCtx.read.json("census_2010.json")

df.registerTempTable('census2010')
tables = sqlCtx.tableNames()
print(tables)
```

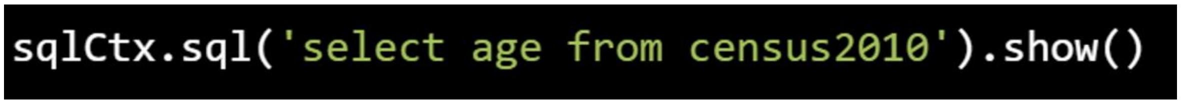
The output will just be the name of the table:



```
['census2010']
```

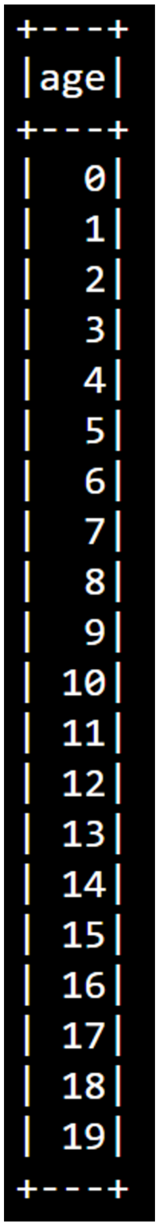
2. Now that we've registered the table within sqlCtx, we can start writing and running SQL queries. With Spark SQL, we represent our query as a string and pass it into the sql() method within the SQLContext object. The sql() method requires a single parameter, the query string. Spark will return the query results as a DataFrame object. This means we'll have to use show() to display the results, due to lazy loading.

We will write a query that returns the age column from the table census2010 and then use show() method to display the first 20 results.



```
sqlCtx.sql('select age from census2010').show()
```

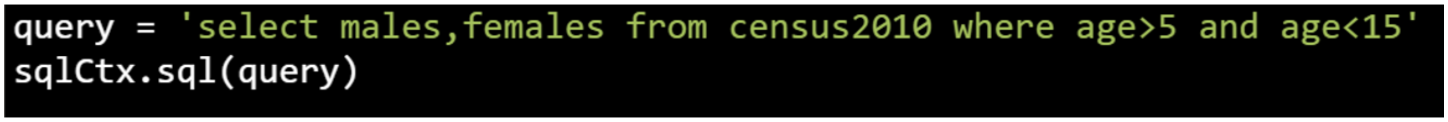
The output will show the first 20 rows of the age column of the table:



```
+---+
|age|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
|  5|
|  6|
|  7|
|  8|
|  9|
| 10|
| 11|
| 12|
| 13|
| 14|
| 15|
| 16|
| 17|
| 18|
| 19|
+---+
```

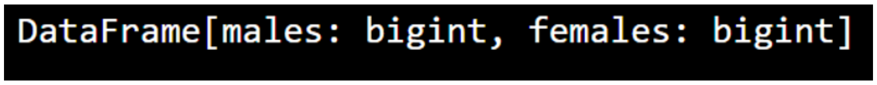
3. In my previous project, we used DataFrame methods to find all of the rows where age was greater than 5. If we only wanted to retrieve data from the males and females columns where that criteria were true, we'd need to chain additional operations to the Spark DataFrame. To return the results in descending order instead of ascending order, we'd have to chain another method. The DataFrame methods are quick and powerful for simple queries, but chaining them can be cumbersome for more advanced queries. SQL shines at expressing complex logic in a more compact manner.

We will write a query which will return the males and females column data where the age range is between 5 to 15.



```
query = 'select males,females from census2010 where age>5 and age<15'
sqlCtx.sql(query)
```

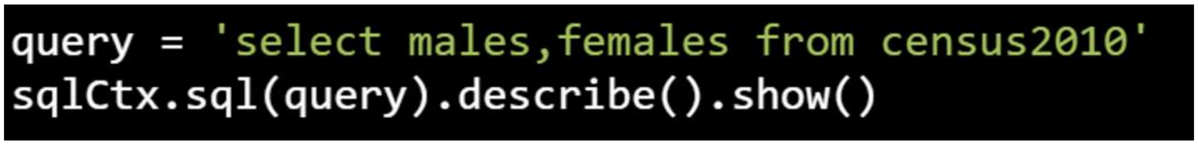
As the output we will see the data type returned:



```
DataFrame[males: bigint, females: bigint]
```

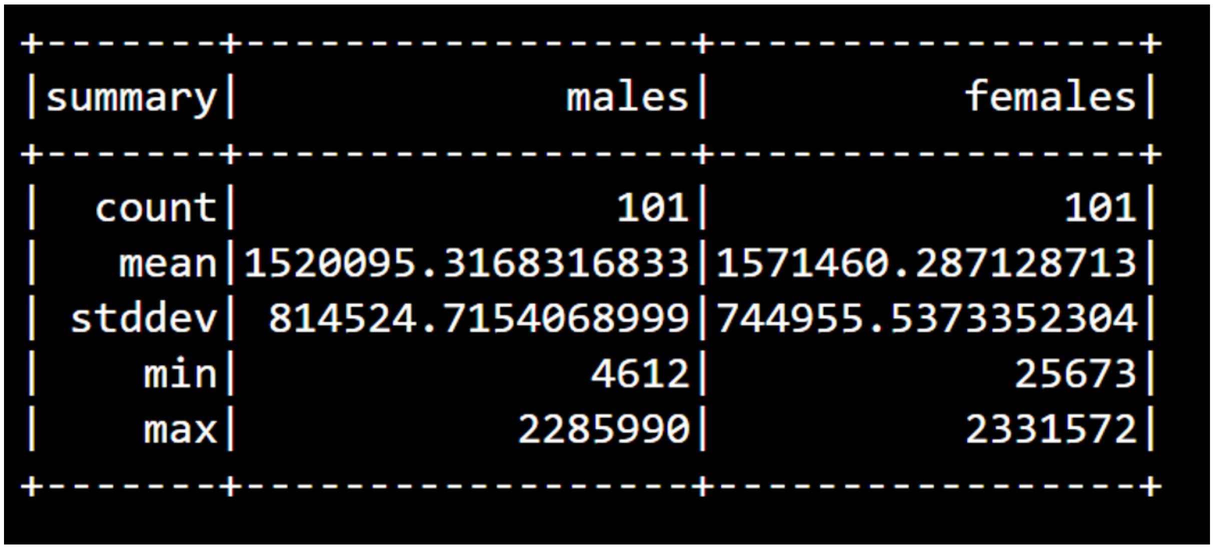
4. Because the results of SQL queries are DataFrame objects, we can combine the best aspects of both DataFrames and SQL to enhance our workflow. For example, we can write a SQL query that quickly returns a subset of our data as a DataFrame.

We will write an SQL query that will simply return the males and females columns from the census2010 table. But this time we will use the describe() method to calculate the summary statistics and show() method to display the results.



```
query = 'select males,females from census2010'
sqlCtx.sql(query).describe().show()
```

With the output that shows us the statistics of the data collected by the query:



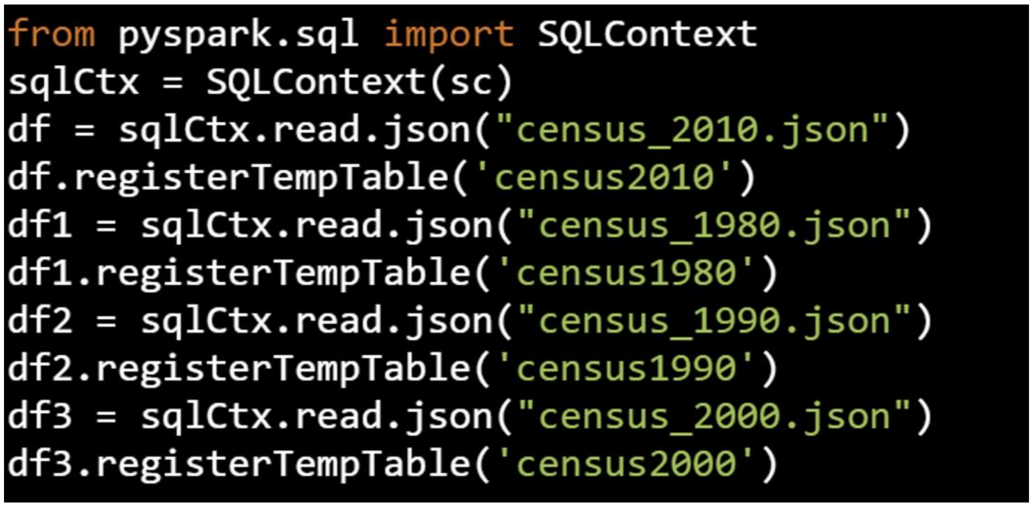
```
+-------+------------------+-----------------+
|summary|             males|          females|
+-------+------------------+-----------------+
|  count|               101|              101|
|   mean|1520095.3168316833|1571460.287128713|
| stddev| 814524.7154068999|744955.5373352304|
|    min|              4612|            25673|
|    max|           2285990|          2331572|
+-------+------------------+-----------------+
```

5. One of the most powerful use cases in SQL is joining tables. Spark SQL takes this a step further by enabling us to run join queries across data from multiple file types. Spark will read any of the file types and formats it supports into DataFrame objects and we can register each of these as tables within the SQLContext object to use for querying.

This enables data professionals to use one common query language, SQL, to interact with lots of different data sources, which is the industry scenario today. The other datasets we'll be using are:

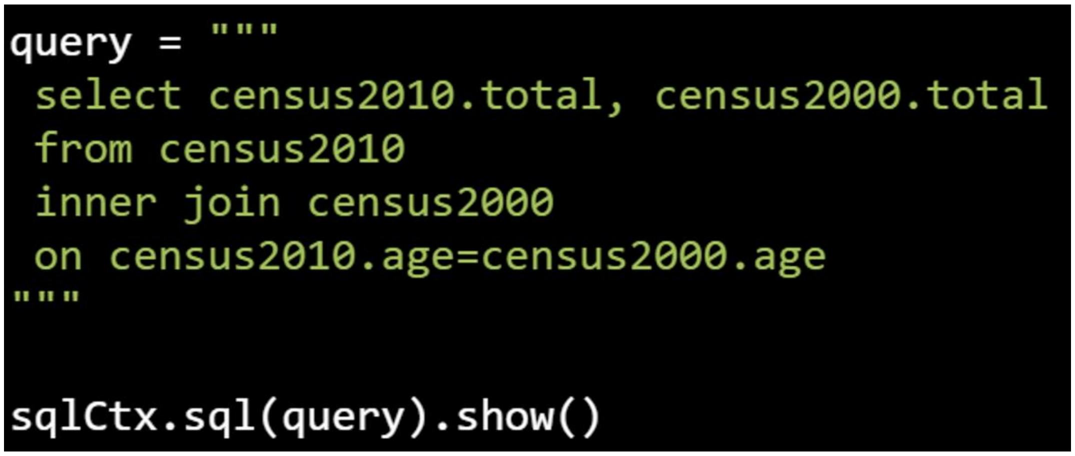
- census_1980.json - 1980 U.S. Census data
- census_1990.json - 1990 U.S. Census data
- census_2000.json - 2000 U.S. Census data

We will first assign these files to their respective dataframes and the assign them to SQL using SQL context.



```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
df = sqlCtx.read.json("census_2010.json")
df.registerTempTable('census2010')
df1 = sqlCtx.read.json("census_1980.json")
df1.registerTempTable('census1980')
df2 = sqlCtx.read.json("census_1990.json")
df2.registerTempTable('census1990')
df3 = sqlCtx.read.json("census_2000.json")
df3.registerTempTable('census2000')
```

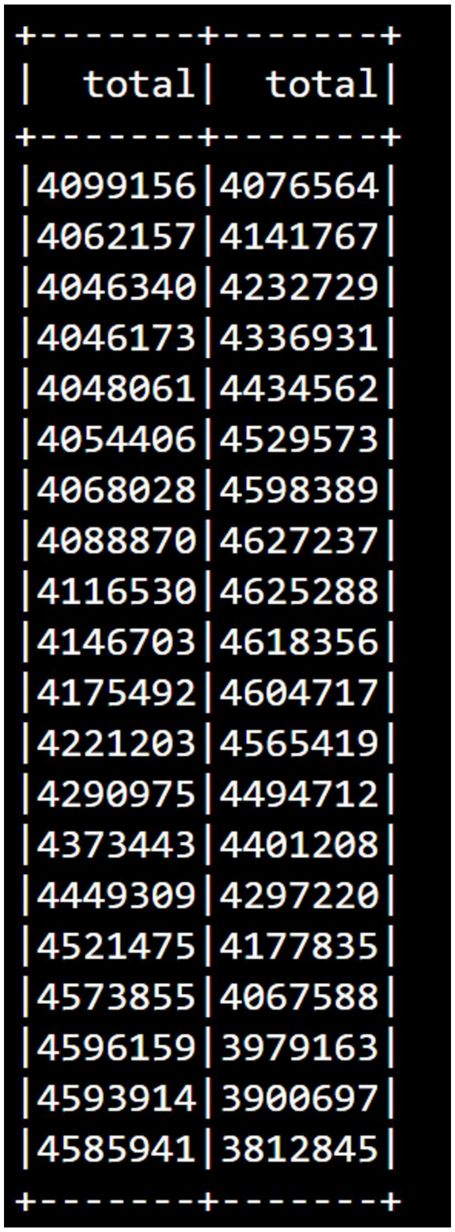
6. We can then perform join operations on all of our datasets using SQL queries. We will use the age column as the joining column.



```
query = """
 select census2010.total, census2000.total
 from census2010
 inner join census2000
 on census2010.age=census2000.age
"""

sqlCtx.sql(query).show()
```

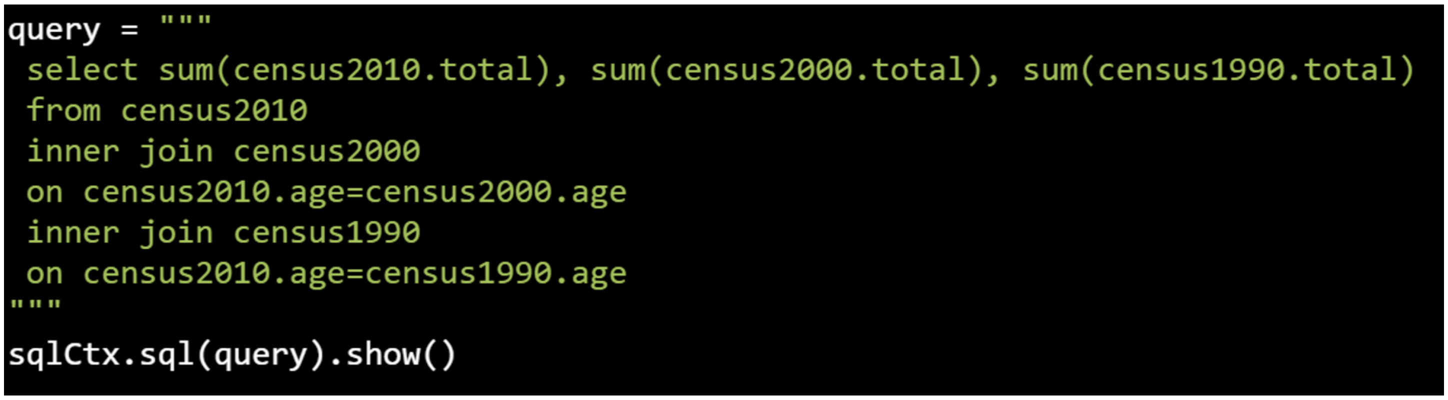
As output, we will see the first 20 columns of the join:



```
+-------+-------+
|  total|  total|
+-------+-------+
|4099156|4076564|
|4062157|4141767|
|4046340|4232729|
|4046173|4336931|
|4048061|4434562|
|4054406|4529573|
|4068028|4598389|
|4088870|4627237|
|4116530|4625288|
|4146703|4618356|
|4175492|4604717|
|4221203|4565419|
|4290975|4494712|
|4373443|4401208|
|4449309|4297220|
|4521475|4177835|
|4573855|4067588|
|4596159|3979163|
|4593914|3900697|
|4585941|3812845|
+-------+-------+
```

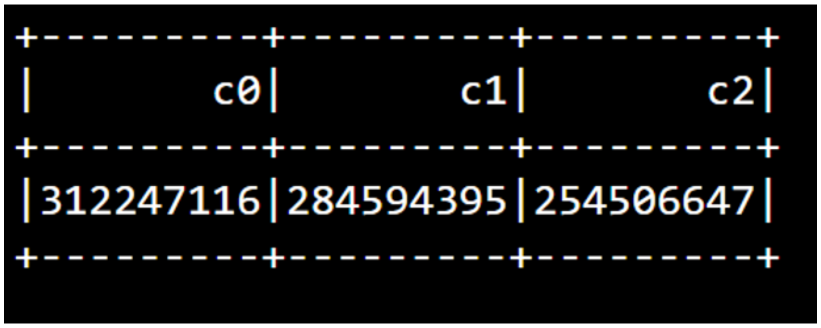
7.Because it is SQL, we can perform many of the aggregation functions on the datasets which was not possible earlier like count(), avg(), sum(), AND or OR.

We will calculate the sums of the ‘total’ column from each of the tables by first performing inner joins using age as the joining column and display it.



```
query = """
 select sum(census2010.total), sum(census2000.total), sum(census1990.total)
 from census2010
 inner join census2000
 on census2010.age=census2000.age
 inner join census1990
 on census2010.age=census1990.age
"""
sqlCtx.sql(query).show()
```

We can see the sum of the columns in the same order as the selection:



```
+---------+---------+---------+
|       c0|       c1|       c2|
+---------+---------+---------+
|312247116|284594395|254506647|
+---------+---------+---------+
```

With this project, we learnt quite a lot of Spark SQL interface and some of its functionalities.