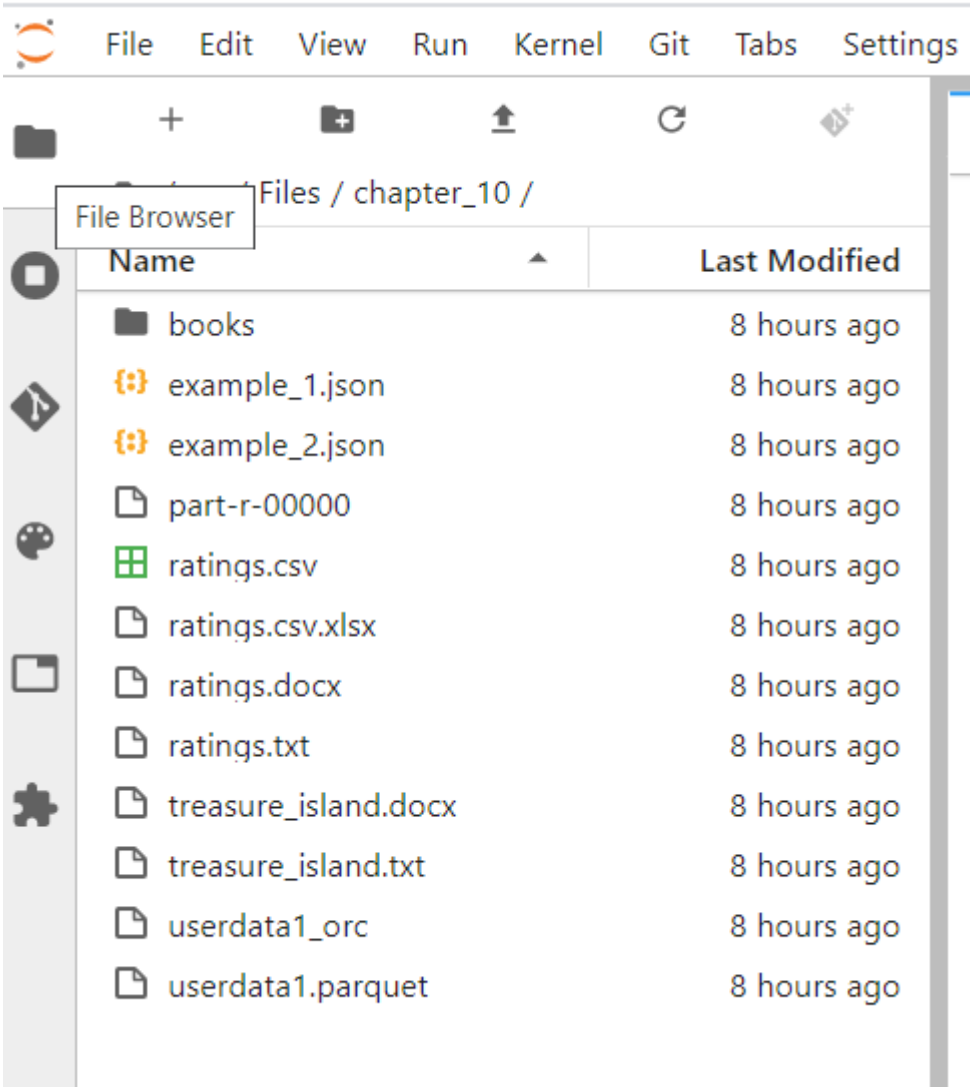
Lab : Apache Spark File Formats - CSV and JSON

Pre-reqs:

- Google Chrome (Recommended)
- Start IntelliJ IDE and open `~/work/ernesto-spark/Files/chapter_10` to view files.



The aim of the following lab exercises is to read and write various file formats in Spark applications. We will cover following topics in this scenario.

- CSV Files
- JSON Files

Prerequisites

We need following packages to perform the lab exercise:

- Java Development Kit
- pyspark

JAVA

Verify the installation with: `java -version`

You'll see the following output:

```
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
```

Install pyspark

Note: Spark is already installed. It is not required to run following command to install

PySpark is available in pypi. To install just run `pip install pyspark`

CSV Files

Similar to the text files we can also save the CSV files using the DataSource API. Each line in the CSV file represents a single record. CSV contains a large number of options when it comes to processing them. With these options we can efficiently handle tricky scenarios such as corrupt records etc.

CSV files support compression, are human readable and are splittable. However, CSV files are not nestable and cannot hold complex data structures. The following is an example to read a CSV file.

Task: CSV Files

Let us now look at reading and writing CSV files to Spark. We have been reading and writing CSV files in the previous chapters. However, let us also see some of many options that can be used while reading and writing CSV files.

Step 1: Download the file ratings.csv from the URL below and save it to the /headless/Desktop/ernesto-spark/Files/chapter_10 folder.

ratings.csv - http://bit.ly/2L8IEBS

Note: We already have cloned a github repository which contains a required file. Open `~/work/ernesto-spark/Files/chapter_10` to view file.

Each line of this file represents one rating of one movie by one user, and has the following format: userId, movieId, rating, timestamp

Step 2: Let us now read this file to Spark from Spark shell using few options.

Open the terminal and fire up the Spark shell `spark-shell`.

Enter into the paste mode and execute the following code. `:paste`

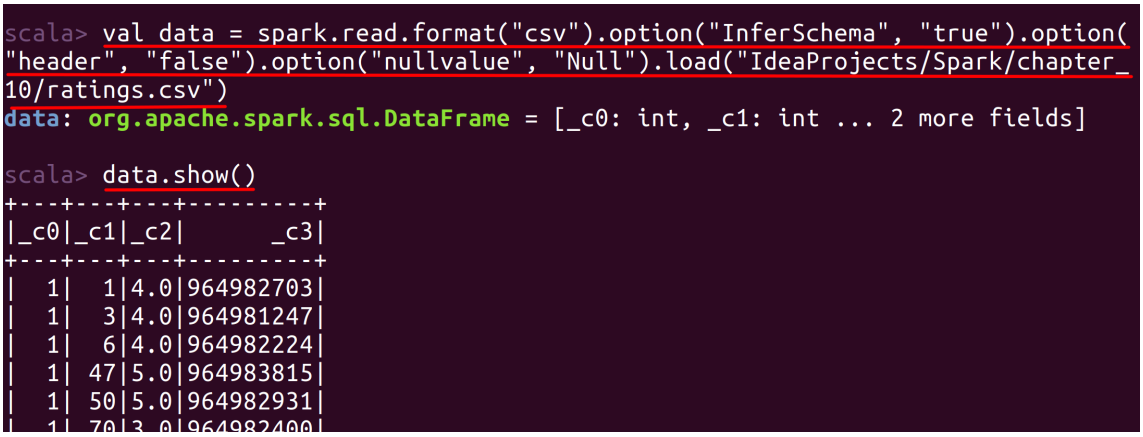
Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val data = spark
.read
.format("csv")
.option("InferSchema", "true")
.option("header", "false")
```

```
.option("nullValue", "Null")
.load("/headless/Desktop/ernesto-spark/Files/chapter_10/ratings.csv")
```

We have used a new option here which is called NullValue. This will replace all the null values with the provided string, which is Null in this case. The default is "". Please check the references section for all the options that can be used while reading or writing CSV files. All the options can be used in this way or inside a map object.

We can then call the show method as shown in the screenshot below to check if it was successful.



Step 3: We can also use the modes we have learned in our theory. Let us see an example.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  dataNew = spark
.read
.format("csv")
.options(Map("InferSchema" -> "true"
, "header" -> "false"
, "nullValue" -> "Null"
, "mode" -> "FAILFAST"))
.load("/headless/Desktop/ernesto-spark/Files/chapter_10/ratings.csv")
```

```
dataNew.show()
```

```
scala> val dataNew = spark.read.format("csv").options(Map("InferSchema" -> "true
", "header" -> "false", "nullvalue" -> "Null", "mode" -> "FAILFAST")).load("Idea
Projects/Spark/chapter_10/ratings.csv")
dataNew: org.apache.spark.sql.DataFrame = [_c0: int, _c1: int ... 2 more fields]

scala> dataNew.show()
+---+---+---+---------+
|_c0|_c1|_c2|      _c3|
+---+---+---+---------+
|  1|  1|4.0|964982703|
|  1|  3|4.0|964981247|
|  1|  6|4.0|964982224|
|  1| 47|5.0|964983815|
|  1| 50|5.0|964982931|
|  1| 70|3.0|964982400|
|  1|101|5.0|964980868|
|  1|110|4.0|964982176|
```

Step 4: Let us now write this dataframe back to the filesystem in CSV format.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
dataNew.write.format("csv").option("sep", "|").save("/headless/Desktop/ernesto-spark/Files/chapter_10/output2")
```

Here, we have used an option called sep which replaces the delimiter from comma to a pipe.

Step 5: Let us check if the save was successful as we desired.

```
cat /headless/Desktop/ernesto-spark/Files/chapter_10/output2/part*
```

Run above command in **terminal 2**. You can also open New terminal by Clicking `File` > `New` > `Terminal` from the top menu.

Task is complete!

Task: JSON Files

Similar to previous taks, let us read and write JSON files. We shall be reading two kinds of JSON files. One is a single line JSON and other is the multi line JSON.

JSON is also one of the popular file formats around which stands for JavaScrit Object Notation. JSON is compressable, splittable and human readable. It is also nested and supports complex data structures. With Spark, we can load a single line JSON and also a multi-line JSON. All we need to do is specify an option for multi-line JSON. However, it is recommend to use single line JSON whenever possible.

Step 1: Download the file example_1.json from the URL below and save it to the /headless/Desktop/ernesto-spark/Files/chapter_10 folder.

example_1.json - http://bit.ly/2IRFI06

Note: We already have cloned a github repository which contains a required file. Open `~/work/ernesto-spark/Files/chapter_10` to view file.

Step 2: The following code is used to read the single line JSON file.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  jsonData = spark.read
.format("json")
.option("multiLine", "false")
.load("/headless/Desktop/ernesto-spark/Files/chapter_10/example_1.json")
```

Step 3: Let us check if we were able to load the JSON file successfully.

`jsonData.show()`

```
scala> val jsonData = spark.read.format("json").option("multiLine", "false").loa
d("IdeaProjects/Spark/chapter_10/example_1.json")
jsonData: org.apache.spark.sql.DataFrame = [color: string, fruit: string ... 1 m
ore field]

scala> jsonData.show()
+-----+-----+-----+
|color|fruit| size|
+-----+-----+-----+
|  Red|Apple|Large|
+-----+-----+-----+
```

Step 4: Let us now load the multi line JSON file. Download the file example_2.json from the URL below and save it to the /headless/Desktop/ernesto-spark/Files/chapter_10 folder.

example_2.json - http://bit.ly/2IL3IST

Note: We already have cloned a github repository which contains a required file. Open `~/work/ernesto-spark/Files/chapter_10` to view file.

Step 5: The following code is used to read the single line JSON file.

Enter into the paste mode and execute the following code. `:paste`

Note: After pasting following code in the scala terminal, Press `Ctrl` + `D` to run code.

```
val  multiJson = spark.read
    .format("json")
    .option("multiLine", "true")
    .load("/headless/Desktop/ernesto-spark/Files/chapter_10/example_2.json")
```

Step 6: Let us now write this dataframe to the filsesystem.

`multiJson.write.format("json").save("/headless/Desktop/ernesto-spark/Files/chapter_10/output3")`

Output

You can check the output by running the following command from a new terminal.

`cat /headless/Desktop/ernesto-spark/Files/chapter_10/output3/part*`

Run above command in **terminal 2**. You can also open New terminal by Clicking `File` > `New` > `Terminal` from the top menu.

Task is complete!