By Sai Kumar

CREATING DATAFRAME (USING json, csv and parquet FILES)

NOTE: You can execute all exercises on Scala or Python, it will be mentioned as "On Scala" for Scala and "On Python" for Python.

First steps to perform

Put the devices.json, ratings.csv and base_stations.parquet files into hdfs.
(Copy the above files into desktop of the VM machine)

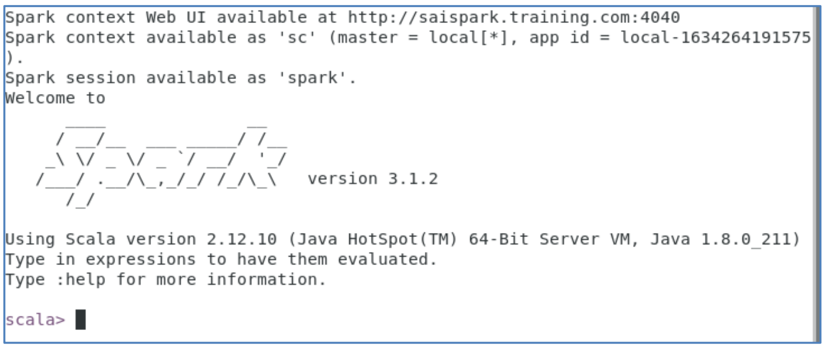
```
[root@saispark ~]# cd   Desktop/

[root@saispark Desktop]# hdfs  dfs   -mkdir    /user/root/jsons

[root@saispark Desktop]# hdfs  dfs   -mkdir   /user/root/csvs

[root@saispark Desktop]# hdfs  dfs   -mkdir   /user/root/parquet

[root@saispark Desktop]# hdfs  dfs   -put   devices.json
/user/root/jsons

[root@saispark Desktop]# hdfs  dfs   -put   ratings.csv
/user/root/csvs

[root@saispark Desktop]# hdfs d fs   -put   base_stations.parquet
/user/root/parquet
```

THE SPARK SHELL

1. Login to Spark shell (Scala or Python) and check "spark" session and executing Linux commands from Spark shell.

On Scala:

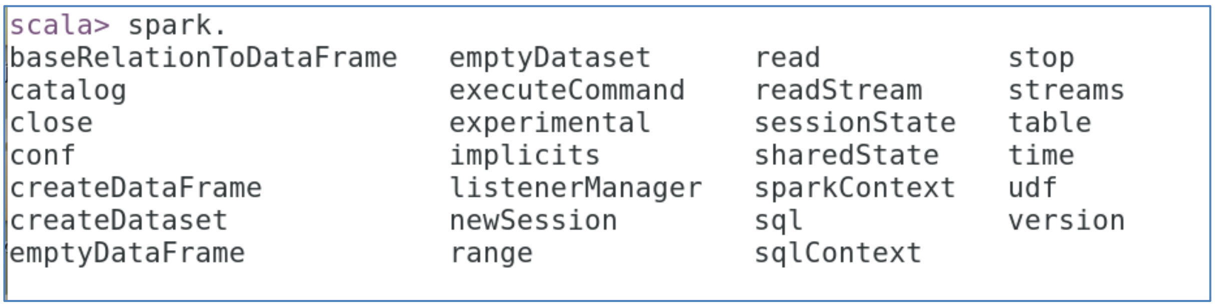
```
[root@saispark ~]# spark-shell --master=yarn
```



```
Spark context Web UI available at http://saispark.training.com:4040
Spark context available as 'sc' (master = local[*], app id = local-1634264191575
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_211)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

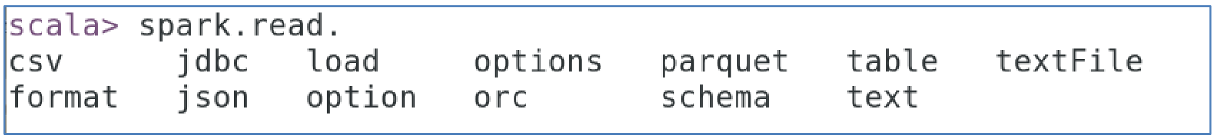
2. Spark creates a SparkSession object for you called spark.
scala> **spark** (Press Enter)
res0: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@5abfb698

3. Using command completion, you can see all the available Spark session methods:
   type **spark.** (spark followed by a dot) and then the TAB key.



```
scala> spark.
baseRelationToDataFrame   emptyDataset      read           stop
catalog                   executeCommand    readStream     streams
close                     experimental      sessionState   table
conf                      implicits         sharedState    time
createDataFrame           listenerManager   sparkContext   udf
createDataset             newSession        sql            version
emptyDataFrame            range             sqlContext
```

scala> **spark.read.** <Press Tab>

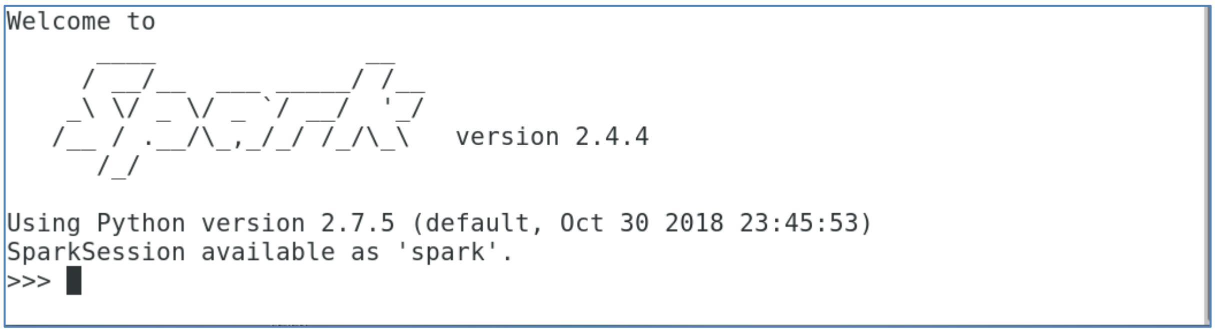


```
scala> spark.read.
csv      jdbc    load     options    parquet    table    textFile
format   json    option   orc        schema     text
```

*Type **sys.exit()** to close scala shell.*

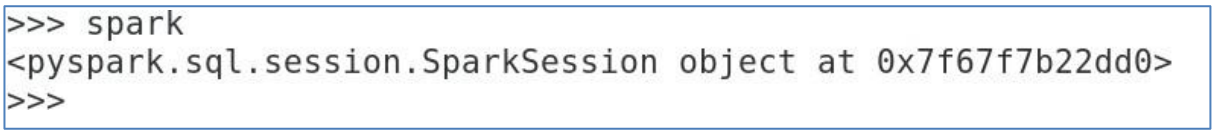
On Python

[root@saispark ~]# **pyspark --master=yarn**



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Python version 2.7.5 (default, Oct 30 2018 23:45:53)
SparkSession available as 'spark'.
>>> 
```

1. Type **spark** and press enter



```
>>> spark
<pyspark.sql.session.SparkSession object at 0x7f67f7b22dd0>
>>>
```

2. >>> **quit()**

HOW TO RUN LINUX / HDFS COMMANDS FROM SCALA

spark-shell --master=yarn

```
scala> import    sys.process._

scala> "ls".!

scala> "hdfs  dfs   -ls    /user/root".!
```

HOW TO RUN LINUX / HDFS COMMANDS FROM PYTHON

pyspark --master=yarn

```
>>> import  os

>>> os.system("ls")

>>> os.system("hdfs dfs -ls /user/root/")
```

CREATING DATAFRAME (DEFAULT SCHEMA ➔ SCHEMA INHERITED BY SPARK)

1. From Csv File

On Scala

Read the file (Please type continuously)

```
scala> val
ratingsdf=spark.read.csv("/user/root/csvs/ratings.csv")
```

Print the schema

```
scala> ratingsdf.printSchema
```

Read the top 20 rows

```
scala> ratingsdf.show()
```

On Python

Read the file

```
>>>
ratingsdf=spark.read.csv("/user/root/csvs/ratings.csv")
```

Print the schema

```
>>> ratingsdf.printSchema()
```

Read the top 20 rows
>>> **ratingsdf.show()**

2. Repeat the same for json file as above using "**spark.read.json**" method.
   File: **/user/root/jsons/devices.json**

3. Using Parquet file
   Examine the parquet file using parquet-tools

   [root@saispark Desktop]**# parquet-tools head base_stations.parquet**

   [root@saispark Desktop]# **parquet-tools schema base_stations.parquet**

   Now from Spark use "**spark.read.parquet**" and read the file, print schema and display the data.

   File: **/user/root/parquet/base_stations.parquet**

USING THE HEADER OPTION FOR CSV FILE

In ratings.csv file the first line is header, as you have observed in earlier exercise Spark has not taken the header and given its own column names. Using format and option methods we will instruct spark to take the first line as header of ratings.csv file.

On Scala

Please type on continuous line.
scala> **val ratingsdf=spark.read.format("csv").option("header","true").load("/user/root/csvs/ratings.csv")**

scala> **ratingsdf.printSchema**

scala> **ratingsdf.show**

On Python

```
>>> 
ratingsdf=spark.read.format("csv").option("header","true").
load("/user/root/csvs/ratings.csv")

>>> ratingsdf.printSchema()

>>> ratingsdf.show()
```

NOTE: The option "header" is case sensitive must be in lower case, however the value "true" can be in any case.

JUST FOR EXPRIENCING

If your spark is running on remote machine and HDFS is on another server then we have to use URI to read the data.

Ex: scala> **val devicesdf=spark.read.json("hdfs://saispark.training.com:8020/user/root/jsons/devices.json")**

It works the same way in Python.

CREATING DATAFRAME FROM MEMORY

On Scala

```
scala> val mydata = List(("sai","kumar"),("sam","tom"))
scala> val myDF= spark.createDataFrame(mydata)
scala> myDF.show
```

On Python

```
>>> mydata = 
[{"NAME":"sai","AGE":20},{"NAME":"sam","AGE":15}]
>>> mydf=spark.createDataFrame(mydata)
>>> mydf.printSchema()
>>> mydf.show()
```

DATAFRAME ACTION COMMANDS

Below action commands prints the result to terminal, do not save the result.

Note: Syntax is same both on Scala and Python

1. **COUNT:** Returns the number of rows.
   devicesdf.count()

2. **FIRST:** Returns the first row(synonym for head())
   devicesdf.first()

3. **TAKE:** Returns the first n rows in array. (synonym for head(n))
   devicesdf.take(4)

4. **SHOW:** Display the first n rows in tabular form (Default 20 rows)
   devicesdf.show()
   devicesdf.show(10)

5. **COLLECT:** Returns all the rows in the dataFrame as an array
   devicesdf.collect()

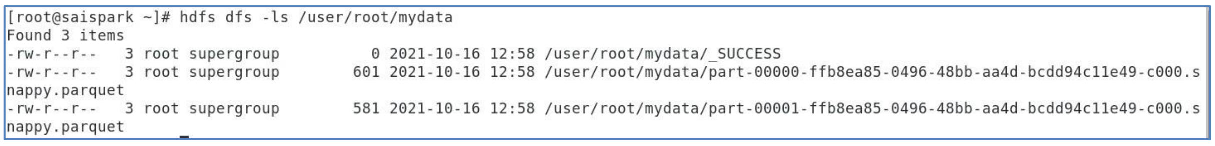
DATAFRAME WRITE

Using write method we can save the result to a file or a data source.

1. Writing data as default parquet format

On Scala

scala> **myDF.write.save("/user/root/mydata")**

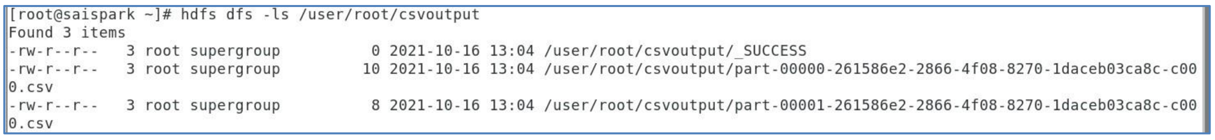


```
[root@saispark ~]# hdfs dfs -ls /user/root/mydata
Found 3 items
-rw-r--r--   3 root supergroup          0 2021-10-16 12:58 /user/root/mydata/_SUCCESS
-rw-r--r--   3 root supergroup        601 2021-10-16 12:58 /user/root/mydata/part-00000-ffb8ea85-0496-48bb-aa4d-bcdd94c11e49-c000.s
nappy.parquet
-rw-r--r--   3 root supergroup        581 2021-10-16 12:58 /user/root/mydata/part-00001-ffb8ea85-0496-48bb-aa4d-bcdd94c11e49-c000.s
nappy.parquet
```

NOTE: Default storage format from Spark 2 is Parquet, and we must specify a directory (must not exist)

2. Saving in same directory (Existing)
   scala> **myDF.write.mode("append").save("/user/root/mydata")**

3. Saving in other File format.
   scala> **myDF.write.csv("/user/root/csvoutput")**



```
[root@saispark ~]# hdfs dfs -ls /user/root/csvoutput
Found 3 items
-rw-r--r--   3 root supergroup          0 2021-10-16 13:04 /user/root/csvoutput/_SUCCESS
-rw-r--r--   3 root supergroup         10 2021-10-16 13:04 /user/root/csvoutput/part-00000-261586e2-2866-4f08-8270-1daceb03ca8c-c00
0.csv
-rw-r--r--   3 root supergroup          8 2021-10-16 13:04 /user/root/csvoutput/part-00001-261586e2-2866-4f08-8270-1daceb03ca8c-c00
0.csv
```

************* ***Happy Learning*** *************