**Environment Setup**

**Download the Dataset**

In this assignmentl you need to download two files, people.txt and people.json into your Sandbox's **tmp** folder. The commands below should be typed into Shell-in-a-Box

1. Assuming you start as **root** user:

cd /tmp

2. Copy and paste the command to download the people.txt:

#Download people.txt

wget https://raw.githubusercontent.com/hortonworks/data-tutorials/master/tutorials/hdp/dataFrame-and-dataset-examples-in-spark-repl/assets/people.txt

3. Copy and paste the command to download the people.json:

#Download people.json

wget https://raw.githubusercontent.com/hortonworks/data-tutorials/master/tutorials/hdp/dataFrame-and-dataset-examples-in-spark-repl/assets/people.json

**Upload the dataset to HDFS**

1. Before moving the files into HDFS you need to login under **hdfs** user in order to give root user permission to perform file operations:

#Login as hdfs user to give root permissions for file operations

su hdfs

cd

2. Next, upload people.txt and people.json files to HDFS:

#Copy files from local system to HDF

hdfs dfs -put /tmp/people.txt /tmp/people.txt

hdfs dfs -put /tmp/people.json /tmp/people.json

3.Verify that both files were copied into HDFS **/tmp** folder by copying the following commands:

#Verify that files were move to HDF

hdfs dfs -ls /tmp

Now, we are ready to start the examples.

4.Launch the Spark Shell:

spark-shell

**DataFrame API Example**

DataFrame API provides easier access to data since it looks conceptually like a Table and a lot of developers from Python/R/Pandas are familiar with it.

At a scala> REPL prompt, type the following:

val df = spark.read.json("/tmp/people.json")

Using df.show, display the contents of the DataFrame:

df.show

You should see an output similar to:

...

+----+-------+

| age| name|

+----+-------+

|null|Michael|

| 30| Andy|

| 19| Justin|

+----+-------+

scala>

**Additional DataFrame API examples**

Now, lets select "name" and "age" columns and increment the "age" column by 1:

df.select(df("name"), df("age") + 1).show()

This will produce an output similar to the following:

...

+-------+---------+

| name|(age + 1)|

+-------+---------+

|Michael| null|

| Andy| 31|

| Justin| 20|

+-------+---------+

scala>

To return people older than 21, use the filter() function:

df.filter(df("age") > 21).show()

This will produce an output similar to the following:

...

+---+----+

|age|name|

+---+----+

| 30|Andy|

+---+----+

scala>

Next, to count the number of people of specific age, use groupBy() & count() functions:

df.groupBy("age").count().show()

This will produce an output similar to the following:

...

+----+-----+

| age|count|

+----+-----+

| 19| 1|

|null| 1|

| 30| 1|

+----+-----+

scala>

**Programmatically Specifying Schema**

Type the following commands(one line a time) into your Spark-shell:

1. Import the necessary libraries

import org.apache.spark.sql.\_

import org.apache.spark.sql.Row

import org.apache.spark.sql.types.\_

import spark.implicits.\_

2. Create and RDD

val peopleRDD = spark.sparkContext.textFile("/tmp/people.txt")

3. Encode the Schema in a string

val schemaString = "name age"

4. Generate the schema based on the string of schema

val fields = schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, nullable = true))

val schema = StructType(fields)

5. Convert records of the RDD (people) to Rows

val rowRDD = peopleRDD.map(\_.split(",")).map(attributes => Row(attributes(0), attributes(1).trim))

6. Apply the schema to the RDD

val peopleDF = spark.createDataFrame(rowRDD, schema)

6. Creates a temporary view using the DataFrame

peopleDF.createOrReplaceTempView("people")

7. SQL can be run over a temporary view created using DataFrames

val results = spark.sql("SELECT name FROM people")

8.The results of SQL queries are DataFrames and support all the normal RDD operations. The columns of a row in the result can be accessed by field index or by field name

results.map(attributes => "Name: " + attributes(0)).show()

This will produce an output similar to the following:

...

+-------------+

| value|

+-------------+

|Name: Michael|

| Name: Andy|

| Name: Justin|

+-------------+

scala>

**DataSet API Example**

If you haven't done so already in previous sections, make sure to upload people data sets (people.txt and people.json) to HDFS: [Environment Setup](https://hortonworks.com/tutorial/dataframe-and-dataset-examples-in-spark-repl/#environment-setup) and imported the libraries in step 1 of **Programmatically Specifying Schema** above:

Finally, if you haven't already:

Launch Spark Shell

spark-shell

The Spark Dataset API brings the best of RDD and Data Frames together, for type safety and user functions that run directly on existing JVM types.

Let's try the simplest example of creating a dataset by applying a *toDS()* function to a sequence of numbers.

At the scala> prompt, copy & paste the following:

val ds = Seq(1, 2, 3).toDS()

ds.show

You should see the following output:

+-----+

|value|

+-----+

| 1|

| 2|

| 3|

+-----+

Moving on to a slightly more interesting example, let's prepare a *Person* class to hold our person data. We will use it in two ways by applying it directly on a hardcoded data and then on a data read from a json file.

To apply *Person* class to hardcoded data type:

case class Person(name: String, age: Long)

val ds = Seq(Person("Andy", 32)).toDS()

When you type

ds.show

you should see the following output of the *ds* Dataset

+----+---+

|name|age|

+----+---+

|Andy| 32|

+----+---+

Finally, let's map data read from *people.json* to a *Person* class. The mapping will be done by name.

val path = "/tmp/people.json"

val people = spark.read.json(path).as[Person] // Creates a DataSet

To view contents of people DataFrame type:

people.show

You should see an output similar to the following:

...

+----+-------+

| age| name|

+----+-------+

|null|Michael|

| 30| Andy|

| 19| Justin|

+----+-------+

Note that the *age* column contains a *null* value. Before we can convert our people DataFrame to a Dataset, let's filter out the *null* value first:

val pplFiltered = people.filter("age is not null")

Now we can map to the *Person* class and convert our DataFrame to a Dataset.

val pplDS = pplFiltered.as[Person]

View the contents of the Dataset type

pplDS.show

You should see the following:

+------+---+

| name|age|

+------+---+

| Andy| 30|

|Justin| 19|

+------+---+

To exit type:

:quit