Work Integrated Learning Programmes

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Science & Information Systems

Big Data Systems – Spark Lab Sheet: 3

Spark DataFrames

1. Objective

Students should be able to

A. Get familiarity with the DataFrame API of spark
B. Get hands-on experience of using Jupyter notebooks with Spark
C. Get hands-on experience of data analysis exercise using Spark SQL

Spark SQL is a Spark module for structured data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations. There are several ways to interact with Spark SQL including SQL and the Dataset API. When computing a result the same execution engine is used, independent of which API/language you are using to express the computation. This unification means that developers can easily switch back and forth between different APIs based on which provides the most natural way to express a given transformation.

One use of Spark SQL is to execute SQL queries. Spark SQL can also be used to read data from an existing Hive installation. When running SQL from within another programming language the results will be returned as a Dataset/DataFrame. You can also interact with the SQL interface using the command-line or over JDBC/ODBC.

A Dataset is a distributed collection of data. Dataset is a new interface added in Spark 1.6 that provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). The Dataset API is available in Scala and Java. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available (i.e. you can access the field of a row by name naturally row.columnName). The case for R is similar.

A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. The DataFrame API is available in Scala, Java, Python, and R. In Scala and Java, a DataFrame is represented by a Dataset of Rows. In the Scala

API, DataFrame is simply a type alias of Dataset[Row]. While, in Java API, users need to use Dataset<Row> to represent a DataFrame.

2. Steps to be performed

Note - It's assumed that student has made a slot reservation using the slot booking interface where Apache Spark framework was selected. The details of the Apache Spark systems to be used is received through an email. If not, please contact the administrators for the same.
Also it's assumed that students are aware of the process of logging into these virtual machines. If not, then get access to the user manual maintained for the usage of remote lab setup.

Preparations -

Data Preparations -

a) Open the terminal by clicking on icon on desktop

a) Prepare the input json file using any file editor. Copy and paste the content present in the attached students.json file in this file.

```
[centos@master ~]$ gedit students.json
```

Installing pySpark

b) For the execution of python programmes on the Spark, a package named pyspark is required. Using the sudo previleges, install the packages with pip command.
pip install pyspark

Installing jupyterlab and notebook

c) For the execution of python programmes on the Spark with SQL operations, we are going to use the Jupyter notebook setup. For that purpose, the following two packages needs to be installed using the sudo user privileges. Once install is done, exit the root view.

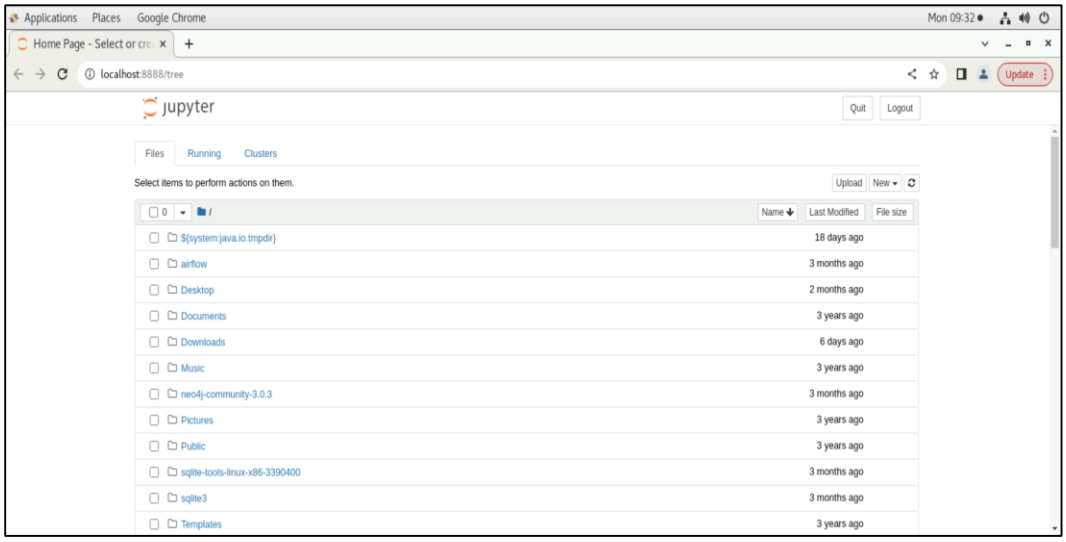
pip install jupyterlab
pip install notebook

More details about it can be found here.

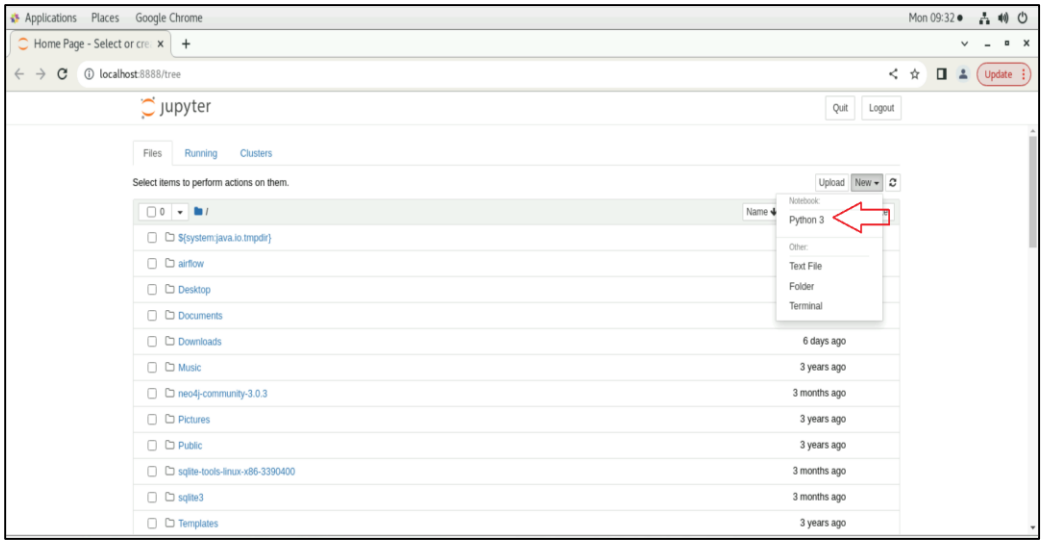
d) Launch the notebook using the following command
Jupyter notebook

```
[centos@master ~]$ jupyter notebook
```

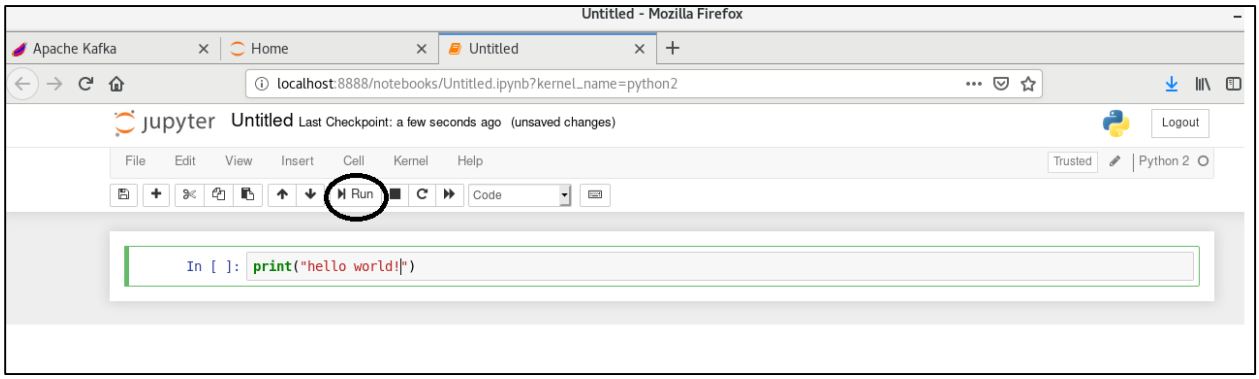
After this a tab will open up in the browser showing the homepage of Jupyter environment which can be used further for the development.



e) Now this page can be used to create a new Python notebook as show in the following screen. Create a new Python 3 notebook.



f) In another tab, a notebook will open up. Check its working by typing following simple Python statement in it and pressing "ctrl + enter" or "shift+enter" keys combination to execute it.



g) Import the required packages and create a spark sql session.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Basics").getOrCreate()
```

```
In [1]: print("hello world!")
        hello world!

In [3]: from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName("Basics").getOrCreate()
        print("session created")
        session created
```

h) Read the student.json data file into a data frame object.

```
df = spark.read.json('students.json')
df
```

```
In [3]: from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName("Basics").getOrCreate()
        print("session created")

        session created

In [4]: df = spark.read.json('students.json')
        df

Out[4]: DataFrame[age: bigint, m1: bigint, m2: bigint, name: string]
```

i) Look at the various aspects of data frame.

```
df.printSchema()
```

```
In [5]: df.printSchema()

        root
         |-- age: long (nullable = true)
         |-- m1: long (nullable = true)
         |-- m2: long (nullable = true)
         |-- name: string (nullable = true)
```

```
df.show()
```

```
In [6]: df.show()

        +---+---+---+------+
        |age| m1| m2|  name|
        +---+---+---+------+
        | 18| 45| 34|Pravin|
        | 30| 15| 23| Pawar|
        | 19| 45| 65|      |
        +---+---+---+------+
```

```
df.columns
```

```
In [7]: df.columns

Out[7]: ['age', 'm1', 'm2', 'name']
```

```
df.describe()
```

```
In [8]: df.describe()
Out[8]: DataFrame[summary: string, age: string, m1: string, m2: string, name: string]
```

j) Try selecting the column of the data frame.

```
df['age']
df.select('age')
df.select('age').show()
```

```
In [10]: df['age']
Out[10]: Column<age>

In [11]: df.select('age')
Out[11]: DataFrame[age: bigint]

In [12]: df.select('age').show()
         +---+
         |age|
         +---+
         | 18|
         | 30|
         | 19|
         +---+
```

k) Try looking at multiple columns with following commands

```
df.head(2)
df.select(['age','name'])
df.select(['age','name']).show()
```

```
In [13]: df.head(2)
Out[13]: [Row(age=18, m1=45, m2=34, name=u'Pravin'),
          Row(age=30, m1=15, m2=23, name=u'Pawar')]

In [14]: df.select(['age','name'])
Out[14]: DataFrame[age: bigint, name: string]

In [15]: df.select(['age','name']).show()
         +---+------+
         |age|  name|
         +---+------+
         | 18|Pravin|
         | 30| Pawar|
         | 19|      |
         +---+------+
```

l) Try creating a new column or renaming the existing column.

```
df.withColumn('newage',df['age']).show()
```

```
In [16]: df.withColumn('newage',df['age']).show()
+---+---+---+------+------+
|age| m1| m2|  name|newage|
+---+---+---+------+------+
| 18| 45| 34|Pravin|    18|
| 30| 15| 23| Pawar|    30|
| 19| 45| 65|      |    19|
+---+---+---+------+------+
```

```
df.withColumnRenamed('age','supernewage').show()
```

```
In [17]: df.withColumnRenamed('age','supernewage').show()
+-----------+---+---+------+
|supernewage| m1| m2|  name|
+-----------+---+---+------+
|         18| 45| 34|Pravin|
|         30| 15| 23| Pawar|
|         19| 45| 65|      |
+-----------+---+---+------+
```

m) Try using the filter selection with all or limited columns.

```
df.filter("age<35").show()
df.filter("age<35").select('name').show()
```

```
In [18]: df.filter("age<35").show()
+---+---+---+------+
|age| m1| m2|  name|
+---+---+---+------+
| 18| 45| 34|Pravin|
| 30| 15| 23| Pawar|
| 19| 45| 65|      |
+---+---+---+------+

In [19]: df.filter("age<35").select('name').show()
+------+
|  name|
+------+
|Pravin|
| Pawar|
|      |
+------+
```

```
df.filter("age<35").select(['age','name']).show()
```

```
In [20]: df.filter("age<35").select(['age','name']).show()
+---+------+
|age|  name|
+---+------+
| 18|Pravin|
| 30| Pawar|
| 19|      |
+---+------+
```

n) Now let's create a SQL representation of this data frame.

```
from pyspark.sql.types import
StructField,StringType,IntegerType,StructType
data_schema = [StructField("age", IntegerType(),
True),StructField("name", StringType(), True)]
student_structure = StructType(fields=data_schema)
```

```
In [21]: from pyspark.sql.types import StructField,StringType,IntegerType,StructType
         data_schema = [StructField("age", IntegerType(), True),StructField("name", StringType(), True)]
         student_structure = StructType(fields=data_schema)
```

o) Read the json file and attach the schema to it. Now we are having access to SQL dataframe which can be further queried like a table in relational database world.

```
df = spark.read.json('students.json',
schema=student_structure)
df.printSchema()
```

```
In [22]: df = spark.read.json('students.json', schema=student_structure)
         df.printSchema()

         root
          |-- age: integer (nullable = true)
          |-- name: string (nullable = true)
```

```
df.createOrReplaceTempView("students")
sql_results = spark.sql("SELECT * FROM students")
sql_results
```

```
In [23]: df.createOrReplaceTempView("students")
         sql_results = spark.sql("SELECT * FROM students")
         sql_results

Out[23]: DataFrame[age: int, name: string]
```

p) Sql_results not showing anything. Try using the show() method on it.

```
sql_results.show()
```

```
In [24]: sql_results.show()

         +---+------+
         |age|  name|
         +---+------+
         | 18|Pravin|
         | 30| Pawar|
         | 19|      |
         +---+------+
```

q) Now let's try writing a new sql query on this temp view.

```
spark.sql("SELECT * FROM students WHERE age=30").show()
```

```
In [25]: spark.sql("SELECT * FROM students WHERE age=30").show()

         +---+-----+
         |age| name|
         +---+-----+
         | 30|Pawar|
         +---+-----+
```

3. Outputs/Results

Students should be able to

- Create a data frame object and further carry out simple data analysis on it
- Create a temporary SQL view and execute SQL queries against it

4. Observations

Students carefully needs to observe

- Various options available with the data frames to access the columns and rows
- Syntax of queries written to interact with the views on Spark SQL

References

A. Spark SQL programming Guide
B. Jupyter notebook installation guide
C. Spark Documentation