Spark DataFrame Basics

Spark DataFrames are the workhouse and main way of working with Spark and Python post Spark 2.0. DataFrames act as powerful versions of tables, with rows and columns, easily handling large datasets. The shift to DataFrames provides many advantages:

- A much simpler syntax
- Ability to use SQL directly in the dataframe
- Operations are automatically distributed across RDDs

If you've used R or even the pandas library with Python you are probably already familiar with the concept of DataFrames. Spark DataFrame expand on a lot of these concepts, allowing you to transfer that knowledge easily by understanding the simple syntax of Spark DataFrames. Remember that the main advantage to using Spark DataFrames vs those other programs is that Spark can handle data across many RDDs, huge data sets that would never fit on a single computer. That comes at a slight cost of some "peculiar" syntax choices, but after this course you will feel very comfortable with all those topics!

Let's get started!

Creating a DataFrame

First we need to start a SparkSession:

In [1]:
```
from pyspark.sql import SparkSession
```

Then start the SparkSession

In [2]:
```
# May take a little while on a local computer
spark = SparkSession.builder.appName("Basics").getOrCreate()
```

You will first need to get the data from a file (or connect to a large distributed file like HDFS, we'll talk about this later once we move to larger datasets on AWS EC2).

In [3]:
```
# We'll discuss how to read other options later.
# This dataset is from Spark's examples

# Might be a little slow locally
df = spark.read.json('people.json')
```

Showing the data

In [4]:
```
# Note how data is missing!
df.show()
```
```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

In [5]:
```
df.printSchema()
```
```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

In [6]:
```
df.columns
```
Out[6]:
```
['age', 'name']
```

In [7]:
```
df.describe()
```
Out[7]:
```
DataFrame[summary: string, age: string]
```

Some data types make it easier to infer schema (like tabular formats such as csv which we will show later).

However you often have to set the schema yourself if you aren't dealing with a .read method that doesn't have inferSchema() built-in.

However you often have to set the schema yourself if you aren't dealing with a .read method that doesn't have inferSchema() built-in.

Spark has all the tools you need for this, it just requires a very specific structure:

In [8]:
```
from pyspark.sql.types import StructField,StringType,IntegerType,StructType
```

Next we need to create the list of Structure fields

```
* :param name: string, name of the field.
* :param dataType: :class:`DataType` of the field.
* :param nullable: boolean, whether the field can be null (None) or not.
```

In [9]:
```
data_schema = [StructField("age", IntegerType(), True),StructField("name", StringType(), True)]
```

In [10]:
```
final_struc = StructType(fields=data_schema)
```

In [11]:
```
df = spark.read.json('people.json', schema=final_struc)
```

In [12]:
```
df.printSchema()
```

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

Grabbing the data

In [13]:
```
df['age']
```

Out[13]:
```
Column<b'age'>
```

In [14]:
```
type(df['age'])
```

Out[14]:
```
pyspark.sql.column.Column
```

In [15]:
```
df.select('age')
```

Out[15]:
```
DataFrame[age: int]
```

In [16]:
```
type(df.select('age'))
```

Out[16]:
```
pyspark.sql.dataframe.DataFrame
```

In [17]:
```
df.select('age').show()
```

```
+----+
| age|
+----+
|null|
|  30|
|  19|
+----+
```

In [18]:
```
# Returns list of Row objects
df.head(2)
```

Out[18]:
```
[Row(age=None, name='Michael'), Row(age=30, name='Andy')]
```

Multiple Columns:

In [19]:
```
df.select(['age','name'])
```

Out[19]:
```
DataFrame[age: int, name: string]
```

In [20]:
```
df.select(['age','name']).show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

Creating new columns

In [21]:
```
# Adding a new column with a simple copy
df.withColumn('newage',df['age']).show()
```

```
+----+-------+------+
| age|   name|newage|
+----+-------+------+
|null|Michael|  null|
|  30|   Andy|    30|
|  19| Justin|    19|
+----+-------+------+
```

In [22]:
```
df.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

In [23]:
```
# Simple Rename
df.withColumnRenamed('age','supernewage').show()
```

```
+-----------+-------+
|supernewage|   name|
+-----------+-------+
|       null|Michael|
|         30|   Andy|
|         19| Justin|
+-----------+-------+
```

More complicated operations to create new columns

In [24]:
```
df.withColumn('doubleage',df['age']*2).show()
```

```
+----+-------+---------+
| age|   name|doubleage|
+----+-------+---------+
|null|Michael|     null|
|  30|   Andy|       60|
|  19| Justin|       38|
+----+-------+---------+
```

In [25]:
```
df.withColumn('add_one_age',df['age']+1).show()
```

```
+----+-------+-----------+
| age|   name|add_one_age|
+----+-------+-----------+
|null|Michael|       null|
|  30|   Andy|         31|
|  19| Justin|         20|
+----+-------+-----------+
```

In [26]:
```
df.withColumn('half_age',df['age']/2).show()
```

```
+----+-------+--------+
| age|   name|half_age|
+----+-------+--------+
|null|Michael|    null|
|  30|   Andy|    15.0|
|  19| Justin|     9.5|
+----+-------+--------+
```

In [27]:
```
df.withColumn('half_age',df['age']/2)
```

Out[27]:
```
DataFrame[age: int, name: string, half_age: double]
```

We'll discuss much more complicated operations later on!

Using SQL

To use SQL queries directly with the dataframe, you will need to register it to a temporary view:

In [28]:
```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")
```

In [29]:
```
sql_results = spark.sql("SELECT * FROM people")
```

In [30]:
```
sql_results
```

Out[30]:
```
DataFrame[age: int, name: string]
```

In [31]:
```
sql_results.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
```