Data Sources: CSV and XML

Lesson Objectives

- After completing this lesson, you should be able to:
  - Read and write CSV files to and from DataFrames
  - Read and write XML files to and from DataFrames
  - Manually specify schemas for reading both CSV and XML files

Spark CSV

- Databricks package to allow reading CSV files in local or distributed filesystems as DataFrames
- `https://github.com/databricks/spark-csv`
- Sample file: `https://github.com/databricks/spark-csv/raw/master/src/test/resources/cars.csv`

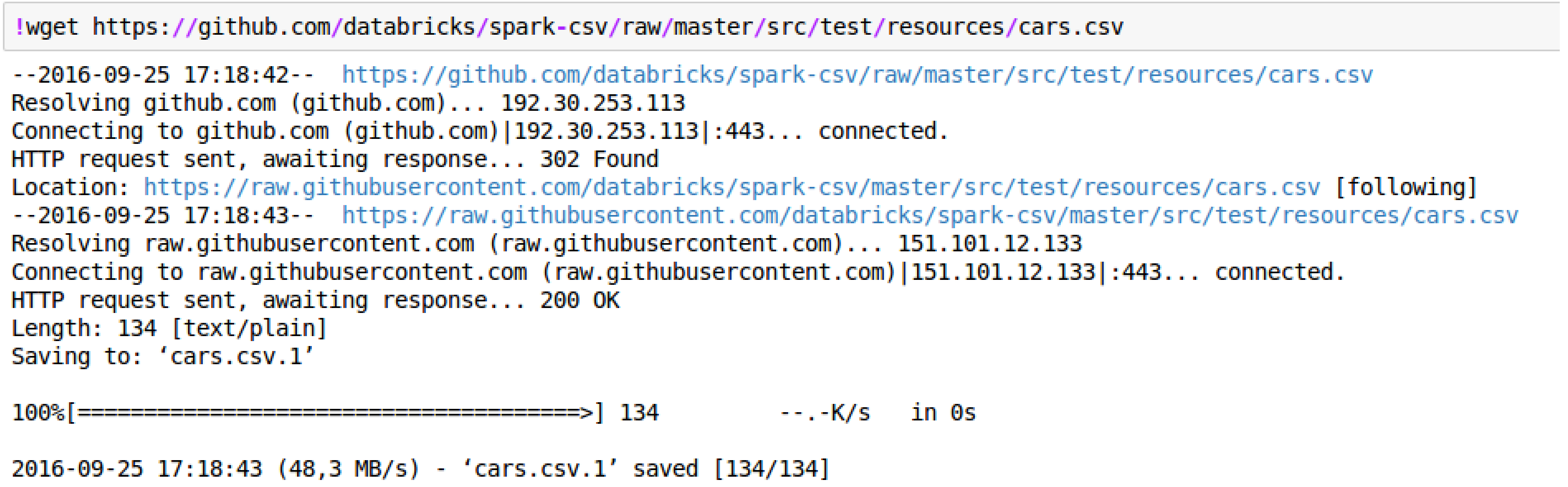
Read Options (1)

Option	Description
path	location of files
header	default = false, if true first line will be used to name columns
delimiter	default = ,
quote	default = "
escape	default = \
charset	default = 'UTF-8', can be set to other valid charset names
parserLib	default = "commons", can be set to "univocity“

Read Options (2)

Option	Description
mode	PERMISSIVE (default): tries to parse all lines: nulls are inserted for missing tokens and extra tokens are ignored DROPMALFORMED: drops lines which have fewer or more tokens than expected or tokens which do not match the schema FAILFAST: aborts with a RuntimeException if encounters any malformed line
inferSchema	default = false, if true, automatically infers column types (requires one extra pass over the data)
Comment	default = #, skip lines beginning with this character
nullValue	string to indicate a null value
dateFormat	default = null (using java.sql.Timestamp.valueOf() and java.sql.Date.valueOf() to parse times and dates), custom date formats follows the format at `java.text.SimpleDateFormat`

Read Example



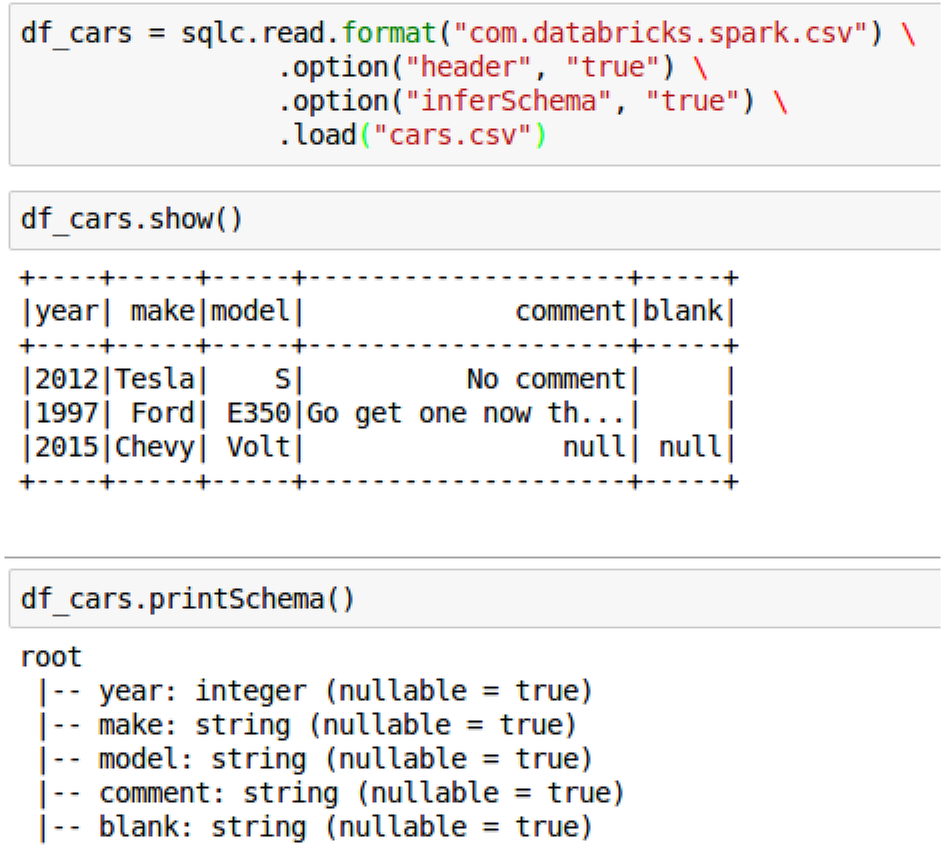
```
!wget https://github.com/databricks/spark-csv/raw/master/src/test/resources/cars.csv
```

```
--2016-09-25 17:18:42--  https://github.com/databricks/spark-csv/raw/master/src/test/resources/cars.csv
Resolving github.com (github.com)... 192.30.253.113
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/databricks/spark-csv/master/src/test/resources/cars.csv [following]
--2016-09-25 17:18:43--  https://raw.githubusercontent.com/databricks/spark-csv/master/src/test/resources/cars.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.12.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 134 [text/plain]
Saving to: ‘cars.csv.1’

100%[======================================>] 134         --.-K/s   in 0s

2016-09-25 17:18:43 (48,3 MB/s) - ‘cars.csv.1’ saved [134/134]
```

Read Example



```
df_cars = sqlc.read.format("com.databricks.spark.csv") \
                .option("header", "true") \
                .option("inferSchema", "true") \
                .load("cars.csv")
```

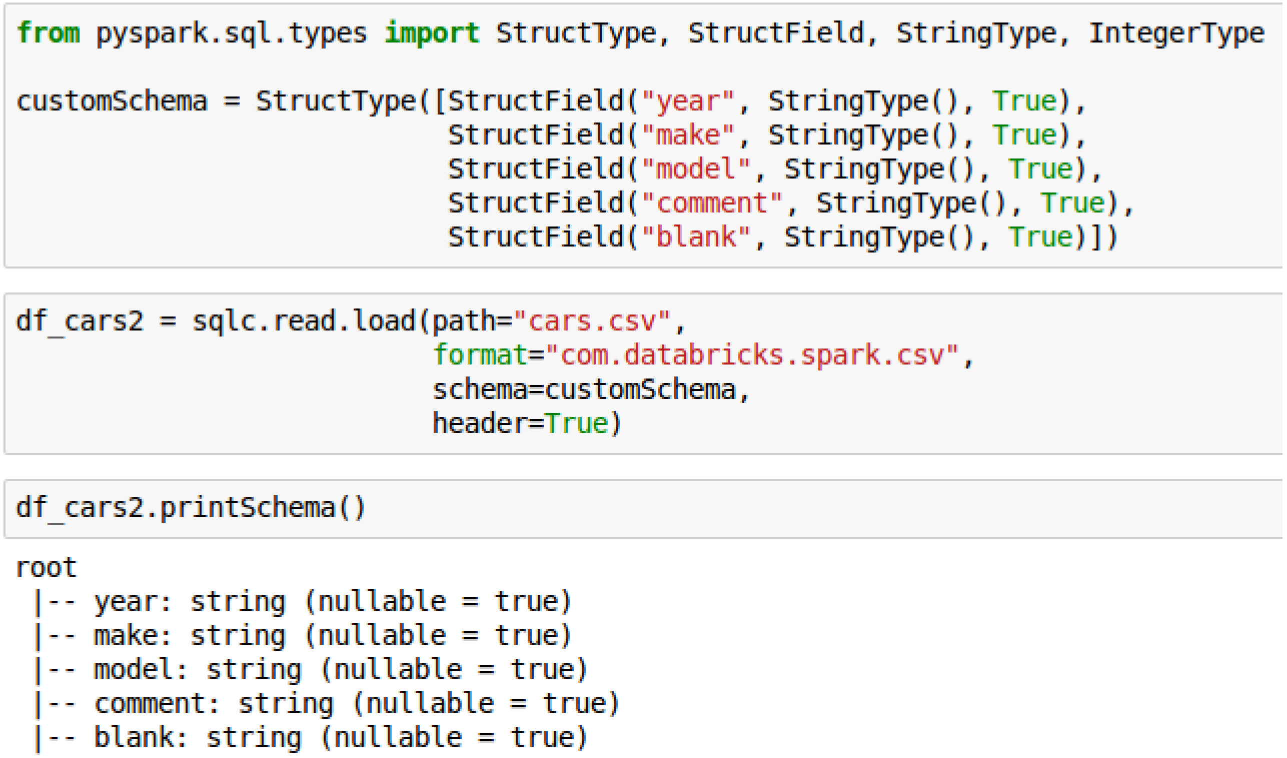
```
df_cars.show()
```

```
+----+-----+-----+--------------------+-----+
|year| make|model|             comment|blank|
+----+-----+-----+--------------------+-----+
|2012|Tesla|    S|          No comment|     |
|1997| Ford| E350|Go get one now th...|     |
|2015|Chevy| Volt|                null| null|
+----+-----+-----+--------------------+-----+
```

```
df_cars.printSchema()
```

```
root
 |-- year: integer (nullable = true)
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)
 |-- comment: string (nullable = true)
 |-- blank: string (nullable = true)
```

Manually Specifying a Schema



```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

customSchema = StructType([StructField("year", StringType(), True),
                           StructField("make", StringType(), True),
                           StructField("model", StringType(), True),
                           StructField("comment", StringType(), True),
                           StructField("blank", StringType(), True)])
```

```
df_cars2 = sqlc.read.load(path="cars.csv",
                          format="com.databricks.spark.csv",
                          schema=customSchema,
                          header=True)
```

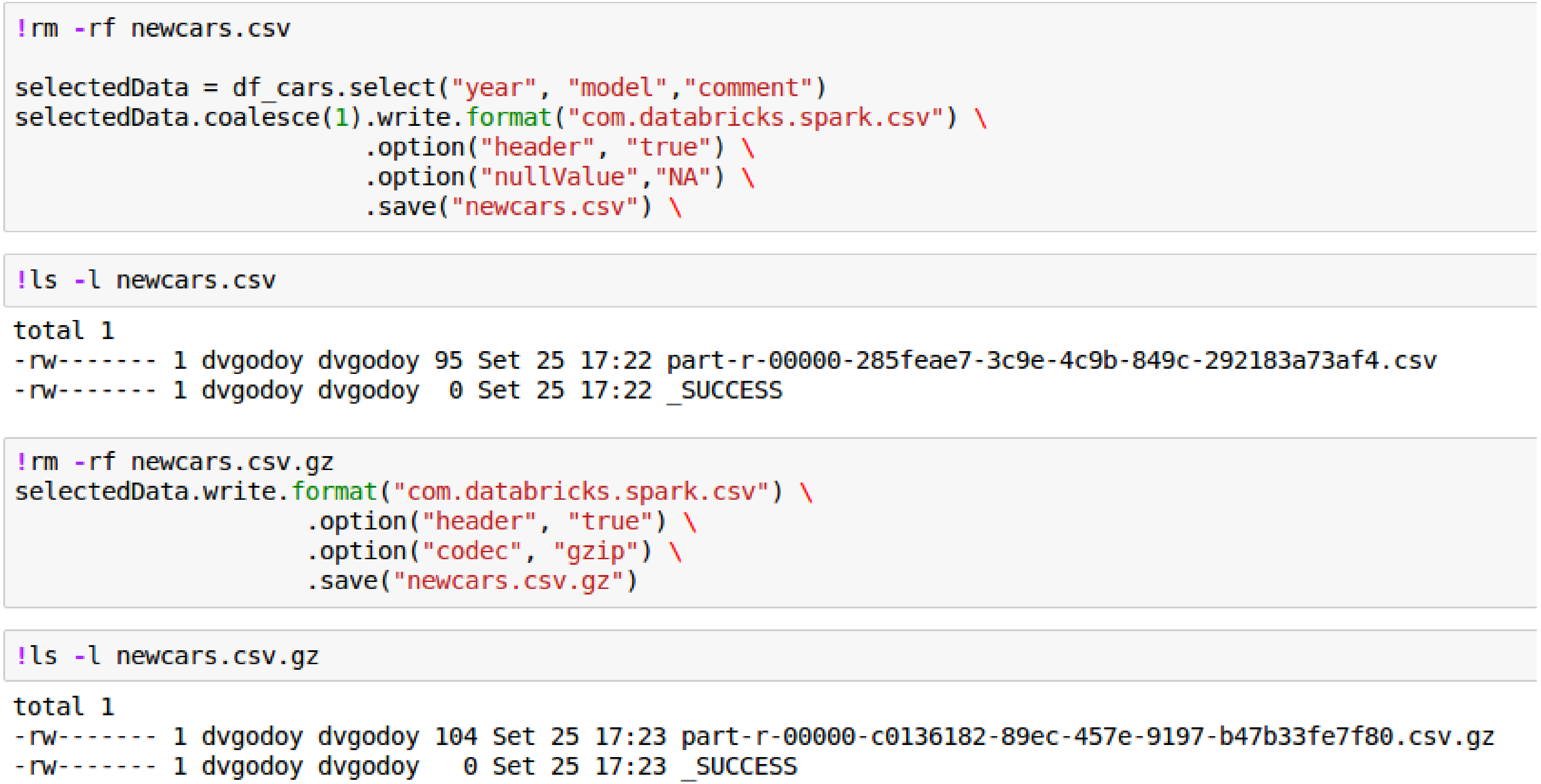
```
df_cars2.printSchema()
```

```
root
 |-- year: string (nullable = true)
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)
 |-- comment: string (nullable = true)
 |-- blank: string (nullable = true)
```

Write Options

Option	Description
path	location of files
header	writes header from the schema in the DataFrame at the first line
delimiter	default = ,
quote	default = " (according to quoteMode)
quoteMode	when to quote fields (ALL, MINIMAL (default), NON_NUMERIC, NONE)
escape	default = \
nullValue	string to indicate a null value
codec	compression codec to use when saving to file (bzip2, gzip, lz4, snappy) or class implementing `org.apache.hadoop.io.compress.CompressionCodec`

Write Example



```
!rm -rf newcars.csv

selectedData = df_cars.select("year", "model","comment")
selectedData.coalesce(1).write.format("com.databricks.spark.csv") \
                        .option("header", "true") \
                        .option("nullValue","NA") \
                        .save("newcars.csv") \
```

```
!ls -l newcars.csv
```

```
total 1
-rw------- 1 dvgodoy dvgodoy 95 Set 25 17:22 part-r-00000-285feae7-3c9e-4c9b-849c-292183a73af4.csv
-rw------- 1 dvgodoy dvgodoy  0 Set 25 17:22 _SUCCESS
```

```
!rm -rf newcars.csv.gz
selectedData.write.format("com.databricks.spark.csv") \
                    .option("header", "true") \
                    .option("codec", "gzip") \
                    .save("newcars.csv.gz")
```

```
!ls -l newcars.csv.gz
```

```
total 1
-rw------- 1 dvgodoy dvgodoy 104 Set 25 17:23 part-r-00000-c0136182-89ec-457e-9197-b47b33fe7f80.csv.gz
-rw------- 1 dvgodoy dvgodoy   0 Set 25 17:23 _SUCCESS
```

Spark XML

- Databricks package to allow reading XML files in local or distributed filesystems as DataFrames
  - `https://github.com/databricks/spark-xml`
- Sample file: `https://github.com/databricks/spark-xml/raw/master/src/test/resources/books.xml`
- In spark-default.conf, add:
  - spark.jars.packages   com.databricks:spark-xml_2.11:0.4.0

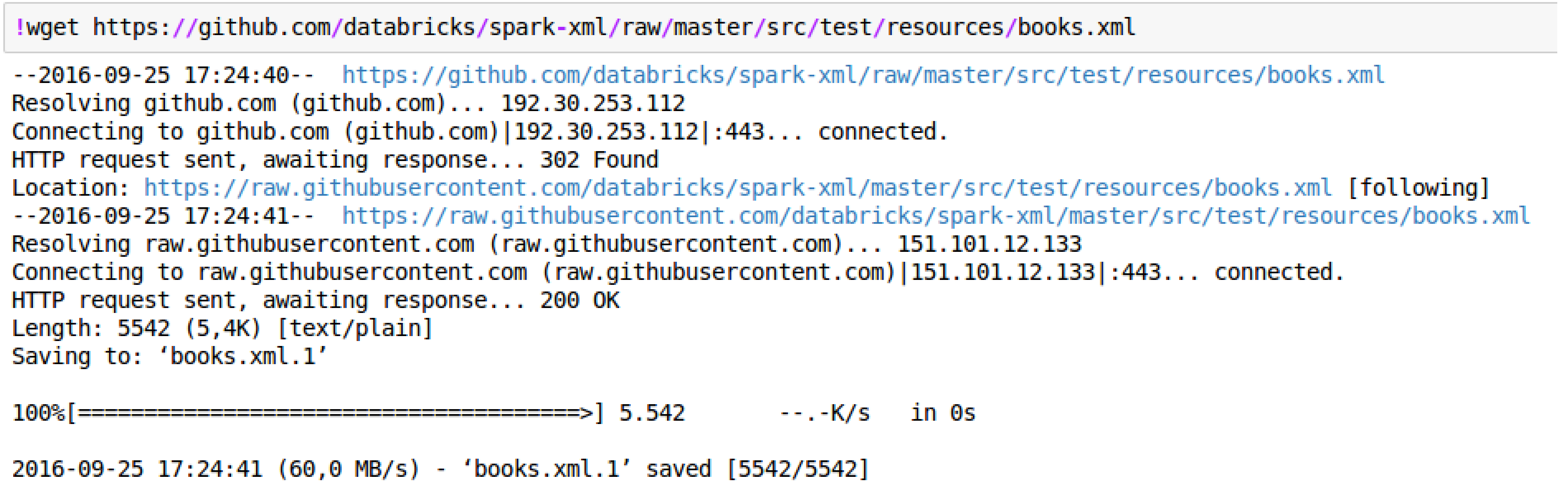
Read Options

Option	Description
path	location of files
rowTag	default = ROW, row tag of the XML file (book, in the sample file)
samplingRatio	default = 1, sampling ratio for inferring schema (0.0 ~ 1)
excludeAttribute	default = false, whether to exclude attributes in elements or not
treatEmpyValuesAsNull	default = false, whether to treat whitespaces as a null value
failFast	default = false, if true, it fails to parse malformed rows in XML files
attributePrefix	default = @, the prefix for field names
valueTag	default = #VALUE, when there are attributes and no child
charset	default = 'UTF-8', can be set to other valid charset names

Write Options

Option	Description
path	location of files
rowTag	default = ROW, row tag of the XML file (book, in the sample file)
rootTag	default = ROWS, root tag of the XML file (books, in the sample file)
nullValue	default = string null, the value to write null value
attributePrefix	default = @, the prefix for field names
valueTag	default = #VALUE, when there are attributes and no child
codec	compression codec to use when saving to file (bzip2, gzip, lz4, snappy) or class implementing `org.apache.hadoop.io.compress.CompressionCodec`

XML Example – Inferred Schema



```
!wget https://github.com/databricks/spark-xml/raw/master/src/test/resources/books.xml
```

```
--2016-09-25 17:24:40--  https://github.com/databricks/spark-xml/raw/master/src/test/resources/books.xml
Resolving github.com (github.com)... 192.30.253.112
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/databricks/spark-xml/master/src/test/resources/books.xml [following]
--2016-09-25 17:24:41--  https://raw.githubusercontent.com/databricks/spark-xml/master/src/test/resources/books.xml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.12.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5542 (5,4K) [text/plain]
Saving to: ‘books.xml.1’

100%[======================================>] 5.542       --.-K/s   in 0s

2016-09-25 17:24:41 (60,0 MB/s) - ‘books.xml.1’ saved [5542/5542]
```

```
!cat books.xml
```

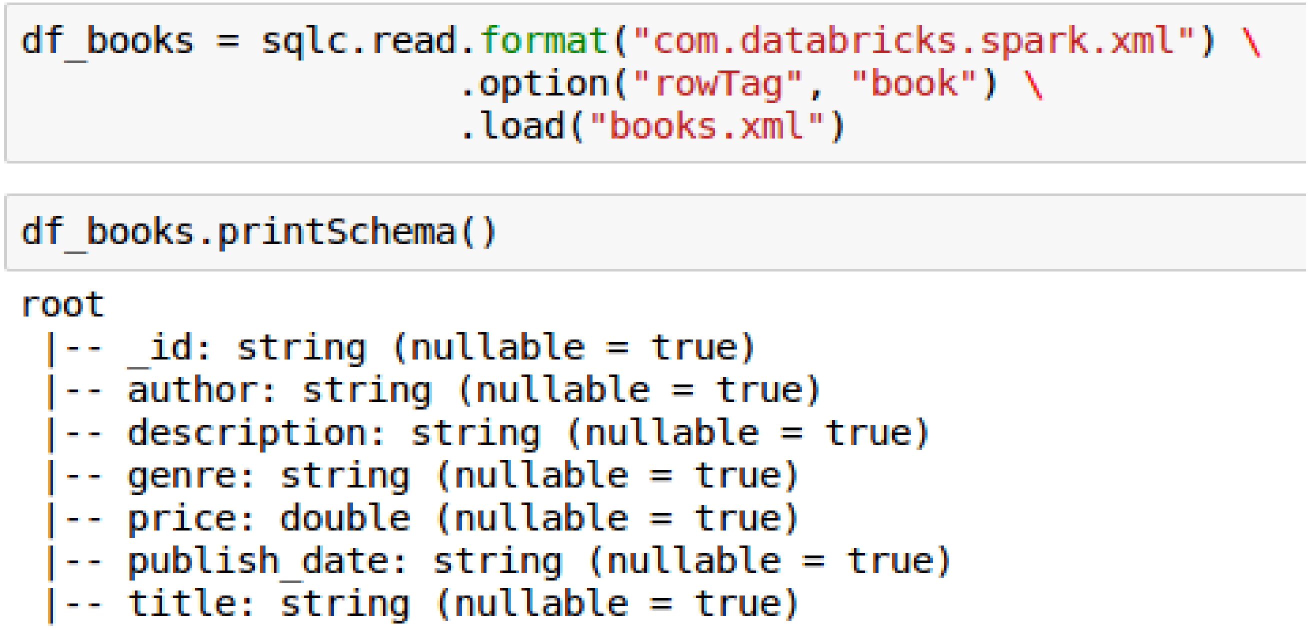
```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>


         An in-depth look at creating applications
         with XML.This manual describes Oracle XML DB, and how you can use it to store, generate, manipulate, manage,
         and query XML data in the database.


         After introducing you to the heart of Oracle XML DB, namely the XMLType framework and Oracle XML DB repository,
         the manual provides a brief introduction to design criteria to consider when planning your Oracle XML DB
         application. It provides examples of how and where you can use Oracle XML DB.


         The manual then describes ways you can store and retrieve XML data using Oracle XML DB, APIs for manipulating
         XMLType data, and ways you can view, generate, transform, and search on existing XML data. The remainder of
         the manual discusses how to use Oracle XML DB repository, including versioning and security,
         how to access and manipulate repository resources using protocols, SQL, PL/SQL, or Java, and how to manage
         your Oracle XML DB application using Oracle Enterprise Manager. It also introduces you to XML messaging and
         Oracle Streams Advanced Queuing XMLType support.
      </description></book><book id="bk102">
```

XML Example – Inferred Schema

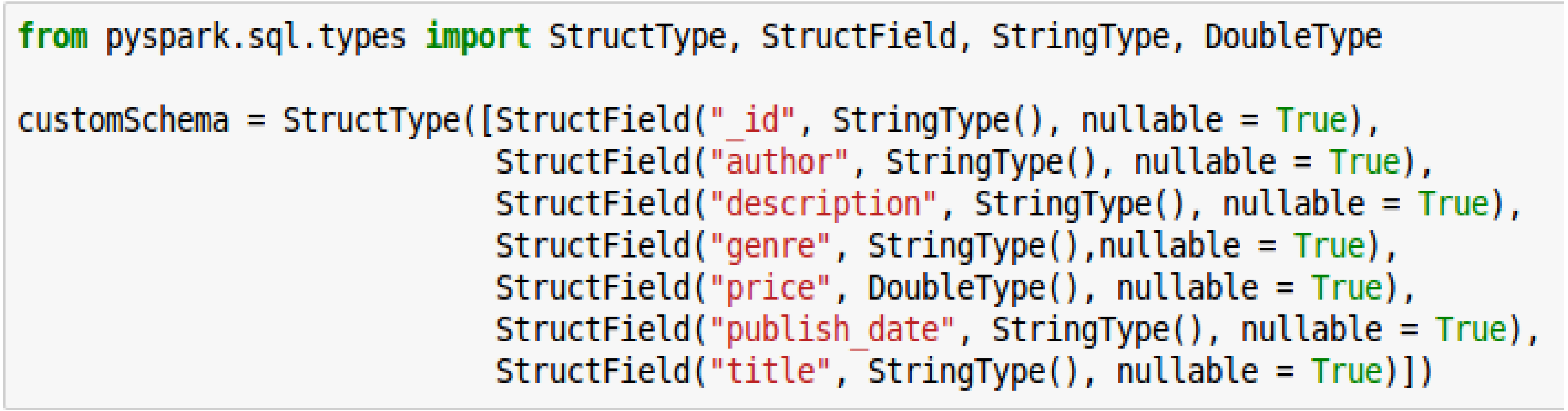


```
df_books = sqlc.read.format("com.databricks.spark.xml") \
                    .option("rowTag", "book") \
                    .load("books.xml")
```

```
df_books.printSchema()
```

```
root
 |-- _id: string (nullable = true)
 |-- author: string (nullable = true)
 |-- description: string (nullable = true)
 |-- genre: string (nullable = true)
 |-- price: double (nullable = true)
 |-- publish_date: string (nullable = true)
 |-- title: string (nullable = true)
```

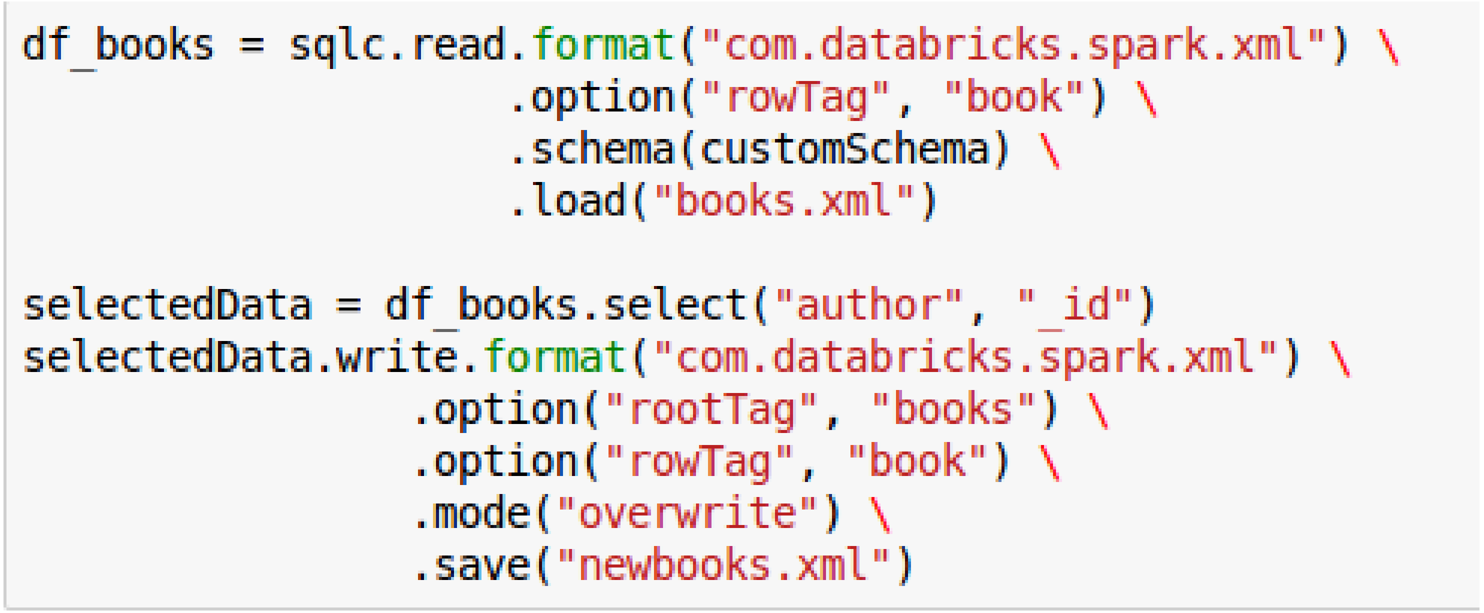
XML Example – Specified Schema



```python
from pyspark.sql.types import StructType, StructField, StringType, DoubleType

customSchema = StructType([StructField("_id", StringType(), nullable = True),
                           StructField("author", StringType(), nullable = True),
                           StructField("description", StringType(), nullable = True),
                           StructField("genre", StringType(),nullable = True),
                           StructField("price", DoubleType(), nullable = True),
                           StructField("publish_date", StringType(), nullable = True),
                           StructField("title", StringType(), nullable = True)])
```

XML Example – Specified Schema



```
df_books = sqlc.read.format("com.databricks.spark.xml") \
                    .option("rowTag", "book") \
                    .schema(customSchema) \
                    .load("books.xml")

selectedData = df_books.select("author", "_id")
selectedData.write.format("com.databricks.spark.xml") \
                .option("rootTag", "books") \
                .option("rowTag", "book") \
                .mode("overwrite") \
                .save("newbooks.xml")
```

Lesson Summary

- Having completed this lesson, you should be able to:
  - Read and write CSV files to and from DataFrames
  - Read and write XML files to and from DataFrames
  - Manually specify schemas for reading both CSV and XML files