Working in Spark requires us to put data into DataFrames. If you're wondering if these DataFrames are comparable to those in Pandas, you're correct—Spark DataFrames are very similar. Just as in Pandas, the first step is to load your data into a DataFrame.

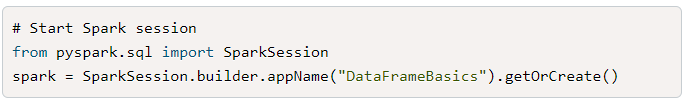
The schema, or structure, for DataFrames and datasets contains the column names and the data types contained within.

The schema can be inferred automatically by letting Spark determine the schema on its own when data is read in or you can define the schema manually and tell spark to use that.

In this module, we'll use the Python version of Spark, PySpark.

Using your Google Colab Notebook, with PySpark installed, follow along with the code.

Create a Spark session by importing the library and setting the spark variable to the code below

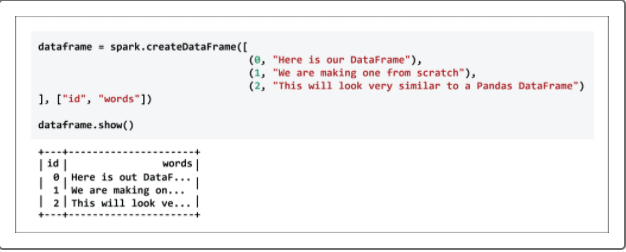


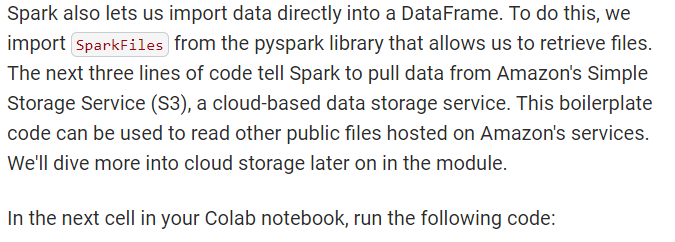
**NOTE**

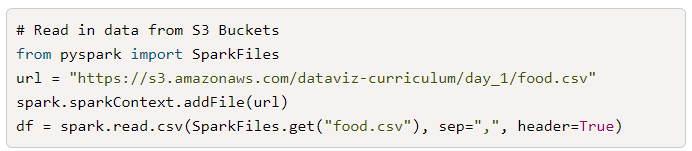
Every time you open a new Google Colab Notebook, run the Spark installation and then start a Spark session. The app name can be different for each notebook.

This creates a Spark application called "DataFrameBasics."

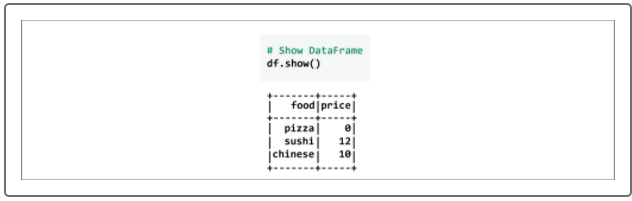
Spark enables us to create a DataFrame from scratch by passing in a list of tuples to the createDataFrame method followed by a list of the column names. The show method will display the DataFrame, which is similar to using the head() function in Pandas. Enter and run the following code:







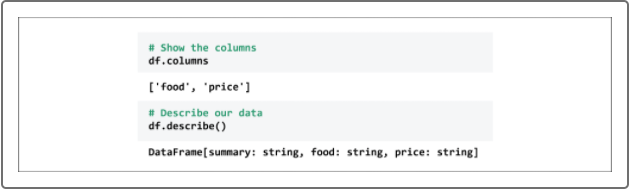
Type the following code to use show() again to display the results, as follows:



Spark will infer the schema from the data, unless otherwise specified. We can check the schema by running the following code:

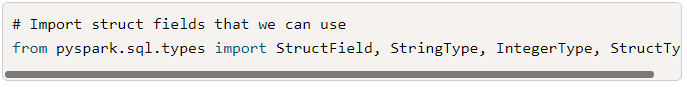


Spark also allows users to view columns and a dataset description by running each of the code blocks:

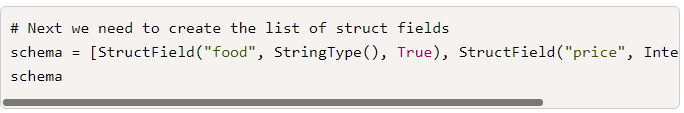


Notice that the DataFrame is claiming that price is a string. Generally, price is either stored as an integer or floating-point number, so you'll need to change this column.

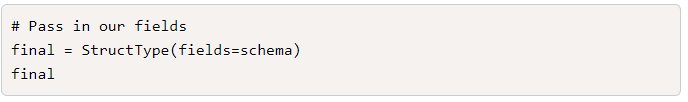
In this case, we can set our schema and then apply it to the data. We'll start by importing the different types of data with the following code:



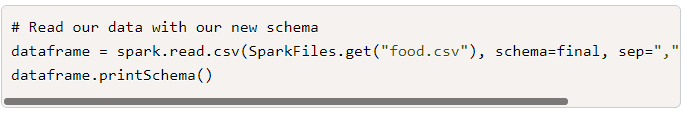
Next, create the schema by creating a StructType, which is one of Spark's complex types, like an array or map. The StructField will define the column name, the data type held, and a Boolean to define whether null values will be included or not:



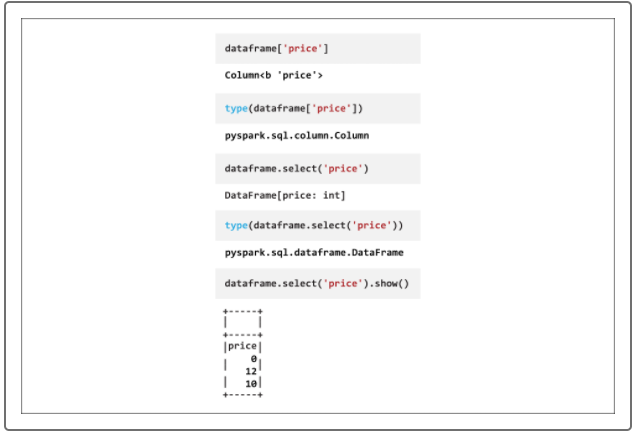
Next, enter the code that will pass the schema just created as fields in a StructType. All this will be stored in a variable called final:



Now that we have a predefined schema, we can read in the data again, only this time passing in our own schema. Type and run the following code in a new notebook cell:



There are a few different ways to access our data with Spark. Run the following commands and look at the results:

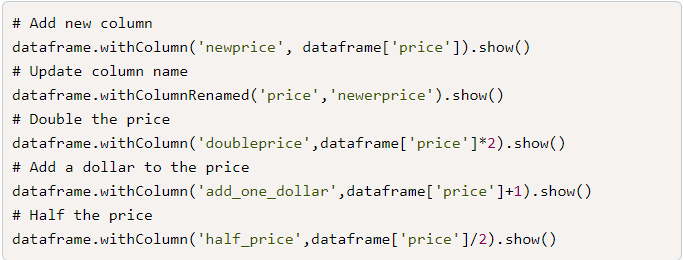


Again, you may notice some similarities to Pandas. For example, in both Pandas and Spark, you can select a column using the DataFrame name, followed by the column's name in square brackets. In Pandas, you can quickly take a look at a DataFrame using head(); in Spark, you can do something similar using show().

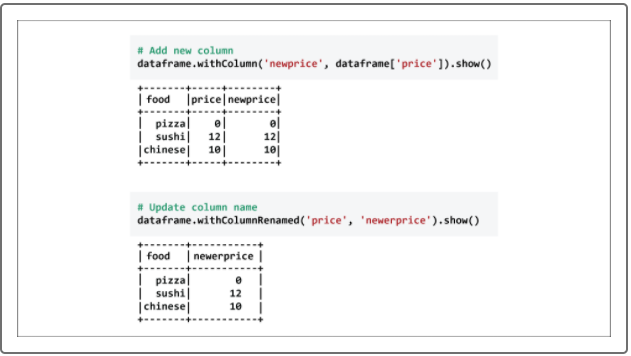
**NOTE**

You might notice that code like dataframe['price'] isn't performing as expected. After running this code, we get the column name, but no results until the show() function runs. show is an action, whereas select is a transformation. We'll cover what this means in the next section.

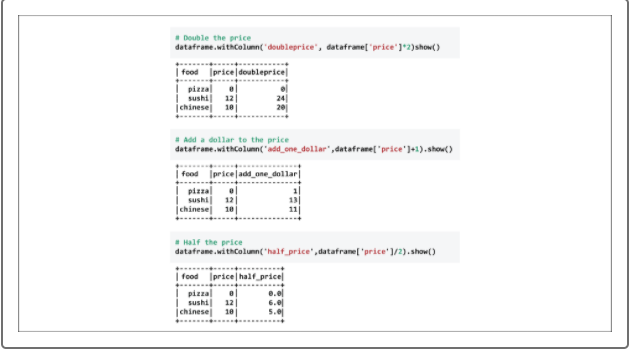
We can manipulate columns in Spark as well. Run the code below and guess what will be displayed:



The first cell duplicates the price column into a new column, preserving all its rows, and naming that column newprice. The second cell simply renames the price column as newprice.



The next three cells created a new column, but they performed some operation on the original:



Next, we'll dive into Spark functions and why some of these commands produced actual results while others just relayed information about our data.