IBM Developer
SKILLS NETWORK

Introduction to SparkSQL

Estimated time needed: **15** minutes

This lab goes over the basic operations of Apache SparkSQL.

APACHE Spark™

Objectives

Spark SQL is a Spark module for structured data processing. It is sed to query structured data inside Spark programs, using either SQL or a familiar DataFrame API.

After completing this lab you will be able to:

- Load a data file into a dataframe
- Create a Table View for the dataframe
- Run basic SQL queries and aggregate data on the table view
- Create a Pandas UDF to perform columnar operations

Setup

For this lab, we are going to be using Python and Spark (PySpark). These libraries should be installed in your lab environment or in SN Labs. Pandas is a popular data science package for Python. In this lab, we use Pandas to load a CSV file from disc to a pandas dataframe in memory. PySpark is the Spark API for Python. In this lab, we use PySpark to initialize the spark context.

In [1]:

```
# Installing required packages
!pip install pyspark
!pip install findspark
!pip install pyarrow==1.0.0
!pip install pandas
!pip install numpy==1.19.5
```

```
Requirement already satisfied: pyspark in /home/jupyterlab/conda/envs/pyth
on/lib/python3.7/site-packages (3.2.1)
Requirement already satisfied: py4j==0.10.9.3 in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (from pyspark) (0.10.9.3)
Requirement already satisfied: findspark in /home/jupyterlab/conda/envs/py
thon/lib/python3.7/site-packages (2.0.0)
Requirement already satisfied: pyarrow==1.0.0 in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (1.0.0)
Requirement already satisfied: numpy>=1.14 in /home/jupyterlab/conda/envs/
python/lib/python3.7/site-packages (from pyarrow==1.0.0) (1.19.5)
Requirement already satisfied: pandas in /home/jupyterlab/conda/envs/pytho
n/lib/python3.7/site-packages (1.3.4)
Requirement already satisfied: python-dateutil>=2.7.3 in /home/jupyterlab/
conda/envs/python/lib/python3.7/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /home/jupyterlab/conda/env
s/python/lib/python3.7/site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.17.3 in /home/jupyterlab/conda/env
s/python/lib/python3.7/site-packages (from pandas) (1.19.5)
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/pyt
hon/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas) (1.1
6.0)
Requirement already satisfied: numpy==1.19.5 in /home/jupyterlab/conda/env
s/python/lib/python3.7/site-packages (1.19.5)
```

In [2]:

```
import findspark
findspark.init()
```

In [3]:

```
import pandas as pd
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
```

Exercise 1 - Spark session

Create and initialize the Spark session needed to load the data frames and operate on it

Task 1: Creating the spark session and context

In [4]:

```
# Creating a spark context class
sc = SparkContext()

# Creating a spark session
spark = SparkSession \
    .builder \
    .appName("Python Spark DataFrames basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/jupyterlab/conda/envs/python/lib/p
ython3.7/site-packages/pyspark/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/jupyterlab/hadoop-2.9.2/share/hado
op/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.
class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explan
ation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setL
ogLevel(newLevel).
22/02/02 01:00:52 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
22/02/02 01:00:54 WARN util.Utils: Service 'SparkUI' could not bind on por
t 4040. Attempting port 4041.
```

Task 2: Initialize Spark session

To work with dataframes we just need to verify that the spark session instance has been created.

In [5]:

```
spark
```

Out[5]:

SparkSession - in-memory
SparkContext

[Spark UI (http://jupyterlab-joaocosta1:4041)](http://jupyterlab-joaocosta1:4041)

Version
`v3.2.1`
Master
`local[*]`
AppName
`pyspark-shell`

Exercise 2 - Loading the Data and creating a table view

In this section, you will first read the CSV file into a Pandas Dataframe and then read it into a Spark Dataframe Pandas is a library used for data manipulation and analysis. The Pandas library offers data structures and operations for creating and manipulating Data Series and DataFrame objects. Data can be imported from various data sources, e.g., Numpy arrays, Python dictionaries, and CSV files. Pandas allows you to manipulate, organize and display the data.

To create a Spark DataFrame we load an external DataFrame, called `mtcars` . This DataFrame includes 32 observations on 11 variables:

colIndex	colName	units/description
[, 1]	mpg	Miles per gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (lb/1000)
[, 7]	qsec	1/4 mile time
[, 8]	vs	V/S
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

Task 1: Load data into a Pandas DataFrame.

Pandas has a convenient function to load CSV data from a URL directly into a pandas dataframe.

In [6]:

```
# Read the file using `read_csv` function in pandas
mtcars = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/data/mtcars.csv')
```

In [7]:

```
# Preview a few records
mtcars.head()
```

Out[7]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [8]:

```
mtcars.rename( columns={'Unnamed: 0':'name'}, inplace=True )
```

Task 2: Loading data into a Spark DataFrame

We use the `createDataFrame` function to load the data into a spark dataframe

In [9]:

```
sdf = spark.createDataFrame(mtcars)
```

Let us look at the schema of the loaded spark dataframe

In [10]:

```
sdf.printSchema()
```

```
root
 |-- name: string (nullable = true)
 |-- mpg: double (nullable = true)
 |-- cyl: long (nullable = true)
 |-- disp: double (nullable = true)
 |-- hp: long (nullable = true)
 |-- drat: double (nullable = true)
 |-- wt: double (nullable = true)
 |-- qsec: double (nullable = true)
 |-- vs: long (nullable = true)
 |-- am: long (nullable = true)
 |-- gear: long (nullable = true)
 |-- carb: long (nullable = true)
```

Task 3: Create a Table View

Creating a table view in Spark SQL is required to run SQL queries programmatically on a DataFrame. A view is a temporary table to run SQL queries. A Temporary view provides local scope within the current Spark session. In this example we create a temporary view using the `createTempView()` function

In [11]:

```
sdf.createTempView("cars")
```

Exercise 3 - Running SQL queries and aggregating data

Once we have a table view, we can run queries similar to querying a SQL table. We perform similar operations to the ones in the DataFrames notebook. Note the difference here however is that we use the SQL queries directly.

In [12]:

```
# Showing the whole table
spark.sql("SELECT * FROM cars").show()
```

```
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+
|               name| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb
|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+
|          Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|   4
|
|      Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|   4
|
|         Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1
|
|     Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3|   1
|
|  Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3|   2
|
|            Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|  0|   3|   1
|
|         Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|  0|   3|   4
|
|          Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2
|
|           Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2
|
|           Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4
|
|          Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4
|
|         Merc 450SE|16.4|  8|275.8|180|3.07| 4.07| 17.4|  0|  0|   3|   3
|
|         Merc 450SL|17.3|  8|275.8|180|3.07| 3.73| 17.6|  0|  0|   3|   3
|
|        Merc 450SLC|15.2|  8|275.8|180|3.07| 3.78| 18.0|  0|  0|   3|   3
|
| Cadillac Fleetwood|10.4|  8|472.0|205|2.93| 5.25|17.98|  0|  0|   3|   4
|
|Lincoln Continental|10.4|  8|460.0|215| 3.0|5.424|17.82|  0|  0|   3|   4
|
|  Chrysler Imperial|14.7|  8|440.0|230|3.23|5.345|17.42|  0|  0|   3|   4
|
|           Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1
|
|        Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2
|
|     Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|   4|   1
|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+
only showing top 20 rows
```

In [13]:

```
# Showing a specific column
spark.sql("SELECT mpg FROM cars").show(5)
```

```
+----+
| mpg|
+----+
|21.0|
|21.0|
|22.8|
|21.4|
|18.7|
+----+
only showing top 5 rows
```

In [14]:

```
# Basic filtering query to determine cars that have a high mileage and low cylinder cou
nt
spark.sql("SELECT * FROM cars where mpg>20 AND cyl < 6").show(5)
```

```
+-----------+----+---+-----+---+----+-----+-----+---+---+----+----+
|       name| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+-----------+----+---+-----+---+----+-----+-----+---+---+----+----+
| Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1|
|  Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2|
|   Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2|
|   Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1|
|Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2|
+-----------+----+---+-----+---+----+-----+-----+---+---+----+----+
only showing top 5 rows
```

In [15]:

```
# Aggregating data and grouping by cylinders
spark.sql("SELECT count(*), cyl from cars GROUP BY cyl").show()
```

```
[Stage 8:=================>                                  (5 + 11)
/ 16]

+--------+---+
|count(1)|cyl|
+--------+---+
|       7|  6|
|      11|  4|
|      14|  8|
+--------+---+
```

Exercise 4 - Create a Pandas UDF to apply a columnar operation

Apache Spark has become the de-facto standard in processing big data. To enable data scientists to leverage the value of big data, Spark added a Python API in version 0.7, with support for user-defined functions (UDF). These user-defined functions operate one-row-at-a-time, and thus suffer from high serialization and invocation overhead. As a result, many data pipelines define UDFs in Java and Scala and then invoke them from Python.

Pandas UDFs built on top of Apache Arrow bring you the *best of both worlds*—the ability to define low-overhead, high-performance UDFs entirely in Python. In this simple example, we will build a Scalar Pandas UDF to convert the wT column from imperial units (1000-lbs) to metric units (metric tons).

In addition, UDFs can be registered and invoked in SQL out of the box by registering a regular python function using the `@pandas_udf()` decorator. We can then apply this UDF to our `wt` column.

Task 1: Importing libraries and registering a UDF

In [16]:

```
# import the Pandas UDF function
from pyspark.sql.functions import pandas_udf, PandasUDFType
```

In [17]:

```
@pandas_udf("float")
def convert_wt(s: pd.Series) -> pd.Series:
    # The formula for converting from imperial to metric tons
    return s * 0.45

spark.udf.register("convert_weight", convert_wt)
```

Out[17]:

```
<function __main__.convert_wt(s: pandas.core.series.Series) -> pandas.cor
e.series.Series>
```

Task 2: Applying the UDF to the tableview

We can now apply the `convert_weight` user-defined-function to our `wt` column from the `cars` table view. This is done very simply using the SQL query shown below. In this example below we show both the original weight (in ton-lbs) and converted weight (in metric tons).

In [18]:

```
spark.sql("SELECT *, wt AS weight_imperial, convert_weight(wt) as weight_metric FROM ca
rs").show()
```

```
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+---------------+-------------+
|               name| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb
|weight_imperial|weight_metric|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+---------------+-------------+
|          Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|   4
|           2.62|        1.179|
|      Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|   4
|          2.875|      1.29375|
|         Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1
|           2.32|        1.044|
|     Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3|   1
|          3.215|      1.44675|
|  Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3|   2
|           3.44|        1.548|
|            Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|  0|   3|   1
|           3.46|        1.557|
|         Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|  0|   3|   4
|           3.57|       1.6065|
|          Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2
|           3.19|       1.4355|
|           Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2
|           3.15|       1.4175|
|           Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4
|           3.44|        1.548|
|          Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4
|           3.44|        1.548|
|         Merc 450SE|16.4|  8|275.8|180|3.07| 4.07| 17.4|  0|  0|   3|   3
|           4.07|       1.8315|
|         Merc 450SL|17.3|  8|275.8|180|3.07| 3.73| 17.6|  0|  0|   3|   3
|           3.73|       1.6785|
|        Merc 450SLC|15.2|  8|275.8|180|3.07| 3.78| 18.0|  0|  0|   3|   3
|           3.78|        1.701|
| Cadillac Fleetwood|10.4|  8|472.0|205|2.93| 5.25|17.98|  0|  0|   3|   4
|           5.25|       2.3625|
|Lincoln Continental|10.4|  8|460.0|215| 3.0|5.424|17.82|  0|  0|   3|   4
|          5.424|       2.4408|
|  Chrysler Imperial|14.7|  8|440.0|230|3.23|5.345|17.42|  0|  0|   3|   4
|          5.345|      2.40525|
|           Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1
|            2.2|         0.99|
|        Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2
|          1.615|      0.72675|
|     Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|   4|   1
|          1.835|      0.82575|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+---------------+-------------+
only showing top 20 rows
```

Practice Questions

Question 1 - Basic SQL operations

Display all Mercedez car rows from the `cars` table view we created earlier. The Mercedez cars have the prefix "Merc" in the car name column.

In [21]:

```
# Code block for learners to answer
spark.sql("SELECT * FROM cars WHERE name LIKE 'Merc%'").show()
```

```
+-----------+----+---+-----+---+----+----+----+---+---+----+----+
|       name| mpg|cyl| disp| hp|drat|  wt|qsec| vs| am|gear|carb|
+-----------+----+---+-----+---+----+----+----+---+---+----+----+
|  Merc 240D|24.4|  4|146.7| 62|3.69|3.19|20.0|  1|  0|   4|   2|
|   Merc 230|22.8|  4|140.8| 95|3.92|3.15|22.9|  1|  0|   4|   2|
|   Merc 280|19.2|  6|167.6|123|3.92|3.44|18.3|  1|  0|   4|   4|
|  Merc 280C|17.8|  6|167.6|123|3.92|3.44|18.9|  1|  0|   4|   4|
| Merc 450SE|16.4|  8|275.8|180|3.07|4.07|17.4|  0|  0|   3|   3|
| Merc 450SL|17.3|  8|275.8|180|3.07|3.73|17.6|  0|  0|   3|   3|
|Merc 450SLC|15.2|  8|275.8|180|3.07|3.78|18.0|  0|  0|   3|   3|
+-----------+----+---+-----+---+----+----+----+---+---+----+----+
```

Double-click **here** for a hint.

Double-click **here** for the solution.

Question 2 - User Defined Functions

In this notebook, we created a UDF to convert weight from imperial to metric units. Now for this exercise, please create a pandas UDF to convert the `mpg` column to `kmpl` (kilometers per liter). You can use the conversion factor of 0.425.

In [20]:

```
# Code block for learners to answer
@pandas_udf("float")
def convert_mileage(s: pd.Series) -> pd.Series:
    # The formula for converting from imperial to metric tons
    return s * 0.425

spark.udf.register("convert_mileage", convert_mileage)

spark.sql("SELECT *, mpg AS mpg, convert_weight(mpg) as kmpl FROM cars").show()
```

```
[Stage 16:=====================================================>      (8 +
1) / 9]

+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+----+------+
|               name| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb
| mpg|  kmpl|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+----+------+
|          Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|   4
|21.0|  9.45|
|      Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|   4
|21.0|  9.45|
|         Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1
|22.8| 10.26|
|     Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3|   1
|21.4|  9.63|
|  Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3|   2
|18.7| 8.415|
|            Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|  0|   3|   1
|18.1| 8.145|
|         Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|  0|   3|   4
|14.3| 6.435|
|          Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2
|24.4| 10.98|
|           Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2
|22.8| 10.26|
|           Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4
|19.2|  8.64|
|          Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4
|17.8|  8.01|
|         Merc 450SE|16.4|  8|275.8|180|3.07| 4.07| 17.4|  0|  0|   3|   3
|16.4|  7.38|
|         Merc 450SL|17.3|  8|275.8|180|3.07| 3.73| 17.6|  0|  0|   3|   3
|17.3| 7.785|
|        Merc 450SLC|15.2|  8|275.8|180|3.07| 3.78| 18.0|  0|  0|   3|   3
|15.2|  6.84|
| Cadillac Fleetwood|10.4|  8|472.0|205|2.93| 5.25|17.98|  0|  0|   3|   4
|10.4|  4.68|
|Lincoln Continental|10.4|  8|460.0|215| 3.0|5.424|17.82|  0|  0|   3|   4
|10.4|  4.68|
|  Chrysler Imperial|14.7|  8|440.0|230|3.23|5.345|17.42|  0|  0|   3|   4
|14.7| 6.615|
|           Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1
|32.4| 14.58|
|        Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2
|30.4| 13.68|
|     Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|   4|   1
|33.9|15.255|
+-------------------+----+---+-----+---+----+-----+-----+---+---+----+----
+----+------+
only showing top 20 rows
```

Double-click **here** for the solution.

Authors

Karthik Muthuraman (https://www.linkedin.com/in/karthik-muthuraman/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id
SkillsNetwork-Channel-SkillsNetworkCoursesIBMBD0225ENSkillsNetwork25716109-2021-01-01)

Other Contributors

Jerome Nilmeier (https://github.com/nilmeier)

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-07-02	0.2	Karthik	Beta launch
2021-06-30	0.1	Karthik	First Draft