Skills Network

Getting Started With Spark using Python

Estimated time needed: **15** minutes

The Python API

Spark is written in Scala, which compiles to Java bytecode, but you can write python code to communicate to the java virtual machine through a library called py4j. Python has the richest API, but it can be somewhat limiting if you need to use a method that is not available, or if you need to write a specialized piece of code. The latency associated with communicating back and forth to the JVM can sometimes cause the code to run slower. An exception to this is the SparkSQL library, which has an execution planning engine that precompiles the queries. Even with this optimization, there are cases where the code may run slower than the native scala version. The general recommendation for PySpark code is to use the "out of the box" methods available as much as possible and avoid overly frequent (iterative) calls to Spark methods. If you need to write high-performance or specialized code, try doing it in scala. But hey, we know Python rules, and the plotting libraries are way better. So, it's up to you!

Objectives

In this lab, we will go over the basics of Apache Spark and PySpark. We will start with creating the SparkContext and SparkSession. We then create an RDD and apply some basic transformations and actions. Finally we demonstrate the basics dataframes and SparkSQL.

After this lab you will be able to:

- Create the SparkContext and SparkSession
- Create an RDD and apply some basic transformations and actions to RDDs
- Demonstrate the use of the basics Dataframes and SparkSQL

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.

In [1]:
```
# Installing required packages
!pip install pyspark
!pip install findspark
```

```
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 1.6 MB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 kB 26.7 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285398 sha256=f
d5483117de54c713787496011b9909edf756da9bf8d6113d86cd6b13e4adaec
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/b7/8e/8f/ba5d017af5f502964eb1358e1d496
a8519de1645936b01810e
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.7 pyspark-3.4.1
Collecting findspark
  Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Installing collected packages: findspark
Successfully installed findspark-2.0.1
```

In [3]:
```python
import findspark
findspark.init()
```

In [4]:
```python
# PySpark is the Spark API for Python. In this lab, we use PySpark to initialize the spark cont
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
```

Exercise 1 - Spark Context and Spark Session

In this exercise, you will create the Spark Context and initialize the Spark session needed for SparkSQL and DataFrames. SparkContext is the entry point for Spark applications and contains functions to create RDDs such as `parallelize()`. SparkSession is needed for SparkSQL and DataFrame operations.

Task 1: Creating the spark session and context

In [5]:
```python
# 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()
```

```
23/07/22 16:03:54 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

Task 2: Initialize Spark session

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

In [6]:
```python
spark
```

Out[6]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v2.4.3`
Master	`local[*]`
AppName	`pyspark-shell`

Exercise 2: RDDs

In this exercise we work with Resilient Distributed Datasets (RDDs). RDDs are Spark's primitive data abstraction and we use concepts from functional programming to create and manipulate RDDs.

Task 1: Create an RDD.

For demonstration purposes, we create an RDD here by calling `sc.parallelize()`
We create an RDD which has integers from 1 to 30.

In [7]:
```
data = range(1,30)
# print first element of iterator
print(data[0])
len(data)
xrangeRDD = sc.parallelize(data, 4)

# this will let us know that we created an RDD
xrangeRDD
```

```
1
```

Out[7]:
```
PythonRDD[1] at RDD at PythonRDD.scala:53
```

Task 2: Transformations

A transformation is an operation on an RDD that results in a new RDD. The transformed RDD is generated rapidly because the new RDD is lazily evaluated, which means that the calculation is not carried out when the new RDD is generated. The RDD will contain a series of transformations, or computation instructions, that will only be carried out when an action is called. In this transformation, we reduce each element in the RDD by 1. Note the use of the lambda function. We also then filter the RDD to only contain elements <10.

In [8]:
```
subRDD = xrangeRDD.map(lambda x: x-1)
filteredRDD = subRDD.filter(lambda x : x<10)
```

Task 3: Actions

A transformation returns a result to the driver. We now apply the `collect()` action to get the output from the transformation.

In [9]:
```
print(filteredRDD.collect())
filteredRDD.count()
```

```
[Stage 0:>                                                          (0 + 4) / 4]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Out[9]:
```
10
```

Task 4: Caching Data

This simple example shows how to create an RDD and cache it. Notice the **10x speed improvement**! If you wish to see the actual computation time, browse to the Spark UI...it's at host:4040. You'll see that the second calculation took much less time!

In [10]:
```
import time

test = sc.parallelize(range(1,50000),4)
test.cache()

t1 = time.time()
# first count will trigger evaluation of count *and* cache
count1 = test.count()
dt1 = time.time() - t1
print("dt1: ", dt1)


t2 = time.time()
# second count operates on cached data only
count2 = test.count()
dt2 = time.time() - t2
```

```
print("dt2: ", dt2)

#test.count()
```

```
dt1:  0.9595217704772949
dt2:  0.2413475513458252
```

Exercise 3: DataFrames and SparkSQL

In order to work with the extremely powerful SQL engine in Apache Spark, you will need a Spark Session. We have created that in the first Exercise, let us verify that spark session is still active.

In [11]:
```
spark
```

Out[11]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v2.4.3`
Master	`local[*]`
AppName	`pyspark-shell`

Task 1: Create Your First DataFrame!

You can create a structured data set (much like a database table) in Spark. Once you have done that, you can then use powerful SQL tools to query and join your dataframes.

In [12]:
```
# Download the data first into a local `people.json` file
!curl https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNet
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    73  100    73    0     0    579      0 --:--:-- --:--:-- --:--:--   579
```

In [14]:
```
# Read the dataset into a spark dataframe using the `read.json()` function
df = spark.read.json("people.json").cache()
```

```
23/07/22 16:26:00 WARN execution.CacheManager: Asked to cache already cached data.
```

In [15]:
```
# Print the dataframe as well as the data schema
df.show()
df.printSchema()
```

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

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

In [16]:
```
# Register the DataFrame as a SQL temporary view
df.createTempView("people")
```

Task 2: Explore the data using DataFrame functions and SparkSQL

In this section, we explore the datasets using functions both from dataframes as well as corresponding SQL queries using sparksql. Note the different ways to achieve the same task!

In [17]:
```
# Select and show basic data columns

df.select("name").show()
df.select(df["name"]).show()
spark.sql("SELECT name FROM people").show()
```

```
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+

+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+

+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
```

In [18]:
```
# Perform basic filtering

df.filter(df["age"] > 21).show()
spark.sql("SELECT age, name FROM people WHERE age > 21").show()
```

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+

+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

In [19]:
```
# Perfom basic aggregation of data

df.groupBy("age").count().show()
spark.sql("SELECT age, COUNT(age) as count FROM people GROUP BY age").show()
```

```
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+

[Stage 31:==================================================>      (66 + 9) / 75]
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    0|
|  30|    1|
+----+-----+
```

Question 1 - RDDs

Create an RDD with integers from 1-50. Apply a transformation to multiply every number by 2, resulting in an RDD that contains the first 50 even numbers.

```
In [ ]: # starter code
        # numbers = range(1, 50)
        # numbers_RDD = ...
        # even_numbers_RDD = numbers_RDD.map(lambda x: ..)
```

```
In [ ]: # Code block for learners to answer
```

Question 2 - DataFrames and SparkSQL

Similar to the `people.json` file, now read the `people2.json` file into the notebook, load it into a dataframe and apply SQL operations to determine the average age in our people2 file.

```
In [ ]: # starter code
        # !curl https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsN
        # df = spark.read...
        # df.createTempView..
        # spark.sql("SELECT ...")
```

```
In [ ]: # Code block for learners to answer
```

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

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

Question 3 - SparkSession

Close the SparkSession we created for this notebook

```
In [ ]: # Code block for learners to answer
```

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

Authors

Karthik Muthuraman

Other Contributors

Jerome 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