DATA7201 Practical session – PySpark library

https://coursemgr.uqcloud.net/data7201

Local Mode VS Yarn Mode – Week 6

There are two modes by which Spark tasks can be executed.

- Local Mode
- Yarn Mode

Local Mode

In this mode, the Spark job is executed locally in your zone. Since it runs locally, it tends to be fast for small datasets. However, this shouldn't be used if the size of the data is considerably large.

Yarn Mode

In this mode, the Spark job you submitted gets distributed and gets executed on the nodes in the cluster. This mode is suitable when the size of the data is considerably large where the data is stored on HDFS. It is called Yarn mode because Yarn is responsible for resource allocation for Spark jobs to execute.

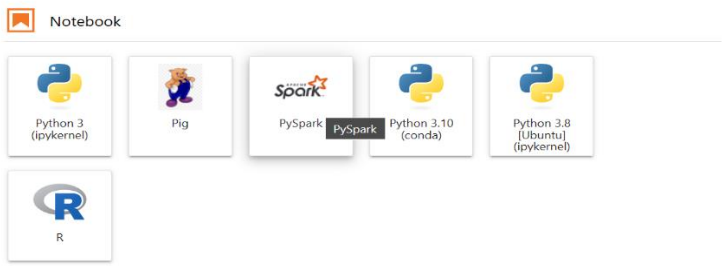
Ways of Executing PySpark Commands

There are two ways by which pyspark can be practiced during this lab.

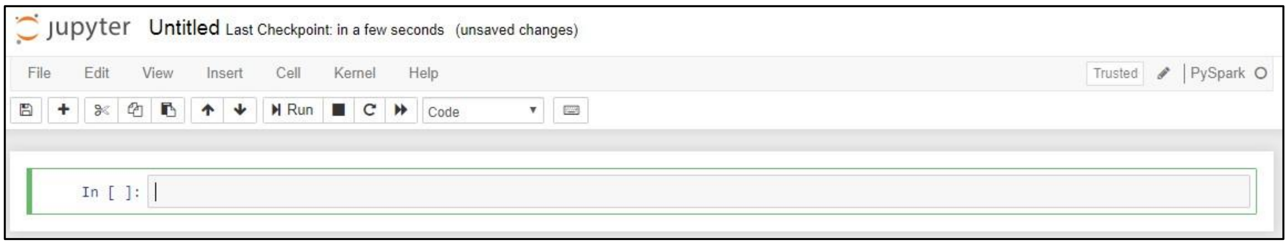
- Jupyter
- Terminal

Option 1 - Accessing PySpark using Jupyter

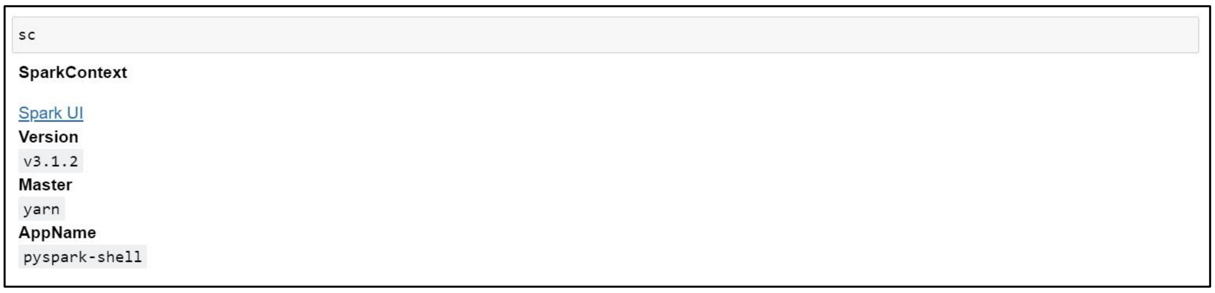
Open Jupyter Lab and select New Launcher> PySpark as shown below.



You will be welcomed with a screen like the one below.


jupyter Untitled Last Checkpoint: in a few seconds (unsaved changes)
File Edit View Insert Cell Kernel Help
Trusted
PySpark
Run
Code
In [ ]:

Spark Context is the starting point of any spark application. It represents the connection to a spark cluster. It's defined in org.apache.spark package. Spark application by default provides "sc" object an instance of SparkContext Class. We can directly use this object when required.



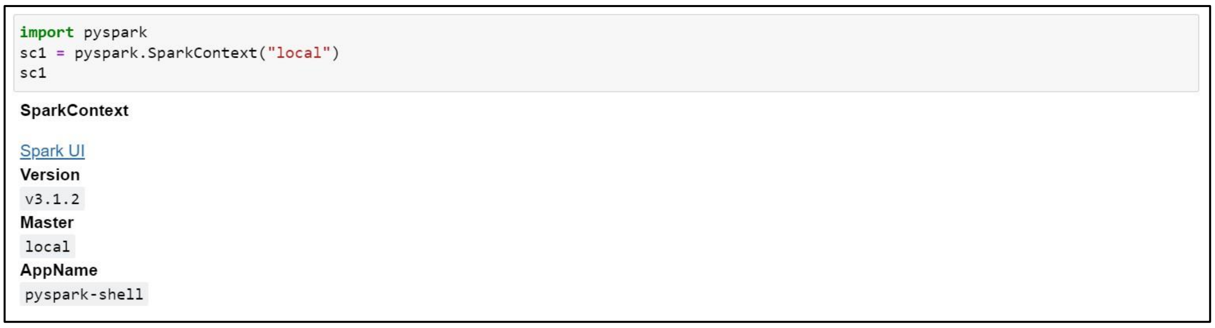
```
sc
```

SparkContext

Spark UI
Version
`v3.1.2`
Master
`yarn`
AppName
`pyspark-shell`

A spark context can also be created using "SparkContext" function. One of the main parameters that it takes is the execution mode, which can be either "local" or "yarn". In the below case it is executed in "local" mode. However, it can be made to execute in yarn mode by replacing "local" with "yarn" below.

Note: At a given point only one SparkContext's instance can be active. Before creating a new instance, stop the existing or current instance by calling stop().



```
import pyspark
sc1 = pyspark.SparkContext("local")
sc1
```

SparkContext

Spark UI
Version
`v3.1.2`
Master
`local`
AppName
`pyspark-shell`

The following exercises and demos can be performed using the SparkContext that you have created above.

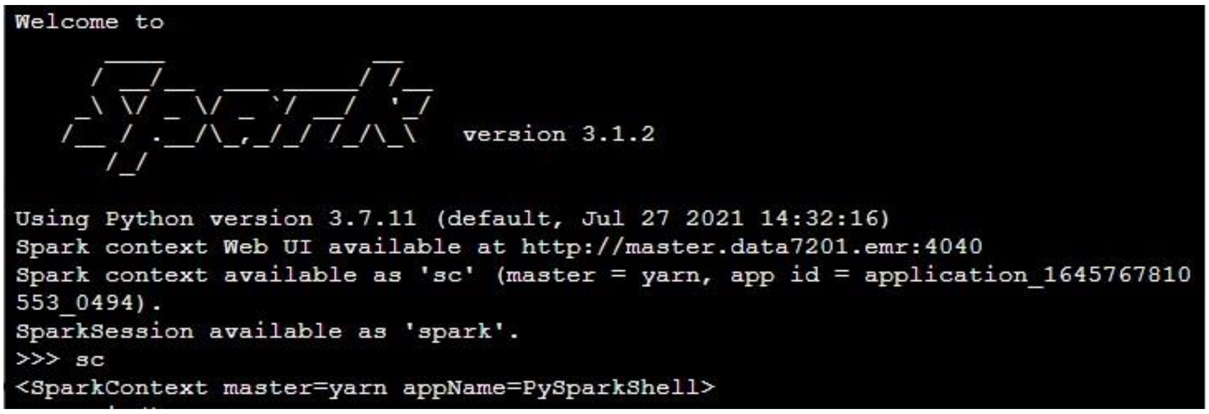
Option 2 - Accessing PySpark using Terminal.

At the data7201- USERNAME shell prompt we can launch pyspark by running

```
[s1@data7201-s1]$ pyspark
```

SparkContext 'local' vs 'yarn client'

Note that this will run pyspark in yarn mode and make full use of cluster. You can check this by asking the following command in the Terminal mode:



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.7.11 (default, Jul 27 2021 14:32:16)
Spark context Web UI available at http://master.data7201.emr:4040
Spark context available as 'sc' (master = yarn, app id = application_1645767810
553_0494).
SparkSession available as 'spark'.
>>> sc
<SparkContext master=yarn appName=PySparkShell>
```

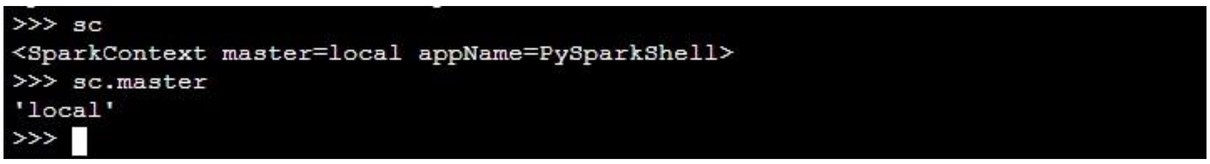
To run pyspark in locally on the node, we first exit this session in the Terminal mode:

```
>>>exit()
```

and run

```
[s1@data7201-s1]$ pyspark --master local
```

Now check again the sparkContext attribute



```
>>> sc
<SparkContext master=local appName=PySparkShell>
>>> sc.master
'local'
>>> 
```

Transformation and Actions

Now, lets work in the Jupyter notebook. Let’s import the necessary packages:

```
>>> from pyspark import SparkConf, SparkContext
```

We will need to create a spark session:

```
>>> conf = (SparkConf()
        .setMaster("yarn") # if "local" then it will ran using the zone server's resources
        .setAppName("sxxxx-prac-4")) # you can choose any name that you want
>>> sc = SparkContext(conf = conf)
```

Important: make sure that the user name is the same as your "s" student id. In case it is different, reboot your zone.

```
>>> sc.sparkUser()
```

The first example first creates an object and then transforms it in an RDD using the 'parallelize' method (https://spark.apache.org/docs/1.6.3/programming-guide.html#parallelized-collections):

```
>>> strings = ["one","two","three"]
>>> s2 = sc.parallelize(strings)
```

We then apply the 'map' transformation. We apply a lambda function which takes 'word' as parameter and returns 'word.upper()' as a result. Note that since this is a transformation it is not immediately computed (lazy execution):

```
>>> s3 = s2.map(lambda word: word.upper())
```

Finally, running an action makes the entire data processing flow to execute on the cluster:

```
>>> s3.collect()
```

`

Lambda Functions

```
>>> a = range(10)
>>> list(a)
>>> rdd = sc.parallelize(a)
>>> rdd.first()
>>> rdd.collect()
```

We can apply functions to our RDDs (in a lazy fashion) defining them as parameters of transformations (e.g., map()):

```
>>> rdd =rdd.map(lambda x: x*10)
```

We can apply functions as parameters of actions (not lazy):

```
>>> rdd.reduce(lambda x,y : x+y)
>>> rdd = rdd.filter(lambda x: x>30)
>>> rdd.collect()
```

EXERCISE 1 :

1. From the object "a", get all elements that are bigger than 5
2. Get the product of the elements resulting from the previous step
3. Get statistics of a by using rdd.stats() from "a".

Getting data from HDFS (word count)

We can also first import data from HDFS as, for example:

```
>>> distFile = sc.textFile("/user/sxxxxx/prac-1/moby10b.txt")
```

We can then apply transformation and action to count words:

```
>>> from operator import add
>>> words = distFile.flatMap(lambda line: line.split(" ")).map(lambda x: (x, 1))
>>> counts = words.reduceByKey(add)
>>> counts.collect()
```

You can also post process the result in Python:

```
>>> output=counts.collect()
>>> for (word, count) in output:
>>>     print("%s: %i" % (word, count))
```

EXERCISE 2:

- Sort the words by frequency and print the top 10 words with more frequency.
  [Hint: use the SortBy method with a function returning the value of the key,value pairs].
  [Hint: use the take method to display the top 10 words].
  https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html#sortBy
  https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html#take

- Sort the words in ascending order and print the top 10.
  [Hint: use the SortByKey].

Using SparkSQL – Week 7

We will be working with only one csv file at the beginning, and then we will load multiple files.

First, we need to load a single csv. We will be using the csv "/data/UKPoliceData2017/2017-07/2017-07-leicestershire-street.csv. It can be loaded by submitting the following code:

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession
   .builder
   .appName("prac-5")  # any name can be put here
   .getOrCreate()

>>> df = spark.read.option("header",True).csv("/data/UKPoliceData2017/2017-07/2017-07-
leicestershire-street.csv")
```

We can now see its content:

```
>>> df.show()
```

Or a subset:

```
>>> df.select('Month','Crime type').show()
```

Look at the columns:

```
>>> df.printSchema()
```

Convert data types:

```
>>> from pyspark.sql.types import *
>>> df = df.withColumn('Month',df['Month'].cast(DateType()))
>>> df.printSchema()
```

Make operations on values:

```
>>> from pyspark.sql.functions import mean, min, max, lit
>>> df.select(min('Month'),max('Month')).show()
```

See more functions here: https://databricks.com/blog/2015/06/02/statistical-and-mathematical-functions-with-dataframes-in-spark.html

We can correlate location and crime type:

```
>>> df.stat.crosstab("LSOA name", "Crime Type").show()
```

We can export it to Pandas, be aware that this option will use your zone resources, don't use it when the data is large.

```
>>> df.stat.crosstab("LSOA name", "Crime Type").toPandas()
```

Find the location of specific crime types:

```
>>> df.filter(df["Crime type"] == 'Burglary').select(df['Crime
type'],df.Latitude,df.Longitude).show()
```

We can also work with SQL directly by transforming the dataframe in a table first:

```
>>> df.createOrReplaceTempView("table1")
>>> spark.sql('select `Crime type`, Latitude, Longitude from table1 where `Crime type` ==
"Burglary"').show()
```

You can add columns to a dataframe:

```
>>> df.withColumn('zero',pyspark.sql.functions.lit(0)).show()

>> df.select('Longitude','Latitude').withColumn('Longitude_times_two', df.Longitude *
2).show()

>>> from pyspark.sql.functions import col, first, last, sum, count, countDistinct, desc
>>> df.select('Latitude', col('Latitude').alias('new_Lat'), (col('Longitude') < 0
).alias('negative_long')).show()
```

Group by crime type and show the first investigation outcome (not very meaningful!):

```
>>> df.groupBy('Crime type').agg(first('Last outcome category').alias("status")).show()
```

You can see the physical execution plan:

```
>>> df.groupBy('Crime type').agg(first('Last outcome category').alias("status")).explain()
```

EXERCISE 3:

- Show how many crimes we have for each crime type (hint: use groupby, agg and count)
- Show distinct 'Last outcome category' for each Crime type (hint: use dropDuplicates)
- Show how many crimes we have for each LSOA code and crime type (hint: groupy by two keys)
- Show the LSOA names where the number of crimes is bigger than 100 (use groupby count and where)
- Sort them by count of crimes in descending order
- See help(df.stat.freqItems) and show the crime type and lsoa name appearing more than 30% (hint: support is 0.3, use show(truncate=False) to see the result)

Using multiple CSV files

We list all files for one location over one year:

```
>>> place = 'west-yorkshire'
>>> year = '2017'
>>> months = ('01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12')

>>> file_names = [f'/data/UKPoliceData2017/{year}-{m}/{year}-{m}-{place}-street.csv' for m
in month]
>>> file_names
```

We can then load all dataframes:

```
>>> annual_dataframes =
(sqlContext.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load(month) for month in file_names)
```

Use the defined function:

```
>>> annual_crimes = spark.read.option("header",True).csv(file_names)
```

And show the total number of crimes by month:

```
>>> annual_crimes.groupby('Month').count().show()
```

Save and Load files in HDFS

Let's create a new folder in your user's folder:

```
hdfs dfs -mkdir prac-5
```

```
hdfs dfs -ls
```

Now, we are going to load the file "flights.csv"

```
>>> df_load = spark.read.option("header",False).csv('hdfs:///data/flights.csv')
```

The last column name needs to be changed

```
>>>  df_load.withColumnRenamed("_c31","id")
>>> df_load.printSchema()
```

The data can be saved in HDFS, we are using csv format.

```
>>> df_load.write.save('/user/uqpvelra/prac-5/flights', format='csv')
```

Notes:

- More information on using Spark and more Python examples: https://spark.apache.org/docs/1.6.3/programming-guide.html
- You can read file from the local file system as well as from HDFS by specifying "file://" or "hdfs://" respectively. When reading from the local file system you should check when launching pyspark on which cluster node your yarn-client is running.
- You can mark an RDD to be persisted using the persist() or cache() methods on it. In Python, objects are serialized as Pickle.