PySpark with Databricks Notebook

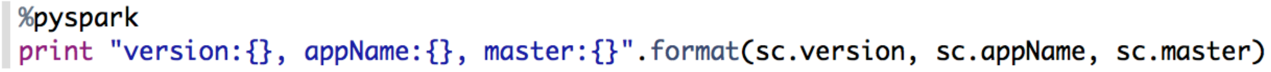
Open the Databricks notebook on your browser. If you have not setup your notebook please refer to “databricksCommunity.pdf” for instructions. There are tasks and exercises in this tutorial. Please answer them in your notebook and send me your work in HTML format either by email (farhang@nailydata.com) or through teams application.

SparkContext and sparkConf

The first step of any Spark driver application is to create a SparkContext. The SparkContext allows your Spark driver application to access the cluster through a resource manager (YARN, or Spark's cluster manager). In order to create a SparkContext instance you should first create a SparkConf. The SparkConf stores configuration parameters that your Spark driver application will pass to SparkContext. Some of these parameters define properties of your Spark driver application e.g. setAppName() gives your Spark driver application a name so you can identify it in the Spark or Yarn UI. Some there configuration parameters (e.g. setMaster command) are used by Spark to allocate resources on the cluster such as the number of cores and memory size used by the executors running on the worker nodes.

In Databricks notebook the SparkContext and the SparkConf are included by default and creation of their instances is not needed. The instance of the SparkContext is represented by “sc”.

To see the default configuration of the Databricks notebook type the following commands[1] :



```
%pyspark
print "version:{}, appName:{}, master:{}".format(sc.version, sc.appName, sc.master)
```

You do not need to write %pyspark on your notebook.
To execute the code either click on the triangle or push shift+enter.

After execution, the spark version, the application name and the number of cores to be used by the cluster are displayed.

RDDs

Resilient Distributed Datasets save the data in a collection of partitions. Any requested transformation are applied in parallel to all partitions of an RDD. This parallel nature of spark and its ability to run the tasks in memory (RAM) makes it one of the fastest management and analysis tools for big data.

[1] Please notice that if you are working with python 3.x, change the print format to print()

Reading and displaying a text file

“http_log_1.txt” is text file that contains the logs of a server including the ip number of users, connection date, viewed file, etc.

Task: Open the text file “http_log_1.txt” by using sc.textFile.

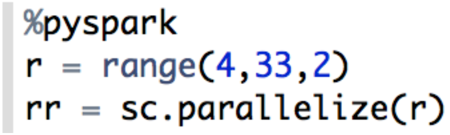
Task: What is the type of the text_file?

Task: The count() function gives back the number of lines of a file. Use text_file.count().

Task: Try the same with first(), “take(2)” and “getNumPartitions()” functions. What do they do?

parallelize

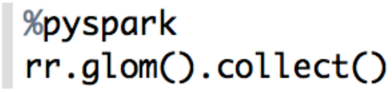
You can transform any Python collection to an RDD by using the parallelize command. Execute the following code:



```
%pyspark
r = range(4,33,2)
rr = sc.parallelize(r)
```

Task: What are the types of r and rr ?
Task: View the 4 first lines of rr.
Task: Run rr.collect(). What does it do ?
Task: How many partitions does rr have ?
Note: “collect()” command is not recommended to be used for big number of data. “take()” should be used instead.

Execute the following line:



```
%pyspark
rr.glom().collect()
```

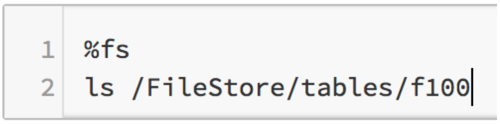
Task: What does the glom() function do ?

Save a text file

Read the text file 100M.data in a variable called “f100”.

Task : Save this file on the disc by using “savAsTextFile” method.

Task : Verify how the file is stored on the disc. Explain why.
Hint :



```
%fs
ls /FileStore/tables/f100
```

Task: Execute: f100. coalesce(1).saveAsTextFile(“/FileStore/tables/f100_2")
Explain how the coalesce() function affect the file storage on the disc.

union, intersection, distinct and subtract

Suppose in a conference there are five meetings (m1,m2, …, m5).

User1 takes [m1, m2, m3]
User2 takes [m2, m3, m4, m5]

Task: Make two lists u1 and u2 then, transform them into RDDs by using the parallelize command.
Hint: For example u1 = [“m1","m2","m3"] is the list for User1.

Task: Find the meetings in common to both users.
Solution: u1.intersection(u2).collect()

Task: Find meeting attended by either user1 or user2. How do you remove the duplicates ?

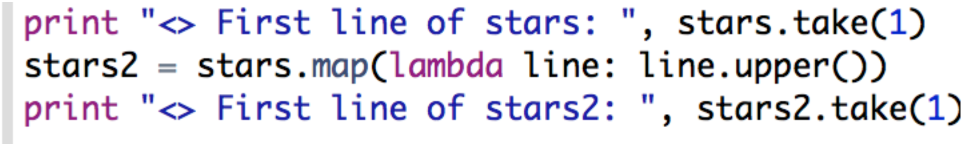
Task: Find meeting only user1 attended (and not user2).

Task: Create two sets of meeting where User1 recommends m1 to User2, and User2 recommends m4 and m5 to User1.

map

The map command applies a function to the RDD. The function can be either a user defined function or a lambda function. The map feeds the file contents element by element to the function. In the following example we transform all characters of the “stars.txt” file to upper case using map and lambda function.

Task: Open the file stars.txt in zeppelin. Then, run the following code:



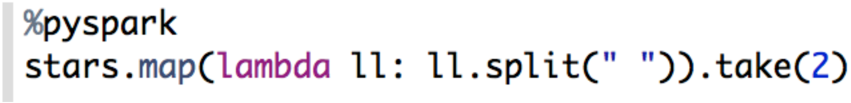
```
print "<> First line of stars: ", stars.take(1)
stars2 = stars.map(lambda line: line.upper())
print "<> First line of stars2: ", stars2.take(1)
```

Task: What is the type of stars2 ? What does upper() function do ?

Task: Play the same game by defining the function named as “makeUpper”. Then apply the map transform with this syntax : stars.map(makeUpper). Print the first line.

Pars a text file (splitting)

Look at the first line of the text_file (“stars.txt”), the fields are separated by space. Hence, this line can be considered as an array of elements separated by spaces. By using the “map” command we can split each line to a collection of elements by taking into account the separator (here is space):



```
%pyspark
stars.map(lambda ll: ll.split(" ")).take(2)
```

The first two lines are shown in two collections (here as lists).

flatMap

Repeat the above line by using the “flatMap” command instead of “map”. What do you infer?

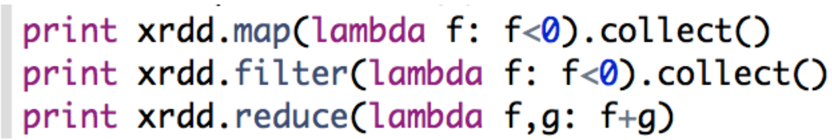
filter

The filter command is useful in applying a logical condition to the RDD to filter the data according to the given condition.

reduce

This function reduces the elements of an RDD using the specified commutative and associative binary operator.

To understand the meaning of the map, filter and reduce, Lets xddd be a RDD of a list of numbers [-2,2] (python list: range(-2,3)). Then run the following lines in your notebook :



```
print xrdd.map(lambda f: f<0).collect()
print xrdd.filter(lambda f: f<0).collect()
print xrdd.reduce(lambda f,g: f+g)
```

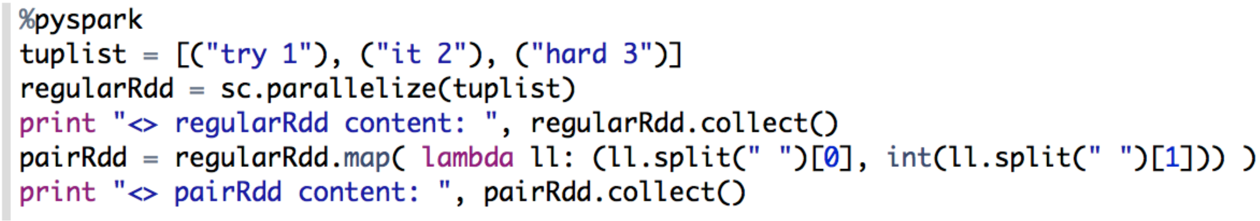
Task: Discuss the results.

Note: map and filter are applied to transform an RDD. The result of a transformation is again an RDD. Whereas reduce is an action and gives a number.

Pair RDD

Pair RDDs have the key-value concept. They can be constructed either from a tuple (or a list of tuples) or RDDs :

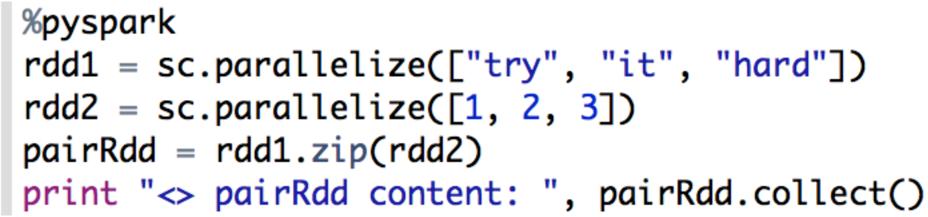
Pair RDD from a list of tuples :



```
%pyspark
tuplist = [("try 1"), ("it 2"), ("hard 3")]
regularRdd = sc.parallelize(tuplist)
print "<> regularRdd content: ", regularRdd.collect()
pairRdd = regularRdd.map( lambda ll: (ll.split(" ")[0], int(ll.split(" ")[1])) )
print "<> pairRdd content: ", pairRdd.collect()
```

Task: Try to completely understand how pairRdd is constructed.
Task: Print the type of regularRdd and pairRdd.

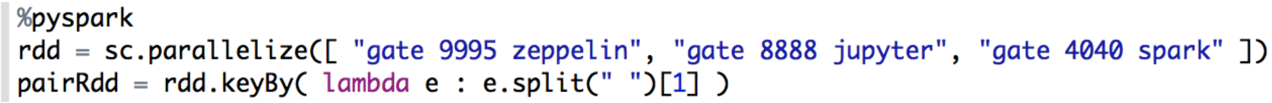
Pair RDD from zipping 2 regular RDDs :



```
%pyspark
rdd1 = sc.parallelize(["try", "it", "hard"])
rdd2 = sc.parallelize([1, 2, 3])
pairRdd = rdd1.zip(rdd2)
print "<> pairRdd content: ", pairRdd.collect()
```

Task: Compare this pairRdd with the previous one.

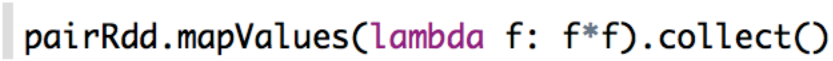
Pair RDD from keyBy :



```
%pyspark
rdd = sc.parallelize([ "gate 9995 zeppelin", "gate 8888 jupyter", "gate 4040 spark" ])
pairRdd = rdd.keyBy( lambda e : e.split(" ")[1] )
```

mapValues

This function replace the map function for pair RDDs:



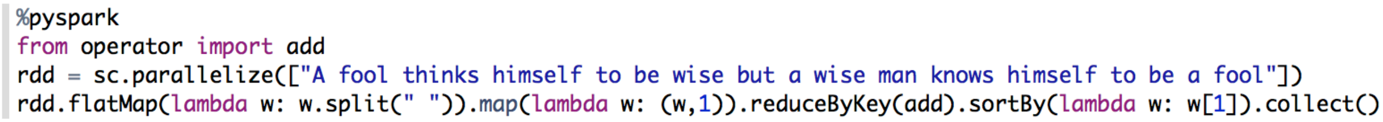
```
pairRdd.mapValues(lambda f: f*f).collect()
```

Task: Try with map instead of mapValues. What happens? Why?

reduceByKey

Spark reduceByKey function merges the values for each key using an associative reduce function. Basically reduceByKey function applies on pair RDDs which contains key-value elements.

Task: Consider the phrase “A fool thinks himself to be wise but a wise man knows himself to be a fool”, then execute the following code :



```
%pyspark
from operator import add
rdd = sc.parallelize(["A fool thinks himself to be wise but a wise man knows himself to be a fool"])
rdd.flatMap(lambda w: w.split(" ")).map(lambda w: (w,1)).reduceByKey(add).sortBy(lambda w: w[1]).collect()
```

You can decompose the last line in four separate lines to see what happens for each command. The map command returns a tuple where the first element (w) is the “key” and the second element (1) is the value for the key.

Task: What is the output of this code?
Task: What lambda function can replace the “add” function in reduceByKey?
Task: What does sortBy do ?

Exercise

1. Créez un RDD numberRdd qui contient la liste des chiffres suivants : [1, 2, 3, 4, 500, 6, 7, 8, 9, 10, 11, 12, 100]

2. Affichez le contenu de numberRdd

3. Affichez les statistiques de numberRdd : moyenne, écart-type , maximum, minimum (utilisez stats())

4. Créez un RDD month1Rdd qui contient la liste des mois suivants : [janvier, février, mars, avril, mai, juin, juillet, août]

5. Créez un RDD month2Rdd qui contient la liste des mois suivants : [septembre, octobre, novembre, décembre]

6. Créez un RDD monthRdd qui regroupe les deux RDD month1Rdd et month2Rdd

7. Créez un RDD monthWithIndexRdd à partir de monthRdd qui contient les indexes et les mois.

8. Ajoutez 1 à chaque index de monthWithIndexRdd (utilisez map())

Exercise

Jeux de données prénoms.txt
Colonnes : prénoms, sexe (m ou f), langage d'utilisation

1. Créez un RDD de type texte file prenomsRdd à partir de fichier Prenoms.csv.

2. Calculez le nombre de lignes du fichier.

3. Retournez les 5 premiers éléments du fichier.

4. Affichez le contenu du RDD.

5. Créez un RDD sexeRdd à partir de prenomsRdd qui ne contient que les sexes des prénoms.

6. A partir de sexeRdd, créez deux RDD (fRdd, mRdd) contenant respectivement le genre féminin et masculin.

7. Calculez le nombre de genre masculin et féminin à partir de fRdd, mRdd.

8. Créez un RDD distinctSexeRdd qui contient les genres distincts de sexeRdd.

9. Enregistrez distinctSexeRdd dans un fichier text.

- 10. Créez un RDD langage1Rdd de prenomsRdd qui ne contient que les langages d'utilisation des prénoms.
  11. Créez un RDD langage2Rdd à partir du RDD langage1Rdd pour que chaque ligne ne contienne qu'un seul langage.

12.Créez un Rdd langagePairRdd à partir de langage2Rdd, contenant pour chaque élément est un couple (langue, 1)

13.Utilisez reduceByKey() pour calculer le nombre d'occurrence de chaque langage d'utilisation dans langagePairRdd.

14.Récupérez les 5 langues qui ont le plus d'occurrence

15. Récupérez les 5 langues qui ont le moins d'occurrence

Exercise : An example for frequency calculation

Compute the frequency of repetition for each word in following phrase :

"A beautiful woman delights the eye ; a wise woman , the understanding ; a pure woman , the soul"

Exercise

Open the file stars.txt. Count how many times each word is repeated in the text. Find the max and min number of repetitions.

Exercise

Open the file "pairExample.txt". Make an RDD with this format :
[(01, May), (03, June), …]

Exercice : resultats_electoraux.csv :

Résultats des élections, Présidentielles, Législatives, Municipales, Européennes et Régionales de 2007 à 2017 par arrondissement, bureau et candidat. Publié sur https://opendata.paris.fr/

Colonnes :

libellé du scrutin, date du scrutin, commune paris 01 à 20, nombre d'inscrits du bureau de vote, nombre de votants du bureau de vote, nom du candidat ou liste, prénom du candidat ou liste, nombre de voix du candidat.

1. Créez un RDD de type texte file rsltElect1Rdd à partir de fichier : resultats_electoraux.csv.

2. Calculez le nombre de lignes du fichier et affichez la première ligne.

3. Ouvrez l'interface Web Spark port 8080 pour surveiller l'état du cluster ainsi que le port 4040 pour surveiller l'avancement des traitements de Job.

4. Créez un RDD rsltElect2Rdd de rsltElect1Rdd, chaque élément est un objet JSON contenant les champs suivants : lib, tour, annee, commune, nbr_inscrits, nbr_votants, nom_prenom, nombre_voix.

5. Enregistrez rsltElect2Rdd dans un seul fichier texte.

6. Filtrez les données présidentielles du premier tour de 2017.

7. Quel sont les deux candidats qui ont le plus de votes ?

Dataframes and SparkSeission

Prior to Spark 2.0.0 sparkContext was used as a channel to access all spark functionality. The spark driver program uses spark context to connect to the cluster through a resource manager (YARN). sparkConf is required to create the spark context object, which stores configuration parameter like appName (to identify your spark driver), application, number of core and memory size of executor running on worker nodes.
In order to use APIs of SQL**,** HIVE**,** and Streaming, separate contexts need to be created.

SPARK 2.0.0 onwards, SparkSession provides a single point of entry to interact with underlying Spark functionality and allows programming Spark with **DataFrame** and Dataset APIs. All the functionality available with sparkContext are also available in sparkSession. In order to use APIs of SQL, HIVE, and Streaming, no need to create separate contexts as sparkSession includes all the APIs.
(https://data-flair.training/forums/topic/sparksession-vs-sparkcontext-in-apache-spark/)

A data frame is a table, or two-dimensional array-like structure, in which each column contains measurements on one variable, and each row contains one case (register or entry). A DataFrame has additional metadata due to its tabular format, which allows Spark to run certain optimisations on the finalised query.

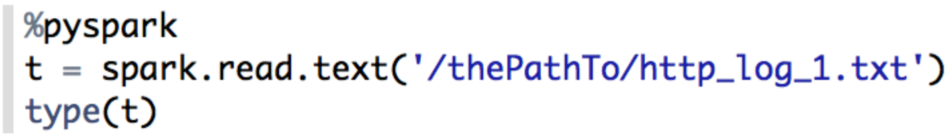
An RDD, on the other hand, is merely a Resilient Distributed Dataset that is more of a blackbox of data that cannot be optimised by the operations since it is immutable.

However, you can go from a DataFrame to an RDD via its "rdd" method, and vice versa (if the RDD is in a tabular format) via the "toDF" method. In general it is recommended to use a DataFrame where possible due to the built-in query optimisation.
(https://stackoverflow.com/questions/31508083/difference-between-dataframe-dataset-and-rdd-in-spark)

In Databricks notebook the instance of SparkSession is created by default as "spark".

Reading and displaying a text file

We reopen the text file "http_log_1.txt" this time with SparkSession :

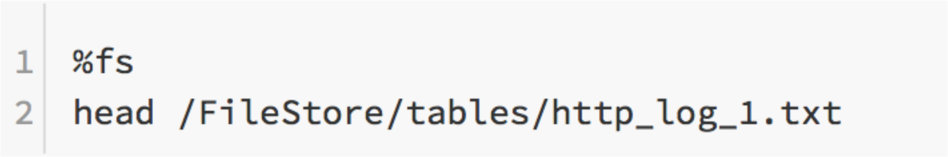


```
%pyspark
t = spark.read.text('/thePathTo/http_log_1.txt')
type(t)
```

Task: To visualise the text file execute " t.show() ". Then try with " t.show(10, False) ". What is the difference?
Task: Use the "count()" command to count the number of lines.

Parsing a text file and converting to a DataFrame

We are going to make a data frame from a text file that has a tabular format but no header (no schema). Before start to code, we have a look at the content of the text file.



```
%fs
head /FileStore/tables/http_log_1.txt
```

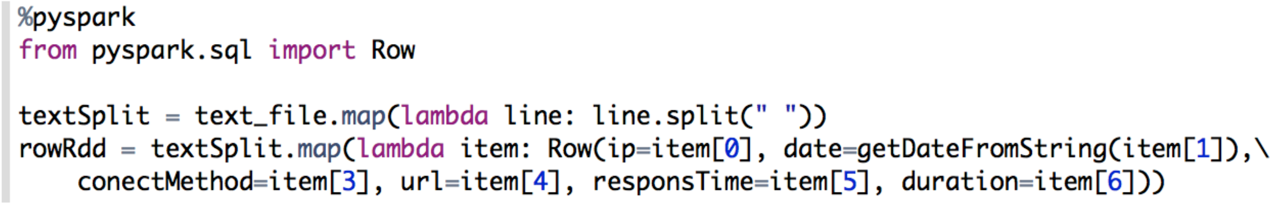
As you see http_log_1.txt can be considered as a columnar file where the 7 columns are separated by space. The columns are respectively the *ip address, the date, zeros column* (+0000])*, connection method, url, response and the connection duration*. We are going to pars this file, transform the date into timestamp type, give name to each column and save all the columns except the zeros column to a DataFrame.

In Spark we usually talk about the “schema” of a dataframe that means the name and type of the columns.

Task: Firstly, the date column has the generic form of “[26/July/2018:10:05:43” which should be converted to standard timestamp. Write a function called “getDateFromString” to do the conversion.

Task : Read the text file “ http_log_1.txt “ as an RDD called “text_file”.

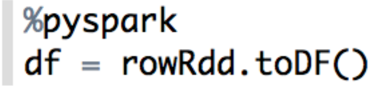
Task : Split the “text_file” elements into another RDD that is called textSplit, then:



```
%pyspark
from pyspark.sql import Row

textSplit = text_file.map(lambda line: line.split(" "))
rowRdd = textSplit.map(lambda item: Row(ip=item[0], date=getDateFromString(item[1]),\
    conectMethod=item[3], url=item[4], responsTime=item[5], duration=item[6]))
```

Task: What is the type of the rowRdd ? Print the first element.



```
%pyspark
df = rowRdd.toDF()
```

Task: What is the type of df ?
Task: Show the first four lines of “df”.
Task: Print the schema of the “df” by using “df.printSchema()”.
Note: To convert an RDD to a DataFrame the RDD should have the tabular form. The conversion can be done by calling the “.toDF()” method.

Task: Do the same job, this time change the type of duration to integer.

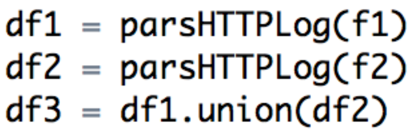
Exercise

Look at data.txt by using linux “head” (we have already seen this command). It is a text file with tabular format separated by “,”. It contains 3 columns: id (string), url (string) and index (int). Open the file as an RDD then pars it by assigning a name to each column. At last make a DataFrame. Show the schema and the first three lines of the DataFrame.

union

Task: Define a function “parseHTTPLog(f)” that do all the jobs of the previous step, i.e. read, pars and convert the text file “f” to a DataFrame.

Then run the following lines:



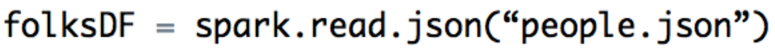
```
df1 = parsHTTPLog(f1)
df2 = parsHTTPLog(f2)
df3 = df1.union(df2)
```

Where f1="http_log_1.txt" and f2=“http_log_2.txt"

Task: How many line do df1, df2 and df3 have ? What does the union method do ?

filter

Open the JSON (JavaScript Object Notation) file people.json :

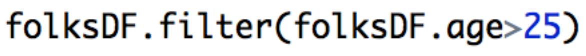


```
folksDF = spark.read.json(“people.json”)
```

JSON is an open-standard file format that uses human-readable text to transmit data. It has a key–value pair structure.

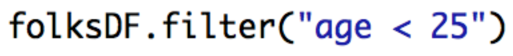
Task: View folksDF schema and print the first line.

Task: Apply the filter function to find people older than 25.
Solution: For filtering, you can either use the python dot notation:



```
folksDF.filter(folksDF.age>25)
```

or apply the sql notation:



```
folksDF.filter("age < 25")
```

Task: show the people with 25 < age < 50 with both notations.

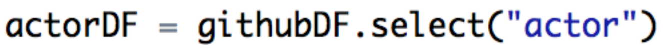
selection

Task: Consider the file github.json which is an archive file containing github activity for a single day. Open this file as githubDF and print the first 5 lines.

You see that It’s hard to understand and view because there are so much data in a single row.

Task: View the schema of githubDF.
You may notice that some columns have several sub-structures. This is one of the properties of JSON file format.

Task: Make a new DataFrame by selecting the actor column :



```
actorDF = githubDF.select("actor")
```

You can also use the select function with dot notation: githubDF.select(**githubDF.actor**)

Task: View the schema of the actorDF and show the first 5 lines.
There are less data to view so it is easier to see the details. As you see, there are nested structures such as avatar, url, id, login, etc.

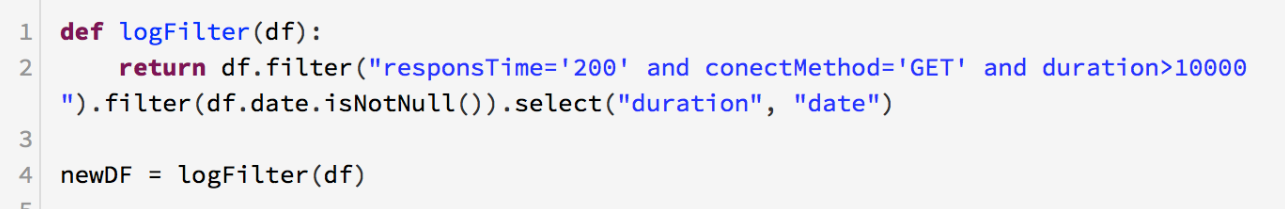
Task: Select the login value of actor and display a few lines.
Hint: use the dot notation.

Task: Use the disntinct() function to count how many unique logins are in data.

Task: githubDF contains a type column. Select this column and show the unique values.

Task: Consider the “type” column of githubDF. Count the number of rows containing “CreateEvent”. Show 5 rows of this kind.
Hint: You should use both selection and filter commands.

You saw that you can filter a DataFrame by applying any logical conditions on the columns. It is also possible to make a new DataFrame from an existing one by selecting any desirable columns. Execute the following code on the already existed “df”:



```
def logFilter(df):
    return df.filter("responsTime='200' and conectMethod='GET' and duration>10000").filter(df.date.isNotNull()).select("duration", "date")

newDF = logFilter(df)
```

Task: Verify the type of newDF.
Task: What does this code do? Show the first 5 lines.
Task: Try the following line:

```
%pyspark
newDF.rdd.getNumPartitions()
```

Note: To get the number of partitions for a DataFrame it should be converted to an RDD. The " .rdd " command does the job.

Exercise

Make a new DataFrame from github.json by selecting three columns: "type", "id" from "actor" and "line" from "payload". Count the number of ids larger than 1,000,000. Show 10 first lines where "line" values are not null (use built-in isNotNull() function).

Plotting with Databricks Notebook

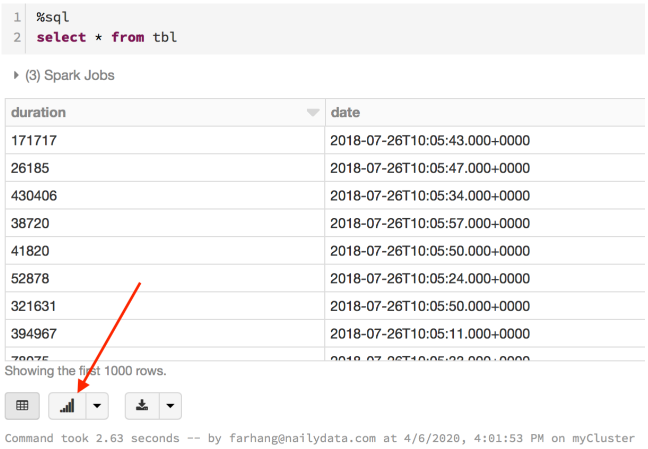
You can plot the columns of a data frame versus each other by using sql interpreter (%sql) of Databricks notebook. Firstly, you should register the DataFrame on the memory as a temporary table :

```
%pyspark
newDF.registerTempTable("tbl")
```

Then, select the columns of "tbl" in sql interpreter:

```
%sql
select * from tbl
```

Click on the "Bar Chart", then click on "Plot Options" :



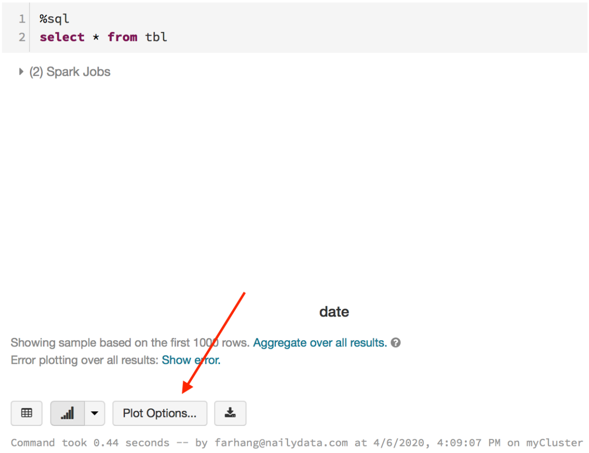
```
%sql
select * from tbl
```

▸ (3) Spark Jobs

duration	date
171717	2018-07-26T10:05:43.000+0000
26185	2018-07-26T10:05:47.000+0000
430406	2018-07-26T10:05:34.000+0000
38720	2018-07-26T10:05:57.000+0000
41820	2018-07-26T10:05:50.000+0000
52878	2018-07-26T10:05:24.000+0000
321631	2018-07-26T10:05:50.000+0000
394967	2018-07-26T10:05:11.000+0000
78975	2018-07-26T10:05:22.000+0000

Showing the first 1000 rows.

Command took 2.63 seconds -- by farhang@nailydata.com at 4/6/2020, 4:01:53 PM on myCluster



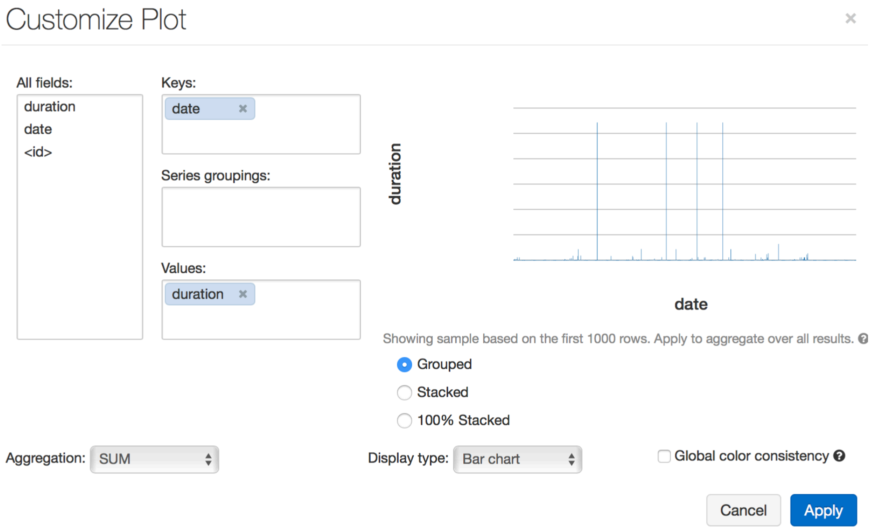
```
%sql
select * from tbl
```

▸ (2) Spark Jobs

Showing sample based on the first 1000 rows. Aggregate over all results.
Error plotting over all results: Show error.

Command took 0.44 seconds -- by farhang@nailydata.com at 4/6/2020, 4:09:07 PM on myCluster

From “All Fields” box, drag the date to the “Keys” and duration to “Values” boxes and click “Apply” :



Customize Plot

All fields:
duration
date
<id>

Keys: date

Series groupings:

Values: duration

Showing sample based on the first 1000 rows. Apply to aggregate over all results.

● Grouped
○ Stacked
○ 100% Stacked

Aggregation: SUM

Display type: Bar chart

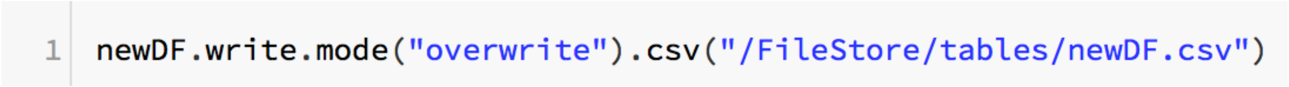
□ Global color consistency

Pass some moments to understand the plot and play with different options of this plotting utility. Plotting is one the first steps of data analysing and extracting information from the data.

Reading and displaying a csv file

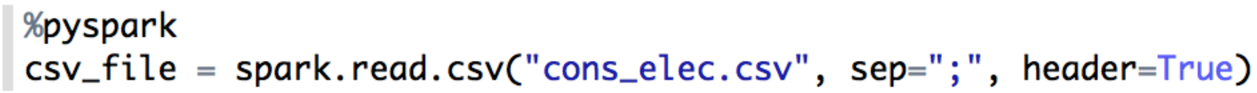
A comma-separated values (CSV) file is a delimited text file that uses comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

To save the “newDF” DataFrame as a csv file:



```
newDF.write.mode("overwrite").csv("/FileStore/tables/newDF.csv")
```

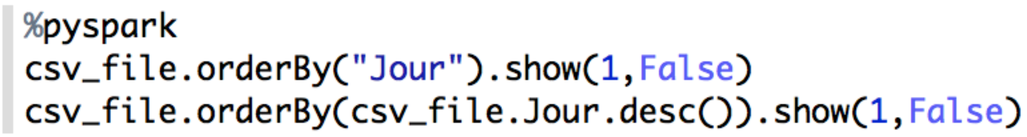
The following code reads and shows a csv file as a data frame. cons_elec.csv is a public dataset considering the daily electricity consumption (in watt unit) in France for different consumer categories. The data in each line are separated by ‘ ; ’ :



```
%pyspark
csv_file = spark.read.csv("cons_elec.csv", sep=";", header=True)
```

Task : Verify the type of csv_file.
Task : Print the schema and four first 3 rows of csv_file.
Task : Remove the “sep” and “header” terms and reprint the first three lines. What happens?

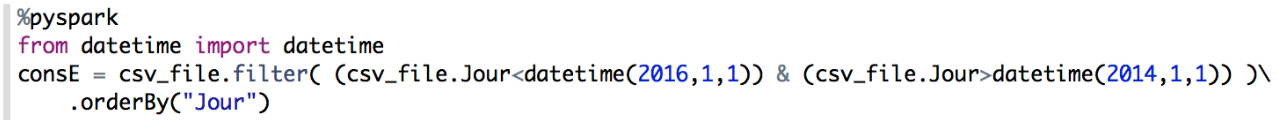
Task: What does this code do ? :



```
%pyspark
csv_file.orderBy("Jour").show(1,False)
csv_file.orderBy(csv_file.Jour.desc()).show(1,False)
```

We are going to plot the consumed power versus the time. We choose the consumption measurements between 2014 and 2016.

Task: Discuss and execute the following code :



```
%pyspark
from datetime import datetime
consE = csv_file.filter( (csv_file.Jour<datetime(2016,1,1)) & (csv_file.Jour>datetime(2014,1,1)) )\
    .orderBy("Jour")
```

Task: Save consE in a temporary table to be displayed by sql interpreter. Plot Jour versus Puissance moyenne journalière in sql interpreter (%sql). This notebook has limitation in plotting all the data, limit your sql select command to limit 1000.

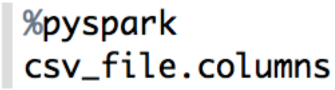
Task: Can you explain the peak of the plot? What do the small maxima and minima correspond to?

Task: Plot the “catégorie cliente” versus “Puissance moyenne journalière”. What do you infer?

Manipulating the DataFrames

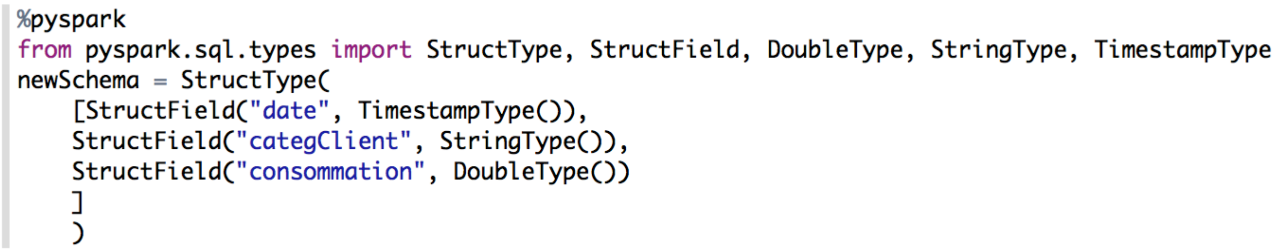
We can change the column names, modify any column contents, add new columns and fill them by values. This is the most important difference between the RDDs and DataFrames. RDDs are immutable.

Look at the column names of the already existed “csv_file” data-frame:



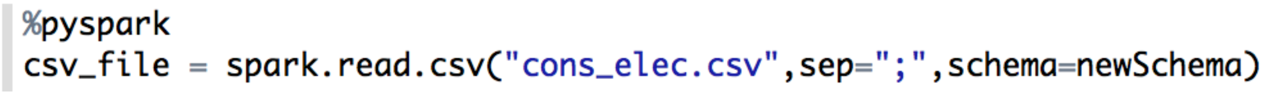
```
%pyspark
csv_file.columns
```

The characters are not well displayed. We can change the name of any column by defining a new data frame schema. We define the “newSchema” function as:



```
%pyspark
from pyspark.sql.types import StructType, StructField, DoubleType, StringType, TimestampType
newSchema = StructType(
    [StructField("date", TimestampType()),
    StructField("categClient", StringType()),
    StructField("consommation", DoubleType())
    ]
    )
```

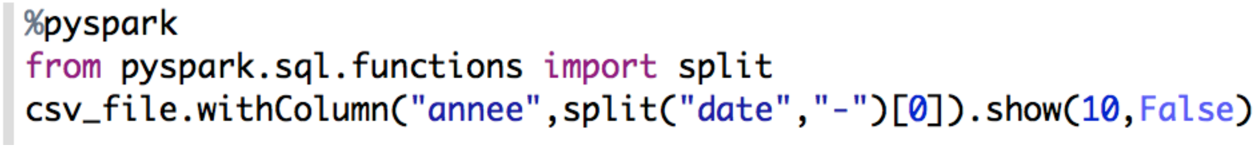
We have defined a new schema by using “StructType” where includes the new column names and their corresponding types. You can choose any name you wish for the columns. We then reopen the “cons_elec.csv” by using the newSchema:



```
%pyspark
csv_file = spark.read.csv("cons_elec.csv",sep=";",schema=newSchema)
```

Task: View the schema of csv_file.

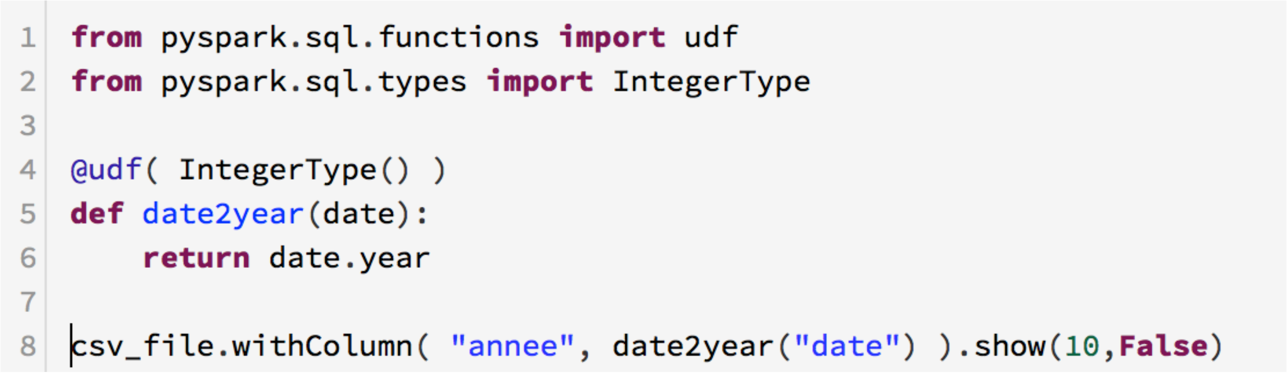
You can see the column names have been changed. You may want to add a new column called “annee” filled by the year of the measurement by extracting the year from the “date” column:



```
%pyspark
from pyspark.sql.functions import split
csv_file.withColumn("annee",split("date","-")[0]).show(10,False)
```

“withColumn” method makes a new column. The first argument is the column name and the second is the column values.

You can do the same job by importing and applying a User Defined Function (udf):



```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

@udf( IntegerType() )
def date2year(date):
    return date.year

csv_file.withColumn( "annee", date2year("date") ).show(10,False)
```

Notice : We try to avoid writing a UDF as much as possible. There are lots of built-in well-optimised functions for PySpark available here.

Exercise : Electricity consumption in France

Open the energy consumption file, cons_elec.csv as a DataFrame.

Task 1 : View the schema of the DataFrame. Change the column names and types by defining a new schema as follow : Date (timestamp), CategClient (string) and Pmoyenne (float).

Task 2 : View the unique years of measurement in descending order. View the unique client categories.

Task 3 : Plot Pmoyenne vs. Date for “Entreprises” for any arbitrary year. Explain why the plot has a comb shape ? Why does it not change for different seasons ?

Task 4 : Compute the mean energy consumption per day for each year and each client category. Call the new column as “MoyenneAnnuelle”. Plot (the distribution of) this mean value for different clients in different years. Explain why in 2013 the mean electricity consumption per day is larger than the other years ?

Exercise : White house visits

Read the file “visits.txt” as an RDD. It is a tabular text file with no schema. The column should be “lsatNam” (string), “firstName” (string), “arrivalTime” (timestamp), “appointmentTime” (timestamp), “meetingLocation” (string) and “event” (string) respectively.

Task 1 : Make a DataFrame from this RDD.

Task 2 : How many have visited White House for the event: DOMESTIC VIOLENCE AWARENESS MONTH LARGE MEETING ?

Task 3 : How many have visited White House for: WAITING FOR SUPERMAN DROP BY VISIT ?

Task 4 : How many “MEDAL OF HONOR CEREMONY” events were organised in 2010 ? How many after 2010 ?

Task 5: How many people were delayed ?

Task 6 : Add a column called “Timing” to your DataFrame. Fill the new column with “Delayed” for the delayed people, “On time” for those on time, and "Arrival time not registered” for those with null arrival time add.

Task 7 : Make a DataFrame with two columns (appointmentTime and event) that contains only the data for 2010. From this DataFrame extract the unique months where the “MEDAL OF HONOR CEREMONY” was awarded.

Exercise : Elections in Paris

Open the file resultats_electoraux.csv. This is a tabular file with columns separated by “;” with no schema. The column names are :
libellé du scrutin, date du scrutin, commune paris 01 à 20, nombre d'inscrits du bureau de vote, nombre de votants du bureau de vote, nom du candidat ou liste, prénom du candidat ou liste, nombre de voix du candidat.

Task 1 : From this file make a DataFrame with the following schema :

1- Label, string
2- Date, timestamp
3- Commune, string
4- nEligibles, integer
5- nElecteurs, integer
6- Nom, string
7- Prenom, string
8- nVotes, integer

Task 2 : Replace all “??” by “e”.

Task 3 : What are the unique labels ? From label column construct two new string columns : “scrutinType” (containing the type of the election e.g. Presidentielle) and “Tour” (containing the election round, 1, 2 or -). Delete the “Label” column.

Task 4 : Make a year column (“Annee”). How many unique years exist for all elections ? What elections are held in those years ?

Task 5 : For first round of presidential election in 2017 verify that all candidates have the same total number of “nEligibles” and “nElecteurs”. Do the same verification by districts.

Task 6 : Plot the participation rate for different Paris districts for first round of presidential election in 2017.

Task 7 : Plot the vote rate for each candidate in different districts for first round of presidential election in 2017.

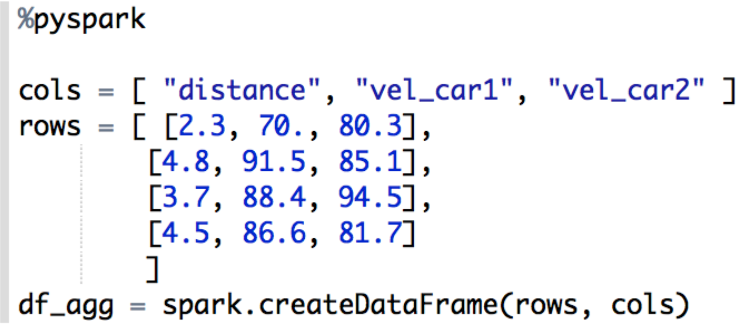
So far you are done with lots of Spark's commands and operations, congratulation !

In the next steps we will deal with more real life cases in terms of data cleaning and analysis. As either a data engineer or a data scientist maybe more the 50% of the your data work belongs to data cleaning process. Since we aim to learn the principles of Spark programming in this tutorial, we are less concerned by data cleaning which is a global problem regardless of what programming language or platform you use.

We continue our tutorial by applying aggregation and join methods on Covid19 data.

Aggregation

Let's make a dataframe about cars velocity-distance with 3 columns. First column is the distance passed and other columns indicate the velocity of two different cars in km/h.

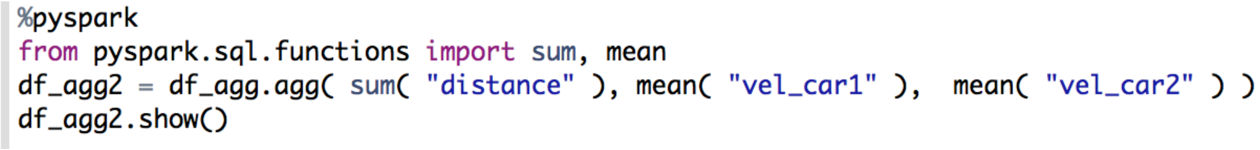


```
%pyspark

cols = [ "distance", "vel_car1", "vel_car2" ]
rows = [ [2.3, 70., 80.3],
         [4.8, 91.5, 85.1],
         [3.7, 88.4, 94.5],
         [4.5, 86.6, 81.7]
         ]
df_agg = spark.createDataFrame(rows, cols)
```

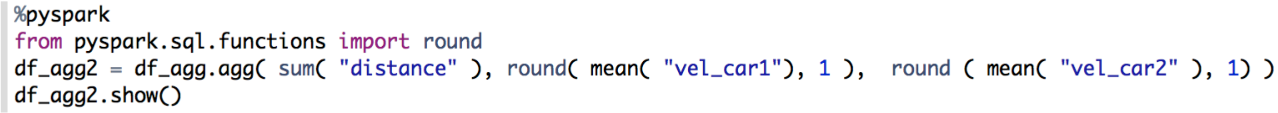
Task : Show the df_agg and its schema.

We are going to compute the total distance passed and mean velocity of the cars :



```
%pyspark
from pyspark.sql.functions import sum, mean
df_agg2 = df_agg.agg( sum( "distance" ), mean( "vel_car1" ),  mean( "vel_car2" ) )
df_agg2.show()
```

Having lots of decimals is meaningless. To round the numbers :



```
%pyspark
from pyspark.sql.functions import round
df_agg2 = df_agg.agg( sum( "distance" ), round( mean( "vel_car1"), 1 ),  round ( mean( "vel_car2" ), 1) )
df_agg2.show()
```

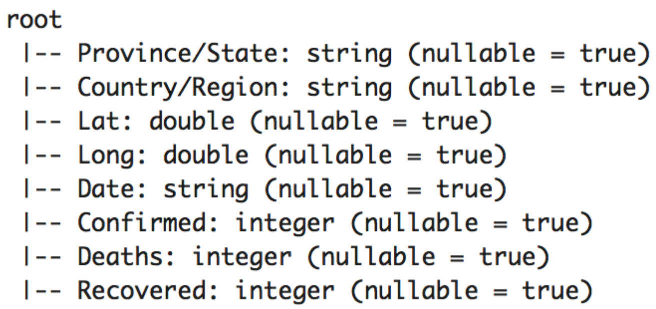
Task : Rename the columns of df_agg2 to "totDistance", "vel_avg1" and "vel_avg1" by applying the withColumnRenamed method.

Groupby

In this part we are going to analyse the Covid-19 data for confirmed, dead and recovered persons in different countries.

Open the file “covid19.csv” in a dataframe called “covid19”.
Task : Print a few initial lines and the schema of covid19.

You should see something like this for the schema :



```
root
 |-- Province/State: string (nullable = true)
 |-- Country/Region: string (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- Date: string (nullable = true)
 |-- Confirmed: integer (nullable = true)
 |-- Deaths: integer (nullable = true)
 |-- Recovered: integer (nullable = true)
```

Task : From covid19 make a new data from called “covid” with following criteria :

1. Includes all column except “Province/State”
2. Rename the “Country/Region” to “country”
3. The dates are in string format. Convert them to timestamp by using the appropriate function.
4. Convert all column names to lower case.

We have omitted the “Province/State” column that contained the distribution of the number of cases in different regions of each country. Hence, the number of cases for countries are not aggregated.

Task : For each country per date calculate the total number of “confirmed”, “deaths” and “recovered” columns. Call the dataframe “covid2”. Order the data by country and date.

Task : Save the the covid2 data frame with a parquet format, “covid.parquet” in the DBFS as a single (one partition) file.

Cross join

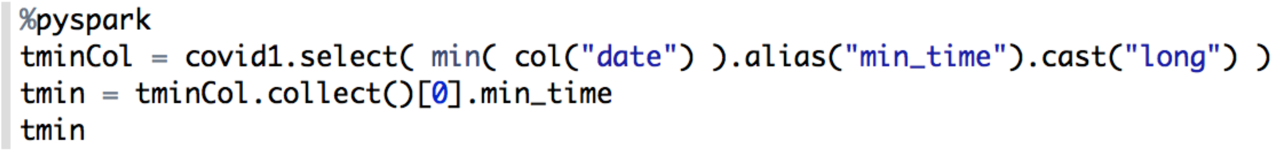
At the end of this cours we are going to compute the contagion rate of this pandemic. To do so we need to know the daily increase in “confirmed”, “deaths” and “recovered” data which are the cumulative numbers. For each country, we should subtract the values of the consecutive days to compute the daily growth of these three columns.

In contrary to Pandas package of Python, in Spark we have only column access and not the element access. That’s because in Spark a dataframe is distributed in different partitions. For this reason, to compute the daily growth we make a self join of the dataframe (Cartesian product) and keep only the rows where the date difference is one day.

Task :

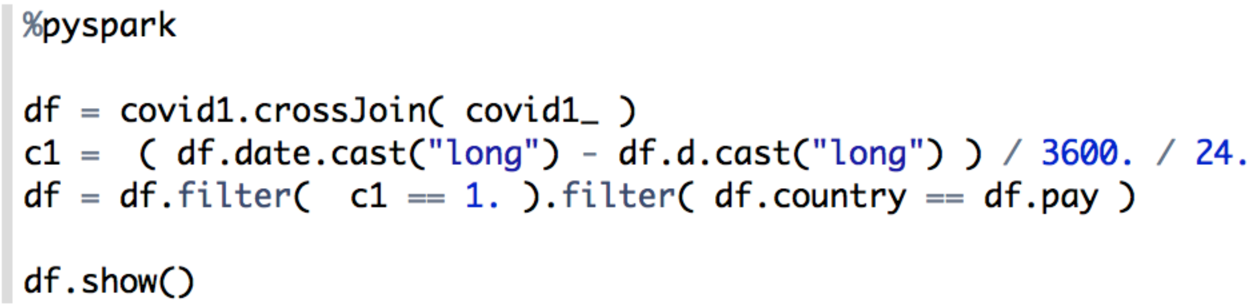
1. Read “covid.parquet” as a datframe called “covid1”.
2. Make a copy of “covid1” in a new dataframe called “covid1_” with column renamed as :
   country —> pay,
   date —> d,
   conformed —> conf
   deaths —> morts

Task : This is good to know how we can convert a dataframe column to Python list or variable. Run the following code and say what “tmin” is ? What is the type and physical unit of “timin” ?



```
%pyspark
tminCol = covid1.select( min( col("date") ).alias("min_time").cast("long") )
tmin = tminCol.collect()[0].min_time
tmin
```

Now, we apply the cross join between “covid1” and “covid1_” :



```
%pyspark

df = covid1.crossJoin( covid1_ )
c1 =  ( df.date.cast("long") - df.d.cast("long") ) / 3600. / 24.
df = df.filter(  c1 == 1. ).filter( df.country == df.pay )

df.show()
```

Task : Explain what happens at each line. Compare the “date” and “d” column of the “df” dataframe.

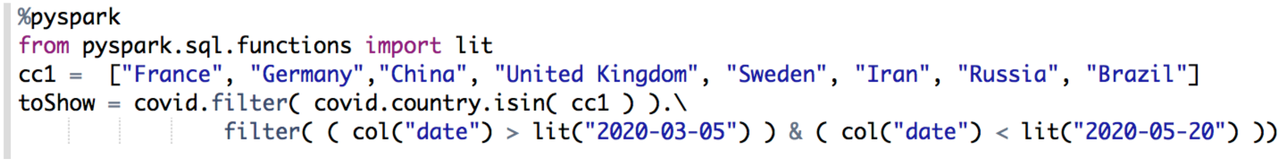
Notice that although the cross join can be both time and memory consuming but the filter commands make it quick and cheap.

Task : From “df” make a new dataframe called “covid” with following criteria :

1. Drop "conf", "morts", "gueris", “pay" and “d" columns.
2. Add three new columns : “newConfirmed”, “newDeaths" and “newRecovered” which are the daily growth.

Task : Print 100 lines for France and have a look in numbers. You see that some cleanings will be needed.

Task : Plot the confirmed ill persons versus time for following countries and time interval :



```
%pyspark
from pyspark.sql.functions import lit
cc1 =  ["France", "Germany","China", "United Kingdom", "Sweden", "Iran", "Russia", "Brazil"]
toShow = covid.filter( covid.country.isin( cc1 ) ).\
                filter( ( col("date") > lit("2020-03-05") ) & ( col("date") < lit("2020-05-20") ))
```

What are the countries with deeper slops ? Do you infer any anomalies in France data ?

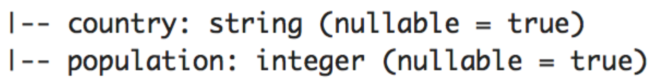
Task : Add the US to the plot.

Inner join

We need to add the countries' population to the "covid" dataframe for further statistical analysis. The number of deaths differs from country to country. To have a clearer comprehension of the situation we should normalise the death number to country population.

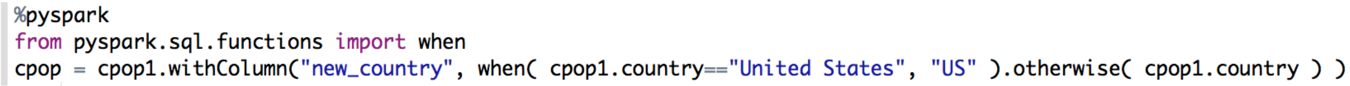
Task : Open the "countryPop.csv" file in a dataframe called "cpop1" then print the schema.

You should see :



```
|-- country: string (nullable = true)
|-- population: integer (nullable = true)
```

We are going to join it with "covid" on "country" column. Before joining two dataframes, we should be sure that the country names are the same in both dataframes. You may want to very it in a systematic way. Here I just show the US case that is called "United States" in "cpop1" dataframe. Hence, we change it as follow :



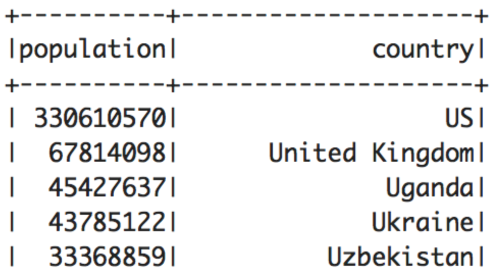
```
%pyspark
from pyspark.sql.functions import when
cpop = cpop1.withColumn("new_country", when( cpop1.country=="United States", "US" ).otherwise( cpop1.country ) )
```

Notice how "when" and "otherwise" pair works.

Task : Drop the "country" column and then rename the "new_country" column as "country".

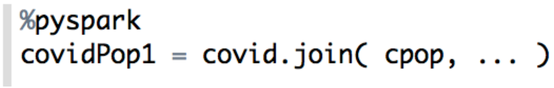
Task : Print the "cpop" dataframe for all countries start by "U".

You should see :



```
+----------+--------------------+
|population|             country|
+----------+--------------------+
| 330610570|                  US|
|  67814098|      United Kingdom|
|  45427637|              Uganda|
|  43785122|             Ukraine|
|  33368859|          Uzbekistan|
```

Task : Inner join the "covid" and "cpop" dataframes on "country" columns :



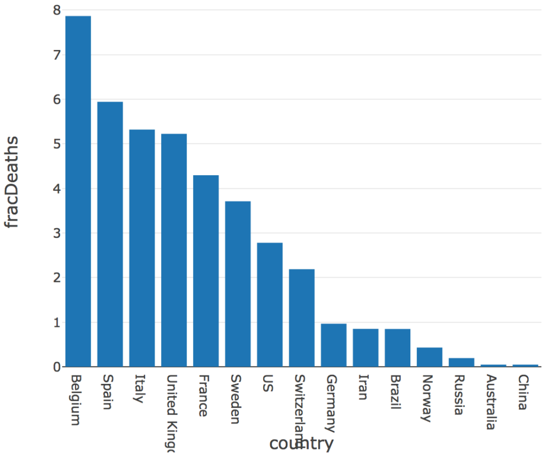
```
%pyspark
covidPop1 = covid.join( cpop, ... )
```

Print the schema. You should have the "population" column with integer type.

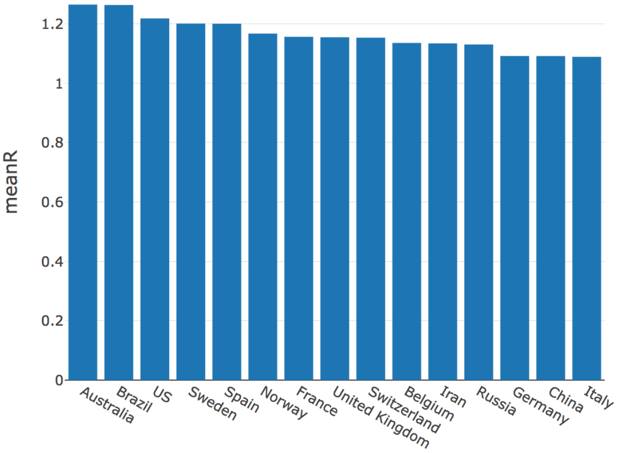
Exercise : Death Rate

Compute the rate of number of sicks, deaths and recovered persons per 10,000 inhabitant for each country. Plot the death rate for following countries for the last day in the data sorted by death rate :

"France", "Germany","China", "Belgium", "Italy", "Spain", "United Kingdom", "Sweden", "US", "Switzerland", "Norway", "Iran", "Russia", "Australia", "Brazil"



Exercise : Mean Day-to-day Confirmed Ratio



It is not easy to compare the data acquisition methods applied by different countries. There are known and unknown differences. However, we can compute the ratio of daily growth of new confirmed cases per country.

This ca be computed as the ratio between the number of new confirmed cases in a day to those in the previous day.

Plot the distribution of the mean day-to-day contagion ratio for the period of 15/02/2020 to 15/05/2020 for the same countries of the previous exercise.

Exercise : Statistical Comparison

Sort the Mean Day-to-day Confirmed Ratio for all countries from 15 February to 15 May. Define a criterion to show if the difference between the min and max ratio are statistically significant.

Exercise : Normalised vs un-normalised deaths

Plot the normalised deaths for "France", "Germany","China", "Belgium", "Italy", "Spain", "United Kingdom", "Sweden", "US", "Switzerland", "Norway", "Iran", "Russia", "Australia", “Brazil" from 15 February to 15 May. Compare it to un-normalised plot for the same countries.

Exercise : Binned data analysis

We are going to bin the daily new confirmation cases in 2-week intervals to compute the number of bins with decrease/increase in virus propagation per country. The solution can be divided in following tasks [2]:

Task 1 : Read the covid19.csv file and apply necessary filed type conversion.

Task 2 : For each country accumulate the daily data over different states.

Task 3 : Compute the new daily confirmations and call it “newConfirmed”.

Task 4 : Plot the “newConfirmed” vs. time for Brazil.

Task 5 : Aggregate the “newConfirmed” over bins of 14 days. Call the corresponding column as “bnc” (binned new confirmed)
Hint : Use window function

Task 6 : Print the schema and five first lines of the binned dataframe. Consider the nested structure of the window column for next task.

Task 7 : Make a dataframe containing only the start time of the window column as “date”.

Task 8 : Plot the “bnc” vs. “date” for Brazil and compare it to the previous plot.

Task 9 : Redo the Task 5 by keeping the “newConfimed” data in a list for each bin. Call the new column as “bncList”. The list size should be 14 maximum. Print a few lines of the new dataframe and compare “bnc” and “bncList” columns.
Hint : Use collect_list function.

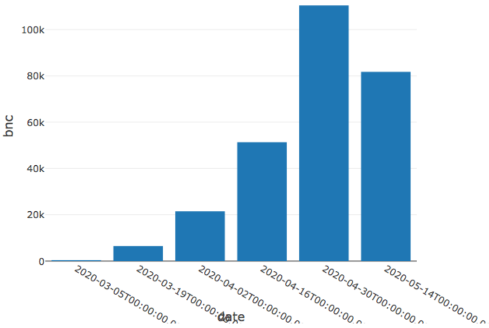
Task 10 : Write a User Defined Function (udf) that reads bncList column, fits a straight line to data points (by using corresponding Python packages) and returns the slope of the fitted line as a float value. If the length of “bncList” is less than 7 return “none” value. Call the new column “slope”. What is the meaning of positive and negative slops ?

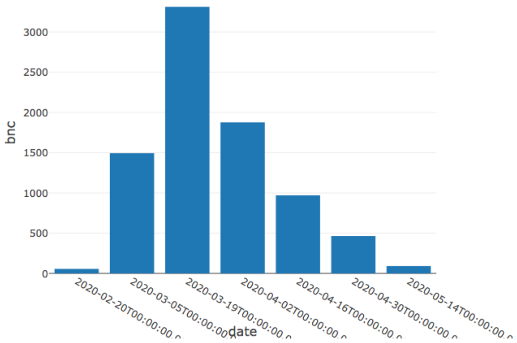
Task 11 : Add a new column called “slopeSign” which is either “+” if the slope is positive or “-” if it is negative. Group the dataframe by country and “slopeSign” and count the corresponding signs. Print the dataframe contains 3 columns (“country”, “slopeSign” and “count”). What are the total number of bins for Brazil and Norway?

Task 12 : Consider only the bins with increase in virus propagation and sort the dataframe by descending order. Compare the values for Brazil and Norway.

Task 13 : Compare the “bnc” vs. date (with null slopes excluded) for Brazil and Norway. Can you explain the number of increase and decrease in virus propagation from the plots ? (the lockdown started on 12 March for Norway)

[2] This may help https://spark.apache.org/docs/latest/api/python/pyspark.sql.html


100k
80k
60k
40k
20k
0
bnc
2020-03-05T00:00:00.0
2020-03-19T00:00:00.0
2020-04-02T00:00:00.0
2020-04-16T00:00:00.0
2020-04-30T00:00:00.0
2020-05-14T00:00:00.0
date


3000
2500
2000
1500
1000
500
0
bnc
2020-02-20T00:00:00.0
2020-03-05T00:00:00.0
2020-03-19T00:00:00.0
2020-04-02T00:00:00.0
2020-04-16T00:00:00.0
2020-04-30T00:00:00.0
2020-05-14T00:00:00.0
date