databrickssolutionsDFsTutorialDatabricks

DataFrames: Reading and displaying a text file in Spa

```
path = "/FileStore/tables/"
f = path+"http_log_1.txt"
t = spark.read.text(f)

print ( "<> show(): \n", t.show(10) )
print ( "<> show(10,False): \n", t.show(10,False) )
print ( "<> Number of entries: ", t.count() )
```

```
+--------------------+
|               value|
+--------------------+
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
|83.149.9.216 [26/...|
+--------------------+
only showing top 10 rows

<> show():
 None
+-----------------------------------------------------------------------------------------------------------------------------------------------+
|value
```

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

```
[Truncated to first 65536 bytes]
83.149.9.216 [26/July/2018:10:05:43 +0000] GET /presentations/logstash-monitorama-2018/images/kibana-dashboard3.png 200 171717
83.149.9.216 [26/July/2018:10:05:47 +0000] GET /presentations/logstash-monitorama-2018/plugin/highlight/highlight.js 200 26185
83.149.9.216 [26/July/2018:10:05:12 +0000] GET /presentations/logstash-monitorama-2018/plugin/zoom-js/zoom.js 200 7697
83.149.9.216 [26/July/2018:10:05:07 +0000] GET /presentations/logstash-monitorama-2018/plugin/notes/notes.js 200 2892
```

```
83.149.9.216 [26/July/2018:10:05:34 +0000] GET /presentations/logstash-monitorama-
2018/images/sad-medic.png 200 430406
83.149.9.216 [26/July/2018:10:05:57 +0000] GET /presentations/logstash-monitorama-
2018/css/fonts/Roboto-Bold.ttf 200 38720
83.149.9.216 [26/July/2018:10:05:50 +0000] GET /presentations/logstash-monitorama-
2018/css/fonts/Roboto-Regular.ttf 200 41820
83.149.9.216 [26/July/2018:10:05:24 +0000] GET /presentations/logstash-monitorama-
2018/images/frontend-response-codes.png 200 52878
83.149.9.216 [26/July/2018:10:05:50 +0000] GET /presentations/logstash-monitorama-
2018/images/kibana-dashboard.png 200 321631
83.149.9.216 [26/July/2018:10:05:46 +0000] GET /presentations/logstash-monitorama-
2018/images/Dreamhost_logo.svg 200 2126
```

```
from datetime import datetime

def getDateFromString(inpDate, formatDate="[%d/%B/%Y:%H:%M:%S"):
    return datetime.strptime(inpDate, formatDate)

getDateFromString('[26/July/2018:10:05:43')
```

```
Out[10]: datetime.datetime(2018, 7, 26, 10, 5, 43)
```

```
from pyspark.sql import Row

text_file = sc.textFile( "/FileStore/tables/http_log_1.txt" )
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=int(item[6])) )
print ( "<> Type of RowRdd: ", type(rowRdd) )
print ( "<> First elements of rowRdd: \n", rowRdd.first() )
```

```
<> Type of RowRdd:  <class 'pyspark.rdd.PipelinedRDD'>
<> First elements of rowRdd:
 Row(conectMethod='GET', date=datetime.datetime(2018, 7, 26, 10, 5, 43), duration=1
71717, ip='83.149.9.216', responsTime='200', url='/presentations/logstash-monitoram
a-2018/images/kibana-dashboard3.png')
```

```
df = rowRdd.toDF()
print ( "<> Type of df: ", type(df) )
print ( "<> The first 4 rows: \n", df.show(4,False) )
print ( "<> The df schema: \n", df.printSchema() )
```

```
<> Type of df:  <class 'pyspark.sql.dataframe.DataFrame'>
+------------+------------------+--------+------------+-----------+--------------
--------------------------------------------------------+
```

```
|conectMethod|date                |duration|ip          |responsTime|url
|
+------------+--------------------+--------+------------+-----------+--------------
-------------------------------------------------------+
|GET         |2018-07-26 10:05:43|171717  |83.149.9.216|200        |/presentations/
logstash-monitorama-2018/images/kibana-dashboard3.png |
|GET         |2018-07-26 10:05:47|26185   |83.149.9.216|200        |/presentations/
logstash-monitorama-2018/plugin/highlight/highlight.js|
|GET         |2018-07-26 10:05:12|7697    |83.149.9.216|200        |/presentations/
logstash-monitorama-2018/plugin/zoom-js/zoom.js       |
|GET         |2018-07-26 10:05:07|2892    |83.149.9.216|200        |/presentations/
logstash-monitorama-2018/plugin/notes/notes.js        |
+------------+--------------------+--------+------------+-----------+--------------
-------------------------------------------------------+
only showing top 4 rows

<> The first 4 rows:
 None
root
 |-- conectMethod: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- duration: long (nullable = true)
 |-- ip: string (nullable = true)
 |-- responsTime: string (nullable = true)
 |-- url: string (nullable = true)

<> The df schema:
 None
```

Exercise

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

```
[Truncated to first 65536 bytes]
01AO1NYA06OAAUCT,http://www.mystore.com/page94,77
8J9M391K0BP16XC1,http://www.mystore.com/page14,1062
08AFGIB7Y74VVTCL,http://www.mystore.com/page90,688
JAFYWO8CH3CG38G6,http://www.mystore.com/page65,399
4TE3J25I675XDC2C,http://www.mystore.com/page98,804
CLNMY9A01YIHMOR0,http://www.mystore.com/page53,285
EKI00YDVIQH0IF3D,http://www.mystore.com/page45,1052
BCS02B2SU4MUJXNE,http://www.mystore.com/page88,813
2G1BTXK3R4D2GJJD,http://www.mystore.com/page16,520
IO1F4QCFJJLH3H93,http://www.mystore.com/page11,1154
35DOCI81LEYAEBBL,http://www.mystore.com/page93,440
VMWCKHXQOKH84OH9,http://www.mystore.com/page28,984
```

```
LD9E3KW551EUT41H,http://www.mystore.com/page90,642
7PMVGZG6YGE8L2M0,http://www.mystore.com/page3,606
TMZ1IOLZUMLO44JK,http://www.mystore.com/page93,74
RO3T2SR5LAAI4IFZ,http://www.mystore.com/page72,516
TK4E9QMKWG051HBK,http://www.mystore.com/page75,788
RF39VY60U19517K5,http://www.mystore.com/page18,327
3WUBIBZI19HBR1XR,http://www.mystore.com/page20,808
```

```
from pyspark.sql import Row

rdd = sc.textFile( "/FileStore/tables/data.txt" )
dft = rdd.map( lambda line : line.split( "," ) ).\
  map( lambda e : Row( id = e[0], url = e[1], index = int(e[2]) ) ).\
  toDF()

print( dft.printSchema() )
dft.show(3, False)
```

```
root
 |-- id: string (nullable = true)
 |-- index: long (nullable = true)
 |-- url: string (nullable = true)

None
+----------------+-----+----------------------------+
|id              |index|url                         |
+----------------+-----+----------------------------+
|01AO1NYA06OAAUCT|77   |http://www.mystore.com/page94|
|8J9M391K0BP16XC1|1062 |http://www.mystore.com/page14|
|08AFGIB7Y74VVTCL|688  |http://www.mystore.com/page90|
+----------------+-----+----------------------------+
only showing top 3 rows
```

DataFrames: union

```
def parsHTTPLog(fname):
    text_file = sc.textFile(fname)
    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=int(item[6])) )
    return rowRdd.toDF()

f1 = path+"http_log_1.txt"
f2 = path+"http_log_2.txt"

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

print ( "Number of lines of f1: ", df1.count() )
print ( "Number of lines of f2: ", df2.count() )
print ( "Number of lines of f3: ", df3.count() )

Number of lines of f1:  3500
Number of lines of f2:  3500
Number of lines of f3:  7000
```

DataFrames: filter

```
from pyspark.sql.functions import col
f = path + "people.json"
folksDF = spark.read.json(f)

print ( "<> folksDF Schema: \n", folksDF.printSchema() )
print ( "<> First line: ", folksDF.first() )

print ( "<> People younger than 25 (dot notation): ",
folksDF.filter(folksDF.age<25).collect() )
print ( "<> People younger than 25 (sql notation): ",
folksDF.filter("age<25").collect() )

print ( "<> People with 25<age<50 (dot notation): ", folksDF.filter(
(folksDF.age>25) & (folksDF.age<50)).show() )

print ( "<> People with 25<age<50 (col ): ", folksDF.filter( (col("age")>25) &
(col("age")<50)).show() )

print ( "<> People with 25<age<50 (sql notation): ", folksDF.filter("age>25 and
age<50").show() )
```

```
root
```

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

<> folksDF Schema:
 None
<> First line:  Row(age=35, gender='M', name='John')
<> People younger than 25 (dot notation):  [Row(age=20, gender='M', name='Mike')]
<> People younger than 25 (sql notation):  [Row(age=20, gender='M', name='Mike')]
+---+------+----+
|age|gender|name|
+---+------+----+
| 35|     M|John|
| 40|     F|Jane|
+---+------+----+

<> People with 25<age<50 (dot notation):  None
+---+------+----+
|age|gender|name|
+---+------+----+
```

DataFrames: selection

```
f = path+"github.json"
githubDF = spark.read.json( f )

print ( "<> githubDF schema: \n", githubDF.printSchema() )
print ( "<> Five first lines: \n", githubDF.take(5) )
#print ( "<> Five first lines: \n", githubDF.show(5, False) )
```

```
root
 |-- actor: struct (nullable = true)
 |    |-- avatar_url: string (nullable = true)
 |    |-- gravatar_id: string (nullable = true)
 |    |-- id: long (nullable = true)
 |    |-- login: string (nullable = true)
 |    |-- url: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- id: string (nullable = true)
 |-- org: struct (nullable = true)
 |    |-- avatar_url: string (nullable = true)
 |    |-- gravatar_id: string (nullable = true)
 |    |-- id: long (nullable = true)
 |    |-- login: string (nullable = true)
 |    |-- url: string (nullable = true)
 |-- payload: struct (nullable = true)
 |    |-- action: string (nullable = true)
 |    |-- before: string (nullable = true)
```

```
 |    |-- comment: struct (nullable = true)
 |    |    |-- _links: struct (nullable = true)
 |    |    |    |-- html: struct (nullable = true)
```

```
print ( "<> Five first lines: \n", githubDF.show(5) )
```

```
+--------------------+--------------------+----------+--------------------+--------
------------+------+--------------------+-----------------+
|               actor|          created_at|        id|                 org|
payload|public|                repo|             type|
+--------------------+--------------------+----------+--------------------+--------
------------+------+--------------------+-----------------+
|[https://avatars....|2015-03-01T00:00:00Z|2614896652|[https://avatars....|[,,,, ,,
,,, maste...|  true|[23934080, Early-...|      CreateEvent|
|[https://avatars....|2015-03-01T00:00:00Z|2614896653|                null|[, 6dda2
86a3a1c25...|  true|[31481156, bezerr...|        PushEvent|
|[https://avatars....|2015-03-01T00:00:00Z|2614896654|                null|[, 6089c
e1d78dc0a...|  true|[31475673, demian...|        PushEvent|
|[https://avatars....|2015-03-01T00:00:00Z|2614896656|                null|[created
,, [, ```...|  true|[31481077, chrsmi...|IssueCommentEvent|
|[https://avatars....|2015-03-01T00:00:00Z|2614896657|                null|[created
,, [, Eve...|  true|[14652644, tedsan...|IssueCommentEvent|
+--------------------+--------------------+----------+--------------------+--------
------------+------+--------------------+-----------------+
only showing top 5 rows

<> Five first lines:
 None
```

```
actorDF = githubDF.select("actor")
print ( "<> actorDF schema: \n")
print (actorDF.printSchema() )
print ( "<> actoDF first 5 lines: \n")
print ( actorDF.show( 5, False) )

loginDF = actorDF.select(actorDF.actor.login.alias("login") )
print ( "<> login first 3 lines: \n")
print( loginDF.show( 3, False ) )
print ( "<> Number of unique logins: ")
print (loginDF.distinct().count() )

dfType = githubDF.select("type").distinct().show(10, False)

typeCrDF= githubDF.select("type").filter("type like '%CreateEvent%' ")
##typeCrDF= githubDF.select("type").filter(githubDF.type.like('%CreateEvent%'))
print ( "<> Number of rows with CreateEvent type: ", typeCrDF.count() )
print ( "<> Five first rows of typeCrDF: \n")
print ( typeCrDF.show(5,False) )
```

```
<> actorDF schema:

root
 |-- actor: struct (nullable = true)
 |    |-- avatar_url: string (nullable = true)
 |    |-- gravatar_id: string (nullable = true)
 |    |-- id: long (nullable = true)
 |    |-- login: string (nullable = true)
 |    |-- url: string (nullable = true)

None
<> actoDF first 5 lines:

+--------------------------------------------------------------------------------
------------------------------------------------+
|actor
|
+--------------------------------------------------------------------------------
------------------------------------------------+
|[https://avatars.githubusercontent.com/u/739622?, , 739622, treydock, https://api
.github.com/users/treydock]                     |
```

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

newDF = logFilter(df)

print ("<> newDF first 5 lines: \n", newDF.show(5,False) )
```

```
+--------+-------------------+
|duration|date               |
+--------+-------------------+
|171717  |2018-07-26 10:05:43|
|26185   |2018-07-26 10:05:47|
|430406  |2018-07-26 10:05:34|
|38720   |2018-07-26 10:05:57|
|41820   |2018-07-26 10:05:50|
+--------+-------------------+
only showing top 5 rows

<> newDF first 5 lines:
 None
```

```
newDF.rdd.getNumPartitions()
```

```
Out[17]: 2
```

```
display( newDF )
```

```
  NameError: name 'newDF' is not defined
```

Exercise

```
gitSelDF = githubDF.select( "type", githubDF.actor.id.alias("id"),
githubDF.payload.comment.line.alias("line") )
fitFiltDF = gitSelDF.filter( " id < 1000000 " )
#print( fitFiltDF.count() )
fitFiltDF.filter( fitFiltDF.line.isNotNull() ).show(3, False)
```

```
+------------------+------+----+
|type              |id    |line|
+------------------+------+----+
|CommitCommentEvent|356564|21  |
|CommitCommentEvent|299842|46  |
|CommitCommentEvent|299842|16  |
+------------------+------+----+
only showing top 3 rows
```

```
dff = githubDF.select( "id", githubDF.actor.id.alias("id2") )
dff.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- id2: long (nullable = true)
```

DataFrames: Reading and displaying a csv file

```
newDF.write.mode("overwrite").csv("/FileStore/tables/newDF.csv", header=True)
```

```
spark.read.csv( "/FileStore/tables/newDF.csv", header=True, inferSchema=True
).show(2, False)
```

```
+--------+-------------------+
|duration|date               |
+--------+-------------------+
|60656   |2018-07-27 01:05:21|
|36492   |2018-07-27 01:05:03|
+--------+-------------------+
only showing top 2 rows
```

```
f = path + "cons_elec.csv"
csv_file = spark.read.csv(f, sep=";", header=True, inferSchema=True)
print ( "<> csv_file type: ", type(csv_file) )
print ( "<> csv_file schema: \n", csv_file.printSchema() )
print ( "<> csv_file 3 rows: \n", csv_file.limit(3).show() )

test = spark.read.csv(f)
print ( "<> test 3 rows: \n", test.limit(3).show() )
```

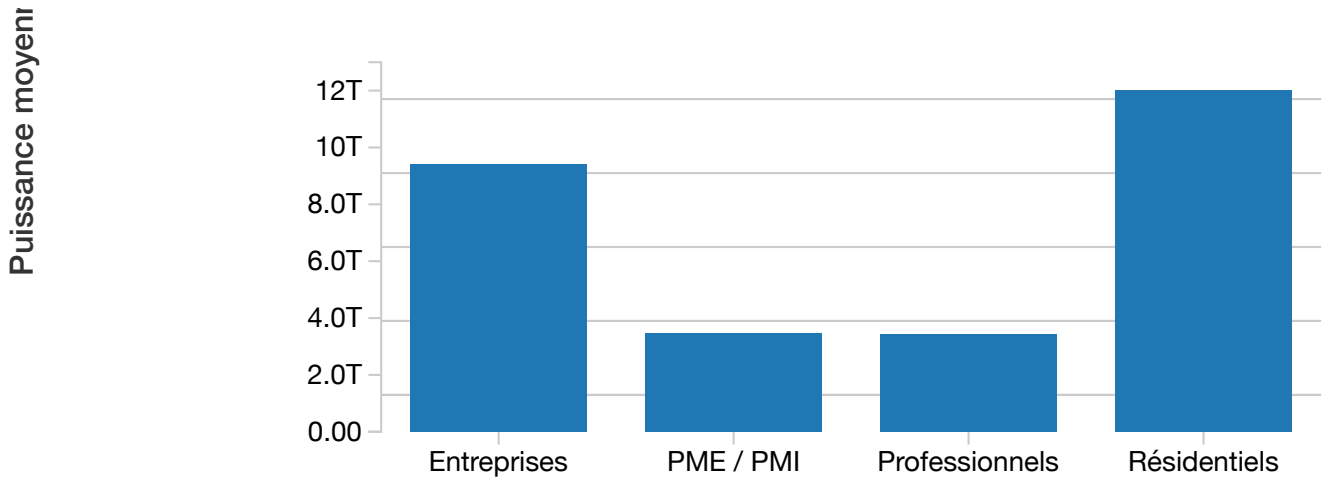
```
<> csv_file type:  <class 'pyspark.sql.dataframe.DataFrame'>
root
 |-- Jour: timestamp (nullable = true)
 |-- Catégorie client: string (nullable = true)
 |-- Puissance moyenne journalière: long (nullable = true)

<> csv_file schema:
 None
+-------------------+----------------+-----------------------------+
|               Jour|Catégorie client|Puissance moyenne journalière|
```

```
+-------------------+----------------+-----------------------------+
|2015-11-13 00:00:00|     Entreprises|                  13882225479|
|2015-11-11 00:00:00|  Professionnels|                   3746294820|
|2015-12-18 00:00:00|  Professionnels|                   5523363437|
+-------------------+----------------+-----------------------------+

<> csv_file 3 rows:
 None
+--------------------+
|                 _c0|
```

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

```
+-------------------+----------------+-----------------------------+
|Jour               |Catégorie client|Puissance moyenne journalière|
+-------------------+----------------+-----------------------------+
|2013-08-25 00:00:00|PME / PMI       |3297476378                   |
+-------------------+----------------+-----------------------------+
only showing top 1 row

+-------------------+----------------+-----------------------------+
|Jour               |Catégorie client|Puissance moyenne journalière|
+-------------------+----------------+-----------------------------+
|2018-08-24 00:00:00|PME / PMI       |4913383577                   |
+-------------------+----------------+-----------------------------+
only showing top 1 row
```

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


display( consE )
```



Aggregated (by sum) in the backend.

```
consE.printSchema()
```

```
root
 |-- Jour: timestamp (nullable = true)
 |-- Catégorie client: string (nullable = true)
 |-- Puissance moyenne journalière: long (nullable = true)
```

DataFrames: Manipulating the DataFrames

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


f = path + "cons_elec.csv"
csv_file = spark.read.csv(f,sep=";",schema=newSchema, header=True)

csv_file.printSchema()
```

```
root
 |-- date: timestamp (nullable = true)
 |-- categClient: string (nullable = true)
 |-- consommation: double (nullable = true)
```

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

```
+-------------------+--------------+---------------+-----+
|date               |categClient   |consommation   |annee|
+-------------------+--------------+---------------+-----+
|2015-11-13 00:00:00|Entreprises   |1.3882225479E10|2015 |
|2015-11-11 00:00:00|Professionnels|3.74629482E9   |2015 |
|2015-12-18 00:00:00|Professionnels|5.523363437E9  |2015 |
|2015-12-06 00:00:00|Résidentiels  |2.254597387E10 |2015 |
|2015-10-30 00:00:00|Résidentiels  |1.5903891494E10|2015 |
|2015-10-20 00:00:00|Entreprises   |1.486203975E10 |2015 |
|2015-11-01 00:00:00|Professionnels|3.813420412E9  |2015 |
|2015-10-23 00:00:00|PME / PMI     |4.84148178E9   |2015 |
|2015-09-29 00:00:00|Professionnels|4.323409934E9  |2015 |
|2014-12-24 00:00:00|Professionnels|6.249969528E9  |2014 |
+-------------------+--------------+---------------+-----+
only showing top 10 rows
```

```
from pysaprk.sql import functions as fun
from pyspark.sql.types import IntegerType

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

#date2year = udf( date2year, IntegerType() )
csv_file.withColumn( "annee", date2year("date") ).show(10,False)
```

```
+-------------------+--------------+---------------+-----+
|date               |categClient   |consommation   |annee|
+-------------------+--------------+---------------+-----+
|2015-11-13 00:00:00|Entreprises   |1.3882225479E10|2015 |
|2015-11-11 00:00:00|Professionnels|3.74629482E9   |2015 |
|2015-12-18 00:00:00|Professionnels|5.523363437E9  |2015 |
|2015-12-06 00:00:00|Résidentiels  |2.254597387E10 |2015 |
|2015-10-30 00:00:00|Résidentiels  |1.5903891494E10|2015 |
|2015-10-20 00:00:00|Entreprises   |1.486203975E10 |2015 |
|2015-11-01 00:00:00|Professionnels|3.813420412E9  |2015 |
|2015-10-23 00:00:00|PME / PMI     |4.84148178E9   |2015 |
|2015-09-29 00:00:00|Professionnels|4.323409934E9  |2015 |
|2014-12-24 00:00:00|Professionnels|6.249969528E9  |2014 |
```

```
+-------------------+--------------+---------------+-----+
only showing top 10 rows
```

```
from pyspark.sql import functions as fun
f = path + "cons_elec.csv"
csv_file.withColumn( "annee", fun.year("date")).show(10, False)
```

```
+-------------------+--------------+---------------+-----+
|date               |categClient   |consommation   |annee|
+-------------------+--------------+---------------+-----+
|2015-11-13 00:00:00|Entreprises   |1.3882225479E10|2015 |
|2015-11-11 00:00:00|Professionnels|3.74629482E9   |2015 |
|2015-12-18 00:00:00|Professionnels|5.523363437E9  |2015 |
|2015-12-06 00:00:00|Résidentiels  |2.254597387E10 |2015 |
|2015-10-30 00:00:00|Résidentiels  |1.5903891494E10|2015 |
|2015-10-20 00:00:00|Entreprises   |1.486203975E10 |2015 |
|2015-11-01 00:00:00|Professionnels|3.813420412E9  |2015 |
|2015-10-23 00:00:00|PME / PMI     |4.84148178E9   |2015 |
|2015-09-29 00:00:00|Professionnels|4.323409934E9  |2015 |
|2014-12-24 00:00:00|Professionnels|6.249969528E9  |2014 |
+-------------------+--------------+---------------+-----+
only showing top 10 rows
```

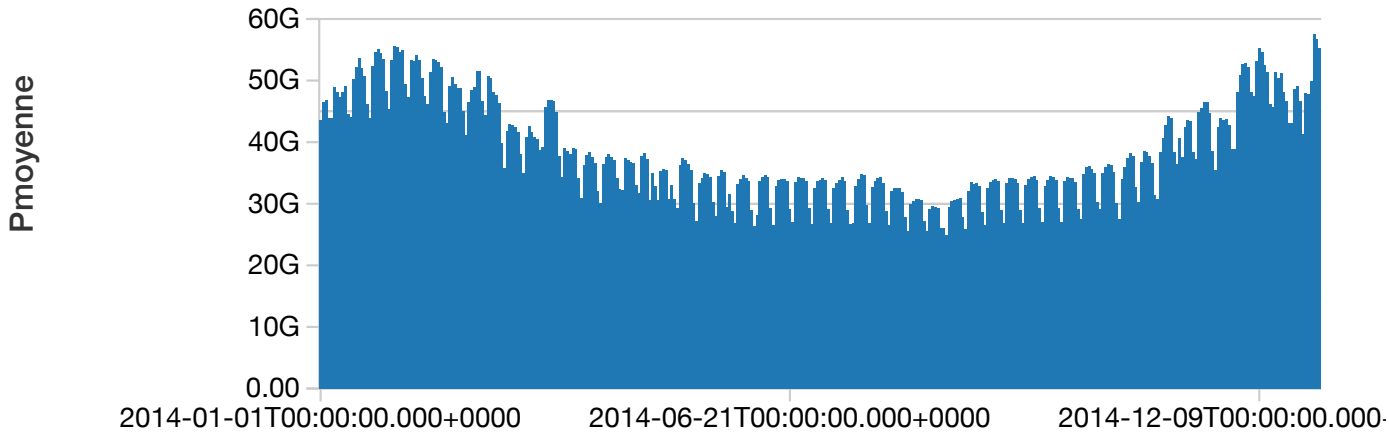
Exercise : Energy consumption in France

```
path = "/FileStore/tables/"
f = path + "cons_elec.csv"

########################## Task 1 : Schema change
cef = spark.read.csv(f, header=True, sep = ";")
#print cef.show(1,False)
print ( "<>Task 1 : File schema : \n", cef.printSchema() )

from pyspark.sql.types import StringType,TimestampType, FloatType, StructField,
StructType

newFields = [ StructField("Date",TimestampType()),
              StructField("CategClient",StringType()),
              StructField("Pmoyenne",FloatType())]

newSchema = StructType( newFields )

cef = spark.read.csv( f,sep=";", schema=newSchema, header=True )
print ( "<>Task 1 : The new schema : \n", cef.printSchema() )
#print cef.show(1,False)

########################## Task 2 : Unique measurement years
from datetime import datetime
from pyspark.sql.functions import udf
from pyspark.sql.types import  IntegerType

def ExtractYear(x) :
    return x.year
ExtractYear = udf(ExtractYear, IntegerType())
print ( "<>Task 2 : unique years in descending order : \n" )
print ( cef.select( "Date" ).withColumn( "Year", ExtractYear(cef.Date) )\
           .select( "Year" ).distinct()\
           .orderBy( "Year",ascending=False ).show() )

print ( cef.select("CategClient").distinct().show() )
```

```
root
 |-- Jour: string (nullable = true)
 |-- Catégorie client: string (nullable = true)
 |-- Puissance moyenne journalière: string (nullable = true)

<>Task 1 : File schema :
 None
root
 |-- Date: timestamp (nullable = true)
```

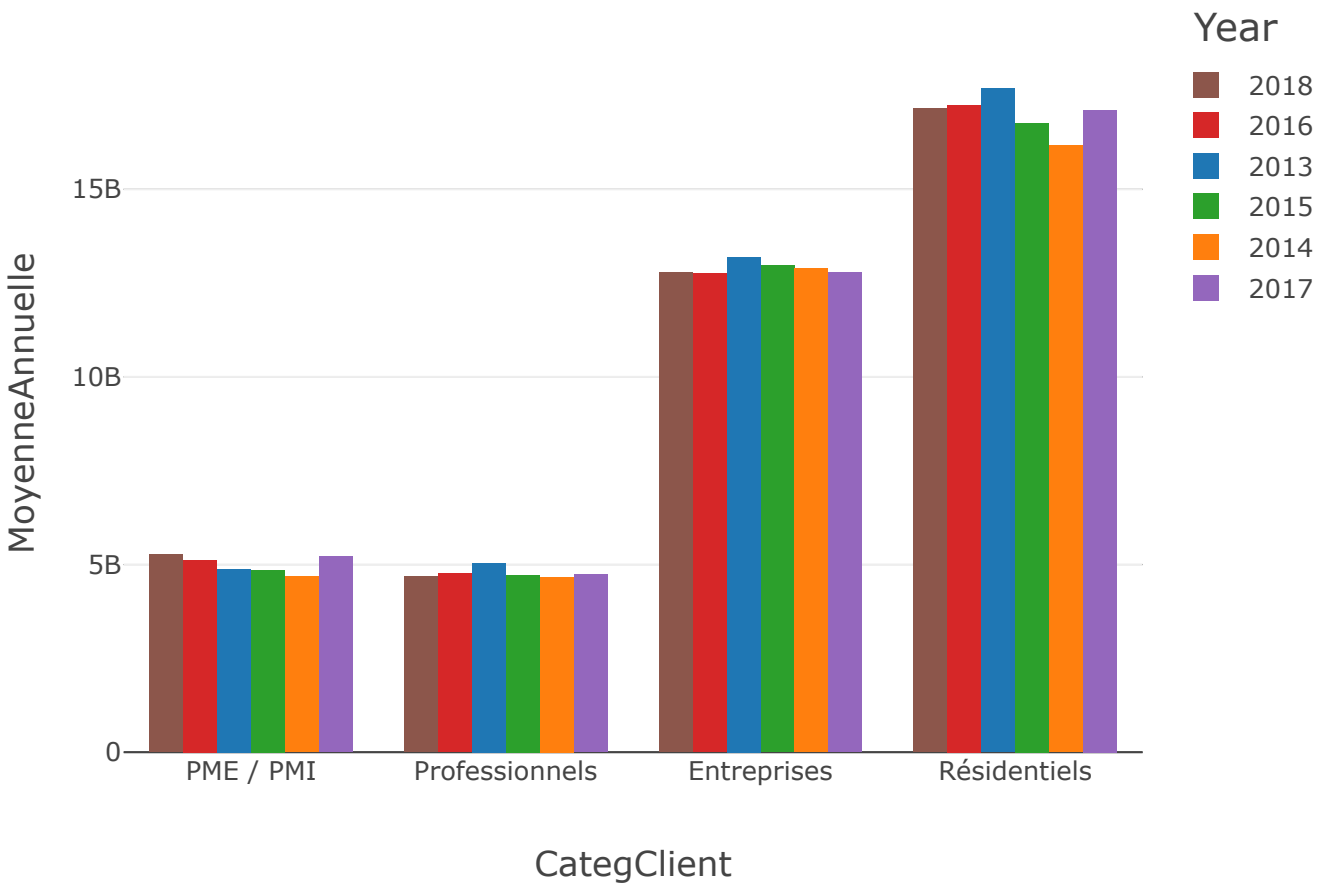
```
 |-- CategClient: string (nullable = true)
 |-- Pmoyenne: float (nullable = true)

<>Task 1 : The new schema :
 None
<>Task 2 : unique years in descending order :

+----+
|Year|
+----+
|2018|
|2017|
```

```
########################## Task 3 : comb shape
cefY = cef.withColumn("Year",ExtractYear(cef.Date))
c14 = cefY.filter( "Year==2014" )
display( c14 )
```



Aggregated (by sum) in the backend.

```
########################## Task 4 : Average consumption per year per category
#cefY.groupBy("CategClient","Year").agg({"Pmoyenne":"avg"}).withColumnRenamed("avg(
Pmoyenne)","MoyenneAnnuelle").show(24)
# Or :
from pyspark.sql import functions as fun
mcy = cefY.groupBy("CategClient", "Year").agg(
fun.avg("Pmoyenne").alias("MoyenneAnnuelle") )
display( mcy )
```



Exercise : White house visite

```
from pyspark.sql import Row
from pyspark.sql.functions import year
from datetime import datetime

nullTime = datetime(1999,12,31,0,0)
def convertDate(x, dateFormat="%m/%d/%Y %H:%M"):
    y = 0
    if(x!='') :
        y = datetime.strptime(x,dateFormat)
    else :
        y = nullTime
    return y

f = path + "visits.txt"
whRdd = sc.textFile(f).map(lambda line: line.split("\t"))

whRow = whRdd.map(lambda elem: Row(lsatName=elem[0], firstName=elem[1],
    arrivalTime=convertDate(elem[2]), appointmentTime=convertDate(elem[3]),
```

```
    meetingLocation=elem[4], event=elem[5]) )

whDf = whRow.toDF()
print( "<> Task 1 : Schema for whDf : \n", whDf.printSchema() )
print( "<> Task 1 : The first 3 lines : \n", whDf.show(10,False) )

print( "<> Task 2 : #Participants for domestic violence event :",\
whDf.filter(whDf.event=='DOMESTIC VIOLENCT AWARENESS MONTH LARGE MEETING.
').count() )

print( "<> Task 3 : #Participants for superman event :",\
whDf.filter(whDf.event=='WAITING FOR SUPERMAN DROP BY VISIT').count() )

t1 = datetime(2010,1,1,0,0)
t2 = datetime(2011,1,1,0,0)

N10 = whDf.filter( (whDf.appointmentTime>t1)&(whDf.appointmentTime<t2)&
(whDf.event.like("%MEDAL OF HONOR CEREMONY%"))
).select("appointmentTime").distinct().count()
print( "<> Task 4 : # medal events in 2010 : ", N10 )
N11 = whDf.filter( (whDf.appointmentTime>t2) & (whDf.event=="MEDAL OF HONOR
CEREMONY/" ) ).\
          select("appointmentTime").\
          distinct().count()

print( "<> Task 4 : # medal events in 2011 : ", N11 )

Timing = whDf.filter(whDf.arrivalTime!=nullTime)
NoArrivalRegistered = whDf.filter(whDf.arrivalTime==nullTime)

Delayed = Timing.filter(Timing.appointmentTime<Timing.arrivalTime)
Ontime = Timing.filter(Timing.appointmentTime>Timing.arrivalTime)

print( "<> Task 5 : # Delayed persons : ", Delayed.count(), ", # On time persons
:", Ontime.count() )

from pyspark.sql.functions import lit
df1 = Delayed.withColumn("timing",lit("Delayed"))
df2 = Ontime.withColumn("timing",lit("On time"))
df3 = NoArrivalRegistered.withColumn("timing",lit("Arrival time not registered"))

df = df1.union(df2).union(df3).orderBy("appointmentTime")
print( "<> Task 6 : Schema with new column : \n", df.printSchema() )
print( df.show(3,False) )
print ( whDf.count() )

df2010 =
whDf.filter(year("appointmentTime")==2010).select("appointmentTime","event")
```

```
awardDates = df2010.filter(df2010.event.like("%MEDAL OF HONOR
CEREMONY%")).select("appointmentTime")

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

#@ udf("string")
def convertMonth(x) :
    return x.strftime('%B')
print ( "<> Task 7 : Medal award months in 2010 :" )
convertMonth = udf(convertMonth,StringType())
awardDates.withColumn("month",convertMonth("appointmentTime")).select("month").dist
inct().show()
```

```
Debug whRdd:  [['BUCKLEY', 'SUMMER', '10/12/2010 14:48', '10/12/2010 14:45', 'WH',
''], ['CLOONEY', 'GEORGE', '10/12/2010 14:47', '10/12/2010 14:45', 'WH', ''], ['PR
ENDERGAST', 'JOHN', '10/12/2010 14:48', '10/12/2010 14:45', 'WH', ''], ['LANIER',
'JAZMIN', '', '10/13/2010 13:00', 'WH', 'BILL SIGNING/'], ['MAYNARD', 'ELIZABETH',
'10/13/2010 12:34', '10/13/2010 13:00', 'WH', 'BILL SIGNING/'], ['MAYNARD', 'GREGO
RY', '10/13/2010 12:35', '10/13/2010 13:00', 'WH', 'BILL SIGNING/'], ['MAYNARD', '
JOANNE', '10/13/2010 12:35', '10/13/2010 13:00', 'WH', 'BILL SIGNING/'], ['MAYNARD
', 'KATHERINE', '10/13/2010 12:34', '10/13/2010 13:00', 'WH', 'BILL SIGNING/'], ['
MAYNARD', 'PHILIP', '10/13/2010 12:35', '10/13/2010 13:00', 'WH', 'BILL SIGNING/']
, ['MOHAN', 'EDWARD', '10/13/2010 12:37', '10/13/2010 13:00', 'WH', 'BILL SIGNING/
'], ['MOHAN', 'KATHLEEN', '10/13/2010 12:38', '10/13/2010 13:00', 'WH', 'BILL SIGN
ING/'], ['MOHAN', 'SARAH', '10/13/2010 12:37', '10/13/2010 13:00', 'WH', 'BILL SIG
NING/'], ['MOUZON', 'LARINA', '', '10/13/2010 13:00', 'WH', 'BILL SIGNING/']]
root
 |-- appointmentTime: timestamp (nullable = true)
 |-- arrivalTime: timestamp (nullable = true)
 |-- event: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- lsatName: string (nullable = true)
 |-- meetingLocation: string (nullable = true)
```

Exercise : Elections in Paris

```
######################### Task 1 : Making the schema
from pyspark.sql.types import TimestampType, StringType, IntegerType, StructType,
StructField

fields = [ StructField("Label", StringType()), StructField("DateS", StringType()),
StructField("Commune", StringType()),\
    StructField("nEligibles", IntegerType()), StructField("nElecteurs",
IntegerType()), StructField("Nom", StringType()), \
    StructField("Prenom", StringType()),StructField("nVotes", IntegerType()) ]
leSchema = StructType(fields)

path = "/FileStore/tables/"
f = path + "resultats_electoraux.csv"
reselec = spark.read.csv(f,sep=";", schema=leSchema)
print ( "<>Task 1 : Schema without Timestamp", reselec.printSchema() )

root
 |-- Label: string (nullable = true)
 |-- DateS: string (nullable = true)
 |-- Commune: string (nullable = true)
 |-- nEligibles: integer (nullable = true)
 |-- nElecteurs: integer (nullable = true)
 |-- Nom: string (nullable = true)
 |-- Prenom: string (nullable = true)
 |-- nVotes: integer (nullable = true)

<>Task 1 : Schema without Timestamp None

######################### Task 1 : Making the schema (continue)
from pyspark.sql import functions as fun
from datetime import datetime

def convertDate(x) :
    if "/" in x :
        return datetime.strptime(x, "%d/%m/%Y")
    return datetime.strptime(x,"%Y-%m-%d %H:%M:%S")
convertDate = udf(convertDate,TimestampType())
#df = reselec.withColumn("Date", convertDate("DateS")).drop("DateS")

df = reselec.withColumn("Date", fun.to_timestamp("DateS", "d/M/y")).drop("DateS")

print ( "<>Task 1 : Schema with Timestamp", df.show(5,False) )


+----------------------------+--------+----------+----------+---------------+-----
----+------+-----------------+
|Label                       |Commune |nEligibles|nElecteurs|Nom            |Preno
```

```
m   |nVotes|Date                 |
+----------------------------+--------+----------+----------+---------------+-----
----+------+-------------------+
|L??gislatives 2012 - 1er Tour|PARIS 04|1353      |831       |MAJDA          |Jacky
|27    |2012-10-06 00:00:00|
|L??gislatives 2012 - 1er Tour|PARIS 04|1353      |831       |GUILLEMINOT    |Elian
e   |1     |2012-10-06 00:00:00|
|L??gislatives 2012 - 1er Tour|PARIS 04|1353      |831       |LAENG CINDY'LEE|Isabe
lle |2     |2012-10-06 00:00:00|
|L??gislatives 2012 - 1er Tour|PARIS 04|1041      |629       |GUILLEMINOT    |Elian
e   |3     |2012-10-06 00:00:00|
|L??gislatives 2012 - 1er Tour|PARIS 04|962       |558       |REY            |St??p
hane|0     |2012-10-06 00:00:00|
+----------------------------+--------+----------+----------+---------------+-----
----+------+-------------------+
only showing top 5 rows

<>Task 1 : Schema with Timestamp None

########################## Task 2 : Regular Expression
from pyspark.sql import functions as fun

import re
@fun.udf( "string" )
def replaceChar(x) :
    return re.sub( r'\?\?', 'e', x)
#df = df.withColumn("Label", replaceChar("Label")).withColumn("Prenom",
replaceChar("Prenom")).withColumn("Nom", replaceChar("Nom"))

for strg in [ "Label", "Prenom", "Nom" ] :
  df = df.withColumn( strg, fun.regexp_replace(strg, r'\?\?', 'e'))

df.show(1,False)


+----------------------------+--------+----------+----------+-----+------+------+--
-----------------+
|Label                       |Commune |nEligibles|nElecteurs|Nom  |Prenom|nVotes|Da
te               |
+----------------------------+--------+----------+----------+-----+------+------+--
-----------------+
|Legislatives 2012 - 1er Tour|PARIS 04|1353      |831       |MAJDA|Jacky |27    |20
12-10-06 00:00:00|
+----------------------------+--------+----------+----------+-----+------+------+--
-----------------+
only showing top 1 row
```

```
######################### Task 3 : Unique labels, adding scrutinType and Tour
columns
print ( "<>Task 3 : Unique labels : \n",
df.select("Label").distinct().show(22,False) )

from pyspark.sql.types import StringType

@fun.udf("string")
def parseLabel(x) :
   if "Europeennes" in x :
       return "-"
   if "1er" in x :
       return "1"
   return "2"

#parseLabel = udf(parseLabel, StringType())
df3 = df.withColumn("scrutinType", fun.split("Label"," ")[0])\
        .withColumn("Tour", parseLabel("Label"))\
        .drop("Label")

print ( "<>Task 3 : New columns : \n", df3.printSchema() )
print ( "<>Task 3 : Unique labels : \n",
df3.select("scrutinType").distinct().show(22,False) )
```

```
+------------------------------+
|Label                         |
+------------------------------+
|Presidentielle 2007 - 1er Tour |
|Legislatives 2017 - 2eme tour  |
|Legislatives 2012 - 1er Tour   |
|Europeennes 2009               |
|Legislatives 2007 - 1er tour   |
|Legislatives 2007 - 2eme tour  |
|Regionales 2015 -  2eme tour   |
|Presidentielle 2017 - 1er tour |
|Legislatives 2017 - 1er tour   |
|Regionales 2eme tour           |
|Presidentielle 2012 - 1er Tour |
|Municipales 2008 - 1er tour    |
|Europeennes 2014               |
|Regionales 2015 - 1er tour     |
|Municipales 2014 - 2eme tour   |
|Presidentielle 2007 - 2eme Tour|
|Municipales 2014 - 1er tour    |
|Legislatives 2012 - 2eme Tour  |
```

```
######################### Task 4 : Adding Annee column
from pyspark.sql.types import IntegerType

def extractYear(x) :
    return x.year
extractYear = udf(extractYear, IntegerType())
#df4 = df3.withColumn("Annee", extractYear("Date"))
df4 = df3.withColumn( "Annee", fun.year("Date") )

#df4.show(300,False)
print ( "<>Task 4 : Adding Annee column: ",
       df4.select("scrutinType","Annee").distinct()\
          .orderBy("Annee", "scrutinType", ascending=False).show() )

from pyspark.sql.functions import collect_list
print ("<>Task 4 : Elections in different years : ",
df4.select("scrutinType","Annee").distinct().groupBy("Annee").agg(collect_list("scr
utinType").alias("Elections")).\
    orderBy("Annee").show(8,False) )
```

```
+--------------+-----+
|   scrutinType|Annee|
+--------------+-----+
|Presidentielle| 2017|
|  Legislatives| 2017|
|   Municipales| 2014|
|Presidentielle| 2012|
|  Legislatives| 2012|
|    Regionales| 2010|
|   Europeennes| 2009|
|   Municipales| 2008|
|Presidentielle| 2007|
|  Legislatives| 2007|
|    Regionales| null|
|   Europeennes| null|
+--------------+-----+

<>Task 4 : Adding Annee column:  None
+-----+------------------------------+
|Annee|Elections                     |
+-----+------------------------------+
```

```
######################### Task 5 : Vote offices consistency check

df5 = df4.filter("Annee=2017 and scrutinType='Presidentielle' and
Tour='1'").distinct()\
         .select("Nom", "nEligibles", "nElecteurs")\
         .groupBy("Nom")\
         .agg( fun.sum("nEligibles").alias("totEligibles"),
               fun.sum("nElecteurs").alias("totElecteurs") )\
         .orderBy("Nom")
print ( "<>Task 5 : Global vote consistency : \n", df5.show() )

df5 = df4.filter("Annee=2017 and scrutinType='Presidentielle' and
Tour='1'").distinct()\
         .select("Nom", "Commune", "nEligibles", "nElecteurs")\
         .groupBy("Nom", "Commune")\
         .agg({"nEligibles" : "sum", "nElecteurs" : "sum"})\
         .withColumnRenamed("sum(nElecteurs)","totElecteurs")\
         .withColumnRenamed("sum(nEligibles)","totEligibles")\
         .orderBy("Commune")
print ( "<>Task 5 : Vote consistency by district : \n", df5.show(300) )
```

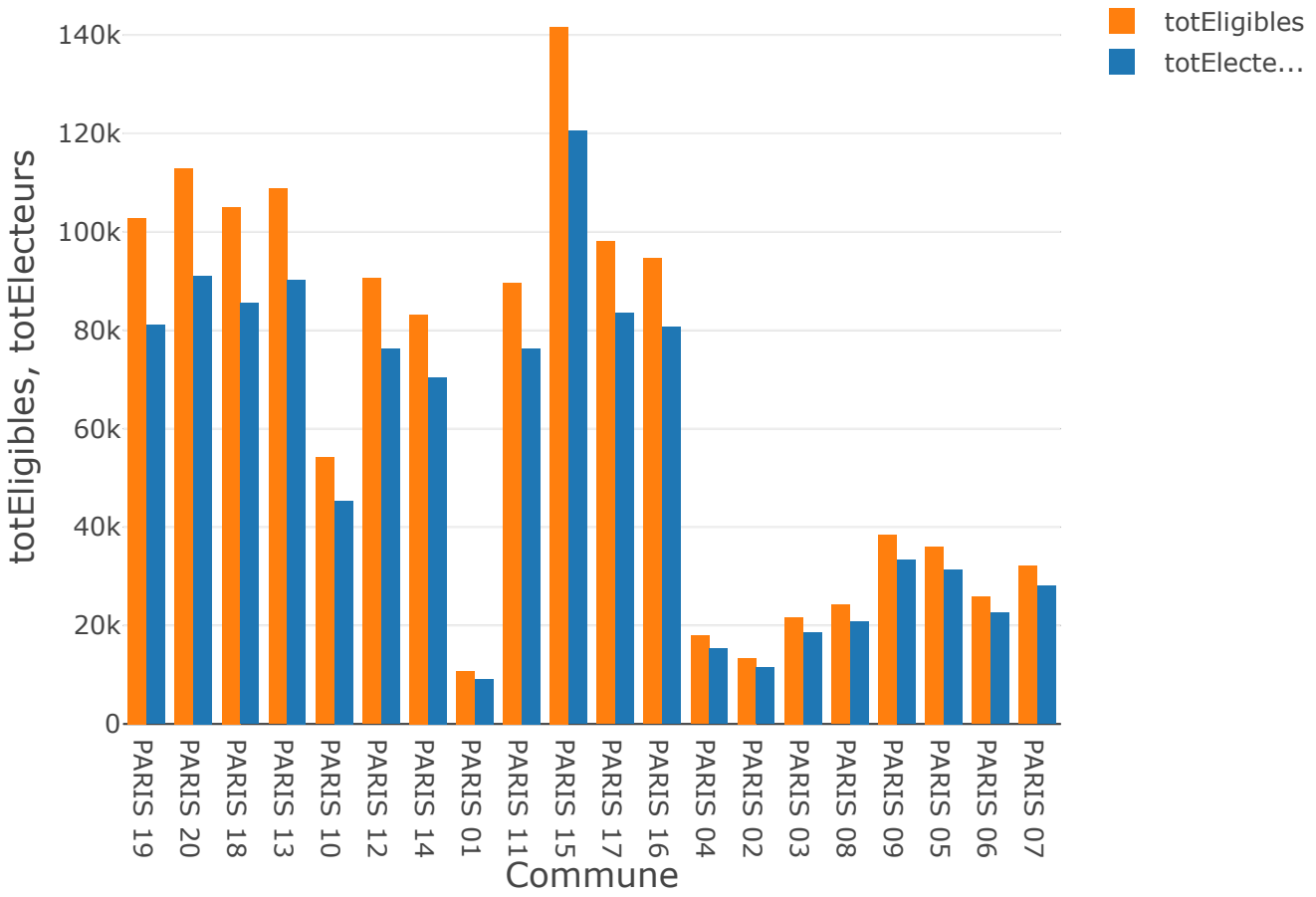
```
+-------------+------------+------------+
|          Nom|totEligibles|totElecteurs|
+-------------+------------+------------+
|      ARTHAUD|     1301637|     1091437|
|   ASSELINEAU|     1301637|     1091437|
|    CHEMINADE|     1301637|     1091437|
|DUPONT-AIGNAN|     1301637|     1091437|
|       FILLON|     1301637|     1091437|
|        HAMON|     1301637|     1091437|
|     LASSALLE|     1301637|     1091437|
|       LE PEN|     1301637|     1091437|
|       MACRON|     1301637|     1091437|
|    MeLENCHON|     1301637|     1091437|
|       POUTOU|     1301637|     1091437|
+-------------+------------+------------+

<>Task 5 : Global vote consistency :
 None
+-------------+--------+------------+------------+
|          Nom| Commune|totElecteurs|totEligibles|
+-------------+--------+------------+------------+
```

```
######################### Task 6 : Plotting the articipation rate

df6 = df5.select("Commune", "totElecteurs",
"totEligibles").distinct().withColumn("Rate",df5.totElecteurs/df5.totEligibles).ord
erBy("Rate")
print ( "<>Task 6 : Participation rate for presidential first round in Paris\n",
df6.show() )
```

```
+--------+------------+------------+------------------+
| Commune|totElecteurs|totEligibles|              Rate|
+--------+------------+------------+------------------+
|PARIS 19|       81047|      102806|0.7883489290508336|
|PARIS 20|       91107|      112905|0.8069350338780391|
|PARIS 18|       85496|      104995|0.8142863945902186|
|PARIS 13|       90221|      108815| 0.829122823140192|
|PARIS 10|       45388|       54176|0.8377879503839338|
|PARIS 12|       76175|       90543|0.8413129673193952|
|PARIS 14|       70443|       83195|0.8467215577859246|
|PARIS 01|        9129|       10775|0.8472389791183295|
|PARIS 11|       76277|       89660|0.8507361142092349|
|PARIS 15|      120530|      141629|0.8510262728678448|
|PARIS 17|       83454|       98018|0.8514150462160012|
|PARIS 16|       80621|       94558|0.8526089807314029|
|PARIS 04|       15294|       17916|0.8536503683858004|
|PARIS 02|       11423|       13360|0.8550149700598803|
|PARIS 03|       18680|       21677|0.8617428610970153|
|PARIS 08|       20872|       24158|0.8639788061925656|
|PARIS 09|       33290|       38462|0.8655296136446363|
|PARIS 05|       31376|       36060|0.8701053799223516|
|PARIS 06|       22548|       25842|0.8725330856744834|
|PARIS 07|       28066|       32087|0.8746844516470845|
+--------+------------+------------+------------------+
```

```
<>Task 6 : Participation rate for presidential first round in Paris
 None
```

```
display(  df6 )
```



Aggregation

```
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)

df_agg.show()
df_agg.printSchema()
```

```
+--------+--------+--------+
|distance|vel_car1|vel_car2|
+--------+--------+--------+
|     2.3|    70.0|    80.3|
|     4.8|    91.5|    85.1|
|     3.7|    88.4|    94.5|
|     4.5|    86.6|    81.7|
```

```
+--------+--------+--------+

root
 |-- distance: double (nullable = true)
 |-- vel_car1: double (nullable = true)
 |-- vel_car2: double (nullable = true)


#from pyspark.sql.functions import sum, mean
from pyspark.sql import functions as fun

df_agg2 = df_agg.agg( fun.sum( "distance" ), fun.mean( "vel_car1"
).alias("meanV_1"),  fun.mean( "vel_car2" ).alias("meanV_2") )
df_agg2.show()


+-------------+-------+-----------------+
|sum(distance)|meanV_1|          meanV_2|
+-------------+-------+-----------------+
|         15.3| 84.125|85.39999999999999|
+-------------+-------+-----------------+


df_agg2 = df_agg.agg( sum( "distance" ),
                      fun.round( mean( "vel_car1"), 1 ).alias("meanV_1"),
                      fun.round ( mean( "vel_car2" ), 1).alias("meanV_2")
                     )
df_agg2.show()


+-------------+-------+-------+
|sum(distance)|meanV_1|meanV_2|
+-------------+-------+-------+
|         15.3|   84.1|   85.4|
+-------------+-------+-------+


df_agg2 = df_agg2.withColumnRenamed( "sum(distance)", "totDistance" ).\
                 withColumnRenamed( "round(avg(vel_car1), 1)", "vel_avg1" ).\
                 withColumnRenamed( "round(avg(vel_car2), 1)", "vel_avg2" )
df_agg2.show()


+-----------+--------+--------+
|totDistance|vel_avg1|vel_avg2|
+-----------+--------+--------+
|       15.3|    84.1|    85.4|
```

```
+-----------+--------+--------+
```

```
path = "/FileStore/tables/"
fcovid = path + "covid19.csv"
covid19 = spark.read.csv( fcovid, sep=",", inferSchema=True, header=True )
covid19.printSchema()
```

```
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)
```

```
covid19.filter( fun.col("Country/Region")=='France' ).show(5, False)
```

```
+----------------+--------------+--------+--------+-------+---------+------+-------
--+
|Province/State  |Country/Region|Lat     |Long    |Date   |Confirmed|Deaths|Recover
ed|
+----------------+--------------+--------+--------+-------+---------+------+-------
--+
|French Guiana   |France        |3.9339  |-53.1258|1/22/20|0        |0     |0
|
|French Polynesia|France        |-17.6797|149.4068|1/22/20|0        |0     |0
|
|Guadeloupe      |France        |16.25   |-61.5833|1/22/20|0        |0     |0
|
|Mayotte         |France        |-12.8275|45.1662 |1/22/20|0        |0     |0
|
|New Caledonia   |France        |-20.9043|165.618 |1/22/20|0        |0     |0
|
+----------------+--------------+--------+--------+-------+---------+------+-------
--+
only showing top 5 rows
```

```
from pyspark.sql.functions import to_timestamp, col
covid = covid19.select( col("Country/Region").alias("country"),
                    to_timestamp( col("Date"), "MM/dd/yy" ).alias("date"),
                    col("Confirmed").alias("confirmed"),
                    col("Deaths").alias("deaths"),
                    col("Recovered").alias("recovered")
                    )
covid.show(5)
covid.printSchema()
```

```
+-----------+-------------------+---------+------+---------+
|    country|               date|confirmed|deaths|recovered|
+-----------+-------------------+---------+------+---------+
|Afghanistan|2020-01-22 00:00:00|        0|     0|        0|
|    Albania|2020-01-22 00:00:00|        0|     0|        0|
|    Algeria|2020-01-22 00:00:00|        0|     0|        0|
|    Andorra|2020-01-22 00:00:00|        0|     0|        0|
|     Angola|2020-01-22 00:00:00|        0|     0|        0|
+-----------+-------------------+---------+------+---------+
only showing top 5 rows

root
 |-- country: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- confirmed: integer (nullable = true)
 |-- deaths: integer (nullable = true)
 |-- recovered: integer (nullable = true)
```

Groupby

```
covid2 = covid.groupby( "country", "date" ).\
        agg( fun.sum("confirmed").alias("confirmed"),
            fun.sum("deaths").alias("deaths"),
            fun.sum("recovered").alias("recovered")
            ).\
        orderBy( "country", "date" )

covid2.show(5)
#covid2.printSchema()
```

```
+-----------+-------------------+---------+------+---------+
|    country|               date|confirmed|deaths|recovered|
+-----------+-------------------+---------+------+---------+
|Afghanistan|2020-01-22 00:00:00|        0|     0|        0|
```

```
|Afghanistan|2020-01-23 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-24 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-25 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-26 00:00:00|        0|     0|        0|
+-----------+-------------------+---------+------+---------+
only showing top 5 rows
```

```
liste = [ fun.sum( c ).alias(c) for c in [ "confirmed", "deaths", "recovered"] ]

covid2_ = covid.groupby( "country", "date" )\
               .agg( *liste )\
               .orderBy( "country", "date" )
covid2_.show(5)
```

```
+-----------+-------------------+---------+------+---------+
|    country|               date|confirmed|deaths|recovered|
+-----------+-------------------+---------+------+---------+
|Afghanistan|2020-01-22 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-23 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-24 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-25 00:00:00|        0|     0|        0|
|Afghanistan|2020-01-26 00:00:00|        0|     0|        0|
+-----------+-------------------+---------+------+---------+
only showing top 5 rows
```

```
#print( covid2.select("country").distinct().count() )
covid2.rdd.getNumPartitions()

Out[35]: 200

path = "/FileStore/tables/"
fname = path + "covid.parquet"
#covid2.write.mode("overwrite").parquet( fname )
```

Cross join

```
from pyspark.sql import functions as fun
f = path + "covid.parquet"
covid1 = spark.read.parquet( f )
covid1_ = covid1.select( fun.col("country").alias("pay"),
                         fun.col("date").alias("d"),
                         fun.col("confirmed").alias("conf"),
                         fun.col("deaths").alias("morts"),
                         fun.col("recovered").alias("gueris")
                         )
covid1_.printSchema()
```

```
root
 |-- pay: string (nullable = true)
 |-- d: timestamp (nullable = true)
 |-- conf: long (nullable = true)
 |-- morts: long (nullable = true)
 |-- gueris: long (nullable = true)
```

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

df.show()
```

```
+-------+-------------------+---------+------+---------+-----+-------------------+-
----+-----+------+
|country|               date|confirmed|deaths|recovered|  pay|                  d|
conf|morts|gueris|
+-------+-------------------+---------+------+---------+-----+-------------------+-
----+-----+------+
|  Chile|2020-04-17 00:00:00|     9252|   116|     3621|Chile|2020-04-16 00:00:00|
8807|  105|  3299|
|  Chile|2020-04-18 00:00:00|     9730|   126|     4035|Chile|2020-04-17 00:00:00|
9252|  116|  3621|
|  Chile|2020-04-19 00:00:00|    10088|   133|     4338|Chile|2020-04-18 00:00:00|
9730|  126|  4035|
|  Chile|2020-04-20 00:00:00|    10507|   139|     4676|Chile|2020-04-19 00:00:00|1
0088|  133|  4338|
|  Chile|2020-04-21 00:00:00|    10832|   147|     4969|Chile|2020-04-20 00:00:00|1
0507|  139|  4676|
|  Chile|2020-04-22 00:00:00|    11296|   160|     5386|Chile|2020-04-21 00:00:00|1
0832|  147|  4969|
|  Chile|2020-04-23 00:00:00|    11812|   168|     5804|Chile|2020-04-22 00:00:00|1
1296|  160|  5386|
|  Chile|2020-04-24 00:00:00|    12306|   174|     6327|Chile|2020-04-23 00:00:00|1
1812|  168|  5804|
```

```
|  Chile|2020-04-25 00:00:00|    12858|   181|     6746|Chile|2020-04-24 00:00:00|1
2306|  174|  6327|
|  Chile|2020-04-26 00:00:00|    13331|   189|     7024|Chile|2020-04-25 00:00:00|1
2858|  181|  6746|
|  Chile|2020-04-27 00:00:00|    13813|   198|     7327|Chile|2020-04-26 00:00:00|1
3331|  189|  7024|
|  Chile|2020-04-28 00:00:00|    14365|   207|     7710|Chile|2020-04-27 00:00:00|1
3813|  198|  7327|
|  Chile|2020-04-29 00:00:00|    14885|   216|     8057|Chile|2020-04-28 00:00:00|1
4365|  207|  7710|
|  Chile|2020-04-30 00:00:00|    16023|   227|     8580|Chile|2020-04-29 00:00:00|1
4885|  216|  8057|
|  Chile|2020-05-01 00:00:00|    17008|   234|     9018|Chile|2020-04-30 00:00:00|1
6023|  227|  8580|
|  Chile|2020-05-02 00:00:00|    18435|   247|     9572|Chile|2020-05-01 00:00:00|1
7008|  234|  9018|
|  Chile|2020-05-03 00:00:00|    19663|   260|    10041|Chile|2020-05-02 00:00:00|1
8435|  247|  9572|
|  Chile|2020-05-04 00:00:00|    20643|   270|    10415|Chile|2020-05-03 00:00:00|1
9663|  260| 10041|
|  Chile|2020-05-05 00:00:00|    22016|   275|    10710|Chile|2020-05-04 00:00:00|2
0643|  270| 10415|
|  Chile|2020-05-06 00:00:00|    23048|   281|    11189|Chile|2020-05-05 00:00:00|2
2016|  275| 10710|
+-------+-------------------+---------+------+---------+-----+-------------------+-
----+-----+------+
only showing top 20 rows
```

```
covid = df.withColumn( "newConfirmed", df.confirmed-df.conf ).\
            withColumn("newDeaths", df.deaths-df.morts).\
            withColumn("newRecovered", df.recovered-df.gueris).\
            drop( "conf", "morts", "gueris", "pay", "d" )

covid.printSchema()

covid.filter( df.country=="France" ).show(100)
```

```
root
 |-- country: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- confirmed: long (nullable = true)
 |-- deaths: long (nullable = true)
 |-- recovered: long (nullable = true)
 |-- newConfirmed: long (nullable = true)
 |-- newDeaths: long (nullable = true)
 |-- newRecovered: long (nullable = true)
```

```
+-------+-------------------+---------+------+---------+------------+---------+---
---------+
|country|               date|confirmed|deaths|recovered|newConfirmed|newDeaths|new
Recovered|
+-------+-------------------+---------+------+---------+------------+---------+---
---------+
| France|2020-01-27 00:00:00|        3|     0|        0|           0|        0|
0|
| France|2020-01-28 00:00:00|        4|     0|        0|           1|        0|
0|
```

```
covid = covid.withColumn("dd", fun.col("date") ).drop("date").withColumnRenamed(
"dd", "date" )
```

Or simply by Windoing and using the lag function

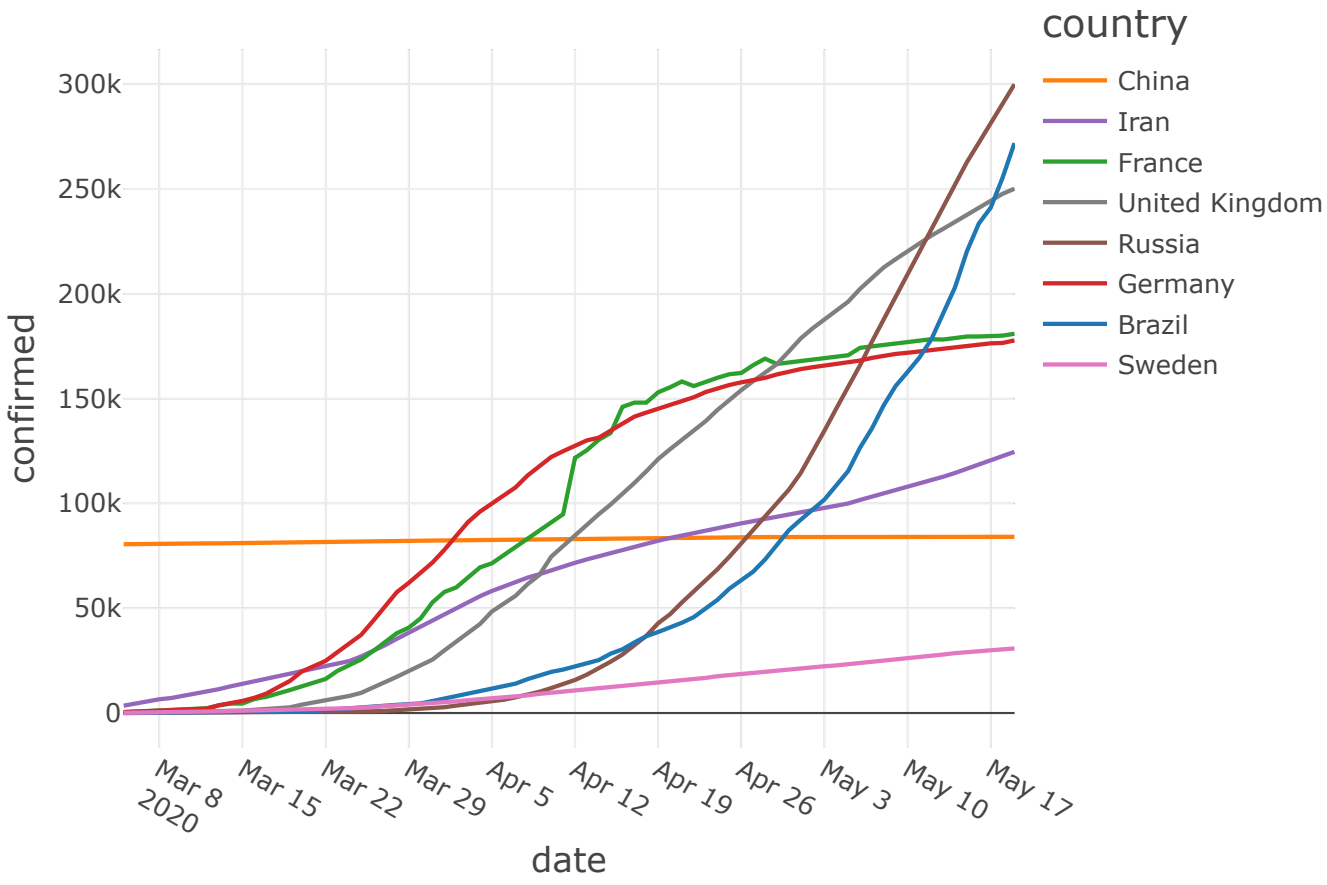
```
from pyspark.sql import Window
win = Window.partitionBy("country").orderBy("date")

covid_ = covid1.withColumn( "newConfirmed", covid1.confirmed - fun.lag(
covid1.confirmed, 1 ).over(win) )\
           .withColumn( "newDeaths", covid1.deaths - fun.lag( covid1.deaths, 1
).over(win) )\
           .withColumn( "newRecovered", covid1.recovered - fun.lag(
covid1.recovered, 1 ).over(win) )

covid_.filter( covid_.country == "France" ).show(100)
```

```
+-------+-------------------+---------+------+---------+------------+---------+---
---------+
|country|               date|confirmed|deaths|recovered|newConfirmed|newDeaths|new
Recovered|
+-------+-------------------+---------+------+---------+------------+---------+---
---------+
| France|2020-01-22 00:00:00|        0|     0|        0|        null|     null|
null|
| France|2020-01-23 00:00:00|        0|     0|        0|           0|        0|
0|
| France|2020-01-24 00:00:00|        2|     0|        0|           2|        0|
0|
| France|2020-01-25 00:00:00|        3|     0|        0|           1|        0|
0|
| France|2020-01-26 00:00:00|        3|     0|        0|           0|        0|
0|
| France|2020-01-27 00:00:00|        3|     0|        0|           0|        0|
0|
```

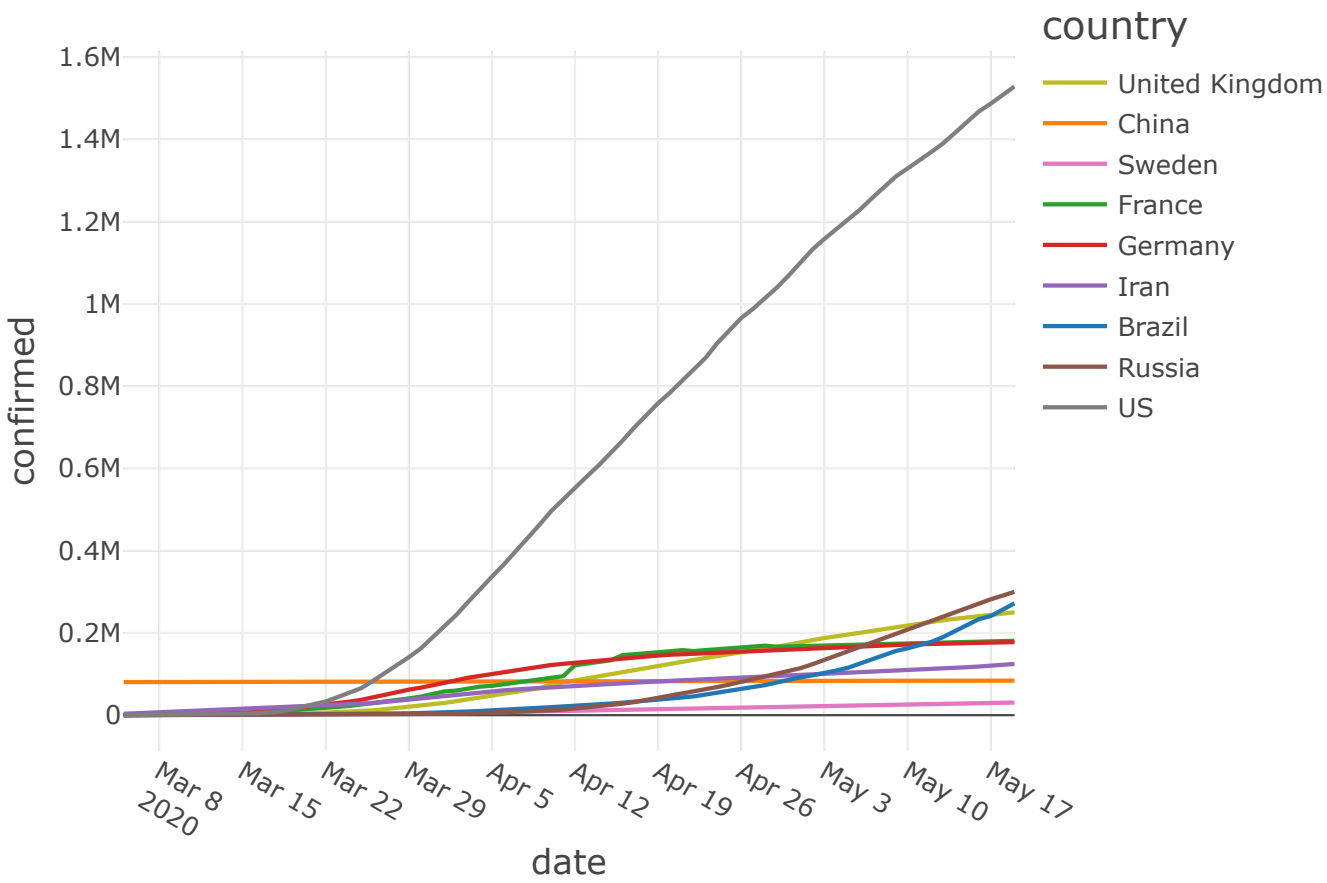
```
| France|2020-01-28 00:00:00|        4|     0|        0|           1|        0|
0|
| France|2020-01-29 00:00:00|        5|     0|        0|           1|        0|
```

```
from pyspark.sql.functions import lit
#cc1 =  ["France", "Germany","China", "Italy", "Spain", "United Kingdom", "Sweden",
"US", "Iran", "Russia", "Australia", "Brazil"]
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") ))
```

```
display( toShow )
```



```
cc =  ["France", "Germany","China", "United Kingdom", "Sweden", "Iran", "Russia",
"Brazil", "US"]
toShow_ = covid.filter( covid.country.isin( cc ) ).\
                filter( ( col("date") > lit("2020-03-05") ) & ( col("date") <
lit("2020-05-20") ))


display( toShow_ )
```



Inner join

```
f = path + "countryPop.csv"
cpop1 = spark.read.csv( f, sep=",", header=True, inferSchema=True )
cpop1.printSchema()
```

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

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

cpop.filter( col("country").like("U%") ).show()
```

```
+----------+--------------------+
|population|             country|
+----------+--------------------+
| 330610570|                  US|
|  67814098|      United Kingdom|
|  45427637|              Uganda|
|  43785122|             Ukraine|
|  33368859|          Uzbekistan|
|   9865845|United Arab Emirates|
|   3471314|             Uruguay|
|    104456| U.S. Virgin Islands|
+----------+--------------------+
```

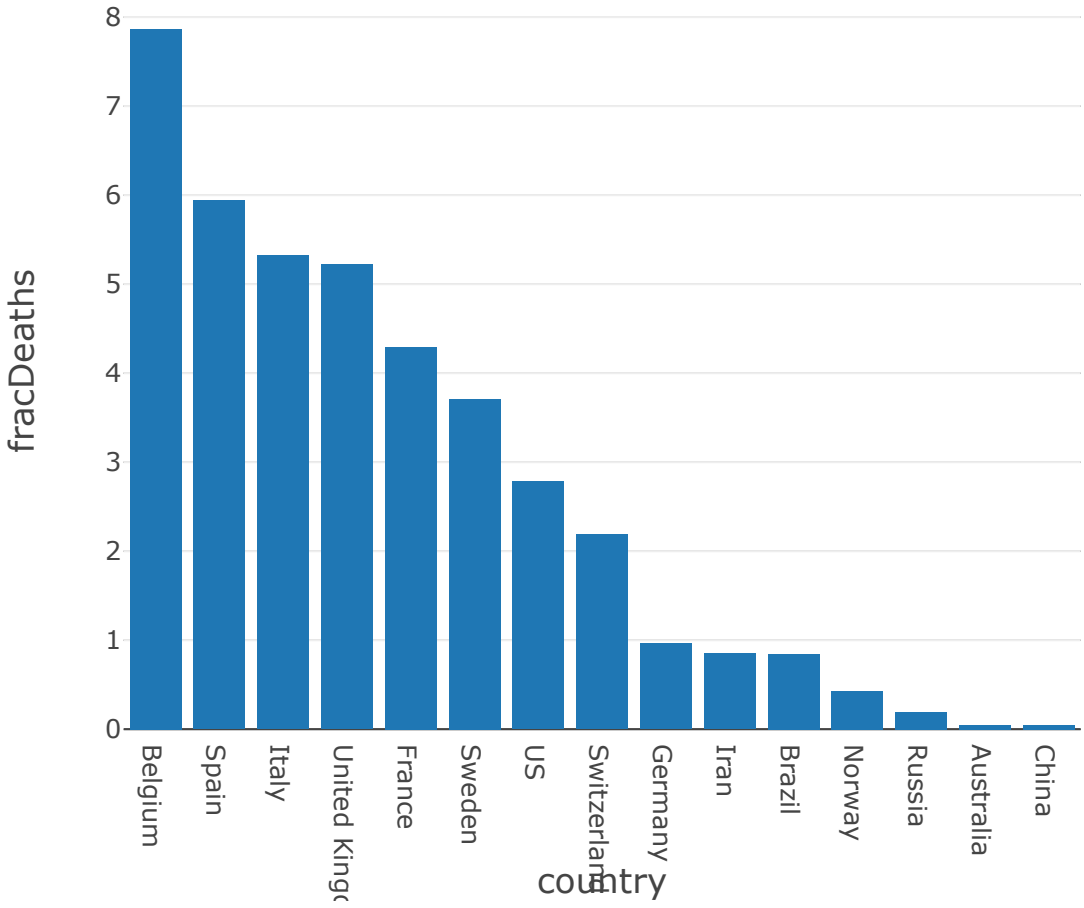
```
covidPop1 = covid.join( cpop, on="country" )
covidPop1.printSchema()
covidPop1.filter( col("country").like("U%") ).show(4, False)
```

```
root
 |-- country: string (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- confirmed: long (nullable = true)
 |-- deaths: long (nullable = true)
 |-- recovered: long (nullable = true)
 |-- newConfirmed: long (nullable = true)
 |-- newDeaths: long (nullable = true)
 |-- newRecovered: long (nullable = true)
 |-- population: integer (nullable = true)

+-------+-------------------+---------+------+---------+------------+---------+----
--------+----------+
|country|date               |confirmed|deaths|recovered|newConfirmed|newDeaths|newR
ecovered|population|
+-------+-------------------+---------+------+---------+------------+---------+----
--------+----------+
|US     |2020-01-23 00:00:00|1        |0     |0        |0           |0        |0
```

```
|330610570 |
|US     |2020-01-24 00:00:00|2        |0     |0        |1           |0        |0
|330610570 |
|US     |2020-01-25 00:00:00|2        |0     |0        |0           |0        |0
|330610570 |
|US     |2020-01-26 00:00:00|5        |0     |0        |3           |0        |0
|330610570 |
+-------+-------------------+---------+------+---------+------------+---------+----
--------+----------+
only showing top 4 rows
```

Exercise : Death rate computation

```
frac = 10000.
covidPop = covidPop1.withColumn( "fracConfrimed",
covidPop1.confirmed*frac/covidPop1.population ).\
                withColumn( "fracDeaths",
covidPop1.deaths*frac/covidPop1.population ).\
                withColumn( "fracRecovered",
covidPop1.recovered*frac/covidPop1.population )


from pyspark.sql import functions as fun
tmax = covidPop.select( fun.max("date").alias("maxt") ).collect()[0].maxt
tmax

Out[48]: datetime.datetime(2020, 5, 19, 0, 0)

from pyspark.sql.functions import lit
cc =  ["France", "Germany","China", "Belgium", "Italy", "Spain", "United Kingdom",
"Sweden",
            "US", "Switzerland", "Norway", "Iran", "Russia", "Australia", "Brazil"]
toShow2 =  covidPop.filter( ( covidPop.date == lit( tmax ) ) &
(covid.country.isin(cc ) ) ).\
                  orderBy("fracDeaths", ascending=False)


display( toShow2 )
```



Day-today confirmed ratio

```
covid_ = covid.select( col("country").alias("pay"), col("date").alias("d"),
col("newConfirmed").alias("ncy") )
covid_.printSchema()
```

```
root
 |-- pay: string (nullable = true)
 |-- d: timestamp (nullable = true)
 |-- ncy: long (nullable = true)
```

```
covid.filter("country = 'France' ").show(100)
```

```
+-------+-------------------+---------+------+---------+------------+---------+---
---------+
|country|               date|confirmed|deaths|recovered|newConfirmed|newDeaths|new
Recovered|
+-------+-------------------+---------+------+---------+------------+---------+---
```

```
---------+
| France|2020-01-27 00:00:00|        3|     0|        0|           0|        0|
0|
| France|2020-01-28 00:00:00|        4|     0|        0|           1|        0|
0|
| France|2020-01-29 00:00:00|        5|     0|        0|           1|        0|
0|
| France|2020-01-30 00:00:00|        5|     0|        0|           0|        0|
0|
| France|2020-01-31 00:00:00|        5|     0|        0|           0|        0|
0|
| France|2020-02-01 00:00:00|        6|     0|        0|           1|        0|
0|
| France|2020-02-02 00:00:00|        6|     0|        0|           0|        0|
0|
| France|2020-02-03 00:00:00|        6|     0|        0|           0|        0|
```

```
covidR1 = covid.crossJoin( covid_ )

dc = ( covidR1.date.cast("long") - covidR1.d.cast("long") ) / 3600. / 24.

covidR = covidR1.filter( (covidR1.country == covidR1.pay) & ( dc == 1. ) ).\
            drop(  "pay", "d", "newDeaths", "recovered", "newRecovered", "deaths"
).\
            filter( (col("ncy") > 0 ) & ( col("newConfirmed") > 1. ) ).\
            withColumn( "R", col("newConfirmed") / col("ncy") )

covidR.filter("country = 'France' ").show(300)
```

```
+-------+-------------------+---------+------------+-----+--------------------+
|country|               date|confirmed|newConfirmed|  ncy|                   R|
+-------+-------------------+---------+------------+-----+--------------------+
| France|2020-02-26 00:00:00|       18|           4|    2|                 2.0|
| France|2020-02-27 00:00:00|       38|          20|    4|                 5.0|
| France|2020-02-28 00:00:00|       57|          19|   20|                0.95|
| France|2020-02-29 00:00:00|      100|          43|   19|   2.263157894736842|
| France|2020-03-01 00:00:00|      130|          30|   43|  0.6976744186046512|
| France|2020-03-02 00:00:00|      191|          61|   30|   2.033333333333333|
| France|2020-03-03 00:00:00|      204|          13|   61| 0.21311475409836064|
| France|2020-03-04 00:00:00|      288|          84|   13|   6.461538461538462|
| France|2020-03-05 00:00:00|      380|          92|   84|  1.0952380952380953|
| France|2020-03-06 00:00:00|      656|         276|   92|                 3.0|
| France|2020-03-07 00:00:00|      959|         303|  276|  1.0978260869565217|
| France|2020-03-08 00:00:00|     1136|         177|  303|  0.5841584158415841|
| France|2020-03-09 00:00:00|     1219|          83|  177|  0.4689265536723164|
| France|2020-03-10 00:00:00|     1794|         575|   83|   6.927710843373494|
| France|2020-03-11 00:00:00|     2293|         499|  575|  0.8678260869565217|
```

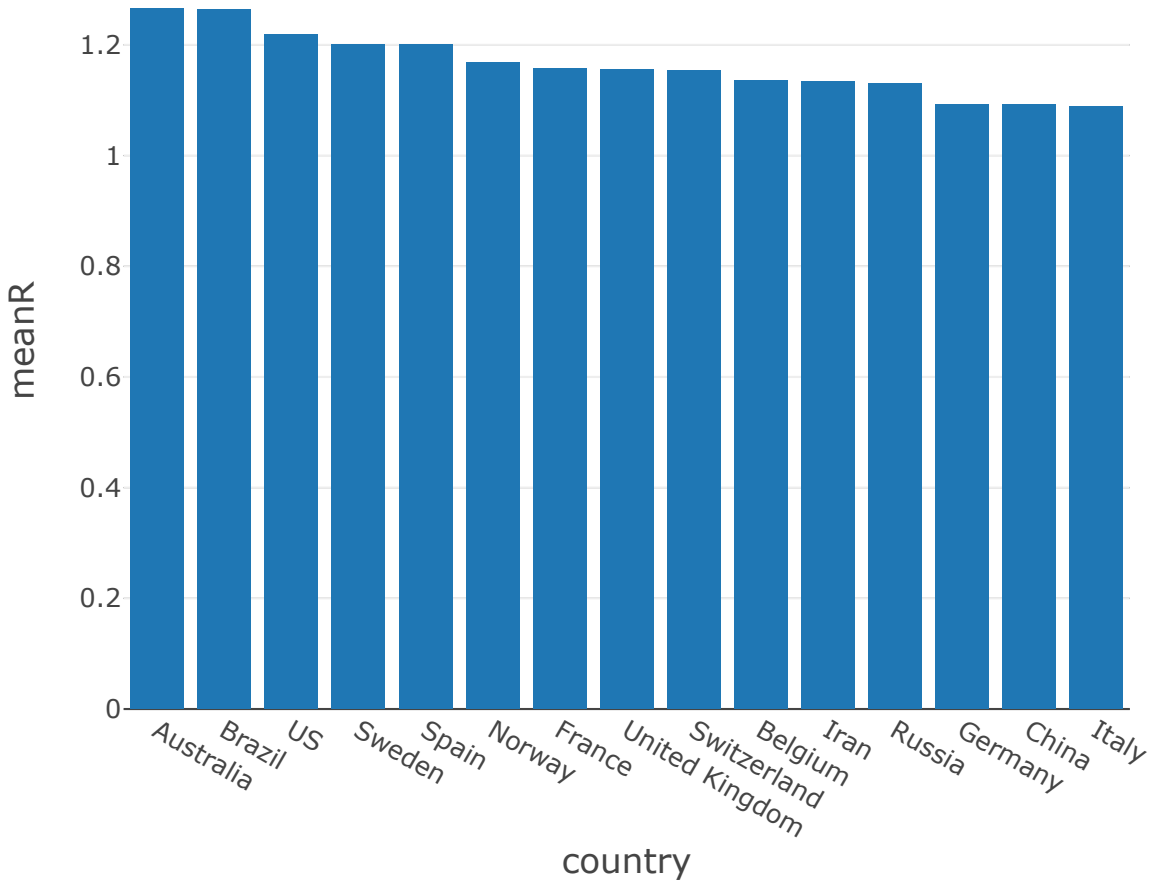
```
| France|2020-03-14 00:00:00|     4496|         815| 1388|  0.5871757925072046|
| France|2020-03-15 00:00:00|     4532|          36|  815|0.044171779141104296|
| France|2020-03-16 00:00:00|     6683|        2151|   36|               59.75|
```

```
from pyspark.sql.functions import expr
from pyspark.sql import Window
median = expr('percentile_approx(R, 0.5)')

part = Window.partitionBy( "country" )

#covidR.filter("country='France'").approxQuantile("R", [.5], 0)[0]
covidRcl = covidR.withColumn( "medianR", median.over(part) )\
                 .filter( col("R") < 5*col("medianR") )
```

```
covidRcl.filter(" country = 'France' ").show()
```

```
+-------+-------------------+---------+------------+----+--------------------+----
--------------+
|country|               date|confirmed|newConfirmed| ncy|                   R|
medianR|
+-------+-------------------+---------+------------+----+--------------------+----
--------------+
| France|2020-02-26 00:00:00|       18|           4|   2|                 2.0|1.02
60764099454214|
| France|2020-02-27 00:00:00|       38|          20|   4|                 5.0|1.02
60764099454214|
| France|2020-02-28 00:00:00|       57|          19|  20|                0.95|1.02
60764099454214|
| France|2020-02-29 00:00:00|      100|          43|  19|   2.263157894736842|1.02
60764099454214|
| France|2020-03-01 00:00:00|      130|          30|  43|  0.6976744186046512|1.02
60764099454214|
| France|2020-03-02 00:00:00|      191|          61|  30|   2.033333333333333|1.02
60764099454214|
| France|2020-03-03 00:00:00|      204|          13|  61| 0.21311475409836064|1.02
60764099454214|
| France|2020-03-05 00:00:00|      380|          92|  84|  1.0952380952380953|1.02
```

```
from pyspark.sql import functions as fun
cc =  ["France", "Germany","China", "Belgium", "Italy", "Spain", "United Kingdom",
"Sweden",
            "US", "Switzerland", "Norway", "Iran", "Russia", "Australia", "Brazil"]

covidMeanR = covidRcl.filter( ( col("country").isin( cc ) ) &
                ( col("date") > lit("2020-02-15") ) &
                ( col("date") < lit("2020-05-15") )
                ).\
        select( "R", "country" ).\
        groupby( "country").\
        agg( fun.round( fun.mean( "R" ), 2).alias("meanR") ).\
        orderBy( "meanR", ascending=False )



display( covidMeanR )
```



Exercise : Statistical Comparison

```
covidMeanRall = covidRcl.filter( ( col("date") > lit("2020-02-15") ) &
                                 ( col("date") < lit("2020-05-15") )
                ).\
        select( "R", "country" ).\
        groupby( "country").\
        agg( fun.round( fun.mean( "R" ), 2).alias("meanR"),
             fun.round( fun.stddev( "R" ), 2).alias( "sigmaR" ),
             fun.count( "R").alias( "n_measure" ) ).\
        orderBy( "meanR", ascending=False )
```

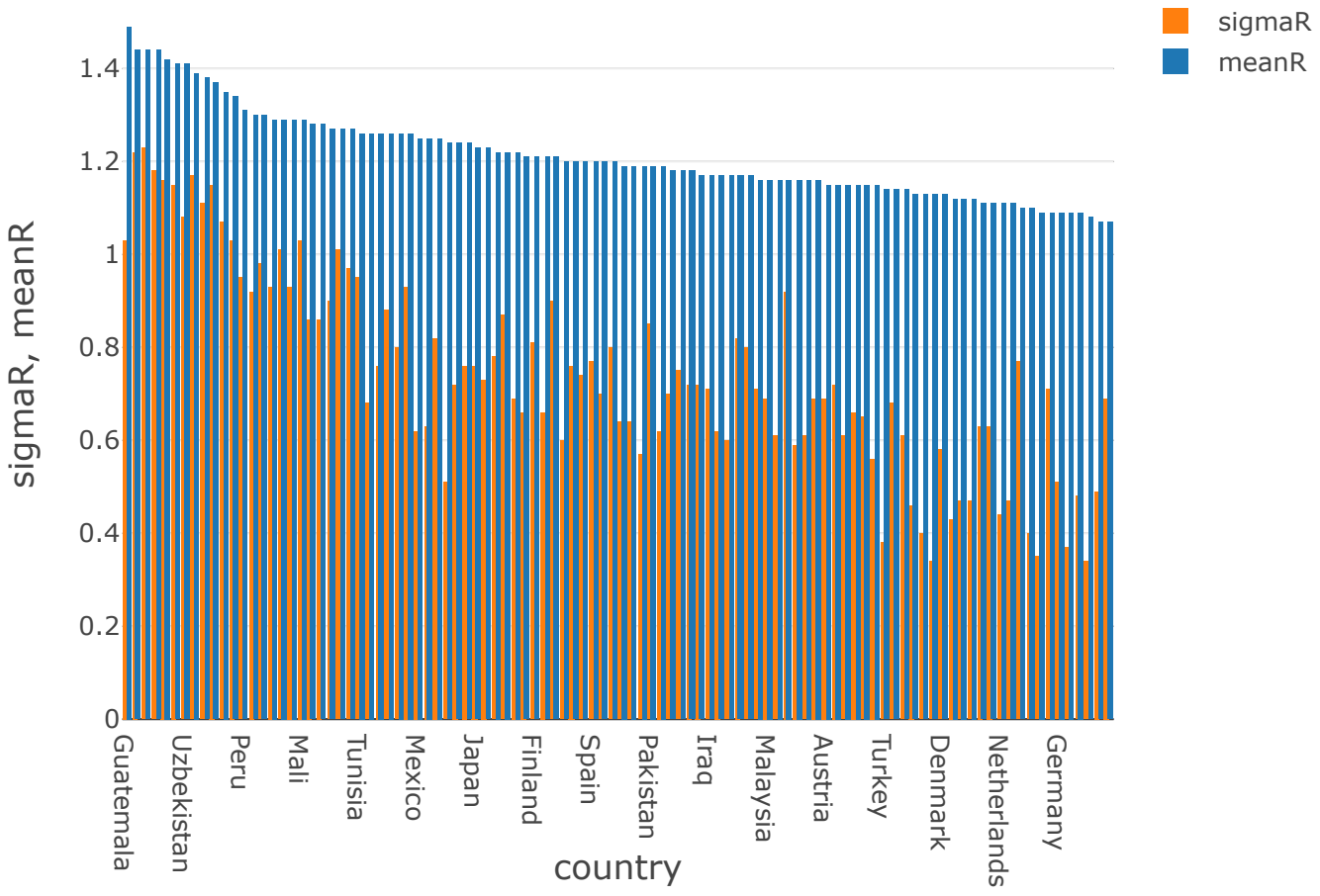
```
covidMeanRall.printSchema()
covidMeanRall.filter( fun.col("n_measure") > 20 )\
             .withColumn( "precision", fun.col( "sigmaR" )/fun.sqrt( "n_measure" )
)\
             .orderBy("meanR", ascending=False).show(200)
```

```
root
 |-- country: string (nullable = true)
 |-- meanR: double (nullable = true)
 |-- sigmaR: double (nullable = true)
 |-- n_measure: long (nullable = false)

+--------------------+-----+------+---------+--------------------+
|             country|meanR|sigmaR|n_measure|           precision|
+--------------------+-----+------+---------+--------------------+
|              Zambia| 1.79|  1.79|       24| 0.36538221996515746|
|          Montenegro| 1.62|   1.3|       26| 0.25495097567963926|
|            Ethiopia| 1.58|  1.37|       33| 0.23848638865930605|
|          Cabo Verde| 1.53|  1.43|       21|  0.3120515830374691|
|           Venezuela| 1.53|  1.24|       33| 0.21585629338506532|
|           Guatemala| 1.49|  1.03|       44| 0.15527834245754826|
|             Bahrain| 1.44|  1.23|       67|  0.1502684165582554|
|              Latvia| 1.44|  1.22|       59|  0.1588304713966051|
|             Lebanon| 1.44|  1.18|       59| 0.15362291495737215|
|             Estonia| 1.42|  1.16|       62| 0.14732014732022097|
|  West Bank and Gaza| 1.42|   1.2|       35| 0.20283702113484398|
|            Honduras| 1.41|  1.15|       45| 0.17143187827498385|
```

```
covidMeanRall.filter( ( fun.col("n_measure") > 20 ) & ( fun.col("country").isin( cc ) ) )\
             .withColumn( "precision", fun.round( fun.col( "sigmaR" )/fun.sqrt( "n_measure" ), 2 ) )\
             .orderBy("meanR", ascending=False).show(200)
```

```
+--------------+-----+------+---------+---------+
|       country|meanR|sigmaR|n_measure|precision|
+--------------+-----+------+---------+---------+
|     Australia| 1.27|   0.9|       73|     0.11|
|        Brazil| 1.26|  0.76|       64|      0.1|
|            US| 1.22|  0.69|       76|     0.08|
|        Sweden|  1.2|  0.74|       73|     0.09|
|         Spain|  1.2|  0.77|       73|     0.09|
|        Norway| 1.17|   0.8|       72|     0.09|
|        France| 1.16|  0.92|       64|     0.12|
|United Kingdom| 1.16|  0.61|       72|     0.07|
|   Switzerland| 1.15|  0.72|       71|     0.09|
|       Belgium| 1.14|  0.61|       71|     0.07|
|        Russia| 1.13|  0.34|       62|     0.04|
|          Iran| 1.13|  0.46|       85|     0.05|
|       Germany| 1.09|  0.51|       76|     0.06|
|         China| 1.09|  0.71|       82|     0.08|
|         Italy| 1.09|  0.37|       81|     0.04|
+--------------+-----+------+---------+---------+
```

```
display( covidMeanRall.filter( fun.col("n_measure") > 40  ) )
```

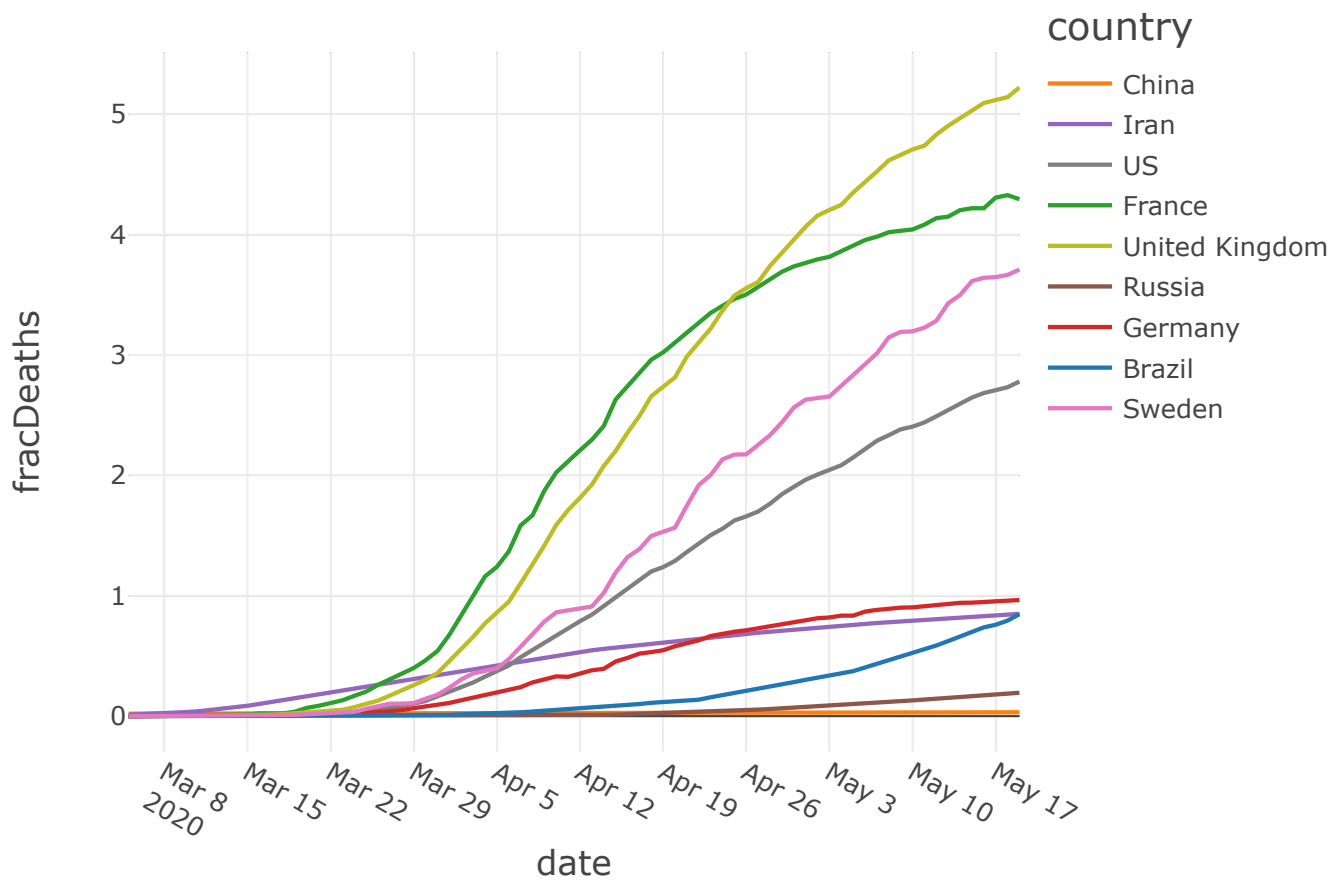



```
covidMeanRall.rdd.getNumPartitions()
```

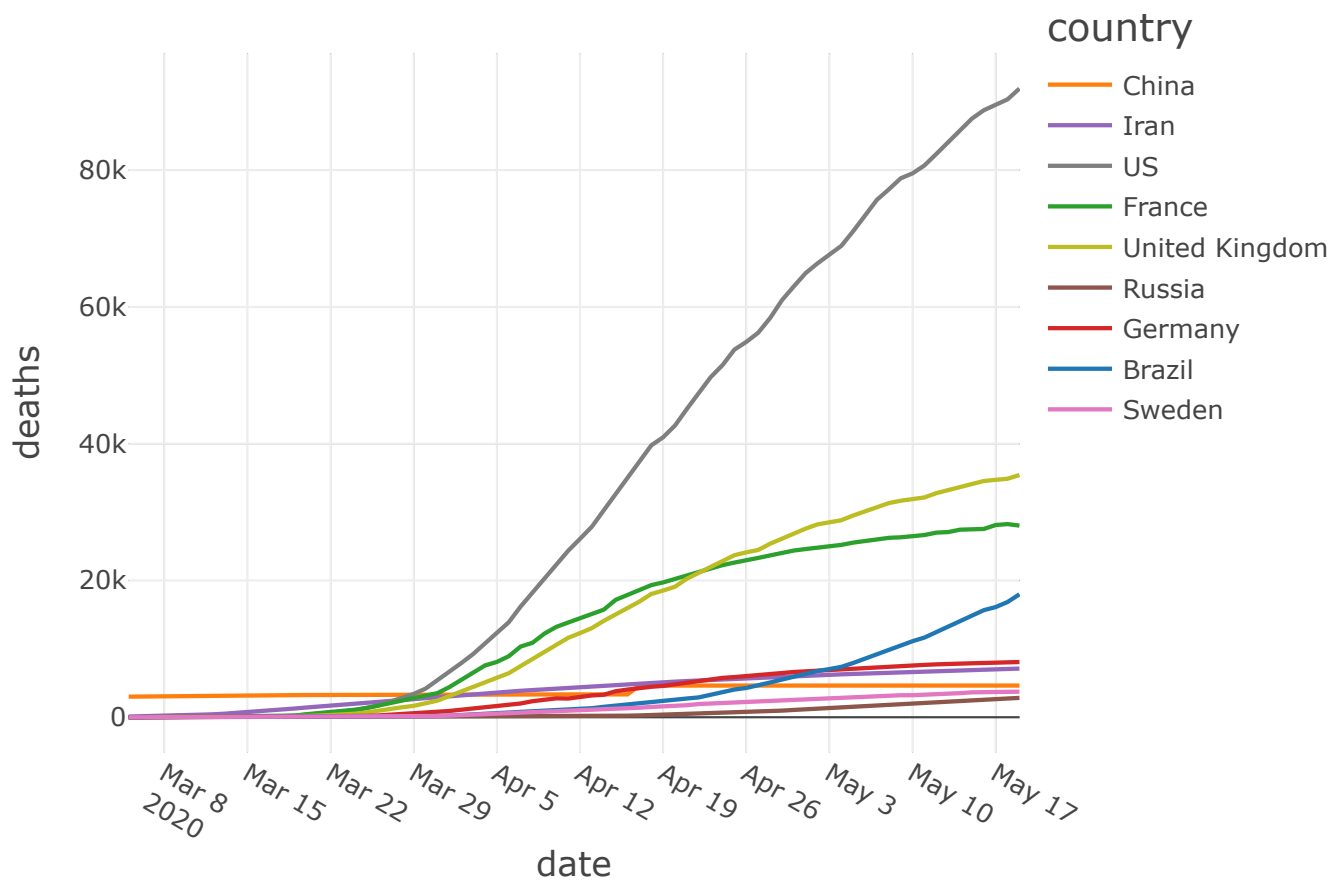
```
Out[88]: 71
```

Exercise : Normalised vs un-normalised deaths

```
cc1 =  ["France", "Germany","China", "United Kingdom", "Sweden", "Iran", "Russia",
"Brazil", "US"]
toShow3 = covidPop.filter( covid.country.isin( cc1 ) ).\
                filter( ( col("date") > lit("2020-03-05") ) & ( col("date") <
lit("2020-05-20") ))

display( toShow3 )
```




```
display( toShow3 )
```




```
cMax = covid.groupby( "country" ).agg( fun.max( "confirmed" ).alias("maxC") )\
            .orderBy( "maxC", ascending=False )\
            .take(3)
```

```
+-------+-------+
|country|   maxC|
+-------+-------+
|     US|1528568|
| Russia| 299941|
| Brazil| 271885|
+-------+-------+
```