Source: Pixabay

The Most Complete Guide to pySpark DataFrames

A bookmarkable cheatsheet containing all the Dataframe Functionality you might need

Rahul Agarwal May 27, 2020 · 17 min read ★

Big Data has become synonymous with Data engineering. But the line between Data Engineering and Data scientists is blurring day by day. At this point in time, I think that Big Data must be in the repertoire of all data scientists.

Reason: ***Too much data is getting generated day by day***

And that brings us to Spark which is one of the most used tools when it comes to working with Big Data.

While once upon a time Spark used to be heavily reliant on RDD manipulations, Spark has now provided a DataFrame API for us Data Scientists to work with. Here is the documentation for the adventurous folks. But while the documentation is good, it does not explain it from the perspective of a Data Scientist. Neither does it properly document the most common use cases for Data Science.

In this post, I will talk about installing Spark, standard Spark functionalities you will need to work with DataFrames, and finally some tips to handle the inevitable errors you will face.

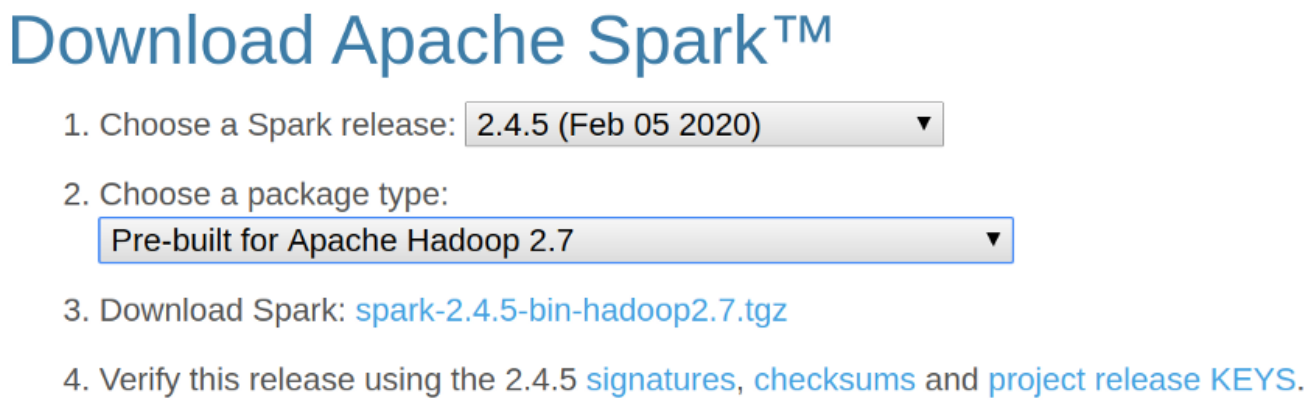
This post is going to be quite long. Actually one of my longest posts on medium, so go on and pick up a Coffee.

Also here is the Table of Contents if you want to skip to a specific section:

Installation

After that, you can just go through these steps:

1. Download the Spark Binary from Apache Spark Website. And click on the Download Spark link to download Spark.



2. Once you have downloaded the above file, you can start with unzipping the file in your home directory. Just Open up the terminal and put these commands in.

```
cd ~
cp Downloads/spark-2.4.5-bin-hadoop2.7.tgz ~
tar -zxvf spark-2.4.5-bin-hadoop2.7.tgz
```

3. Check your Java Version. As of version 2.4 Spark works with Java 8. You can check your Java Version using the command `java -version` on the terminal window.

I had Java 11 in my machine, so I had to run the following commands on my terminal to install and change default Java to Java 8:

```
sudo apt install openjdk-8-jdk
sudo update-alternatives --config java
```

You will need to manually select the Java version 8 by typing the selection number.

```
  1            /usr/lib/jvm/java-11-openjdk-amd64/bin/java      1111      manual mode
  2            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      manual mode

Press <enter> to keep the current choice[*], or type selection number: 2
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java to provide /usr/bin/java (java) in manual mode
```

Rechecking Java version should give something like:

```
(base) rahul@rahul-MS-7A93:~$ java -version
openjdk version "1.8.0_252"
OpenJDK Runtime Environment (build 1.8.0_252-8u252-b09-1~18.04-b09)
OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)
```

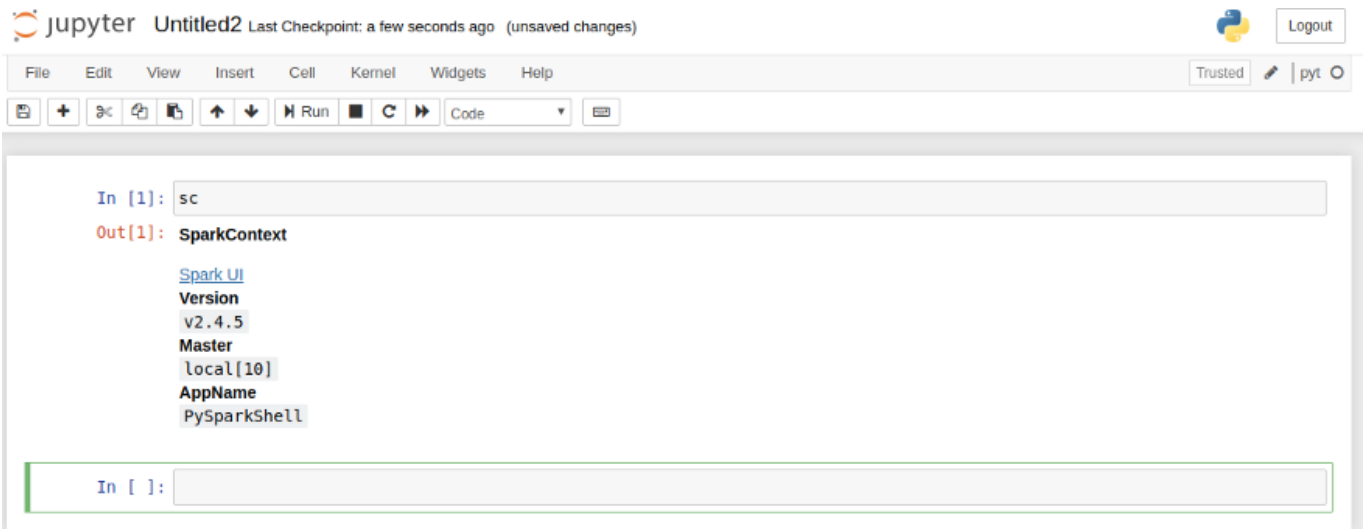
4. Edit your `~/.bashrc` file and add the following lines at the end of the file:

```
function pysparknb ()
{
#Spark path
SPARK_PATH=~/spark-2.4.5-bin-hadoop2.7

export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

# For pyarrow 0.15 users, you have to add the line below or you will
get an error while using pandas_udf
export ARROW_PRE_0_15_IPC_FORMAT=1

# Change the local[10] to local[numCores in your machine]
$SPARK_PATH/bin/pyspark --master local[10]
}
```

5. Source `~/.bashrc`

```
source ~/.bashrc
```

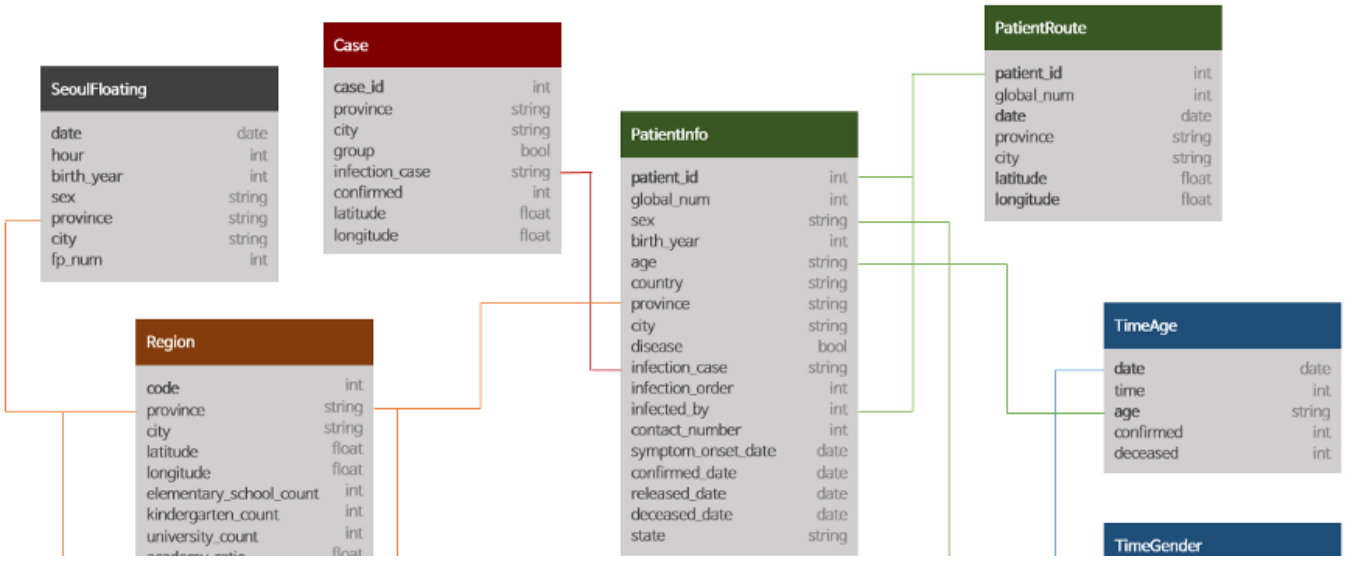
6. Run the `pysparknb` function in the terminal and you will be able to access the notebook. You will be able to open a new notebook as well as the `sparkcontext` will be loaded automatically.

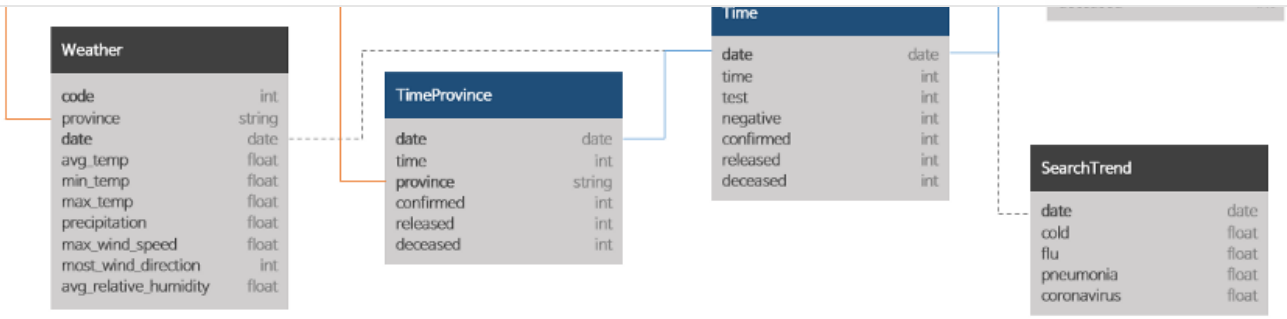


Data

With the installation out of the way, we can move to the more interesting part of this post. I will be working with the Data Science for COVID-19 in South Korea, which is one of the most detailed datasets on the internet for COVID.

Please note that I will be using this dataset to showcase some of the most useful functionalities of Spark, but this should not be in any way considered a data exploration exercise for this amazing dataset.





Source: Kaggle

I will mainly work with the following three tables only in this post:

- Cases
- Region
- TimeProvince

You can find all the code at the GitHub repository.

1. Basic Functions

Read

We can start by loading the files in our dataset using the spark.read.load command. This command reads parquet files, which is the default file format for spark, but you can add the parameter `format` to read .csv files using it.

```
cases =
spark.read.load("/home/rahul/projects/sparkdf/coronavirusdataset/Case
.csv",format="csv", sep=",", inferSchema="true", header="true")
```

See a few rows in the file

```
cases.show()
```

```
+-------+--------+--------------+-----+--------------------+---------+---------+----------+
|case_id|province|          city|group|      infection_case|confirmed| latitude| longitude|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+
|1000001|   Seoul|     Yongsan-gu| true|       Itaewon Clubs|       72|37.538621|126.992652|
|1000002|   Seoul|        Guro-gu| true| Guro-gu Call Center|       98|37.508163|126.884387|
|1000003|   Seoul| Dongdaemun-gu| true|       Dongan Church|       20|37.592888|127.056766|
|1000004|   Seoul|        Guro-gu| true|Manmin Central Ch...|       41|37.481059|126.894343|
|1000005|   Seoul|  Eunpyeong-gu| true|Eunpyeong St. Mar...|       14| 37.63369|  126.9165|
|1000006|   Seoul|  Seongdong-gu| true|    Seongdong-gu APT|       13| 37.55713|  127.0403|
|1000007|   Seoul|     Jongno-gu| true|Jongno Community ...|       10| 37.57681|   127.006|
|1000008|   Seoul|        Jung-gu| true|Jung-gu Fashion C...|        7|37.562405|126.984377|
|1000009|   Seoul|from other city| true|  Shincheonji Church|        8|        -|         -|
|1000010|   Seoul|              -|false|     overseas inflow|      321|        -|         -|
|1000011|   Seoul|              -|false|contact with patient|       18|        -|         -|
|1000012|   Seoul|              -|false|                 etc|       24|        -|         -|
|1100001|   Busan|     Dongnae-gu| true|       Onchun Church|       39| 35.21628|  129.0771|
|1100002|   Busan|from other city| true|  Shincheonji Church|       12|        -|         -|
|1100003|   Busan|     Suyeong-gu| true|Suyeong-gu Kinder...|        5| 35.16708|  129.1124|
|1100004|   Busan|    Haeundae-gu| true|Haeundae-gu Catho...|        6| 35.20599|  129.1256|
|1100005|   Busan|         Jin-gu| true|      Jin-gu Academy|        4| 35.17371|  129.0633|
|1100006|   Busan|from other city| true|Cheongdo Daenam H...|        1|        -|         -|
|1100007|   Busan|              -|false|     overseas inflow|       25|        -|         -|
|1100008|   Busan|              -|false|contact with patient|       18|        -|         -|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+
only showing top 20 rows
```

This file contains the cases grouped by way of the infection spread. This might have helped in the rigorous tracking of Corona Cases in South Korea.

The way this file looks is great right now, but sometimes as we increase the number of columns, the formatting becomes not too great. I have noticed that the following trick helps in displaying in pandas format in my Jupyter Notebook. The `.toPandas()` function converts a spark dataframe into a pandas Dataframe which is easier to show.

```
cases.limit(10).toPandas()
```

```
cases.limit(10).toPandas()
```

	case_id	province	city	group	infection_case	confirmed	latitude	longitude
0	1000001	Seoul	Yongsan-gu	True	Itaewon Clubs	72	37.538621	126.992652
1	1000002	Seoul	Guro-gu	True	Guro-gu Call Center	98	37.508163	126.884387
2	1000003	Seoul	Dongdaemun-gu	True	Dongan Church	20	37.592888	127.056766
3	1000004	Seoul	Guro-gu	True	Manmin Central Church	41	37.481059	126.894343
4	1000005	Seoul	Eunpyeong-gu	True	Eunpyeong St. Mary's Hospital	14	37.63369	126.9165
5	1000006	Seoul	Seongdong-gu	True	Seongdong-gu APT	13	37.55713	127.0403
6	1000007	Seoul	Jongno-gu	True	Jongno Community Center	10	37.57681	127.006
7	1000008	Seoul	Jung-gu	True	Jung-gu Fashion Company	7	37.562405	126.984377
8	1000009	Seoul	from other city	True	Shincheonji Church	8	-	-
9	1000010	Seoul		False	overseas inflow	321		

Sometimes we would like to change the name of columns in our Spark Dataframes. We can do this simply using the below command to change a single column:

```
cases = cases.withColumnRenamed("infection_case","infection_source")
```

Or for all columns:

```
cases = cases.toDF(*['case_id', 'province', 'city', 'group',
'infection_case', 'confirmed',
       'latitude', 'longitude'])
```

Select Columns

We can select a subset of columns using the `select` keyword.

```
cases = cases.select('province','city','infection_case','confirmed')
cases.show()
```

```
cases.show()
```

```
+--------+--------------+--------------------+---------+
|province|          city|      infection_case|confirmed|
+--------+--------------+--------------------+---------+
|   Seoul|     Yongsan-gu|       Itaewon Clubs|       72|
|   Seoul|        Guro-gu| Guro-gu Call Center|       98|
|   Seoul|  Dongdaemun-gu|       Dongan Church|       20|
|   Seoul|        Guro-gu|Manmin Central Ch...|       41|
|   Seoul|   Eunpyeong-gu|Eunpyeong St. Mar...|       14|
|   Seoul|   Seongdong-gu|    Seongdong-gu APT|       13|
|   Seoul|      Jongno-gu|Jongno Community ...|       10|
|   Seoul|        Jung-gu|Jung-gu Fashion C...|        7|
|   Seoul|from other city|  Shincheonji Church|        8|
|   Seoul|              -|     overseas inflow|      321|
|   Seoul|              -|contact with patient|       18|
|   Seoul|              -|                 etc|       24|
|   Busan|     Dongnae-gu|       Onchun Church|       39|
|   Busan|from other city|  Shincheonji Church|       12|
|   Busan|     Suyeong-gu|Suyeong-gu Kinder...|        5|
|   Busan|    Haeundae-gu|Haeundae-gu Catho...|        6|
|   Busan|         Jin-gu|      Jin-gu Academy|        4|
|   Busan|from other city|Cheongdo Daenam H...|        1|
|   Busan|              -|     overseas inflow|       25|
|   Busan|              -|contact with patient|       18|
+--------+--------------+--------------------+---------+
only showing top 20 rows
```

not change after performing this command as we don't assign it to any variable.

```
cases.sort("confirmed").show()
```

```
cases.sort("confirmed").show()
```

```
+-----------------+--------------+--------------------+---------+
|         province|          city|      infection_case|confirmed|
+-----------------+--------------+--------------------+---------+
|          Jeju-do|             -|contact with patient|        0|
|       Gangwon-do|             -|contact with patient|        0|
|          Gwangju|             -|                 etc|        0|
|            Busan|from other city|Cheongdo Daenam H...|        1|
|          Gwangju|             -|contact with patient|        1|
|          Jeju-do|from other city|       Itaewon Clubs|        1|
|           Sejong|from other city|  Shincheonji Church|        1|
|           Sejong|             -|                 etc|        1|
|Chungcheongnam-do|             -|contact with patient|        1|
|     Jeollabuk-do|from other city|  Shincheonji Church|        1|
|     Jeollanam-do|from other city|  Shincheonji Church|        1|
|          Incheon|from other city|  Shincheonji Church|        2|
|     Jeollanam-do|      Muan-gun|Manmin Central Ch...|        2|
|          Daejeon|from other city|  Shincheonji Church|        2|
|     Jeollanam-do|             -|contact with patient|        2|
|            Daegu|from other city|Cheongdo Daenam H...|        2|
|          Daejeon|from other city|Seosan-si Laboratory|        2|
| Gyeongsangnam-do|from other city|       Onchun Church|        2|
|          Daejeon|        Seo-gu|Korea Forest Engi...|        3|
|           Sejong|             -|     overseas inflow|        3|
+-----------------+--------------+--------------------+---------+
only showing top 20 rows
```

But that is inverted. We want to see the most cases at the top. We can do this using the `F.desc` function:

```
# descending Sort
from pyspark.sql import functions as F
cases.sort(F.desc("confirmed")).show()
```

```
# descending Sort
from pyspark.sql import functions as F
cases.sort(F.desc("confirmed")).show()
```

```
+-----------------+--------------+--------------------+---------+
|         province|          city|      infection_case|confirmed|
+-----------------+--------------+--------------------+---------+
|            Daegu|         Nam-gu|  Shincheonji Church|     4510|
|            Daegu|             -|contact with patient|      929|
|            Daegu|             -|                 etc|      724|
| Gyeongsangbuk-do|from other city|  Shincheonji Church|      566|
|            Seoul|             -|     overseas inflow|      321|
|      Gyeonggi-do|             -|     overseas inflow|      225|
|            Daegu|   Dalseong-gun|Second Mi-Ju Hosp...|      196|
| Gyeongsangbuk-do|             -|contact with patient|      192|
```

```
|            Seoul|     Yongsan-gu|       Itaewon Clubs|       72|
|      Gyeonggi-do|    Seongnam-si|River of Grace Co...|       72|
| Gyeongsangbuk-do|    Bonghwa-gun|Bonghwa Pureun Nu...|       68|
| Gyeongsangbuk-do|   Gyeongsan-si|Gyeongsan Seorin ...|       66|
|      Gyeonggi-do|               -|contact with patient|       57|
|      Gyeonggi-do|   Uijeongbu-si|Uijeongbu St. Mar...|       50|
+-----------------+---------------+--------------------+---------+
only showing top 20 rows
```

We can see the most cases in a logical area in South Korea originated from `Shincheonji Church`.

Cast

Though we don't face it in this dataset, there might be scenarios where Pyspark reads a double as integer or string, In such cases, you can use the cast function to convert types.

```
from pyspark.sql.types import DoubleType, IntegerType, StringType

cases = cases.withColumn('confirmed',
F.col('confirmed').cast(IntegerType()))

cases = cases.withColumn('city', F.col('city').cast(StringType()))
```

Filter

We can filter a data frame using multiple conditions using AND(&), OR(|) and NOT(~) conditions. For example, we may want to find out all the different infection_case in Daegu Province with more than 10 confirmed cases.

```
cases.filter((cases.confirmed>10) & (cases.province=='Daegu')).show()
```

```
cases.filter((cases.confirmed>10) & (cases.province=='Daegu')).show()
+--------+-----------+--------------------+---------+
|province|       city|      infection_case|confirmed|
+--------+-----------+--------------------+---------+
|   Daegu|     Nam-gu|  Shincheonji Church|     4510|
|   Daegu|Dalseong-gun|Second Mi-Ju Hosp...|      196|
|   Daegu|     Seo-gu|Hansarang Convale...|      128|
|   Daegu|Dalseong-gun|Daesil Convalesce...|      100|
|   Daegu|    Dong-gu|     Fatima Hospital|       37|
|   Daegu|          -|     overseas inflow|       24|
|   Daegu|          -|contact with patient|      929|
|   Daegu|          -|                 etc|      724|
+--------+-----------+--------------------+---------+
```

pandas `groupBy` with the exception that you will need to import `pyspark.sql.functions`. Here is the list of functions you can use with this function module.

```
from pyspark.sql import functions as F

cases.groupBy(["province","city"]).agg(F.sum("confirmed")
,F.max("confirmed")).show()
```

```
from pyspark.sql import functions as F

cases.groupBy(["province","city"]).agg(F.sum("confirmed") ,F.max("confirmed")).show()
```

```
+---------------+--------------+-------------+-------------+
|        province|          city|sum(confirmed)|max(confirmed)|
+---------------+--------------+-------------+-------------+
|Gyeongsangnam-do|       Jinju-si|           10|           10|
|          Seoul|        Guro-gu|          139|           98|
|         Daejeon|             -|           27|           10|
|    Jeollabuk-do|from other city|            1|            1|
|Gyeongsangnam-do|Changnyeong-gun|            7|            7|
|          Seoul|             -|          363|          321|
|         Jeju-do|from other city|            1|            1|
|Gyeongsangbuk-do|             -|          336|          192|
|Gyeongsangnam-do|   Geochang-gun|           18|           10|
|         Incheon|from other city|           22|           20|
|           Busan|             -|           72|           29|
|           Daegu|        Seo-gu|          128|          128|
|           Busan|     Suyeong-gu|            5|            5|
|     Gyeonggi-do|   Uijeongbu-si|           50|           50|
|          Seoul|     Yongsan-gu|           72|           72|
|           Daegu|             -|         1677|          929|
|     Gyeonggi-do|    Seongnam-si|           94|           72|
|Gyeongsangnam-do|from other city|           34|           32|
|     Gyeonggi-do|       Suwon-si|           10|           10|
|         Daejeon|from other city|            4|            2|
+---------------+--------------+-------------+-------------+
only showing top 20 rows
```

If you don't like the new column names, you can use the `alias` keyword to rename columns in the `agg` command itself.

```
cases.groupBy(["province","city"]).agg(
    F.sum("confirmed").alias("TotalConfirmed"),\
    F.max("confirmed").alias("MaxFromOneConfirmedCase")\
    ).show()
```

```
cases.groupBy(["province","city"]).agg(
    F.sum("confirmed").alias("TotalConfirmed"),\
    F.max("confirmed").alias("MaxFromOneConfirmedCase")\
    ).show()
```

```
|          Daejeon|               -|           27|                   10|
|    Jeollabuk-do|from other city|            1|                    1|
|Gyeongsangnam-do|Changnyeong-gun|            7|                    7|
|           Seoul|               -|          363|                  321|
|         Jeju-do|from other city|            1|                    1|
|Gyeongsangbuk-do|               -|          336|                  192|
|Gyeongsangnam-do|   Geochang-gun|           18|                   10|
|         Incheon|from other city|           22|                   20|
|           Busan|               -|           72|                   29|
|           Daegu|         Seo-gu|          128|                  128|
|           Busan|     Suyeong-gu|            5|                    5|
|     Gyeonggi-do|   Uijeongbu-si|           50|                   50|
|           Seoul|     Yongsan-gu|           72|                   72|
|           Daegu|               -|         1677|                  929|
|     Gyeonggi-do|    Seongnam-si|           94|                   72|
|Gyeongsangnam-do|from other city|           34|                   32|
|     Gyeonggi-do|       Suwon-si|           10|                   10|
|          Daejeon|from other city|            4|                    2|
+----------------+---------------+-------------+----------------------+
only showing top 20 rows
```

Joins

To Start with Joins we will need to introduce one more CSV file. We will go with the region file which contains region information such as elementary_school_count, elderly_population_ratio, etc.

```
regions =
spark.read.load("/home/rahul/projects/sparkdf/coronavirusdataset/Regi
on.csv",format="csv", sep=",", inferSchema="true", header="true")
regions.limit(10).toPandas()
```

```
regions = spark.read.load("/home/rahul/projects/sparkdf/coronavirusdataset/Region.csv",format="csv", sep=",", inferSchema="true", header="true")
regions.limit(10).toPandas()
```

	code	province	city	latitude	longitude	elementary_school_count	kindergarten_count	university_count	academy_ratio	elderly_population_ratio	elderly_alone_ratio	nursing_home_count
0	10000	Seoul	Seoul	37.566953	126.977977	607	830	48	1.44	15.38	5.8	22739
1	10010	Seoul	Gangnam-gu	37.518421	127.047222	33	38	0	4.18	13.17	4.3	3088
2	10020	Seoul	Gangdong-gu	37.530492	127.123837	27	32	0	1.54	14.55	5.4	1023
3	10030	Seoul	Gangbuk-gu	37.639938	127.025508	14	21	0	0.67	19.49	8.5	628
4	10040	Seoul	Gangseo-gu	37.551166	126.849506	36	56	1	1.17	14.39	5.7	1060
5	10050	Seoul	Gwanak-gu	37.478290	126.951502	22	33	1	0.89	15.12	4.9	909
6	10060	Seoul	Gwangjin-gu	37.538712	127.082366	22	33	3	1.16	13.75	4.8	723
7	10070	Seoul	Guro-gu	37.495632	126.887650	26	34	3	1.00	16.21	5.7	741
8	10080	Seoul	Geumcheon-gu	37.456852	126.895229	18	19	0	0.96	16.15	6.7	475
9	10090	Seoul	Nowon-gu	37.654259	127.056294	42	66	6	1.39	15.40	7.4	952

We want to get this information in our cases file by joining the two DataFrames. We can do this by using:

```
cases = cases.join(regions, ['province','city'],how='left')
cases.limit(10).toPandas()
```

1	Seoul	Guro-gu	Guro-gu Call Center	98	10070.0	37.495632	126.887650	26.0	34.0	3.0	1.00	16.21	5.7	741.0
2	Seoul	Dongdaemun-gu	Dongan Church	20	10110.0	37.574552	127.039721	21.0	31.0	4.0	1.06	17.26	6.7	832.0
3	Seoul	Guro-gu	Manmin Central Church	41	10070.0	37.495632	126.887650	26.0	34.0	3.0	1.00	16.21	5.7	741.0
4	Seoul	Eunpyeong-gu	Eunpyeong St. Mary's Hospital	14	10220.0	37.603481	126.929173	31.0	44.0	1.0	1.09	17.00	6.5	874.0
5	Seoul	Seongdong-gu	Seongdong-gu APT	13	10160.0	37.563277	127.036647	21.0	30.0	2.0	0.97	14.76	5.3	593.0
6	Seoul	Jongno-gu	Jongno Community Center	10	10230.0	37.572999	126.979189	13.0	17.0	3.0	1.71	18.27	6.8	668.0
7	Seoul	Jung-gu	Jung-gu Fashion Company	7	10240.0	37.563988	126.997530	12.0	14.0	2.0	0.94	18.42	7.4	728.0
8	Seoul	from other city	Shincheonji Church	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	Seoul	-	overseas inflow	321	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2. Broadcast/Map Side Joins

Sometimes you might face a scenario where you need to join a very big table(~1B Rows) with a very small table(~100–200 rows). The scenario might also involve increasing the size of your database like in the example below.

BIG

	A	price
0	1	0
1	1	1
2	1	2
3	1	3
4	2	4
5	2	5
6	2	6
7	2	7

X

SMALL

	A	agg
0	1	sum
1	1	mean
2	1	max
3	1	min
4	2	sum
5	2	mean

=

	A	price	agg
0	1	0	sum
1	1	0	mean
2	1	0	max
3	1	0	min
4	1	1	sum
5	1	1	mean
6	1	1	max
7	1	1	min
8	1	2	sum
9	1	2	mean
10	1	2	max
11	1	2	min
12	1	3	sum
13	1	3	mean
14	1	3	max
15	1	3	min
16	2	4	sum
17	2	4	mean
18	2	5	sum
19	2	5	mean
20	2	6	sum
21	2	6	mean
22	2	7	sum
23	2	7	mean

operations to a particular key. But assuming that the data for each key in the Big table is large, it will involve a lot of data movement. And sometimes so much that the application itself breaks. A small optimization then you can do when joining on such big tables(assuming the other table is small) is to broadcast the small table to each machine/node when you perform a join. You can do this easily using the broadcast keyword. This has been a lifesaver many times with Spark when everything else fails.

```
from pyspark.sql.functions import broadcast
cases = cases.join(broadcast(regions),
['province','city'],how='left')
```

3. Use SQL with DataFrames

If you want, you can also use SQL with data frames. Let us try to run some SQL on the cases table.

We first register the cases dataframe to a temporary table cases_table on which we can run SQL operations. As you can see, the result of the SQL select statement is again a Spark Dataframe.

```
cases.registerTempTable('cases_table')
newDF = sqlContext.sql('select * from cases_table where
confirmed>100')
newDF.show()
```

```
cases.registerTempTable('cases_table')
newDF = sqlContext.sql('select * from cases_table where confirmed>100')
```

```
newDF.show()
```

```
+-------+----------------+--------------+-----+--------------------+---------+---------+----------+
|case_id|        province|          city|group|      infection_case|confirmed| latitude| longitude|
+-------+----------------+--------------+-----+--------------------+---------+---------+----------+
|1000010|           Seoul|             -|false|     overseas inflow|      321|        -|         -|
|1200001|           Daegu|        Nam-gu| true|  Shincheonji Church|     4510| 35.84008|  128.5667|
|1200002|           Daegu|  Dalseong-gun| true|Second Mi-Ju Hosp...|      196|35.857375|128.466651|
|1200003|           Daegu|        Seo-gu| true|Hansarang Convale...|      128|35.885592|128.556649|
```

```
|6000002| Gyeongsangbuk-do|   Cheongdo-gun| true|Cheongdo Daenam H...|      120| 35.64887|  128.7368|
|6000011| Gyeongsangbuk-do|               -|false|contact with patient|      192|        -|         -|
|6000012| Gyeongsangbuk-do|               -|false|                 etc|      134|        -|         -|
+-------+-----------------+---------------+-----+--------------------+---------+---------+----------+
```

I have shown a minimal example above, but you can use pretty much complex SQL queries involving GROUP BY, HAVING, AND ORDER BY clauses as well as aliases in the above query.

4. Create New Columns

There are many ways that you can use to create a column in a PySpark Dataframe. I will try to show the most usable of them.

Using Spark Native Functions

The most pysparkish way to create a new column in a PySpark DataFrame is by using built-in functions. This is the most performant programmatical way to create a new column, so this is the first place I go whenever I want to do some column manipulation.

We can use `.withcolumn` along with PySpark SQL functions to create a new column. In essence, you can find String functions, Date functions, and Math functions already implemented using Spark functions. Our first function, the `F.col` function gives us access to the column. So if we wanted to add 100 to a column, we could use `F.col` as:

```
import pyspark.sql.functions as F
casesWithNewConfirmed = cases.withColumn("NewConfirmed", 100 +
F.col("confirmed"))

casesWithNewConfirmed.show()
```

```
casesWithNewConfirmed = cases.withColumn("NewConfirmed", 100 + F.col("confirmed"))
casesWithNewConfirmed.show()
```

```
+-------+--------+--------------+-----+--------------------+---------+---------+----------+------------+
|case_id|province|          city|group|      infection_case|confirmed| latitude| longitude|NewConfirmed|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+------------+
|1000001|   Seoul|  Yongsan-gu| true|       Itaewon Clubs|       72|37.538621|126.992652|         172|
```

```
|1000007|   Seoul|      Jongno-gu| true|Jongno Community ...|       10| 37.57681|   127.006|         110|
|1000008|   Seoul|        Jung-gu| true|Jung-gu Fashion C...|        7|37.562405|126.984377|         107|
|1000009|   Seoul|from other city| true|  Shincheonji Church|        8|        -|         -|         108|
|1000010|   Seoul|              -|false|     overseas inflow|      321|        -|         -|         421|
|1000011|   Seoul|              -|false|contact with patient|       18|        -|         -|         118|
|1000012|   Seoul|              -|false|                 etc|       24|        -|         -|         124|
|1100001|   Busan|     Dongnae-gu| true|       Onchun Church|       39| 35.21628|  129.0771|         139|
|1100002|   Busan|from other city| true|  Shincheonji Church|       12|        -|         -|         112|
|1100003|   Busan|     Suyeong-gu| true|Suyeong-gu Kinder...|        5| 35.16708|  129.1124|         105|
|1100004|   Busan|    Haeundae-gu| true|Haeundae-gu Catho...|        6| 35.20599|  129.1256|         106|
|1100005|   Busan|         Jin-gu| true|      Jin-gu Academy|        4| 35.17371|  129.0633|         104|
|1100006|   Busan|from other city| true|Cheongdo Daenam H...|        1|        -|         -|         101|
|1100007|   Busan|              -|false|     overseas inflow|       25|        -|         -|         125|
|1100008|   Busan|              -|false|contact with patient|       18|        -|         -|         118|
+-------+--------+---------------+-----+--------------------+---------+---------+----------+------------+
only showing top 20 rows
```

We can also use math functions like `F.exp` function:

```
casesWithExpConfirmed = cases.withColumn("ExpConfirmed",
F.exp("confirmed"))

casesWithExpConfirmed.show()
```

```
casesWithExpConfirmed = cases.withColumn("ExpConfirmed", F.exp("confirmed"))
casesWithExpConfirmed.show()
```

```
+-------+--------+---------------+-----+--------------------+---------+---------+----------+--------------------+
|case_id|province|           city|group|      infection_case|confirmed| latitude| longitude|        ExpConfirmed|
+-------+--------+---------------+-----+--------------------+---------+---------+----------+--------------------+
|1000001|   Seoul|     Yongsan-gu| true|       Itaewon Clubs|       72|37.538621|126.992652|1.858671745284127...|
|1000002|   Seoul|        Guro-gu| true| Guro-gu Call Center|       98|37.508163|126.884387|3.637970947608805E42|
|1000003|   Seoul|  Dongdaemun-gu| true|       Dongan Church|       20|37.592888|127.056766| 4.851651954097903E8|
|1000004|   Seoul|        Guro-gu| true|Manmin Central Ch...|       41|37.481059|126.894343|6.398434935300549...|
|1000005|   Seoul|   Eunpyeong-gu| true|Eunpyeong St. Mar...|       14| 37.63369|  126.9165|  1202604.2841647768|
|1000006|   Seoul|   Seongdong-gu| true|    Seongdong-gu APT|       13| 37.55713|  127.0403|   442413.3920089205|
|1000007|   Seoul|      Jongno-gu| true|Jongno Community ...|       10| 37.57681|   127.006|  22026.465794806718|
|1000008|   Seoul|        Jung-gu| true|Jung-gu Fashion C...|        7|37.562405|126.984377|  1096.6331584284585|
|1000009|   Seoul|from other city| true|  Shincheonji Church|        8|        -|         -|  2980.9579870417283|
|1000010|   Seoul|              -|false|     overseas inflow|      321|        -|         -|2.56170249311968E139|
|1000011|   Seoul|              -|false|contact with patient|       18|        -|         -| 6.565996913733051E7|
|1000012|   Seoul|              -|false|                 etc|       24|        -|         -|2.648912212984347E10|
|1100001|   Busan|     Dongnae-gu| true|       Onchun Church|       39| 35.21628|  129.0771|8.659340042399374...|
|1100002|   Busan|from other city| true|  Shincheonji Church|       12|        -|         -|  162754.79141900392|
|1100003|   Busan|     Suyeong-gu| true|Suyeong-gu Kinder...|        5| 35.16708|  129.1124|   148.4131591025766|
|1100004|   Busan|    Haeundae-gu| true|Haeundae-gu Catho...|        6| 35.20599|  129.1256|   403.4287934927351|
|1100005|   Busan|         Jin-gu| true|      Jin-gu Academy|        4| 35.17371|  129.0633|  54.598150033144236|
|1100006|   Busan|from other city| true|Cheongdo Daenam H...|        1|        -|         -|   2.718281828459045|
|1100007|   Busan|              -|false|     overseas inflow|       25|        -|         -|7.200489933738588E10|
|1100008|   Busan|              -|false|contact with patient|       18|        -|         -| 6.565996913733051E7|
+-------+--------+---------------+-----+--------------------+---------+---------+----------+--------------------+
only showing top 20 rows
```

There are a lot of other functions provided in this module, which are enough for most simple use cases. You can check out the functions list here.

Using Spark UDFs

multiple columns. While Spark SQL functions do solve many use cases when it comes to column creation, I use Spark UDF whenever I need more matured Python functionality.

To use Spark UDFs, we need to use the `F.udf` function to convert a regular python function to a Spark UDF. We also need to specify the return type of the function. In this example the return type is `StringType()`

```
import pyspark.sql.functions as F
from pyspark.sql.types import *
def casesHighLow(confirmed):
    if confirmed < 50:
        return 'low'
    else:
        return 'high'

#convert to a UDF Function by passing in the function and return type
of function
casesHighLowUDF = F.udf(casesHighLow, StringType())

CasesWithHighLow = cases.withColumn("HighLow",
casesHighLowUDF("confirmed"))
CasesWithHighLow.show()
```

```
import pyspark.sql.functions as F
from pyspark.sql.types import *
def casesHighLow(confirmed):
    if confirmed < 50:
        return 'low'
    else:
        return 'high'

#convert to a UDF Function by passing in the function and return type of function
casesHighLowUDF = F.udf(casesHighLow, StringType())

CasesWithHighLow = cases.withColumn("HighLow", casesHighLowUDF("confirmed"))
CasesWithHighLow.show()
```

```
+-------+--------+--------------+-----+--------------------+---------+---------+----------+-------+
|case_id|province|          city|group|      infection_case|confirmed| latitude| longitude|HighLow|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+-------+
|1000001|   Seoul|     Yongsan-gu| true|       Itaewon Clubs|       72|37.538621|126.992652|   high|
|1000002|   Seoul|        Guro-gu| true| Guro-gu Call Center|       98|37.508163|126.884387|   high|
|1000003|   Seoul|  Dongdaemun-gu| true|       Dongan Church|       20|37.592888|127.056766|    low|
|1000004|   Seoul|        Guro-gu| true|Manmin Central Ch...|       41|37.481059|126.894343|    low|
|1000005|   Seoul|   Eunpyeong-gu| true|Eunpyeong St. Mar...|       14| 37.63369|  126.9165|    low|
|1000006|   Seoul|   Seongdong-gu| true|    Seongdong-gu APT|       13| 37.55713|  127.0403|    low|
|1000007|   Seoul|      Jongno-gu| true|Jongno Community ...|       10| 37.57681|   127.006|    low|
|1000008|   Seoul|        Jung-gu| true|Jung-gu Fashion C...|        7|37.562405|126.984377|    low|
|1000009|   Seoul|from other city| true|  Shincheonji Church|        8|        -|         -|    low|
|1000010|   Seoul|              -|false|     overseas inflow|      321|        -|         -|   high|
|1000011|   Seoul|              -|false|contact with patient|       18|        -|         -|    low|
|1000012|   Seoul|              -|false|                 etc|       24|        -|         -|    low|
|1100001|   Busan|     Dongnae-gu| true|       Onchun Church|       39| 35.21628|  129.0771|    low|
|1100002|   Busan|from other city| true|  Shincheonji Church|       12|        -|         -|    low|
|1100003|   Busan|     Suyeong-gu| true|Suyeong-gu Kinder...|        5| 35.16708|  129.1124|    low|
```

```
only showing top 20 rows
```

Using RDDs

This might seem a little odd, but sometimes both the spark UDFs and SQL functions are not enough for a particular use-case. I have observed the RDDs being much more performant in some use-cases in real life. You might want to utilize the better partitioning that you get with spark RDDs. Or you may want to use group functions in Spark RDDs.

Whatever the case be, I find this way of using RDD to create new columns pretty useful for people who have experience working with RDDs that is the basic building block in the Spark ecosystem. Don't worry much if you don't understand it. It is just here for completion.

The process below makes use of the functionality to convert between `Row` and `pythondict` objects. We convert a row object to a dictionary. Work with the dictionary as we are used to and convert that dictionary back to row again. This might come in handy in a lot of situations.

```
import math
from pyspark.sql import Row

def rowwise_function(row):
    # convert row to python dictionary:
    row_dict = row.asDict()
    # Add a new key in the dictionary with the new column name and
value.
    # This might be a big complex function.
    row_dict['expConfirmed'] = float(np.exp(row_dict['confirmed']))
    # convert dict to row back again:
    newrow = Row(**row_dict)
    # return new row
    return newrow

# convert cases dataframe to RDD
cases_rdd = cases.rdd

# apply our function to RDD
cases_rdd_new = cases_rdd.map(lambda row: rowwise_function(row))
```

```
casesNewDf.show()
```

```
import math
from pyspark.sql import Row
def rowwise_function(row):
    # convert row to python dictionary:
    row_dict = row.asDict()
    # Add a new key in the dictionary with the new column name and value.
    # This might be a big complex function.
    row_dict['expConfirmed'] = float(np.exp(row_dict['confirmed']))
    # convert dict to row back again:
    newrow = Row(**row_dict)
    # return new row
    return newrow

# convert cases dataframe to RDD
cases_rdd = cases.rdd

# apply our function to RDD
cases_rdd_new = cases_rdd.map(lambda row: rowwise_function(row))

# Convert RDD Back to DataFrame
casesNewDf = sqlContext.createDataFrame(cases_rdd_new)

casesNewDf.show()
```

```
+-------+--------------+---------+--------------------+-----+--------------------+---------+----------+--------+
|case_id|          city|confirmed|        expConfirmed|group|      infection_case| latitude| longitude|province|
+-------+--------------+---------+--------------------+-----+--------------------+---------+----------+--------+
|1000001|     Yongsan-gu|       72|1.858671745284127...| true|       Itaewon Clubs|37.538621|126.992652|   Seoul|
|1000002|        Guro-gu|       98|3.6379709476088O5E42| true| Guro-gu Call Center|37.508163|126.884387|   Seoul|
|1000003|  Dongdaemun-gu|       20| 4.851651954097903E8| true|       Dongan Church|37.592888|127.056766|   Seoul|
|1000004|        Guro-gu|       41|6.398434935300549...| true|Manmin Central Ch...|37.481059|126.894343|   Seoul|
|1000005|   Eunpyeong-gu|       14|  1202604.2841647768| true|Eunpyeong St. Mar...| 37.63369|  126.9165|   Seoul|
|1000006|   Seongdong-gu|       13|   442413.3920089205| true|    Seongdong-gu APT| 37.55713|  127.0403|   Seoul|
|1000007|      Jongno-gu|       10|  22026.465794806718| true|Jongno Community ...| 37.57681|   127.006|   Seoul|
|1000008|        Jung-gu|        7|  1096.6331584284585| true|Jung-gu Fashion C...|37.562405|126.984377|   Seoul|
|1000009|from other city|        8|  2980.9579870417283| true|  Shincheonji Church|        -|         -|   Seoul|
|1000010|              -|      321|2.56170249311968E139|false|     overseas inflow|        -|         -|   Seoul|
|1000011|              -|       18| 6.565996913733051E7|false|contact with patient|        -|         -|   Seoul|
|1000012|              -|       24|2.648912212984347E10|false|                 etc|        -|         -|   Seoul|
|1100001|     Dongnae-gu|       39|8.659340042399374...| true|       Onchun Church| 35.21628|  129.0771|   Busan|
|1100002|from other city|       12|  162754.79141900392| true|  Shincheonji Church|        -|         -|   Busan|
|1100003|     Suyeong-gu|        5|   148.4131591025766| true|Suyeong-gu Kinder...| 35.16708|  129.1124|   Busan|
|1100004|    Haeundae-gu|        6|   403.4287934927351| true|Haeundae-gu Catho...| 35.20599|  129.1256|   Busan|
|1100005|         Jin-gu|        4|  54.598150033144236| true|      Jin-gu Academy| 35.17371|  129.0633|   Busan|
|1100006|from other city|        1|   2.718281828459045| true|Cheongdo Daenam H...|        -|         -|   Busan|
|1100007|              -|       25|7.200489933738588E10|false|     overseas inflow|        -|         -|   Busan|
|1100008|              -|       18| 6.565996913733051E7|false|contact with patient|        -|         -|   Busan|
+-------+--------------+---------+--------------------+-----+--------------------+---------+----------+--------+
only showing top 20 rows
```

Using Pandas UDF

This functionality was introduced in the Spark version 2.3.1. And this allows you to use pandas functionality with Spark. I generally use it when I have to run a groupBy operation on a Spark dataframe or whenever I need to create rolling features and want to use Pandas rolling functions/window functions rather than Spark window functions which we will go through later in this post.

dataframe in turn from this function.

The only complexity here is that we have to provide a schema for the output Dataframe. We can use the original schema of a dataframe to create the outSchema.

```
cases.printSchema()
```

```
cases.printSchema()
```

```
root
 |-- case_id: integer (nullable = true)
 |-- province: string (nullable = true)
 |-- city: string (nullable = true)
 |-- group: boolean (nullable = true)
 |-- infection_case: string (nullable = true)
 |-- confirmed: integer (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
```

Here I am using Pandas UDF to get normalized confirmed cases grouped by infection_case. The main advantage here is that I get to work with pandas dataframes in Spark.

```
from pyspark.sql.types import IntegerType, StringType, DoubleType, BooleanType
from pyspark.sql.types import StructType, StructField

# Declare the schema for the output of our function

outSchema = StructType([StructField('case_id',IntegerType(),True),
                        StructField('province',StringType(),True),
                        StructField('city',StringType(),True),
                        StructField('group',BooleanType(),True),
                        StructField('infection_case',StringType(),True),
                        StructField('confirmed',IntegerType(),True),
                        StructField('latitude',StringType(),True),
                        StructField('longitude',StringType(),True),
                        StructField('normalized_confirmed',DoubleType(),True)
                       ])
# decorate our function with pandas_udf decorator
@F.pandas_udf(outSchema, F.PandasUDFType.GROUPED_MAP)
def subtract_mean(pdf):
```

```
22        pdf['normalized_confirmed'] = v
23        return pdf
24
25    confirmed_groupwise_normalization = cases.groupby("infection_case").apply(subtract_mean)
26
27    confirmed_groupwise_normalization.limit(10).toPandas()
```

pandas_udf.py hosted with ♡ by GitHub view raw

	case_id	province	city	group	infection_case	confirmed	latitude	longitude	normalized_confirmed
0	1000005	Seoul	Eunpyeong-gu	True	Eunpyeong St. Mary's Hospital	14	37.63369	126.9165	0.000000
1	1700001	Sejong	Sejong	True	Ministry of Oceans and Fisheries	30	36.504713	127.265172	0.000000
2	2000005	Gyeonggi-do	Seongnam-si	True	Bundang Jesaeng Hospital	22	37.38833	127.1218	0.000000
3	6000005	Gyeongsangbuk-do	Chilgok-gun	True	Milal Shelter	36	36.0581	128.4941	0.000000
4	1000001	Seoul	Yongsan-gu	True	Itaewon Clubs	72	37.538621	126.992652	45.000000
5	2000010	Gyeonggi-do	from other city	True	Itaewon Clubs	8	-	-	-19.000000
6	7000004	Jeju-do	from other city	True	Itaewon Clubs	1	-	-	-26.000000
7	3000003	Gangwon-do	Wonju-si	True	Wonju-si Apartments	3	37.342762	127.983815	0.000000
8	1000003	Seoul	Dongdaemun-gu	True	Dongan Church	20	37.592888	127.056766	0.000000
9	1000010	Seoul	-	False	overseas inflow	321	-	-	275.294118

5. Spark Window Functions

Window functions may make a whole blog post in itself. Here I will talk about some of the most important window functions available in spark.

For this, I will also use one more data CSV, which has dates present as that will help with understanding Window functions much better. I will use the TimeProvince dataframe which contains daily case information for each province.

```
timeprovince = spark.read.load("/home/rahul/projects/sparkdf/coronavirusdataset/TimeProvince.csv",format="csv", \
                    sep=",", inferSchema="true", header="true")
timeprovince.show()
```

```
+-------------------+----+----------------+---------+--------+--------+
|               date|time|        province|confirmed|released|deceased|
+-------------------+----+----------------+---------+--------+--------+
|2020-01-20 00:00:00|  16|           Seoul|        0|       0|       0|
|2020-01-20 00:00:00|  16|           Busan|        0|       0|       0|
|2020-01-20 00:00:00|  16|           Daegu|        0|       0|       0|
|2020-01-20 00:00:00|  16|         Incheon|        1|       0|       0|
|2020-01-20 00:00:00|  16|         Gwangju|        0|       0|       0|
|2020-01-20 00:00:00|  16|         Daejeon|        0|       0|       0|
|2020-01-20 00:00:00|  16|           Ulsan|        0|       0|       0|
```

```
|2020-01-20 00:00:00|  16|     Jeollanam-do|        0|       0|       0|
|2020-01-20 00:00:00|  16| Gyeongsangbuk-do|        0|       0|       0|
|2020-01-20 00:00:00|  16| Gyeongsangnam-do|        0|       0|       0|
|2020-01-20 00:00:00|  16|          Jeju-do|        0|       0|       0|
|2020-01-21 00:00:00|  16|            Seoul|        0|       0|       0|
|2020-01-21 00:00:00|  16|            Busan|        0|       0|       0|
|2020-01-21 00:00:00|  16|            Daegu|        0|       0|       0|
+-------------------+----+-----------------+---------+--------+--------+
only showing top 20 rows
```

Ranking

You can get rank as well as dense_rank on a group using this function. For example, you may want to have a column in your cases table that provides the rank of infection_case based on the number of infection_case in a province. We can do this by:

```
from pyspark.sql.window import Window

windowSpec =
Window().partitionBy(['province']).orderBy(F.desc('confirmed'))

cases.withColumn("rank",F.rank().over(windowSpec)).show()
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy(F.desc('confirmed'))
cases.withColumn("rank",F.rank().over(windowSpec)).show()
```

```
+-------+----------------+--------------+-----+-------------------+---------+---------+----------+----+
|case_id|        province|          city|group|     infection_case|confirmed| latitude| longitude|rank|
+-------+----------------+--------------+-----+-------------------+---------+---------+----------+----+
|1700001|          Sejong|        Sejong| true|Ministry of Ocean...|       30|36.504713|127.265172|   1|
|1700002|          Sejong|        Sejong| true|gym facility in S...|        8| 36.48025|   127.289|   2|
|1700004|          Sejong|            -|false|    overseas inflow|        3|        -|         -|   3|
|1700005|          Sejong|            -|false|contact with patient|        3|        -|         -|   3|
|1700003|          Sejong|from other city| true|  Shincheonji Church|        1|        -|         -|   5|
|1700006|          Sejong|            -|false|                etc|        1|        -|         -|   5|
|1600001|           Ulsan|from other city| true|  Shincheonji Church|       16|        -|         -|   1|
|1600002|           Ulsan|            -|false|    overseas inflow|       15|        -|         -|   2|
|1600004|           Ulsan|            -|false|                etc|        7|        -|         -|   3|
|1600003|           Ulsan|            -|false|contact with patient|        4|        -|         -|   4|
|4000002|Chungcheongbuk-do|     Goesan-gun| true|Goesan-gun Jangye...|       11| 36.82422|  127.9552|   1|
|4000005|Chungcheongbuk-do|            -|false|                etc|        8|        -|         -|   2|
|4000003|Chungcheongbuk-do|            -|false|    overseas inflow|        7|        -|         -|   3|
|4000001|Chungcheongbuk-do|from other city| true|  Shincheonji Church|        6|        -|         -|   4|
|4000004|Chungcheongbuk-do|            -|false|contact with patient|        6|        -|         -|   4|
|3000001|       Gangwon-do|from other city| true|  Shincheonji Church|       17|        -|         -|   1|
|3000004|       Gangwon-do|            -|false|    overseas inflow|       14|        -|         -|   2|
|3000002|       Gangwon-do|from other city| true|Uijeongbu St. Mar...|       10|        -|         -|   3|
|3000006|       Gangwon-do|            -|false|                etc|        7|        -|         -|   4|
|3000003|       Gangwon-do|      Wonju-si| true| Wonju-si Apartments|        3|37.342762|127.983815|   5|
+-------+----------------+--------------+-----+-------------------+---------+---------+----------+----+
only showing top 20 rows
```

Lag Variables

create such features using the lag function with window functions. Here I am trying to get the confirmed cases 7 days before. I am filtering to show the results as the first few days of corona cases were zeros. You can see here that the lag_7 day feature is shifted by 7 days.

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy('date')
timeprovinceWithLag =
timeprovince.withColumn("lag_7",F.lag("confirmed",
7).over(windowSpec))

timeprovinceWithLag.filter(timeprovinceWithLag.date>'2020-03-
10').show()
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy('date')
timeprovinceWithLag = timeprovince.withColumn("lag_7",F.lag("confirmed", 7).over(windowSpec))

timeprovinceWithLag.filter(timeprovinceWithLag.date>'2020-03-10').show()
```

```
+-------------------+----+--------+---------+--------+--------+-----+
|               date|time|province|confirmed|released|deceased|lag_7|
+-------------------+----+--------+---------+--------+--------+-----+
|2020-03-10 00:00:00|   0|  Sejong|        8|       0|       0|    1|
|2020-03-11 00:00:00|   0|  Sejong|       10|       0|       0|    1|
|2020-03-12 00:00:00|   0|  Sejong|       15|       0|       0|    1|
|2020-03-13 00:00:00|   0|  Sejong|       32|       0|       0|    1|
|2020-03-14 00:00:00|   0|  Sejong|       38|       0|       0|    2|
|2020-03-15 00:00:00|   0|  Sejong|       39|       0|       0|    3|
|2020-03-16 00:00:00|   0|  Sejong|       40|       0|       0|    6|
|2020-03-17 00:00:00|   0|  Sejong|       40|       0|       0|    8|
|2020-03-18 00:00:00|   0|  Sejong|       41|       0|       0|   10|
|2020-03-19 00:00:00|   0|  Sejong|       41|       0|       0|   15|
|2020-03-20 00:00:00|   0|  Sejong|       41|       0|       0|   32|
|2020-03-21 00:00:00|   0|  Sejong|       41|       2|       0|   38|
|2020-03-22 00:00:00|   0|  Sejong|       41|       3|       0|   39|
|2020-03-23 00:00:00|   0|  Sejong|       42|       3|       0|   40|
|2020-03-24 00:00:00|   0|  Sejong|       42|       3|       0|   40|
|2020-03-25 00:00:00|   0|  Sejong|       44|       3|       0|   41|
|2020-03-26 00:00:00|   0|  Sejong|       44|       8|       0|   41|
|2020-03-27 00:00:00|   0|  Sejong|       44|       9|       0|   41|
|2020-03-28 00:00:00|   0|  Sejong|       44|       9|       0|   41|
|2020-03-29 00:00:00|   0|  Sejong|       46|      11|       0|   41|
+-------------------+----+--------+---------+--------+--------+-----+
only showing top 20 rows
```

Rolling Aggregations

Sometimes it helps to provide rolling averages to our models. For example, we might want to have a rolling 7-day sales sum/mean as a feature for our sales regression model. Let us calculate the rolling mean of confirmed cases for the last 7 days here. This is what a lot of the people are already doing with this dataset to see the real trends.

```
windowSpec =
Window().partitionBy(['province']).orderBy('date').rowsBetween(-6,0)

timeprovinceWithRoll =
timeprovince.withColumn("roll_7_confirmed",F.mean("confirmed").over(w
indowSpec))

timeprovinceWithRoll.filter(timeprovinceWithLag.date>'2020-03-
10').show()
```

```
from pyspark.sql.window import Window

windowSpec = Window().partitionBy(['province']).orderBy('date').rowsBetween(-6,0)
timeprovinceWithRoll = timeprovince.withColumn("roll_7_confirmed",F.mean("confirmed").over(windowSpec))
timeprovinceWithRoll.filter(timeprovinceWithLag.date>'2020-03-10').show()
```

```
+-------------------+----+--------+---------+--------+--------+------------------+
|               date|time|province|confirmed|released|deceased|  roll_7_confirmed|
+-------------------+----+--------+---------+--------+--------+------------------+
|2020-03-10 00:00:00|   0|  Sejong|        8|       0|       0| 3.142857142857143|
|2020-03-11 00:00:00|   0|  Sejong|       10|       0|       0| 4.428571428571429|
|2020-03-12 00:00:00|   0|  Sejong|       15|       0|       0| 6.428571428571429|
|2020-03-13 00:00:00|   0|  Sejong|       32|       0|       0|10.857142857142858|
|2020-03-14 00:00:00|   0|  Sejong|       38|       0|       0|              16.0|
|2020-03-15 00:00:00|   0|  Sejong|       39|       0|       0|21.142857142857142|
|2020-03-16 00:00:00|   0|  Sejong|       40|       0|       0|              26.0|
|2020-03-17 00:00:00|   0|  Sejong|       40|       0|       0|30.571428571428573|
|2020-03-18 00:00:00|   0|  Sejong|       41|       0|       0|              35.0|
|2020-03-19 00:00:00|   0|  Sejong|       41|       0|       0|38.714285714285715|
|2020-03-20 00:00:00|   0|  Sejong|       41|       0|       0|              40.0|
|2020-03-21 00:00:00|   0|  Sejong|       41|       2|       0| 40.42857142857143|
|2020-03-22 00:00:00|   0|  Sejong|       41|       3|       0|40.714285714285715|
|2020-03-23 00:00:00|   0|  Sejong|       42|       3|       0|              41.0|
|2020-03-24 00:00:00|   0|  Sejong|       42|       3|       0|41.285714285714285|
|2020-03-25 00:00:00|   0|  Sejong|       44|       3|       0|41.714285714285715|
|2020-03-26 00:00:00|   0|  Sejong|       44|       8|       0|42.142857142857146|
|2020-03-27 00:00:00|   0|  Sejong|       44|       9|       0| 42.57142857142857|
|2020-03-28 00:00:00|   0|  Sejong|       44|       9|       0|              43.0|
|2020-03-29 00:00:00|   0|  Sejong|       46|      11|       0|43.714285714285715|
+-------------------+----+--------+---------+--------+--------+------------------+
only showing top 20 rows
```

There are a few things here to understand. First is the **`rowsBetween(-6,0)`** function that we are using here. This function has a form of `rowsBetween(start,end)` with both start and end inclusive. Using this we only look at the past 7 days in a particular window including the current_day. Here 0 specifies the current_row and -6 specifies the seventh row previous to current_row. Remember we count starting from 0.

So to get `roll_7_confirmed` for date `2020-03-22` we look at the confirmed cases for dates `2020-03-22 to 2020-03-16` and take their mean.

If we had used **`rowsBetween(-7,-1)`** we would just have looked at past 7 days of data and not the current_day.

current_row to get running totals. I am calculating cumulative_confirmed here.

```
from pyspark.sql.window import Window

windowSpec =
Window().partitionBy(['province']).orderBy('date').rowsBetween(Window
.unboundedPreceding,Window.currentRow)
timeprovinceWithRoll =
timeprovince.withColumn("cumulative_confirmed",F.sum("confirmed").ove
r(windowSpec))
timeprovinceWithRoll.filter(timeprovinceWithLag.date>'2020-03-
10').show()
```

```
from pyspark.sql.window import Window

windowSpec = Window().partitionBy(['province']).orderBy('date').rowsBetween(\
            Window.unboundedPreceding,Window.currentRow)
timeprovinceWithRoll = timeprovince.withColumn("cumulative_confirmed",F.sum("confirmed").over(windowSpec))
timeprovinceWithRoll.filter(timeprovinceWithLag.date>'2020-03-10').show()
```

```
+-------------------+----+--------+---------+--------+--------+--------------------+
|               date|time|province|confirmed|released|deceased|cumulative_confirmed|
+-------------------+----+--------+---------+--------+--------+--------------------+
|2020-03-10 00:00:00|   0|  Sejong|        8|       0|       0|                  33|
|2020-03-11 00:00:00|   0|  Sejong|       10|       0|       0|                  43|
|2020-03-12 00:00:00|   0|  Sejong|       15|       0|       0|                  58|
|2020-03-13 00:00:00|   0|  Sejong|       32|       0|       0|                  90|
|2020-03-14 00:00:00|   0|  Sejong|       38|       0|       0|                 128|
|2020-03-15 00:00:00|   0|  Sejong|       39|       0|       0|                 167|
|2020-03-16 00:00:00|   0|  Sejong|       40|       0|       0|                 207|
|2020-03-17 00:00:00|   0|  Sejong|       40|       0|       0|                 247|
|2020-03-18 00:00:00|   0|  Sejong|       41|       0|       0|                 288|
|2020-03-19 00:00:00|   0|  Sejong|       41|       0|       0|                 329|
|2020-03-20 00:00:00|   0|  Sejong|       41|       0|       0|                 370|
|2020-03-21 00:00:00|   0|  Sejong|       41|       2|       0|                 411|
|2020-03-22 00:00:00|   0|  Sejong|       41|       3|       0|                 452|
|2020-03-23 00:00:00|   0|  Sejong|       42|       3|       0|                 494|
|2020-03-24 00:00:00|   0|  Sejong|       42|       3|       0|                 536|
|2020-03-25 00:00:00|   0|  Sejong|       44|       3|       0|                 580|
|2020-03-26 00:00:00|   0|  Sejong|       44|       8|       0|                 624|
|2020-03-27 00:00:00|   0|  Sejong|       44|       9|       0|                 668|
|2020-03-28 00:00:00|   0|  Sejong|       44|       9|       0|                 712|
|2020-03-29 00:00:00|   0|  Sejong|       46|      11|       0|                 758|
+-------------------+----+--------+---------+--------+--------+--------------------+
only showing top 20 rows
```

6. Pivot Dataframes

Sometimes we may need to have the dataframe in flat format. This happens frequently in movie data where we may want to show genres as columns instead of rows. We can

```
pivotedTimeprovince =
timeprovince.groupBy('date').pivot('province').agg(F.sum('confirmed')
.alias('confirmed') , F.sum('released').alias('released'))

pivotedTimeprovince.limit(10).toPandas()
```

```
pivotedTimeprovince = timeprovince.groupBy('date').pivot('province') \
                      .agg(F.sum('confirmed').alias('confirmed') , F.sum('released').alias('released'))
pivotedTimeprovince.limit(10).toPandas()
```

	date	Busan_confirmed	Busan_released	Chungcheongbuk-do_confirmed	Chungcheongbuk-do_released	Chungcheongnam-do_confirmed	Chungcheongnam-do_released	Daegu_confirmed	Daegu_released
0	2020-02-02	0	0	0	0	0	0	0	0
1	2020-04-19	130	115	45	38	141	122	6832	5743
2	2020-03-04	92	2	11	0	82	0	4007	11
3	2020-01-31	0	0	0	0	0	0	0	0
4	2020-04-17	130	112	45	37	139	115	6827	5626
5	2020-02-22	11	0	3	0	1	0	193	0
6	2020-04-11	126	100	45	28	138	106	6814	5271
7	2020-05-11	141	125	52	42	143	134	6861	6318
8	2020-03-16	107	53	31	6	115	12	6066	734
9	2020-01-20	0	0	0	0	0	0	0	0

10 rows × 35 columns

One thing to note here is that we need to provide an aggregation always with the pivot function even if the data has a single row for a date.

7. Unpivot/Stack Dataframes

This is just the opposite of the pivot. Given a pivoted dataframe like above, can we go back to the original?

Yes, we can. But the way is not that straightforward. For one we will need to replace - with _ in the column names as it interferes with what we are about to do. We can simply

```
newColnames = [x.replace("-","_") for x in
pivotedTimeprovince.columns]

pivotedTimeprovince = pivotedTimeprovince.toDF(*newColnames)
```

Now we will need to create an expression which looks like the below:

```
"stack(34, 'Busan_confirmed' , Busan_confirmed,'Busan_released' ,
Busan_released,'Chungcheongbuk_do_confirmed' ,

.
.
.

'Seoul_released' , Seoul_released,'Ulsan_confirmed' ,
Ulsan_confirmed,'Ulsan_released' , Ulsan_released) as (Type,Value)"
```

The general format is as follows:

```
"stack(<cnt of columns you want to put in one column>,
'firstcolname', firstcolname , 'secondcolname' ,secondcolname ......)
as (Type, Value)"
```

It may seem daunting, but we can create such an expression using our programming skills.

```
expression = ""
cnt=0
for column in pivotedTimeprovince.columns:
    if column!='date':
        cnt +=1
        expression += f"'{column}' , {column},"

expression = f"stack({cnt}, {expression[:-1]}) as (Type,Value)"
```

```
unpivotedTimeprovince =
pivotedTimeprovince.select('date',F.expr(exprs))
```

```
unpivotedTimeprovince = pivotedTimeprovince.select('date',F.expr(exprs))
unpivotedTimeprovince.show()
```

```
+-------------------+--------------------+-----+
|               date|                Type|Value|
+-------------------+--------------------+-----+
|2020-02-02 00:00:00|     Busan_confirmed|    0|
|2020-02-02 00:00:00|      Busan_released|    0|
|2020-02-02 00:00:00|Chungcheongbuk_do...|    0|
|2020-02-02 00:00:00|Chungcheongbuk_do...|    0|
|2020-02-02 00:00:00|Chungcheongnam_do...|    0|
|2020-02-02 00:00:00|Chungcheongnam_do...|    0|
|2020-02-02 00:00:00|     Daegu_confirmed|    0|
|2020-02-02 00:00:00|      Daegu_released|    0|
|2020-02-02 00:00:00|   Daejeon_confirmed|    0|
|2020-02-02 00:00:00|    Daejeon_released|    0|
|2020-02-02 00:00:00|Gangwon_do_confirmed|    0|
|2020-02-02 00:00:00| Gangwon_do_released|    0|
|2020-02-02 00:00:00|   Gwangju_confirmed|    0|
|2020-02-02 00:00:00|    Gwangju_released|    0|
|2020-02-02 00:00:00|Gyeonggi_do_confi...|    8|
|2020-02-02 00:00:00|Gyeonggi_do_released|    0|
|2020-02-02 00:00:00|Gyeongsangbuk_do_...|    0|
|2020-02-02 00:00:00|Gyeongsangbuk_do_...|    0|
|2020-02-02 00:00:00|Gyeongsangnam_do_...|    0|
|2020-02-02 00:00:00|Gyeongsangnam_do_...|    0|
+-------------------+--------------------+-----+
only showing top 20 rows
```

And voila! we have got our dataframe in a vertical format. There are quite a few column creations, filters, and join operations needed to get exactly the same format as before, but I will not get into those.

8. Salting

Sometimes it might happen that a lot of data goes to a single executor since the same key is assigned for a lot of rows in our data. Salting is another way that helps you to manage data skewness.

So assuming we want to do the sum operation when we have skewed keys. We can start by creating the Salted Key and then doing a double aggregation on that key as the sum of a sum still equals sum. To understand this assume we need the sum of confirmed

1. Create a Salting Key

We first create a salting key using a concatenation of infection_case column and a random_number between 0 to 9. In case your key is even more skewed, you can split it in even more than 10 parts.

```
cases = cases.withColumn("salt_key",
F.concat(F.col("infection_case"), F.lit("_"),
F.monotonically_increasing_id() % 10))
```

This is how the table looks after the operation:

```
cases.show()
```

```
+-------+--------+--------------+-----+--------------------+---------+---------+----------+--------------------+
|case_id|province|          city|group|      infection_case|confirmed| latitude| longitude|            salt_key|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+--------------------+
|1000001|   Seoul|    Yongsan-gu| true|       Itaewon Clubs|       72|37.538621|126.992652|     Itaewon Clubs_0|
|1000002|   Seoul|       Guro-gu| true| Guro-gu Call Center|       98|37.508163|126.884387|Guro-gu Call Cent...|
|1000003|   Seoul| Dongdaemun-gu| true|       Dongan Church|       20|37.592888|127.056766|     Dongan Church_2|
|1000004|   Seoul|       Guro-gu| true|Manmin Central Ch...|       41|37.481059|126.894343|Manmin Central Ch...|
|1000005|   Seoul|  Eunpyeong-gu| true|Eunpyeong St. Mar...|       14| 37.63369|  126.9165|Eunpyeong St. Mar...|
|1000006|   Seoul|  Seongdong-gu| true|    Seongdong-gu APT|       13| 37.55713|  127.0403|  Seongdong-gu APT_5|
|1000007|   Seoul|     Jongno-gu| true|Jongno Community ...|       10| 37.57681|   127.006|Jongno Community ...|
|1000008|   Seoul|       Jung-gu| true|Jung-gu Fashion C...|        7|37.562405|126.984377|Jung-gu Fashion C...|
|1000009|   Seoul|from other city| true|  Shincheonji Church|        8|        -|         -|Shincheonji Church_8|
|1000010|   Seoul|             -|false|     overseas inflow|      321|        -|         -|   overseas inflow_9|
|1000011|   Seoul|             -|false|contact with patient|       18|        -|         -|contact with pati...|
|1000012|   Seoul|             -|false|                 etc|       24|        -|         -|               etc_1|
|1100001|   Busan|    Dongnae-gu| true|       Onchun Church|       39| 35.21628|  129.0771|     Onchun Church_2|
|1100002|   Busan|from other city| true|  Shincheonji Church|       12|        -|         -|Shincheonji Church_3|
|1100003|   Busan|    Suyeong-gu| true|Suyeong-gu Kinder...|        5| 35.16708|  129.1124|Suyeong-gu Kinder...|
|1100004|   Busan|   Haeundae-gu| true|Haeundae-gu Catho...|        6| 35.20599|  129.1256|Haeundae-gu Catho...|
|1100005|   Busan|        Jin-gu| true|      Jin-gu Academy|        4| 35.17371|  129.0633|    Jin-gu Academy_6|
|1100006|   Busan|from other city| true|Cheongdo Daenam H...|        1|        -|         -|Cheongdo Daenam H...|
|1100007|   Busan|             -|false|     overseas inflow|       25|        -|         -|   overseas inflow_8|
|1100008|   Busan|             -|false|contact with patient|       18|        -|         -|contact with pati...|
+-------+--------+--------------+-----+--------------------+---------+---------+----------+--------------------+
only showing top 20 rows
```

2. First Groupby on salt key

```
cases_temp =
cases.groupBy(["infection_case","salt_key"]).agg(F.sum("confirmed")).
show()
```

```
|       Onchun Church|     Onchun Church_2|           39|
|Bundang Jesaeng H...|Bundang Jesaeng H...|           22|
|Cheongdo Daenam H...|Cheongdo Daenam H...|            1|
|Seosan-si Laboratory|Seosan-si Laborat...|            9|
|Bonghwa Pureun Nu...|Bonghwa Pureun Nu...|           68|
|Seosan-si Laboratory|Seosan-si Laborat...|            2|
|       Onchun Church|     Onchun Church_7|            2|
| Guro-gu Call Center|Guro-gu Call Cent...|           48|
|Goesan-gun Jangye...|Goesan-gun Jangye...|           11|
|gym facility in C...|gym facility in C...|          103|
|contact with patient|contact with pati...|           58|
|contact with patient|contact with pati...|          929|
|Geochang-gun Woon...|Geochang-gun Woon...|            8|
|Cheongdo Daenam H...|Cheongdo Daenam H...|          120|
|                 etc|               etc_1|           24|
|Daesil Convalesce...|Daesil Convalesce...|          100|
|     overseas inflow|   overseas inflow_2|           22|
|Jung-gu Fashion C...|Jung-gu Fashion C...|            7|
|Second Mi-Ju Hosp...|Second Mi-Ju Hosp...|          196|
|     overseas inflow|   overseas inflow_8|           78|
+--------------------+--------------------+-------------+
only showing top 20 rows
```

3. Second Group On the original Key

```
cases_answer = cases_temp.groupBy(["infection_case"]).agg(F.sum("salt_confirmed").alias("final_confirmed"))
cases_answer.show()
```

```
+--------------------+---------------+
|      infection_case|final_confirmed|
+--------------------+---------------+
|Eunpyeong St. Mar...|             14|
|Ministry of Ocean...|             30|
|Bundang Jesaeng H...|             22|
|       Milal Shelter|             36|
|       Itaewon Clubs|             81|
| Wonju-si Apartments|              3|
|       Dongan Church|             20|
|     overseas inflow|            777|
|Gyeongsan Cham Jo...|             16|
|Hansarang Convale...|            128|
|Second Mi-Ju Hosp...|            196|
|Gyeongsan Seorin ...|             66|
|Changnyeong Coin ...|              7|
|      Jin-gu Academy|              4|
|Korea Forest Engi...|              3|
|Goesan-gun Jangye...|             11|
|         Wings Tower|             10|
|     Geochang Church|             10|
|     Fatima Hospital|             37|
|Bonghwa Pureun Nu...|             68|
+--------------------+---------------+
only showing top 20 rows
```

Here we saw how the sum of sum can be used to get the final sum. You can also make use of facts like:

- min of min is min
- max of max is max
- sum of count is count

Some More Tips and Tricks

Caching

Spark works on the lazy execution principle. What that means is that nothing really gets executed until you use an action function like the `.count()` on a dataframe. And if you do a `.count` function, it generally helps to cache at this step. So I have made it a point to cache() my dataframes whenever I do a `.count()` operation.

```
df.cache().count()
```

Save and Load from an intermediate step

```
df.write.parquet("data/df.parquet")
df.unpersist()
spark.read.load("data/df.parquet")
```

When you work with Spark you will frequently run with memory and storage issues. While in some cases such issues might be resolved using techniques like broadcasting, salting or cache, sometimes just interrupting the workflow and saving and reloading the whole dataframe at a crucial step has helped me a lot. This helps spark to let go of a lot of memory that gets utilized for storing intermediate shuffle data and unused caches.

Repartitioning

You might want to repartition your data if you feel your data has been skewed while working with all the transformations and joins. The simplest way to do it is by using:

```
df = df.repartition(1000)
```

columns to repartition using:

```
df = df.repartition('cola', 'colb','colc','cold')
```

You can get the number of partitions in a data frame using:

```
df.rdd.getNumPartitions()
```

You can also check out the distribution of records in a partition by using the `glom` function. This helps in understanding the skew in the data that happens while working with various transformations.

```
df.glom().map(len).collect()
```

Reading Parquet File in Local

Sometimes you might want to read the parquet files in a system where Spark is not available. In such cases, I normally use the below code:

```
from glob import glob
def load_df_from_parquet(parquet_directory):
   df = pd.DataFrame()
   for file in glob(f"{parquet_directory}/*"):
      df = pd.concat([df,pd.read_parquet(file)])
   return df
```

Conclusion

Source: Pixabay

This was a big post and congratulations on you reaching the end. These are the most common functionalities I end up using in my day to day job.

Hopefully, I've covered the Dataframe basics well enough to pique your interest and help you get started with Spark. If you want to learn more about how Spark Started or RDD basics take a look at this post

You can find all the code at this GitHub repository where I keep code for all my posts.

Continue Learning

Also, if you want to learn more about Spark and Spark DataFrames, I would like to call out these excellent courses on Big Data Essentials: HDFS, MapReduce and Spark RDD and Big Data Analysis: Hive, Spark SQL, DataFrames and GraphFrames by Yandex on Coursera.

I am going to be writing more of such posts in the future too. Let me know what you think about the series. Follow me up at **Medium** or Subscribe to my **blog** to be informed about them. As always, I welcome feedback and constructive criticism and can be reached on Twitter @mlwhiz.

Also, a small disclaimer — There might be some affiliate links in this post to relevant resources, as sharing knowledge is never a bad idea.