

University of Salford, MSc Data Science

Module: Big Data Tools and Techniques

Date: Trimester 2, 2024-2025

Session: Workshop Week 4

Topic: DataFrames in Spark

Tools: Databricks Community Edition

Instructors: Nathan Topping, Dr Taha Mansouri, and Dr Kaveh Kiani

Objectives:

After completing this workshop, you will be able to:

- Creating different DataFrames
- Running various DataFrame operations
- Creating a rerunnable notebook

Table of Contents

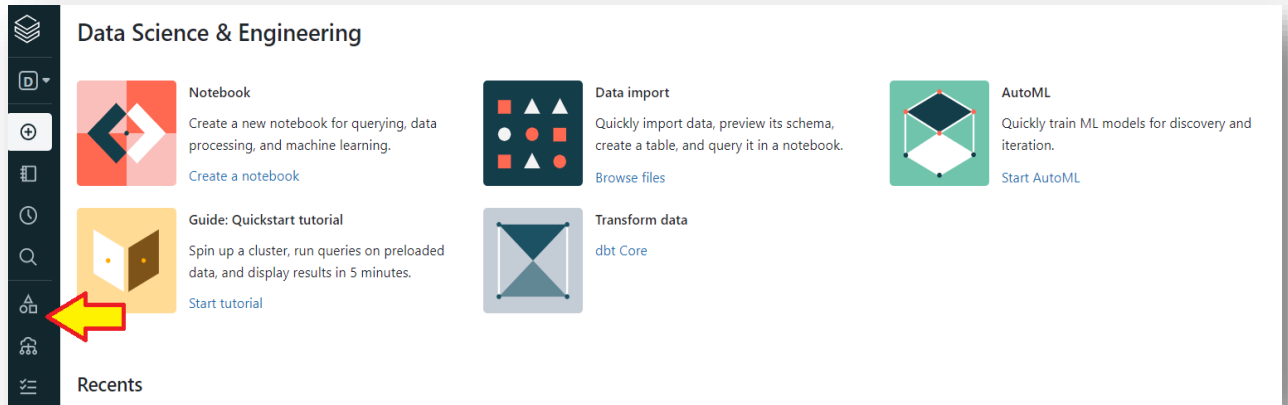
Part 1: Fire up the Databricks workspace	3
Part 2: Create a new Python notebook.....	3
Part 3: Uploading data, creating and manipulating DataFrames	5
1) Put webpage data on Databricks	5
2) View beginning lines of the file to plan the next step.....	6
3) Different ways of creating a DataFrame.....	7
4) Checking schemas of the created DataFrames and unifying DataFrames.	12
5) Using Pair RDDs to manipulate data and create a new DataFrame	15
Part 4: Making your notebook rerunnable: A challenge for you.....	19
References and Resources:.....	20

Part 1: Fire up the Databricks workspace

1. Log in to your Databricks Community Edition account

<https://community.cloud.databricks.com/login.html?nextUrl=%2F>

2. You will need to create a cluster to start your analysis – this gives you access to a machine to use. Click on “Compute” on the main page.



3. Click on “Create Compute” and type in a new name for the cluster, any name that you like.
4. Select the most recent runtime (currently, runtime 12.1 (Scala 2.12, Spark 3.3.1)).

Note that it will take a few minutes to create the cluster. After some time, the green circle next to the cluster name will gain a green tick meaning the cluster has successfully started up.

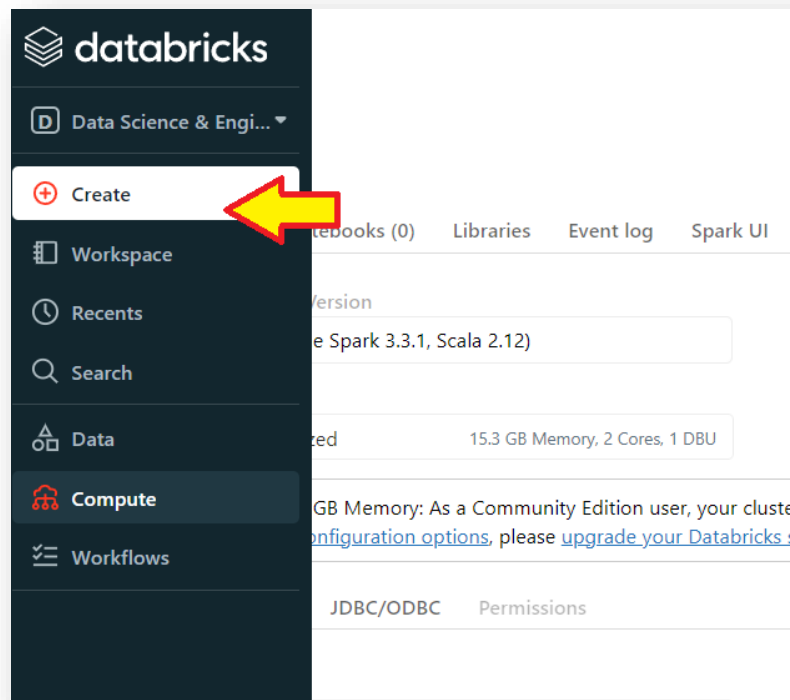
Part 2: Create a new Python notebook

Databricks programs are written in Notebooks. A notebook is a collection of runnable cells which contain your commands. Look at

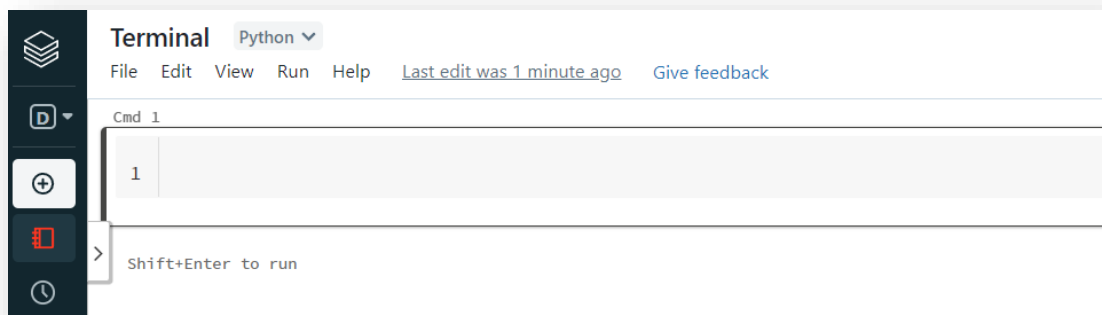
<https://docs.databricks.com/notebooks/notebooks-use.html>

for an overview of how to use a Databricks notebook.

1. Click on the button “Create” and select “Notebook”.



2. Name your notebook, e.g. “DataFrame week 4”, leave (or select) the language to be “Python” and the cluster should be your currently created cluster. This will give you access to a notebook which looks something like this:



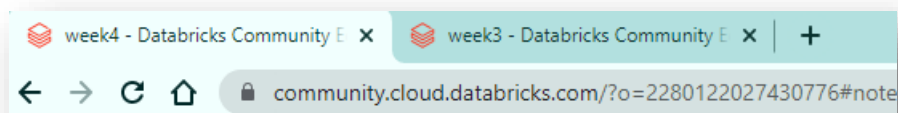
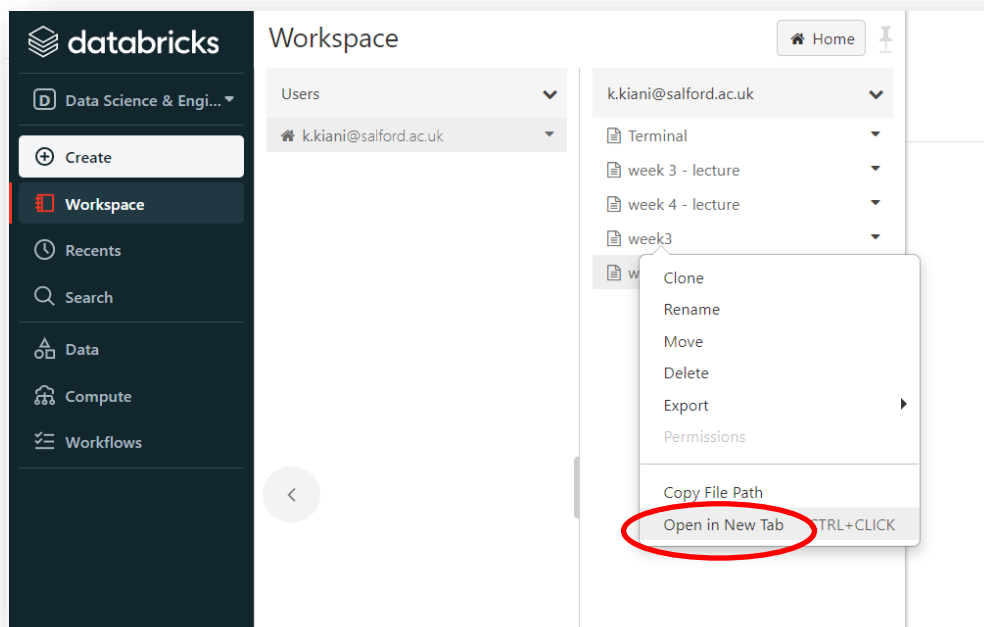
Part 3: Uploading data, creating and manipulating DataFrames

In this lab, you will use Spark DataFrames to process some data. During the lab if you feel you need to retrieve what you have learnt during the DataFrame lecture go back and review the lecture slides.

1) Put webpage data on Databricks

This lab uses the zip archive **webpage.zip** from Blackboard. You should either use the reusable notebook you wrote in a previous lab to unzip this and place the resulting webpage directory in DBFS's `/FileStore/tables` or you should follow the steps from a previous lab to do this manually.

Your week 3 notebook has been saved in your databricks account. If you want to review what you did to unzip data, follow the following image to open week 4 and week 3 notebooks in 2 browser tabs. This will help you to write unzip command in the week 4 notebook easily.



After unzipping you should have the webpage directory in DBFS like:

```
Cmd 7

1  dbutils.fs.ls("/FileStore/tables/" + fileroot)

Out[7]: [FileInfo(path='dbfs:/FileStore/tables/webpage/part-m-00000', name='part-m-00000', size=751, modificationTime=1675601519000),
FileInfo(path='dbfs:/FileStore/tables/webpage/part-m-00001', name='part-m-00001', size=874, modificationTime=1675601518000),
FileInfo(path='dbfs:/FileStore/tables/webpage/part-m-00002', name='part-m-00002', size=804, modificationTime=1675601520000),
FileInfo(path='dbfs:/FileStore/tables/webpage/part-m-00003', name='part-m-00003', size=774, modificationTime=1675601519000)]
```

Unzipped webpage directory has 4 files “part-m-00000” to “part-m-00003”.

But why file names are like “part-m-00000”?

The output files are by default named part-**x**-yyyyy where:

- 1) x is either ‘**m**’ or ‘**r**’, depending on whether the job was a map only job, or reduce
- 2) yyyyy is the Mapper, or Reducer task number

So, if a job has 4 mappers, the generated files will be like: part-m-00000 to part-m-00003, one for each mapper task.

2) View beginning lines of the file to plan the next step

Let’s see few lines of one of the files:

```
Cmd 8

1  dbutils.fs.head("/FileStore/tables/webpage/part-m-00000")

Out[17]: '1\tsorrento_f00l_sales.html\ttheme.css,code.js,sorrento_f00l.jpg\n2\ttitanic_2100_sales.html\ttheme.css,code.js,titanic_2100.jpg\n3\tmeetoo_3.0_sales.html\ttheme.css,code.js,meetoo_3.0.jpg\n4\tmeetoo_3.1_sales.html\ttheme.css,code.js,meetoo_3.1.jpg\n5\tifruit_1_sales.html\ttheme.css,code.js,ifruit_1.jpg\n6\tifruit_3_sales.html\ttheme.css,code.js,ifruit_3.jpg\n7\tifruit_2_sales.html\ttheme.css,code.js,ifruit_2.jpg\n8\tifruit_5_sales.html\ttheme.css,code.js,ifruit_5.jpg\n9\ttitanic_1000_sales.html\ttheme.css,code.js,titanic_1000.jpg\n10\tmeetoo_1.0_sales.html\ttheme.css,code.js,meetoo_1.0.jpg\n11\tsorrento_f21l_sales.html\ttheme.css,code.js,sorrento_f21l.jpg\n12\tifruit_4_sales.html\ttheme.css,code.js,ifruit_4.jpg\n13\tsorrento_f23l_sales.html\ttheme.css,code.js,sorrento_f23l.jpg\n'
```

The content of the file is very unstructured, and we cannot easily figure out exactly what information is there and in what order they have been arranged.

To make the content more readable use `splitlines()` method. The method will split your string into a list by **splitting at line breaks**. If you have a close look to the above image **\n** is the line break (new line) sign.

You can therefore make your output much more presentable by passing through this list using `splitlines()` function. Following is an application of the `splitlines()` function. How can you use this to get the job done in DBFS? The key is what should you put instead of **X**.

```
for line in X.splitlines():
    print (line)
```

X could be **the path to the file with the head method** so try the following piece of command:

```
for line in dbutils.fs.head("/FileStore/tables/webpage/part-m-00000").splitlines():
    print(line)
```

```
1 for line in dbutils.fs.head("/FileStore/tables/webpage/part-m-00000").splitlines():
2     print(line)
```

```
1 sorrento_f00l_sales.html      theme.css,code.js,sorrento_f00l.jpg
2 titanic_2100_sales.html      theme.css,code.js,titanic_2100.jpg
3 meetoo_3.0_sales.html        theme.css,code.js,meetoo_3.0.jpg
4 meetoo_3.1_sales.html        theme.css,code.js,meetoo_3.1.jpg
5 ifruit_1_sales.html          theme.css,code.js,ifruit_1.jpg
6 ifruit_3_sales.html          theme.css,code.js,ifruit_3.jpg
7 ifruit_2_sales.html          theme.css,code.js,ifruit_2.jpg
8 ifruit_5_sales.html          theme.css,code.js,ifruit_5.jpg
9 titanic_1000_sales.html      theme.css,code.js,titanic_1000.jpg
10 meetoo_1.0_sales.html        theme.css,code.js,meetoo_1.0.jpg
11 sorrento_f21l_sales.html      theme.css,code.js,sorrento_f21l.jpg
12 ifruit_4_sales.html          theme.css,code.js,ifruit_4.jpg
13 sorrento_f23l_sales.html      theme.css,code.js,sorrento_f23l.jpg
```

Now, content is tidier and more readable. The 3 columns are id (an integer), followed by two string columns: webpage and associated files. Note that the data in the last column, the associated files, is a comma-delimited string. We're going to extract the data in that column, split the string and create a new dataset containing each webpage and its associated files in separate rows.

3) Different ways of creating a DataFrame

You should create a new DataFrame called webpages from the webpage data. You should try creating this in three different ways:

- 1) Using `createDataFrame`,
- 2) Using `spark.read.csv()`,
- 3) Using `.toDF()` method from an RDD.

Method (1) is using `createDataFrame`, where you create a schema and an RDD and use these to create the DataFrame.

```
# Using createDataFrame

from pyspark.sql.types import *

mySchema = StructType ([
    StructField("index", IntegerType()) ,
    StructField("webpage", StringType()) ,
    StructField("associated_files", StringType())])

myRDD = sc.textFile("/FileStore/tables/webpage/*"). \
    map(lambda line: line.split("\t")). \
    map(lambda values: [int(values[0]) , values[1] , values[2]])

webpages1 = spark.createDataFrame(myRDD , mySchema)

webpages1.show(5)
```

```
1  # Using createDataFrame
2
3  from pyspark.sql.types import *
4
5  mySchema = StructType([StructField ("index", IntegerType() ,
6                                StructField("webpage", StringType()) ,
7                                StructField("associated_files", StringType()))])
8
9
10 myRDD = sc.textFile("/FileStore/tables/webpage/*"). \
11     map(lambda line: line.split ("\t")). \
12     map(lambda values: [int(values[0]) , values[1] , values[2]])
13
14 webpages1 = spark.createDataFrame(myRDD , mySchema)
15
16 webpages1.show(5)
```

Data type of the first column

To define 3 columns of the dataframe

To create an RDD

To generate a dataframe from RDD

▶ (1) Spark Jobs

▶  webpages1: pyspark.sql.dataframe.DataFrame = [index: integer, webpage: string ... 1 more field]

index	webpage	associated_files
1	sorrento_f00l_sal...	theme.css,code.js...
2	titanic_2100_sale...	theme.css,code.js...
3	meetoo_3.0_sales....	theme.css,code.js...
4	meetoo_3.1_sales....	theme.css,code.js...
5	ifruit_1_sales.html	theme.css,code.js...


only showing top 5 rows

By default, show() method truncates long columns (sorrento_f00l_sal...) however, you can change this behaviour by **passing a Boolean value False to show() method to display the full content**. Try, webpages.show(5, truncate = False)

in createDataFrame method you can run the method without defining a schema and spark will infer a schema itself. But column names will not be as you want. Also, by default columns could have null values. Check the result if you remove schema.

```
3 from pyspark.sql.types import *
4
5 myRDD = sc.textFile("/FileStore/tables/webpage/*"). \
6     map(lambda line: line.split("\t")). \
7     map(lambda values: [int(values[0]) , values[1] , values[2]])
8
9 webpages1_test = spark.createDataFrame(myRDD)
10
11 webpages1_test.show(5, truncate = False)
12
13 webpages1_test.printSchema()
```

▶ (2) Spark Jobs

▶  webpages1_test: pyspark.sql.dataframe.DataFrame = [_1: long, _2: string ... 1 more field]

_1	_2	_3
1	sorrento_f00l_sales.html	theme.css,code.js,sorrento_f00l.jpg
2	titanic_2100_sales.html	theme.css,code.js,titanic_2100.jpg
3	meetoo_3.0_sales.html	theme.css,code.js,meetoo_3.0.jpg
4	meetoo_3.1_sales.html	theme.css,code.js,meetoo_3.1.jpg
5	ifruit_1_sales.html	theme.css,code.js,ifruit_1.jpg

only showing top 5 rows

root

```
-- _1: long (nullable = true)
-- _2: string (nullable = true)
-- _3: string (nullable = true)
```

Method (2) is using `spark.read.csv()`.

Spark SQL provides `spark.read().csv("file_name")` to read a file or directory of files in CSV format into Spark DataFrame, and `DataFrame.write().csv("path")` to write to a CSV file. Function `option()` can be used to customize the behaviour of reading or writing, such as controlling behaviour of the header, delimiter character, character set, and so on.

Don't forget that you will need to change the default delimiter to `\t`.

```
Webpages2 = spark.read.options(delimiter = "\t").csv("/FileStore/tables/webpage/*")
```

```
Webpages2.show(5, truncate = False)
```

```
Cmd 19

1  webpages2 = spark.read.options(delimiter = "\t").csv("/FileStore/tables/webpage/*")
2
3  webpages2.show(5, truncate = False)
```

► (2) Spark Jobs

► webpages2: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 1 more field]

	_c0	_c1	_c2
14	titanic_2200_sales.html	theme.css,code.js,titanic_2200.jpg	
15	ronin_novelty_note_1_sales.html	theme.css,code.js,ronin_novelty_note_1.jpg	
16	titanic_2500_sales.html	theme.css,code.js,titanic_2500.jpg	
17	ronin_novelty_note_3_sales.html	theme.css,code.js,ronin_novelty_note_3.jpg	
18	ronin_novelty_note_2_sales.html	theme.css,code.js,ronin_novelty_note_2.jpg	

only showing top 5 rows

Based on the `\t` delimiter `spark.read.csv()` method has detected DataFrame columns and has inferred an schema and you see the extracted DataFrame. Name of the columns has been generated like `c0`, `c1`, ... that are not representative of the contents. Check the schema of the DataFrame. Check the following image of the row data and find out why you have used `\t` as delimiter.

```
Cmd 8
1 dbutils.fs.head("/FileStore/tables/webpage/part-m-00000")

Out[17]: '1\tsorrento_f00l_sales.html\ttheme.css,code.js,sorrento_f00l.jpg\n2\ttitanic_2100_sales.html\ttheme.css,code.js,titanic_2100.jpg\n3\tmeetoo_3.0_sales.html\ttheme.css,code.js,meetoo_3.0.jpg\n4\tmeetoo_3.1_sales.html\ttheme.css,code.js,meetoo_3.1.jpg\n5\tifruit_1_sales.html\ttheme.css,code.js,ifruit_1.jpg\n6\tifruit_3_sales.html\ttheme.css,code.js,ifruit_3.jpg\n7\tifruit_2_sales.html\ttheme.css,code.js,ifruit_2.jpg\n8\tifruit_5_sales.html\ttheme.css,code.js,ifruit_5.jpg\n9\ttitanic_1000_sales.html\ttheme.css,code.js,titanic_1000.jpg\n10\tmeetoo_1.0_sales.html\ttheme.css,code.js,meetoo_1.0.jpg\n11\tsorrento_f21l_sales.html\ttheme.css,code.js,sorrento_f21l.jpg\n12\tifruit_4_sales.html\ttheme.css,code.js,ifruit_4.jpg\n13\tsorrento_f23l_sales.html\ttheme.css,code.js,sorrento_f23l.jpg\n'
```

Method (3) is `.toDF`. This method is used to create a DataFrame with the specified column names and it creates DataFrame from RDD. Since RDD is schema-less (without column names and data types), converting from RDD to DataFrame gives you default column names as `_1`, `_2` and so on and data type as String.



```
myRDD = sc.textFile("/FileStore/tables/webpage/*")

colRDD = myRDD.map(lambda line: line.split("\t"))

webpages3 = colRDD.toDF()

webpages3.show(5, truncate = False)
```

```
Cmd 20
1 myRDD = sc.textFile("/FileStore/tables/webpage/*")
2
3 colRDD = myRDD.map(lambda line: line.split("\t"))
4
5 #print(colRDD.take(3))
6
7 webpages3 = colRDD.toDF()
8
9 webpages3.show(5, truncate = False)
```

▶ (2) Spark Jobs

▶ webpages3: pyspark.sql.dataframe.DataFrame = [_1: string, _2: string ... 1 more field]

_1	_2	_3
1	sorrento_f00l_sales.html	theme.css,code.js,sorrento_f00l.jpg
2	titanic_2100_sales.html	theme.css,code.js,titanic_2100.jpg
3	meetoo_3.0_sales.html	theme.css,code.js,meetoo_3.0.jpg
4	meetoo_3.1_sales.html	theme.css,code.js,meetoo_3.1.jpg
5	ifruit_1_sales.html	theme.css,code.js,ifruit_1.jpg

only showing top 5 rows

4) Checking schemas of the created DataFrames and unifying DataFrames.

Now you have created 3 DataFrames (webpages1, webpages2, webpages3) from one dataset. You should check schemas of DataFrames and if schemas are not identical then should see how you can unify them. Run `printSchema()` for all 3 DataFrames.

```
Cmd 24
1  webpages1.printSchema()
2
3  webpages2.printSchema()
4
5  webpages3.printSchema()
6

root
 |-- num: integer (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)

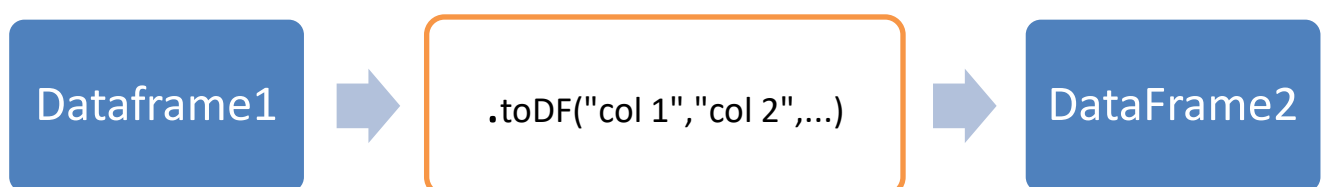
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)

root
 |-- _1: string (nullable = true)
 |-- _2: string (nullable = true)
 |-- _3: string (nullable = true)
```

The column names and data types are different.

We want to keep the schema of webpages1 and change other schemas to match webpages1.

There are many methods to change the columns' name. For example, `.toDF()` has another useful functionality. You can pass a DataFrame to the function and change its columns name.



Cmd 21

```
1 webpages2 = webpages2.toDF("index", "webpage" , "associated_files")
2
3 webpages2.show(5, truncate = False)
```

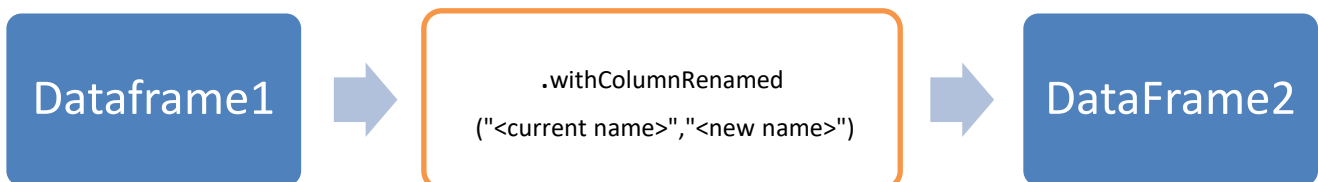
▶ (1) Spark Jobs

▶  webpages2: pyspark.sql.dataframe.DataFrame = [index: string, webpage: string ... 1 more field]

index	webpage	associated_files
14	titanic_2200_sales.html	theme.css,code.js,titanic_2200.jpg
15	ronin_novelty_note_1_sales.html	theme.css,code.js,ronin_novelty_note_1.jpg
16	titanic_2500_sales.html	theme.css,code.js,titanic_2500.jpg
17	ronin_novelty_note_3_sales.html	theme.css,code.js,ronin_novelty_note_3.jpg
18	ronin_novelty_note_2_sales.html	theme.css,code.js,ronin_novelty_note_2.jpg

only showing top 5 rows

Another way to change columns name is as follows but here you should change each columns name in a separate function.



Cmd 22

```
1 webpages3 = webpages3.withColumnRenamed("_1" , "index"). \
2 withColumnRenamed("_2" , "webpage"). \
3 withColumnRenamed("_3" , "associated_files")
4
5 webpages3.show(5, truncate = False)
```

▶ (1) Spark Jobs

▶  webpages3: pyspark.sql.dataframe.DataFrame = [index: string, webpage: string ... 1 more field]

index	webpage	associated_files
1	sorrento_f00l_sales.html	theme.css,code.js,sorrento_f00l.jpg
2	titanic_2100_sales.html	theme.css,code.js,titanic_2100.jpg
3	meetoo_3.0_sales.html	theme.css,code.js,meetoo_3.0.jpg
4	meetoo_3.1_sales.html	theme.css,code.js,meetoo_3.1.jpg
5	ifruit_1_sales.html	theme.css,code.js,ifruit_1.jpg

only showing top 5 rows

Now, all 3 DataFrames have similar columns.

Cmd 24

```
1 webpages1.printSchema()
2
3 webpages2.printSchema()
4
5 webpages3.printSchema()
6

root
 |-- index: integer (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)

root
 |-- index: string (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)

root
 |-- index: string (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)
```

First columns of webpages2 and webpages3 must be converted to integer. One way to change it is using withColumn and cast.

Cmd 26

```
1 webpages2 = webpages2.withColumn("index", webpages2.index.cast("int"))
2
3 webpages3 = webpages3.withColumn("index", webpages3.index.cast("int"))

▶ webpages2: pyspark.sql.dataframe.DataFrame = [index: integer, webpage: string ... 1 more field]
▶ webpages3: pyspark.sql.dataframe.DataFrame = [index: integer, webpage: string ... 1 more field]
```

Check the schemas again to make sure the all webpages DataFrames have the same schema.

Cmd 27

```
1  webpages1.printSchema()
2
3  webpages2.printSchema()
4
5  webpages3.printSchema()

root
 |-- index: integer (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)

root
 |-- index: integer (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)

root
 |-- index: integer (nullable = true)
 |-- webpage: string (nullable = true)
 |-- associated_files: string (nullable = true)
```

5) Using Pair RDDs to manipulate data and create a new DataFrame

Create a new DataFrame by selecting the webpage and associated files columns from the existing DataFrame.

```
assocfilesDF = webpages1.select(webpages1.webpage , webpages1.associated_files)
assocfilesDF.show(5 , truncate = False)
```

In order to manipulate the data using Spark, convert the DataFrame into a Pair RDD using the map method. The input into the map method is a Row object. The key is the webpage value, and the value is the associated files string.

```
afilesRDD = assocfilesDF.rdd.map(lambda row:(row.webpage , row.associated_files))
afilesRDD.take(5)
```

Cmd 31

```
1 afilesRDD = assocfilesDF.rdd.map(lambda row:(row.webpage , row.associated_files))
2
3 afilesRDD.take(5)
```

► (1) Spark Jobs

```
Out[37]: [('sorrento_f00l_sales.html', 'theme.css,code.js,sorrento_f00l.jpg'),
('titanic_2100_sales.html', 'theme.css,code.js,titanic_2100.jpg'),
('meetoo_3.0_sales.html', 'theme.css,code.js,meetoo_3.0.jpg'),
('meetoo_3.1_sales.html', 'theme.css,code.js,meetoo_3.1.jpg'),
('ifruit_1_sales.html', 'theme.css,code.js,ifruit_1.jpg')]
```

Now that you have an RDD, you can use the familiar flatMapValues transformation to split and extract the filenames in the associated_files column.

```
afilesRDD2 = afilesRDD.flatMapValues(lambda x: x.split(","))
afilesRDD2.take(10)
```

Cmd 33

```
1 afilesRDD2 = afilesRDD.flatMapValues(lambda x: x.split(","))
2
3 afilesRDD2.take(10)
```

► (1) Spark Jobs

```
Out[39]: [('sorrento_f00l_sales.html', 'theme.css'),
('sorrento_f00l_sales.html', 'code.js'),
('sorrento_f00l_sales.html', 'sorrento_f00l.jpg'),
('titanic_2100_sales.html', 'theme.css'),
('titanic_2100_sales.html', 'code.js'),
('titanic_2100_sales.html', 'titanic_2100.jpg'),
('meetoo_3.0_sales.html', 'theme.css'),
('meetoo_3.0_sales.html', 'code.js'),
('meetoo_3.0_sales.html', 'meetoo_3.0.jpg'),
('meetoo_3.1_sales.html', 'theme.css')]
```

Same webpage different files


Now, create a new DataFrame from the final RDD.

```
afileDF = afilesRDD2.toDF( ["web_page_num" , "associated_file"] )  
afileDF.show(5 , truncate = False)
```

Cmd 36

```
1  afileDF = afilesRDD2.toDF( ["web_page_num" , "associated_file"] )  
2  
3  afileDF.show(5 , truncate = False)
```

▶ (2) Spark Jobs

▶  afileDF: pyspark.sql.dataframe.DataFrame = [web_page_num: string, associated_file: string]

```
+-----+-----+  
|web_page_num          |associated_file |  
+-----+-----+  
|sorrento_f00l_sales.html|theme.css      |  
|sorrento_f00l_sales.html|code.js        |  
|sorrento_f00l_sales.html|sorrento_f00l.jpg|  
|titanic_2100_sales.html |theme.css      |  
|titanic_2100_sales.html |code.js        |  
+-----+-----+
```

only showing top 5 rows

Cmd 38

```
1  afileDF.printSchema()
```

```
root  
|-- web_page_num: string (nullable = true)  
|-- associated_file: string (nullable = true)
```

Now, select those rows of the `afileDF` that contain `.jpg` file extensions in the `associated_file` column and save in a new DataFrame, `afileDF_jpg`.

Try to figure out what should put instead of `??`

```
afileDF_jpg = afileDF.filter(??[?].like(??) )  
afileDF_jpg.show(15 , truncate = False)
```

Output should be:

```
+-----+-----+  
|web_page_num      |associated_file      |  
+-----+-----+  
|sorrento_f00l_sales.html  |sorrento_f00l.jpg   |  
|titanic_2100_sales.html   |titanic_2100.jpg    |  
|meetoo_3.0_sales.html     |meetoo_3.0.jpg      |  
|meetoo_3.1_sales.html     |meetoo_3.1.jpg      |  
|ifruit_1_sales.html       |ifruit_1.jpg         |  
|ifruit_3_sales.html       |ifruit_3.jpg         |  
|ifruit_2_sales.html       |ifruit_2.jpg         |  
|ifruit_5_sales.html       |ifruit_5.jpg         |  
|titanic_1000_sales.html   |titanic_1000.jpg    |  
|meetoo_1.0_sales.html     |meetoo_1.0.jpg      |  
|sorrento_f21l_sales.html  |sorrento_f21l.jpg   |  
|ifruit_4_sales.html       |ifruit_4.jpg         |  
|sorrento_f23l_sales.html  |sorrento_f23l.jpg   |  
|titanic_2200_sales.html   |titanic_2200.jpg    |  
|ronin_novelty_note_1_sales.html|ronin_novelty_note_1.jpg|  
+-----+-----+  
only showing top 15 rows
```

Your 2 DataFrames (`afileDF` and `afileD_jpg`) contain the processed data, so save them in **Parquet** format (the default) in `/FileStore/tables/webpage` directory and check the files have been created.

```
afileDF.write.save("/FileStore/tables/webpage_files_all")  
afileDF_jpg.write.save("/FileStore/tables/webpage_files_jpg")  
dbutils.fs.ls("/FileStore/tables/")
```

Part 4: Making your notebook rerunnable: A challenge for you

You have written and executed many cells today and you know that we call all these pieces of code together a notebook, from cell one down to the end.

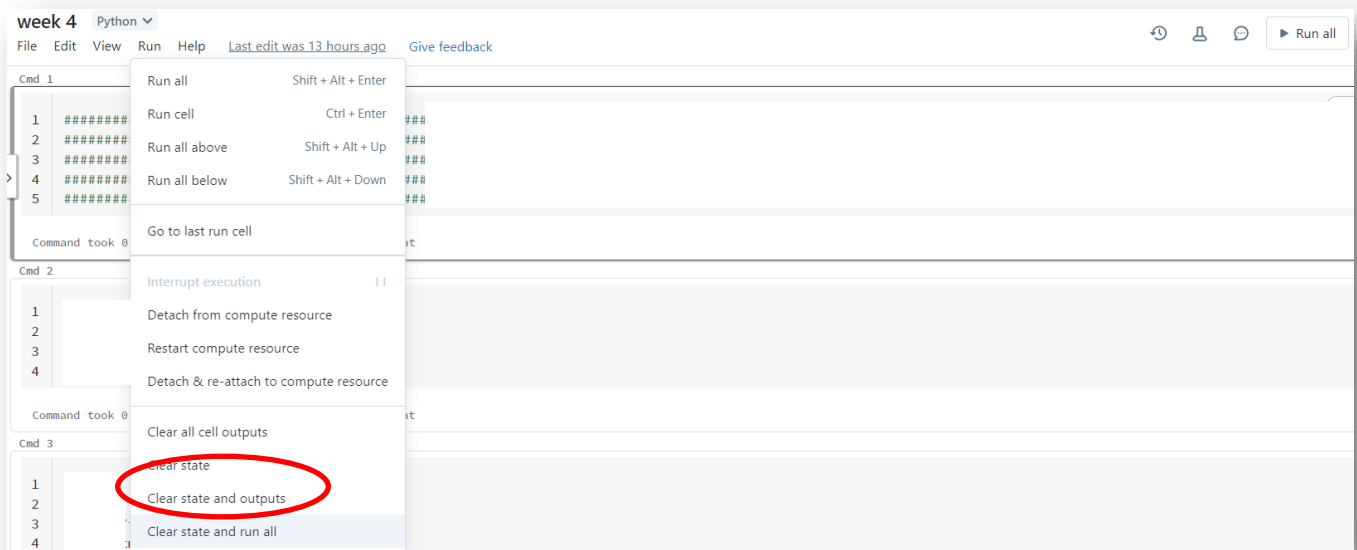
Now, we want you to check if your entire notebook is rerunnable in one go or not.

Go to the toolbar of the notebook there is a “Run” tab and there are multiple options for running the notebook. If you choose “Clear state and run all” then your notebook will be fully fresh without any output (don't worry your codes will stay the same) and then will be run from the first cell to the last cell one by one.

If the “Clear state and run all” command couldn't execute a cell, then the execution of the subsequent cells will be terminated. In that situation, you should go and find out why your notebook couldn't run that cell and try to debug the issue.

This is very important for you to learn it, either as a future Data Scientist or to get a better mark in the module assignment. In the final assignment, we will ask you to make sure that your notebook is rerunnable.

Make sure you can complete this stage then you can say I have completed this workshop.



References and Resources:

<https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.sql.DataFrame.html>

<https://sparkbyexamples.com/pyspark/different-ways-to-create-dataframe-in-pyspark/>