

University of Salford, MSc Data Science

Module: Big Data Tools and Techniques

Date: Trimester 2, 2024-2025

Session: Workshop Week 3

Topic: Resilient Distributed Systems (RDDs) in Spark

Tools: Databricks Community Edition

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

Objectives:

After completing this workshop, you will be able to:

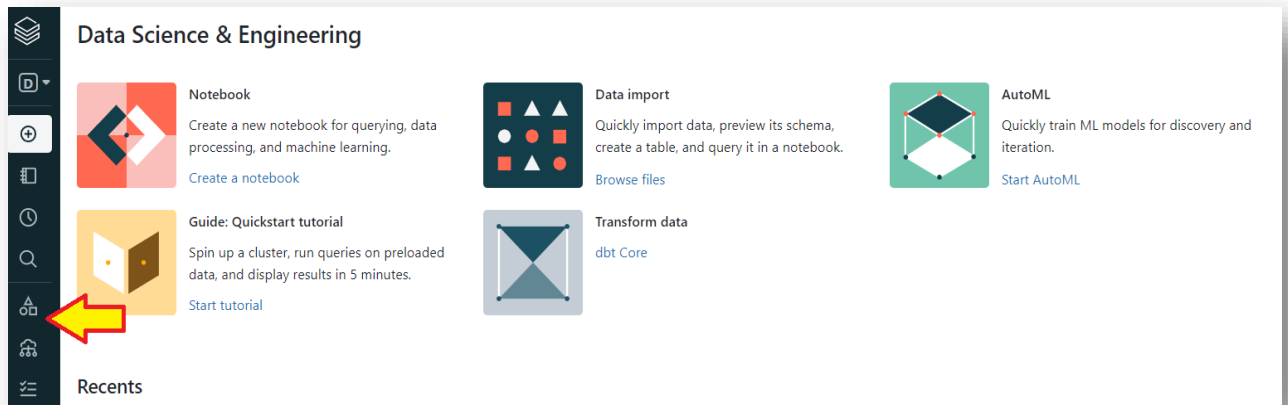
- Creating different RDDs
- Running various RDD operations
- Creating and working with pair RDDs

Table of Contents

Part 1: Fire up the Databricks workspace	3
Part 2: Create a Python notebook.....	3
Part 3: Working with single RDDs	5
1) Put activations data on Databricks.....	5
2) Run different Transformation and Action operations.....	8
Part 4: Working with pair RDDs.....	16
1) Put accounts and logs data on Databricks.....	16
2) Exploring logs data	18
3) Join logs data with accounts data	25
References and Resources:.....	28

Part 1: Fire up the Databricks workspace

1. Log in to your Databricks Community Edition account.
<https://community.cloud.databricks.com/login.html>
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.

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.

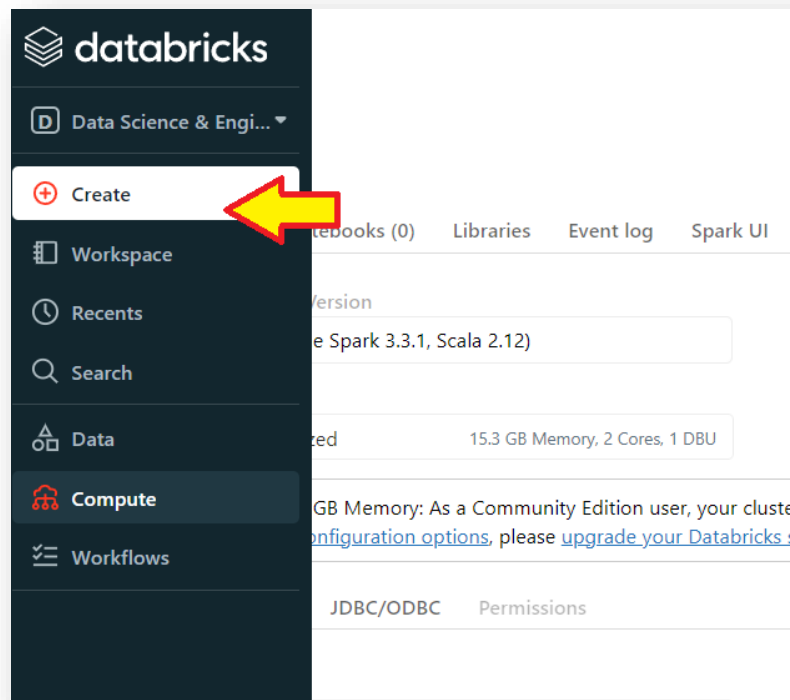
Part 2: Create a Python notebook

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

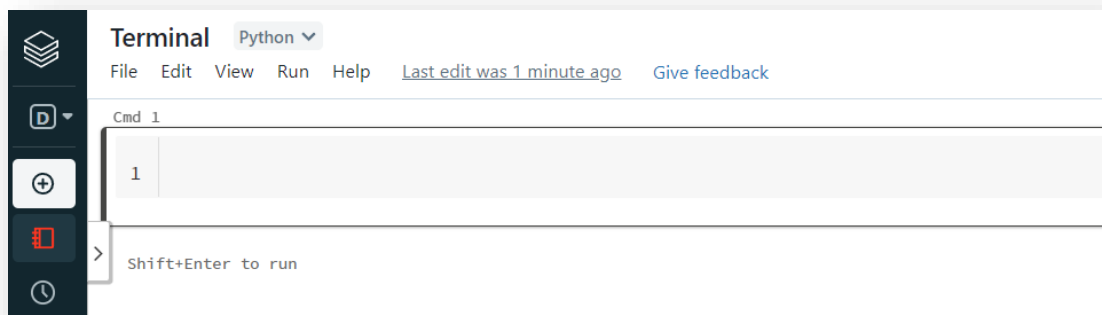
<https://docs.databricks.com/notebooks/notebooks-use.html>

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. “RDD week 3”, 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:

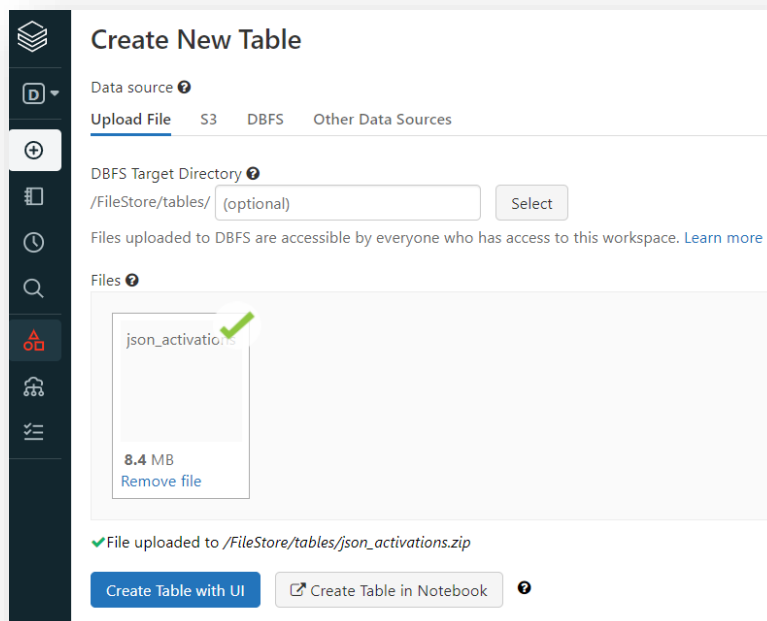


Part 3: Working with single RDDs

In this part of the lab, you will parse a set of activation records in JSON format to extract the account numbers and model names. Spark is commonly used for ETL (Extract/Transform/Load) operations. Sometimes data is stored in line-oriented records, like the web logs in the previous exercise, but sometimes the data is in a multi-line format that must be processed as a whole file. In this exercise you will practice working with file-based instead of line-based formats. Review week 3, lecture notes about **sc.textFile** versus **sc.wholeTextFiles** functions.

1) Put activations data on Databricks

- 1) Download the **activations.zip** file from Blackboard. This will download it to the machine you're using. To get the zip file onto Databricks, we'll use the Databricks user interface (UI).
- 2) Log into Databricks and ensure you are on the home screen - this is the page which says "Data Science & Engineering" (You can get to this screen by clicking the topmost icon on the left).
- 3) There should be multiple icons, one icon is titled "Data Import". Click on "Browse files" below this section and select the **activations.zip** file that you have downloaded to your system. Wait for the file to be uploaded, but do not click on anything else on that page after the upload finishes.



- 4) By default, the UI uploads to /FileStore/tables/. Back in a notebook, use the `dbutils.fs.ls` command to check that the file has been uploaded.

```
dbutils.fs.ls("/FileStore/tables/")
```

- 5) We need to extract this zip archive. Since the `dbutils` toolkit doesn't provide an `unzip` command, you need to copy the file to the driver node (local file system), extract there using a shell command, and put the extracted contents back into DBFS. So, first copy the `activations.zip` from DBFS to the `/tmp` directory on your driver node using `dbutils` and verify the file has been copied.

```
dbutils.fs.cp("/FileStore/tables/activations.zip", "file:/tmp/")
```

Check the `tmp` directory to see if `activations.zip` is in it.

```
%sh  
ls /tmp/
```

- 6) Use the UNIX command `unzip` to extract the contents. I.e. assuming success in Point 6, you should be able to run:

```
%sh  
unzip -d /tmp/ /tmp/activations.zip
```

which should return information about the creation of an `activations` directory and that various files have been inflated into it. The `-d` option tells `unzip` to perform the extraction into `tmp` rather than your current working directory. Confirm this has worked with an `ls` of the `/tmp/` and `/tmp/activations` directories.

```
%sh  
ls /tmp/activations
```

You should see many `json` files in the `activations'` directory:

```
1 %sh
2
3 ls /tmp/activations
4
```

```
2008-10.json
2008-11.json
2008-12.json
2009-01.json
2009-02.json
2009-03.json
2009-04.json
2009-05.json
2009-06.json
2009-07.json
2009-08.json
2009-09.json
2009-10.json
2009-11.json
2009-12.json
2010-01.json
2010-02.json
2010-03.json
2010-04.json
2010-05.json
2010-06.json
```

- 7) Now, we want to move unzipped files to DBFS. Create a DBFS directory activations in the /FileStore/ directory using the `dbutils.fs.mkdirs` command.

```
dbutils.fs.mkdirs("FileStore/tables/activations")
```

- 8) Now you need to move (using the `dbutils.fs.mv` command) the contents of the local /tmp/activations directory into your newly created DBFS /FileStore/activations directory so you can perform some Spark processing. If you get an error when trying to use this command, you should look up its details so you can address the problem. Check the contents of /FileStore/activations and confirm it contains the expected files.

```
dbutils.fs.mv("file:/tmp/activations", "/FileStore/tables/activations", True)
```

```
dbutils.fs.ls("FileStore/tables/activations/")
```

- 9) Let's count number of json files in the activations directory. There are 66 json files.

```
len(dbutils.fs.ls("FileStore/tables/activations/"))
```

- 10) Use the `dbutils.fs.head` command to view the format of one of the files (2008-10.json). Although perhaps not formatted as well, it should look something like this:

```
{
  "activations": {
    "activation": [
      {
        "timestamp": "1238425765",
        "type": "phone",
        "account-number": "494",
        "device-id": "0dff21c2-84a8-4775-8c95-27b79c3ef79d",
        "phone-number": "2097791928",
        "model": "Sorrento F00L"
      },
      ...
    ]
  }
}
```

```
dbutils.fs.head("FileStore/tables/activations/2008-10.json")
```

2) Run different Transformation and Action operations

Your code should process a set of activation JSON files and extract the account number and device model for each activation, and save the list to a file formatted as account number:model. The output will look something like:

```
1234:iFruit 1
987:Sorrento F00L
4566:iFruit 1
...
```

- 11) Use `wholeTextFiles` to create an RDD from the activations dataset. The resulting RDD will consist of tuples, in which the first value is the name of the file, and the second value is the contents of the file (JSON) as a string.

```
myrdd1 = sc.wholeTextFiles('/FileStore/tables/activations/')
```

Now we have an RDD file and it contains all 66 json files in the activations directory. Since we have 66 json files in the folder, `myrdd1` has 66 elements. `myrdd1` structure is like:

('1st file path', '{content of the 1st file}')
('2nd file path', '{content of the 2nd file}')
...
('66th file path', '{content of the 66th file}')

Let's explore **first 2 elements of myrdd1** (or first two json files):

```
myrdd1.take(2)
```

Path of the 1st json file

Content of the 1st file

```

{"_id": "c14dfbs:/FileStore/tables/activations/2008-10.json",
  "activations": {
    "activation": {
      "timestamp": "12bea35e-3ecd-4ee7-b0dd-ad428d953f32",
      "phone-number": "7600763387",
      "phone": "phone",
      "account-number": "426",
      "device-id": "38a1566d-509",
      "timestamp": "1225446947",
      "type": "phone",
      "phone-number": "91622096560",
      "model": "Sorrento F00L",
      "device-id": "404",
      "device-id": "93d7d0eb-7551-4e11-ab59-91f34c04378f",
      "timestamp": "12255444099",
      "type": "phone",
      "account-number": "396402",
      "model": "iFruit 1",
      "timestamp": "122e48-d347-4ec5-b010-9d9a3b13d999",
      "phone-number": "6503155955",
      "phone": "phone",
      "account-number": "96",
      "device-id": "351a0384-9af6-4e",
      "timestamp": "1225406348",
      "type": "phone",
      "phone-number": "5109683453",
      "model": "Sorrento F00L",
      "device-id": "e4ceefc4-8765-4beb-8b10-9b3d73eec535",
      "phone-number": "1225386391",
      "type": "phone",
      "account-number": "475",
      "model": "Titanic 1000",
      "timestamp": "1225371984",
      "phone-number": "e99-80c9-c4b0a030108e",
      "phone-number": "6507811206",
      "model": "account-number": "306",
      "device-id": "d743093b-8bcf-4ede-a475-4bc29936",
      "timestamp": "1225349055",
      "type": "phone",
      "account-number": "9098824980",
      "model": "MeeToo 1.6",
      "timestamp": "c14c3c3d-8b55-49bb-931d-7b2472aaa750",
      "phone-number": "40828870"
    }
  }
}

```

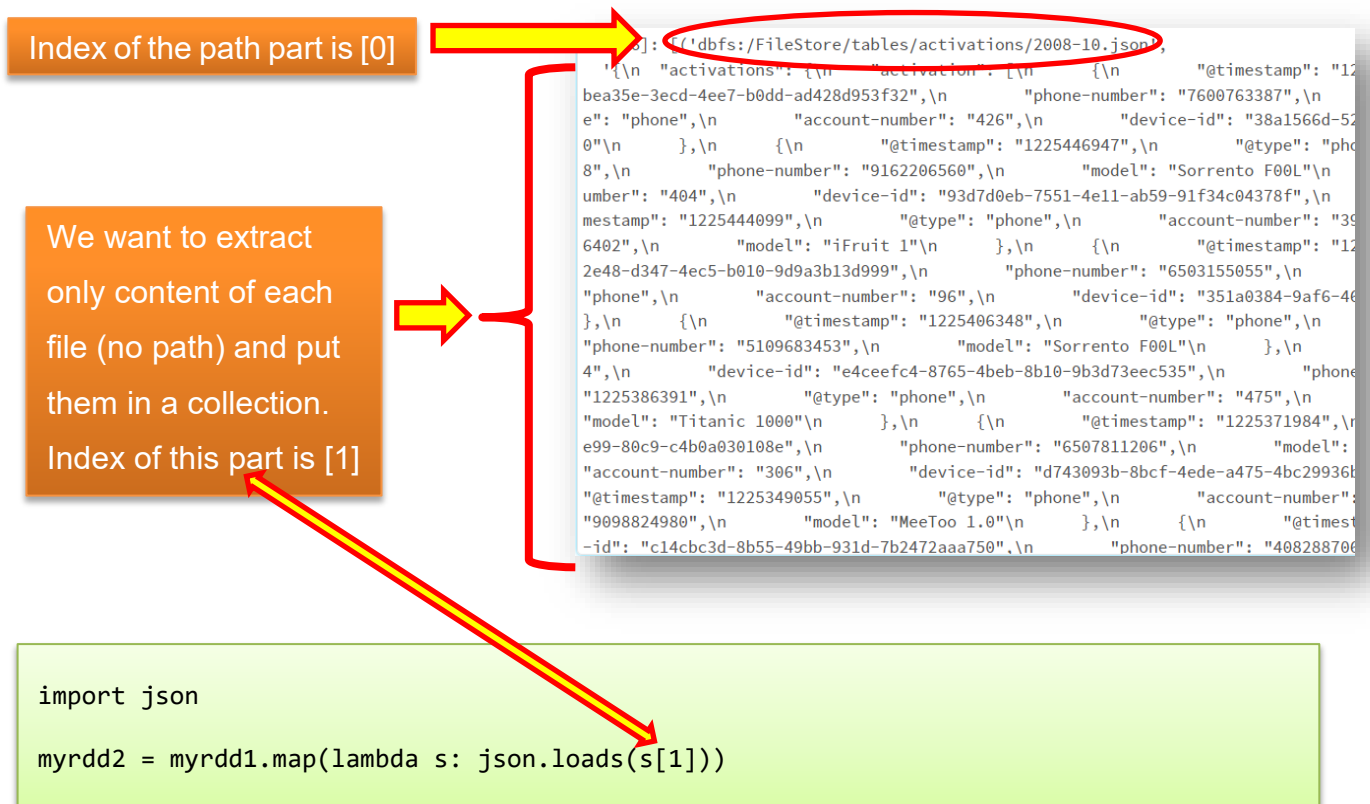
If you scroll down the output, you can find the second element of myrdd1:

path of the 2nd json file

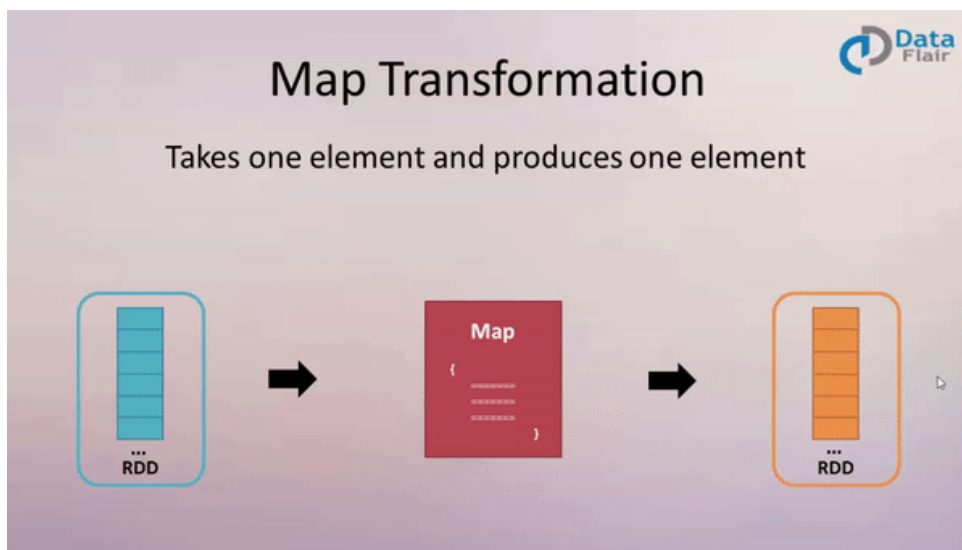
Content of the 2nd file

[illegible]

- 12) Each json file can contain many activation records; map the contents of each file to a new RDD elements.



You know that the map function is a RDD transformation operation. It takes each element of a RDD, run a function over the element and returns a new element. So, **one element in and one element out** (click on the image to see the GIF version).



<https://data-flair.training/blogs/apache-spark-map-vs-flatmap/>

Let's see the first element of the myrdd2. You can see multiple phone activations in the first element. For example, at the '@timestamp': '1225462088' 1st activity and the 2nd activity at the '@timestamp': '1225461447' and so on.

```
1 myrdd2.take(1)

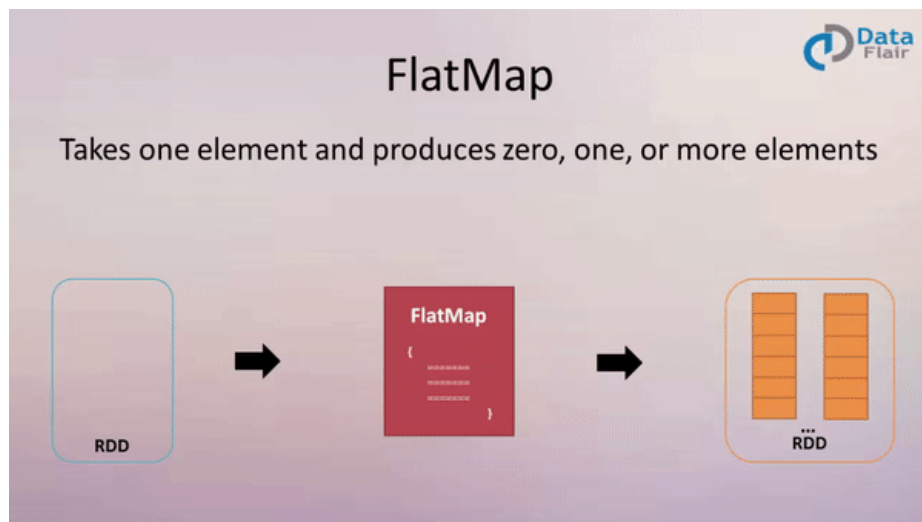
▶ (1) Spark Jobs

Out[46]: [{ 'activations': { 'activation': [ { '@timestamp': '1225462088',
  '@type': 'phone',
  'account-number': '9763',
  'device-id': 'debea35e-3ecd-4ee7-b0dd-ad428d953f32',
  'phone-number': '7600763387',
  'model': 'MacTos 1.0' },
  { '@timestamp': '1225461447',
  '@type': 'phone',
  'account-number': '426',
  'device-id': '38a1566d-524e-4137-bad8-b597d09b54b0',
  'phone-number': '5102521038',
  'model': 'Titanic 1000' },
  { '@timestamp': '1225446947',
  '@type': 'phone',
  'account-number': '383',
  'device-id': '513024a3-a828-4674-9ff6-041e9a851f18',
  'phone-number': '9162206560',
  'model': 'Sorrento F00L' },
  { '@timestamp': '1225445908',
  '@type': 'phone',
  'account-number': '1404' }
```

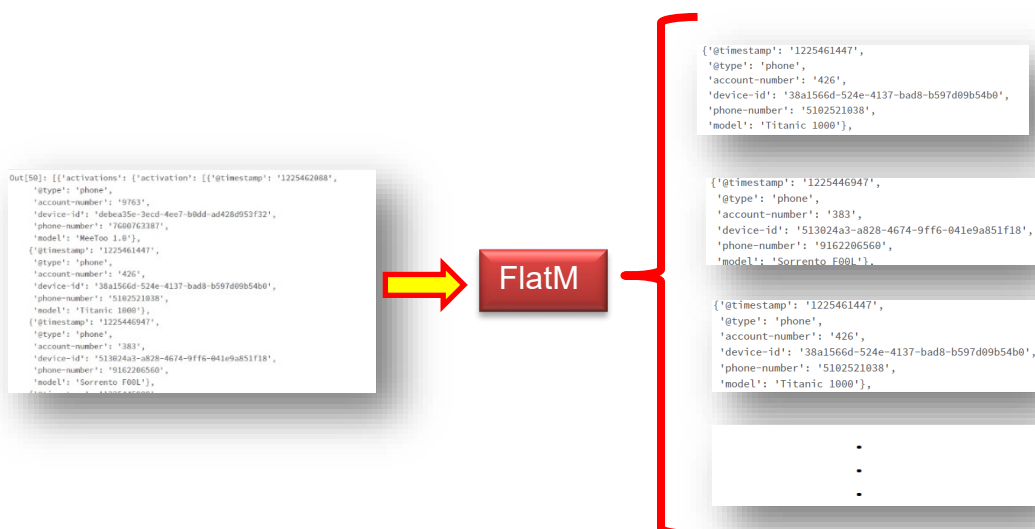
We want to extract all activations from all elements of myrdd2 and store them in new RDD. In other words, each activation must be a separate element in the new RDD. If you remember, we had 66 json files and inside each json file there are many phone activation records like:

```
{ '@timestamp': '1225461447',  
  '@type': 'phone',  
  'account-number': '426',  
  'device-id': '38a1566d-524e-4137-bad8-b597d09b54b0',  
  'phone-number': '5102521038',  
  'model': 'Titanic 1000'},
```

We want to extract all these elements and store in a new RDD where each element of the new RDD is one of these activations. We need to run FlatMap RDD transformation operation. FlatMap takes one element in but can send multiple elements out (**click on the image to see the GIF version**)



<https://data-flair.training/blogs/apache-spark-map-vs-flatmap/>



13) Apply a FlatMap operation to extract all activations.

```
myrdd3 = myrdd2.flatMap(lambda x: x["activations"]["activation"])
```

Let's see first 3 elements of myrdd3:

```
myrdd3.take(3)
```

```
1 myrdd3.take(3)
```

► (1) Spark Jobs

```
Out[55]: [{'@timestamp': '1225462088',
  '@type': 'phone',
  'account-number': '9763',
  'device-id': 'debea35e-3ecd-4ee7-b0dd-ad428d953f32',
  'phone-number': '7600763387',
  'model': 'MeeToo 1.0'},
 {'@timestamp': '1225461447',
  '@type': 'phone',
  'account-number': '426',
  'device-id': '38a1566d-524e-4137-bad8-b597d09b54b0',
  'phone-number': '5102521038',
  'model': 'Titanic 1000'},
 {'@timestamp': '1225446947',
  '@type': 'phone',
  'account-number': '383',
  'device-id': '513024a3-a828-4674-9ff6-041e9a851f18',
  'phone-number': '9162206560',
  'model': 'Sorrento F00L'}]
```

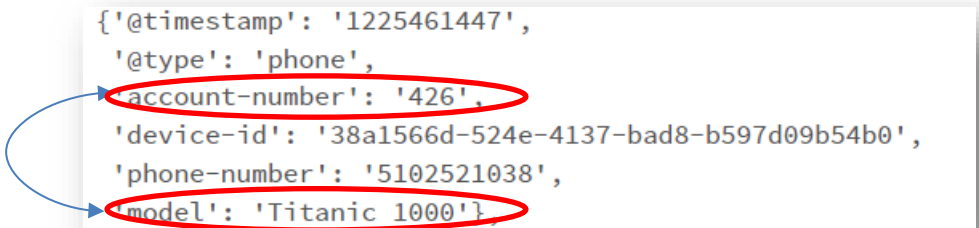
How many elements does myrdd3 have?

```
1 myrdd3.count()
```

► (1) Spark Jobs

```
Out[54]: 194764
```

- 14) Then, we want to bind account-number and model together and remove other parts. Here we should run a map transformation, because we want one element in and one element out.



```
{ '@timestamp': '1225461447',  
  '@type': 'phone',  
  'account-number': '426',  
  'device-id': '38a1566d-524e-4137-bad8-b597d09b54b0',  
  'phone-number': '5102521038',  
  'model': 'Titanic 1000' }
```

```
myrdd4 = myrdd3.map(lambda x: x["account-number"] + ":" + x["model"])
```

Cmd 19

```
1 myrdd4.take(10)
```

► (1) Spark Jobs

```
Out[58]: ['9763:MeeToo 1.0',  
          '426:Titanic 1000',  
          '383:Sorrento F00L',  
          '404:MeeToo 1.0',  
          '393:iFruit 1',  
          '53:MeeToo 1.0',  
          '96:iFruit 1',  
          '283:Sorrento F00L',  
          '464:MeeToo 1.0',  
          '475:Titanic 1000']
```

15) Finally, save the RDD file into a .txt format in the DBFS directory:
/FileStore/tables/account-models

```
myrdd4.saveAsTextFile("/FileStore/tables/account-models")
```

.txt file has been partitioned and saved into two parts:

Cmd 21

```
1 dbutils.fs.ls("/FileStore/tables/account-models/")
```

```
Out[63]: [FileInfo(path='dbfs:/FileStore/tables/account-models/_SUCCESS', name='_SUCCESS', size=0, modificationTime=1675005373000),  
FileInfo(path='dbfs:/FileStore/tables/account-models/part-00000', name='part-00000', size=1858986, modificationTime=1675005373000),  
FileInfo(path='dbfs:/FileStore/tables/account-models/part-00001', name='part-00001', size=1885226, modificationTime=1675005368000)]
```

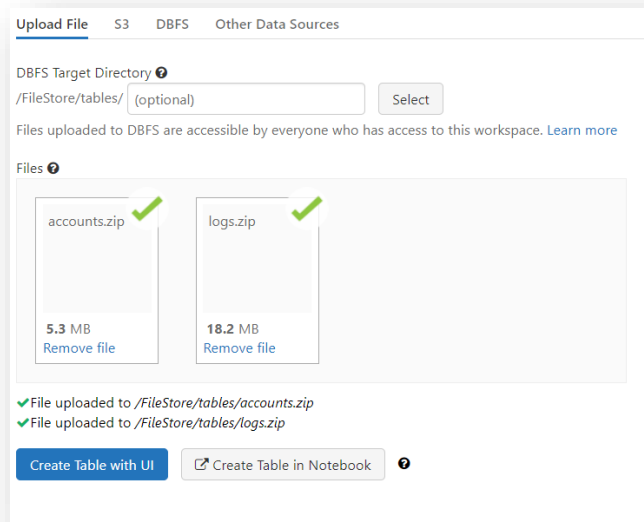
Part 4: Working with pair RDDs

1) Put accounts and logs data on Databricks

Download two zipped files `accounts.zip` and `logs.zip` from Blackboard.

You can follow the steps in Part 3 to unzip and upload two files into DBFS but this time we want to teach you how to write a reusable piece of code to do all the steps without hesitation.

- 1) First you should upload the two zip files in the DBFS:



- 2) To define a reusable codes first step is to define a variable (here is “fileroot”) for assigning to it each file name that you want to prepare. Then copy the file from DBFS to the tmp folder in the local filesystem.

Start with the `accounts` file and after running all the steps come back to the following command and change “`accounts`” to “`logs`” (`fileroot = "logs"`) and don't change anything else.

```
fileroot = "accounts"

dbutils.fs.cp("/FileStore/tables/" + fileroot + ".zip", "file:/tmp/")
```

- 3) Local filesystem or better to say shell command line doesn't know the “`fileroot`” variable and you should make this variable accessible by the command line.

```
import os

os.environ['fileroot'] = fileroot
```


4) Now, you can use fileroot in the command line. Unzip the file.

```
%sh  
unzip -d /tmp /tmp/$fileroot.zip
```

5) Then make a directory in DBFS and bring back the unzipped file to it.

```
dbutils.fs.mkdirs("/FileStore/tables/" + fileroot)
```

```
dbutils.fs.mv("file:/tmp/" + fileroot , "/FileStore/tables/" + fileroot , True)
```

6) Remove the zipped file from DBFS (if you don't want to keep it) and check the content of the folder.

```
# use if you like to remove the .zip file, otherwise keep it.  
dbutils.fs.rm("/FileStore/tables/" + fileroot + ".zip")
```

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

2) Exploring logs data

In this exercise you will be reducing and joining large datasets, which can take a bit of time. You may wish to perform the labs below using a smaller dataset, consisting of only a few of the web log files, rather than all of them. Remember that you can specify a wildcard; e.g. `textFile("/FileStore/logs/*6")` would include only file names ending with the digit 6, while `textFile("/FileStore/logs/*6*")` will correspond to file names containing the digit 6 etc.

7) Using map-reduce, count the number of requests from each user.

- (a) Use map to create a Pair RDD with the user ID as the key, and the integer 1 as the value. (The user ID is the third field in each line.) Your data will look something like this:

(<i>userid</i> , 1)
(<i>userid</i> , 1)
(<i>userid</i> , 1)
...

- (b) Use reduce sum the values for each user ID. Your RDD data will be similar to:

(<i>userid</i> , 5)
(<i>userid</i> , 7)
(<i>userid</i> , 2)
...

List logs files and select one of them to check the structure of the files.

```
dbutils.fs.ls("/FileStore/tables/logs" )
```

```
1 dbutils.fs.ls("/FileStore/tables/logs" )

Out[144]: [FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-15.log', name='2013-09-15.log', size=521343, modificationTime=1675019176000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-16.log', name='2013-09-16.log', size=484079, modificationTime=1675019158000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-17.log', name='2013-09-17.log', size=527399, modificationTime=1675019170000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-18.log', name='2013-09-18.log', size=485105, modificationTime=1675019166000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-19.log', name='2013-09-19.log', size=508553, modificationTime=1675019146000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-20.log', name='2013-09-20.log', size=492134, modificationTime=1675019168000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-21.log', name='2013-09-21.log', size=489117, modificationTime=1675019140000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-22.log', name='2013-09-22.log', size=535780, modificationTime=1675019192000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-23.log', name='2013-09-23.log', size=501768, modificationTime=1675019172000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-24.log', name='2013-09-24.log', size=489344, modificationTime=1675019159000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-25.log', name='2013-09-25.log', size=487857, modificationTime=1675019152000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-26.log', name='2013-09-26.log', size=523186, modificationTime=1675019145000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-27.log', name='2013-09-27.log', size=506195, modificationTime=1675019167000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-28.log', name='2013-09-28.log', size=492600, modificationTime=1675019183000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-29.log', name='2013-09-29.log', size=481479, modificationTime=1675019141000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-09-30.log', name='2013-09-30.log', size=525367, modificationTime=1675019199000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-10-01.log', name='2013-10-01.log', size=484920, modificationTime=1675019200000),
FileInfo(path='dbfs:/FileStore/tables/logs/2013-10-02.log', name='2013-10-02.log', size=537150, modificationTime=1675019137000)]
```

```
dbutils.fs.head("/FileStore/tables/logs/2013-09-15.log" )
```

```
1 dbutils.fs.head("/FileStore/tables/logs/2013-09-15.log" )

[Truncated to first 65536 bytes]
Out[145]: '3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 14417 "http://www.loudacre.com" "Loudacre Mobile Browser iFruit 1"\n3.94.78.5 - 69827 [15/Sep/2013:23:58:36
+0100] "GET /theme.css HTTP/1.0" 200 3576 "http://www.loudacre.com" "Loudacre Mobile Browser iFruit 1"\n19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00277.html HTTP/1.0" 200 15517 "htt
p://www.loudacre.com" "Loudacre Mobile Browser Ronin S1"\n19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /theme.css HTTP/1.0" 200 13353 "http://www.loudacre.com" "Loudacre Mobile Browser Ronin
S1"\n129.133.56.105 - 2489 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 18500 "http://www.loudacre.com" "Loudacre Mobile Browser Sorrento F00L"\n129.133.56.105 - 2489 [15/Sep/2013:2
3:58:34 +0100] "GET /theme.css HTTP/1.0" 200 12295 "http://www.loudacre.com" "Loudacre Mobile Browser Sorrento F00L"\n217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /ronin_s4_sales.html HTTP/
1.0" 200 845 "http://www.loudacre.com" "Loudacre Mobile Browser MeeToo 1.0"\n217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /theme.css HTTP/1.0" 200 738 "http://www.loudacre.com" "Loudacre Mo
bile Browser MeeToo 1.0"\n217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /code.js HTTP/1.0" 200 938 "http://www.loudacre.com" "Loudacre Mobile Browser MeeToo 1.0"\n217.150.149.167 - 4712 [15/S
ep/2013:23:56:06 +0100] "GET /ronin_s4.jpg HTTP/1.0" 200 5552 "http://www.loudacre.com" "Loudacre Mobile Browser MeeToo 1.0"\n209.151.12.34 - 45922 [15/Sep/2013:23:55:09 +0100] "GET /KBDOC-00259.html HT
TP/1.0" 200 19362 "http://www.loudacre.com" "Loudacre Mobile Browser Sorrento F11L"\n209.151.12.34 - 45922 [15/Sep/2013:23:55:09 +0100] "GET /theme.css HTTP/1.0" 200 17795 "http://www.loudacre.com" "Lo
udacre Mobile Browser Sorrento F11L"\n184.97.84.245 - 144 [15/Sep/2013:23:54:55 +0100] "GET /KBDOC-00052.html HTTP/1.0" 200 12499 "http://www.loudacre.com" "Loudacre CSR Browser"\n184.97.84.245 - 144 [1
5/Sep/2013:23:54:55 +0100] "GET /theme.css HTTP/1.0" 200 4979 "http://www.loudacre.com" "Loudacre CSR Browser"\n233.60.251.2 - 33908 [15/Sep/2013:23:51:43 +0100] "GET /KBDOC-00292.html HTTP/1.0" 200 477
9 "http://www.loudacre.com" "Loudacre Mobile Browser Ronin S2"\n233.60.251.2 - 33908 [15/Sep/2013:23:51:43 +0100] "GET /theme.css HTTP/1.0" 200 15871 "http://www.loudacre.com" "Loudacre Mobile Browser
Ronin S2"\n160.134.139.204 - 51340 [15/Sep/2013:23:50:11 +0100] "GET /KBDOC-00016.html HTTP/1.0" 200 1156 "http://www.loudacre.com" "Loudacre Mobile Browser MeeToo 3.0"\n160.134.139.204 - 51340 [15/Sep/
2013:23:50:11 +0100] "GET /theme.css HTTP/1.0" 200 13083 "http://www.loudacre.com" "Loudacre Mobile Browser MeeToo 3.0"\n19.209.18.222 - 13392 [15/Sep/2013:23:48:48 +0100] "GET /KBDOC-00083.html HTTP/1.
0" 200 4398 "http://www.loudacre.com" "Loudacre Mobile Browser Titanic 2200"\n19.209.18.222 - 13392 [15/Sep/2013:23:48:48 +0100] "GET /theme.css HTTP/1.0" 200 4054 "http://www.loudacre.com" "Loudacre M
obile Browser Titanic 2200"\n230.220.223.28 - 12643 [15/Sep/2013:23:48:42 +0100] "GET /KBDOC-00135.html HTTP/1.0" 200 1063 "http://www.loudacre.com" "Loudacre Mobile Browser Sorrento F20L"\n230.220.223.
28 - 12643 [15/Sep/2013:23:48:42 +0100] "GET /theme.css HTTP/1.0" 200 16864 "http://www.loudacre.com" "Loudacre Mobile Browser Sorrento F20L"\n129.253.238.61 - 120 [15/Sep/2013:23:48:35 +0100] "GET /KBD
OC-00053.html HTTP/1.0" 200 7108 "http://www.loudacre.com" "Loudacre CSR Browser"\n129.253.238.61 - 120 [15/Sep/2013:23:48:35 +0100] "GET /theme.css HTTP/1.0" 200 19169 "http://www.loudacre.com" "Louda
```

`\n` means New Line;

it is end of one line
(record) and start of
another line (record).
It shows a line
delimited file

```
...0" 200 3576 "http://www.
...loudacre.com" "Loudacre
le Browser Ronin S1"\n129
00L"\n129.133.56.105 - 248
712 [15/Sep/2013:23:56:06
```

```
myrdd1 = sc.textFile('/FileStore/tables/logs/2014-03*')

myrdd1.take(2)
```

We can clearly see that each line of the log file is one element of myrdd1

```
1 myrdd1.take(2)
```

Out[153]: ['34.28.1.122 - 65255 [01/Mar/2014:23:57:51 +0100] "GET /ifruit_3a_sales.html HTTP/1.0" 200 11416 "http://www.loudacre.com" "Loudacre Mobile Browser Titanic 2300"',
'34.28.1.122 - 65255 [01/Mar/2014:23:57:51 +0100] "GET /theme.css HTTP/1.0" 200 14933 "http://www.loudacre.com" "Loudacre Mobile Browser Titanic 2300"']

If you look at the first line, how you can extract **user ID** '65255' and then make a pair with number 1 like (65255,1)? If you assume white space (or " ") is the delimiter then index of the "65255" is [2]. Why?

String	Index
'34.28.1.122	Index is 0 or [0]
-	Index is 1 or [1]
65255	Index is 2 or [2]

```
myrdd2 = myrdd1.map(lambda s:(s.split(" ")[2] , 1))
```

```
1 myrdd2.take(10)
```

Out[156]: [('65255', 1),
('65255', 1),
('65255', 1),
('65255', 1),
('96482', 1),
('96482', 1),
('152', 1),
('152', 1),
('54322', 1),
('54322', 1)]

After concluding the map transformation, we can use reduce transformation.

```
myrdd3 = myrdd2.reduceByKey(lambda a,b: a + b)
```

```
1 myrdd3 = myrdd2.reduceByKey(lambda a,b: a + b)
2
3 myrdd3.take(10)
```

► (1) Spark Jobs

```
Out[157]: [('197', 126),
('91', 130),
('58327', 4),
('102728', 4),
('72770', 4),
('132', 152),
('117273', 4),
('39', 114),
('127', 134),
('67523', 2)]
```

8) Use countByKey to determine how many users visited the site for each frequency. That is, how many users visited once, twice, three times and so on.

(a) Use map to reverse the key and value, like this:

(5, <i>userid</i>)
(7, <i>userid</i>)
(2, <i>userid</i>)
...

```
myrdd4 = myrdd3.map(lambda pair :(pair[1] , pair[0]))
```

Cmd 40

```
1 myrdd4 = myrdd3.map(lambda pair :(pair[1] , pair[0]))
2
3 myrdd4.take(10)
```

► (1) Spark Jobs

```
Out[158]: [(126, '197'),
(130, '91'),
(4, '58327'),
(4, '102728'),
(4, '72770'),
(152, '132'),
(4, '117273'),
(114, '39'),
(134, '127'),
(2, '67523')]
```

(b) Use the `countByKey` action to return a collection of (frequency, user-count) pairs. `countDict` is no more an RDD, it is a dictionary (you can't use `take()` to see its content)

```
countDict = myrdd4.countByKey()  
  
sorted(countDict.items())
```

```
1  countDict = myrdd4.countByKey()  
2  
3  sorted(countDict.items())  
4  
5  #countDict
```

► (1) Spark Jobs

```
Out[171]: [(2, 11709),  
(3, 52),  
(4, 6580),  
(5, 19),  
(6, 2373),  
(7, 11),  
(8, 795),  
(9, 8),  
(10, 228),  
(11, 2),  
(12, 69),  
(14, 20),  
(16, 7),  
(86, 1),  
(94, 1),
```

- 9) Create an RDD where the user ID is the key, and the value is the list of all the IP addresses that user has connected from. (IP address is the first field in each request line.)

```
Cmd 36
1 myrdd1.take(2)

(1) Spark Jobs
Out[153]: [34.28.1.122 - 65255 [01/Mar/2014:23:57:51 +0100] "GET /ifruit_3a_sales.html HTTP/1.0" 200 11416 "http://www.loudacre.com" "Loudacre Mobile Browser Titanic 2300",
'34.28.1.122 - 65255 [01/Mar/2014:23:57:51 +0100] "GET /theme.css HTTP/1.0" 200 14933 "http://www.loudacre.com" "Loudacre Mobile Browser Titanic 2300"]
```

```
myrdd6 = myrdd1.map(lambda x:(x.split(" ")[2] , x.split(" ")[0]))
```

```
1 myrdd6 = myrdd1.map(lambda x:(x.split(" ")[2] , x.split(" ")[0]))
2
3 myrdd6.take(10)
```

```
(1) Spark Jobs
Out[172]: [('65255', '34.28.1.122'),
('65255', '34.28.1.122'),
('65255', '34.28.1.122'),
('65255', '34.28.1.122'),
('96482', '240.181.17.144'),
('96482', '240.181.17.144'),
('152', '73.33.209.18'),
('152', '73.33.209.18'),
('54322', '141.4.199.31'),
('54322', '141.4.199.31')]
```

```
myrdd7 = myrdd6.groupByKey()
myrdd7.take(2)
```

```
Cmd 43
1 myrdd7 = myrdd6.groupByKey()
2
3 myrdd7.take(2)

(1) Spark
Out[178]: [(197, <pyspark.resultiterable.ResultIterable at 0x7f5cf8f45678>),
('91', <pyspark.resultiterable.ResultIterable at 0x7f5cf8f45ac0>)]
```

User id Collection of all IPs

```
myrdd8 = myrdd7.map(lambda x : (x[0] , list(x[1])))
```

```
myrdd8.take(2)
```

```
1 myrdd8 = myrdd7.map(lambda x : (x[0] , list(x[1])))
2
3 myrdd8.take(2)
```

► (1) Spark Jobs

User id

Out[177]: [('197',
['214.211.152.147',
'214.211.152.147',
'246.96.252.129',
'246.96.252.129',
'65.128.78.81',
'65.128.78.81',
'69.196.27.95',
'69.196.27.95',
'38.137.246.73',
'38.137.246.73',
'59.68.167.53',
'59.68.167.53',
'252.196.191.142',
'252.196.191.142',
'158.142.219.170',
'158.142.219.170',
'210.45.14.255',
'210.45.14.255',

IPs

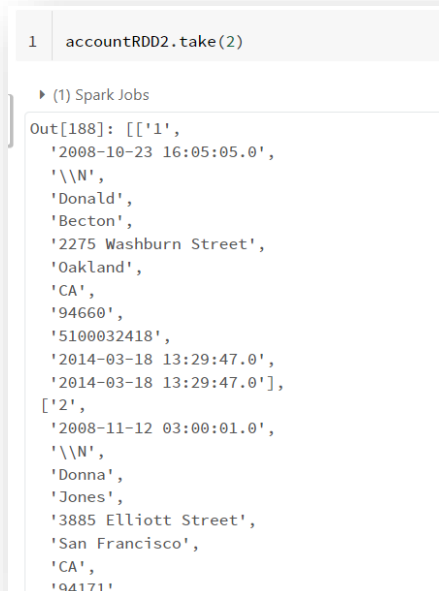
3) Join logs data with accounts data

Review the accounts data. The first field in each line is the user ID, which corresponds to the user ID in the web server logs. The other fields include account details such as creation date, first and last name and so on.

- 10) Join the accounts data with the logs data to produce a dataset keyed by user ID which contains the user account information and the number of website hits for that user.

(a) Create an RDD based on the accounts data consisting of key/value-array pairs: (userid, [values...])

```
accountRDD1 = sc.textFile('/FileStore/tables/accounts/part-*.')  
accountRDD1.take(2)
```



```
1 accountRDD2.take(2)  
  
▶ (1) Spark Jobs  
Out[188]: [['1',  
'2008-10-23 16:05:05.0',  
'\\N',  
'Donald',  
'Becton',  
'2275 Washburn Street',  
'Oakland',  
'CA',  
'94660',  
'5100032418',  
'2014-03-18 13:29:47.0',  
'2014-03-18 13:29:47.0'],  
['2',  
'2008-11-12 03:00:01.0',  
'\\N',  
'Donna',  
'Jones',  
'3885 Elliott Street',  
'San Francisco',  
'CA',  
'94171',
```

```
accountRDD2 = accountRDD1.map(lambda x: x.split(','))  
accountRDD3 = accountRDD2.map(lambda x: (x[0] , x))  
accountRDD3.take(3)
```

```

3 accountRDD2 = accountRDD1.map(lambda x: x.split(','))
4
5 accountRDD3 = accountRDD2.map(lambda x: (x[0] , x))
6
7 accountRDD3.take(3)

```

► (1) Spark Jobs

```

Out[187]: [('1',
['1',
'2008-10-23 16:05:05.0',
'\\N',
'Donald',
'Becton',
'2275 Washburn Street',
'Oakland',
'CA',
'94660',
'5100032418',
'2014-03-18 13:29:47.0',
'2014-03-18 13:29:47.0']),
('2',
['2',
'2008-11-12 03:00:01.0',
'\\N',
'Donna',
'Jones',
'3885 Elliott Street',
'San Francisco',

```

(b) Join the Pair RDD with the set of user-id/hit-count pairs calculated in the first step (it was myrdd3).

```
accountRDD4 = accountRDD3.join(myRDD3)
```

```
accountRDD4.take(2)
```

```

Out[189]: [('24',
([24',
'2008-11-13 00:08:34.0',
'\\N',
'Gwendolyn',
'Waters',
'4030 Leisure Lane',
'Sacramento',
'CA',
'94283',
'9164156733',
'2014-03-18 13:29:47.0',
'2014-03-18 13:29:47.0'],
128)),
('33',
(['33',
'2008-12-04 11:11:21.0',
'\\N',
'Marcia',
'Prince',
'2681 Rivendell Drive',

```

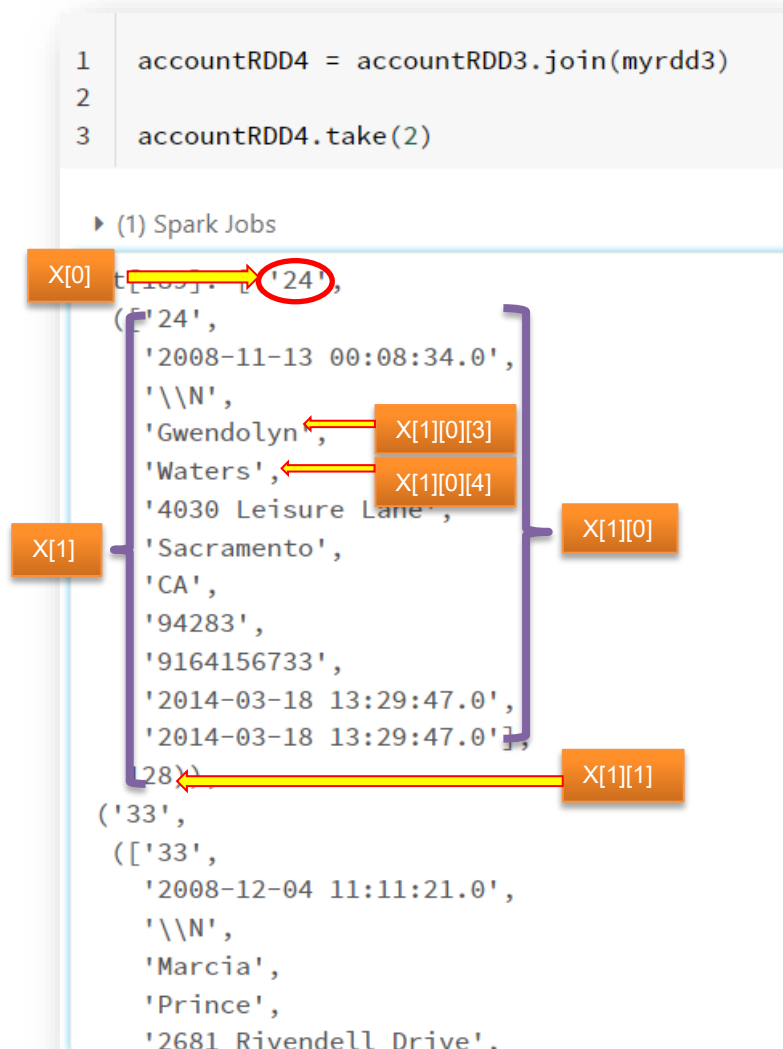
User id 24

With these characteristics

128 times visited the website

- (c) Display the user ID, hit count, and first name (3rd value) and last name (4th value) for the first 5 elements.

```
accountRDD5 = accountRDD4.map(lambda x: x[0] + " " + str(x[1][1]) + " " +  
x[1][0][3] + " " + x[1][0][4] + "\n")
```



```
accountRDD5.take(5)
```

```
Cmd 51
```

```
1 accountRDD5.take(5)
```

► (1) Spark Jobs

```
Out[191]: ['24 128 Gwendolyn Waters\n',  
'33 136 Marcia Prince\n',  
'56 142 Terrell Hardiman\n',  
'70 122 Debra Newman\n',  
'74 124 Terry Todd\n']
```

References and Resources:

<https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html>

<https://sparkbyexamples.com/pyspark-rdd/>