University of Salford, MSc Data Science

Module: Big Data Tools and Techniques

Date: Trimester 2, 2024-2025

Session: Workshop Week 2

Topic: Databricks File System (DBFS)

Tools: Databricks Community Edition

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

Objectives:

After completing this workshop, you will be able to:

- Work with Local Filesystem and DBFS

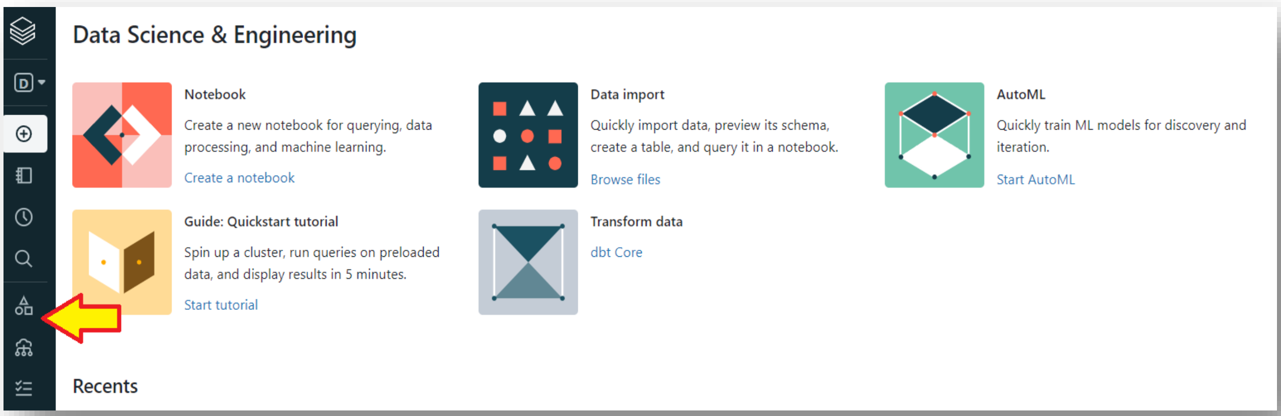
Table of Contents

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.



Data Science & Engineering

Notebook
Create a new notebook for querying, data processing, and machine learning.
Create a notebook

Data import
Quickly import data, preview its schema, create a table, and query it in a notebook.
Browse files

AutoML
Quickly train ML models for discovery and iteration.
Start AutoML

Guide: Quickstart tutorial
Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.
Start tutorial

Transform data
dbt Core

Recents

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.

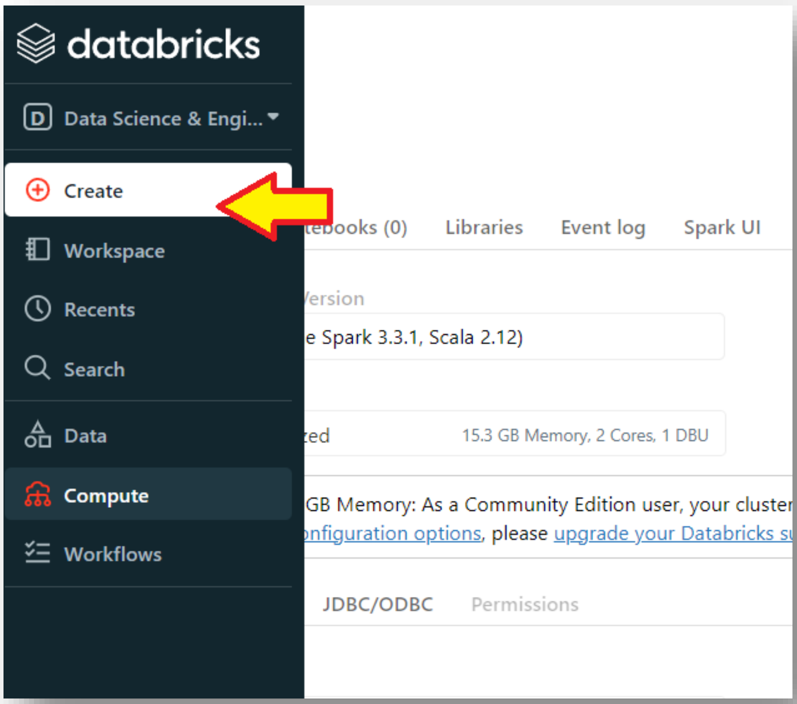
Part 2: Create a Python notebook

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

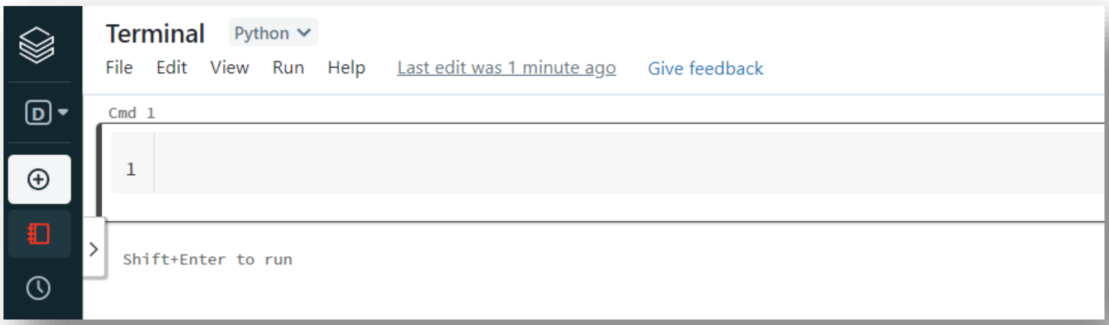
https://docs.databricks.com/notebooks/notebooks-use.html

for an overview of how to use a Databricks notebook.

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


databricks
Data Science & Engi...
Create
Workspace
Recents
Search
Data
Compute
Workflows
tebooks (0)
Libraries
Event log
Spark UI
Version
e Spark 3.3.1, Scala 2.12)
ted
15.3 GB Memory, 2 Cores, 1 DBU
GB Memory: As a Community Edition user, your cluster
onfiguration options, please upgrade your Databricks su
JDBC/ODBC
Permissions

2. Name your notebook, e.g. "Week 2", 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:

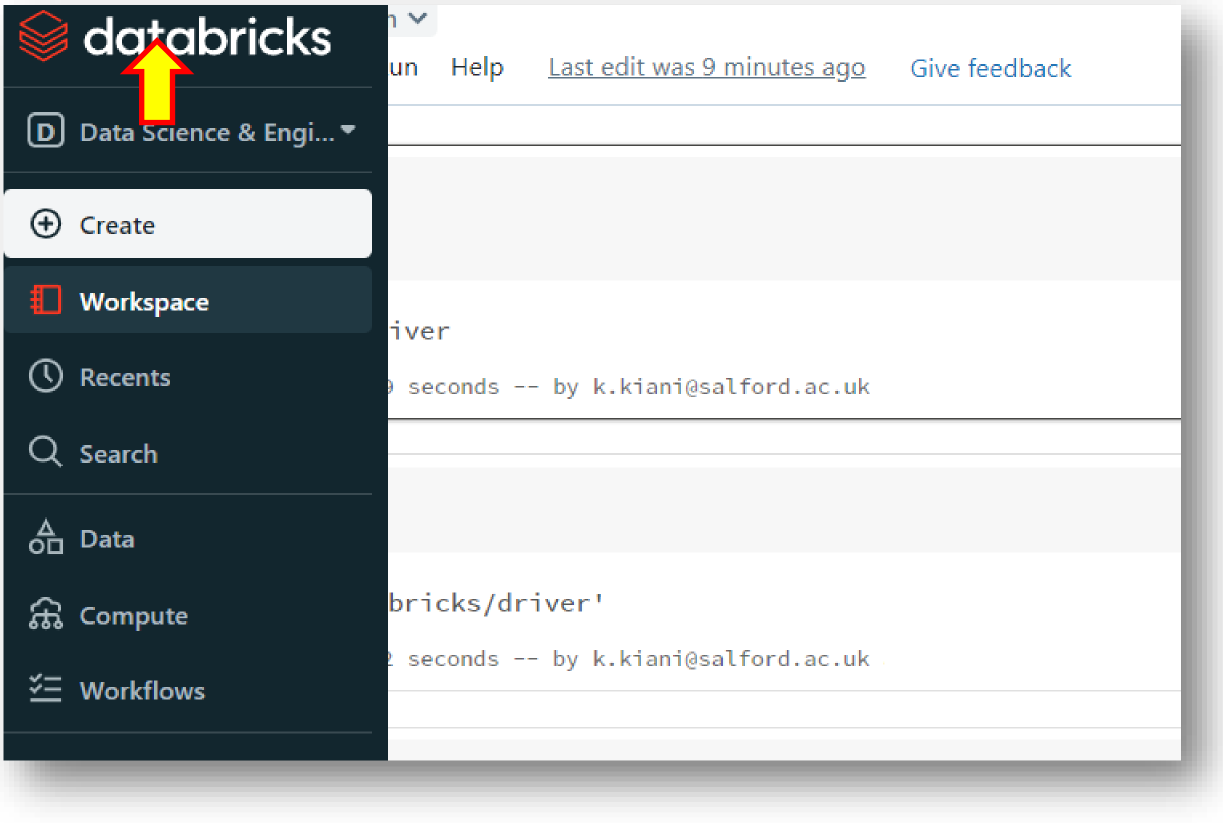


Part 3: Filesystem of a Local Driver Node Vs Databricks Filesystem (DBFS)

When you create a cluster for yourself in Databricks, the platform assigns you a temporary volume storage attached to the running cluster. This is Local Driver Node and it has a Local filesystem. It is called Local because it has attached to your cluster. When you terminate the cluster files that you have stored in it will be deleted.

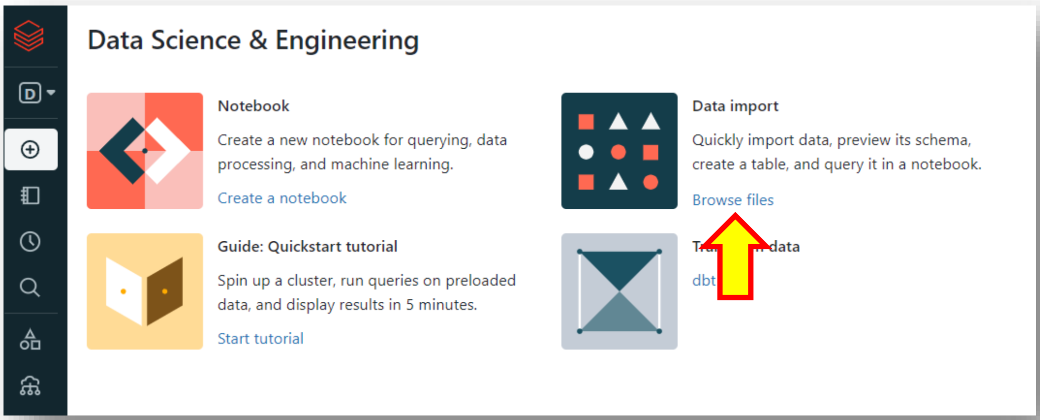
But Databricks File system (DBFS) is a distributed file system mounted into a Databricks workspace and available on Databricks clusters. This means if you put a file in DBFS and then terminate the cluster and then create a new cluster still you will have the file. Let's examine this.

Click on the databricks icon to redirect to the Home Page.

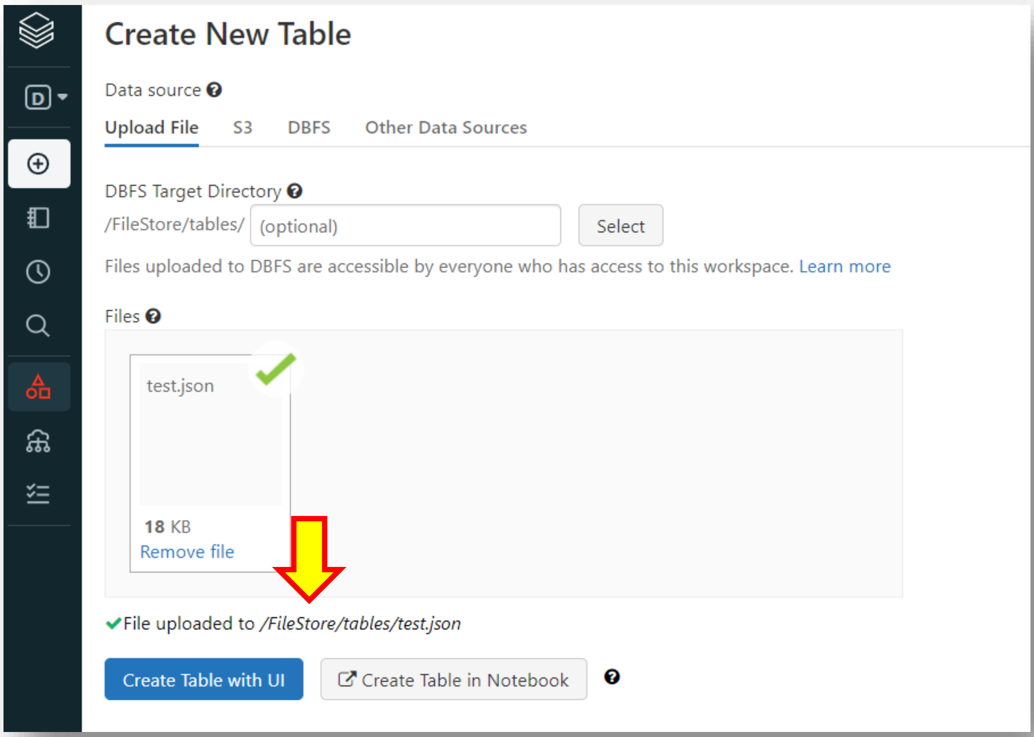


```
iver
seconds -- by k.kiani@salford.ac.uk
bricks/driver'
seconds -- by k.kiani@salford.ac.uk
```

1. Download the test.zip file from the Blackboard.
2. Unzip the test.zip and you must have test.json file in your local machine (usually Downloads folder).
3. In the Home Page click on the **Browse files** and select "test.json".


Data Science & Engineering
Notebook
Create a new notebook for querying, data processing, and machine learning.
Create a notebook
Data import
Quickly import data, preview its schema, create a table, and query it in a notebook.
Browse files
Guide: Quickstart tutorial
Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.
Start tutorial
data
dbt

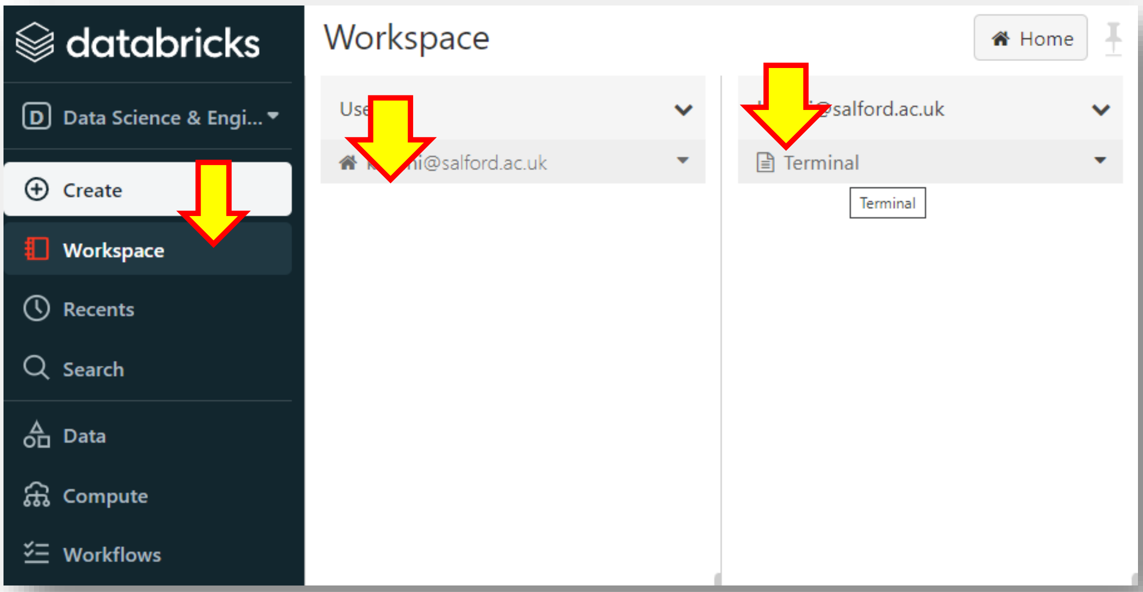
Automatically the data set will be uploaded to the “FileStore/tables” DBFS location



Files uploaded to DBFS are accessible by everyone who has access to this workspace.

✓File uploaded to */FileStore/tables/test.json*

Now, go back to the “Terminal” notebook.



Part 4: Using the `dbutils.fs` module to work with the Local and DBFS file systems

Before we continue working on this week's workshop, it is useful to have a quick introduction to libraries, modules, functions and classes in Python.

Libraries: In Python, a library is a collection of modules that provide pre-written functionalities to perform various tasks. Libraries save time and effort by offering reusable code that can be imported into your programs. If you have already taken Machine Learning & Data Mining, you will be familiar with libraries such as Scikit-Learn, pandas, Numpy, Matplotlib, etc.
Example: `dbutils` is a library provided by Databricks that provides utilities for working within Databricks

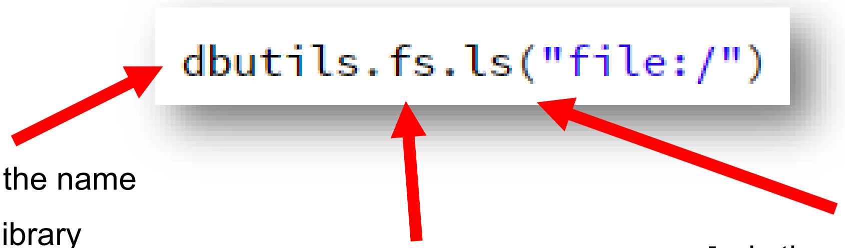
Modules: A module is a file containing Python code, which can define functions, classes, and variables. Modules allow you to organise the code within a library logically and makes it reusable.
Example: Within the `dbutils` library, there is a module named `fs` that deals with file system operations.

Functions: Functions are blocks of reusable code that perform a specific task. They are defined within modules and can be called from other parts of your program.

Example: Inside the `dbutils.fs` module, there is a function `ls` which lists the files within a specified directory.

Classes: Classes are a way to bundle data and functionality together. They allow you to create objects which have attributes (data) and methods (functions) that operate on that data. We will come back to this in future weeks.

So, the following line of code means:



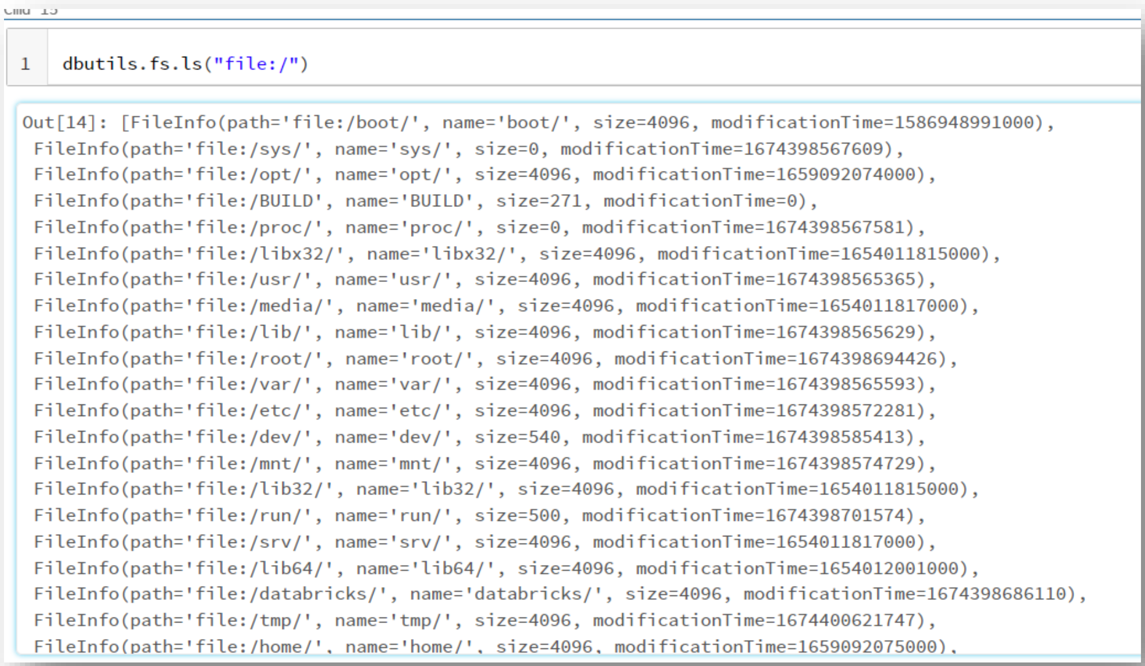
```
dbutils.fs.ls("file:/")
```

`dbutils` is the name of the library

`fs` is the name of module within the `dbutils` library

`ls` is the name of function we want to call within the `dbutils.fs` module

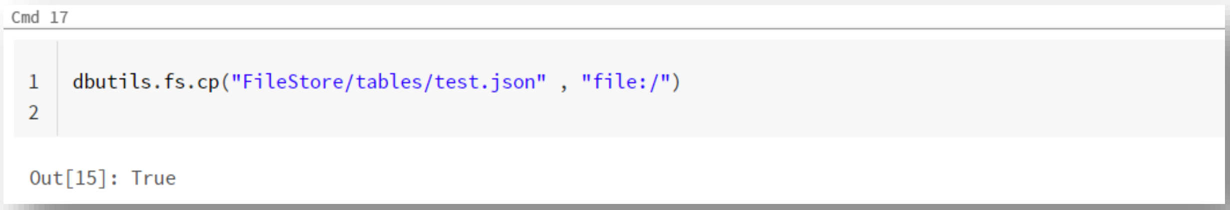
Open a new cell and type the following command to check the contents of the local filesystem. test.json is not in the local filesystem yet.



```
dbutils.fs.ls("file:/")
```

```
Out[14]: [FileInfo(path='file:/boot/', name='boot/', size=4096, modificationTime=1586948991000),
 FileInfo(path='file:/sys/', name='sys/', size=0, modificationTime=1674398567609),
 FileInfo(path='file:/opt/', name='opt/', size=4096, modificationTime=1659092074000),
 FileInfo(path='file:/BUILD', name='BUILD', size=271, modificationTime=0),
 FileInfo(path='file:/proc/', name='proc/', size=0, modificationTime=1674398567581),
 FileInfo(path='file:/libx32/', name='libx32/', size=4096, modificationTime=1654011815000),
 FileInfo(path='file:/usr/', name='usr/', size=4096, modificationTime=1674398565365),
 FileInfo(path='file:/media/', name='media/', size=4096, modificationTime=1654011817000),
 FileInfo(path='file:/lib/', name='lib/', size=4096, modificationTime=1674398565629),
 FileInfo(path='file:/root/', name='root/', size=4096, modificationTime=1674398694426),
 FileInfo(path='file:/var/', name='var/', size=4096, modificationTime=1674398565593),
 FileInfo(path='file:/etc/', name='etc/', size=4096, modificationTime=1674398572281),
 FileInfo(path='file:/dev/', name='dev/', size=540, modificationTime=1674398585413),
 FileInfo(path='file:/mnt/', name='mnt/', size=4096, modificationTime=1674398574729),
 FileInfo(path='file:/lib32/', name='lib32/', size=4096, modificationTime=1654011815000),
 FileInfo(path='file:/run/', name='run/', size=500, modificationTime=1674398701574),
 FileInfo(path='file:/srv/', name='srv/', size=4096, modificationTime=1654011817000),
 FileInfo(path='file:/lib64/', name='lib64/', size=4096, modificationTime=1654012001000),
 FileInfo(path='file:/databricks/', name='databricks/', size=4096, modificationTime=1674398686110),
 FileInfo(path='file:/tmp/', name='tmp/', size=4096, modificationTime=1674400621747),
 FileInfo(path='file:/home/', name='home/', size=4096, modificationTime=1659092075000),
```

Now copy test.json from DBFS to the local filesystem using the cp function which is in the dbutils.fs module.

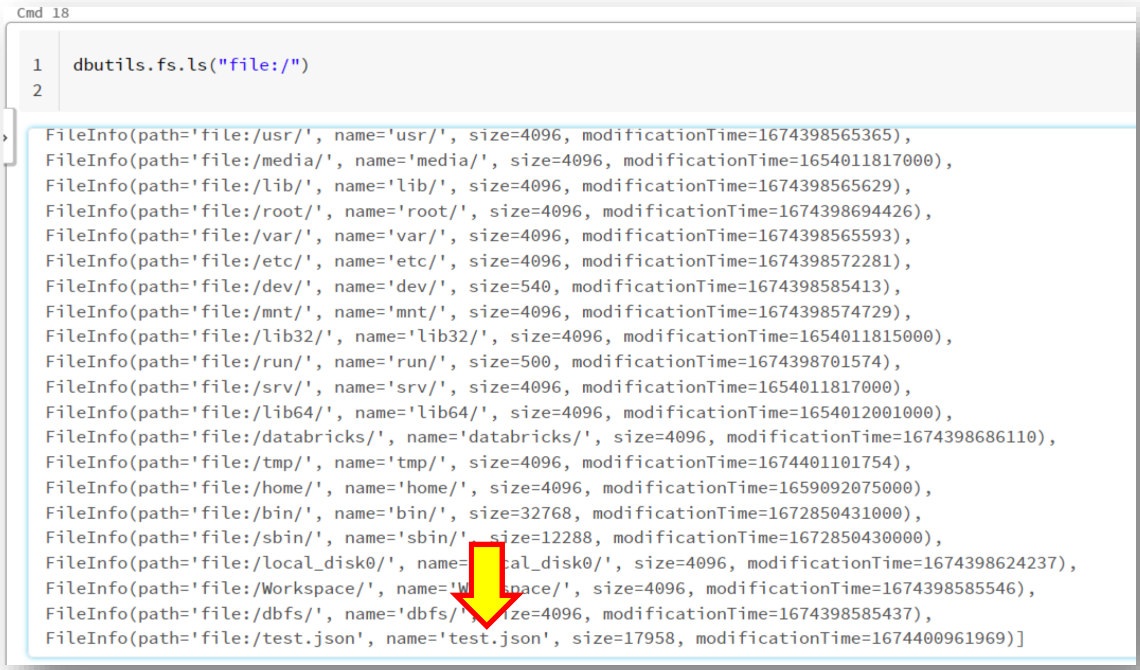


Cmd 17

```
dbutils.fs.cp("FileStore/tables/test.json" , "file:/")
```

```
Out[15]: True
```

If you check contents of the local fille system, test.json is in the filesystem now.



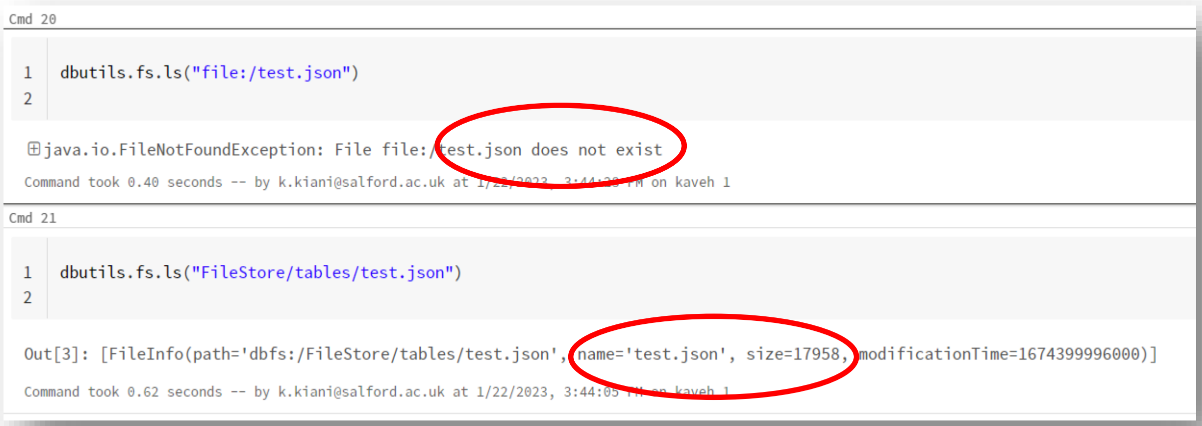
Cmd 18

```
dbutils.fs.ls("file:/")
```

```
FileInfo(path='file:/usr/', name='usr/', size=4096, modificationTime=1674398565365),
FileInfo(path='file:/media/', name='media/', size=4096, modificationTime=1654011817000),
FileInfo(path='file:/lib/', name='lib/', size=4096, modificationTime=1674398565629),
FileInfo(path='file:/root/', name='root/', size=4096, modificationTime=1674398694426),
FileInfo(path='file:/var/', name='var/', size=4096, modificationTime=1674398565593),
FileInfo(path='file:/etc/', name='etc/', size=4096, modificationTime=1674398572281),
FileInfo(path='file:/dev/', name='dev/', size=540, modificationTime=1674398585413),
FileInfo(path='file:/mnt/', name='mnt/', size=4096, modificationTime=1674398574729),
FileInfo(path='file:/lib32/', name='lib32/', size=4096, modificationTime=1654011815000),
FileInfo(path='file:/run/', name='run/', size=500, modificationTime=1674398701574),
FileInfo(path='file:/srv/', name='srv/', size=4096, modificationTime=1654011817000),
FileInfo(path='file:/lib64/', name='lib64/', size=4096, modificationTime=1654012001000),
FileInfo(path='file:/databricks/', name='databricks/', size=4096, modificationTime=1674398686110),
FileInfo(path='file:/tmp/', name='tmp/', size=4096, modificationTime=1674401101754),
FileInfo(path='file:/home/', name='home/', size=4096, modificationTime=1659092075000),
FileInfo(path='file:/bin/', name='bin/', size=32768, modificationTime=1672850431000),
FileInfo(path='file:/sbin/', name='sbin/', size=12288, modificationTime=1672850430000),
FileInfo(path='file:/local_disk0/', name='local_disk0/', size=4096, modificationTime=1674398624237),
FileInfo(path='file:/Workspace/', name='Workspace/', size=4096, modificationTime=1674398585546),
FileInfo(path='file:/dbfs/', name='dbfs/', size=4096, modificationTime=1674398585437),
FileInfo(path='file:/test.json', name='test.json', size=17958, modificationTime=1674400961969)]
```

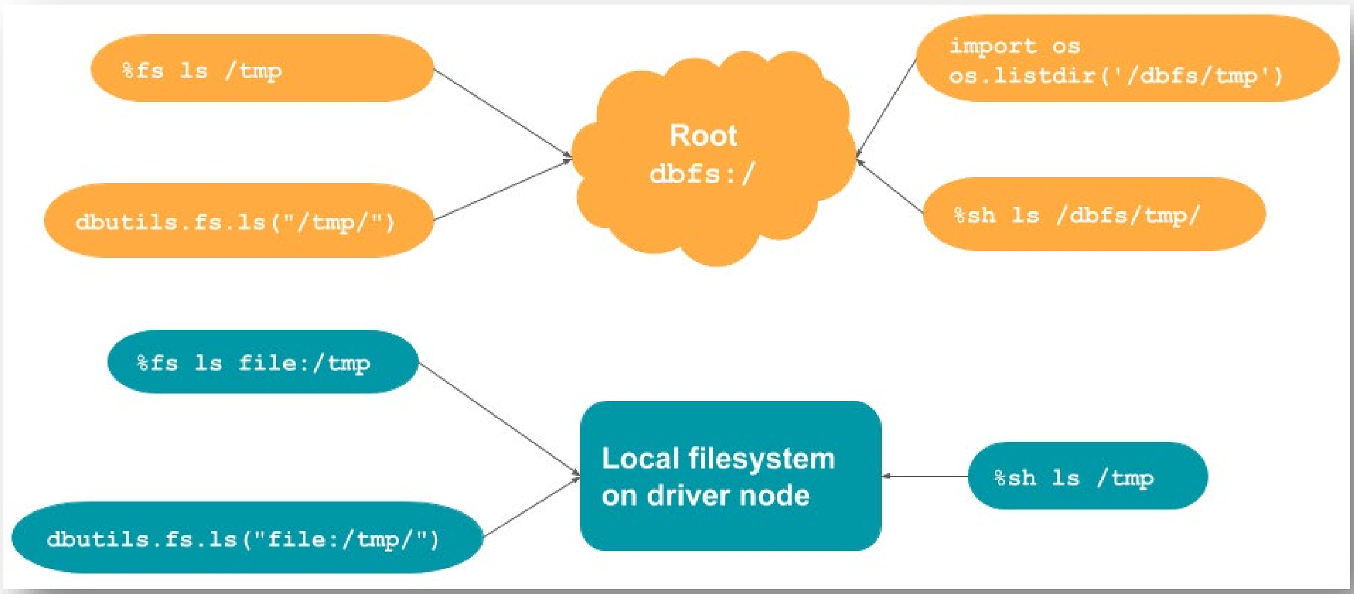
Now you have test.json in 2 locations, DBFS and the local filesystem. At this stage, if you terminate the cluster and then create a new cluster and run the two below commands, you will see that test.json is still in the DBFS but is not in the new local filesystem.

Note: Don't terminate your cluster in the workshop because it will take a long time for Databricks to create a new cluster for you. Check the existence of the test.json next time when you create a cluster.



```
Cmd 20

dbutils.fs.ls("file:/test.json")

⊞java.io.FileNotFoundException: File file:/test.json does not exist
Command took 0.40 seconds -- by k.kiani@salford.ac.uk at 1/22/2023, 3:44:20 PM on kaveh 1
Cmd 21

dbutils.fs.ls("FileStore/tables/test.json")

Out[3]: [FileInfo(path='dbfs:/FileStore/tables/test.json', name='test.json', size=17958, modificationTime=1674399996000)]
Command took 0.62 seconds -- by k.kiani@salford.ac.uk at 1/22/2023, 3:44:05 PM on kaveh 1
```

Following image shows how to access a directory (e.g. ***tmp*** directory) in two different filesystems with different means.



Source: https://docs.databricks.com/files/index.html

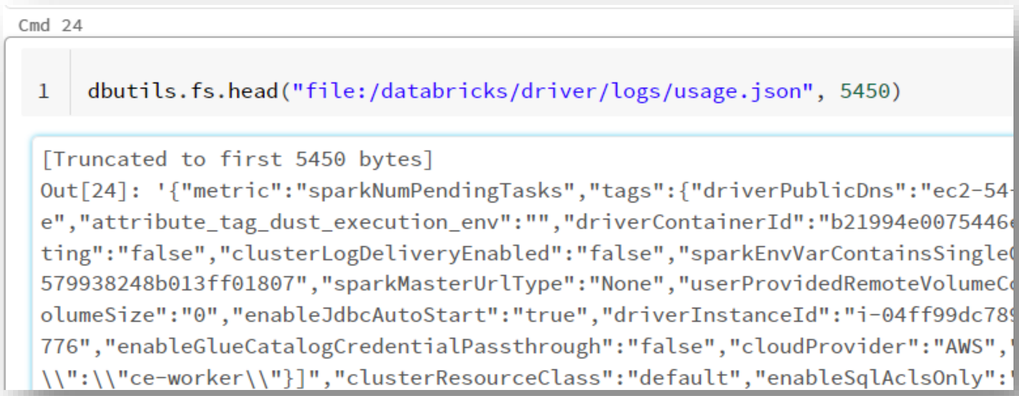
Also you can check this YouTube link for more clarification.

Part 5: Working with the DBFS using Python

In this part, we will explore the various data storage options available and do some (sequential) file manipulations using Python. This means we want to interact with the DBFS without Linux commands but with Python.

1. Use the **dbsutil.fs** utility to view the first 5450 bytes of the local file

/databricks/driver/logs/usage.json



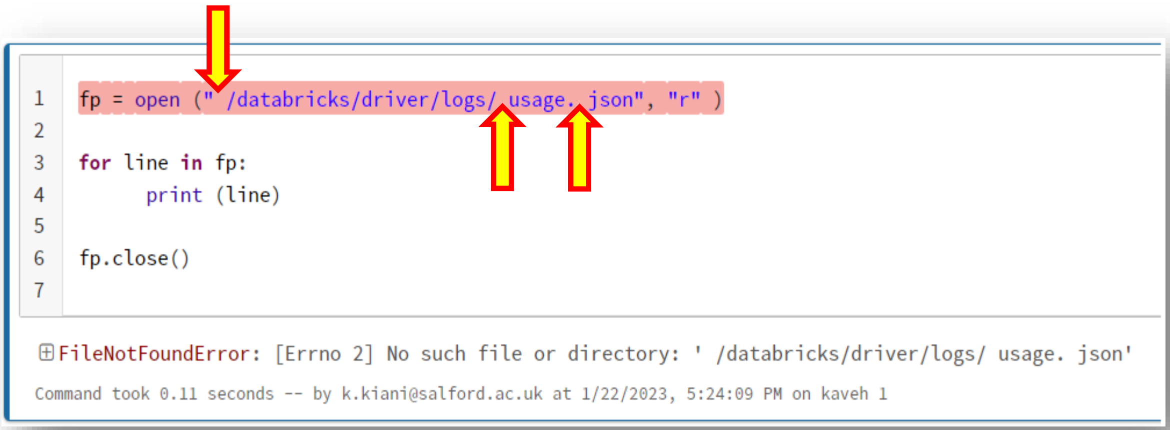
```
Cmd 24
dbutils.fs.head("file:/databricks/driver/logs/usage.json", 5450)

[Truncated to first 5450 bytes]
Out[24]: '{"metric":"sparkNumPendingTasks","tags":{"driverPublicDns":"ec2-54
e","attribute_tag_dust_execution_env":"","driverContainerId":"b21994e0075446
ting":"false","clusterLogDeliveryEnabled":"false","sparkEnvVarContainsSingle
579938248b013ff01807","sparkMasterUrlType":"None","userProvidedRemoteVolumeC
olumeSize":"0","enableJdbcAutoStart":"true","driverInstanceId":"i-04ff99dc78
776","enableGlueCatalogCredentialPassthrough":"false","cloudProvider":"AWS",
\\":\\"ce-worker\\"}]","clusterResourceClass":"default","enableSqlAclsOnly":
```

2. For small datasets, and a simple processing task, we can read and process a file locally line by line. The following code opens the local **logs/usage.json** file and reads it line by line. Since the file path is a string, it must be enclosed in quotes.

```
fp = open("/databricks/driver/logs/usage.json", "r")

for line in fp:
      print(line)
fp.close()
```

This opens your file for reading ("r") and creates a file pointer (fp) that you can use to access the file's contents. It uses a for loop to go through all the lines in the file and when finished, it closes the file.

Note: If you copy codes from the workshops' handout and paste them into the notebook, you must remove white spaces before running, especially strings inside the **" "**. Also, check the Python script indentations.



```
fp = open (" /databricks/driver/logs/ usage. json", "r" )

for line in fp:
      print (line)

fp.close()

FileNotFoundError: [Errno 2] No such file or directory: ' /databricks/driver/logs/ usage. json'
Command took 0.11 seconds -- by k.kiani@salford.ac.uk at 1/22/2023, 5:24:09 PM on kaveh 1
```

3. To introduce a bit of **reusability**, you can declare a cell which defines a variable containing your file path first, and then modify the cell defined in previous task to use the new variable. Later if you define another filepath your function will work

with the new filepath:

```
filepath = "/databricks/driver/logs/usage.json"
```

```
fp = open(filepath , "r" )

for line in fp:
       print(line)
fp.close()
```

4. To demonstrate the reusability, copy the sample logs/part-00000 file from DBFS to your local filesystem (so you can run your local data-based function):

```
dbutils.fs.cp("/databricks-datasets/sample_logs/part-00000","file:/tmp/sample_log.txt")
```

5. Now change the filepath variable to correspond to “file:/tmp/sample log.txt" and run the print lines cell again. Did the output change?

```
filepath = "/tmp/sample_log.txt"
```

```
fp = open(filepath , "r")

for line in fp:
       print(line)
fp.close()
```

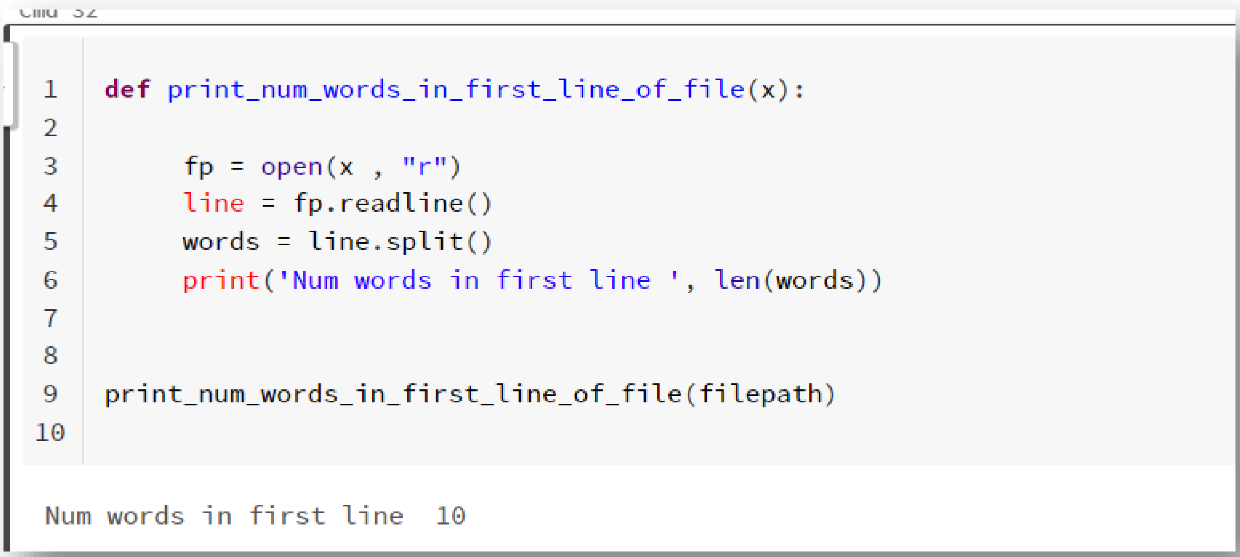
6. Python can count the lines in the file using a counter variable.

```
fp = open(filepath , "r")
counter = 0
for line in fp:
       counter += 1
fp.close()

print("Number of lines", counter)
```

7. Following is a new function which takes a file path and prints the number of objects separated by whitespace on the first line of the file? Let’s run this function on /tmp/sample log.txt.

```
filepath = "/tmp/sample_log.txt"
```

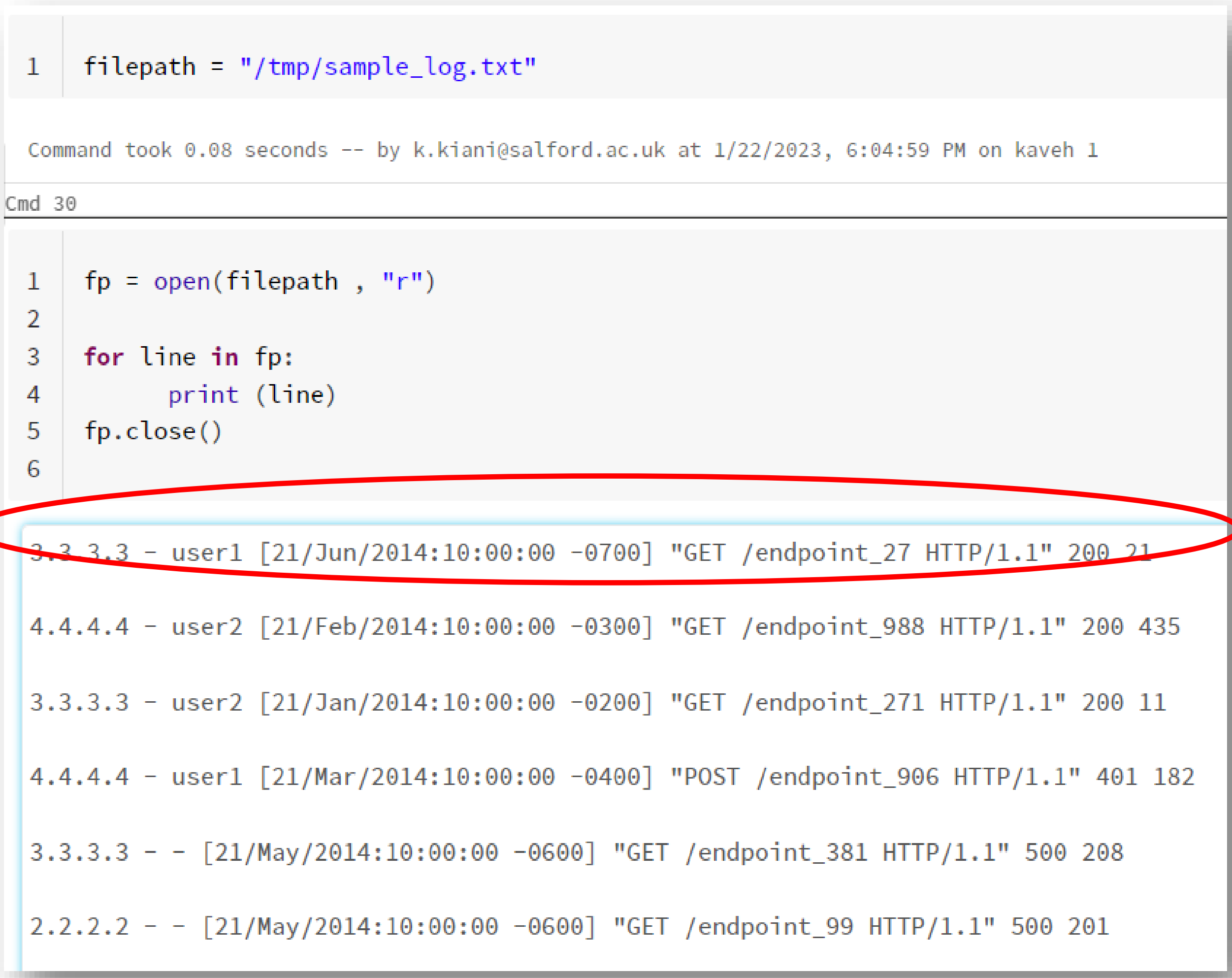
```
def print_num_words_in_first_line_of_file(x):

     fp = open(x , "r")
     line = fp.readline()
     words = line.split()
     print('Num words in first line', len(words))


print_num_words_in_first_line_of_file(filepath)
```



```
def print_num_words_in_first_line_of_file(x):

     fp = open(x , "r")
     line = fp.readline()
     words = line.split()
     print('Num words in first line ', len(words))


print_num_words_in_first_line_of_file(filepath)

```

```
Num words in first line  10
```

Answer is 10 but why? Let’s check the first line of the sample_log.txt



```
filepath = "/tmp/sample_log.txt"
```

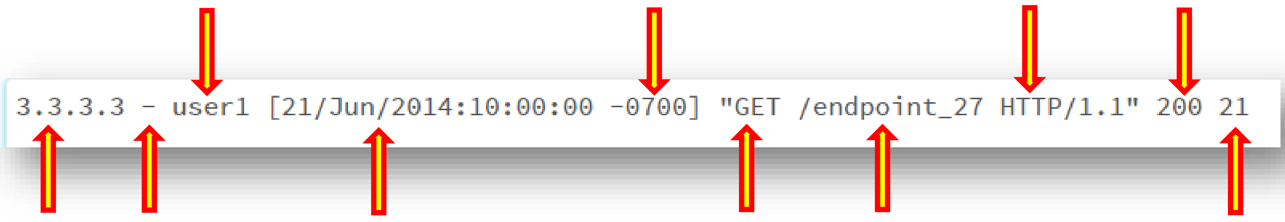
```
Command took 0.08 seconds -- by k.kiani@salford.ac.uk at 1/22/2023, 6:04:59 PM on kaveh 1
```

Cmd 30

```python
fp = open(filepath , "r")

for line in fp:
      print (line)
fp.close()

```

```
3.3.3.3 - user1 [21/Jun/2014:10:00:00 -0700] "GET /endpoint_27 HTTP/1.1" 200 21

4.4.4.4 - user2 [21/Feb/2014:10:00:00 -0300] "GET /endpoint_988 HTTP/1.1" 200 435

3.3.3.3 - user2 [21/Jan/2014:10:00:00 -0200] "GET /endpoint_271 HTTP/1.1" 200 11

4.4.4.4 - user1 [21/Mar/2014:10:00:00 -0400] "POST /endpoint_906 HTTP/1.1" 401 182

3.3.3.3 - - [21/May/2014:10:00:00 -0600] "GET /endpoint_381 HTTP/1.1" 500 208

2.2.2.2 - - [21/May/2014:10:00:00 -0600] "GET /endpoint_99 HTTP/1.1" 500 201
```

If you count the number of strings between white spaces, it will be 10.



```
3.3.3.3 - user1 [21/Jun/2014:10:00:00 -0700] "GET /endpoint_27 HTTP/1.1" 200 21
```

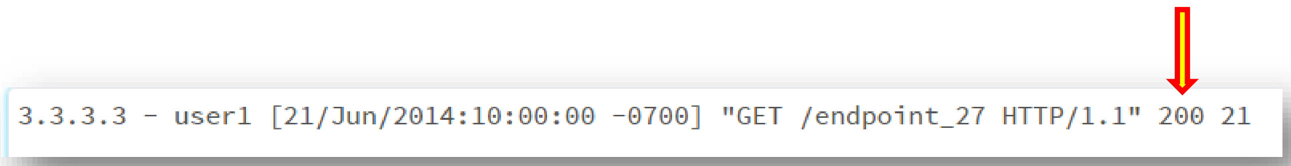
Part 6: Additional Challenges

Challenge 1

For the first challenge, write a function which counts the total number of words in a file. To help you, think about how we could combine the loop we used to count the number of lines in a file (on page 12) with the function we defined on page 13 to count the number of words on the first line of a file. Use this function on the **sample_log.txt** file.

Challenge 2

Each log entry contains an HTTP response code – these are three-digit codes issued by a server in response to a request from a client. These provide an easy way of communicating how the server responded to the request. The response code for a successful HTTP request is 200.



```
3.3.3.3 - user1 [21/Jun/2014:10:00:00 -0700] "GET /endpoint_27 HTTP/1.1" 200 21
```

Have a look at the below code. Can you work out what this code does?

```
fp = open(filepath , "r")
counter = 0
for line in fp:
      if line.split(" ")[8] == "200":
            counter += 1
fp.close()

print(counter)
```

The file we are working on is saved within the local file system. This means we can also work with the file using Linux commands, as we did last week.
Referring back to the workshop materials from last week, can you write a Linux command which will produce the same output as the above Python code?

References and Resources:

https://docs.databricks.com/files/index.html

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