

Table of Contents

1.	Prerequisite	3
2.	Install Spark in centos Linux – 60 Minutes(D)	9
3.	Analysis Using Spark- Overview	14
4.	Explore DataFrames using pyspark – 35 Minutes	19
5.	Explore DataFrame and Schema – 30 Minutes.....	24
6.	Parquet, Use User Defined Functions & SQL – 60 Minutes(D).....	29
7.	Explore Data with DataFrame Queries – 60 Minutes.....	45
8.	Explore Interactive Analysis with pySpark – 30 Minutes.....	56
9.	Explore Transformation and Action – 45 Minutes	62
10.	Explore with Pair RDD – 90 Minutes.....	69
11.	Explore Spark SQL – 45 Minutes.....	100
12.	Explore Pyspark Applications – 30 Minutes	111
13.	Installing Jupyter for Spark – 35 Minutes.(D)	113
14.	Explore Spark Two Nodes Cluster – 90 Minutes	Error! Bookmark not defined.
15.	Explore Spark on Hadoop YARN – 150 Minutes.....	Error! Bookmark not defined.
16.	Jobs Monitoring : Using Web UI – 45 Minutes.....	Error! Bookmark not defined.
17.	Persisting Data – 40 Minutes	Error! Bookmark not defined.
18.	Broadcast & Repartition – 45 Minutes	Error! Bookmark not defined.

- 19. **Spark – Hive Integration – 45 Minutes** **Error! Bookmark not defined.**
- 20. Spark integration with Hive Cluster (Local Mode)... **Error! Bookmark not defined.**
- 21. Annexure & Errata: **Error! Bookmark not defined.**
 - Unable to start spark shell with the following error **Error! Bookmark not defined.**
 - issue: Cannot assign requested address..... **Error! Bookmark not defined.**
 - Unable to start spark shell: **Error! Bookmark not defined.**
 - Yum repo config..... **Error! Bookmark not defined.**
- Issue in Pyspark: TypeError: Too many parameters for typing.Iterable; actual 2, expected 1*** **Error! Bookmark not defined.**
 - Issue: Traceback (most recent call last):..... **Error! Bookmark not defined.**
 - File "/usr/local/lib/python3.6/site-packages/notebook/notebookapp.py", line 2027, in init_server_extensions..... **Error! Bookmark not defined.**
 - Bash profile..... **Error! Bookmark not defined.**

Last Updated: 18 Aug 2023

1. Prerequisite

Using Docker Environment:

Install Docker environment according to your environment.

Create **sparko** container:

Let us create a common network for our spark nodes.

```
#docker network create --driver bridge spark-net
```

```
#docker run -it --name sparko --hostname sparko --privileged --network spark-net -v  
/Users/henrypotsangbam/software/:/Software -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081  
-p 8090:8090 -p 8888:8888 centos /usr/sbin/init
```

Download the required software

Apache Spark – Version 3.0.1 – Prebuilt for Apache Hadoop 3.2 and later.

File : [spark-3.0.1-bin-hadoop3.2.tgz](https://spark.apache.org/downloads.html) / [spark-3.2.1-bin-hadoop3.2.tgz](https://spark.apache.org/downloads.html)

Url : <https://spark.apache.org/downloads.html>

Download Location: (hadoop-3.2.2.tar.gz)

<https://hadoop.apache.org/releases.html>

python : 3.8

Inflate all require software in /opt folder.

JDK installation⑧) (jdk-8u45-linux-x64.tar.gz)

<https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html>

Steps for deploying two node spark cluster in docker:

- Create two containers using the above image
 - o Spark0
 - o Spark1
- Create a network to connect between these two nodes.

You should pull the following images from docker hub.(centos:7)

(base) Henrys-MacBook-Air:~ henrypotsangbam\$ docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
spark-kafka	latest	a0c16a0aebc1	2 days ago	3.84GB
centos	7	7e6257c9f8d8	4 weeks ago	203MB

Let us create a common network for our two nodes cluster.

```
#docker network create --driver bridge spark-net
```

Create first node, spark0 container

5 Learning Spark - PySpark

```
#docker run -it --name sparko --hostname spark0 --privileged -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081 -p 8090:8090 centos /usr/sbin/init
```

Docker command with Host folder mounted. (Skip)

```
# docker run -it --name spark0 --hostname spark0 --privileged --network spark-net -v /Volumes/Samsung_T5/software/:/Software -v /Volumes/Samsung_T5/software/install/:/opt -v /Volumes/Samsung_T5/software/data/:/data -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081 -p 8090:8090 centos:7 /usr/sbin/init
```

To connect a **running** container to an existing user-defined bridge, use the docker network connect command. If the sparko is already started without the network being attached execute the following else skip the following step.

```
$ docker network connect spark-net sparko
```

Start the first container:

```
#docker start sparko
```

Connect to you first node:

```
# docker exec -it sparko /usr/bin/bash
```

Start the second node and perform the installation as specify in the first lab.

6 Learning Spark - PySpark

```
#docker run -dit --name spark1 -p 8082:8081 -p 4041:4040 --network spark-net --entrypoint  
/bin/bash centos:7
```

or

Docker command with Host folder mounted. (Skip)

```
# docker run -it --name spark1 --hostname spark1 --privileged --network spark-net -v  
/Users/henrypotsangbam/Documents/Docker:/opt -p 4041:4040 -p 8082:8081 centos:7 /usr/sbin/init
```

Confirm the Network status:

Inspect the network and find the IP addresses of the two containers

```
#docker network inspect spark-net
```

```
"ConfigOnly": false,
"Containers": [
    "e34277dc489b480dd6fb4528c84a333a40a322c054be3877da827005a84f593e": {
        "Name": "spark1",
        "EndpointID": "d5c0c550a1b58782b590909ffb56e3c66826d4ccb046853fbe7c4f5646239038",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
    },
    "ef2232096b600d78c553dc7cab22b8b0bbdb7065d2d59eb57637d36699839ac7": {
        "Name": "spark0",
        "EndpointID": "e3bfa88d7a67b2f6b82cd2543637eee22abcaeeb9c8ff1c00415d47308d96da8",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
    }
],
```

Ensure to substitute IPs accordingly.

Attach to the spark-master container and test its communication to the spark-worker container using both it's IP address and then using its container name.

```
# ping spark0
```

```
[root@ef2232096b60 /]# ping spark0
PING spark0 (172.19.0.2) 56(84) bytes of data.
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=3 ttl=64 time=0.062 ms
^C
--- spark0 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2060ms
rtt min/avg/max/mdev = 0.062/0.076/0.104/0.019 ms
[root@ef2232096b60 /]# ping spark1
PING spark1 (172.19.0.3) 56(84) bytes of data.
64 bytes from spark1.spark-net (172.19.0.3): icmp_seq=1 ttl=64 time=0.202 ms
64 bytes from spark1.spark-net (172.19.0.3): icmp_seq=2 ttl=64 time=0.204 ms
^C
--- spark1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.202/0.203/0.204/0.001 ms
#[root@ef2232096b60 /]#
```

Docker container for hadoop cluster.

```
#docker run -it --name hadoopo --privileged -p 8088:8088 -p 9870:9870 -p 9864:9864 -p 8032:8032
-p 8188:8188 -p 8020:8020 --hostname hadoopo centos /usr/sbin/init
```

```
# docker run -it --name spark2 --hostname spark2 --privileged -v
/Users/henrypotsangbam/Documents/Software:/Software -p 8081:8080 -p 4042:4040 centos
/usr/sbin/init
```

2. Install Spark in centos Linux – 60 Minutes(D)

Extract jdk and rename the folder to jdk.

```
#cd /Software  
#tar -xvf jdk*.tar.gz -C /opt  
#cd /opt  
#mv jdk* jdk
```

Extract spark-X.X.o-bin-hadoop.X.tar to /opt. Ensure to replace the spark file name according to your environment.

```
#cd /Software  
# tar -xvf spark-x-bin-hadoop.x.tgz.gz -C /opt/
```

It should create a folder spark.X, rename to spark folder.

```
#cd /opt  
#mv spark* spark
```

Set the JAVA_HOME and initialize the PATH variable.

Open the profile and update with the following statements.

```
#vi ~/.bashrc  
  export JAVA_HOME=/opt/jdk  
  export PATH=$PATH:$JAVA_HOME/bin  
  export PATH=$PATH:/opt/spark/bin
```

Install Python version 3.6.

Python 3.6 or above is required to run PySpark program.

```
#yum install -y python3.8
```

Type bash, to initialize the profile.

Go to the installation directory:

```
# pyspark
```

```
[root@spark0 python]# pyspark
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/06/27 04:54:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

    _/\_   _/\_ _/\_ /_ \_
   / \ \_ \ \_ \_ \ \_ \_ \
  /_ / .\_,\_,\_,\_,\_,\_,\_
  /_ \_ \_ \_ \_ \_ \_ \_ \_ \
version 3.1.2

Using Python version 3.6.8 (default, Nov 16 2020 16:55:22)
Spark context Web UI available at http://spark0:4040
Spark context available as 'sc' (master = local[*], app id = local-1624769653946).
SparkSession available as 'spark'.
>>> 
```

Enter the following command in the pyspark console to count each alphabets.

Notebook - Hello Pyspark

Let's look at a short example where we read in a text file as a DataFrame, show a sample of the strings read, and count the total number of lines in the file. This simple example illustrates the use of the high-level Structured APIs, which we will cover later. The `show(10, false)` operation on the DataFrame only displays the first 10 lines without truncating; by default the `truncate` Boolean flag is true

// Code Begin. ****

```
#Display the spark version  
spark.version  
#Reading file in the specified location  
strings = spark.read.text("/opt/spark/README.md")  
#show the first 10 lines  
strings.show(10, truncate=False)  
#Get the count of lines in the file  
strings.count()  
  
// Code Ends Here#####
```

```
In [9]: #Display the spark version  
spark.version  
Out[9]: '3.3.0'  
  
In [13]: #Reading file in the specified location  
strings = spark.read.text("/opt/spark/README.md")  
  
In [11]: #show the first 10 lines  
strings.show(10, truncate=False)  
+-----+  
| value |  
+-----+  
| # Apache Spark  
|  
| Spark is a unified analytics engine for large-scale data processing. It provides  
| high-level APIs in Scala, Java, Python, and R, and an optimized engine that  
| supports general computation graphs for data analysis. It also supports a  
| rich set of higher-level tools including Spark SQL for SQL and DataFrames,  
| pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing,  
| and Structured Streaming for stream processing.  
|  
| <https://spark.apache.org/>  
+-----+  
only showing top 10 rows  
  
In [12]: #Get the count of lines in the file  
strings.count()  
Out[12]: 124
```

Web UI of the spark shell can be accessed using the following URL. You can click on each tab and verify the screen.

<http://sparko:4040/jobs/> or
<http://localhost:4040/jobs/>

The screenshot shows the PySparkShell application UI with the following details:

- Spark Version:** 3.3.0
- Jobs Tab:** Selected
- Completed Jobs (3):**
 - Job ID 3: count at NativeMethodAccessorImpl.java:0, Submitted: 2023/07/02 10:14:08, Duration: 58 ms, Stages: 1/1 (1 skipped), Tasks: 1/1 (1 skipped)
 - Job ID 2: count at NativeMethodAccessorImpl.java:0, Submitted: 2023/07/02 10:14:08, Duration: 70 ms, Stages: 1/1, Tasks: 1/1
 - Job ID 1: showString at NativeMethodAccessorImpl.java:0, Submitted: 2023/07/02 10:13:54, Duration: 0.3 s, Stages: 1/1, Tasks: 1/1
- User:** root
- Total Uptime:** 16 min
- Scheduling Mode:** FIFO
- Completed Jobs:** 3
- Failed Jobs:** 1

Further details on this web UI will be discussed later.

You have successfully installed spark for local development.

Note: You can start with default cores by the following command.

pyspark --master local[*]

-----Lab Ends Here-----

3. Analysis Using Spark- Overview

Demonstrate this after installation lab.

This Tutorial will demonstrate how we can use spark for data analytics - Overview.

Scenario - Upload the log file of a web access and find the following:

Which IP request url the most?

What is the max byte return from a server?

How many requests receive from each client?

Import the necessary packages.

```
from pyspark.sql import Row
from pyspark.sql.functions import col
```

Let us initialize log entries, that will be used for analysis.

```
data = [('64.242.88.10 -- [07/Mar/2004:16:05:49 -0800] "GET
/twiki/bin/edit/Main/Double_bounce_sender?topicparent>Main.ConfigurationVariables HTTP/1.1"
401 12846'), \
('64.242.88.10 -- [07/Mar/2004:16:06:51 -0800] "GET
/twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523'), \
('94.242.88.10 -- [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations
HTTP/1.1" 401 12851')]
```

Convert the log entries into RDD

```
rdd = spark.sparkContext.parallelize(data)
```

Let us have a look how the RDD or data look like

```
rdd.collect()
```

I would like to access the column using a name. So Let us intitialize RDD with a schema colum.

```
parts = rdd.map(lambda l: l.split(" "))
log = parts.map(lambda p: Row(ip=p[0], ts=(p[3]), atype = p[5], url=p[6] , code = p[8] , byte = p[9]))
```

Have I split the record fields correctly? Let us review the transformation

```
parts.collect()
```

I would like to use the Dataframe API. Let us convert from RDD to DF

```
# Infer the schema, and register the DataFrame as a table.
mylog = spark.createDataFrame(log)
```

Have a view of our DF!

```
mylog.take(2)
```

Now, we have the data in spark, let us answer the queries we have define in the first paragraphs.

Which IP request url the most?

```
mylog.groupBy("ip").count().orderBy(col("count").desc()).first()  
Row(ip=u'64.242.88.10', count=2)
```

What is the max byte return from a server?

```
mylog.select("ip","byte").orderBy("byte").show()  
mylog.select("ip","byte").orderBy("byte").first()
```

How many request receives from each client?

```
mylog.select("ip","url").groupBy("ip").count().show()
```

```
In [20]: from pyspark.sql import Row
from pyspark.sql.functions import col

In [35]: data = [('64.242.88.10 -- [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12846',
('64.242.88.10 -- [07/Mar/2004:16:06:51 -0800]"GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523',
('94.242.88.10 -- [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851')

In [36]: rdd = spark.sparkContext.parallelize(data)

In [37]: rdd.collect()

Out[37]: ['64.242.88.10 -- [07/Mar/2004:16:05:49 -0800] "GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1" 401 12846',
'64.242.88.10 -- [07/Mar/2004:16:06:51 -0800] "GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1" 200 4523',
'94.242.88.10 -- [07/Mar/2004:16:30:29 -0800] "GET /twiki/bin/attach/Main/OfficeLocations HTTP/1.1" 401 12851']

In [28]: parts = rdd.map(lambda l: l.split(" "))
log = parts.map(lambda p: Row(ip=p[0], ts=(p[3]), atype = p[5], url=p[6] , code = p[8] , byte = p[9]))

In [29]: parts.collect()

Out[29]: [['64.242.88.10',
'_',
'_',
'[07/Mar/2004:16:05:49',
'-0800]',
'"GET',
'/twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables',
'HTTP/1.1',
'401',
'12846'],
['64.242.88.10']]
```

```
In [30]: mylog = spark.createDataFrame(log)

In [31]: mylog.take(2)

Out[31]: [Row(ip='64.242.88.10', ts='[07/Mar/2004:16:05:49', atype='"GET', url='/twiki/bin/edit/Main/Double_bounce_sender?topicparent>Main.ConfigurationVariables', code='401', byte='12846'),
Row(ip='64.242.88.10', ts='[07/Mar/2004:16:06:51', atype='"GET', url='/twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2', code='200', byte='4523')]

In [32]: mylog.groupBy("ip").count().orderBy(col("count").desc()).first()
Row(ip=u'64.242.88.10', count=2)

Out[32]: Row(ip='64.242.88.10', count=2)

In [33]: mylog.select("ip","byte").orderBy("byte").show()
mylog.select("ip","byte").orderBy("byte").first()

+-----+---+
|      ip| byte|
+-----+---+
|64.242.88.10|12846|
|94.242.88.10|12851|
|64.242.88.10| 4523|
+-----+---+


Out[33]: Row(ip='64.242.88.10', byte='12846')

In [34]: mylog.select("ip","url").groupBy("ip").count().show()

+-----+---+
|      ip|count|
+-----+---+
|64.242.88.10|    2|
|94.242.88.10|    1|
```

----- Lab Ends Here -----

4. Explore DataFrames using pyspark – 35 Minutes

Following features of Spark will be demonstrated here using pyspark:

- Loading json file.
- Understand its schema
- Select required fields.
- Apply filter.

Create a text file **users.json** which contains sample data as listed below in **data** folder:

```
{"name":"Alice", "pcode":"94304"}  
{"name":"Brayden", "age":30, "pcode":"94304"}  
{"name":"Carla", "age":19, "pcode":"10036"}  
{"name":"Diana", "age":46}  
{"name":"Etienne", "pcode":"94104"}
```

Start pyspark-shell

```
#pyspark
```

Initiate the pyspark-shell from the folder which you have created the above file.

```
#Define the data location file  
dFile = "/opt/data/users.json"
```

```
#Read the users json file as a dataframe.  
usersDF = spark.read.json(dFile)
```

```
#Find out the schema of the uploaded file  
usersDF.printSchema()
```

As shown below, three fields will be displayed according to the json fields specified in the text file.

```
In [11]: #Define the data location file  
dFile = "/opt/data/users.json"
```

```
In [2]: #Read the users json file as a dataframe.  
usersDF = spark.read.json(dFile)
```

```
In [4]: #Find out the schema of the uploaded file  
usersDF.printSchema()
```

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

```
#Let us find out the first 3 records to have a sample data.  
users = usersDF.take(3)  
print(users)
```

```
In [6]: users = usersDF.take(3)
print(users)

[Row(age=None, name='Alice', pcode='94304'), Row(age=30, name='Brayden', pcode='94304'), Row(age=19, name='Carla',
pcode='10036')]
```

usersDF.show()

```
In [7]: usersDF.show()
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Etienne	94104

Out of the three fields, we are interested in only name and age fields. So, let us create a dataframe with only these two fields and apply a filter expression in which only person greater than 20 years are there in the dataframe.

```
nameAgeDF = usersDF.select("name", "age")
nameAgeOver20DF = nameAgeDF.where("age > 20")
nameAgeOver20DF.show()
```

```
In [8]: nameAgeDF = usersDF.select("name", "age")
nameAgeOver20DF = nameAgeDF.where("age > 20")
nameAgeOver20DF.show()
```

```
+-----+---+
|    name|age |
+-----+---+
|Brayden| 30|
| Diana| 46|
+-----+---+
```

```
usersDF.select("name", "age").where("age > 20").show()
```

```
In [10]: # Apply the above transformation in a single line
usersDF.select("name", "age").where("age > 20").show()
```

```
+-----+---+
|    name|age |
+-----+---+
|Brayden| 30|
| Diana| 46|
+-----+---+
```

You can also combine the functions as shown above. You will get the same result.

----- Lab Ends Here -----

5. Explore DataFrame and Schema – 30 Minutes

You will understand the following:

- Defining schema and mapping to dataframe while loading json file.
- Saving the dataframe to json file.

Create an input text file `/opt/data/people.csv`

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

Execute the following in the pyspark-shell.

Import and define the Structure.

```
#Define the Data file Location.
dFile = "/opt/data/people.csv"
```

```
#Import and define the Structure.
from pyspark.sql.types import *
columnsList = [
```

```
StructField("pcode", StringType()),  
StructField("lastName", StringType()),  
StructField("firstName", StringType()),  
StructField("age", IntegerType())]  
peopleSchema = StructType(columnsList)
```

Schema of the file is defined before loading the file in spark session.

```
#Specify the schema along with the loading instruction.  
usersDF = spark.read.option("header","true").schema(peopleSchema).csv(dFile)
```

Let us have a view of the schema of the file.

```
usersDF.printSchema()
```

```
In [2]: #Define the Data file Location.  
dFile = "/opt/data/people.csv"  
  
In [4]: #Import and define the Structure.  
from pyspark.sql.types import *  
columnsList = [  
    StructField("pcode", StringType()),  
    StructField("lastName", StringType()),  
    StructField("firstName", StringType()),  
    StructField("age", IntegerType())]  
peopleSchema = StructType(columnsList)  
  
In [5]: #Specify the schema along with the loading instruction.  
usersDF = spark.read.option("header","true").schema(peopleSchema).csv(dFile)  
  
In [6]: usersDF.printSchema()  
  
root  
|-- pcode: string (nullable = true)  
|-- lastName: string (nullable = true)  
|-- firstName: string (nullable = true)  
|-- age: integer (nullable = true)
```

// As shown above, age is an Integer, which will be mapped to String by default in case it's not defined in the structure schema.

Let us perform projection, we are interested in two fields only. You can use select method for this.

```
nameAgeDF = usersDF.select("firstname","age")  
nameAgeDF.show()
```

```
In [7]: nameAgeDF = usersDF.select("firstname","age")
nameAgeDF.show()
```

```
+-----+-----+
|firstname| age|
+-----+-----+
| Grace | null |
| Alan   | 32   |
| Ada    | 28   |
| Charles| 49   |
| Niklaus| 48   |
+-----+-----+
```

You can save the dataframe consisting of First Name and age to a file.

```
# Save the dataframe in a folder - /opt/data/ages
destFolder = "/opt/data/ages"
nameAgeDF.write.json(destFolder)
```

Open a terminal and verify the file. It will create a folder by the name **ages** and inside that output will be there.

Change the location to the directory of data folder.

```
#cd /opt/data
```

Have a view of files inside the folder **ages**. Output may varies depending on your environment.

```
#tree ages/
```

Print the file. Replace the file with that of your system.

#more ages/part-00000-2e18ce82-62a9-466c-9fdc-981c70c0192c-c000.json

```
[root@spark0 data]# tree ages/
ages/
└── part-00000-2e18ce82-62a9-466c-9fdc-981c70c0192c-c000.json
    └── _SUCCESS

0 directories, 2 files
[root@spark0 data]# more ages/part-00000-2e18ce82-62a9-466c-9fdc-981c70c0192c-c000.json
{"firstname": "Grace"}
{"firstname": "Alan", "age": 32}
{"firstname": "Ada", "age": 28}
{"firstname": "Charles", "age": 49}
{"firstname": "Niklaus", "age": 48}
[root@spark0 data]#
```

Assignment:

Try selecting first name and Last name of the data and save in a separate file.

----- Lab Ends Here -----

6. Parquet, Use User Defined Functions & SQL – 60 Minutes(D)

Notebook: Parquet , Use User Defined Functions & SQL

Let us write as json file into a parquet and read it using dataframe.

Open a spark-shell.

Read a json file and then save it to a parquet file.

```
peopleDF = spark.read.json("/Software/data/people.json")
```

```
peopleDF.printSchema()
```

```
peopleDF.show()
```

```
peopleDF.write.parquet (" /opt/data/people.parquet")
```

```
In [2]: peopleDF = spark.read.json("/Software/data/people.json")
```

```
In [3]: peopleDF.printSchema()
```

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

```
In [4]: peopleDF.show()
```

age	name
null	Michael
30	Andy
19	Justin

```
In [5]: peopleDF.write.parquet ("/opt/data/people.parquet")
```

Read the parquet file created above, Parquet files are self-describing so the schema is preserved. The result of loading a Parquet file is also a DataFrame

```
parquetFileDF = spark.read.parquet("/opt/data/people.parquet")
parquetFileDF.show()
```

```
In [6]: parquetFileDF = spark.read.parquet("/opt/data/people.parquet")
```

```
In [7]: parquetFileDF.show()
```

```
+---+---+
| age| name|
+---+---+
| null|Michael|
| 30 | Andy |
| 19 | Justin|
+---+---+
```

Parquet files can also be used to create a temporary view and then used in SQL statements

```
parquetFileDF.createOrReplaceTempView("parquetFile")
```

```
namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN  
13 AND 19")
```

```
namesDF.show()
```

```
In [8]: parquetFileDF.createOrReplaceTempView("parquetFile")
```

```
In [9]: namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")
```

```
In [15]: namesDF.show()
```

```
+----+  
| name|  
+----+  
|Justin|  
+----+
```

List the folders to view the parquet file created.

```
!ls -lt /opt/data/people.parquet/
```

```
In [51]: !ls -lt /opt/data/people.parquet/
```

```
total 4  
-rw-r--r-- 1 root root 0 Apr 22 10:48 _SUCCESS  
-rw-r--r-- 1 root root 738 Apr 22 10:48 part-00000-5bef6875-5754-4923-a1a8-bcd9287d68ea-c000.snappy.parquet
```

Next read another **json** file that contains movie information and then applied **snappy** compression while creating parquet file.

```
movieDF = spark.read.json("/Software/data/movies.json")
movieDF.write.option("compression",
"snappy").mode("overwrite").parquet("/opt/data/par_movies")
```

list the files.

```
!ls /opt/data/par_movies
```

```
In [17]: movieDF = spark.read.json("/Software/data/movies.json")
In [19]: movieDF.write.option("compression", "snappy").mode("overwrite").parquet("/opt/data/par_movies")
In [20]: !ls /opt/data/par_movies
part-00000-1a500d9a-c9bf-4ac5-af47-93be3672ba83-c000.snappy.parquet _SUCCESS
```

You can also view information about the parquet file using parquet tools.

Install the tools.

```
!pip install parquet-tools
```

```
In [21]: !pip install parquet-tools
Collecting parquet-tools
  Downloading parquet_tools-0.2.15-py3-none-any.whl (31 kB)
Collecting boto3<2.0.0,>=1.34.11 (from parquet-tools)
  Downloading boto3-1.34.40-py3-none-any.whl (139 kB)
    139.3/139.3 kB 1.3 MB/s eta 0:00:00a 0:00:01
Collecting colorama<0.5.0,>=0.4.6 (from parquet-tools)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting halo<0.0.32,>=0.0.31 (from parquet-tools)
  Using cached halo-0.0.31.tar.gz (11 kB)
  Preparing metadata (setup.py) ... done
INFO: pip is looking at multiple versions of parquet-tools to determine which version is compatible with other requirements. This could take a while.
Collecting parquet-tools
  Downloading parquet_tools-0.2.14-py3-none-any.whl (31 kB)
Collecting halo<0.0.30,>=0.0.29 (from parquet-tools)
  Downloading halo-0.0.29-py3-none-any.whl (10 kB)
Requirement already satisfied: pandas>=1 in /usr/local/lib64/python3.8/site-packages (from parquet-tools) (2.0.3)
Collecting pyarrow[parquet]
```

To view the data insides the file. You need to substitute the parquet file with that of the above.

```
!parquet-tools show /opt/data/par_movies/part-00000-2119ab8b-593a-402d-a685-9d24159aac96-c000.snappy.parquet
```

```
!pip install parquet-tools

!parquet-tools show /opt/data/par_movies/part-00000-2119ab8b-593a-402d-a685-9d24159aac96-c000.snappy.parquet
```

Certification	Label	PeakChart	Released	Title
{'ARIA': '2x Platinum', 'BPI': '2x Platinum', 'BVMF': None, 'CRIA': 'Diamond', 'IFPI': 'Gold', 'NVPI': 'Gold', 'RIAA': '8x Platinum', 'SNEP': 'Gold'}	9 Led Zeppelin		Atlantic	{'AUS': 9, 'UK': 6, 'US': 10} 01/12/1969
{'ARIA': '4xPlatinum', 'BPI': '4xPlatinum', 'BVMF': 'Platinum', 'CRIA': '9x Platinum', 'IFPI': 'Gold', 'NVPI': None, 'RIAA': 'Diamond', 'SNEP': 'Platinum'}	9 Led Zeppelin II		Atlantic	{'AUS': 1, 'UK': 1, 'US': 1} 10/22/1969
{'ARIA': None, 'BPI': 'Platinum', 'BVMF': 'Gold', 'CRIA': '3x Platinum', 'IFPI': 'Gold', 'NVPI': 'Gold', 'RIAA': '6x Platinum', 'SNEP': 'Platinum'}	9 Led Zeppelin III		Atlantic	{'AUS': 1, 'UK': 1, 'US': 1} 10/5/1970
{'ARIA': '9x Platinum', 'BPI': '6x Platinum', 'BVMF': '3x Gold', 'CRIA': '2xDiamond', 'IFPI': '2x Platinum', 'NVPI': 'Platinum', 'RIAA': 'Diamond', 'SNEP': '2x Platinum'}	9 Led Zeppelin IV		Atlantic	{'AUS': 2, 'UK': 1, 'US': 2} 11/8/1971
{'ARIA': None, 'BPI': 'Platinum', 'BVMF': 'Gold', 'CRIA': None, 'IFPI': None, 'NVPI': None, 'RIAA': 'Diamond', 'SNEP': 'Gold'}	3 Houses of the Holy		Atlantic	{'AUS': 1, 'UK': 1, 'US': 1} 03/28/1973

To get the metadata about the file.

```
!parquet-tools inspect /opt/data/par_movies/part-00000-2119ab8b-593a-402d-a685-9d24159aac96-c000.snappy.parquet
```

```
In [62]: !parquet-tools inspect /opt/data/par_movies/part-00000-2119ab8b-593a-402d-a685-9d24159aac96-c000.snappy.parquet
#####
file meta data #####
created_by: parquet-mr version 1.12.2 (build 77e30c8093386ec52c3cfa6c34b7ef3321322c94)
num_columns: 14
num_rows: 9
num_row_groups: 1
format_version: 1.0
serialized_size: 2951
```

As you can see above, there are 14 columns and 9 rows in the file.

Next.

Define UDF and invoke in the Query.

Load CSV data and convert into Dataframe

```
df = spark.read.format("csv") \  
    .option("header", "true") \  
    .load("/Software/data/sfpd.csv")
```

Register as a Table, so that we can perform SQL operation on it.

```
df.createOrReplaceTempView("sfpd")
```

You can execute an SQL query using the above table.

```
sql("select * from sfpd").show(false)
```

```
In [26]: df = spark.read.format("csv") \
    .option("header", "true") \
    .load("/Software/data/sfpd.csv")
```

```
In [28]: df.createOrReplaceTempView("sfpd")
```

```
In [34]: sql("select * from sfpd").show(truncate=False)
```

IncidentNum	Category	PdDistrict	Resolution	Address	Descriptor	X	Y	DayOfWeek	Date	Time
PdId										
150467944	LARCENY/THEFT			TAYLOR ST	GRAND THEFT FROM LOCKED AUTO / OFARRELL ST			Thursday	05/28/2015	23:59
59	TENDERLOIN	NONE				-122.411328369311	37.7859963050476			(37.7859963050476)
476,				15046794406244						
150468629	VEHICLE THEFT				STOLEN TRUCK			Thursday	05/28/2015	23:59
59	MISSION	NONE		100 Block of PORTOLA DR		-122.444276468858	37.7484965077695			(37.7484965077695)
695,				15046862907025						
156133410	LARCENY/THEFT				PETTY THEFT OF PROPERTY			Thursday	05/28/2015	23:55
55	NORTHERN	NONE		1300 Block of POLK ST		-122.420422400634	37.7889233553137			(37.7889233553137)
137,				15613341006372						
150478769	VEHICLE THEFT				STOLEN AUTOMOBILE			Thursday	05/28/2015	23:50
50	CENTRAL	NONE		500 Block of UNION ST		-122.407974943839	37.8006466385111			(37.8006466385111)
111,				15047876907021						

The **date** field in this dataset is a String of the form “mm/dd/yy”. We are going to create a function to extract the year from the date field. There are two ways to use user defined functions with Spark DataFrames. You can define it inline and use it within DataFrame operations or use it in SQL queries.

We can then find answers to questions such as: What is the number of incidents by year?

Define a function that extracts characters after the last ‘/’ from the string.

```
from dateutil.parser import parse
def getyear(s):
    dt = parse(s)
    year = dt.year
    return year
```

Register the function in the SQL Context

```
from pyspark.sql import *
sqlContext = SQLContext(sc)
sqlContext.udf.register("getyear",getyear)
```

Using the registered the udf in a SQL query, find the count of incidents by year.

```
incyearSQL = sql("SELECT getyear(sfpd.Date), count(sfpd.IncidntNum) AS
countbyyear " +
                  " FROM sfpd GROUP BY getyear(sfpd.Date)
ORDER BY countbyyear DESC")
incyearSQL.show()
```

```
In [63]: from dateutil.parser import parse
def getyear(s):
    dt = parse(s)
    year = dt.year
    return year
```

```
In [64]: from pyspark.sql import *
sqlContext = SQLContext(sc)
sqlContext.udf.register("getyear",getyear)
```

24/02/13 14:34:59 WARN SimpleFunctionRegistry: The function getyear replaced a previously registered function
.

```
Out[64]: <function __main__.getyear(s)>
```

Now you can use convertUDF() on a DataFrame column as a regular build-in function.

```
In [65]: incyearSQL = sql("SELECT getyear(sfpd.Date), count(sfpd.IncidntNum) AS countbyyear " +
                         " FROM sfpd GROUP BY getyear(sfpd.Date) ORDER BY countbyyear DESC")
```

```
In [66]: incyearSQL.show()
```

getyear(Date)	countbyyear
2015	9999

Find the addresses and resolution of VANDALISM incidents in 2015.

```
van2015 = sql("SELECT category,address,resolution, date FROM sfpd WHERE
getyear(date)='2015' " +
               " AND category='VANDALISM'")
van2015.show()
```

van2015.count()

```
In [72]: van2015 = sql("SELECT category,address,resolution, date FROM sfpd WHERE getyear(date)='2015' " +  
" AND category='VANDALISM'")  
van2015.show()
```

category	address	resolution	date
VANDALISM	600 Block of 42ND AV	NONE	05/28/2015
VANDALISM	HOLLIS ST / ELLIS ST	NONE	05/28/2015
VANDALISM	1300 Block of POL...	NONE	05/28/2015
VANDALISM	JACKSON ST / WALN...	NONE	05/28/2015
VANDALISM	LINCOLN WY / 43RD AV	NONE	05/28/2015
VANDALISM	200 Block of DUBO...	NONE	05/28/2015
VANDALISM	2300 Block of PAC...	NONE	05/28/2015
VANDALISM	BAY ST / KEARNY ST	NONE	05/28/2015
VANDALISM	100 Block of EDDY ST	ARREST, BOOKED	05/28/2015
VANDALISM	HARRISON ST / 13T...	NONE	05/28/2015
VANDALISM	GOUGH ST / OTIS ST	NONE	05/28/2015
VANDALISM	20TH ST / ILLINOI...	NONE	05/28/2015
VANDALISM	800 Block of HYDE ST	NONE	05/28/2015
VANDALISM	900 Block of BROD...	ARREST, BOOKED	05/28/2015
VANDALISM	1100 Block of GOU...	NONE	05/28/2015
VANDALISM	JOHNFKENNEDY DR /...	NONE	05/28/2015
VANDALISM	3200 Block of 22N...	ARREST, BOOKED	05/28/2015
VANDALISM	1900 Block of KEA...	NONE	05/28/2015
VANDALISM	3000 Block of 16T...	NONE	05/28/2015
VANDALISM	8TH ST / HOOPER ST	NONE	05/28/2015

only showing top 20 rows

```
In [74]: van2015.count()
```

```
Out[74]: 513
```

SQL Engine:

In this mode, end-users or applications can interact with Spark SQL directly to run SQL queries, without the need to write any code.

To start the JDBC/ODBC server, run the following in the Spark directory:

```
#cd /opt/spark  
#./sbin/start-thriftserver.sh
```

You can list the port running thriftserver using the following command.

```
#lsof -i:10000
```

```
[root@306633508e8b spark]# pwd  
/opt/spark  
[root@306633508e8b spark]# ./sbin/start-thriftserver.sh  
starting org.apache.spark.sql.hive.thriftserver.HiveThriftServer2, logging to /opt/spark/logs/spark--org.apache.spark.sql.hive.thriftserver.HiveThriftServer2-1-306633508e8b.out  
[root@306633508e8b spark]# lsof -i:10000  
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME  
java 467 root 352u IPv4 95886 0t0 TCP *:ndmp (LISTEN)  
[root@306633508e8b spark]#
```

If it display as shown above, then the services is up.

Now you can use beeline to test the Thrift JDBC/ODBC server:

#beeline

Connect to the JDBC/ODBC server in beeline with:

beeline> !connect jdbc:hive2://localhost:10000

It will ask for the username, enter any say – **henry** leave the password blank.

```
[root@spark0 spark]# beeline
Beeline version 2.3.9 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Enter username for jdbc:hive2://localhost:10000: henry
Enter password for jdbc:hive2://localhost:10000:
Connected to: Spark SQL (version 3.3.0)
Driver: Hive JDBC (version 2.3.9)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000> █
```

Let us create a table and execute a query on it.

```
CREATE TABLE PEOPLE (name String, age int) using org.apache.spark.sql.json  
OPTIONS (path "/Software/data/people.json");  
  
select * from people;
```

```
Transaction isolation: TRANSACTION_REPEATABLE_READ  
0: jdbc:hive2://localhost:10000> CREATE TABLE PEOPLE (name STring, age int) using org.apache.spark.sql.jso  
n OPTIONS (path "/software/people.json");  
+-----+  
| Result |  
+-----+  
+-----+  
No rows selected (1.866 seconds)  
0: jdbc:hive2://localhost:10000> select * from people;  
+-----+-----+  
| name | age |  
+-----+-----+  
| Michael | NULL |  
| Andy | 30 |  
| Justin | 19 |  
+-----+-----+  
3 rows selected (3.677 seconds)  
0: jdbc:hive2://localhost:10000> ■
```

----- Lab Ends Here -----

7. Explore Data with DataFrame Queries – 60 Minutes

We will understand the following in this lab:

- Join
- Accessing Columns in DF

Use the following data for this lab:

/opt/data/people.csv

```
PCODE,lastName,firstName,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

Load the people data and fetch column, age in the following way:

```
dataFile = "/opt/data/people.csv"
```

```
#Import and define the Structure.
from pyspark.sql.types import *
columnsList = [
    StructField("PCODE", StringType()),
    StructField("lastName", StringType()),
    StructField("firstName", StringType()),
    StructField("age", IntegerType())]
```

```
peopleSchema = StructType(columnsList)
```

```
peopleDF = spark.read.option("header","true").schema(peopleSchema).csv(dataFile)
```

```
In [12]: dataFile = "/opt/data/people.csv"
```

Reload this page

```
In [13]: #Import and define the Structure.
```

```
from pyspark.sql.types import *
columnsList = [
    StructField("pcode", StringType()),
    StructField("lastName", StringType()),
    StructField("firstName", StringType()),
    StructField("age", IntegerType())
]
peopleSchema = StructType(columnsList)
```

```
In [14]: peopleDF = spark.read.option("header","true").schema(peopleSchema).csv(dataFile)
```

Different way of fetching column in a dataframe. Here retrieves only **age**.

```
peopleDF["age"]
```

```
peopleDF.age
```

```
In [19]: print(peopleDF['age'])
```

```
Column<'age'>
```

```
In [16]: peopleDF.age
```

```
Out[16]: Column<'age'>
```

```
peopleDF.select(peopleDF["age"]).show()  
peopleDF.select(peopleDF.age).show()
```

```
In [17]: peopleDF.select(peopleDF["age"]).show()
```

```
+---+  
| age|  
+---+  
|null|  
| 32|  
| 28|  
| 49|  
| 48|  
+---+
```

```
In [24]: peopleDF.select(peopleDF.age).show()
```

```
+---+  
| age|  
+---+  
|null|  
| 32|  
| 28|  
| 49|  
| 48|  
+---+
```

Manipulate the column age i.e multiple age by 10.

```
peopleDF.select("lastName", peopleDF.age * 10).show()
```

```
In [20]: peopleDF.select("lastName", peopleDF.age * 10).show()
```

lastName	(age * 10)
Hopper	null
Turing	320
Lovelace	280
Babbage	490
Wirth	480

You can chain the Queries. Select last name, multiply the age by 10 times and show the result. Assign column name as age_10 for the derived column.

```
peopleDF.select("lastName", (peopleDF.age * 10).alias("age_10")).show()
```

```
In [21]: peopleDF.select("lastName", (peopleDF.age * 10).alias("age_10")).show()
```

lastName	age_10
Hopper	null
Turing	320
Lovelace	280
Babbage	490
Wirth	480

Perform aggregation. Find the no of people in a pincode.

```
peopleDF.groupBy("PCODE").count().show()
```

```
In [22]: peopleDF.groupBy("PCODE").count().show()
```

PCODE	count
87501	1
94020	2
02134	2

Next let us join two dataframes:

people-no-pcode.csv -> Contains information of person along with a pincode.

pcodes.csv -> Contains information about an area along with a pincode.

/opt/data/people-no-pcode.csv

```
PCODE,lastName,firstName,age  
02134,Hopper,Grace,52  
,Turing,Alan,32  
94020,Lovelace,Ada,28  
87501,Babbage,Charles,49  
02134,Wirth,Niklaus,48
```

/opt/data/pcodes.csv

```
PCODE,CITY,STATE  
02134,Boston,MA  
94020,Palo Alto,CA  
87501,Santa Fe,NM  
60645,Chicago,IL
```

We will determine the city of every person by joining the above two files.
Load the people and code files in DF.

```
peopleDF = spark.read.option("header","true").csv("/opt/data/people-no-pcode.csv")  
pcodesDF = spark.read.option("header","true").csv("/opt/data/pcodes.csv")
```

Have a glimpse of the datasets.

```
peopleDF.take(2)  
pcodesDF.take(2)
```

```
In [25]: peopleDF = spark.read.option("header","true").csv("/opt/data/people-no-pcode.csv")  
pcodesDF = spark.read.option("header","true").csv("/opt/data/pcodes.csv")
```

```
In [26]: peopleDF.take(2)
```

```
Out[26]: [Row(pcode='02134', lastName='Hopper', firstName='Grace', age='52'),  
          Row(pcode=None, lastName='Turing', firstName='Alan', age='32')]
```

```
In [27]: pcodesDF.take(2)
```

```
Out[27]: [Row(pcode='02134', city='Boston', state='MA'),  
          Row(pcode='94020', city='Palo Alto', state='CA')]
```

Perform Inner Join on pincode -> pcode.

```
peopleDF.join(pcodesDF, "pcode").show()
```

```
Reload this page .eDF.join(pcodesDF, "pcode").show()
```

pcode	lastName	firstName	age	city	state
02134	Hopper	Grace	52	Boston	MA
94020	Lovelace	Ada	28	Palo Alto	CA
87501	Babbage	Charles	49	Santa Fe	NM
02134	Wirth	Niklaus	48	Boston	MA

Perform the Left outer join.

Find all person along with city information in case the area information is available.

```
peopleDF.join(pcodesDF,peopleDF["pcode"] == pcodesDF.pcode, "left_outer").show()
```

```
In [29]: peopleDF.join(pcodesDF,peopleDF["pcode"] == pcodesDF.pcode, "left_outer").show()
```

pcode	lastName	firstName	age	pcode	city	state
02134	Hopper	Grace	52	02134	Boston	MA
null	Turing	Alan	32	null	null	null
94020	Lovelace	Ada	28	94020	Palo Alto	CA
87501	Babbage	Charles	49	87501	Santa Fe	NM
02134	Wirth	Niklaus	48	02134	Boston	MA

You can see null value in the pcode of the second row. Since there is no corresponding city information.

Joining on Columns with Different Names

Here, the people file contains pin code and its refer as zip in the area location file.

/opt/data/zcodes.csv

```
zip,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM
60645,Chicago,IL
```

Join with the pcode and zip of the second file.

```
zcodesDF = spark.read.option("header","true").csv("/opt/data/zcodes.csv")
peopleDF.join(zcodesDF, peopleDF.pcode == zcodesDF.zip).show()
```

```
In [30]: zcodesDF = spark.read.option("header","true").csv("/opt/data/zcodes.csv")
peopleDF.join(zcodesDF, peopleDF.pcode == zcodesDF.zip).show()
```

pcode	lastName	firstName	age	zip	city	state
02134	Hopper	Grace	52	02134	Boston	MA
94020	Lovelace	Ada	28	94020	Palo Alto	CA
87501	Babbage	Charles	49	87501	Santa Fe	NM
02134	Wirth	Niklaus	48	02134	Boston	MA

Try performing to get all person information even if the city information is not available in the above

query. Output should be as shown below.

pcode	lastName	firstName	age	zip	city	state
02134	Hopper	Grace	52	02134	Boston	MA
null	Turing	Alan	32	null	null	null
94020	Lovelace	Ada	28	94020	Palo Alto	CA
87501	Babbage	Charles	49	87501	Santa Fe	NM
02134	Wirth	Niklaus	48	02134	Boston	MA

----- Lab Ends Here -----

8. Explore Interactive Analysis with pySpark – 30 Minutes

Optional : export PYSPARK_PYTHON=python3.6 (In case you want to change the python version. Update in the bin\pyspark scripts of load environment sh file)

Start it by running the following in the Spark directory:

```
#pyspark
```

```
#Load a text file to perform some of the transformations provided by spark.  
textFile = sc.textFile("/opt/spark/README.md")
```

```
#Count the no of lines in the file.  
textFile.count()
```

```
#Get the first line  
textFile.first()
```

```
#Fetch the lines which begins with Spark.  
linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

```
#Get the number of rows  
linesWithSpark.count()
```

```
Apache Spark  
version 2.1.0  
  
Using Python version 2.6.6 (r266:84292, Oct 12 2012 14:23:48)  
SparkSession available as 'spark'.  
>>> textFile = sc.textFile("README.md")  
>>> textFile.count()  
104  
>>> textFile.first()  
u'# Apache Spark'  
  
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)  
>>> linesWithSpark.count()  
20  
>>> █
```

Sometimes there is a need to create RDD from a given Collection.

```
myData = ["Alice", "Carlos", "Frank", "Barbara"]  
myRDD = sc.parallelize(myData)
```

```
#Print all the element in RDD  
for make in myRDD.collect(): print(make)  
  
#Print the first 2 elements only.  
for make in myRDD.take(2): print(make)  
  
#Save the RDD to a file.  
myRDD.saveAsTextFile("/opt/data/mydata/")
```

```
In [6]: myData = ["Alice", "Carlos", "Frank", "Barbara"]  
myRDD = sc.parallelize(myData)
```

```
In [8]: for make in myRDD.collect(): print(make)
```

```
Alice  
Carlos  
Frank  
Barbara
```

```
In [9]: for make in myRDD.take(2): print (make)
```

```
Alice  
Carlos
```

```
In [10]: myRDD.saveAsTextFile("/opt/data/mydata/")
```

You can verify the data in the folder /opt/data/mydata/
tree mydata/

No. of files depend on the cores of your system.

```
# more mydata/part-00001
```

```
[root@spark0 data]# tree mydata/
mydata/
├── part-00000
├── part-00001
├── part-00002
├── part-00003
├── part-00004
├── part-00005
└── _SUCCESS

0 directories, 7 files
[root@spark0 data]# more mydata/part-00000
[root@spark0 data]# more mydata/part-00001
Alice
[root@spark0 data]# more mydata/part-00002
Carlos
[root@spark0 data]# []
```

Add one more RDD

```
myData1 = ["Ram","Shyam","Krishna","Vishnu"]
myRDD1 = sc.parallelize(myData1)
```

create a new RDD by combining the second RDD to the previous RDD.

```
myNames = myRDD.union(myRDD1)
```

Collect and display the contents of the new myNames RDD.

```
for name in myNames.collect(): print(name)
```

```
In [10]: myRDD.saveAsTextFile("/opt/data/mydata/")
```

```
In [11]: myData1 = ["Ram", "Shyam", "Krishna", "Vishnu"]
myRDD1 = sc.parallelize(myData1)
```

```
In [12]: myNames = myRDD.union(myRDD1)
```

```
In [13]: for name in myNames.collect(): print(name)
```

```
Alice
Carlos
Frank
Barbara
Ram
Shyam
Krishna
Vishnu
```

-----Lab Ends Here -----

9. Explore Transformation and Action – 45 Minutes

In this lab, we will explore various transformation method provided by spark RDD

- Map
- Mapfilter
- GroupBy etc

Let's make a new RDD from the text of the README file in the Spark source directory:
(SPARK_HOME/README.md)

```
textFile = sc.textFile("/opt/spark/README.md")
```

Let's start with a few actions:

```
textFile.count() # Number of items in this RDD
```

```
textFile.first() # First item in this RDD
```

```
In [5]: #Load the data file  
textFile = sc.textFile("/opt/spark/README.md")
```

```
In [6]: # Number of items in this RDD  
textFile.count()
```

Out[6]: 124

```
In [7]: textFile.first() # First item in this RDD
```

Out[7]: '# Apache Spark'

Now let's use a transformation. We will use the `filter` transformation to return a new RDD with a subset of the items in the file.

```
linesWithSpark = textFile.filter(lambda line: "Spark" in line)  
linesWithSpark.collect()
```

```
In [8]: linesWithSpark = textFile.filter(lambda line: "Spark" in line)
linesWithSpark.collect()

Out[8]: ['# Apache Spark',
'Spark is a unified analytics engine for large-scale data processing. It provides',
'rich set of higher-level tools including Spark SQL for SQL and DataFrames,',
'pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing,',
'[[PySpark Coverage]](https://codecov.io/gh/apache/spark/branch/master/graph/badge.svg)](https://codecov.io/gh/ap
ache/spark)',
'You can find the latest Spark documentation, including a programming',
'## Building Spark',
'Spark is built using [Apache Maven](https://maven.apache.org/).',
'To build Spark and its example programs, run:',
'[Building Spark](https://spark.apache.org/docs/latest/building-spark.html).',
'For general development tips, including info on developing Spark using an IDE, see ["Useful Developer Tools"](
https://spark.apache.org/developer-tools.html).',
'The easiest way to start using Spark is through the Scala shell:',
'Spark also comes with several sample programs in the `examples` directory.',
'./bin/run-example SparkPi',
'MASTER=spark://host:7077 ./bin/run-example SparkPi',
'Testing first requires [building Spark](#building-spark). Once Spark is built, tests',
'Spark uses the Hadoop core library to talk to HDFS and other Hadoop-supported',
'Hadoop, you must build Spark against the same version that your cluster runs.',
'in the online documentation for an overview on how to configure Spark.',
'Please review the [Contribution to Spark guide](https://spark.apache.org/contributing.html)']
```

We can chain together transformations and actions:

`textFile.filter(lambda line : "Spark" in line).count() # How many lines contain "Spark"?`

Let's say we want to find the line with the most words

`textFile.map(lambda line : len(line.split(" "))).reduce(lambda a, b: a if (a > b) else b)`

```
In [9]: textFile.filter(lambda line : "Spark" in line).count() # How many lines contain "Spark"?
```

```
Out[9]: 20
```

```
In [11]: textFile.map(lambda line : len(line.split(" "))).reduce(lambda a, b: a if (a > b) else b)
```

```
Out[11]: 16
```

One common data flow pattern is MapReduce, as popularized by Hadoop. Spark can implement MapReduce flows easily:

```
wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
```

Here, we combined the `flatMap`, `map` and `reduceByKey` transformations to compute the per-word counts in the file as an RDD of (String, Int) pairs. To collect the word counts in our shell, we can use the `collect` action:

```
wordCounts.collect()
```

```
In [8]: wordCounts.collect()
```

```
Out[8]: [('#', 1),
          ('Apache', 1),
          ('Spark', 15),
          ('is', 7),
          ('unified', 1),
          ('analytics', 1),
          ('engine', 2),
          ('It', 2),
          ('provides', 1),
          ('high-level', 1),
          ('APIs', 1),
          ('in', 5),
```

let's mark our `linesWithSpark` dataset to be cached:

```
linesWithSpark.cache()
```

```
linesWithSpark.count()
```

```
In [14]: linesWithSpark.cache()  
linesWithSpark.count()
```

```
Out[14]: 20
```

Next, we will convert RDD to Dataframe.

Create a file `/opt/data/people.txt`

```
02134,Hopper,Grace,52  
94020,Turing,Alan,32  
94020,Lovelace,Ada,28  
87501,Babbage,Charles,4
```

Import the necessary packages.

```
from pyspark.sql.types import *  
  
from pyspark.sql import Row
```

Load the data file and convert into Row Class.

```
rowRDD = sc.textFile("/opt/data/people.txt").map(lambda line : line.split(",")).map(lambda values :  
Row(values[0],values[1],values[2],values[3]))
```

```
In [15]: from pyspark.sql.types import *
from pyspark.sql import Row

In [16]: rowRDD = sc.textFile("/opt/data/people.txt").map(lambda line : line.split(",")).map(lambda values :
Row(values[0],values[1],values[2],values[3]))

In [18]: #Display two records of the RDD.
rowRDD.take(2)

Out[18]: [<Row('02134', 'Hopper', 'Grace', '52')>,
<Row('94020', 'Turing', 'Alan', '32')>]
```

Display two records of the RDD.
rowRDD.take(2)

```
In [18]: #Display two records of the RDD.
Reload this page
rowRDD.take(2)
```

```
Out[18]: [<Row('02134', 'Hopper', 'Grace', '52')>,
<Row('94020', 'Turing', 'Alan', '32')>]
```

Convert the RDD to data Frame.
myDF = rowRDD.toDF()
#Display two records of the data frame.
myDF.show(2)

In [20]: #Display two records of the data frame.
myDF.show(2)

```
+---+---+---+---+
| _1| _2| _3| _4|
+---+---+---+---+
|02134|Hopper|Grace| 52|
|94020|Turing| Alan| 32|
+---+---+---+---+
only showing top 2 rows
```

----- Lab Ends Here -----

10. Explore with Pair RDD – 90 Minutes

Lab Overview

In this activity, you will load SFPD data from a CSV file. You will create pair RDD and apply pair RDD operations to explore the data.

Scenario:

Our dataset is a .csv file that consists of SFPD incident data from SF OpenData (<https://data.sfgov.org/>). For each incident, we have the following information:

Field	Description	Example Value
IncidentNum	Incident number	150561637
Category	Category of incident	NON-CRIMINAL
Descript	Description of incident	FOUND_PROPERTY
DayOfWeek	Day of week that incident occurred	Sunday
Date	Date of incident	6/28/15
Time	Time of incident	23:50
PdDistrict	Police Department District	TARAVAL

Resolution	Resolution	NONE
Address	Address	1300_Block_of_LA_PLAYA_ST
X	X-coordinate of location	-122.5091348
Y	Y-coordinate of location	37.76119777
PdID	Department ID	15056163704013

The dataset has been modified to decrease the size of the files and also to make it easier to use. We use this same dataset for all the labs.

Load Data into Apache Spark Objectives

- Launch the Spark interactive shell
- Load data into Spark
- Explore data in Apache Spark

Launch the Spark Interactive Shell

1. To launch the Interactive Shell, at the command line, run the following command:

```
#pyspark --master local[2]
```

Load Data into Spark

To load the data we are going to use the `SparkContext` method `textFile`. The `SparkContext` is available in the interactive shell as the variable `sc`. We also want to split the file by the separator “,”.

We define the mapping for our input variables. While this isn’t a necessary step, it makes it easier to refer to the different fields by names.

```
val IncidntNum = 0 val Category = 1 val  
Descript = 2 val DayOfWeek = 3 val  
Date = 4  
val Time = 5  
val PdDistrict = 6 val Resolution = 7  
val Address = 8 val x = 9  
val Y = 10 val PdId = 11
```

To load data into Spark, at the `pyspark` command prompt:

```
sfpdRDD = sc.textFile("/Software/data/sfpd_no_header.csv").map(lambda line: line.split(","))
```

Explore data using RDD operations

What transformations and actions would you use in each case? Complete the command with the appropriate transformations and actions.

How do you see the first element of the inputRDD (`sfpdRDD`)?

```
sfpdRDD.first()
```

```
In [7]: sfpdRDD = sc.textFile("/Software/data/sfpd_no_header.csv").map(lambda line: line.split(","))
In [8]: sfpdRDD.first()
Out[8]: ['150467944',
'LARCENY/THEFT',
'GRAND THEFT FROM LOCKED AUTO',
'Thursday',
'05/28/2015',
'23:59',
'TENDERLOIN',
'NONE',
'TAYLOR ST / OFARRELL ST',
'-122.411328369311',
'37.7859963050476',
'"(37.7859963050476',
' -122.411328369311)"',
'15046794406244']
```

What do you use to see the first 5 elements of the RDD?

`sfpdRDD.take(5)`

```
In [9]: sfpdRDD.take(5)
```

```
Out[9]: [['150467944',
'LARCENY/THEFT',
'GRAND THEFT FROM LOCKED AUTO',
'Thursday',
'05/28/2015',
'23:59',
'TENDERLOIN',
'NONE',
'TAYLOR ST / OFARRELL ST',
'-122.411328369311',
'37.7859963050476',
'"(37.7859963050476',
' -122.411328369311)"',
'15046794406244'],
['150468629',
'VEHICLE THEFT',
'STOLEN TRUCK',
'Thursday',
'05/28/2015',
'23:59',
'MISSION',
'NONE',
'100 Block of PORTOLA DR',
'-122.444276468858',
```

What is the total number of incidents?

```
totincs = sfpdRDD.count()
print(totincs)
```

```
In [6]: totincs = sfpdRDD.count()  
print(totincs)
```

```
9999
```

What is the total number of distinct resolutions?

```
totres = sfpdRDD.map(lambda inc : inc[7]).distinct().count()  
totres
```

```
In [7]: totres = sfpdRDD.map(lambda inc : inc[7]).distinct().count()  
totres
```

```
Out[7]: 150
```

List the distinct PdDistricts.

```
totresdistricts = sfpdRDD.map(lambda inc : inc[6]).collect()  
totresdistricts
```

```
In [18]: totresdistricts = sfpdRDD.map(lambda inc : inc[6]).collect()
```

```
In [19]: totresdistricts
```

```
Out[19]: ['TENDERLOIN',
          'MISSION',
          'NORTHERN',
          'CENTRAL',
          'BAYVIEW',
          'MISSION',
          'NORTHERN',
          'MISSION',
          'MISSION',
          'MISSION',
          'MISSION',
          'MISSION',
          'MISSION',
          'SOUTHERN',
          '22:55',
          'MISSION',
          '22:55',
          '22:52',
          'PARK',
```

In the previous activity we explored the data in the sfpdRDD. We used RDD operations. In this activity, we will create pairRDD to find answers to questions about the data.

Objectives

- Create pairRDD & apply pairRDD operations
- Join pairRDD

Create pair RDD & apply pair RDD operations

```
# sfpdRDD = sc.textFile("/Software/data/sfpd_no_header.csv").map(lambda rec: rec.split(","))
```

Which five districts have the highest incidents?

Note: This is similar to doing a word count. First, use the `map` transformation to create a pair RDD with the key being the field on which you want to count. In this case, the key would be PdDistrict and the value is “1” for each count of the district.

To get the total count, use `reduceByKey ((a,b) => a+b)`.

To find the **top 5**, you have to do a sort in descending. However, sorting on the result of the `reduceByKey` will sort on the District rather than the count. Apply the `map` transformation to create a pairRDD with the count being the key and the value being the district.

Step to be follow (Hints)

- a. Use a `map` transformation to create pair RDD from sfpdRDD of the form: [(PdDistrict, 1)]

- b. Use **reduceByKey ((a,b) => a+b)** to get the count of incidents in each district. Your result will be a pairRDD of the form [(PdDistrict, count)]
- c. Use map again to get a pairRDD of the form [(count, PdDistrict)]
- d. Use **sortByKey (false)** on [(count, PdDistrict)]
- e. Use **take (5)** to get the top 5.

Answer:

```
top5Dists = sfpdRDD.filter(lambda incident: len(incident) == 14) \
    .map(lambda incident: (incident[6],1) ).reduceByKey(lambda x,y : x+y) \
    .map(lambda incinfo: (incinfo[1], incinfo[0]))    \
    .reduceByKey(lambda x,y : x+y)           \
    .sortByKey(False)

top5Dists.take(5)
```

```
In [84]: sfpdRDD = sc.textFile("/Software/data/sfpd_no_header.csv").map(lambda rec: rec.split(","))

In [85]: top5Dists = sfpdRDD.filter(lambda incident: len(incident) == 14) \
    .map(lambda incident: (incident[6], 1)).reduceByKey(lambda x,y : x+y) \
    .map(lambda incinfo: (incinfo[1], incinfo[0])) \
    .reduceByKey(lambda x,y : x+y) \
    .sortByKey(False)

In [86]: top5Dists.take(5)

Out[86]: [(1000, 'SOUTHERN'),
           (801, 'NORTHERN'),
           (704, 'CENTRAL'),
           (604, 'TARAVAL'),
           (582, 'MISSION')]
```

Add filter to validate proper record i.e it should have exactly 14 fields only in each record.

Which five addresses have the highest incidents?

- f. Create pairRDD (map)
- g. Get the count for key (reduceByKey)
- h. Pair RDD with key and count switched (map)
- i. Sort in descending order (sortByKey)
- j. Use **take (5)** to get the top 5

Answer:

```
top5Adds = sfpdRDD \  
.map(lambda incident : (incident[8],1)) \  
.reduceByKey(lambda x,y : x+y) \  
.map(lambda ad_count : (ad_count[1],ad_count[0])) \  
.sortByKey(False).take(5)
```

```
top5Adds
```

```
In [68]: top5Adds = sfpdRDD \  
.map(lambda incident : (incident[8],1)) \  
.reduceByKey(lambda x,y : x+y) \  
.map(lambda ad_count : (ad_count[1],ad_count[0])) \  
.sortByKey(False).take(5)
```

```
In [70]: top5Adds
```

```
Out[70]: [(1783, 'BOOKED'),  
(1535, 'NONE'),  
(628, '"ARREST'),  
(154, '800 Block of BRYANT ST'),  
(35, '1000 Block of POTRERO AV')]
```

What are the top **three** categories of incidents?

Answer:

```
top3Cat = sfpdRDD.filter(lambda incident: len(incident) == 14) \  
    .map(lambda incident: (incident[1],1)) \  
    .reduceByKey(lambda x,y : x+y) \  
    .map(lambda cat_count : (cat_count[1],cat_count[0])) \  
    .sortByKey(False).take(3)
```

top3Cat

```
In [76]: top3Cat = sfpdRDD.filter(lambda incident: len(incident) == 14) \  
    .map(lambda incident: (incident[1],1)) \  
    .reduceByKey(lambda x,y : x+y) \  
    .map(lambda cat_count : (cat_count[1],cat_count[0])) \  
    .sortByKey(False).take(3)
```

```
In [77]: top3Cat
```

```
Out[77]: [(2669, 'LARCENY/THEFT'), (633, 'NON-CRIMINAL'), (627, 'VEHICLE THEFT')]
```

What is the count of incidents by district?

Answer:

```
num_inc_dist = sfpdRDD.filter(lambda incident: len(incident) == 14) \
    .map(lambda incident:(incident[6],1)) \
    .countByKey()
```

```
num_inc_dist
```

```
In [89]: num_inc_dist = sfpdRDD.filter(lambda incident: len(incident) == 14) \
    .map(lambda incident:(incident[6],1)) \
    .countByKey()
```

```
In [90]: num_inc_dist
```

```
Out[90]: defaultdict(int,
    {'TENDERLOIN': 306,
     'MISSION': 582,
     'NORTHERN': 801,
     'CENTRAL': 704,
     'BAYVIEW': 493,
     'PARK': 385,
     'SOUTHERN': 1000,
     'TARAVAL': 604,
     'INGLESIDE': 511,
     'RICHMOND': 404})
```



Caution! For large datasets, don't use countByKey.

Join Pair RDDs

This activity illustrates how joins work in Spark (python). There are two small datasets provided for this activity - J_AddCat.csv and J_AddDist.csv.

J_AddCat.csv - Category; Address		J_AddDist.csv - PdDistrict; Address		
1.	EMBEZZLEMENT	100_Block_of_JEFFERSON_ST	1.	INGLESIDE
2.	EMBEZZLEMENT	SUTTER_ST/MASON_ST	2.	SOUTHERN
3.	BRIBERY	0_Block_of_SOUTH PARK_AV	3.	MISSION
4.	EMBEZZLEMENT	1500_Block_of_15TH_ST	4.	RICHMOND
5.	EMBEZZLEMENT	200_Block_of_BUSH_ST	5.	SOUTHERN
6.	BRIBERY	1900_Block_of_MISSION_ST	6.	SOUTHERN

7.	EMBEZZLEMENT	800_Block_of_MARKET_ST	7.	BAYVIEW	1400_Block_of_VANDYKE_AV
8.	EMBEZZLEMENT	2000_Block_of_MARKET_ST	8.	SOUTHERN	1100_Block_of_MISSION_ST
9.	BRIBERY	1400_Block_of_CLEMENT_ST	9.	TARAVAL	0_Block_of_CHUMASERO_DR
10.	EMBEZZLEMENT	1600_Block_of_FILLMORE_ST			
11.	EMBEZZLEMENT	1100_Block_of_SELBY_ST			
12.	BAD CHECKS	100_Block_of_BLUXOME_ST			

Based on the data above, answer the following questions:

- Given these two datasets, you want to find the type of incident and district for each address. What is one way of doing this? (HINT: An operation on pairs or pairRDDs)

What should the keys be for the two pairRDDs?

[You can use joins on pairs or pairRDD to get the information. The key for the pairRDD is the address.]

- What is the size of the resulting dataset from a join? Why?

 **Note:** Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a join is the same as an inner join and only keys

[A join is the same as an inner join and only keys that are present in both RDDs are output. If you compare the addresses in both the datasets, you find that there are five addresses in common and they are unique. Thus the resulting dataset will contain five elements. If there are multiple values for the same key, the resulting RDD will have an entry for every possible pair of values with that key from the two RDDs.]

- If you did a right outer join on the two datasets with Address/Category being the source

 **Note:** Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a right outer join results in a pair RDD that has RDD, what would be the size of the resulting dataset? Why?

[A right outer join results in a pair RDD that has entries for each key in the other pairRDD. If the source RDD contains data from J_AddCat.csv and the “other” RDD is represented by J_AddDist.csv, then since “other” RDD has 9 distinct addresses, the size of the result of a right outer join is 9.]

4. If you did a left outer join on the two datasets with Address/Category being the source



Note: Look at the datasets above and estimate what a join on the two would return if they were joined on the key - from #1.

Remember that a left outer join results in a pair RDD that has RDD, what would be the size of the resulting dataset? Why?

[A left outer join results in a pair RDD that has entries for each key in the source pairRDD. If the source RDD contains data from J_AddCat.csv and the “other” RDD is represented by J_AddDist.csv, then since “source” RDD has 13 distinct addresses, the size of the result of a left outer join is 13.]

Load each dataset into separate pairRDDs with “address” being the key.



Note: once you load the text file, split on the “,” and then apply the `map` transformation to create a pairRDD where address is

```
# catAdd = sc.textFile("/Software/data/J_AddCat.csv").map(lambda x :  
x.split(",")).map(lambda x : (x[1],x[0]))  
# distAdd = sc.textFile("/Software/data/J_AddDist.csv").map(lambda x :  
x.split(",")).map(lambda x : (x[1],x[0]))  
catAdd.take(2)  
distAdd.take(2)
```

List the incident category and district for those addresses that have both category and district information. Verify that the size estimated earlier is correct.

```
catJdist = catAdd.join(distAdd)  
catJdist.collect()  
catJdist.count()  
catJdist.take(5)
```

```
%pyspark
catJdist = catAdd.join(distAdd)

Took 1 sec. Last updated by anonymous at June 28 2021, 4:28:22 PM.
```

```
%pyspark
catJdist.collect()

[("u'1400_Block_of_CLEMENT_ST", "u'BRIBERY", "u'RICHMOND'), ("u'1400_Block_of_VANDYKE_AV", "u'BRIBERY", "u'BAYVIEW'), ("u'0_Block_of_SOUTHPARK_AV", "u'BRIBERY", "u'SOUTHERN'), ("u'1900_Block_of_MISSION_ST", "u'BRIBERY", "u'MISSION'), ("u'100_Block_of_BLUXOME_ST", "u'BAD CHECKS", "u'SOUTHERN')]

Took 4 sec. Last updated by anonymous at June 28 2021, 4:28:37 PM.
```

```
%pyspark
catAdd.take(2)

[("u'100_Block_of_JEFFERSON_ST", "u'EMBEZZLEMENT"), ("u'SUTTER_ST/MASON_ST", "u'EMBEZZLEMENT")]

Took 1 sec. Last updated by anonymous at June 28 2021, 4:29:48 PM.
```

```
%pyspark
distAdd.take(2)

[("u'100_Block_of_ROME_ST", "u'INGLESIDE'), ("u'0_Block_of_SOUTHPARK_AV", "u'SOUTHERN')]
```

List the incident category and district for all addresses irrespective of whether each address has category and district information.

```
catJdist1 = catAdd.leftOuterJoin(distAdd)
catJdist1.collect()
catJdist1.count()
```

```
%pyspark  
catJdist1 = catAdd.leftOuterJoin(distAdd)
```

FINISHED ▶ ✎ 📄 ⚙

Took 0 sec. Last updated by anonymous at June 28 2021, 4:34:10 PM.

```
%pyspark  
catJdist1.collect()
```

SPARK JOB FINISHED ▶ ✎ 📄 ⚙

```
[('SUTTER_ST/MASON_ST', ('EMBEZZLEMENT', None)), ('1100_Block_of_SELBY_ST', ('EMBEZZLEMENT', None)), ('1400_Block_of_CLEMENT_ST', ('BRIBERY', 'RICHMOND')), ('1400_Block_of_VANDYKE_AV', ('BRIBERY', 'BAYVIEW')), ('1500_Block_of_15TH_ST', ('EMBEZZLEMENT', None)), ('2000_Block_of_MARKET_ST', ('EMBEZZLEMENT', None)), ('0_Block_of_SOUTHPARK_AV', ('BRIBERY', 'SOUTHERN')), ('100_Block_of_JEFFERSON_ST', ('EMBEZZLEMENT', None)), ('1600_Block_of_FILLMORE_ST', ('EMBEZZLEMENT', None)), ('200_Block_of_BUSH_ST', ('EMBEZZLEMENT', None)), ('1900_Block_of_MISSION_ST', ('BRIBERY', 'MISSION')), ('800_Block_of_MARKET_ST', ('EMBEZZLEMENT', None)), ('100_Block_of_BLUXOME_ST', ('BAD CHECKS', 'SOUTHERN'))]
```

Took 1 sec. Last updated by anonymous at June 28 2021, 4:34:25 PM.

```
%pyspark  
catJdist1.count()
```

SPARK JOB FINISHED ▶ ✎ 📄 ⚙

13

Took 0 sec. Last updated by anonymous at June 28 2021, 4:34:25 PM.

List the incident district and category for all addresses irrespective of whether each address has category and district information. Verify that the size estimated earlier is correct.

```
catJdist2 = catAdd.rightOuterJoin(distAdd)  
catJdist2.collect()  
catJdist2.count()
```

```
%pyspark  
catJdist2 = catAdd.rightOuterJoin(distAdd)
```

FINISHED ▶ ✎ 📄 ⚙

Took 1 sec. Last updated by anonymous at June 28 2021, 4:35:26 PM.

```
%pyspark  
catJdist2.collect()
```

SPARK JOB FINISHED ▶ ✎ 📄 ⚙

```
[(u'1400_Block_of_CLEMENT_ST', (u'BRIBERY', u'RICHMOND')), (u'0_Block_of_CHUMASERO_DR', (None, u'TARAVAL')), (u'1400_Block_of_VANDYKE_AV', (u'BRIBERY', u'BAYVIEW')), (u'0_Block_of_SOUTHPARK_AV', (u'BRIBERY', u'SOUTHERN')), (u'1100_Block_of_MISSION_ST', (None, u'SOUTHERN')), (u'1900_Block_of_MISSION_ST', (u'BRIBERY', u'MISSION')), (u'300_Block_of_BERRY_ST', (None, u'SOUTHERN')), (u'100_Block_of_ROME_ST', (None, u'INGLESIDE')), (u'100_Block_of_BLUXOME_ST', (u'BAD CHECKS', u'SOUTHERN'))]
```

Took 1 sec. Last updated by anonymous at June 28 2021, 4:35:40 PM.

```
%pyspark  
catJdist2.count()
```

SPARK JOB FINISHED ▶ ✎ 📄 ⚙

9

Took 1 sec. Last updated by anonymous at June 28 2021, 4:35:48 PM.

Explore Partitioning

In this activity we see how to determine the number of partitions, the type of partitioner and how to specify partitions in a transformation.

Objective

- Explore partitioning in RDDs

Explore partitioning in RDDs



Note To find partition size: `rdd.partitions.size` (Scala); `rdd.getNumPartitions()` (in Python) To determine the partitioner: `rdd.partitionner` (Scala);

Note: Ensure that you have started the spark shell with --master local[*] (Refer the First Lab)

1. How many partitions are there in the sfpdRDD?
`sfpdRDD.getNumPartitions()`
2. How do you find the type of partitioner for sfpdRDD?
`sfpdRDD.partitionner`

```
%pyspark  
sfpdRDD.getNumPartitions()
```

2

Took 0 sec. Last updated by anonymous at June 29 2021, 9:10:17 AM.

```
%pyspark  
print(sfpdRDD.partitioner)
```

None

Took 0 sec. Last updated by anonymous at June 28 2021, 4:40:24 PM.

The above result will depend on the no of cores. In my case its 2 cores so 2.

If there is **no partitioner** the partitioning is not based upon characteristic of data but distribution is random and uniformed across nodes.

3. Create a pair RDD when only the length of the items is 14 per record that its 14 fields.

```
incByDists = sfpdRDD.filter(lambda rec : len(rec) == 14 ).map(lambda incident :  
(incident(6),1)).reduceByKey(lambda x,y : x+y)
```

How many partitions does incByDists have?

```
incByDists.getNumPartitions()
```

What type of partitioner does incByDists have?

```
print(incByDists.partition)
```



Q: Why does incByDists have that partitioner?

A: reduceByKey automatically uses the Hash Partitioner

By default PySpark implementation uses *hash partitioning* as the partitioning function.

```
%pyspark  
incByDists.getNumPartitions()
```

2

Took 0 sec. Last updated by anonymous at June 29 2021, 9:18:33 AM.

```
%pyspark  
print(incByDists.partition)  
<pyspark.rdd.Partitioner object at 0x7f5acb8ac0d0>
```

Took 0 sec. Last updated by anonymous at June 29 2021, 9:24:51 AM.

4. Add a map

```
inc_map = incByDists.map(lambda (incidence, district) : (district,incidence))
```

How many partitions does inc_map have? `inc_map.getNumPartitions()`

What type of partitioner does incByDists have? `print(incByDists.partitionert)`



Q: Why does inc_map not have the same partitioner as its parent?

A: Transformations such as map() cause the new RDD to forget the parent's partitioning information because a map() can modify the key of each record.

```
%pyspark  
inc_map = incByDists.map(lambda (incidence, district) : (district,incidence))
```

Took 0 sec. Last updated by anonymous at June 29 2021, 9:35:33 AM.

```
%pyspark  
inc_map.getNumPartitions()
```

2

Took 0 sec. Last updated by anonymous at June 29 2021, 9:35:37 AM.

```
%pyspark  
print(inc_map.partitioner)
```

None

Took 0 sec. Last updated by anonymous at June 29 2021, 9:36:06 AM.

5. Add groupByKey

inc_group = sfpdRDD.map(lambda incident: (incident[6],1)).groupByKey()

What type of partitioner does inc_group have?

inc_group.partitioner



Q: Why does inc_group have this type of partitioner?

A: This is because groupByKey will automatically result in a hash partitioned RDD..

```
%pyspark  
inc_group = sfpdRDD.map(lambda incident: (incident[6],1)).groupByKey()
```

Took 1 sec. Last updated by anonymous at June 28 2021, 4:52:39 PM.

```
%pyspark  
inc_group.partitionert  
<pyspark.rdd.Partitioner object at 0x7f42a6f8d8d0>
```

Took 0 sec. Last updated by anonymous at June 28 2021, 4:55:45 PM.

6. Create two pairRDD

```
catAdd = sc.textFile("/Software/data/J_AddCat.csv").map(lambda x : x.split(",")).map(lambda x :(x[1],x[0]))  
distAdd = sc.textFile("/Software/data/J_AddDist.csv").map(lambda x : x.split(",")).map(lambda x :(x[1],x[0]))
```

7. You can specify the number of partitions when you use the join operation.

```
catJdist = catAdd.join(distAdd,8)
```

How many partitions does the joined RDD have?

```
catJdist.getNumPartitions()
```

What type of partitioner does catJdist have?

```
catJdist.partition
```

```
%pyspark
catAdd = sc.textFile("/opt/data/J_AddCat.csv").map(lambda x : x.split(",")).map(lambda x : (x[1],x[0]))
distAdd = sc.textFile("/opt/data/J_AddDist.csv").map(lambda x : x.split(",")).map(lambda x : (x[1],x[0]))
```

Took 0 sec. Last updated by anonymous at June 28 2021, 4:58:42 PM.

```
%pyspark
catJdist = catAdd.join(distAdd,8)
```

Took 1 sec. Last updated by anonymous at June 28 2021, 4:59:02 PM.

```
%pyspark
catJdist.getNumPartitions()
```

8

Took 0 sec. Last updated by anonymous at June 28 2021, 4:59:35 PM.

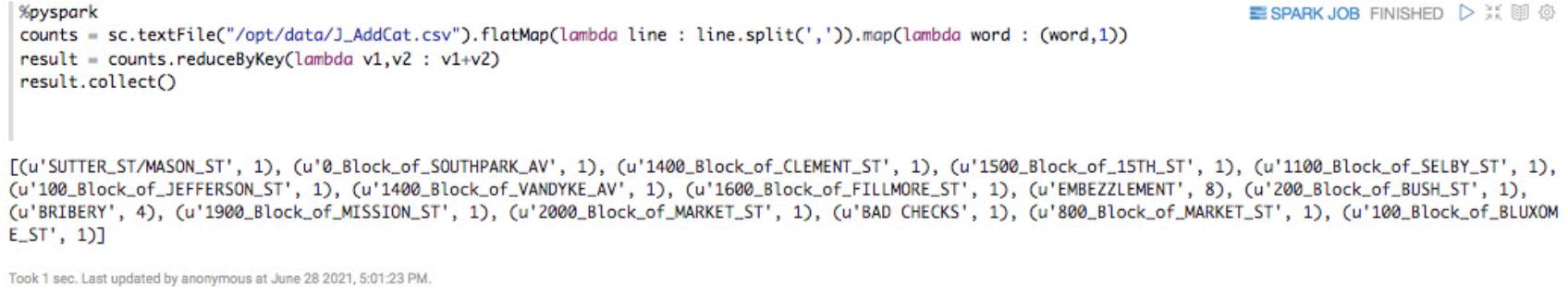
```
%pyspark
catJdist.partition
```

<pyspark.rdd.Partitioner object at 0x7f42a2454510>

Word Count Using Pair.

Finally, let us perform the word count application using the Pair data model.

```
counts = sc.textFile("/Software/data/J_AddCat.csv") \
    .flatMap(lambda line : line.split(',')) \
    .map(lambda word : (word,1))
result = counts.reduceByKey(lambda v1,v2 : v1+v2)
result.collect()
```



```
%pyspark
counts = sc.textFile("/opt/data/J_AddCat.csv").flatMap(lambda line : line.split(',')).map(lambda word : (word,1))
result = counts.reduceByKey(lambda v1,v2 : v1+v2)
result.collect()

[(u'SUTTER_ST/MASON_ST', 1), (u'0_Block_of_SOUTHPARK_AV', 1), (u'1400_Block_of_CLEMENT_ST', 1), (u'1500_Block_of_15TH_ST', 1), (u'1100_Block_of_SELBY_ST', 1), (u'100_Block_of_JEFFERSON_ST', 1), (u'1400_Block_of_VANDYKE_AV', 1), (u'1600_Block_of_FILLMORE_ST', 1), (u'EMBEZZLEMENT', 8), (u'200_Block_of_BUSH_ST', 1), (u'BRIBERY', 4), (u'1900_Block_of_MISSION_ST', 1), (u'2000_Block_of_MARKET_ST', 1), (u'BAD CHECKS', 1), (u'800_Block_of_MARKET_ST', 1), (u'100_Block_of_BLUXOME_ST', 1)]
```

Took 1 sec. Last updated by anonymous at June 28 2021, 5:01:23 PM.

Errata: Lambda parameter errors. Sol – Remove the enclosing bracket.

```
>>> incByDists = sfpdRDD.filter(lambda rec : len(rec) == 14 ).map(lambda incident : (incident(6),1)).reduce
ByKey(lambda (x,y) : x+y)
  File "<stdin>", line 1
    incByDists = sfpdRDD.filter(lambda rec : len(rec) == 14 ).map(lambda incident : (incident(6),1)).reduce
ByKey(lambda (x,y) : x+y)
^
SyntaxError: invalid syntax
>>> incByDists = sfpdRDD.filter(lambda rec : len(rec) == 14 ).map(lambda incident : (incident(6),1)).reduce
ByKey(lambda x,y : x+y)
>>> incByDists.getNumPartitions()
2
>>> print(incByDists.partitioner)
```

----- Lab Ends Here -----

11. Explore Spark SQL – 45 Minutes

Spark SQL can convert an RDD of Row objects to a DataFrame, inferring the datatypes. Rows are constructed by passing a list of key/value pairs as kwargs to the Row class.

Copy the `person.txt` file to `/Software/data/` folder.

```
Henry,42  
Rajnita,40  
Henderson,14  
Tiraj,5
```

Open a terminal from `/Software/data` folder and enter the following command.

Using Spark SQL – RDD data structure. (Inferring the Schema Using Reflection)

```
#pyspark
```

Create an RDD of Person objects and register it as a table.

```
peopleL = sc.textFile("/Software/data/person.txt")  
peopleL.take(4)  
  
from pyspark.sql import Row  
parts = peopleL.map(lambda l: l.split(","))  
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
```

```
# Infer the schema, and register the DataFrame as a table.  
schemaPeople = spark.createDataFrame(people)  
schemaPeople.createOrReplaceTempView("people")
```

SQL statements can be run by using the sql methods provided by sqlContext.

```
# SQL can be run over DataFrames that have been registered as a table.  
kids = spark.sql("SELECT name FROM people WHERE age >= 1 AND age <= 9")
```

The results of SQL queries are DataFrames and support all the normal RDD operations.

```
# The results of SQL queries are Dataframe objects.  
# rdd returns the content as an :class:`pyspark.RDD` of :class:`Row`.  
kidNames = kids.rdd.map(lambda p: "Name: " + p.name).collect()  
for name in kidNames:  
    print(name)
```

```
In [1]: peopleL = sc.textFile("/Software/data/person.txt")
peopleL.take(4)
```

```
Out[1]: ['Henry,42', 'Rajnita,40', 'Henderson,14', 'Tiraj,5']
```

```
In [2]: parts = peopleL.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
```

```
In [4]: from pyspark.sql import Row
```

```
In [5]: # Infer the schema, and register the DataFrame as a table.
schemaPeople = spark.createDataFrame(people)
schemaPeople.createOrReplaceTempView("people")
```

```
In [6]: # SQL can be run over DataFrames that have been registered as a table.
kids = spark.sql("SELECT name FROM people WHERE age >= 1 AND age <= 9")
```

```
In [8]: # The results of SQL queries are Dataframe objects.
# rdd returns the content as an :class:`pyspark.RDD` of :class:`Row`.
kidNames = kids.rdd.map(lambda p: "Name: " + p.name).collect()
for name in kidNames:
    print(name)
```

```
Name: Tiraj
```

Create a temporary view:

```
# Register the DataFrame as a SQL temporary view  
schemaPeople.createTempView("kids")
```

Display the output.

```
spark.sql("select * from kids").show()
```

```
%pyspark  
# Register the DataFrame as a SQL temporary view  
schemaPeople.createTempView("kids")
```

Took 0 sec. Last updated by anonymous at June 29 2021, 1:07:51 PM. (outdated)

```
%pyspark  
spark.sql("select * from kids").show()
```

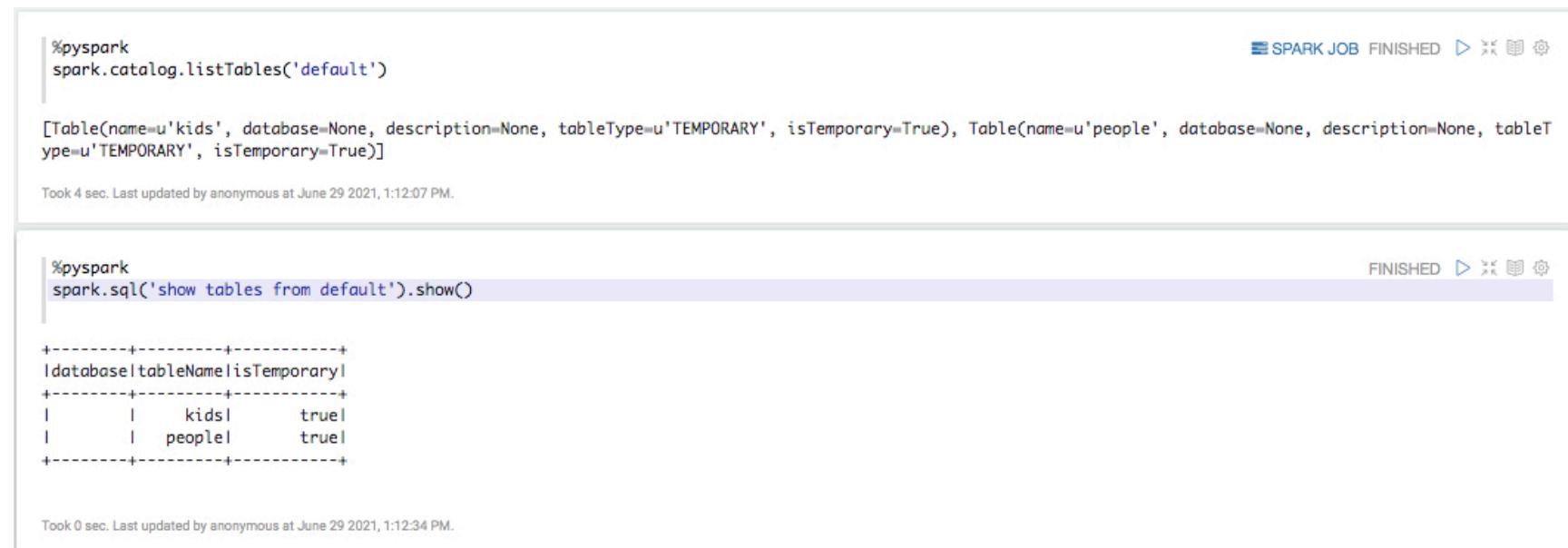
```
+---+  
| age|    name|  
+---+  
| 42|    Henry|  
| 40| Rajnital|  
| 14|Henderson|  
| 5|     Tiraj|  
+---+
```

Show the Tables or View List.

```
spark.catalog.listTables('default')
```

or

```
spark.sql('show tables from default').show()
```



```
%pyspark
spark.catalog.listTables('default')

[Table(name=u'kids', database=None, description=None, tableType=u'TEMPORARY', isTemporary=True), Table(name=u'people', database=None, description=None, tableType=u'TEMPORARY', isTemporary=True)]
```

Took 4 sec. Last updated by anonymous at June 29 2021, 1:12:07 PM.

```
%pyspark
spark.sql('show tables from default').show()
```

database	tableName	isTemporary
	kids	true
	people	true

Took 0 sec. Last updated by anonymous at June 29 2021, 1:12:34 PM.

Using DataFrame and SQL:

/Software/data/people.json

```
{"name":"Michael"}  
{"name":"Andy", "age":30}  
{"name":"Justin", "age":19}
```

Load the JSON file and store it as a parquet file

```
peopleDF = spark.read.format("json").load("/Software/data/people.json")  
peopleDF.select("name", "age").write.format("parquet").save("namesAndAges.parquet")  
peopleDF.select("name", "age").show
```

```
In [20]: peopleDF.select("name", "age").show()
```

name	age
Michael	null
Andy	30
Justin	19

With a parquet file, it can Query directly with SQL.

```
sqlDF = spark.sql("SELECT * FROM parquet.`namesAndAges.parquet`")
sqlDF.show()
```

```
%pyspark
sqlDF = spark.sql("SELECT * FROM parquet.`namesAndAges.parquet`")
```

Took 2 sec. Last updated by anonymous at June 29 2021, 1:20:11 PM.

```
%pyspark
sqlDF.show()
```

name	age
Michael	null
Andy	30
Justin	19

Took 2 sec. Last updated by anonymous at June 29 2021, 1:20:22 PM.

Perform some operations on the Dataframe.

In Python, it's possible to access a DataFrame's columns either by attribute (`df.age`) or by indexing (`df['age']`).

```
#convert the people RDD to Dataframe  
df = people.toDF()  
  
# Select everybody, but increment the age by 1  
df.select(df['name'], df['age'] + 1).show()  
  
# Select people older than 21  
df.filter(df['age'] > 21).show()  
  
# Count people by age  
df.groupBy("age").count().show()
```

```
%pyspark  
#convert the people RDD to Dataframe  
df = people.toDF()
```

Took 0 sec. Last updated by anonymous at June 29 2021, 1:23:13 PM. (outdated)

```
%pyspark  
df.select(df['name'], df['age'] + 1).show()
```

```
+-----+  
|    name| (age + 1)|  
+-----+  
| Henry |     43 |  
| Rajnita|     41 |  
|Henderson|     15 |  
| Tiraj |      6 |  
+-----+
```

Took 1 sec. Last updated by anonymous at June 29 2021, 1:23:24 PM.

```
%pyspark  
df.filter(df['age'] > 21).show()
```

```
+-----+  
| age|    name|  
+-----+  
| 42 | Henry |  
| 40 |Rajnita|  
+-----+
```

```
%pyspark  
df.groupBy("age").count().show()
```

age	count
5	1
14	1
42	1
40	1

Took 11 sec. Last updated by anonymous at June 29 2021, 1:23:57 PM.

Using aggregate functions:

Determine the max and min age.

spark.sql("select min(age) from kids ").show()

spark.sql("select max(age) from kids ").show()

```
%pyspark  
spark.sql("select min(age) from kids ").show()  
  
+-----+  
|min(age)|  
+-----+  
|      51|  
+-----+
```

Took 1 sec. Last updated by anonymous at June 29 2021, 2:01:51 PM.

```
%pyspark  
spark.sql("select max(age) from kids ").show()  
  
+-----+  
|max(age)|  
+-----+  
|     42|  
+-----+
```

Took 1 sec. Last updated by anonymous at June 29 2021, 2:02:16 PM.

Lab Ends Here

12. Explore Pyspark Applications – 30 Minutes

In this lab, let us develop a python application and submit to Spark for execution. Ensure that you have created a folder `/opt/code/pyspark` for storing all the python code inside it.

Let us create a simple Spark application, `SimpleApp.py`:

```
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "/opt/spark/README.md" # Should be some file on your system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()

numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print("Lines with a: %i, lines with b: %i" % (numAs, numBs))

sc.stop()
```

This program just counts the number of lines containing ‘a’ and the number containing ‘b’ in a text file. Note that you’ll need to replace `YOUR_SPARK_HOME /logFile` with the location where Spark is installed in your machine.

We can run this application using the `bin/spark-submit` script:

Use spark-submit to run your application, change directory to /spark before executing the following command

```
$ spark-submit --master local[2] SimpleApp.py
```

```
[root@localhost spark]# ./spark-2.1.0/bin/spark-submit --master local[2] workspace/SimpleApp.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
17/04/14 09:30:52 WARN Utils: Your hostname, localhost.localdomain resolves to a
loopback address: 127.0.0.1; using 192.168.150.128 instead (on interface eth0)
17/04/14 09:30:52 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
```

You can view the job status with the following URL

<http://localhost:4040/jobs/>

```
649 ms on localhost (executor driver) (1/1)
17/04/14 09:32:21 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have
all completed, from pool
Lines with a: 62, lines with b: 30
17/04/14 09:32:22 INFO SparkUI: Stopped Spark web UI at http://192.168.150.128:4
040
```

In this way, you can deploy any python application in the spark cluster.

----- Lab Ends Here -----

13. Installing Jupyter for Spark – 35 Minutes.(D)

You can install Jupyter using pip or anaconda.

Using pip – We will use this for our lab.

```
#yum install epel-release  
# yum install python3.8 pr 3.9
```

Configure the default Python version

```
#alternatives --config python3  
or  
#update-alternatives --config python3
```

Install python 3.8 first.

Using pip

```
#yum install wget  
# wget https://bootstrap.pypa.io/get-pip.py  
#python3.8 get-pip.py
```

Any issue refer the following for installation steps.

<https://pip.pypa.io/en/stable/installation/>

```
# yum install -y python38 python38-devel  
  
#pip3.8 install --upgrade setuptools  
#pip3.8 install --upgrade pip3.8  
#pip3.8 install jupyterlab  
#pip3.8 install jupyter
```

To changes the latest Python version.

```
Successfully installed argon2-cffi-21.3.0 argon2-cffi-bindings-21.2.0 async-generator-1.10 backcall-0.2.0 bleach-4.1.  
0 defusedxml-0.7.1 ipykernel-5.5.6 ipython-7.16.3 jedi-0.17.2 jupyter-server-1.13.1 jupyterlab-3.2.9 jupyterlab-pygme  
nts-0.1.2 jupyterlab-server-2.10.3 mistune-0.8.4 nbclassic-0.3.5 nbclient-0.5.9 nbconvert-6.0.7 notebook-6.4.8 packag  
ing-21.3 pandocfilters-1.5.0 parso-0.7.1 pexpect-4.8.0 pickleshare-0.7.5 prompt-toolkit-3.0.28 pygments-2.11.2 pypars  
ing-3.0.7 testpath-0.5.0 wcwidth-0.2.5 webencodings-0.5.1  
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system pa  
ckage manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv  
[root@spark0 opt]#
```

To run the notebook:

```
#jupyter notebook --generate-config
```

Start Jupyter in the different port.

```
#jupyter notebook --generate-config  
[Writing default config to: /root/.jupyter/jupyter_notebook_config.py]
```

Update the following properties.

```
#vi ~/.jupyter/jupyter_notebook_config.py
```

```
c.NotebookApp.ip = 'spark0'  
c.NotebookApp.open_browser = False  
c.NotebookApp.allow_remote_access = True  
c.NotebookApp.allow_root = True
```

or

```
c.ServerApp.ip = 'spark0'  
c.ServerApp.open_browser = False  
c.ServerApp.allow_remote_access = True  
c.ServerApp.allow_root = True
```

If you want to update the port no.

```
## The port the notebook server will listen on (env: JUPYTER_PORT).  
# Default: 8888  
c.NotebookApp.port = 8200
```

Start with config parameter.

```
#jupyter lab --ServerApp.port=8080 --allow-root --ServerApp.allow_remote_access=True
```

Or

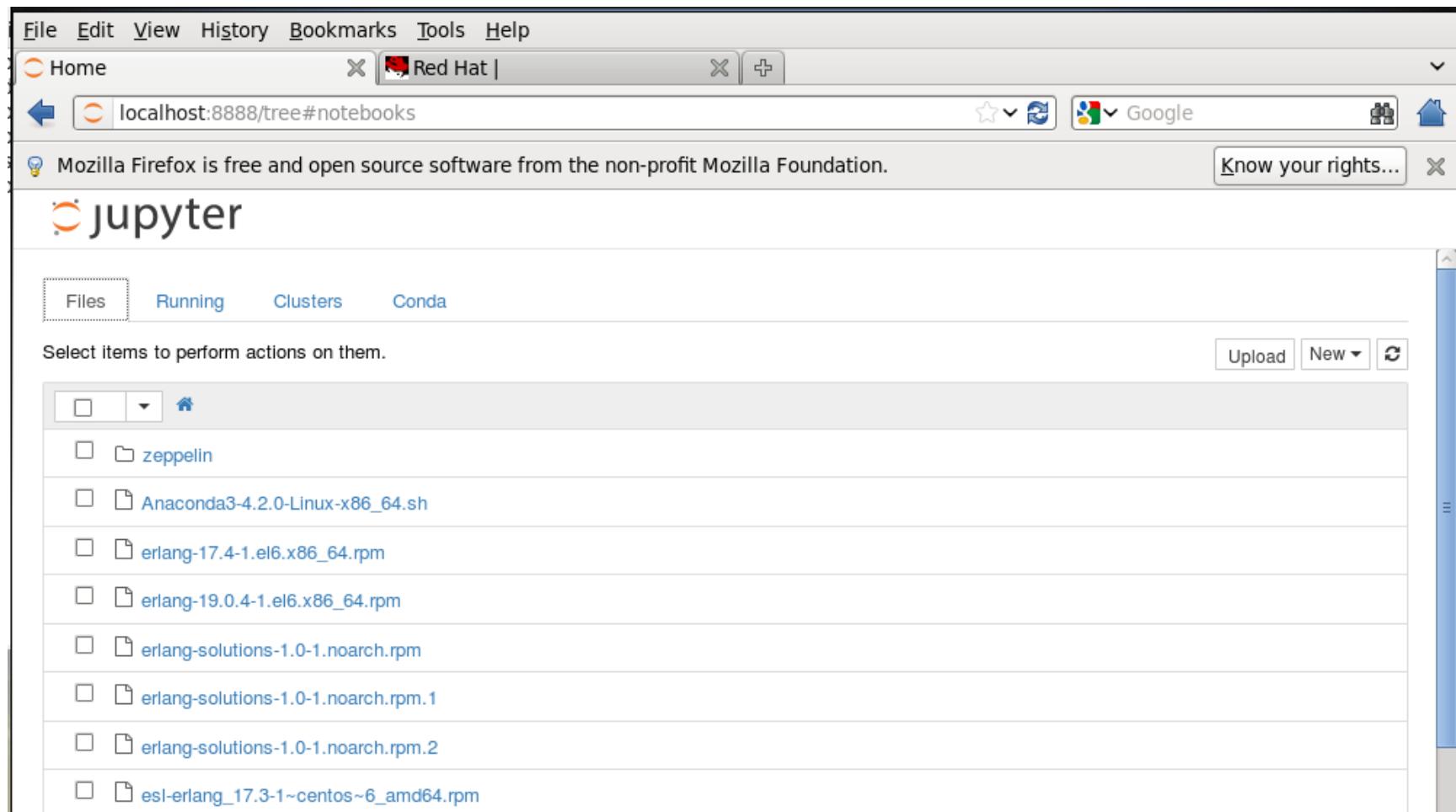
```
#jupyter lab
```

Or

```
#jupyter notebook
```

Copy the URL in the web browser.

```
[root@tos Software]# jupyter notebook
[I 10:53:18.707 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[I 10:53:18.717 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 10:53:18.870 NotebookApp] [nb_conda] enabled
[I 10:53:19.132 NotebookApp] [nb_anacondacloud] enabled
[I 10:53:19.283 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 10:53:19.283 NotebookApp] X nbpresent PDF export DISABLED: No module named 'nbconvertpdf'
[I 10:53:19.389 NotebookApp] Serving notebooks from local directory: /Software
[I 10:53:19.389 NotebookApp] 0 active kernels
[I 10:53:19.389 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 10:53:19.390 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[W 10:53:19.393 NotebookApp] No web browser found: could not locate runnable browser.
[I 10:54:52.881 NotebookApp] 302 GET / (::1) 1.13ms
```



Let us configure Notebook for Pyspark.

Update PySpark driver environment variables: add these lines to your `~/.bashrc` (or `~/.zshrc`) file.

```
export PYSPARK_PYTHON=python3.8
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
```

Restart your terminal and launch PySpark again:

```
$ pyspark
```

Now, this command should start a Jupyter Notebook in your web browser. Create a new notebook by clicking on ‘New’ > ‘Notebooks Python [default]’.

Copy and paste Pi calculation script and run it by pressing Shift + Enter.

```
// Code Begin

import random
num_samples = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0, num_samples)).filter(inside).count()

pi = 4 * count / num_samples
print(pi)

sc.stop()
// Code Ends.
```

```
In [1]: import random
num_samples = 100000000

def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1

count = sc.parallelize(range(0, num_samples)).filter(inside).count()

pi = 4 * count / num_samples
print(pi)

sc.stop()

3.14156176
```

You have successfully configured pyspark on Jupyter.

Add the following in the bash profile and open jupyter with **pnotebook** for integrating hive commands with Jupyter.

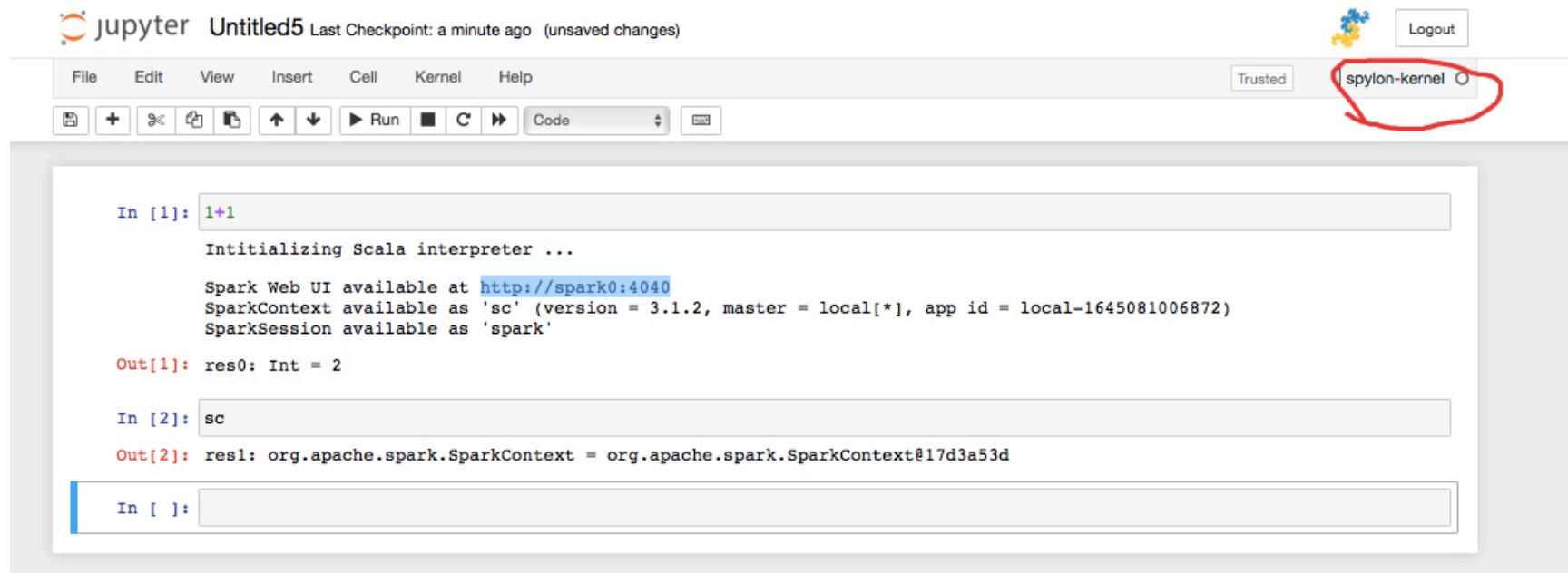
```
#Required for pyspark+kafka+hive integration.
alias pnotebook='pyspark --packages org.apache.spark:spark-sql-kafka-0-
10_2.12:3.3.0,org.apache.spark:spark-streaming-kafka-0-10_2.12:3.3.0
--conf spark.sql.catalogImplementation=hive'
```

Configure scala on Jupyter.

Option I: Using spylon – Execute Scala spark program.

```
#pip3 install spylon-kernel  
#python3 -m spylon_kernel install  
#export SPARK_HOME=/opt/spark  
#jupyter notebook
```

In the web console, select New -> Spylon-kernel and enter the following commands.

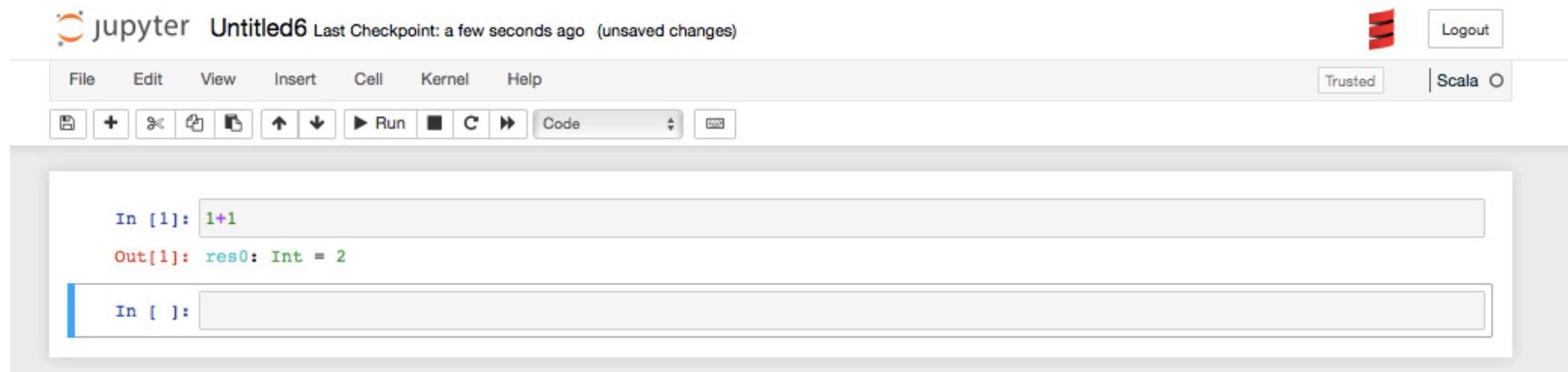


The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter Untitled5 Last Checkpoint: a minute ago (unsaved changes)
- Kernel:** spylon-kernel (highlighted with a red circle)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Help
- Cell 1 (In [1]):** 1+1
Initialization output:
Intitializing Scala interpreter ...
Spark Web UI available at <http://spark0:4040>
SparkContext available as 'sc' (version = 3.1.2, master = local[*], app id = local-1645081006872)
SparkSession available as 'spark'
- Cell 2 (Out[1]):** res0: Int = 2
- Cell 3 (In [2]):** sc
Output:
Out[2]: res1: org.apache.spark.SparkContext = org.apache.spark.SparkContext@17d3a53d
- Cell 4 (In []):** (empty)

Option 2: using Scala in Jupyter

<https://almond.sh/docs/quick-start-install>



Or using Anaconda: (Optional)

Anaconda 4.2.0

For Linux

Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.

[Changelog](#)

Download the installer

Optional: Verify data integrity with [MD5 or SHA-256 More info](#)

In your terminal window type one of the below and follow the instructions:

Python 3.5 version

`bash Anaconda3-4.2.0-Linux-x86_64.sh`

Python 2.7 version

`bash Anaconda2-4.2.0-Linux-x86_64.sh`

NOTE: Include the "bash" command even if you are not using the bash shell.

```
[root@tos Software]# bash
[root@tos Software]# sh Anaconda3-4.2.0-Linux-x86_64.sh

Welcome to Anaconda3 4.2.0 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.

Please, press ENTER to continue
>>> 
```

```
Anaconda3 will now be installed into this location:
/root/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below
```

```
[/root/anaconda3] >>> /anaconda3

Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /root/.bashrc ? [yes|no]
[no] >>> yes
```

```
Prepending PATH=/anaconda3/bin to PATH in /root/.bashrc
A backup will be made to: /root/.bashrc-anaconda3.bak
```

```
For this change to become active, you have to open a new terminal.
```

```
Thank you for installing Anaconda3!
```

```
Share your notebooks and packages on Anaconda Cloud!
Sign up for free: https://anaconda.org
```

```
[root@tos Software]# 
```

#bash

Run the following command to install Jupyter.

```
#conda install jupyter
```

```
[root@tos Software]# conda install jupyter
Fetching package metadata .....
Solving package specifications: .....

Package plan for installation in environment /anaconda3:

The following packages will be downloaded:

  package          |      build
  -----|-----
  conda-env-2.6.0   |          0      502 B
  requests-2.12.4    |      py35_0     800 KB
  pyopenssl-16.2.0   |      py35_0      70 KB
  conda-4.3.7        |      py35_0     491 KB
  -----
                           Total:     1.3 MB
```

```
The following packages will be UPDATED:

conda:      4.2.9-py35_0 --> 4.3.7-py35_0
pyopenssl: 16.0.0-py35_0 --> 16.2.0-py35_0
requests:   2.11.1-py35_0 --> 2.12.4-py35_0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6.100% [########################################] Time: 0:00:00 129.08 kB/s
requests-2.12.100% [########################################] Time: 0:00:11 70.63 kB/s
pyopenssl-16.2 100% [########################################] Time: 0:00:01 53.66 kB/s
conda-4.3.7-py 100% [########################################] Time: 0:00:04 104.02 kB/s
Extracting packages ...
[    COMPLETE      ]|########################################| 100%
Unlinking packages ...
[    COMPLETE      ]|########################################| 100%
Linking packages ...
[    COMPLETE      ]|########################################| 100%
dbus post-link :: /etc/machine-id not found ..
dbus post-link :: .. using /proc/sys/kernel/random/boot_id
[root@tos Software] #
```

Reference:

<https://www.sicara.ai/blog/2017-05-02-get-started-pyspark-jupyter-notebook-3-minutes>

<https://community.hortonworks.com/articles/75551/installing-and-exploring-spark-20-with-jupyter-not.html>

<https://medium.com/@bogdan.cojocar/how-to-run-scala-and-spark-in-the-jupyter-notebook-328a80090b3b>

-----Lab Ends Here -----