Contents

Setting up Lab Environment

Step1: Installing VMware Player from Training Bundle.

- Under training folder, navigate to "Additional Software" folder.
- Find the executable file named 'VmWarePlayer.exe' and complete the installation.

Step2: Extracting the VMware Image

- From the same training bundle, locate the folder named VM image> 'cloudera-quickstart-vm-5.12.0-0-vmware'
- Unzip/Extract the files and using the VMware player browse to this location to get started with virtual machine.

Step3: Loading the VMware Image

- Right click and select Open or Double click on 'VMware Player' and launch the same.
- Select the option as "Open a Virtual Machine"
- Navigate to the location where you have extracted the VMware Image in earlier step.

- This will start the Virtual Machine

Step4: Login to the VM

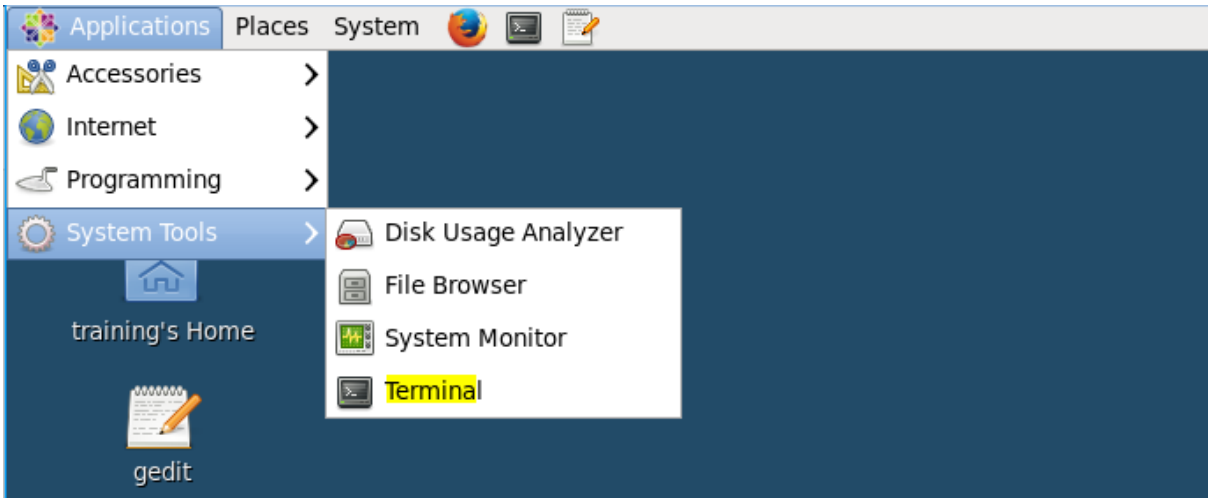
Exercise1: Practising Basic HDFS Commands

We will practise HDFS command line interface and web-based Hue File Browser to perform operations on files.

HDFS Command line Interface

- Open terminal

Navigate to Application > System Tools > Terminal



Operations on Files and Directories:

To view the content of root directory in HDFS

```
hdfs dfs -ls /
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 6 items
drwxrwxrwx   - hdfs  supergroup          0 2017-07-19 06:29 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2018-04-15 08:27 /hbase
drwxr-xr-x   - solr  solr                0 2017-07-19 06:31 /solr
drwxrwxrwt   - hdfs  supergroup          0 2018-04-15 08:27 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2017-07-19 06:31 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-07-19 06:31 /var
[cloudera@quickstart ~]$
```

To view the content of /user directory

```
hdfs dfs -ls /user
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user
Found 8 items
drwxr-xr-x   - cloudera cloudera            0 2017-07-19 06:28 /user/cloudera
drwxr-xr-x   - mapred   hadoop              0 2017-07-19 06:29 /user/history
drwxrwxrwx   - hive     supergroup          0 2017-07-19 06:31 /user/hive
drwxrwxrwx   - hue      supergroup          0 2017-07-19 06:30 /user/hue
drwxrwxrwx   - jenkins  supergroup          0 2017-07-19 06:29 /user/jenkins
drwxrwxrwx   - oozie    supergroup          0 2017-07-19 06:30 /user/oozie
drwxrwxrwx   - root     supergroup          0 2017-07-19 06:29 /user/root
drwxr-xr-x   - hdfs     supergroup          0 2017-07-19 06:31 /user/spark
[cloudera@quickstart ~]$
```

To view the content of home directory

```
hdfs dfs -ls
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls
[cloudera@quickstart ~]$
```

For an empty directory the prompt does not show any error while querying whereas if the directory doesn't exist it will throw an error.

Example: Query a non-existing directory say /music

```
hdfs dfs -ls /music
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /music
ls: `/music': No such file or directory
[cloudera@quickstart ~]$
```

Uploading Files to HDFS

- Create a directory named 'input' in HDFS, we will use this directory throughout the exercises.

```
hdfs dfs -mkdir input
```

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir input
[cloudera@quickstart ~]$
```

- Ensure that directory is created by running ls command

```
hdfs dfs -ls
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - cloudera cloudera          0 2018-04-15 08:34 input
[cloudera@quickstart ~]$
```

- Create a test file named 'sample.txt' on local filesystem

```
## Create sample.txt file using vi editor

Hello, How are you?

How was the day?

What are you doing?
```

```
[cloudera@quickstart ~]$ pwd
/home/cloudera
[cloudera@quickstart ~]$ vi sample.txt
[cloudera@quickstart ~]$ cat sample.txt
Hello, How are you?
How was the day?
What are you doing?

[cloudera@quickstart ~]$
```

- Upload the file to HDFS to input directory

```
hdfs dfs -put sample.txt input
```

```
[cloudera@quickstart ~]$ hdfs dfs -put sample.txt input
[cloudera@quickstart ~]$ hdfs dfs -ls input
Found 1 items
-rw-r--r--   1 cloudera cloudera         58 2018-04-15 08:37 input/sample.txt
[cloudera@quickstart ~]$
```

- To download the file from HDFS to local filesystem

```
cd /home/cloudera

hdfs dfs -get input/sample.txt /home/cloudera/Downloads
```

```
[cloudera@quickstart ~]$ hdfs dfs -get input/sample.txt /home/cloudera/Downloads
[cloudera@quickstart ~]$ ls /home/cloudera/Downloads
sample.txt
[cloudera@quickstart ~]$ cat /home/cloudera/Downloads/sample.txt
Hello, How are you?
How was the day?
What are you doing?

[cloudera@quickstart ~]$
```

- To remove the file from HDFS

```
hdfs dfs -rm -r input/sample.txt
```

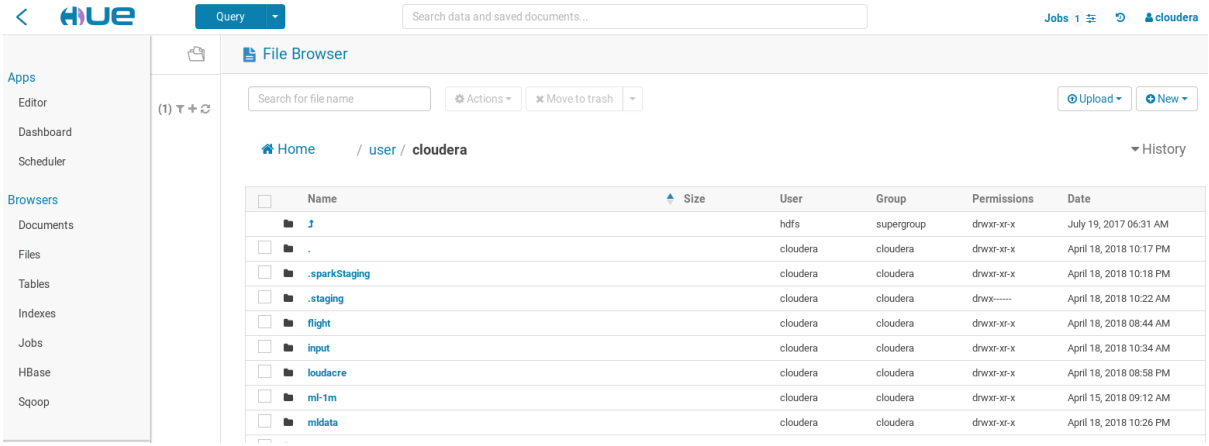
```
[cloudera@quickstart ~]$ hdfs dfs -rm -r input/sample.txt
Deleted input/sample.txt
[cloudera@quickstart ~]$ hdfs dfs -ls input
[cloudera@quickstart ~]$
```

Web-Based Interface (Hue File Browser)

- Open a web browser on your VM and navigate to bookmark tab and select HUE. This can be alternatively launched by specifying http://<hostname>:8888

- Enter username as 'cloudera' and password 'cloudera'.

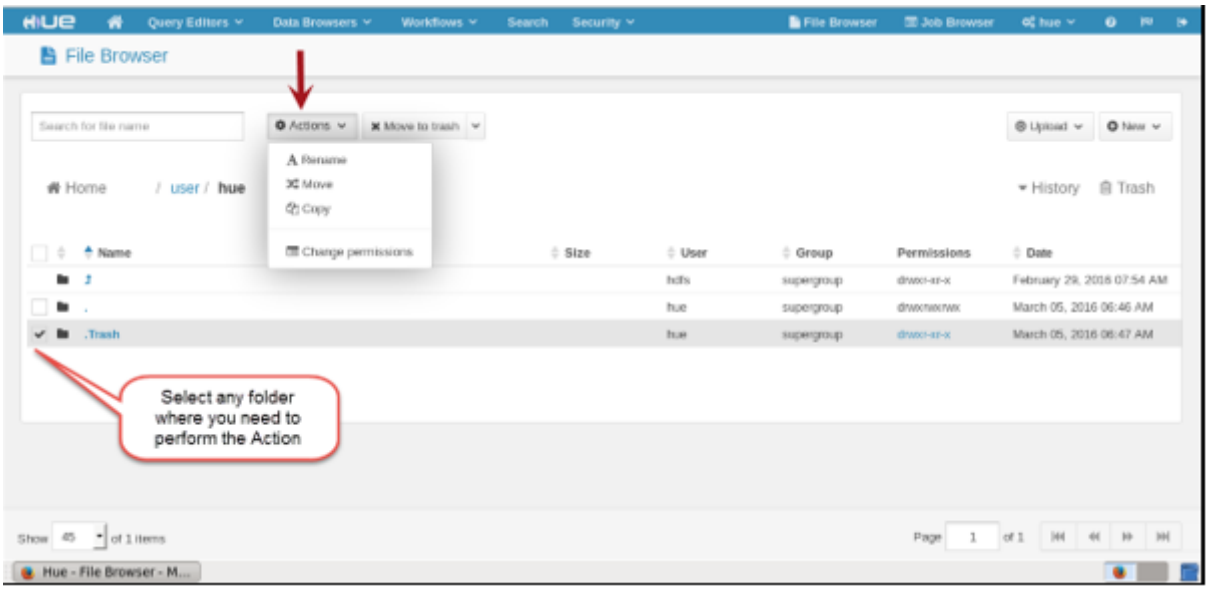
- Navigate to files under Browsers



By default, the content of your HDFS home directory will be displayed. Locate the input directory created in previous step on the file browser.

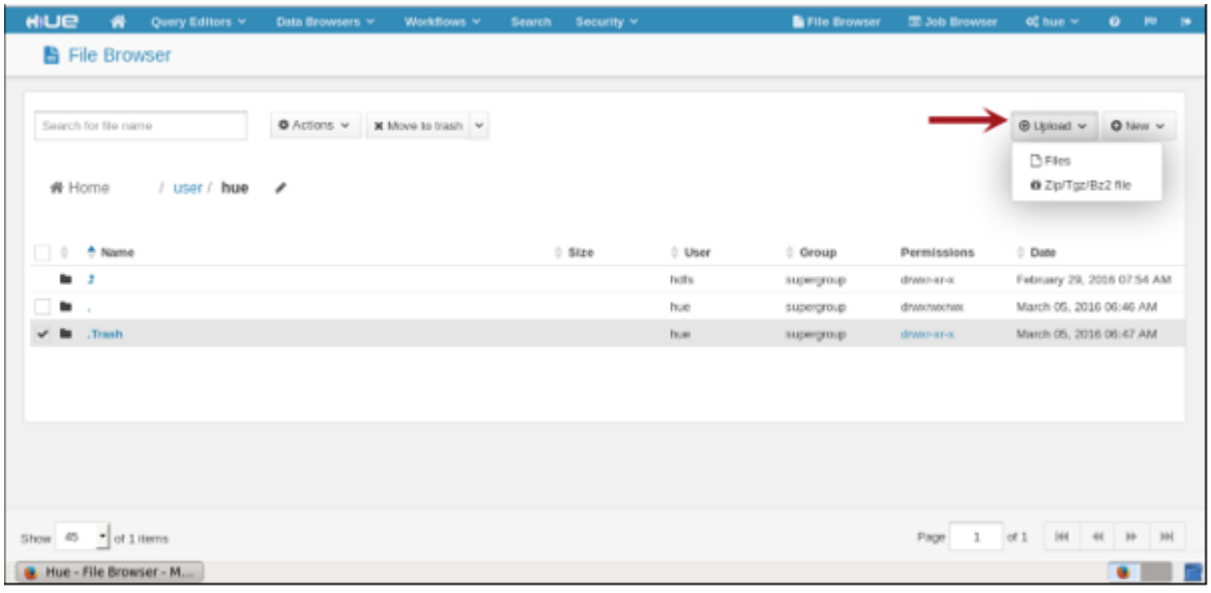
- Click on leading slash to view the content of root directory

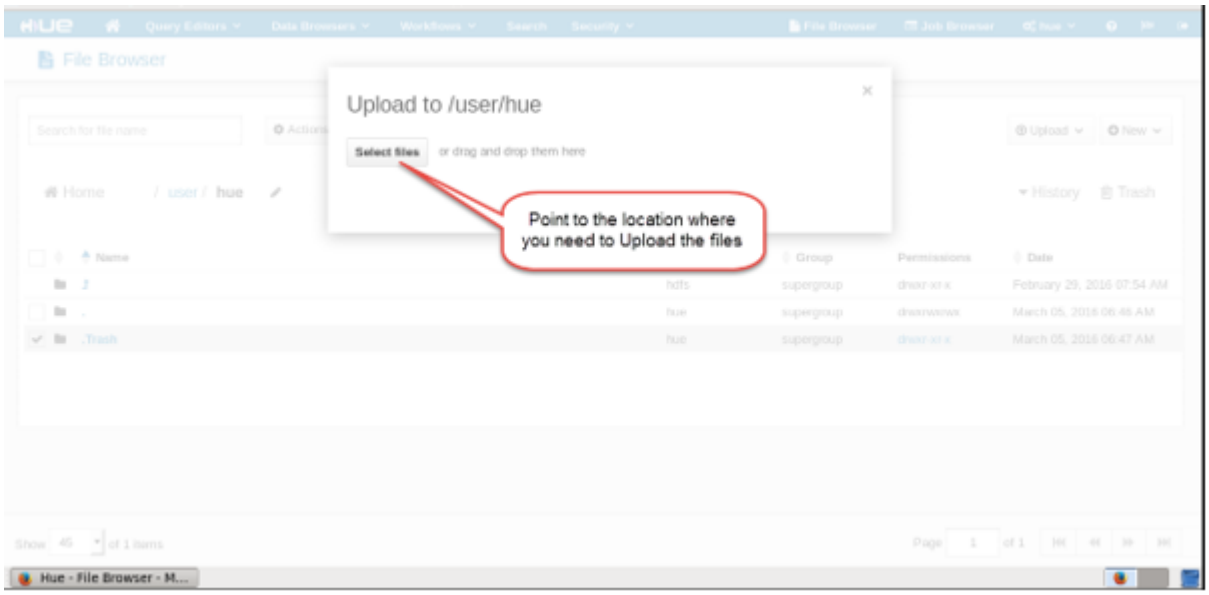
- To perform operation on a file, select the file and several options will be enabled



- To upload the file, click on Upload button. From the dropdown you can choose to upload a plain file or zipped file

- Click on Upload > Files > Select files





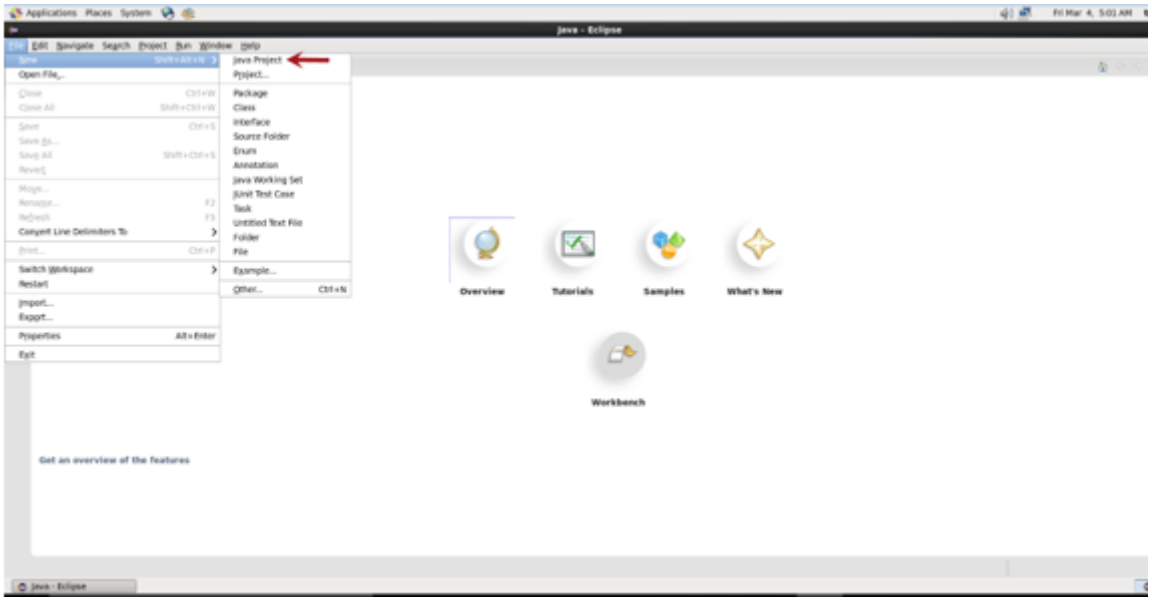
- To remove the file, move it to Trash

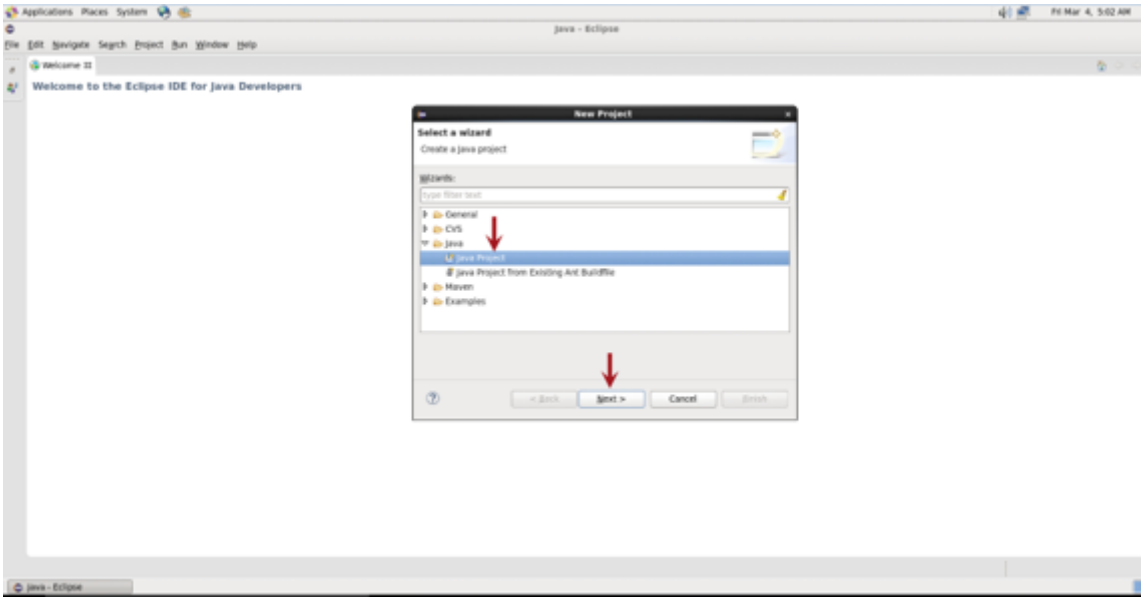
Exercise2: Running MapReduce Job

Prerequisites: Create an Eclipse WordCount Project

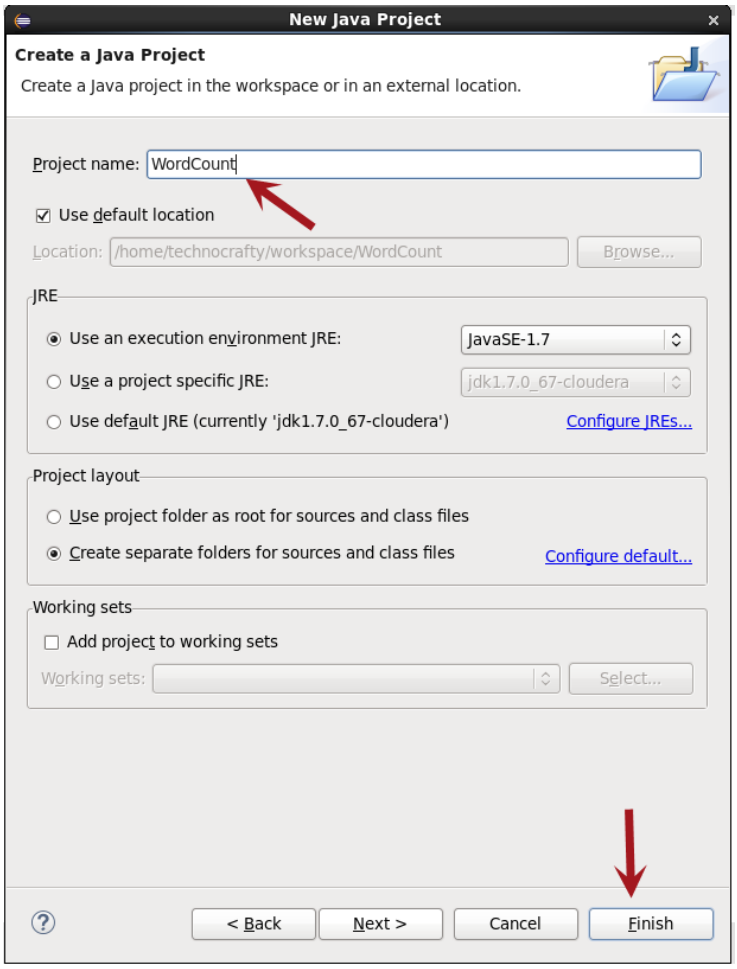
Step1: Select the workspace location and click on "OK".

Step2: Select File > New > Java Project > Next

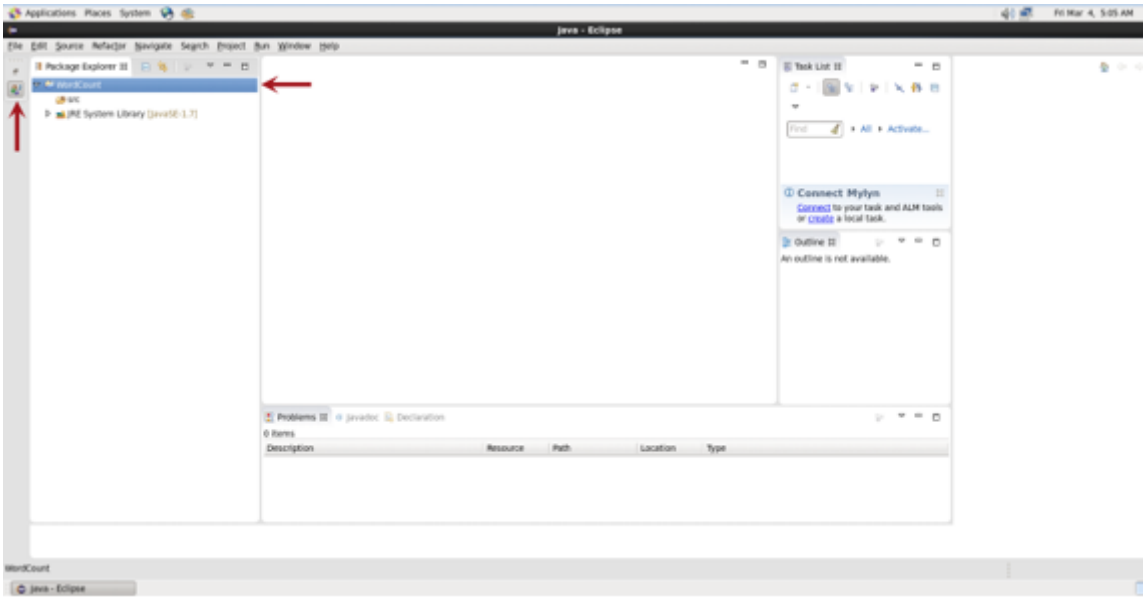




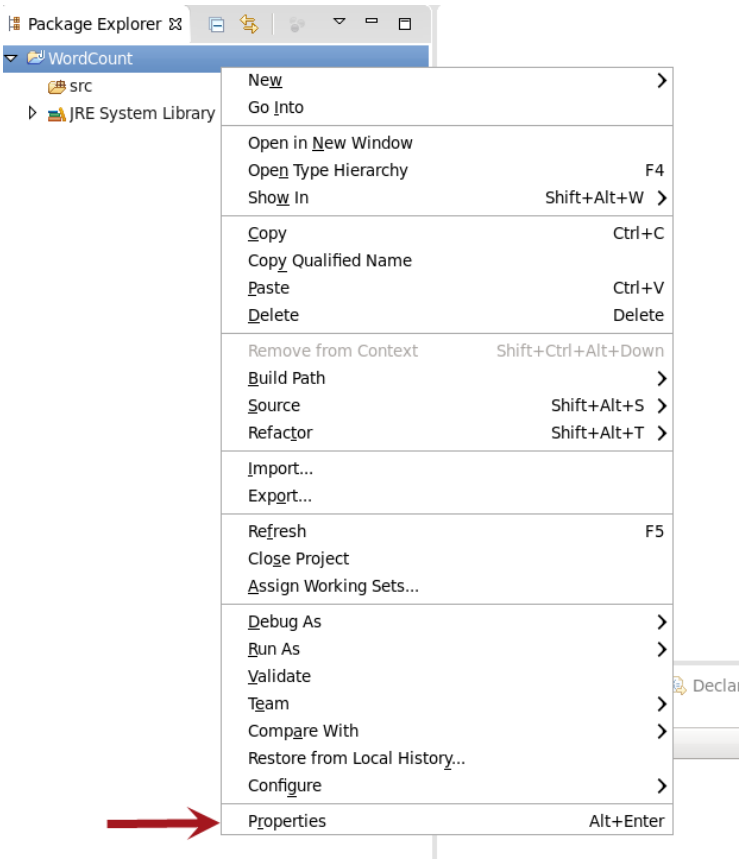
Step3: Name the project as "WordCount" and click "Finish"



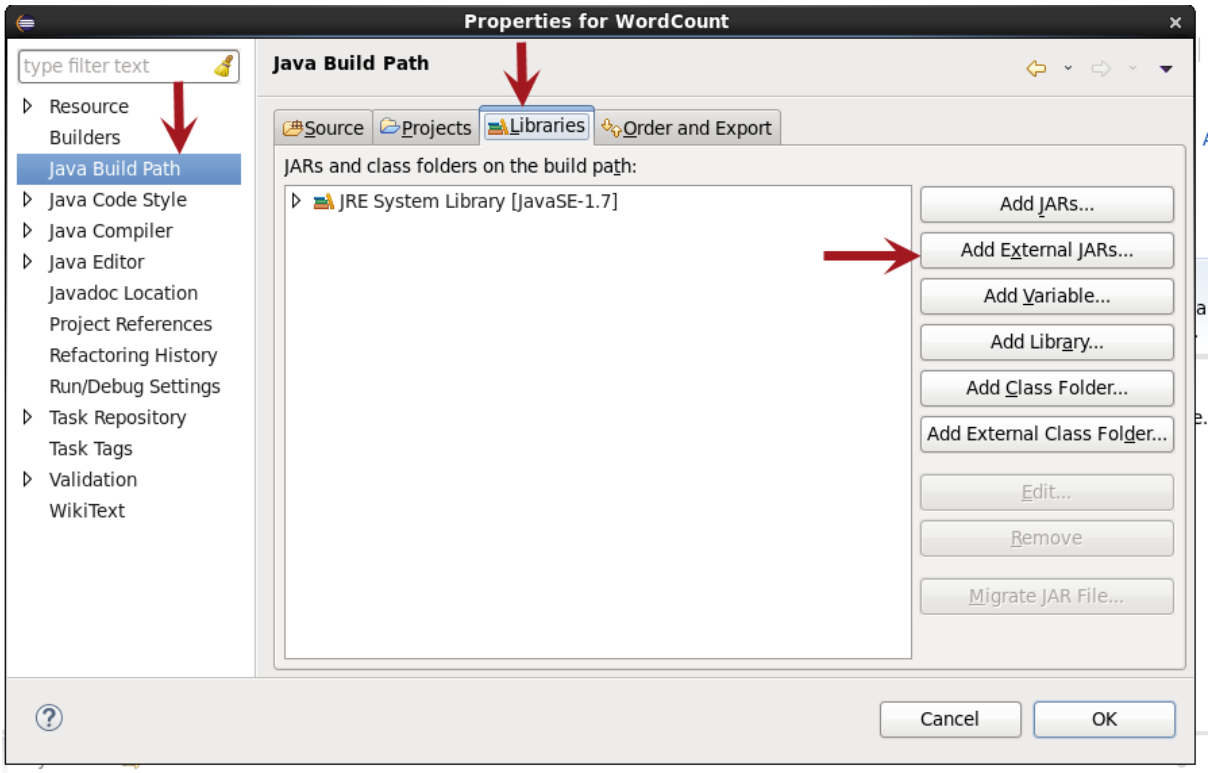
Step4: Expand the "Package Explorer" tab and locate your new project i.e. "WordCount".



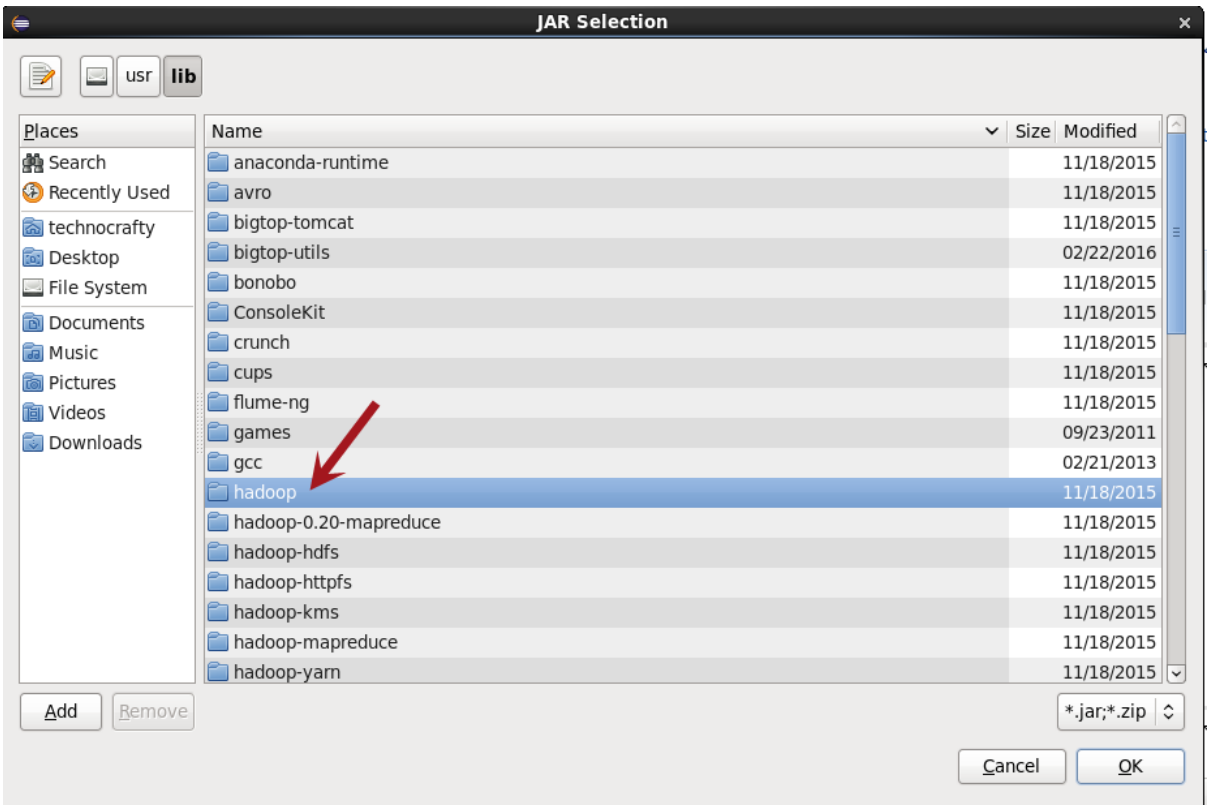
Step5: Export Hadoop Libraries by Right Click on WordCount project and selecting Properties



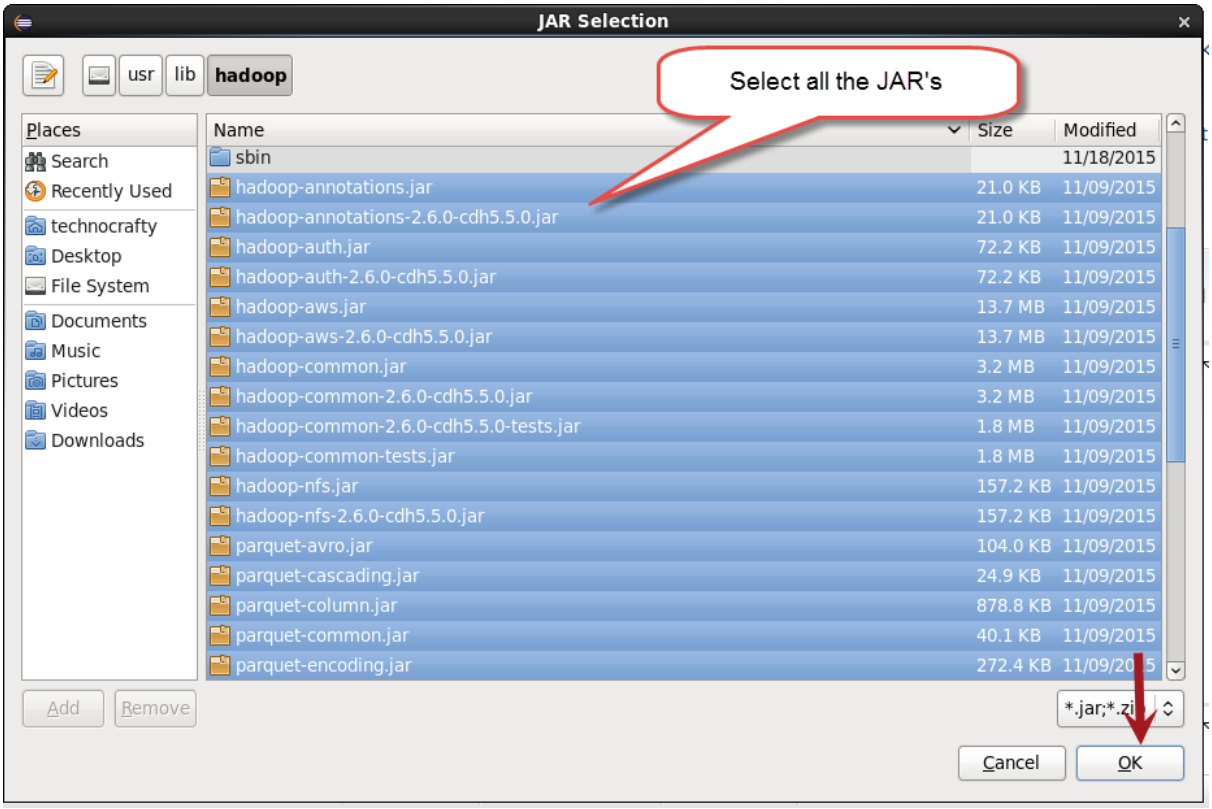
Step6: Click on Java Build Path from left panel and select "Libraries" tab from right panel view



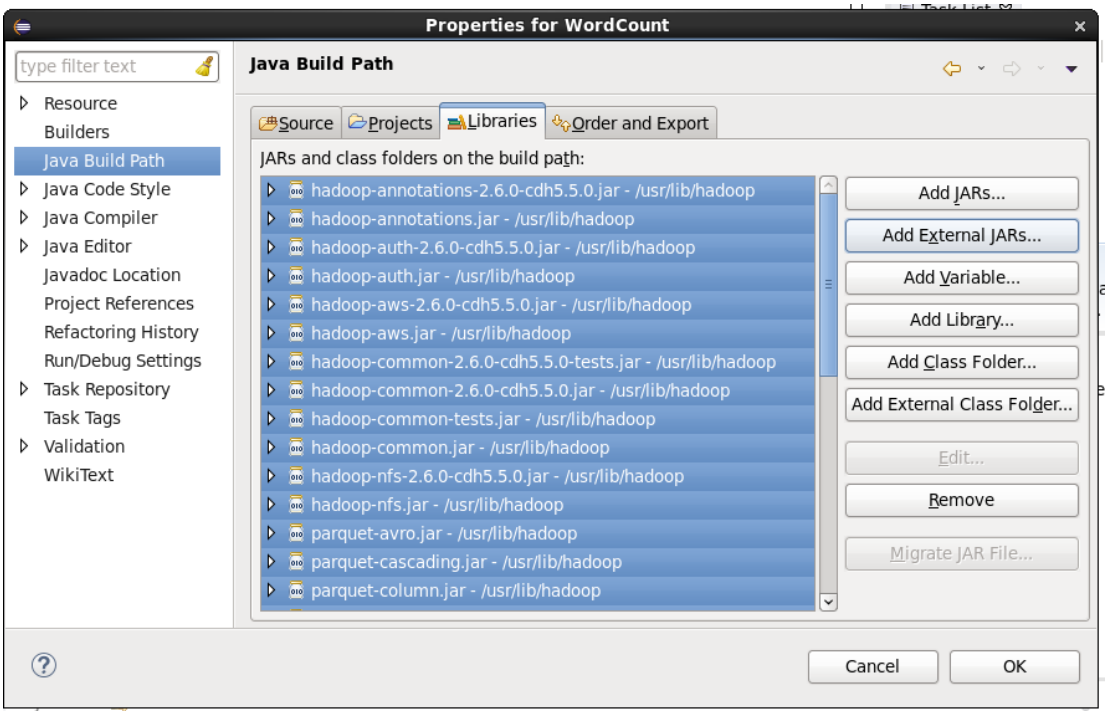
Step7: Select "Add External JAR…", then navigate to File System /usr/lib/hadoop location to import all jar files.



Step8: Select all the JAR available under this folder

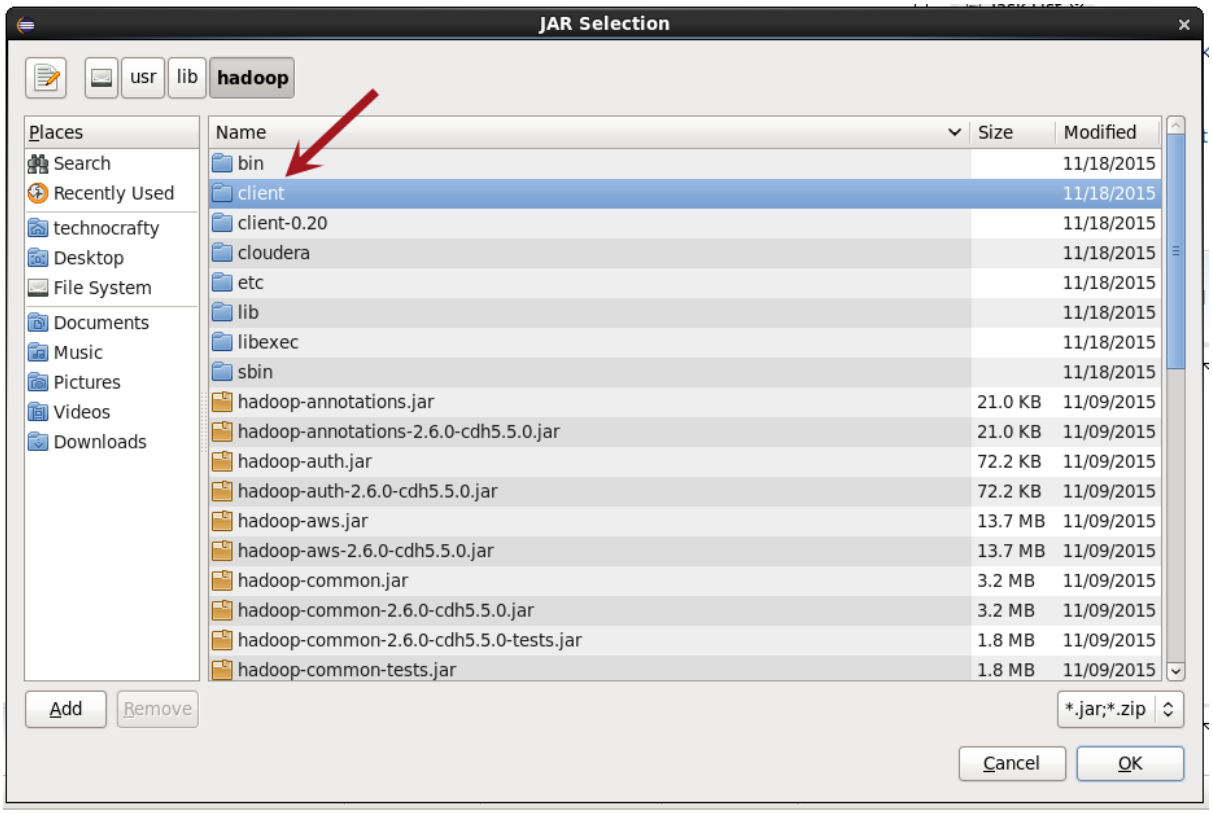


Click on "OK"

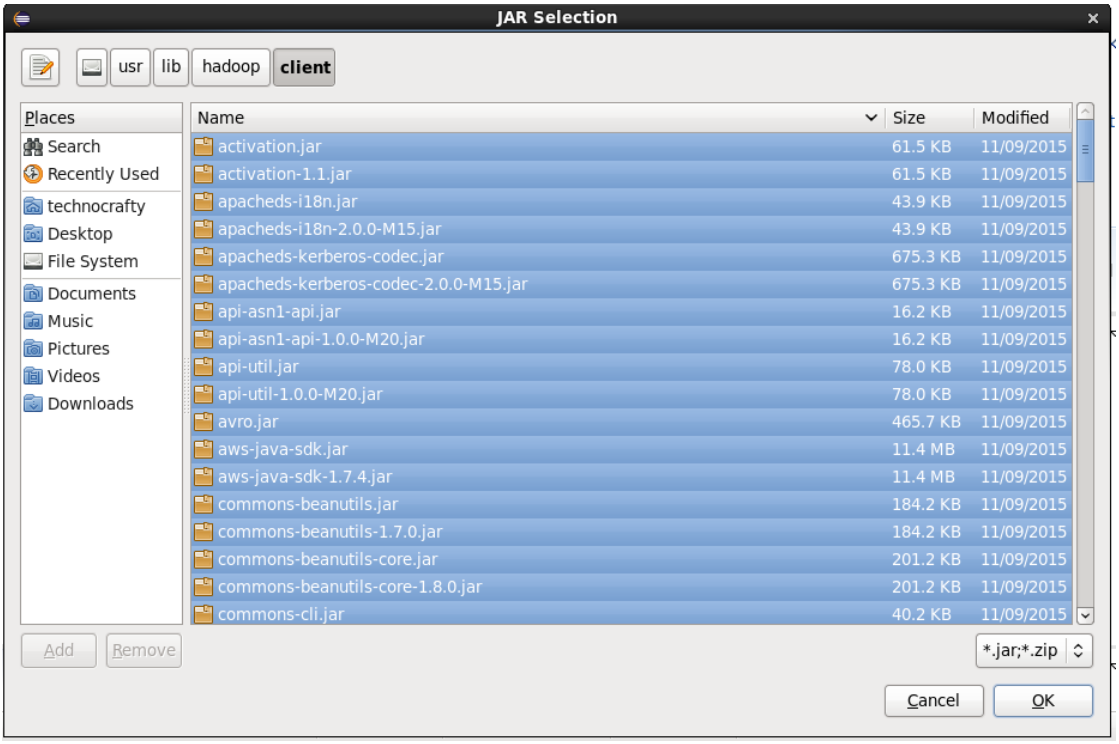


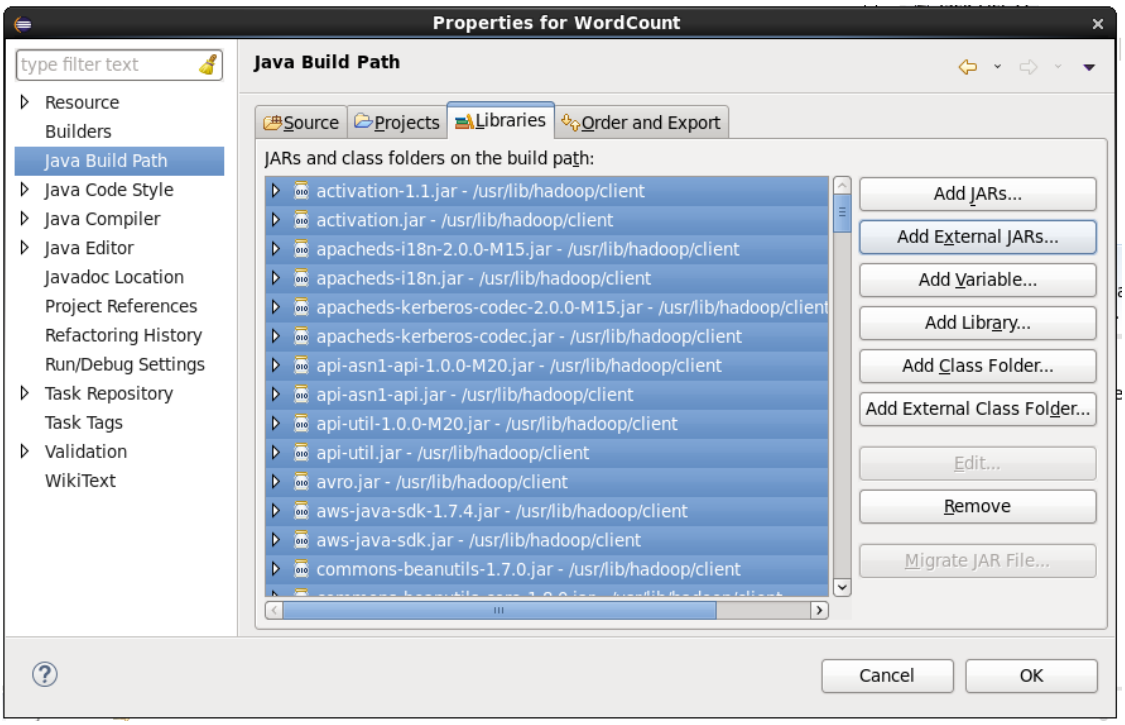
Step 9: Perform same steps for /usr/lib/hadoop-mapreduce and /usr/lib/hadoop-yarn

Step10: Now grab all the library JAR files under “Client”, by following the same method.

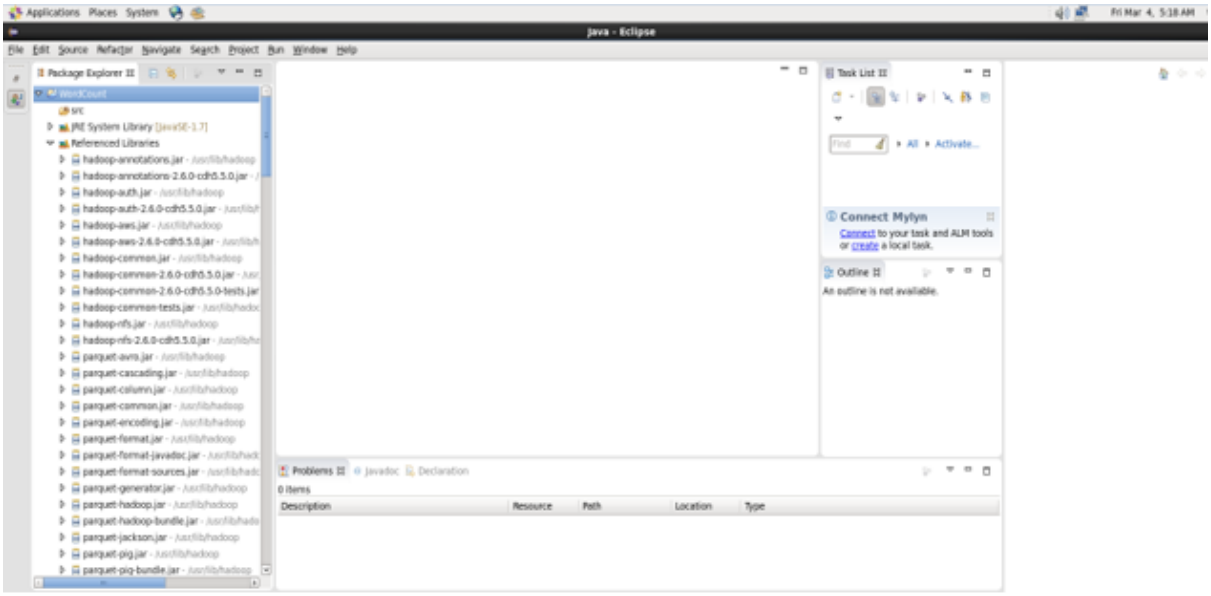


Select the files and click OK

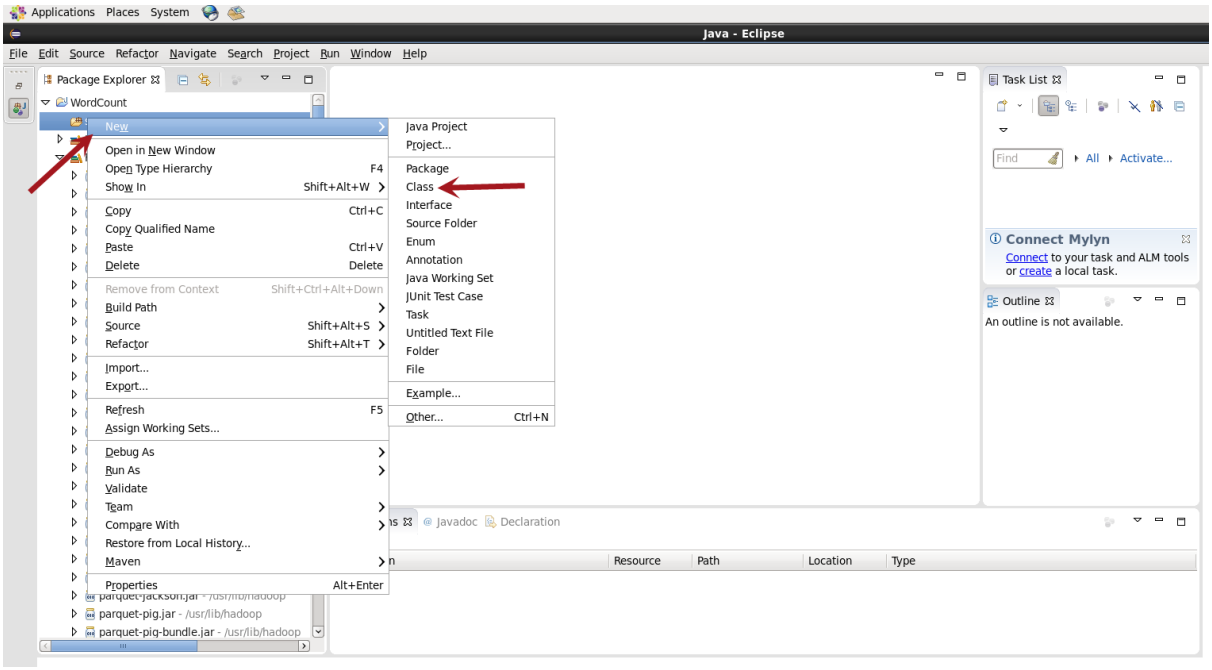




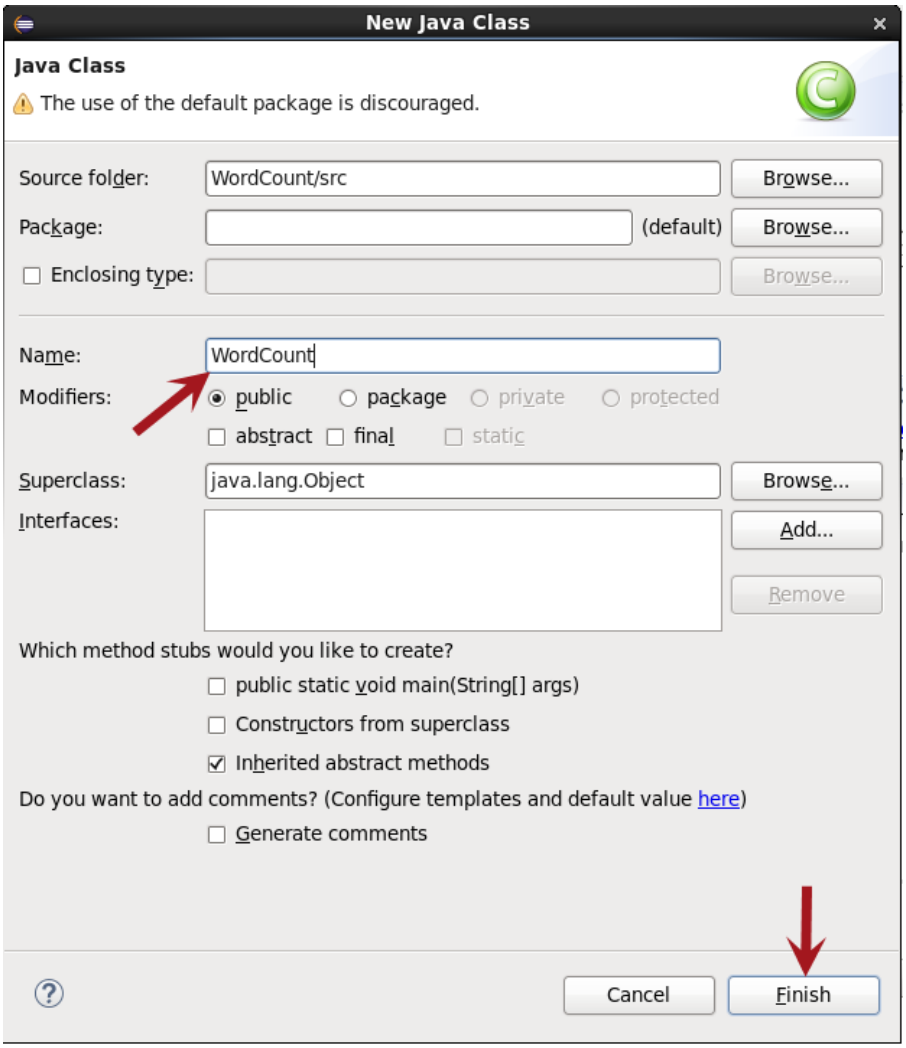
Step11: Finally, you will find all the JAR files under "Referenced Libraries"



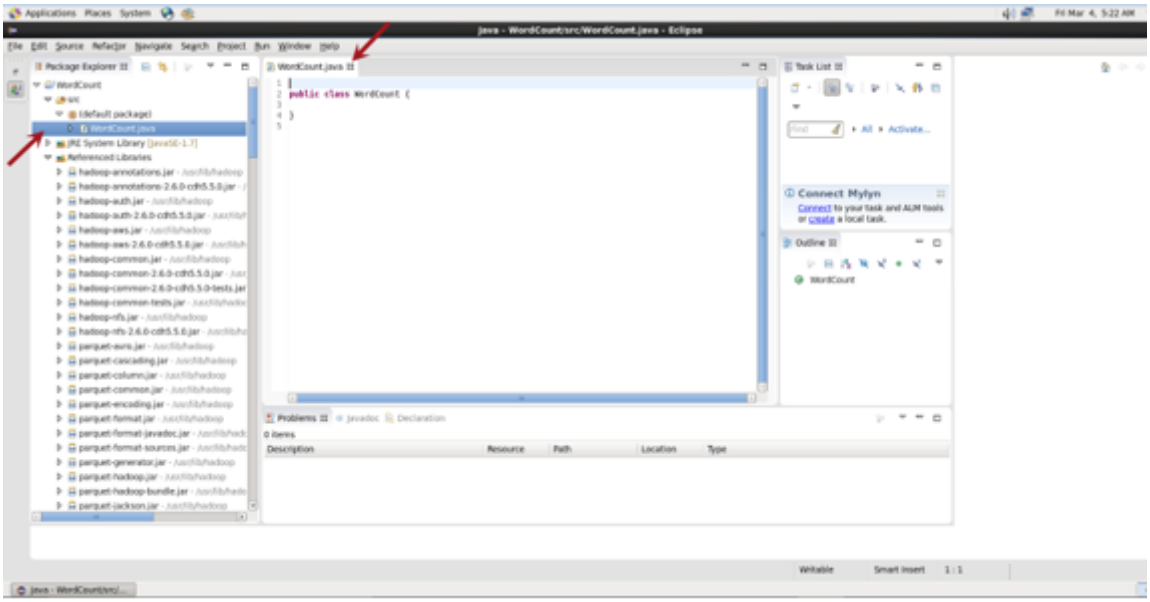
Step12: Creating the class files, right click on src under WordCount and select New > Class



Step13: Enter the name of Java Class as "WordCount" and click on Finish.



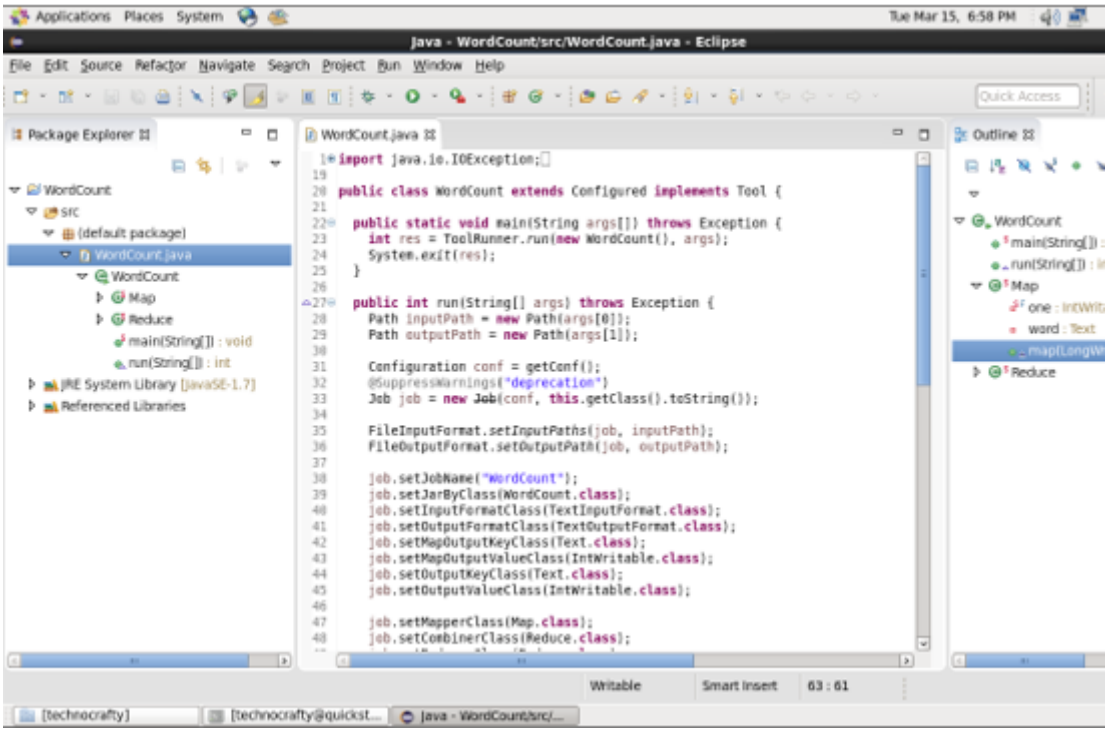
Step14: Now it’s ready to take your input Mapreduce program



Step15: MapReduce Wordcount Program

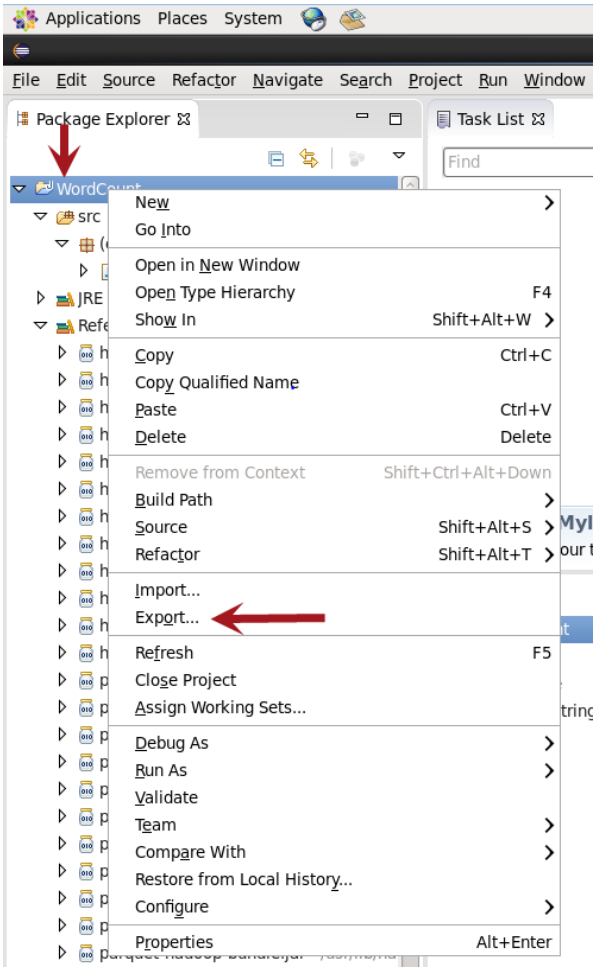
WordCount.txt

Step 16: Save the program

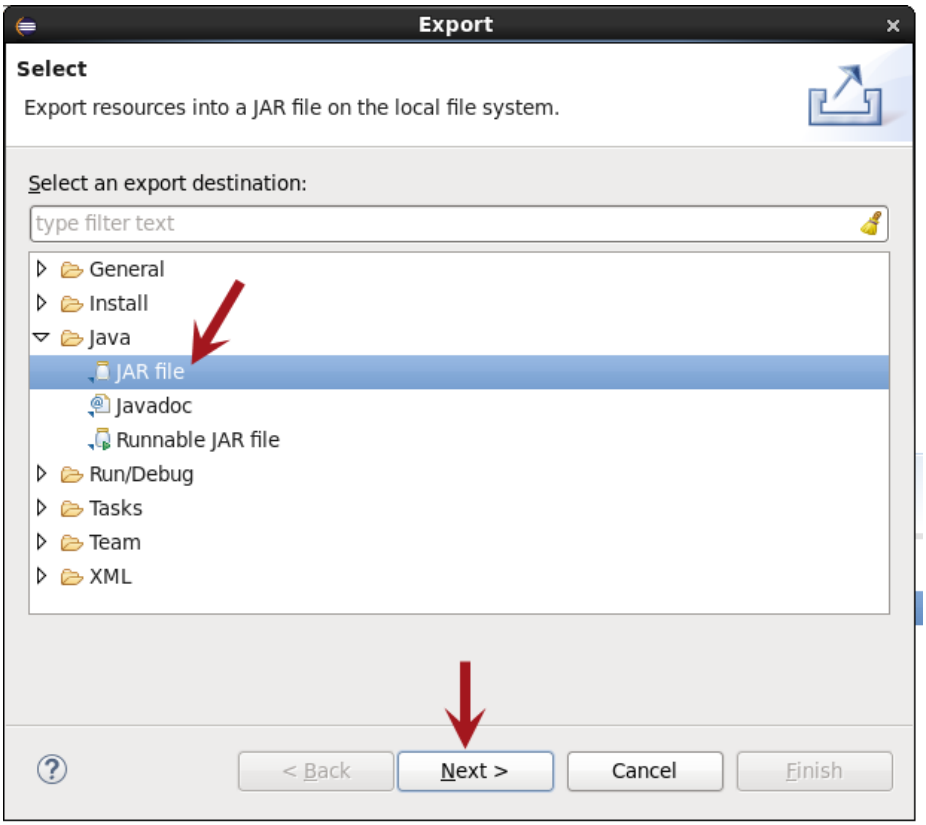


Ensure that there is no compilation error, before exporting the JAR file.

Step17: Exporting the JAR file, for that Right Click on WordCount project and select “Export”



Step 18: In the next panel, expand Java and select JAR file



Step19: Name the JAR as "WordCount.jar" and select OK.

Step20: Browse to the location where you want to save the JAR and select FINISH.

Step21: Verify the JAR file at the target workspace location on the terminal.

```
[cloudera@quickstart workspace]$ pwd
/home/cloudera/workspace
[cloudera@quickstart workspace]$ ls -ltr
total 12
drwxrwxr-x 6 cloudera cloudera 4096 Jul 19  2017 training
drwxrwxr-x 5 cloudera cloudera 4096 Apr 18 10:08 WordCount
-rw-rw-r-- 1 cloudera cloudera 3559 Apr 18 10:16 wordcount.jar
[cloudera@quickstart workspace]$
```

- Let's run a simple MapReduce WordCount program.

Step1: Import the data from local filesystem to HDFS

Under home directory of local filesystem, locate a folder named "datasets" followed by a file named Joyce.txt

```
[technocrafty@technocrafty dataset]$ pwd
/home/technocrafty/dataset
[technocrafty@technocrafty dataset]$ ls -ltr
total 4
-rw-rw-r-- 1 technocrafty technocrafty 764 Dec  6 18:44 joyce.txt
[technocrafty@technocrafty dataset]$
```

Step2: Create a directory named "input" in the HDFS and import "Joyce.txt" file.

```
[cloudera@quickstart dataset]$ hdfs dfs -put joyce.txt input
[cloudera@quickstart dataset]$ hdfs dfs -ls input
Found 2 items
-rw-r--r--   1 cloudera cloudera  382725387 2018-04-15 09:16 input/crime_incidents.csv
-rw-r--r--   1 cloudera cloudera        764 2018-04-18 10:19 input/joyce.txt
[cloudera@quickstart dataset]$
```

Step3: Now run the program by using the WordCount.jar created in previous steps

```
$ hadoop jar wordCount.jar WordCount input/joyce.txt output
```

```
[cloudera@quickstart workspace]$ hadoop jar wordcount.jar WordCount input/joyce.txt output
18/04/18 10:19:54 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/192.168.149.145:8032
18/04/18 10:19:55 INFO input.FileInputFormat: Total input paths to process : 1
18/04/18 10:19:55 INFO mapreduce.JobSubmitter: number of splits:1
18/04/18 10:19:55 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524061798209_0004
18/04/18 10:19:55 INFO impl.YarnClientImpl: Submitted application application_1524061798209_0004
18/04/18 10:19:55 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1524061798209_0004/
18/04/18 10:19:55 INFO mapreduce.Job: Running job: job_1524061798209_0004
```

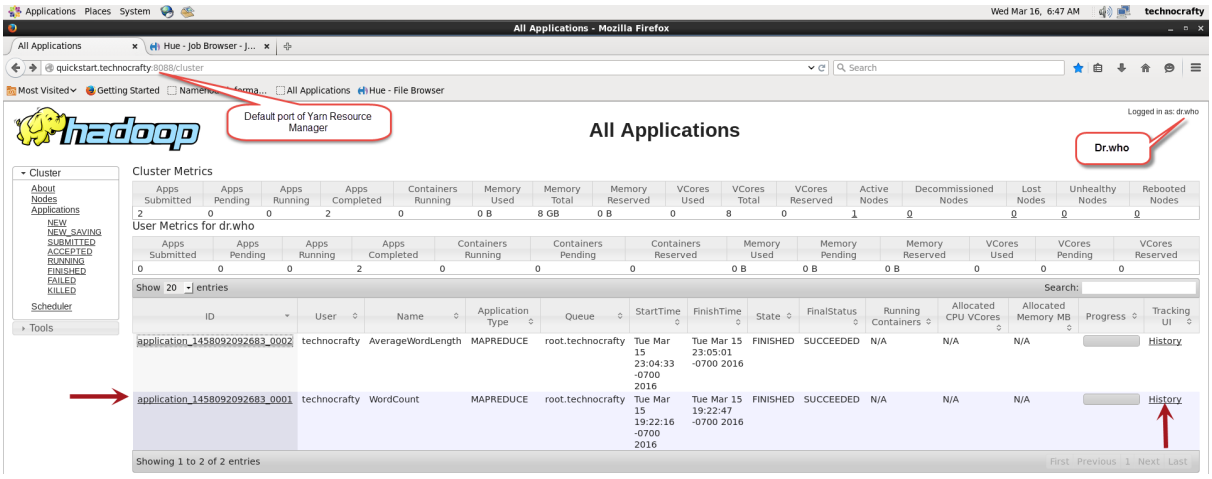
```
    File System Counters
        FILE: Number of bytes read=725062
        FILE: Number of bytes written=1675035
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=1573213
        HDFS: Number of bytes written=527547
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
```

```
          Total time spent by all maps in occupied slots
(ms)=7395
          Total time spent by all reduces in occupied slots
(ms)=8454
          Total time spent by all map tasks (ms)=7395
          Total time spent by all reduce tasks (ms)=8454
          Total vcore-seconds taken by all map tasks=7395
          Total vcore-seconds taken by all reduce tasks=8454
          Total megabyte-seconds taken by all map
tasks=7572480
          Total megabyte-seconds taken by all reduce
tasks=8656896
     Map-Reduce Framework
          Map input records=33056
          Map output records=267980
          Map output bytes=2601827
          Map output materialized bytes=725062
          Input split bytes=134
          Combine input records=267980
          Combine output records=50094
          Reduce input groups=50094
          Reduce shuffle bytes=725062
          Reduce input records=50094
          Reduce output records=50094
          Spilled Records=100188
          Shuffled Maps =1
          Failed Shuffles=0
          Merged Map outputs=1
          GC time elapsed (ms)=169
          CPU time spent (ms)=4370
          Physical memory (bytes) snapshot=344633344
          Virtual memory (bytes) snapshot=3008765952
          Total committed heap usage (bytes)=226562048
     Shuffle Errors
          BAD_ID=0
          CONNECTION=0
          IO_ERROR=0
          WRONG_LENGTH=0
          WRONG_MAP=0
          WRONG_REDUCE=0
     File Input Format Counters
          Bytes Read=1573079
     File Output Format Counters
          Bytes Written=527547
```

Check the output file:

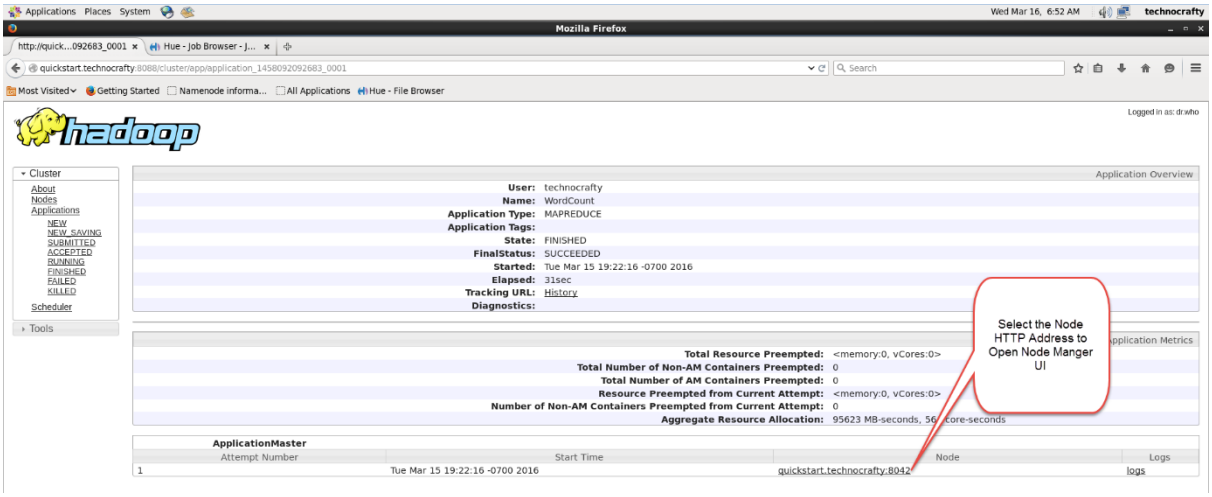
```
[cloudera@quickstart dataset]$ hdfs dfs -ls output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-18 10:20 output/_SUCCESS
-rw-r--r--   1 cloudera cloudera        827 2018-04-18 10:20 output/part-r-00000
[cloudera@quickstart dataset]$
```

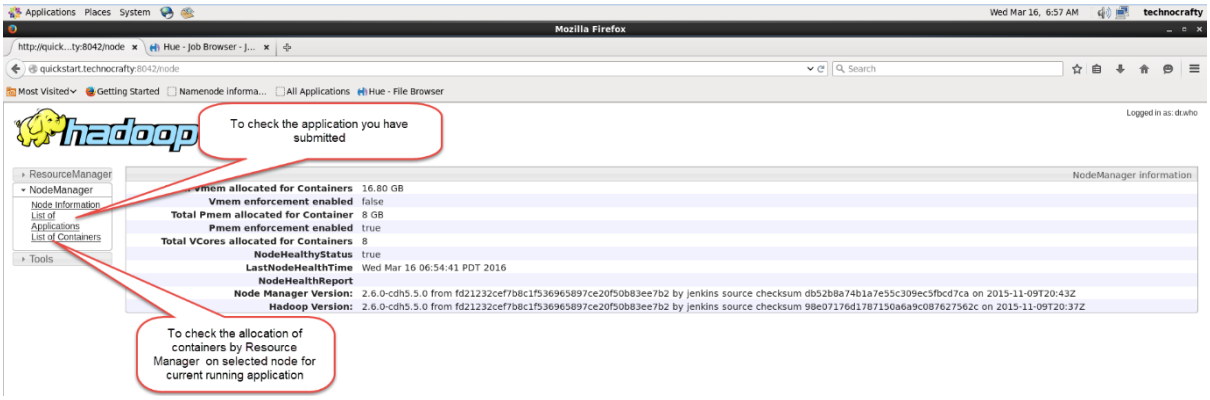
The same counters and job status can be checked from Resource Manager URL http://quickstart.cloudera:8088/cluster



Note

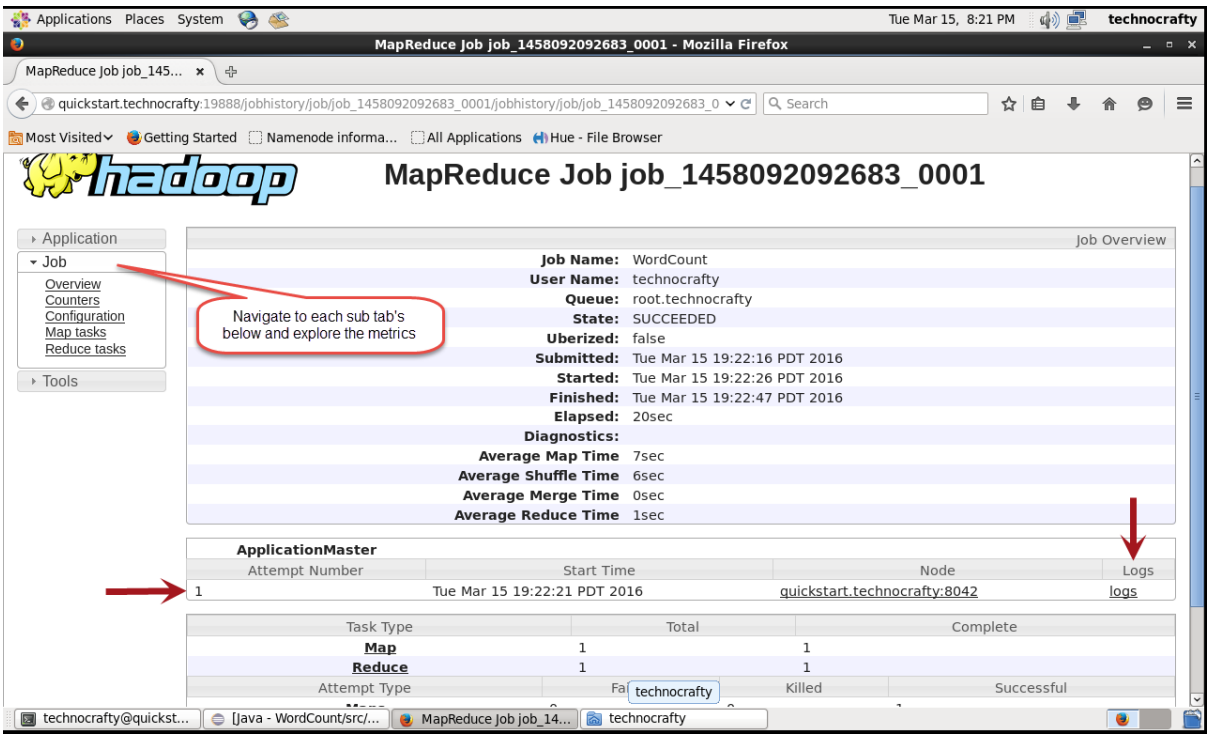
You may notice that YARN says you are logged in as dr.who. This is what is displayed when user authentication is disabled for the cluster. If user authentication were enabled, you would have to log in as a valid user to view the YARN UI, and your actual user name would be displayed, together with user metrics such as how many applications you had run, how much system resources your applications used and so on.





Note

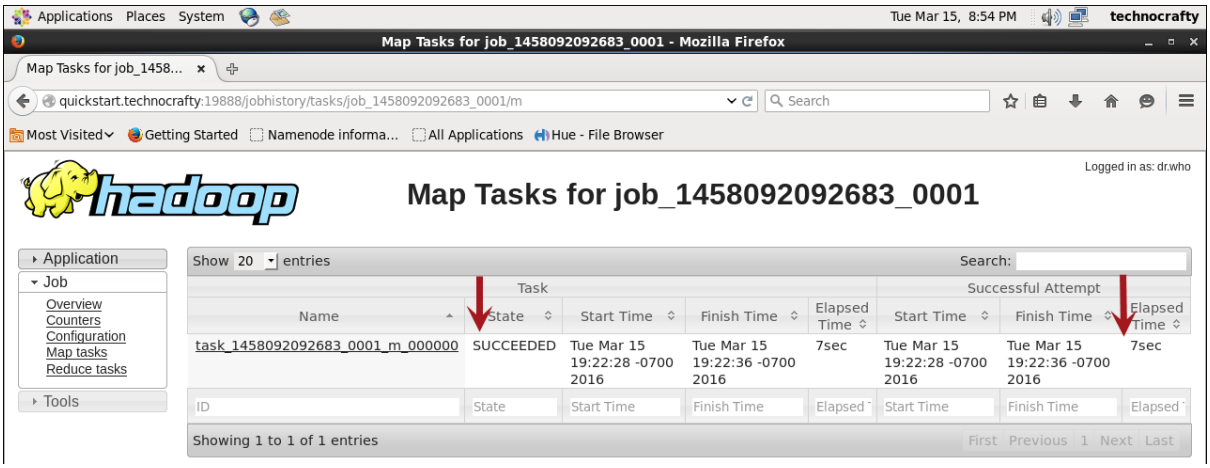
No container information will be available for stopped or completed application. Application must be active or running to view the allocation of containers by Resource Manager.

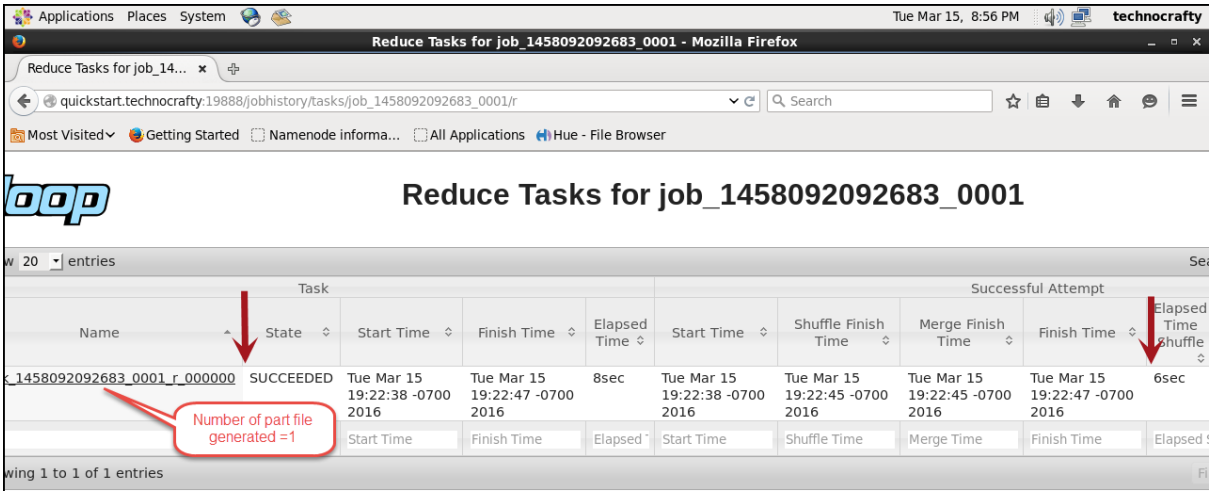


Counter Group	Counters			
File System Counters	Name	Map	Reduce	Total
	FILE: Number of bytes read	0	725062	725062
	FILE: Number of bytes written	837545	837490	1675035
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1573213	0	1573213
	HDFS: Number of bytes written	0	527547	527547
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	3	3	6
	HDFS: Number of write operations	0	2	2
Job Counters	Name	Map	Reduce	Total
	Data-local map tasks	0	0	1
	Launched map tasks	0	0	1
	Launched reduce tasks	0	0	1
	Total megabyte-seconds taken by all map tasks	0	0	7572480
	Total megabyte-seconds taken by all reduce tasks	0	0	8656896
	Total time spent by all map tasks (ms)	0	0	7395
	Total time spent by all maps in occupied slots (ms)	0	0	7395
	Total time spent by all reduce tasks (ms)	0	0	8454
	Total time spent by all reduces in occupied slots (ms)	0	0	8454
	Total vcore-seconds taken by all map tasks	0	0	7395
	Total vcore-seconds taken by all reduce tasks	0	0	8454

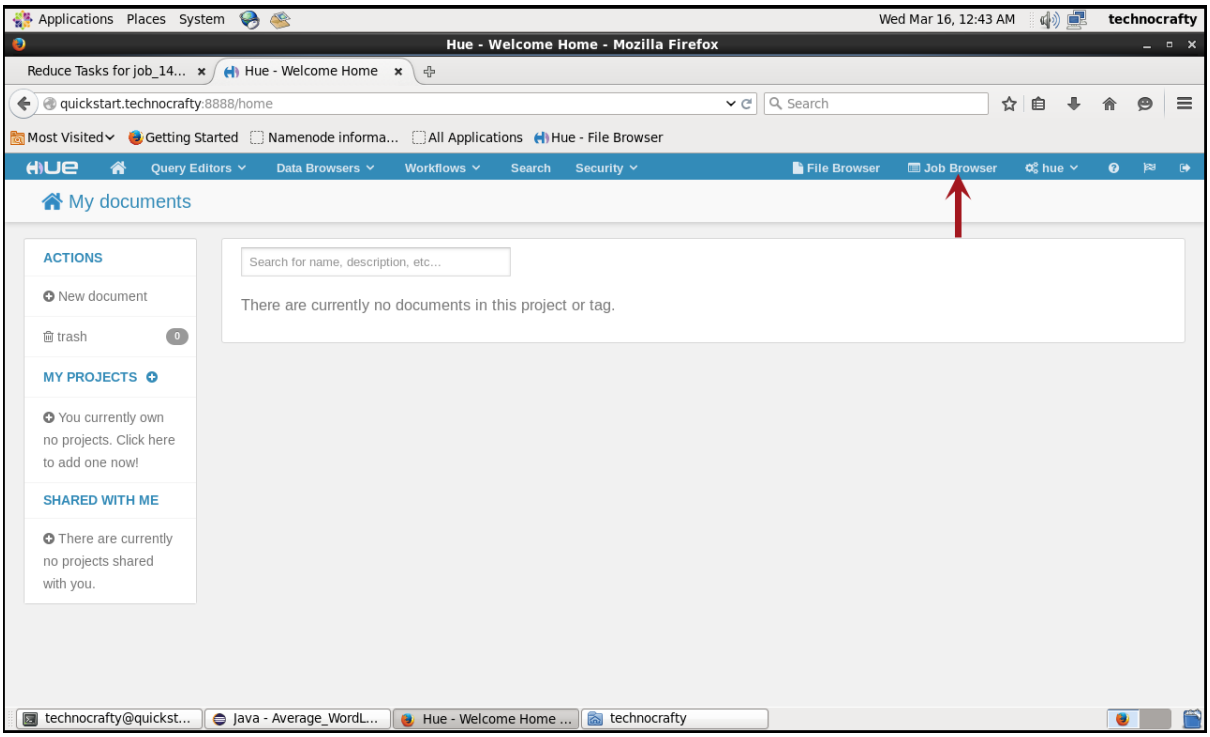
	Name	Map	Reduce	Total
Map-Reduce Framework	Combine input records	267980	0	267980
	Combine output records	50094	0	50094
	CPU time spent (ms)	2350	2020	4370
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	90	79	169
	Input split bytes	134	0	134
	Map input records	33056	0	33056
	Map output bytes	2601827	0	2601827
	Map output materialized bytes	725062	0	725062
	Map output records	267980	0	267980
	Merged Map outputs	0	1	1
	Physical memory (bytes) snapshot	223735808	120897536	344633344
	Reduce input groups	0	50094	50094
	Reduce input records	0	50094	50094
	Reduce output records	0	50094	50094
	Reduce shuffle bytes	0	725062	725062
	Shuffled Maps	0	1	1
	Spilled Records	50094	50094	100188
	Total committed heap usage (bytes)	165744640	60817408	226562048
	Virtual memory (bytes) snapshot	1501155328	1507610624	3008765952

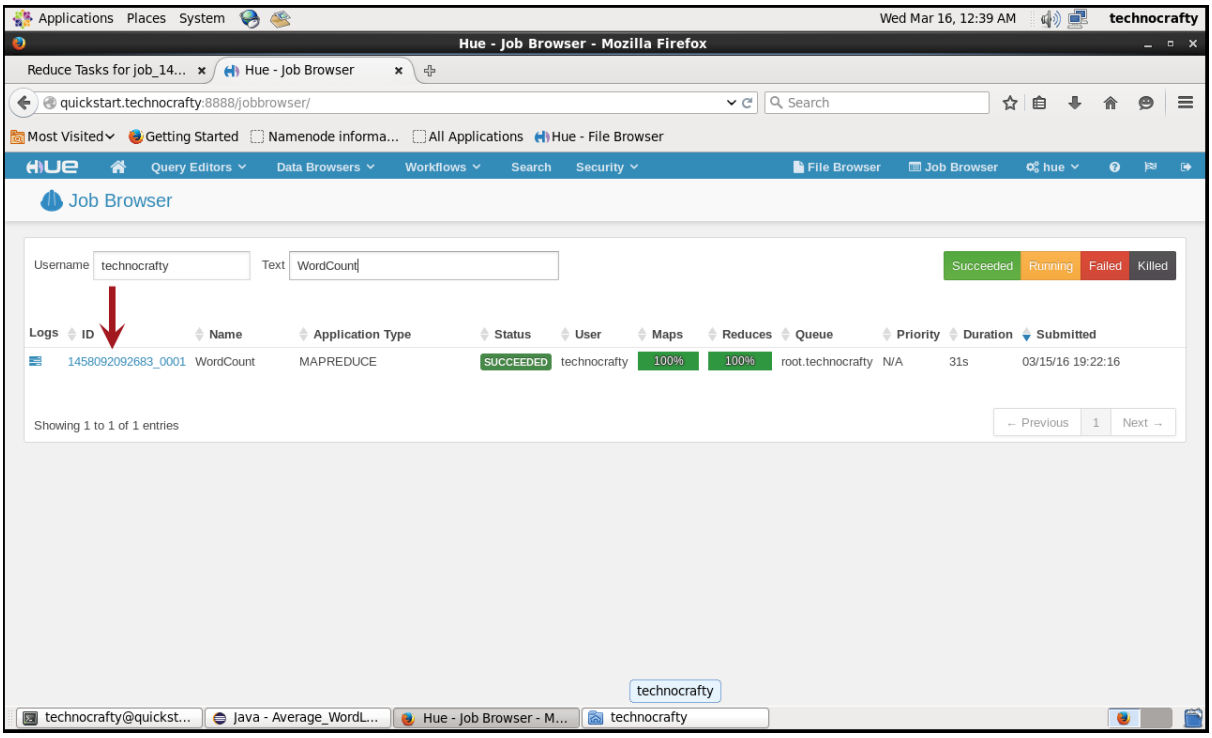
	Name	Map	Reduce	Total
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Name	Map	Reduce	Total
	Bytes Read	1573079	0	1573079
File Output Format Counters	Name	Map	Reduce	Total
	Bytes Written	0	527547	527547

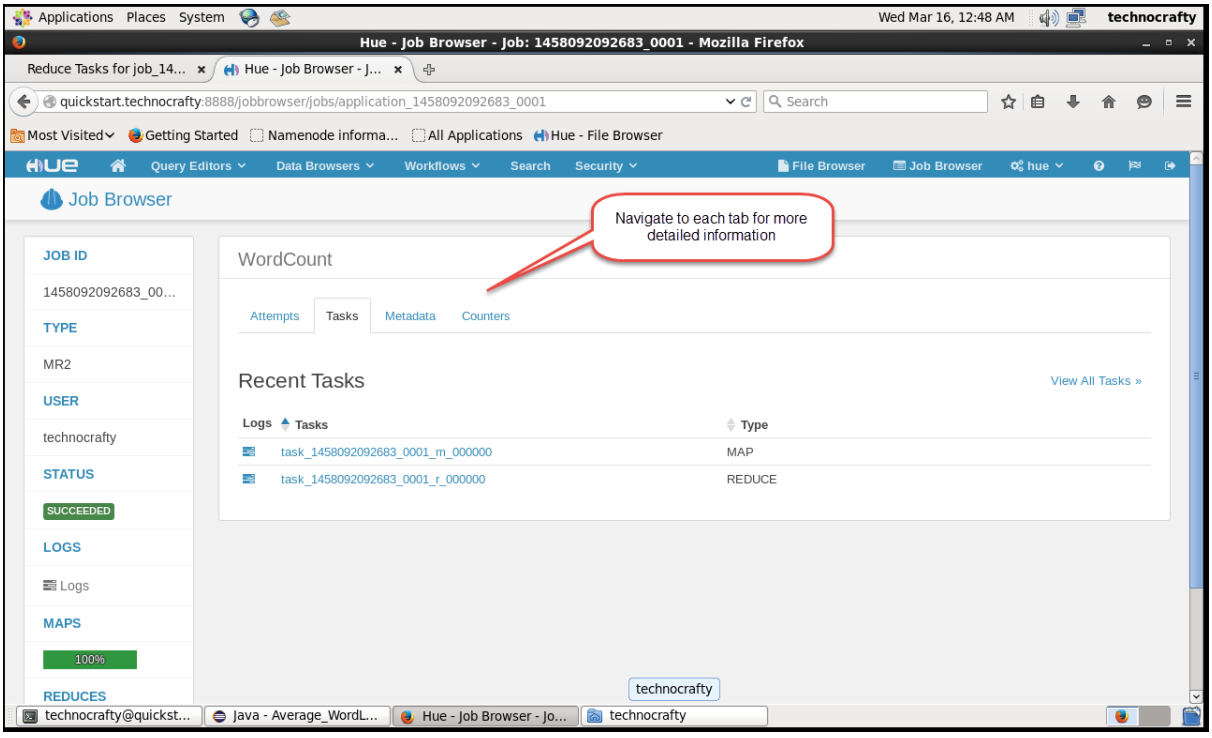




➢ On HUE browser, navigate to Job Browser







➢ Let's run the same job with multiple reducers

```
$ hadoop jar wordcount.jar WordCount -Dmapred.reduce.tasks=4
input/joyce.txt output1
```

```
[cloudera@quickstart dataset]$ hdfs dfs -ls output1
Found 5 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-18 10:22 output1/_SUCCESS
-rw-r--r--   1 cloudera cloudera        286 2018-04-18 10:22 output1/part-r-00000
-rw-r--r--   1 cloudera cloudera        204 2018-04-18 10:22 output1/part-r-00001
-rw-r--r--   1 cloudera cloudera        149 2018-04-18 10:22 output1/part-r-00002
-rw-r--r--   1 cloudera cloudera        188 2018-04-18 10:22 output1/part-r-00003
[cloudera@quickstart dataset]$
```

The number of output files generated is equal to the number of reducers.

Compare the metrics count while running the same Mapreduce program with four reducers with earlier output.

Exercise3: Just Enough Scala

Will we cover following topics in this Hands-On:

- Setting up Scala
- Performing Basic Input/Output
- Arithmetic Operations
- Scala Variables and Typing
- Expression
- Control Structure
- Functions
- Procedures
- Collections
  - Tuples, Lists and Maps
- Iterating with Data efficiently
- Project using all above concepts

Setting Up Scala

Note: This installation step can be skipped, to practice scala commands one can also make use of spark-shell.

Download Scala Binaries from http://www.scala-lang.org/download/

Unzip the scala-2.11.6.tgz file using the following command as shown below.

```
tar -xvf scala-2.12.4.tgz
```

```
[training@localhost Downloads]$ tar -xvf scala-2.12.4.tgz
scala-2.12.4/
scala-2.12.4/lib/
scala-2.12.4/lib/scala-swing_2.12-2.0.0.jar
scala-2.12.4/lib/scala-reflect.jar
scala-2.12.4/lib/scala-parser-combinators_2.12-1.0.6.jar
scala-2.12.4/lib/scala-compiler.jar
scala-2.12.4/lib/scala-library.jar
scala-2.12.4/lib/scala-xml_2.12-1.0.6.jar
scala-2.12.4/lib/scalap-2.12.4.jar
scala-2.12.4/lib/jline-2.14.5.jar
scala-2.12.4/bin/
```

Now we are in the downloads directory where Scala binaries are present. Just go to the bin directory.

```
[training@localhost Downloads]$ cd scala-2.12.4
[training@localhost scala-2.12.4]$ ls -ltr
total 16
drwxrwxr-x 3 training training 4096 Oct 11 00:59 man
drwxrwxr-x 2 training training 4096 Oct 11 00:59 lib
drwxrwxr-x 4 training training 4096 Oct 11 00:59 doc
drwxrwxr-x 2 training training 4096 Oct 11 00:59 bin
[training@localhost scala-2.12.4]$ cd bin
[training@localhost bin]$ ls -ltr
total 80
-rwxrwxr-x 1 training training 4993 Oct 11 00:59 scalap.bat
-rwxrwxr-x 1 training training 6233 Oct 11 00:59 scalap
-rwxrwxr-x 1 training training 4995 Oct 11 00:59 scaladoc.bat
-rwxrwxr-x 1 training training 6234 Oct 11 00:59 scaladoc
-rwxrwxr-x 1 training training 4987 Oct 11 00:59 scalac.bat
-rwxrwxr-x 1 training training 6230 Oct 11 00:59 scalac
-rwxrwxr-x 1 training training 5013 Oct 11 00:59 scala.bat
-rwxrwxr-x 1 training training 6243 Oct 11 00:59 scala
-rwxrwxr-x 1 training training 5005 Oct 11 00:59 fsc.bat
-rwxrwxr-x 1 training training 6239 Oct 11 00:59 fsc
```

Now enter the scala shell as shown below.

```
$./scala
```

```
[training@localhost Downloads]$ cd scala-2.12.4
[training@localhost scala-2.12.4]$ cd bin
[training@localhost bin]$ ./scala
Welcome to Scala 2.12.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60).
Type in expressions for evaluation. Or try :help.

scala>
```

Scala shell also supports directive commands, they start with a colon, for example

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicits [-v]          show the implicits in scope
:javap <path|class>      disassemble a file or class name
:line <id>|<line>        place line(s) at the end of history
:load <path>             interpret lines in a file
:paste [-raw] [path]     enter paste mode or paste a file
:power                   enable power user mode
:quit                    exit the interpreter
:replay [options]        reset the repl and replay all previous commands
:require <path>          add a jar to the classpath
```

To quit the shell

```
scala> :quit
[training@localhost bin]$
```

Performing Basic Input/Output

Writing and Running Code in Scala Shell

Launch Scala shell or just type spark-shell

```
[training@localhost bin]$ ./scala
Welcome to Scala 2.12.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60).
Type in expressions for evaluation. Or try :help.

scala>
```

The Scala shell enables you to have an interactive session with Scala. You will use this to explore some of the basic principles of the Scala language.

Create an immutable variable called ab and assign the integer value 25 to it:

```
scala> val ab = 25
ab: Int = 25

scala>
```

The name, type, and value of the variable are displayed as confirmation.

Enter the variable as a command:

```
scala> ab
res0: Int = 25

scala>
```

Note two things: first, that a variable entered without any command returns its value, because all commands in Scala are expressions that return a value; and that the return value is assigned to an automatically created result variable, res0.

Use the built-in print function to display the value of ab:

```
scala> print(ab)
25
scala>
```

Try entering a string:

```
scala> "Hello, World"
res2: String = Hello, World

scala>
```

Entering a literal like this also returns a value, and the value is assigned to a result variable.

Try entering an invalid command; enter the print function without the final parenthesis:

```
scala> print(ab
     |
     |
You typed two blank lines.  Starting a new command.

scala>
```

Note the indented vertical bar prompt. This means Scala detected that the command is not yet complete, and you can continue typing. Or, if you want to cancel entry of the command, hit [ENTER] twice:

Finally, try referencing an invalid variable:

```
scala> print(cd)
<console>:12: error: not found: value cd
       print(cd)
             ^

scala>
```

Note: The caret (^) indicates the point in the entered code where the error was encountered. This can help you determine which part of a complex line of code is causing the syntax error.

Arithmetic operations in Scala

```
# Addition

1+2
```

```
scala> 1+2
res0: Int = 3

scala>
```

Note: Scala has assigned a default name to the result set.

Also identified the datatype by itself

```
# Multiplication with previous result

res0*2
```

```
scala> res0*2
res1: Int = 6

scala>
```

```
# Assignment of variable

var a = 1+2
```

```
scala> var a = 1+2
a: Int = 3

scala>
```

```
# Reassign the variable a and notice the error

a = 6+"Muthu"
```

```
scala> a = 6+"Muthu"
<console>:25: error: type mismatch;
 found   : String
 required: Int
       a = 6+"Muthu"
            ^

scala>
```

Note: The error occurred because of datatype mismatch. This itself identifies that operation on a variable must be of type integer.

```
# Using val type

val msg = "Hello World"

msg = "Hello!"
```

```
scala> val msg = "Hello World"
msg: String = Hello World

scala> msg = "Hello!"
<console>:25: error: reassignment to val
       msg = "Hello!"
           ^

scala>
```

Note: Error occurred because val is immutable. So, the value cannot be reassigned

```
var msg = "Hello World"
msg = "Hello!"
```

```
scala> var msg = "Hello World"
msg: String = Hello World

scala> msg = "Hello!"
msg: String = Hello!

scala>
```

Note: No error occurred while reassignment because var is mutable.

```
var myVar = 10;
```

```
scala> var myVar = 10;
myVar: Int = 10

scala>
```

```
myVar = "Kumar"
```

```
scala> myVar = "Kumar"
<console>:25: error: type mismatch;
 found   : String("Kumar")
 required: Int
       myVar = "Kumar"
               ^

scala>
```

Note: Error occurred because of datatype mismatch. Var is assigned value of Integer at the first time it is assigned.

Expressions in Scala

In Scala, a {} block is a list of expressions, and result is also an expression. The value of the block is the value of last expression of it.

```
val x = {val a = 10; val b = 100; b-a}

val x = {val a = 10; val b = 100; b-a; b*a}
```

```
scala> val x = {val a = 10; val b = 100; b-a}
x: Int = 90

scala> val x = {val a = 10; val b = 100; b-a; b*a}
x: Int = 1000

scala>
```

Control structures in Scala

Scala: If > Else

```
var x=5
```

```
scala> var x=5
x: Int = 5

scala>
```

```
val s = if (x > 0 && x < 6) 1 else 0
```

```
scala> val s = if (x > 0 && x < 6) 1 else 0
s: Int = 1

scala>
```

```
val s = if (x > 0 && x < 6) "positive" else 0
```

```
scala> val s = if (x > 0 && x < 6) "positive" else 0
s: Any = positive

scala>
```

Scala: While Loop

```
var args = "500"
```

```
scala> var args = "500"
args: String = 500

scala>
```

```
var i = 0
```

```
scala> var i = 0
i: Int = 0

scala>
```

```
while (i < args.length) { println(args(i)); i += 1 }
```

```
scala> while (i < args.length)
     | {
     | println(args(i))
     | i += 1
     | }
5
0
0

scala>
```

Scala: Foreach loop

```
var args = "Hello"
```

```
scala> var args = "Hello"
args: String = Hello
```

```
args.foreach(arg => println(arg))
```

```
scala> args.foreach(arg => println(arg))
H
e
l
l
o
```

```
args.foreach(println)
```

```
scala> args.foreach(println)
H
e
l
l
o
```

Scala: For loop

```
val in = "Hello World"
```

```
scala> val in = "Hello World"
in: String = Hello World

scala>
```

```
var sum = 0
```

```
scala> var sum = 0
sum: Int = 0

scala>
```

```
for (i <-0 until in.length) sum += i

print(sum)
```

```
scala> for (i <- 0 until in.length) sum += i


scala> print(sum)
55
scala>
```

Scala: Advanced for Loop

```
for (i <- 1 to 3; j <-1 to 3) println(i*10 + j)
```

```
scala> print(sum)
55
scala> for (i <- 1 to 3; j <-1 to 3)
     | println(i*10 + j)
11
12
13
21
22
23
31
32
33

scala>
```

Scala: Conditional for loop

```
for (i <- 1 to 3; j <-1 to 3 if i==j) println(i*10 + j)
```

```
scala> for (i <- 1 to 3; j <-1 to 3 if i==j)
     | println(i*10 + j)
11
22
33
```

Scala: Variables in for loop

```
for (i <: 1 to 3; x = 4 - i; j<: x to 3) println(10*i + j)
```

```
scala> for (i <- 1 to 3; x = 4 - i; j<- x to 3)
     | println(10*i + j)
13
22
23
31
32
33
```

Scala: For Loop with Yield concept

```
val x = for (i <: 1 to 20) yield i*2.5

for (i <-x) println(i)
```

```
scala> val x = for (i <- 1 to 20) yield i*2.5
x: scala.collection.immutable.IndexedSeq[Double] = Vector(2.5, 5.0, 7.5, 10.0, 1
2.5, 15.0, 17.5, 20.0, 22.5, 25.0, 27.5, 30.0, 32.5, 35.0, 37.5, 40.0, 42.5, 45.
0, 47.5, 50.0)
```

```
scala> for (i <- x) println(i)
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
```

Scala: Functions

Note:

1. There is no need of return statement in scala functions
2. We need to specify the datatype for recursive function.

```
def area(radius: Double):Double = {3.14*radius*radius}
```

```
scala> def area(radius: Double):Double = {3.14*radius*radius}
area: (radius: Double)Double

scala>
```

```
area(10.09)
```

```
scala> area(10.09)
res3: Double = 319.677434
```

```
def area(radius: Int):Double = {3.14*radius*radius}

area(10)
```

```
scala> def area(radius: Int):Double = {3.14*radius*radius}
area: (radius: Int)Double

scala> area(10)
res4: Double = 314.0
```

```
def factorial(n:Int):Int = if (n==0) 1 else n*factorial(n-1)

factorial(5)
```

```
scala> def factorial(n:Int):Int = if (n==0) 1 else n*factorial(n-1)
factorial: (n: Int)Int

scala>

scala> factorial(5)
res5: Int = 120

scala>
```

Scala: Procedures

Scala has special functions which don't return any value. If there is a scala function without a preceding "=" symbol, then the return type of the function is Unit. Such functions are called procedures. Procedures do not return any value in Scala.

```
def rect_area (length: Float, breadth: Float){val area = length*breadth; print(area)}

rect_area(1,2)
```

```
scala> def rect_area (length: Float, breadth: Float){val area = length*breadth;
print(area)}
rect_area: (length: Float, breadth: Float)Unit

scala> rect_area(1,2)
2.0
scala>
```

Note: Same rules of default and named arguments apply on procedures as well

Scala: Collections

Different type of collections available in Scala are:

- Array
- ArrayBuffers
- Maps
- Tuples
- Lists

Scala Collections: Array Fixed length arrays

```
val n = new Array[Int](10)
```

```
scala> val n = new Array[Int](10)
n: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

scala>
```

```
val s = new Array[String](10)
```

```
scala> val s = new Array[String](10)
s: Array[String] = Array(null, null, null, null, null, null, null, null, null, n
ull)

scala>
```

```
val st = Array("Hello","World")
```

```
scala> val st = Array("Hello","World")
st: Array[String] = Array(Hello, World)

scala>
```

```
import Array._

var secondRange = range(10,20)
```

```
scala> import Array._
import Array._

scala> var secondRange = range(10,20)
secondRange: Array[Int] = Array(10, 11, 12, 13, 14, 15, 16, 17, 18, 19)
```

Single dimensional array

```
var array1 = Array(1.9, 2.9, 3.4, 3.5)
```

```
scala> var array1 = Array(1.9, 2.9, 3.4, 3.5)
array1: Array[Double] = Array(1.9, 2.9, 3.4, 3.5)
```

```
for (x <- array1) {

println(x)

}
```

```
scala> for (x <- array1) {
     | println(x)
     | }
1.9
2.9
3.4
3.5
```

Array Multi dimensional

```
import Array._

var arrayMultiDim = ofDim[Int](3,3)
```

```
scala> import Array._
import Array._

scala> var arrayMultiDim = ofDim[Int](3,3)
arrayMultiDim: Array[Array[Int]] = Array(Array(0, 0, 0), Array(0, 0, 0), Array(0
, 0, 0))
```

```
for (i <- 0 to 2){

for (j <- 0 to 2){

arrayMultiDim(i)(j) = j;

}

}

for (i <- 0 to 2){

for (j <- 0 to 2){
```

```
print (" " + arrayMultiDim(i)(j));

}

println();

}
```

```
scala> for (i <- 0 to 2){
     | for (j <- 0 to 2){
     | arrayMultiDim(i)(j) = j;
     | }
     | }

scala> for (i <- 0 to 2){
     | for (j <- 0 to 2){
     | print (" " + arrayMultiDim(i)(j));
     | }
     | println();
     | }
 0 1 2
 0 1 2
 0 1 2

scala>
```

```
import Array._

var firstRange = range(10, 20, 2)

var secondRange = range(10,20)
```

```
scala> import Array._
import Array._

scala> var firstRange = range(10, 20, 2)
firstRange: Array[Int] = Array(10, 12, 14, 16, 18)

scala> var secondRange = range(10,20)
secondRange: Array[Int] = Array(10, 11, 12, 13, 14, 15, 16, 17, 18, 19)

scala>
```

```
for (x <- firstRange) {

print(" " + x)

}

for (x <- secondRange) {
```

```
print(" " + x)

}
```

```
scala> for (x <- firstRange) {
     | print(" " + x)
     | }
 10 12 14 16 18
scala>
```

```
scala> for (x <- secondRange) {
     | print(" " + x)
     | }
 10 11 12 13 14 15 16 17 18 19
scala>
```

Scala Collections: ArrayBuffer

Variable length arrays(Array Buffers)is similar to java ArrayLists

```
import scala.collection.mutable.ArrayBuffer

val a = ArrayBuffer[Int]()
```

```
scala> import scala.collection.mutable.ArrayBuffer
import scala.collection.mutable.ArrayBuffer

scala> val a = ArrayBuffer[Int]()
a: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer()

scala>
```

```
a += 1
```

```
scala> a += 1
res9: a.type = ArrayBuffer(1)

scala>
```

```
a += (2,3,5)
```

```
scala> a += (2,3,5)
res10: a.type = ArrayBuffer(1, 2, 3, 5)

scala>
```

```
a++=Array(6,7,8)
```

```
scala> a++=Array(6,7,8)
res11: a.type = ArrayBuffer(1, 2, 3, 5, 6, 7, 8)

scala>
```

```
for (el <- a) println(el)
```

```
scala> for (el <- a) println(el)
1
2
3
5
6
7
8
```

Transformation

```
var yieldedVal = for (el<: a if el % 2==0) yield (2*el)

println("Yielded Value " + yieldedVal.mkString(" ** "))
```

```
scala> var yieldedVal = for (el<- a if el % 2==0) yield (2*el)
yieldedVal: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(4, 12, 16)


scala> println("Yielded Value " + yieldedVal.mkString(" ** "))
Yielded Value 4 ** 12 ** 16
```

Note: mkString will bring delimeter.

Sorting

```
val mySortTest = Array(1,7,2,9)
```

```
scala> val mySortTest = Array(1,7,2,9)
mySortTest: Array[Int] = Array(1, 7, 2, 9)
```

```
println("Actual Value " + mySortTest.mkString(" ** "))
```

```
scala> println("Actual Value " + mySortTest.mkString(" ** "))
Actual Value 1 ** 7 ** 2 ** 9
```

```
scala.util.Sorting.quickSort(mySortTest)
```

```
scala> scala.util.Sorting.quickSort(mySortTest)
```

```
println("Sorted value " + mySortTest.mkString(" ** "))
```

```
scala> println("Sorted value " + mySortTest.mkString(" ** "))
Sorted value 1 ** 2 ** 7 ** 9
```

Scala Collections: Arrays and ArrayBuffers

Common Operations:

```
a.trimEnd(2) //remove last 2 elements

a.insert(2,9) //Adds element at 2nd index

a.insert(2,10,11,12) //Adds a list

a.remove(2) //Remove an element

a.remove(2,3) //Removes 3 elements from index 2

Array(1,2,3,4).sum

Array(1,5,9,8).max
```

```
scala> a.trimEnd(2)

scala> a.insert(2,9)

scala> a.insert(2,10,11,12)

scala> a.remove(2)
res20: Int = 10

scala> a.remove(2,3)

scala> Array(1,2,3,4).sum
res22: Int = 10

scala> Array(1,5,9,8).max
res23: Int = 9
```

Scala Collections: Maps

What is key value pair

Name: Muthu Age: 30 Key :: Value

In Scala, a map is a collection of pair. A pair is a group of two values (Not necessarily of same type)

```
val mapping = Map ("Vishal" -> "Kumar", "Vijay" -> "Verma")
```

```
scala> val mapping = Map ("Vishal" -> "Kumar", "Vijay" -> "Verma")
mapping: scala.collection.immutable.Map[String,String] = Map(Vishal -> Kumar, Vi
jay -> Verma)
```

Accessing Maps

```
val mapping = scala.collection.mutable.Map("Vishal" -> "K",
"Vijay" -> "V")
```

```
scala> val mapping = scala.collection.mutable.Map("Vishal" -> "K", "Vijay" -> "V")
mapping: scala.collection.mutable.Map[String,String] = Map(Vishal -> K, Vijay -> V)

scala>
```

```
val x = mapping("Vishal")
```

```
scala> val x = mapping("Vishal")
x: String = K
```

```
val x = mapping.getOrElse("Vish", 0)
```

```
scala> val x = mapping.getOrElse("Vish", 0)
x: Any = 0
```

```
mapping -= "Vishal"
```

```
scala> mapping -= "Vishal"
res25: mapping.type = Map(Vijay -> V)
```

```
mapping += ("Ajay" :> "Sharma")
```

```
scala> mapping += ("Ajay" -> "Sharma")
res27: mapping.type = Map(Ajay -> Sharma, Vijay -> V)
```

Trying to give duplicate keys in Map

Note: Mapping took only 2 values and not three

Trying to modify an immutable map

Printing

```
val mapping1 = Map ("Vishal" -> "Kumar", "Vijay" -> "Verma",
"Vijay" -> "Verma1")
```

```
scala> val mapping1 = Map ("Vishal" -> "Kumar", "Vijay" -> "Verma", "Vijay" -> "Verma1")
mapping1: scala.collection.immutable.Map[String,String] = Map(Vishal -> Kumar, Vijay -> Ver
ma1)
```

```
mapping1 += ("Muthu" -> "Kumar")
```

```
scala> mapping1 += ("Muthu" -> "Kumar")
<console>:30: error: value += is not a member of scala.collection.immutable.Map[String,Stri
ng]
       mapping1 += ("Muthu" -> "Kumar")
                ^
```

```
println("Printing keys of mapping1: " + mapping1.keys)
```

```
scala> println("Printing keys of mapping1: " + mapping1.keys)
Printing keys of mapping1: Set(Vishal, Vijay)
```

```
println("Printing values of mapping1: " + mapping1.values)
```

```
scala> println("Printing values of mapping1: " + mapping1.values)
Printing values of mapping1: MapLike(Kumar, Verma1)
```

Iterating Maps

```
val mapping3 = for ((k,v) <- mapping) yield (v,k)
```

```
scala> val mapping3 = for ((k,v) <- mapping) yield (v,k)
mapping3: scala.collection.mutable.Map[String,String] = Map(V -> Vijay, Sharma -> Ajay)
```

```
println("printing keys of mapping3: " + mapping3.keys)
```

```
scala> println("printing keys of mapping3: " + mapping3.keys)
printing keys of mapping3: Set(V, Sharma)
```

```
println("printing keys of mapping3: " + mapping3)
```

```
scala> println("printing keys of mapping3: " + mapping3)
printing keys of mapping3: Map(V -> Vijay, Sharma -> Ajay)
```

```
println("getting a value of Vishal from mapping1: " +
mapping1.getOrElse("Vishal",0))
```

```
scala> println("getting a value of Vishal from mapping1: " + mapping1.getOrElse("Vishal",0)
)
getting a value of Vishal from mapping1: Kumar
```

```
println("getting a value of Viss from mapping1: " +
mapping1.getOrElse("Viss","Not Found"))
```

```
scala> println("getting a value of Viss from mapping1: " + mapping1.getOrElse("Viss","Not F
ound"))
getting a value of Viss from mapping1: Not Found
```

Scala Collections: Tuples

Tuple is more generalized form of pair Tuple has more than two values of potentially different types.

Accessing the tuple elements

```
val a = (1,4, "Bob", "Jack")
```

In tuples, offset starts with 1 and not 0. Tuples are typically used for functions which return more than one value.

```
scala> val a = (1,4, "Bob", "Jack")
a: (Int, Int, String, String) = (1,4,Bob,Jack)
```

```
a._2 or a _2 //returns 4
```

```
scala> a._2
res38: Int = 4
```

```
val t = (11,22,33,44)
```

```
scala> val t = (11,22,33,44)
t: (Int, Int, Int, Int) = (11,22,33,44)
```

```
val sum = t._1+t._2+t._3+t._4
```

```
scala> val sum = t._1+t._2+t._3+t._4
sum: Int = 110
```

```
println ("Sum of elements: " + sum)
```

```
scala> println ("Sum of elements: " + sum)
Sum of elements: 110
```

Scala Collections: Lists

List is either Nil or a combination of head and tail elements where tail is again a list

Eg:

```
val lst = List(1,2)
```

```
scala> val lst = List(1,2)
lst: List[Int] = List(1, 2)
```

```
lst.head
```

```
scala> lst.head
res40: Int = 1
```

```
lst.tail
```

```
scala> lst.tail
res41: List[Int] = List(2)
```

:: operator adds a new list from given head and tail

```
2 :: List(4,5)
```

```
scala> 2 :: List(4,5)
res43: List[Int] = List(2, 4, 5)
```

Note: List is comparable to ArrayBuffer. ArrayBuffer is used where suppose a log file is read continuously and error is getting fetch and added. Whereas List should be used where it is more read intensive. We can use iterator to iterate over a list, but recursion is a preferred practice in scala

Use Case: Loudacre

Introducing Loudacre

Loudacre is a (fictional) mobile phone carrier. In many of the exercises, you will use Loudacre device status data from mobile phones. Whenever a phone does soft or hard restart, it gathers error information and sends it back to Loudacre's central facility where the data is collected for analysis.

Start by examining the log file at /home/cloudera/dataset/loudacre.log using an editor of your choice. This is a typical line of data from that file:

```
2014-03-15:10:10:31,Titanic 4000,
1882b564_c7e0_4315_aa24_228c0155ee1b,58,36,39,31,15
0,TRUE,enabled,enabled,37.819722,-122.478611
```

Here are the data fields shown in that line:

Field	Sample Data
Date and time	2014-03-15:10:10:31
Model name and number	Titanic 4000
Unique device ID	1882b564-c7e0-4315-aa24-228c0155ee1b
Device temperature (Celsius)	58
Ambient temperature (Celsius)	36
Battery available (percent)	39
Signal strength (percent)	31
CPU utilization (percent)	15
RAM memory usage (percent)	0
GPS Status (enabled=TRUE, disabled=FALSE)	TRUE

Field	Sample Data
Bluetooth status (enabled/ disabled/connected)	enabled
WiFi status (enabled/ disabled/connected)	enabled
Latitude	37.819722
Longitude	-122.478611

- Reading and Displaying a Data File

Enter the following code in the Scala shell to print out each line of the loudacre.log file.

File location: /home/cloudera/dataset/

File Name: loudacre.log

```
[technocrafty@technocrafty dataset]$ pwd
/home/technocrafty/dataset
[technocrafty@technocrafty dataset]$ ls -ltr *.log*
-rw------- 1 technocrafty technocrafty 67077 Nov 19  2015 loudacre.log
[technocrafty@technocrafty dataset]$
```

```
scala> import scala.io.Source
```

```
scala>
Source.fromFile("/home/cloudera/dataset/loudacre.log").foreach
(print)
```

```
scala> import scala.io.Source
import scala.io.Source

scala> Source.fromFile("/home/cloudera/dataset/loudacre.log").foreach(print)
2014-03-15:10:10:20,iFruit 1,6474caf1-7bbf-4594-a526-9ba8ea82e151,0,15,71,77,0,4
0,TRUE,enabled,connected,37.90310537,-121.5614513
2014-03-15:10:10:20,iFruit 2,5c2d40d8-b1e0-4c2a-b050-06ca6c590741,1,28,66,67,40,
49,TRUE,disabled,connected,34.12789546,-108.9681595
2014-03-15:10:10:20,iFruit 3,27178d24-3a61-42f7-a784-e3263f25cc6f,1,30,91,89,41,
17,TRUE,enabled,enabled,37.92489617,-122.2068682
2014-03-15:10:10:20,iFruit 3A,fe4674a7-0632-494a-aaf3-f5865a724e1c,0,21,95,73,19
,62,TRUE,disabled,enabled,38.0180756,-120.0678602
2014-03-15:10:10:20,iFruit 5,86e4bd60-5c72-4558-a019-caf1de9d14f1,0,29,68,32,63,
62,TRUE,enabled,connected,39.47088617,-119.6599261
2014-03-15:10:10:20,MeeToo 1.0,ef8c7564-0a1a-4650-a655-c8bbd5f8f943,0,31,63,70,3
9,27,TRUE,enabled,enabled,37.43210889,-121.4850296
```

Scala Variables and Typing

Exploring Mutability, Reassignment, and Redefinition of Variables

- Create an immutable variable and reassign a value to it. What happens?

```
scala> val ambientTemp = 19

scala> ambientTemp = 47
```

```
scala> val ambientTemp = 19
ambientTemp: Int = 19

scala> ambientTemp = 42
<console>:13: error: reassignment to val
       ambientTemp = 42
                   ^

scala>
```

Reassignment is not permitted because ambientTemp is defined as an immutable variable.

- Redefine the variable and assign a new value having an inferred type of Double:

```
scala> val ambientTemp = 27.0
```

```
scala>  val ambientTemp = 27.0
ambientTemp: Double = 27.0

scala>
```

- Create a mutable variable with an inferred type of Int. What happens when you assign a value of a different type, for example, a Double?

  What happens when you try to change the value to a new value of the same type?

  Answer: it fails with a type mismatch error.

```
scala> var deviceTemp = 21

scala> deviceTemp = ambientTemp

scala> deviceTemp = 45
```

```
scala> var deviceTemp = 21
deviceTemp: Int = 21

scala> deviceTemp = ambientTemp
<console>:14: error: type mismatch;
 found   : Double
 required: Int
       deviceTemp = ambientTemp
                    ^

scala> deviceTemp = 45
deviceTemp: Int = 45

scala>
```

- It is possible to redefine the mutable variable with a different inferred type:

```
scala> var deviceTemp = 33.1
```

```
scala>  var deviceTemp = 33.1
deviceTemp: Double = 33.1

scala>
```

- You can define variables of explicit types:

```
scala> val phoneModel: String = "Sorrento"

scala> val temp: Int = 30

scala> val location: Double = 33.1765
```

```
scala> val phoneModel: String = "Sorrento"
phoneModel: String = Sorrento

scala> val temp: Int = 30
temp: Int = 30

scala> val location: Double = 33.1765
location: Double = 33.1765

scala>
```

Working with Numeric Variables

In this exercise, you will use the Scala shell to explore integer and floating-point types by converting Celsius temperatures to Fahrenheit.

```
scala> val tempC = 27

scala> val c_to_f = 9/5

scala> val tempF = tempC * c_to_f + 32
```

```
scala>  val tempC = 27
tempC: Int = 27

scala> val c_to_f = 9/5
c_to_f: Int = 1

scala> val tempF = tempC * c_to_f + 32
tempF: Int = 59

scala>
```

Note: The variables and final results are all integers. c_to_f has the value 1, causing the conversion to be incorrect with tempF = 59 degrees Fahrenheit.

- Repeat the calculation using floating point numbers to correctly convert 27 degrees Celsius to 80.6 degrees Fahrenheit.

```
scala> val c_to_f = 9.0/5.0

scala> val tempF = tempC * c_to_f + 32
```

```
scala> val c_to_f = 9.0/5.0
c_to_f: Double = 1.8

scala> val tempF = tempC * c_to_f + 32
tempF: Double = 80.6

scala>
```

- You can find the type of the resulting object using getClass:

```
scala> println(tempC.getClass, tempF.getClass)
```

```
scala> println(tempC.getClass, tempF.getClass)
(int,double)

scala>
```

Using Scala Operators

Loudacre reports CPU utilization in percentages. Use Scala interactively to explore Scala's operators

- Calculate the average of two CPU utilization percentages:

```
scala> val cpuT1 = 17

scala> val cpuT2 = 38

scala> val averageCPU = (cpuT1 + cpuT2) / 2
```

```
scala> val cpuT1 = 17
cpuT1: Int = 17

scala> val cpuT2 = 38
cpuT2: Int = 38

scala> val averageCPU = (cpuT1 + cpuT2) / 2
averageCPU: Int = 27

scala>
```

- The calculation returned an integer result of 27. Try assigning the variable type explicitly and note the implicit conversion:

```
scala> val averageCPU: Double = (cpuT1 + cpuT2) / 2
```

```
scala> val averageCPU: Double = (cpuT1 + cpuT2) / 2
averageCPU: Double = 27.0

scala>
```

- Although the result type above has been converted to a Double, 27.0, the answer is not quite right - it should be 27.5. The result of dividing two integers will be an integer. Had either the dividend or the divisor been a Double, floating point division would have been performed, yielding 27.5 as the result. Try it:

```
scala> val averageCPU: Double = (cpuT1 + cpuT2) / 2.0
```

```
scala>  val averageCPU: Double = (cpuT1 + cpuT2) / 2.0
averageCPU: Double = 27.5

scala>
```

- Print the result of the calculation. Note the implicit conversion of Double to String using the String + concatenation operator.

```
scala> println("Average CPU: " + averageCPU)
```

```
scala> println("Average CPU: " + averageCPU)
Average CPU: 27.5

scala>
```

- New and improved phone software can reduce CPU utilization by 12%. Using in-line operations, display the expected new CPU results after the improvement has been implemented.

```
scala> val reduction = 12/100.0

scala> println("Expected CPU utilization: " + (cpuT2-(cpuT2 *
reduction) + "%"))
```

```
scala> val reduction = 12/100.0
reduction: Double = 0.12

scala> println("Expected CPU utilization: " + (cpuT2-(cpuT2 * reduction) + "%"))

Expected CPU utilization: 33.44%

scala>
```

- Increment CPU utilization at time 1 by 1% using the increment operator +=. What happens?

```
scala> cpuT1 += 1
```

```
scala> cpuT1 += 1
<console>:14: error: value += is not a member of Int
  Expression does not convert to assignment because receiver is not assignable.
       cpuT1 += 1
             ^

scala>
```

- How would you fix this problem?

  Answer: use a var instead of a val to define cpuT1.

```
scala> var cpuT1 =17

scala> cpuT1 += 1
```

```
scala> var cpuT1 = 17
cpuT1: Int = 17

scala> cpuT1 += 1

scala> println(cpuT1)
18

scala>
```

- Explore casting between different variable types using conversion methods.

```
scala> val cpuT1: Int = 35

scala> val cpuT2: Double = 37.23

scala> cpuT1.toDouble

scala> cpuT2.toInt

scala> cpuT2.toString

scala> "cpuT1: " + cpuT1 + "   cpuT2: " + cpuT2
```

```
scala> val cpuT1: Int = 35
cpuT1: Int = 35

scala> val cpuT2: Double = 37.23
cpuT2: Double = 37.23

scala> cpuT1.toDouble
res11: Double = 35.0

scala> cpuT2.toInt
res12: Int = 37

scala> cpuT2.toString
res13: String = 37.23

scala> "cpuT1: " + cpuT1 + "   cpuT2: " + cpuT2
res14: String = cpuT1: 35   cpuT2: 37.23

scala>
```

- What other methods are available on the Int type? Enter cpuT1, followed by a dot, and then press [TAB] to see a list.

```
scala> cpuT1.[TAB]
```

```
scala> cpuT1.
!=   >            floatValue     isValidInt     to              toRadians
%    >=           floor          isValidLong    toBinaryString  toShort
&    >>           getClass       isValidShort   toByte          unary_+
*    >>>          intValue       isWhole        toChar          unary_-
+    ^            isInfinite     longValue      toDegrees       unary_~
-    abs          isInfinity     max            toDouble        underlying
/    byteValue    isNaN          min            toFloat         until
<    ceil         isNegInfinity  round          toHexString     |
<<   compare      isPosInfinity  self           toInt
<=   compareTo    isValidByte    shortValue     toLong
==   doubleValue  isValidChar    signum         toOctalString

scala> cpuT1.
```

Importing and Using Mathematical Functions

- Math functions in Scala are provided through a built-in library called math. Try using the sqrt function.

```
scala> math.sqrt(192)
```

```
scala> math.sqrt(192)
res16: Double = 13.856406460551018

scala>
```

- If you plan to call the sqrt function often, it is useful to import the function to enable you to invoke it more simply as sqrt:

```
scala> import scala.math.sqrt

scala> sqrt(192)
```

```
scala> import scala.math.sqrt
import scala.math.sqrt

scala> sqrt(192)
res17: Double = 13.856406460551018

scala>
```

Working with Strings

- Use the following single string to represent one record in the loudacre.log file.

```
scala> val record = "2014-03-15:10:10:31, Titanic 4000,
1882b564-c7e0-4315-aa24-228c0155ee1b, 58, 36, 39, 31, 15, 0,
TRUE, enabled, enabled, 37.819722,-122.478611"
```

```
scala> val record = "2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-
228c0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, enabled, enabled, 37.819722,-122.4786
11"
record: String = 2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-228c
0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, enabled, enabled, 37.819722,-122.478611

scala>
```

- Print out the record (println(record)) and the length of the record (record.length). The record should be 141 characters long.

```
scala> println(record)

scala> record.length
```

```
scala> println(record)
2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-228c0155ee1b, 58, 36,
 39, 31, 15, 0, TRUE, enabled, enabled, 37.819722,-122.478611

scala> record.length
res19: Int = 141

scala>
```

- Use the contains method to determine whether the model name Titanic is in the record.

```
scala> record.contains(“Titanic”)
```

```
scala> record.contains("Titanic")
res20: Boolean = true

scala>
```

- Use indexOf to locate the index of the first character of Titanic. It’s found at position 21.

```
scala> val startPos = record.indexOf(“T”)
```

```
scala> val startPos = record.indexOf("T")
startPos: Int = 21

scala>
```

- Use the substring method to output the model name in upper case using the toUpperCase method. Use function

calls to compute the starting and ending indexes rather than hardcoding the values.

```
scala> record.substring(startPos, startPos +
"Titanic".length).toUpperCase
```

```
scala> record.substring(startPos, startPos + "Titanic".length).toUpperCase
res21: String = TITANIC

scala>
```

- Get the four-digit model number that follows the model name and remember to skip the blank that separates the model name from the model number. The model number is 4000.

```
scala> val modelNum = record.substring(startPos +
"Titanic".length + 1, startPos + "Titanic".length + 5)
```

```
scala> val modelNum = record.substring(startPos + "Titanic".length + 1, startPos
 + "Titanic".length + 5)
modelNum: String = 4000

scala>
```

Booleans

In this exercise you will extract the Bluetooth and WiFi fields from the record string and use boolean operators to test them.

The Bluetooth and WiFi status is in the fourth and third elements from the end of the record, respectively. Each field can contain one of three values: enabled, disabled, or connected.

- Create a string as follows

```
scala> val record2 = "2014-03-15:10:10:31, Titanic 4000,
1882b564-c7e0-4315-aa24-228c0155ee1b, 58, 36, 39, 31, 15, 0,
TRUE, disabled, enabled, 37.819722,-122.478611"
```

```
scala> val record2 = "2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24
-228c0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, disabled, enabled, 37.819722,-122.47
8611"
record2: String = 2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-228
c0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, disabled, enabled, 37.819722,-122.478611

scala>
```

- You can use contains to test whether a string is contained within another string

```
scala> record2.contains("connected")

scala> record2.contains("disabled")
```

```
scala> record2.contains("connected")
res22: Boolean = false

scala> record2.contains("disabled")
res23: Boolean = true

scala>
```

- You can use indexOf to find the start position of a string within another string. Find the start of the Bluetooth field, currently set to disabled. It starts at position 102.

```
scala> record2.indexOf("disabled")
```

```
scala> record2.indexOf("disabled")
res24: Int = 102

scala>
```

- Use substring together with indexOf to extract the Bluetooth state and store it in val bluetooth.

```
scala> val bluetooth =
record2.substring(record2.indexOf("disabled"),
record2.indexOf("disabled") + "disabled".length)
```

```
scala> val bluetooth = record2.substring(record2.indexOf("disabled"), record2.in
dexOf("disabled") + "disabled".length)
bluetooth: String = disabled

scala>
```

- Perform similar steps to find the WiFi state and store it in val wifi.

```
scala> record2.indexOfSlice("enabled")
```

```
scala> record2.indexOfSlice("enabled")
res25: Int = 112

scala>
```

Exploring Tuples, Lists, and Maps

In this exercise, you will practice working with various types of collections and tuples.

Working with Tuples

- Create the following tuple in the Scala shell. Verify its type with getClass

```
scala> val phoneTuple = ("Titanic", "3000", "disabled",
"connected")

scala> phoneTuple.getClass
```

```
scala> val phoneTuple = ("Titanic", "3000", "disabled", "connected")
phoneTuple: (String, String, String, String) = (Titanic,3000,disabled,connected)

scala>
```

- You can access elements of a tuple using the _n selector. Display the first and third elements of phoneTuple.

```
scala> phoneTuple._1

scala> phoneTuple._3
```

```
scala> phoneTuple._1
res30: String = Titanic

scala> phoneTuple._3
res31: String = disabled

scala>
```

- You can also access elements of the tuple using phoneTuple.productElement(n), but keep in mind that n is zero-based when using this technique. Display the first and third elements of phoneTuple using this technique.

```
scala> phoneTuple.productElement(0)

scala> phoneTuple.productElement(2)
```

```
scala> phoneTuple.productElement(0)
res32: Any = Titanic

scala> phoneTuple.productElement(2)
res33: Any = disabled

scala>
```

Check the arity of the tuple.

```
scala> phoneTuple.productArity
```

```
scala> phoneTuple.productArity
res34: Int = 4

scala>
```

Working with Lists

- Start by converting the phoneList tuple to a list:

```
scala> val phoneList = phoneTuple.productIterator.toList
```

```
scala> val phoneList = phoneTuple.productIterator.toList
phoneList: List[Any] = List(Titanic, 3000, disabled, connected)

scala>
```

- Using the [TAB] key, determine which type – the tuple or the list – has more methods?

```
scala> phoneTuple.[TAB]

scala> phoneList.[TAB]
```

```
scala> phoneTuple.
_1   _3   canEqual   equals     productArity     productIterator   toString
_2   _4   copy       hashCode   productElement   productPrefix

scala> phoneTuple.
```

```
scala> phoneList.
++              flatMap             min                 sortBy
++:             flatten             minBy               sortWith
+:              fold                mkString            sorted
/:              foldLeft            nonEmpty            span
:+              foldRight           orElse              splitAt
::              forall              padTo               startsWith
:::             foreach             par                 stringPrefix
:\              genericBuilder      partition           sum
WithFilter      groupBy             patch               tail
addString       grouped             permutations        tails
aggregate       hasDefiniteSize     prefixLength        take
andThen         hashCode            product             takeRight
apply           head                productArity        takeWhile
applyOrElse     headOption          productElement      to
canEqual        indexOf             productIterator     toArray
collect         indexOfSlice        productPrefix       toBuffer
collectFirst    indexWhere          reduce              toIndexedSeq
combinations    indices             reduceLeft          toIterable
companion       init                reduceLeftOption    toIterator
compose         inits               reduceOption        toList
contains        intersect           reduceRight         toMap
```

There are many more methods available for processing lists than there are for processing tuples. Let's look at some important ones.

- Let's convert our list to a comma-separated list of values, which we could then import into a spreadsheet. You can use the map method to apply an operation to each member of the list.

```
scala> val csvStr1 = phoneList.map("\"" + _ + "\"").toString
scala> println(csvStr1)
```

```
scala> val csvStr1 = phoneList.map("\"" + _ + "\"").toString
csvStr1: String = List("Titanic", "3000", "disabled", "connected")

scala> println(csvStr1)
List("Titanic", "3000", "disabled", "connected")

scala>
```

The above is close to what we want; however, did you notice that the println shows the collection type, List?

- You can use mkString to convert the list to a string and leave off the collection type.

```
scala> val csvStr2 = phoneList.map("\"" + _ + "\"").mkString

scala> println(csvStr2)
```

```
scala> val csvStr2 = phoneList.map("\"" + _ + "\"").mkString
csvStr2: String = "Titanic""3000""disabled""connected"

scala> println(csvStr2)
"Titanic""3000""disabled""connected"

scala>
```

- This is closer yet, but the values run together. Provide a comma separator as an argument to mkString to separate each value by a comma.

```
scala> val csvStr3 = phoneList.map("\"" + _ +
"\"").mkString(", ") scala> println(csvStr3)
```

```
scala> val csvStr3 = phoneList.map("\"" + _ + "\"").mkString(", ")
csvStr3: String = "Titanic", "3000", "disabled", "connected"

scala> println(csvStr3)
"Titanic", "3000", "disabled", "connected"

scala>
```

Iterating through Data Efficiently

In this exercise, you will explore flow control and program structure in Scala.

Comparing Approaches to Iteration

In this section, you will process a collection of floating point numbers. The numbers represent phone charge percentages, and you will convert them to milliamp-hours (mAh).

You will repeat the same task using different approaches inorder to compare them.

- Start by defining a function to convert percentage to mAh:

```
scala> def mAh(percent: Double) = (percent/100) * 5400
```

```
scala> def mAh(percent: Double) = (percent/100) * 5400
mAh: (percent: Double)Double

scala>
```

- Define a list of phone battery charge percentages:

```
scala> val phoneBattery = List(82.3, 31.6, 72.5, 64.7)
```

```
scala> val phoneBattery = List(82.3, 31.6, 72.5, 64.7)
phoneBattery: List[Double] = List(82.3, 31.6, 72.5, 64.7)

scala>
```

- Classic iterative approach: use a for loop with a range to iterate over the list and calculate and print the results.

```
scala> for (i <- 0 until phoneBattery.length)
println(phoneBattery(i) + "% = " + mAh(phoneBattery(i)) + "
mAh")
```

```
scala> for (i <- 0 until phoneBattery.length) println(phoneBattery(i) + "% = " +
 mAh(phoneBattery(i)) + " mAh")
82.3% = 4444.2 mAh
31.6% = 1706.4 mAh
72.5% = 3915.0 mAh
64.7% = 3493.8 mAh

scala>
```

- Iteration with a generator: instead of using a counting variable with a range, use the List with the for generator.

```
scala> for (percent <- phoneBattery) println(percent + "% = "
+ mAh(percent) + " mAh")
```

```
scala> for (percent <- phoneBattery) println(percent + "% = " + mAh(percent) + "
 mAh")
82.3% = 4444.2 mAh
31.6% = 1706.4 mAh
72.5% = 3915.0 mAh
64.7% = 3493.8 mAh

scala>
```

- List comprehension: instead of having the for loop print out each value, have it return a new List containing the converted values by using for and yield.

```
scala> val myList = for (percent <- phoneBattery) yield
mAh(percent)
```

```
scala> val myList = for (percent <- phoneBattery) yield mAh(percent)
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- Now you can loop through the resulting list to display the values any number of ways, such as:

```
scala> myList.foreach(println)
```

```
scala>  myList.foreach(println)
4444.2
1706.4
3915.0
3493.8

scala>
```

- The map method can be used as a convenience method for for and yield:

```
scala> val myList = phoneBattery.map(percent => mAh(percent))
```

```
scala> val myList = phoneBattery.map(percent => mAh(percent))
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- Try the same code as in the preceding step, substituting a different name for percent. This works because the parameter name need not match the parameter name given when mAh was defined.

```
scala> val myList = phoneBattery.map(anyName => mAh(anyName))
```

```
scala> val myList = phoneBattery.map(anyName => mAh(anyName))
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- Try using a placeholder parameter in the above map call as a shortcut:

```
scala> val myList = phoneBattery.map(mAh(_))
```

```
scala> val myList = phoneBattery.map(mAh(_))
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- Or as an even more succinct shortcut:

```
scala> val myList = phoneBattery.map(mAh)
```

```
scala> val myList = phoneBattery.map(mAh)
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- Anonymous function: another option, rather than defining and using the named function mAh, is to pass the conversion function as an anonymous function.

```
scala> val myList = phoneBattery.map(percent => percent  / 100
* 5400)
```

```
scala> val myList = phoneBattery.map(percent => percent  / 100 * 5400)
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

- You can also use a placeholder parameter in an anonymous function:

```
scala> val myList = phoneBattery.map(_ / 100 * 5400)
```

```
scala> val myList = phoneBattery.map(_ / 100 * 5400)
myList: List[Double] = List(4444.2, 1706.4, 3915.0, 3493.8)

scala>
```

Using Iterators

In this section, you will explore a List and an Iterator in the Scala shell.

- Create a list called myBrands with the following values: iFruit, MeToo, Titanic, Ronin, Sorrento:

```
scala> val myBrands =
List("iFruit","MeToo","Titanic","Ronin","Sorrento")
```

```
scala> val myBrands = List("iFruit","MeToo","Titanic","Ronin","Sorrento")
myBrands: List[String] = List(iFruit, MeToo, Titanic, Ronin, Sorrento)

scala>
```

- Now create an iterator myIt that refers to myBrands. Does it have items left in the iterator? Yes, it does.

```
scala> var myIt = myBrands.toIterator

scala> myIt.hasNext
```

```
scala> var myIt = myBrands.toIterator
myIt: Iterator[String] = non-empty iterator

scala> myIt.hasNext
res41: Boolean = true

scala>
```

- Use the following code to step through the iterator twice. What do you think will happen? Will the second time work the same as the first? No, it does not.

```
scala> while(myIt.hasNext) println(myIt.next)

scala> while(myIt.hasNext) println(myIt.next)
```

```
scala> while(myIt.hasNext) println(myIt.next)
iFruit
MeToo
Titanic
Ronin
Sorrento

scala>
```

```
scala> while(myIt.hasNext) println(myIt.next)

scala>
```

- Create a new iterator then check how many items are in the iterator using size. Do you predict this method will consume the iterator or not? Check by calling hasNext. It returns false.

```
scala> myIt = myBrands.toIterator

scala> myIt.size scala> myIt.hasNext
```

```
scala> myIt = myBrands.toIterator
myIt: Iterator[String] = non-empty iterator

scala> myIt.size
res44: Int = 5

scala> myIt.hasNext
res45: Boolean = false

scala>
```

Using for with a File Source Iterator

- In an earlier exercise, you saw how to read and display the contents of a file using code like this:

```
scala> import scala.io.Source

scala> val fname = "/home/cloudera/dataset/loudacre.log"

scala> var fsource = Source.fromFile(fname)
```

What type is fsource? It reports that fsource is a nonempty iterator.

```
scala> import scala.io.Source
import scala.io.Source

scala> val fname = "/home/cloudera/dataset/loudacre.log"
fname: String = /home/cloudera/dataset/loudacre.log

scala> var fsource = Source.fromFile(fname)
fsource: scala.io.BufferedSource = non-empty iterator

scala>
```

- Use a for loop to display the first 500 characters of the file:

```
scala> for (x <- 1 to 500) print(fsource.next)
```

```
scala> for (x <- 1 to 500) print(fsource.next)
2014-03-15:10:10:20,iFruit 1,6474caf1-7bbf-4594-a526-9ba8ea82e151,0,15,71,77,0,4
0,TRUE,enabled,connected,37.90310537,-121.5614513
2014-03-15:10:10:20,iFruit 2,5c2d40d8-b1e0-4c2a-b050-06ca6c590741,1,28,66,67,40,
49,TRUE,disabled,connected,34.12789546,-108.9681595
2014-03-15:10:10:20,iFruit 3,27178d24-3a61-42f7-a784-e3263f25cc6f,1,30,91,89,41,
17,TRUE,enabled,enabled,37.92489617,-122.2068682
2014-03-15:10:10:20,iFruit 3A,fe4674a7-0632-494a-aaf3-f5865a724e1c,0,21,95,73,19
,62,TRUE,disabled,enabled,38.
scala>
```

- Try repeating the 500 count for loop again. Is the output the same as the first time? No, it is not.

  fsource is a character-based iterator over the file. Subsequent calls read from the last cursor location

```
scala> for (x <- 1 to 500) print(fsource.next)
```

```
scala> for (x <- 1 to 500) print(fsource.next)
0180756,-120.0678602
2014-03-15:10:10:20,iFruit 5,86e4bd60-5c72-4558-a019-caf1de9d14f1,0,29,68,32,63,
62,TRUE,enabled,connected,39.47088617,-119.6599261
2014-03-15:10:10:20,MeeToo 1.0,ef8c7564-0a1a-4650-a655-c8bbd5f8f943,0,31,63,70,3
9,27,TRUE,enabled,enabled,37.43210889,-121.4850296
2014-03-15:10:10:20,MeeToo 1.0,23eba027-b95a-4729-9a4b-a3cca51c5548,0,20,21,86,5
4,34,TRUE,enabled,enabled,39.43789083,-120.9389785
2014-03-15:10:10:20,MeeToo 1.0,16085fbf-cda5-4489-84b9-0fad888f9e7a,0,29,26,97,8
,11,TR
```

- Use the getLines method to return a line-based iterator:

```
scala> var flines = Source.fromFile(fname).getLines
```

```
scala> var flines = Source.fromFile(fname).getLines
flines: Iterator[String] = non-empty iterator

scala>
```

- Print out the first line of the file:

```
scala> print(flines.next)
```

```
scala> print(flines.next)
2014-03-15:10:10:20,iFruit 1,6474caf1-7bbf-4594-a526-9ba8ea82e151,0,15,71,77,0,4
0,TRUE,enabled,connected,37.90310537,-121.5614513
scala>
```

- Use flines to display the remaining lines in the file.

```
scala> flines.foreach(println)
```

```
scala> flines.foreach(println)
2014-03-15:10:10:20,iFruit 2,5c2d40d8-b1e0-4c2a-b050-06ca6c590741,1,28,66,67,40,
49,TRUE,disabled,connected,34.12789546,-108.9681595
2014-03-15:10:10:20,iFruit 3,27178d24-3a61-42f7-a784-e3263f25cc6f,1,30,91,89,41,
17,TRUE,enabled,enabled,37.92489617,-122.2068682
2014-03-15:10:10:20,iFruit 3A,fe4674a7-0632-494a-aaf3-f5865a724e1c,0,21,95,73,19
,62,TRUE,disabled,enabled,38.0180756,-120.0678602
2014-03-15:10:10:20,iFruit 5,86e4bd60-5c72-4558-a019-caf1de9d14f1,0,29,68,32,63,
62,TRUE,enabled,connected,39.47088617,-119.6599261
2014-03-15:10:10:20,MeeToo 1.0,ef8c7564-0a1a-4650-a655-c8bbd5f8f943,0,31,63,70,3
9,27,TRUE,enabled,enabled,37.43210889,-121.4850296
2014-03-15:10:10:20,MeeToo 1.0,23eba027-b95a-4729-9a4b-a3cca51c5548,0,20,21,86,5
4,34,TRUE,enabled,enabled,39.43789083,-120.9389785
2014-03-15:10:10:20,MeeToo 1.0,16085fbf-cda5-4489-84b9-0fad888f9e7a,0,29,26,97,8
,11,TRUE,enabled,enabled,34.048762,-111.9288717
```

- What happens if you try to call flines.next now that you have already iterated through the whole file?

  It throws java.util.NoSuchElementException: next on empty iterator.

```
scala> flines.next
```

```
scala> flines.next
java.util.NoSuchElementException: next on empty iterator
  at scala.collection.Iterator$$anon$2.next(Iterator.scala:38)
  at scala.collection.Iterator$$anon$2.next(Iterator.scala:36)
  at scala.io.BufferedSource$BufferedLineIterator.next(BufferedSource.scala:79)
  at scala.io.BufferedSource$BufferedLineIterator.next(BufferedSource.scala:64)
  ... 28 elided

scala>
```

Parsing String Input Using split

The next task is to go through each line in the file and parse it to extract the Device ID field

- Start by exploring a single line of data in the Scala shell. Use the following single string to represent one record in the loudacre.log file.

```
scala> val record = "2014-03-15:10:10:31, Titanic 4000,
1882b564-c7e0-4315-aa24-228c0155ee1b, 58, 36, 39, 31, 15, 0,
TRUE, enabled, enabled, 37.819722, -122.478611"
```

```
scala> val record = "2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-
228c0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, enabled, enabled, 37.819722, -122.478
611"
record: String = 2014-03-15:10:10:31, Titanic 4000, 1882b564-c7e0-4315-aa24-228c
0155ee1b, 58, 36, 39, 31, 15, 0, TRUE, enabled, enabled, 37.819722, -122.478611

scala>
```

- Now split the string into its respective fields, using the comma (,) as the delimiter character. The split method creates an array of values.

```
scala> val fields = record.split(',')
```

```
scala> val fields = record.split(',')
fields: Array[String] = Array(2014-03-15:10:10:31, " Titanic 4000", " 1882b564-c
7e0-4315-aa24-228c0155ee1b", " 58", " 36", " 39", " 31", " 15", " 0", " TRUE", "
 enabled", " enabled", " 37.819722", " -122.478611")

scala>
```

- Now write a program that displays the Device ID from each line of data in the loudacre.log data file.

  • Use the String.split method to separate the record into individual fields using the comma delimiters

  • Print out the third field (the device ID field) for each record

```
scala> for (line <- Source.fromFile(fname).getLines()) {
       print(line.split(‘,’)(2))
       }
```

```
scala> for (line <- Source.fromFile(fname).getLines()) {
     | println(line.split(',')(2))
     | }
6474caf1-7bbf-4594-a526-9ba8ea82e151
5c2d40d8-b1e0-4c2a-b050-06ca6c590741
27178d24-3a61-42f7-a784-e3263f25cc6f
fe4674a7-0632-494a-aaf3-f5865a724e1c
86e4bd60-5c72-4558-a019-caf1de9d14f1
ef8c7564-0a1a-4650-a655-c8bbd5f8f943
23eba027-b95a-4729-9a4b-a3cca51c5548
16085fbf-cda5-4489-84b9-0fad888f9e7a
e205ee4a-e7f8-44ba-bfb0-2f4a7a604e09
ff375011-34f0-4758-bade-e68cea787115
8316b507-7620-47aa-b56b-cae5cb2cd819
673f7e4b-d52b-44fc-8826-aea460c3481a
668e6f06-a8aa-4be5-8609-899c45d3caa8
```

Using filter to Limit Processing

- There are six instances of "Titanic 4000" model phones in the Loudacre log file. Modify your solution to the last section to print out the device IDs for these six records only. Use the Iterator.filter method to limit processing to only those lines containing Titanic 4000 phone records.

```
scala>
Source.fromFile(fname).getLines().filter(_.contains("4000")).f
oreach(line => println(line.split(',')(2)))
```

```
scala> Source.fromFile(fname).getLines().filter(_.contains("4000")).foreach(line
 => println(line.split(',')(2)))
27a418dc-9f20-4cb9-9c69-92d16c596bad
3d88c332-61fa-46f0-a171-84a8b02dd52a
c9b5d5d4-19a2-4259-bff7-e915d967d3f4
54994404-ab72-4e5f-a6bc-0d485b3806f2
57d50d31-ce40-440d-a2b3-a041eb9a29f5
1882b564-c7e0-4315-aa24-228c0155ee1b

scala>
```

Exercise4: Spark with Scala

Launching Spark

- Spark shell can be launched in two ways i.e. Scala and Python.
- To launch Scala spark shell, follow below steps

```
$ spark-shell
```

```
[cloudera@quickstart ~]$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/St
aticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
```

You will see several INFO and WARNING message on the command prompt after launching spark-shell, which can be disregarded.

Logs will appear as below and finally you will get Scala Prompt

```
SQL context available as sqlContext.

scala>
```

- Spark creates a SparkContext object called sc, verify that the object exists

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@a23c46d
```

- To know various SparkContext methods which are available, type sc. (sc followed by dot) and then TAB key

```
scala> sc.
accumulable                   accumulableCollection
accumulator                   addFile
addJar                        addSparkListener
appName                       applicationAttemptId
applicationId                 asInstanceOf
binaryFiles                   binaryRecords
broadcast                     cancelAllJobs
cancelJobGroup                clearCallSite
clearFiles                    clearJars
clearJobGroup                 defaultMinPartitions
defaultMinSplits              defaultParallelism
emptyRDD                      externalBlockStoreFolderName
files                         getAllPools
getCheckpointDir              getConf
getExecutorMemoryStatus       getExecutorStorageStatus
getLocalProperty              getPersistentRDDs
getPoolForName                getRDDStorageInfo
getSchedulingMode             hadoopConfiguration
hadoopFile                    hadoopRDD
initLocalProperties           isInstanceOf
isLocal                       isStopped
jars                          killExecutor
killExecutors                 makeRDD
```

Spark is based on the concept of Resilient Distributed Dataset (RDD), which is fault tolerant collection of elements that can be operated in parallel.

Two ways to create RDDs:

- Parallelizing an existing collection
- Referencing a dataset from an External Storage System

Parallelized collection:

To create a parallelized collection holding numbers 1 to 6, use sc.parallelize method

```
val data = Array (1,2,3,4,5,6)

val distData = sc.parallelize(data)
```

Once 'distData' RDD is created, we can perform Action such as:

- Finding the sum, mean & variance

```
distData.sum

distData.mean

distData.variance
```

External Storage System:

Spark can create distributed datasets from any storage system supported by Hadoop, Spark supports text files, Sequence Files and any other Hadoop Input Format.

Load a file from your Local Filesystem say joyce.txt using sc.textFile method.

```
val textFile = sc.textFile
("file:/home/cloudera/dataset/joyce.txt")
```

```
scala> val textFile = sc.textFile("file:/home/cloudera/dataset/joyce.txt")
textFile: org.apache.spark.rdd.RDD[String] = file:/home/cloudera/dataset/joyce.t
xt MapPartitionsRDD[1] at textFile at <console>:32

scala>
```

Operations on RDD will be discussed in next section.

Exploring Data using RDD operations:

Transformations

filter

Return a new dataset formed by selecting those elements of the source on which *func* returns true.

```
val a = sc.parallelize(1 to 10)

val b = a.filter(_% 2 == 0)

b.collect
```

```
scala> val a = sc.parallelize(1 to 10)
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:21

scala> val b = a.filter(_% 2 == 0)
b: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at filter at <console>:23

scala> b.collect
16/08/30 06:59:25 INFO spark.SparkContext: Starting job: collect at <console>:26
```

```
16/08/30 06:59:28 INFO scheduler.DAGScheduler: ResultStage 0 (collect at <console>:26) finished in 0.316 s
16/08/30 06:59:28 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 233 ms on localhost (1/1)
16/08/30 06:59:28 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/08/30 06:59:28 INFO scheduler.DAGScheduler: Job 0 finished: collect at <console>:26, took 2.285340 s
res0: Array[Int] = Array(2, 4, 6, 8, 10)
```

map

Return a new distributed dataset formed by passing each element of the source through a function *func*.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"))

val b = a.map(_.length)

val c = a.zip(b)

c.collect
```

```
scala> val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"))
a: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:21

scala> val b = a.map(_.length)
b: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[3] at map at <console>:23

scala> val c = a.zip(b)
c: org.apache.spark.rdd.RDD[(String, Int)] = ZippedPartitionsRDD2[4] at zip at <console>:25

scala> c.collect
16/08/30 07:03:58 INFO spark.SparkContext: Starting job: collect at <console>:28
```

```
16/08/30 07:03:58 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1). 1147 bytes result sent to driver
16/08/30 07:03:58 INFO scheduler.DAGScheduler: ResultStage 1 (collect at <console>:28) finished in 0.062 s
16/08/30 07:03:58 INFO scheduler.DAGScheduler: Job 1 finished: collect at <console>:28, took 0.108580 s
16/08/30 07:03:58 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 64 ms on localhost (1/1)
res1: Array[(String, Int)] = Array((dog,3), (salmon,6), (salmon,6), (rat,3), (elephant,8))
```

distinct

Return a new dataset that contains the distinct elements of the source dataset.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)

c.distinct.collect
```

```
scala> val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)
c: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[5] at parallelize at <console>:21

scala> c.distinct.collect
16/08/30 07:07:16 INFO spark.SparkContext: Starting job: collect at <console>:24


16/08/30 07:07:19 INFO executor.Executor: Finished task 1.0 in stage 3.0 (TID 5). 1171 bytes result sent to driver
16/08/30 07:07:19 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0 (TID 5) in 76 ms on localhost (2/2)
16/08/30 07:07:19 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/08/30 07:07:19 INFO scheduler.DAGScheduler: ResultStage 3 (collect at <console>:24) finished in 0.385 s
16/08/30 07:07:19 INFO scheduler.DAGScheduler: Job 2 finished: collect at <console>:24, took 2.185199 s
res2: Array[String] = Array(Dog, Cat, Gnu, Rat)
```

cartesian

When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).

```
val x = sc.parallelize(List(1,2,3,4,5))

val y = sc.parallelize(List(6,7,8,9,10))

x.cartesian(y).collect
```

```
scala> val x = sc.parallelize(List(1,2,3,4,5))
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:21

scala> val y = sc.parallelize(List(6,7,8,9,10))
y: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:21

scala> x.cartesian(y).collect
16/08/30 07:10:11 INFO spark.SparkContext: Starting job: collect at <console>:26


16/08/30 07:10:11 INFO scheduler.DAGScheduler: ResultStage 4 (collect at <console>:26) finished in 0.145 s
16/08/30 07:10:11 INFO scheduler.DAGScheduler: Job 3 finished: collect at <console>:26, took 0.264577 s
16/08/30 07:10:11 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0 (TID 6) in 153 ms on localhost (1/1)
16/08/30 07:10:11 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose tasks have all completed, from pool
res3: Array[(Int, Int)] = Array((1,6), (1,7), (1,8), (1,9), (1,10), (2,6), (2,7), (2,8), (2,9), (2,10), (3,6), (3,7), (3,8), (3,9), (3,10), (4,6), (4,7), (4,8), (4,9), (4,10), (5,6), (5,7), (5,8), (5,9), (5,10))
```

coalesce

Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.

```
val y = sc.parallelize(1 to 10, 10)

val z = y.coalesce(2, false)

z.partitions.length
```

```
scala> val y = sc.parallelize(1 to 10, 10)
y: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:21

scala> val z = y.coalesce(2, false)
z: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[13] at coalesce at <console>:23

scala> z.partitions.length
res4: Int = 2
```

filterByRange

Returns an RDD containing only the items in the key range specified.

```
val randRDD = sc.parallelize(List( (2,"cat"), (6, "mouse"),(7, "cup"), (3, "book"), (4, "tv"), (1, "screen"), (5, "heater")), 3)

val sortedRDD = randRDD.sortByKey()

sortedRDD.filterByRange(1,3).collect
```

```
16/08/30 07:14:40 INFO executor.Executor: Finished task 0.0 in stage 7.0 (TID 13). 1357 bytes result sent to driver
16/08/30 07:14:40 INFO scheduler.DAGScheduler: ResultStage 7 (collect at <console>:26) finished in 0.080 s
16/08/30 07:14:40 INFO scheduler.DAGScheduler: Job 5 finished: collect at <console>:26, took 0.527731 s
res5: Array[(Int, String)] = Array((1,screen), (2,cat), (3,book))
```

flatMap

Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

```
val a = sc.parallelize(1 to 10, 5)

a.flatMap(1 to _).collect
```

```
16/08/30 07:18:44 INFO executor.Executor: Finished task 4.0 in stage 9.0 (TID 20). 974 bytes result sent to
16/08/30 07:18:44 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 9.0 (TID 20) in 24 ms on localho
16/08/30 07:18:44 INFO scheduler.DAGScheduler: ResultStage 9 (collect at <console>:24) finished in 0.141 s
16/08/30 07:18:44 INFO scheduler.DAGScheduler: Job 7 finished: collect at <console>:24, took 0.193634 s
res6: Array[Int] = Array(1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7,
, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
val b = sc.parallelize(List(1, 2, 3), 2).flatMap(x => List(x, x, x)).collect

res85: Array[Int] = Array(1, 1, 1, 2, 2, 2, 3, 3, 3)
```

```
16/08/30 07:17:29 INFO executor.Executor: Running task 1.0 in stage 8.0 (TID 15)
16/08/30 07:17:29 INFO executor.Executor: Finished task 1.0 in stage 8.0 (TID 15). 922 bytes result sent to driver
16/08/30 07:17:29 INFO scheduler.DAGScheduler: ResultStage 8 (collect at <console>:23) finished in 0.104 s
16/08/30 07:17:29 INFO scheduler.DAGScheduler: Job 6 finished: collect at <console>:23, took 0.126929 s
b: Array[Int] = Array(1, 1, 1, 2, 2, 2, 3, 3, 3)
```

```
val lines = sc.parallelize(List("hello world", "hi"))

val words = lines.flatMap(line => line.split(" "))

words.first()  // returns "hello"
```

```
16/08/30 07:21:01 INFO scheduler.DAGScheduler: ResultStage 10 (first at <console>:26) finished in 0.006 s
16/08/30 07:21:01 INFO scheduler.DAGScheduler: Job 8 finished: first at <console>:26, took 0.035576 s
res7: String = hello
```

groupBy

```
val a = sc.parallelize(1 to 9, 3)

a.groupBy(x => { if (x % 2 == 0) "even" else "odd" }).collect
```

```
16/08/30 07:23:14 INFO scheduler.DAGScheduler: ResultStage 12 (collect at <console>:26) finished in 0.182 s
16/08/30 07:23:14 INFO scheduler.DAGScheduler: Job 9 finished: collect at <console>:26, took 0.447740 s
res8: Array[(String, Iterable[Int])] = Array((even,CompactBuffer(2, 4, 6, 8)), (odd,CompactBuffer(1, 3, 5, 7
, 9)))
```

keys

Extracts the keys from all contained tuples and returns them in a new RDD.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"panther", "eagle"), 2)

val b = a.map(x => (x.length, x))

b.keys.collect
```

```
16/08/30 07:25:04 INFO scheduler.DAGScheduler: ResultStage 13 (collect at <console>:28) finished in 0.089 s
16/08/30 07:25:04 INFO scheduler.DAGScheduler: Job 10 finished: collect at <console>:28, took 0.131419 s
res9: Array[Int] = Array(3, 5, 4, 3, 7, 5)
```

union

```
val seta = sc.parallelize(1 to 10)

val setb = sc.parallelize(5 to 15)

(seta union setb).collect
```

```
16/08/30 07:26:33 INFO scheduler.DAGScheduler: ResultStage 14 (collect at <console>:26) finished in 0.143 s
16/08/30 07:26:33 INFO scheduler.DAGScheduler: Job 11 finished: collect at <console>:26, took 0.318252 s
res10: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
```

zip

Joins two RDDs by combining the i-th of either partition with each other. The resulting RDD will consist of two-component tuples which are interpreted as key-value pairs by the methods provided by the PairRDDFunctions extension.

```
val a = sc.parallelize(1 to 100, 3)
val b = sc.parallelize(101 to 200, 3)
a.zip(b).collect
```

```
16/08/30 07:30:02 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 15.0 (TID 34) in 35 ms on localh
ost (3/3)
16/08/30 07:30:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 15.0, whose tasks have all completed, fr
om pool
res11: Array[(Int, Int)] = Array((1,101), (2,102), (3,103), (4,104), (5,105), (6,106), (7,107), (8,108), (9,
109), (10,110), (11,111), (12,112), (13,113), (14,114), (15,115), (16,116), (17,117), (18,118), (19,119), (2
0,120), (21,121), (22,122), (23,123), (24,124), (25,125), (26,126), (27,127), (28,128), (29,129), (30,130),
(31,131), (32,132), (33,133), (34,134), (35,135), (36,136), (37,137), (38,138), (39,139), (40,140), (41,141)
, (42,142), (43,143), (44,144), (45,145), (46,146), (47,147), (48,148), (49,149), (50,150), (51,151), (52,15
2), (53,153), (54,154), (55,155), (56,156), (57,157), (58,158), (59,159), (60,160), (61,161), (62,162), (63,
163), (64,164), (65,165), (66,166), (67,167), (68,168), (69,169), (70,170), (71,171), (72,172), (73,173), (7
4,174), (75,175), (76,176), (77,177), (78...
scala>
```

Action

collect

Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu",
"Rat"), 2)

c.collect
```

```
16/08/30 07:31:52 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 16.0 (TID 36) in 7 ms on localho
st (2/2)
16/08/30 07:31:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, fr
om pool
16/08/30 07:31:52 INFO scheduler.DAGScheduler: ResultStage 16 (collect at <console>:24) finished in 0.010 s
16/08/30 07:31:52 INFO scheduler.DAGScheduler: Job 13 finished: collect at <console>:24, took 0.018261 s
res12: Array[String] = Array(Gnu, Cat, Rat, Dog, Gnu, Rat)
```

collectAsMap

Similar to collect, but works on key-value RDDs and converts them into Scala maps to preserve their key-value structure.

```
val a = sc.parallelize(List(1, 2, 1, 3), 1)

val b = a.zip(a)

b.collectAsMap
```

```
16/08/30 07:33:33 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 17.0 (TID 37) in 145 ms on local
host (1/1)
16/08/30 07:33:33 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 17.0, whose tasks have all completed, fr
om pool
res13: scala.collection.Map[Int,Int] = Map(2 -> 2, 1 -> 1, 3 -> 3)
```

count

Return the number of elements in the dataset.

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog"), 2)

c.count

res2: Long = 4
```

```
16/08/30 07:34:52 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 18.0 (TID 39) in 11 ms on localh
ost (2/2)
16/08/30 07:34:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 18.0, whose tasks have all completed, fr
om pool
16/08/30 07:34:52 INFO scheduler.DAGScheduler: ResultStage 18 (count at <console>:24) finished in 0.040 s
16/08/30 07:34:52 INFO scheduler.DAGScheduler: Job 15 finished: count at <console>:24, took 0.054256 s
res14: Long = 4
```

reduce

Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

```
val a = sc.parallelize(1 to 100, 3)

a.reduce(_+_)
```

```
16/08/30 08:32:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 19.0, whose tasks have all completed, fr
om pool
16/08/30 08:32:49 INFO scheduler.DAGScheduler: ResultStage 19 (reduce at <console>:24) finished in 0.099 s
16/08/30 08:32:49 INFO scheduler.DAGScheduler: Job 16 finished: reduce at <console>:24, took 0.315500 s
res15: Int = 5050
```

take

Return an array with the first *n* elements of the dataset.

```
val b = sc.parallelize(List("dog", "cat", "ape", "salmon", "gnu"), 2)

b.take(2)
```

```
16/08/30 08:34:15 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 20.0, whose tasks have all completed, fr
om pool
16/08/30 08:34:15 INFO scheduler.DAGScheduler: ResultStage 20 (take at <console>:24) finished in 0.035 s
16/08/30 08:34:15 INFO scheduler.DAGScheduler: Job 17 finished: take at <console>:24, took 0.142047 s
res16: Array[String] = Array(dog, cat)
```

```
val b = sc.parallelize(1 to 100, 5)

b.take(30)
```

```
16/08/30 08:35:57 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 22.0, whose tasks have all completed, fr
om pool
16/08/30 08:35:58 INFO scheduler.DAGScheduler: ResultStage 22 (take at <console>:24) finished in 0.004 s
16/08/30 08:35:58 INFO scheduler.DAGScheduler: Job 19 finished: take at <console>:24, took 0.064874 s
res17: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
 24, 25, 26, 27, 28, 29, 30)
```

first

Return the first element of the dataset (similar to take(1)).

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog"), 2)

c.first
```

```
16/08/30 08:37:15 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 23.0, whose tasks have all completed, fr
om pool
16/08/30 08:37:15 INFO scheduler.DAGScheduler: ResultStage 23 (first at <console>:24) finished in 0.006 s
16/08/30 08:37:15 INFO scheduler.DAGScheduler: Job 20 finished: first at <console>:24, took 0.016040 s
res18: String = Gnu
```

countByValue

Returns a map that contains all unique values of the RDD and their respective occurrence counts.

```
val b = sc.parallelize(List(1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1))

b.countByValue
```

```
16/08/30 08:38:18 INFO scheduler.DAGScheduler: ResultStage 25 (countByValue at <console>:24) finished in 0.0
40 s
16/08/30 08:38:18 INFO scheduler.DAGScheduler: Job 21 finished: countByValue at <console>:24, took 0.495353
s
res19: scala.collection.Map[Int,Long] = Map(5 -> 1, 1 -> 6, 6 -> 1, 2 -> 3, 7 -> 1, 3 -> 1, 8 -> 1, 4 -> 2)
```

lookup

Scans the RDD for all keys that match the provided value and returns their values as a Scala sequence.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat",
"panther", "eagle"), 2)

val b = a.map(x => (x.length, x))

b.lookup(5)
```

```
16/08/30 08:39:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, fr
om pool
16/08/30 08:39:47 INFO scheduler.DAGScheduler: ResultStage 26 (lookup at <console>:28) finished in 0.087 s
16/08/30 08:39:47 INFO scheduler.DAGScheduler: Job 22 finished: lookup at <console>:28, took 0.188596 s
res20: Seq[String] = WrappedArray(tiger, eagle)
```

max

Returns the largest element in the RDD

```
val y = sc.parallelize(10 to 30)

y.max
```

```
16/08/30 08:41:08 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 27.0, whose tasks have all completed, fr
om pool
16/08/30 08:41:08 INFO scheduler.DAGScheduler: ResultStage 27 (max at <console>:24) finished in 0.042 s
16/08/30 08:41:08 INFO scheduler.DAGScheduler: Job 23 finished: max at <console>:24, took 0.115601 s
res21: Int = 30
```

```
val a = sc.parallelize(List((10, "dog"), (3, "tiger"), (9,
"lion"), (18, "cat")))

a.max
```

```
16/08/30 08:42:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 28.0, whose tasks have all completed, fr
om pool
16/08/30 08:42:29 INFO scheduler.DAGScheduler: ResultStage 28 (max at <console>:24) finished in 0.031 s
16/08/30 08:42:29 INFO scheduler.DAGScheduler: Job 24 finished: max at <console>:24, took 0.054414 s
res22: (Int, String) = (18,cat)
```

min

Returns the smallest element in the RDD

```
val y = sc.parallelize(10 to 30)

y.min
```

```
16/08/30 08:43:49 INFO executor.Executor: Finished task 0.0 in stage 29.0 (TID 53). 1031 bytes result sent t
o driver
16/08/30 08:43:49 INFO scheduler.DAGScheduler: ResultStage 29 (min at <console>:24) finished in 0.075 s
16/08/30 08:43:49 INFO scheduler.DAGScheduler: Job 25 finished: min at <console>:24, took 0.170125 s
res23: Int = 10
```

```
val a = sc.parallelize(List((10, "dog"), (3, "tiger"), (9,
"lion"), (8, "cat")))

a.min
```

```
16/08/30 08:45:21 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 30.0, whose tasks have all completed, fr
om pool
16/08/30 08:45:21 INFO scheduler.DAGScheduler: ResultStage 30 (min at <console>:24) finished in 0.012 s
16/08/30 08:45:21 INFO scheduler.DAGScheduler: Job 26 finished: min at <console>:24, took 0.020695 s
res24: (Int, String) = (3,tiger)
```

mean

Calls stats and extracts the mean component.

```
val a = sc.parallelize(List(9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0,
2.1, 7.4, 7.5, 7.6, 8.8, 10.0, 8.9, 5.5), 3)

a.mean
```

```
16/08/30 08:46:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 31.0, whose tasks have all completed, fr
om pool
16/08/30 08:46:49 INFO scheduler.DAGScheduler: ResultStage 31 (mean at <console>:24) finished in 0.425 s
16/08/30 08:46:49 INFO scheduler.DAGScheduler: Job 27 finished: mean at <console>:24, took 0.526866 s
res25: Double = 5.3
```

variance

Calls stats and extracts either variance-component or corrected sampleVariance-component.

```
val a = sc.parallelize(List(9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0,
2.1, 7.4, 7.5, 7.6, 8.8, 10.0, 8.9, 5.5), 3)
a.variance
```

```
16/08/30 08:48:05 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 32.0, whose tasks have all completed, fr
om pool
16/08/30 08:48:05 INFO scheduler.DAGScheduler: ResultStage 32 (variance at <console>:24) finished in 0.018 s
16/08/30 08:48:05 INFO scheduler.DAGScheduler: Job 28 finished: variance at <console>:24, took 0.032808 s
res26: Double = 10.605333333333332
```

PairRDD

countByKey

Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

```
val c = sc.parallelize(List((3, "Gnu"), (3, "Yak"), (5, "Mouse"), (3, "Dog")), 2)

c.countByKey
```

```
16/08/30 08:49:39 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 34.0, whose tasks have all completed, fr
om pool
16/08/30 08:49:39 INFO scheduler.DAGScheduler: ResultStage 34 (countByKey at <console>:24) finished in 0.073
 s
16/08/30 08:49:39 INFO scheduler.DAGScheduler: Job 29 finished: countByKey at <console>:24, took 0.384196 s
res27: scala.collection.Map[Int,Long] = Map(3 -> 3, 5 -> 1)
```

groupByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2)

val b = a.keyBy(_.length)

b.groupByKey.collect
```

```
16/08/30 08:51:02 INFO executor.Executor: Finished task 1.0 in stage 36.0 (TID 68). 1703 bytes result sent t
o driver
16/08/30 08:51:02 INFO scheduler.DAGScheduler: ResultStage 36 (collect at <console>:26) finished in 0.042 s
16/08/30 08:51:02 INFO scheduler.DAGScheduler: Job 30 finished: collect at <console>:26, took 0.415007 s
res28: Array[(Int, Iterable[String])] = Array((4,CompactBuffer(lion)), (6,CompactBuffer(spider)), (3,Compact
Buffer(dog, cat)), (5,CompactBuffer(tiger, eagle)))
```

reduceByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V

```
val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2)
```

```
val b = a.map(x => (x.length, x))

b.reduceByKey(_ + _).collect
```

```
16/08/30 08:52:58 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 38.0 (TID 72) in 25 ms on localh
ost (2/2)
16/08/30 08:52:58 INFO scheduler.DAGScheduler: ResultStage 38 (collect at <console>:28) finished in 0.033 s
16/08/30 08:52:58 INFO scheduler.DAGScheduler: Job 31 finished: collect at <console>:28, took 0.413629 s
res29: Array[(Int, String)] = Array((3,dogcatowlgnuant))
```

foldByKey

Very similar to *fold*, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists of two-component tuples.

```
val deptEmployees =
  List
  (
   ("dept1",("kumar1",1000.0)),
   ("dept1",("kumar2",1200.0)),
   ("dept2",("kumar3",2200.0)),
   ("dept2",("kumar4",1400.0)),
   ("dept2",("kumar5",1000.0)),
   ("dept2",("kumar6",800.0)),
   ("dept1",("kumar7",2000.0)),
   ("dept1",("kumar8",1000.0)),
   ("dept1",("kumar9",500.0))
   )


val employeeRDD = sc.makeRDD(deptEmployees)


val maxByDept =
employeeRDD.foldByKey(("dummy",Double.MinValue))((acc,element)
=> if(acc._2 > element._2)acc else element)
```

```
println("Maximum salaries in each dept" +
maxByDept.collect().toList)
```

```
16/08/30 08:56:35 INFO executor.Executor: Finished task 0.0 in stage 40.0 (TID 74). 1375 bytes result sent t
o driver
16/08/30 08:56:35 INFO scheduler.DAGScheduler: ResultStage 40 (collect at <console>:28) finished in 0.087 s
16/08/30 08:56:35 INFO scheduler.DAGScheduler: Job 32 finished: collect at <console>:28, took 0.351474 s
Maximum salaries in each deptList((dept2,(kumar3,2200.0)), (dept1,(kumar7,2000.0)))
```

cogroup

When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples.

```
val a = sc.parallelize(List(1, 2, 1, 3), 1)

val b = a.map((_, "b"))

val c = a.map((_, "c"))

b.cogroup(c).collect
```

```
16/08/30 08:58:50 INFO executor.Executor: Finished task 0.0 in stage 43.0 (TID 77). 2177 bytes result sent t
o driver
16/08/30 08:58:50 INFO scheduler.DAGScheduler: ResultStage 43 (collect at <console>:28) finished in 0.249 s
16/08/30 08:58:50 INFO scheduler.DAGScheduler: Job 33 finished: collect at <console>:28, took 0.668327 s
res31: Array[(Int, (Iterable[String], Iterable[String]))] = Array((1,(CompactBuffer(b, b),CompactBuffer(c, c
))), (3,(CompactBuffer(b),CompactBuffer(c))), (2,(CompactBuffer(b),CompactBuffer(c))))
```

```
val x = sc.parallelize(List((1, "apple"), (2, "banana"), (3,
"orange"), (4, "kiwi")), 2)

val y = sc.parallelize(List((5, "computer"), (1, "laptop"),
(1, "desktop"), (4, "iPad")), 2)

x.cogroup(y).collect
```

```
16/08/30 09:00:55 INFO executor.Executor: Finished task 1.0 in stage 46.0 (TID 83). 2189 bytes result sent t
o driver
16/08/30 09:00:55 INFO scheduler.DAGScheduler: ResultStage 46 (collect at <console>:26) finished in 0.123 s
16/08/30 09:00:55 INFO scheduler.DAGScheduler: Job 34 finished: collect at <console>:26, took 0.624333 s
res32: Array[(Int, (Iterable[String], Iterable[String]))] = Array((4,(CompactBuffer(kiwi),CompactBuffer(iPad
))), (2,(CompactBuffer(banana),CompactBuffer())), (1,(CompactBuffer(apple),CompactBuffer(laptop, desktop))),
 (3,(CompactBuffer(orange),CompactBuffer())), (5,(CompactBuffer(),CompactBuffer(computer))))
```

Join

Performs an inner join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)

val b = a.keyBy(_.length)

val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turke
y","wolf","bear","bee"), 3)

val d = c.keyBy(_.length)

b.join(d).collect
```

```
16/08/30 09:02:41 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 49.0, whose tasks have all completed, fr
om pool
16/08/30 09:02:41 INFO scheduler.DAGScheduler: ResultStage 49 (collect at <console>:30) finished in 0.470 s
16/08/30 09:02:41 INFO scheduler.DAGScheduler: Job 35 finished: collect at <console>:30, took 1.019005 s
res33: Array[(Int, (String, String))] = Array((6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)),
 (6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)), (3,(dog,dog)), (3,(dog,cat)), (3,(dog,gnu)),
 (3,(dog,bee)), (3,(rat,dog)), (3,(rat,cat)), (3,(rat,gnu)), (3,(rat,bee)))
```

leftOuterJoin

Performs a left outer join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)

val b = a.keyBy(_.length)

val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turke
y","wolf","bear","bee"), 3)

val d = c.keyBy(_.length)

b.leftOuterJoin(d).collect
```

```
16/08/30 09:03:59 INFO scheduler.DAGScheduler: Job 36 finished: collect at <console>:30, took 0.675634 s
16/08/30 09:03:59 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 52.0, whose tasks have all completed, fr
om pool
res34: Array[(Int, (String, Option[String]))] = Array((6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))),
(6,(salmon,Some(turkey))), (6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))), (6,(salmon,Some(turkey))),
(3,(dog,Some(dog))), (3,(dog,Some(cat))), (3,(dog,Some(gnu))), (3,(dog,Some(bee))), (3,(rat,Some(dog))), (3,
(rat,Some(cat))), (3,(rat,Some(gnu))), (3,(rat,Some(bee))), (8,(elephant,None))
```

rightOuterJoin

Performs a right outer join using two key-value RDDs.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)
```

```
val b = a.keyBy(_.length)

val c =
sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turke
y","wolf","bear","bee"), 3)

val d = c.keyBy(_.length)

b.rightOuterJoin(d).collect
```

```
16/08/30 09:05:30 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 55.0, whose tasks have all completed, fr
om pool
16/08/30 09:05:30 INFO scheduler.DAGScheduler: ResultStage 55 (collect at <console>:30) finished in 0.181 s
16/08/30 09:05:30 INFO scheduler.DAGScheduler: Job 37 finished: collect at <console>:30, took 1.003812 s
res35: Array[(Int, (Option[String], String))] = Array((6,(Some(salmon),salmon)), (6,(Some(salmon),rabbit)),
(6,(Some(salmon),turkey)), (6,(Some(salmon),salmon)), (6,(Some(salmon),rabbit)), (6,(Some(salmon),turkey)),
(3,(Some(dog),dog)), (3,(Some(dog),cat)), (3,(Some(dog),gnu)), (3,(Some(dog),bee)), (3,(Some(rat),dog)), (3,
(Some(rat),cat)), (3,(Some(rat),gnu)), (3,(Some(rat),bee)), (4,(None,wolf)), (4,(None,bear)))
```

keyBy

Constructs two-component tuples (key-value pairs) by applying a function on each data item. The result of the function becomes the key and the original data item becomes the value of the newly created tuples.

```
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat",
"elephant"), 3)

val b = a.keyBy(_.length)

b.collect
```

```
16/08/30 09:14:29 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 56.0 (TID 113) in 7 ms on localh
ost (3/3)
16/08/30 09:14:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 56.0, whose tasks have all completed, fr
om pool
16/08/30 09:14:29 INFO scheduler.DAGScheduler: ResultStage 56 (collect at <console>:26) finished in 0.053 s
16/08/30 09:14:29 INFO scheduler.DAGScheduler: Job 38 finished: collect at <console>:26, took 0.073531 s
res36: Array[(Int, String)] = Array((3,dog), (6,salmon), (6,salmon), (3,rat), (8,elephant))
```

mapValues

Takes the values of a RDD that consists of two-component tuples, and applies the provided function to transform each value. Then, it forms new two-component tuples using the key and the transformed value and stores them in a new RDD.

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)

val b = a.map(x => (x.length, x))

b.mapValues("x" + _ + "x").collect
```

```
16/08/30 09:21:21 INFO executor.Executor: Finished task 1.0 in stage 57.0 (TID 115). 1118 bytes result sent to driver
16/08/30 09:21:21 INFO scheduler.DAGScheduler: ResultStage 57 (collect at <console>:28) finished in 0.082 s
16/08/30 09:21:21 INFO scheduler.DAGScheduler: Job 39 finished: collect at <console>:28, took 0.182411 s
res37: Array[(Int, String)] = Array((3,xdogx), (5,xtigerx), (4,xlionx), (3,xcatx), (7,xpantherx), (5,xeaglex))
```

cache

The cache( ) method is a shorthand for using the default storage level, which is StorageLevel.MEMORY_ONLY (store deserialized objects in memory)

```
val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)

c.getStorageLevel
```

```
scala> val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2)
c: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[99] at parallelize at <console>:21

scala> c.getStorageLevel
res38: org.apache.spark.storage.StorageLevel = StorageLevel(false, false, false, false, 1)
```

```
c.cache

c.getStorageLevel
```

```
scala> c.cache
res39: c.type = ParallelCollectionRDD[99] at parallelize at <console>:21

scala> c.getStorageLevel
res40: org.apache.spark.storage.StorageLevel = StorageLevel(false, true, false, true, 1)
```

repartition

Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.

```
val rdd = sc.parallelize(List(1, 2, 10, 4, 5, 2, 1, 1, 1), 3)

rdd.partitions.length

val rdd2 = rdd.repartition(5)

rdd2.partitions.length
```

```
scala> val rdd = sc.parallelize(List(1, 2, 10, 4, 5, 2, 1, 1, 1), 3)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[102] at parallelize at <console>:21

scala> rdd.partitions.length
res43: Int = 3

scala> val rdd2 = rdd.repartition(5)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[106] at repartition at <console>:23

scala> rdd2.partitions.length
res44: Int = 5
```

Developing with Spark

REPL

Example: Counting the occurrence of lines having a specific word in it

```
val textFile =
sc.textFile("file:/home/cloudera/dataset/joyce.txt")
```

```
scala> val textFile = sc.textFile("file:/home/cloudera/dataset/joyce.txt")
textFile: org.apache.spark.rdd.RDD[String] = file:/home/cloudera/dataset/joyce.t
xt MapPartitionsRDD[1] at textFile at <console>:32

scala>
```

```
textFile.count() //Return the number of elements in the
dataset
```

```
16/08/30 09:34:43 INFO executor.Executor: Finished task 0.0 in stage 59.0 (TID 136). 2082 bytes result sent
to driver
16/08/30 09:34:43 INFO scheduler.DAGScheduler: ResultStage 59 (count at <console>:24) finished in 2.238 s
16/08/30 09:34:43 INFO scheduler.DAGScheduler: Job 41 finished: count at <console>:24, took 2.310447 s
res45: Long = 33056
```

```
textFile.first() //Return the first element of the dataset
```

```
16/08/30 09:35:03 INFO executor.Executor: Finished task 0.0 in stage 60.0 (TID 137). 2101 bytes result sent
to driver
16/08/30 09:35:03 INFO scheduler.DAGScheduler: ResultStage 60 (first at <console>:24) finished in 0.035 s
16/08/30 09:35:03 INFO scheduler.DAGScheduler: Job 42 finished: first at <console>:24, took 0.091601 s
res46: String = The Project Gutenberg EBook of Ulysses, by James Joyce
```

```
val linesWithJoyce = textFile.filter(line =>
line.contains("Joyce"))

linesWithJoyce.count() //How many lines contains "Joyce"
```

```
16/08/30 09:35:26 INFO executor.Executor: Finished task 0.0 in stage 61.0 (TID 138). 2082 bytes result sent
to driver
16/08/30 09:35:26 INFO scheduler.DAGScheduler: ResultStage 61 (count at <console>:26) finished in 0.064 s
16/08/30 09:35:26 INFO scheduler.DAGScheduler: Job 43 finished: count at <console>:26, took 0.094628 s
res47: Long = 4
```

```
Example: Add up the sizes of all the lines

val lineLength = textFile.map(s => s.length)

val totalLength = lineLength.reduce((a, b) => a + b)
//Aggregate the elements of the dataset using a function
```

```
16/08/30 09:40:54 INFO executor.Executor: Finished task 0.0 in stage 62.0 (TID 139). 2160 bytes result sent
to driver
16/08/30 09:40:54 INFO scheduler.DAGScheduler: ResultStage 62 (reduce at <console>:31) finished in 0.262 s
16/08/30 09:40:54 INFO scheduler.DAGScheduler: Job 44 finished: reduce at <console>:31, took 0.337723 s
totalLength: Int = 1506967
```

```
Example: To find out the line with maximum words

textFile.map(line => line.split(" ").size).reduce((a,b) =>
if(a>b) a else b)
```

```
16/08/30 09:41:48 INFO executor.Executor: Finished task 0.0 in stage 63.0 (TID 140). 2160 bytes result sent
to driver
16/08/30 09:41:48 INFO scheduler.DAGScheduler: ResultStage 63 (reduce at <console>:30) finished in 1.048 s
16/08/30 09:41:48 INFO scheduler.DAGScheduler: Job 45 finished: reduce at <console>:30, took 1.070554 s
res48: Int = 22
```

Analyze Movie Lens Dataset using RDD

Source: http://grouplens.org/datasets/movielens/

Objective: What are the occupations of the users who have rated the iconic movie Titanic as 5?

Information about Dataset:

RATINGS FILE DESCRIPTION

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

UserIDs range between 1 and 6040

MovieIDs range between 1 and 3952

Ratings are made on a 5-star scale (whole-star ratings only)

Timestamp is represented in seconds since the epoch as returned by time(2)

Each user has at least 20 ratings

USERS FILE DESCRIPTION

User information is in the file "users.dat" and is in the following format:

UserID::Gender::Age::Occupation::Zip-code

All demographic information is provided voluntarily by the users and is not checked for accuracy. Only users who have provided some demographic information is included in this data set.

Gender is denoted by a "M" for male and "F" for female

Age is chosen from the following ranges:

1: "Under 18"

18: "18-24"

25: "25-34"

35: “35-44”

45: “45-49”

50: “50-55”

56: “56+”

Occupation is chosen from the following choices:

0: “other” or not specified

1: “academic/educator”

2: “artist”

3: “clerical/admin”

4: “college/grad student”

5: “customer service”

6: “doctor/health care”

7: “executive/managerial”

8: “farmer”

9: “homemaker”

10: “K-12 student”

11: “lawyer”

12: “programmer”

13: “retired”

14: “sales/marketing”

15: “scientist”

16: “self-employed”

17: “technician/engineer”

18: “tradesman/craftsman”

19: “unemployed”

20: “writer”

MOVIES FILE DESCRIPTION

Movie information is in the file “movies.dat” and is in the following format:

MovieID::Title::Genres

Titles are identical to titles provided by the IMDB (including year of release)

Genres are pipe-separated and are selected from the following genres:

Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western

Steps:

Load the files 'movies.dat', 'users.dat' and 'ratings.dat' into HDFS from local filesystem (datasets directory)

```
hdfs dfs -mkdir ml-1m

cd /home/cloudera/dataset/ml-1m

hdfs dfs -put *.dat ml-1m
```

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir ml-1m

[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ cd dataset/
[cloudera@quickstart dataset]$ cd ml-1m/
[cloudera@quickstart ml-1m]$ hdfs dfs -put *.dat ml-1m
[cloudera@quickstart ml-1m]$ hdfs dfs -ls ml-1m
Found 3 items
-rw-r--r--   1 cloudera cloudera     171308 2018-04-15 09:12 ml-1m/movies.dat
-rw-r--r--   1 cloudera cloudera   24594131 2018-04-15 09:12 ml-1m/ratings.dat
-rw-r--r--   1 cloudera cloudera     134368 2018-04-15 09:12 ml-1m/users.dat
[cloudera@quickstart ml-1m]$
```

1. Create 3 RDD for movies, users and ratings respectively

```
val movies = sc.textFile("ml-1m/movies.dat")

val users = sc.textFile("ml-1m/users.dat")

val ratings = sc.textFile("ml-1m/ratings.dat")
```

2. Print first 10 lines from the movies rdd

```
movies.take(10).foreach(println)
```

```
16/08/31 23:41:52 INFO scheduler.DAGScheduler: Job 0 finished: take at <console>
:24, took 1.564272 s
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children's
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
```

3. Find out the id of the movie Titanic

```
movies.filter{_.contains("Titanic")}.collect
```

```
16/08/31 23:42:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose t
asks have all completed, from pool
res1: Array[String] = Array(1721::Titanic (1997)::Drama|Romance, 2157::Chamberma
id on the Titanic, The (1998)::Romance, 3403::Raise the Titanic (1980)::Drama|Th
riller, 3404::Titanic (1953)::Action|Drama)
```

4. Print 10 lines from ratings RDD

```
ratings.take(10).foreach(println)
```

```
16/08/31 23:43:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose t
asks have all completed, from pool
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
1::1287::5::978302039
1::2804::5::978300719
1::594::4::978302268
1::919::4::978301368
```

5. Create a filtered rdd based on the users who have rated Titanic as 1

```
val titanicRating1 = ratings.map(_.split("::")).filter{rec =>
rec(0).toInt == 1721 && rec(2).toInt == 5}

titanicRating1.take(10).foreach(rec =>
println(rec.mkString("\t")))
```

```
16/08/31 23:44:48 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose t
asks have all completed, from pool
1721    3087    5       974709030
1721    1485    5       974706178
1721    3247    5       974706026
1721    2463    5       974708897
1721    2321    5       974707939
1721    1569    5       974706315
1721    3397    5       974708938
1721    2918    5       974708773
```

6. Convert the previous RDD by assigning user id as key

```
val byUser = titanicRating1.keyBy(_(1))

byUser.take(3)
```

```
res4: Array[(String, Array[String])] = Array((3087,Array(1721, 3087, 5, 97470903
0)), (1485,Array(1721, 1485, 5, 974706178)), (3247,Array(1721, 3247, 5, 97470602
6)))
```

7. Show some sample values from users rdd

```
users.take(10).foreach(println)
```

```
16/08/31 23:47:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose t
asks have all completed, from pool
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
4::M::45::7::02460
5::M::25::20::55455
6::F::50::9::55117
7::M::35::1::06810
8::M::25::12::11413
9::M::25::17::61614
10::F::35::1::95370
```

8. Create pair RDD pairRddUsers from users rdd using userid as key

```
val pairRddUsers = users.keyBy(_.split("::")(0))

pairRddUsers.take(3)
```

```
16/08/31 23:47:53 INFO scheduler.DAGScheduler: Job 6 finished: take at <console>
:26, took 0.354851 s
res6: Array[(String, String)] = Array((1,1::F::1::10::48067), (2,2::M::56::16::7
0072), (3,3::M::25::15::55117))
```

9. Join the last Pair RDD to users RDD

```
val rddJoined = byUser.join(pairRddUsers)

rddJoined.take(3)
```

```
16/08/31 23:49:35 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 9.0, whose t
asks have all completed, from pool
res7: Array[(String, (Array[String], String))] = Array((3247,(Array(1721, 3247,
5, 974706026),3247::M::35::15::92649)), (3397,(Array(1721, 3397, 5, 974708938),3
397::M::25::5::55414)), (1485,(Array(1721, 1485, 5, 974706178),1485::F::25::9::8
0538)))
```

10.Extract the occupations of the users from the joined RDD

```
rddJoined.map(_._2._2.split("::")(3)).distinct.collect
```

```
16/08/31 23:50:12 INFO scheduler.DAGScheduler: Job 8 finished: collect at <conso
le>:34, took 1.229909 s
res8: Array[String] = Array(7, 15, 5, 9, 12, 13, 1)
```

Assignment: Analyse Crime Data from SFPD

Objective: Using above transformation and action, Analyse Crime Data from San Francisco Police Department.

Refer: Zeppelin netebook SFPD Dataset Assignment

Dataset: /home/cloudera/dataset/crime_incidents.csv

```
[cloudera@quickstart dataset]$ pwd
/home/cloudera/dataset
[cloudera@quickstart dataset]$ ls -ltr crime*
-rwxrw-rw- 1 cloudera cloudera 382725387 Mar 25  2016 crime_incidents.csv
[cloudera@quickstart dataset]$
```

Upload to HDFS:

```
hdfs dfs -put crime_incidents.csv input
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put crime_incidents.csv input
[cloudera@quickstart dataset]$ hdfs dfs -ls input
Found 1 items
-rw-r--r--   1 cloudera cloudera  382725387 2018-04-15 09:16 input/crime_inciden
ts.csv
[cloudera@quickstart dataset]$
```

Steps:

1. Create an RDD with crime incident data.
2. Take first five records of RDD created and print the same.
3. Reference each field with an index map.
4. Show the first value of an RDD.
5. Check number of partition.
6. Capture the line containing header in RDD.
7. Create an RDD with only records eliminating the header.
8. Find the number of incident records?
9. List all the categories from incident data?
10. Find total number of categories?
11. What are the total number of incidents in each category?
12. Find out on which day each of incidents occurred most?

Exercise6: Spark with Python

RDD Transformations and Actions

Important Terms

Let's quickly go over some important terms:

Term	Definition
RDD	Resilient Distributed Dataset
Transformation	Spark operation that produces an RDD
Action	Spark operation that produces a local object
Spark Job	Sequence of transformations on data with a final action

Creating an RDD

There are two common ways to create an RDD:

Method	Result
sc.parallelize(array)	Create RDD of elements of array (or list)
sc.textFile(path/to/file)	Create RDD of lines from file

RDD Transformations

We can use transformations to create a set of instructions we want to perform on the RDD (before we call an action and actually execute them).

Transformation Example	Result
filter(lambda x: x % 2 == 0)	Discard non-even elements

Transformation Example	Result
map(lambda x: x * 2)	Multiply each RDD element by 2
map(lambda x: x.split())	Split each string into words
flatMap(lambda x: x.split())	Split each string into words and flatten sequence
sample(withReplacement=True,0.25)	Create sample of 25% of elements with replacement
union(rdd)	Append rdd to existing RDD
distinct()	Remove duplicates in RDD
sortBy(lambda x: x, ascending=False)	Sort elements in descending order

RDD Actions

Once you have your 'recipe' of transformations ready, what you will do next is execute them by calling an action. Here are some common actions:

Action	Result
collect()	Convert RDD to in-memory list
take(3)	First 3 elements of RDD
top(3)	Top 3 elements of RDD
takeSample(withReplacement=True,3)	Create sample of 3 elements with replacement
sum()	Find element sum (assumes numeric elements)
mean()	Find element mean (assumes numeric elements)

Action	Result
stdev()	Find element deviation (assumes numeric elements)

Operations on RDD

To load data:

```
lines = sc.parallelize(["hello world", "hi"])

words = lines.flatMap(lambda line: line.split(" "))

words.first()  # returns "hello"
```

```
>>> lines = sc.parallelize(["hello world", "hi"])
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> words.first()
16/09/17 09:38:59 INFO spark.SparkContext: Starting job: runJob at PythonRDD.sca
la:393

'hello'
>>>
```

```
nums = sc.parallelize([1, 2, 3, 4])

squared = nums.map(lambda x: x * x).collect()

for num in squared: print "%i " % (num)
```

```
>>> for num in squared: print "%i " % (num)
...
1
4
9
16
>>>
```

Creating an RDD from a text file:

Creating the textfile

%%writefile test2.txt

```
first

second line

the third line

then a fourth line
```

Now let's perform some transformations and actions on this text file:

```
# Show RDD

sc.textFile('/user/cloudera/dataset/test2.txt')
```

```
# Save a reference to this RDD

text_rdd = sc.textFile('/user/cloudera/dataset/test2.txt')
```

```
In [6]: text_rdd = sc.textFile('/user/training/data/test2.txt')
```

```
# Map a function (or lambda expression) to each line

# Then collect the results.

text_rdd.map(lambda line: line.split()).collect()
```

```
In [7]: text_rdd.map(lambda line: line.split()).collect()
Out[7]:
[[u'first'],
 [u'second', u'line'],
 [u'the', u'third', u'line'],
 [u'then', u'a', u'fourth', u'line'],
 []]
```

Map vs flatMap

```
# Collect everything as a single flat map

text_rdd.flatMap(lambda line: line.split()).collect()
```

```
In [8]: text_rdd.flatMap(lambda line: line.split()).collect()
Out[8]:
[u'first',
 u'second',
 u'line',
 u'the',
 u'third',
 u'line',
 u'then',
 u'a',
 u'fourth',
 u'line']
```

RDDs and Key Value Pairs

Now that we've worked with RDDs and how to aggregate values with them, we can begin to consider working with Key Value Pairs. To do this, let's create some fake data as a new text file.

This data represents some services sold to customers for some SAAS business.

```
%%writefile services.txt
```

```
#EventId    Timestamp    Customer    State    ServiceID
Amount

201        10/13/2017      100        NY        131
100.00

204        10/18/2017      700        TX        129
450.00

202        10/15/2017      203        CA        121
200.00

206        10/19/2017      202        CA        131
500.00

203        10/17/2017      101        NY        173
750.00

205        10/19/2017      202        TX        121
200.00
```

```
[cloudera@quickstart dataset]$ pwd
/home/cloudera/dataset
[cloudera@quickstart dataset]$ ls services.txt
services.txt
[cloudera@quickstart dataset]$
```

```
[cloudera@quickstart dataset]$ cat services.txt
#EventId    Timestamp    Customer   State    ServiceID    Amount
201       10/13/2017      100       NY       131          100.00
204       10/18/2017      700       TX       129          450.00
202       10/15/2017      203       CA       121          200.00
206       10/19/2017      202       CA       131          500.00
203       10/17/2017      101       NY       173          750.00
205       10/19/2017      202       TX       121          200.00

[cloudera@quickstart dataset]$
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put services.txt input
[cloudera@quickstart dataset]$ hdfs dfs -ls input
Found 3 items
-rw-r--r--   1 cloudera cloudera  382725387 2018-04-15 09:16 input/crime_incidents.csv
-rw-r--r--   1 cloudera cloudera        764 2018-04-18 10:19 input/joyce.txt
-rw-r--r--   1 cloudera cloudera        456 2018-04-18 10:34 input/services.txt
[cloudera@quickstart dataset]$
```

```
services = sc.textFile('/user/cloudera/input/services.txt')
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0.cloudera2
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkSession available as 'spark'.
>>> services = sc.textFile('/user/cloudera/input/services.txt')
>>>
```

```
services.take(2)
```

```
[u'#EventId    Timestamp    Customer   State    ServiceID    Amount', u'201
  10/13/2017      100       NY       131          100.00']
>>>
```

```
services.map(lambda x: x.split())
```

```
>>> services.map(lambda x: x.split())
PythonRDD[3] at RDD at PythonRDD.scala:43
>>>
```

```
services.map(lambda x: x.split()).take(3)
```

```
[[u'#EventId', u'Timestamp', u'Customer', u'State', u'ServiceID', u'Amount'], [u
'201', u'10/13/2017', u'100', u'NY', u'131', u'100.00'], [u'204', u'10/18/2017',
 u'700', u'TX', u'129', u'450.00']]
>>>
```

Let's remove that first hash-tag!

```
services.map(lambda x: x[1:] if x[0]=='#' else x).take(3)
```

```
[u'EventId    Timestamp    Customer   State    ServiceID    Amount', u'201
 10/13/2017      100       NY       131          100.00', u'204       10/18/2017
      700       TX       129          450.00']
>>>
```

```
services.map(lambda x: x[1:] if x[0]=='#' else x).map(lambda
x: x.split()).take(2)
```

```
[[u'EventId', u'Timestamp', u'Customer', u'State', u'ServiceID', u'Amount'], [u'
201', u'10/13/2017', u'100', u'NY', u'131', u'100.00']]
>>>
```

Using Key Value Pairs for Operations

Let us now begin to use methods that combine lambda expressions that use a ByKey argument. These ByKey methods will assume that your data is in a Key,Value form.

For example, let's find out the total sales per state:

```
# From Previous
cleanServ = services.map(lambda x: x[1:] if x[0]=='#' else
x).map(lambda x: x.split())

cleanServ.take(4)
```

```
[[u'EventId', u'Timestamp', u'Customer', u'State', u'ServiceID', u'Amount'], [u'
201', u'10/13/2017', u'100', u'NY', u'131', u'100.00'], [u'204', u'10/18/2017',
u'700', u'TX', u'129', u'450.00'], [u'202', u'10/15/2017', u'203', u'CA', u'121'
, u'200.00']]
```

```
# Practice grabbing fields
cleanServ.map(lambda lst: (lst[3],lst[-1])).take(5)
```

```
[(u'State', u'Amount'), (u'NY', u'100.00'), (u'TX', u'450.00'), (u'CA', u'200.00
'), (u'CA', u'500.00')]
>>>
```

Working with RDDs in Python

- In this exercise, you will use the Spark shell to work with RDDs.

Data files (local)

/home/cloudera/dataset/frostroad.txt

/home/cloudera/dataset/makes1.txt

/home/cloudera/dataset/makes2.txt

- Read and Display Data from a Text File

In this exercise, you will use the Spark shell to work with RDDs.

You will start reading a simple text file into a Resilient Distributed Dataset (RDD) and displaying the contents. You

will then create two new RDDs and use transformations to union them and remove duplicates.

- Upload the text file to HDFS directory loudacre.

```
$ hdfs dfs -mkdir loudacre

$ hdfs dfs -put /home/cloudera/dataset/frostroad.txt loudacre
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put frostroad.txt loudacre
[cloudera@quickstart dataset]$ hdfs dfs -ls loudacre
Found 7 items
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 08:06 loudacre/accountdevice
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_zip94913
-rw-r--r--   1 cloudera cloudera       5483 2018-04-15 09:20 loudacre/devices.json
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:49 loudacre/devices_parquet
-rw-r--r--   1 cloudera cloudera        730 2018-04-18 10:40 loudacre/frostroad.txt
```

- In the Spark shell, define an RDD based on the frostroad.txt text file.

```
pyspark> myRDD =
sc.textFile("/user/cloudera/loudacre/frostroad.txt")
```

```
>>> myRDD = sc.textFile("/user/cloudera/loudacre/frostroad.txt")
>>>
```

- Using command completion, you can see all the available transformations and operations you can perform on an RDD. Type myRDD. and then the TAB key.

- Spark has not yet read the file. It will not do so until you perform an action on the RDD. Try counting the number of elements in the RDD using the count action:

```
pyspark> myRDD.count()
```

```
17/12/06 19:19:08 INFO scheduler.DAGScheduler: ResultStage 7 (count at <stdin>:1
) finished in 0.028 s
17/12/06 19:19:08 INFO scheduler.DAGScheduler: Job 7 finished: count at <stdin>:
1, took 0.034992 s
23
>>> 
```

- Call the collect operation to return all data in the RDD to the Spark driver. Take note of the type of the return value; in Python will be a list of strings, and in Scala it will be an array of strings.

Note: collect returns the entire set of data. This is convenient for very small RDDs like this one, but be careful using collect for more typical large sets of data.

```
pyspark> lines = myRDD.collect()
```

- Display the contents of the collected data by looping through the collection.

```
pyspark> for line in lines: print line
```

```
>>> for line in lines: print line
...
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
```

Transform Data in an RDD

In this exercise, you will load two text files containing the names of various cell phone makes and append one to the other.

Review the two text files you will be using by viewing (without editing) the file in a separate window. The files are makes1.txt and makes2.txt in the /home/cloudera/dataset directory.

- Upload the two-text file to HDFS directory /loudacre.

```
$ hdfs dfs -put /home/cloudera/dataset/makes*.txt loudacre/
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put makes*.txt loudacre/
[cloudera@quickstart dataset]$ hdfs dfs -ls loudacre
Found 9 items
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 08:06 loudacre/accountdevice
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_zip94913
-rw-r--r--   1 cloudera cloudera       5483 2018-04-15 09:20 loudacre/devices.json
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:49 loudacre/devices_parquet
-rw-r--r--   1 cloudera cloudera        730 2018-04-18 10:40 loudacre/frostroad.txt
-rw-r--r--   1 cloudera cloudera        102 2018-04-18 10:43 loudacre/makes1.txt
-rw-r--r--   1 cloudera cloudera         95 2018-04-18 10:43 loudacre/makes2.txt
```

- In Spark, create an RDD called makes1RDD based on the loudacre/makes1.txt file.

```
pyspark> makes1RDD = sc.textFile("loudacre/makes1.txt")
```

```
>>> makes1RDD = sc.textFile("loudacre/makes1.txt")
17/12/06 19:25:36 INFO storage.MemoryStore: Block broadcast_11 stored as values
in memory (estimated size 353.1 KB, free 510.3 MB)
17/12/06 19:25:36 INFO storage.MemoryStore: Block broadcast_11_piece0 stored as
bytes in memory (estimated size 30.3 KB, free 510.3 MB)
17/12/06 19:25:36 INFO storage.BlockManagerInfo: Added broadcast_11_piece0 in me
mory on localhost:32933 (size: 30.3 KB, free: 511.0 MB)
17/12/06 19:25:36 INFO spark.SparkContext: Created broadcast 11 from textFile at
 NativeMethodAccessorImpl.java:-2
>>>
```

- Display the contents of the makes1RDD data using collect and then looping through returned collection.

```
pyspark> for make in makes1RDD.collect(): print make
```

```
17/12/06 19:26:07 INFO storage.BlockManagerInfo: Removed broadcast_1_piece0 on l
ocalhost:32933 in memory (size: 3.1 KB, free: 511.0 MB)
Sorrento
Titanic
Sorrento
Titanic
MeeToo
MeeToo
MeeToo
Titanic
MeeToo
MeeToo
Titanic
Sorrento
Sorrento
```

- Repeat the previous steps to create and display an RDD called makes2RDD based on the second file, loudacre/makes2.txt.

```
pyspark> makes2RDD = sc.textFile("loudacre/makes2.txt")
```

```
>>> makes2RDD = sc.textFile("loudacre/makes2.txt")
17/12/06 19:27:27 INFO storage.MemoryStore: Block broadcast_13 stored as values
in memory (estimated size 353.1 KB, free 510.0 MB)
17/12/06 19:27:27 INFO storage.MemoryStore: Block broadcast_13_piece0 stored as
bytes in memory (estimated size 30.3 KB, free 510.0 MB)
17/12/06 19:27:27 INFO storage.BlockManagerInfo: Added broadcast_13_piece0 in me
mory on localhost:32933 (size: 30.3 KB, free: 511.0 MB)
17/12/06 19:27:27 INFO spark.SparkContext: Created broadcast 13 from textFile at
 NativeMethodAccessorImpl.java:-2
>>>
```

```
pyspark> for make in makes2RDD.collect(): print make
```

```
17/12/06 19:27:51 INFO scheduler.DAGScheduler: Job 10 finished: collect at <stdi
n>:1, took 0.018616 s
Titanic
MeeToo
MeeToo
iFruit
iFruit
Titanic
MeeToo
iFruit
Titanic
Ronin
Titanic
Titanic
Titanic
>>> 
```

- Create a new RDD by appending the second RDD to the first using the union transformation.

```
pyspark> allMakesRDD = makes1RDD.union(makes2RDD)
```

```
>>> allMakesRDD = makes1RDD.union(makes2RDD)
>>> 
```

- Collect and display the contents of the new allMakesRDD RDD.

```
pyspark> for allmake in allMakesRDD.collect(): print allmake
```

```
17/12/06 19:31:27 INFO scheduler.DAGScheduler: Job 11 finished: collect at <stdi
n>:1, took 0.030173 s
Sorrento
Titanic
Sorrento
Titanic
MeeToo
MeeToo
MeeToo
Titanic
MeeToo
MeeToo
Titanic
Sorrento
Sorrento
Titanic
MeeToo
MeeToo
iFruit
iFruit
Titanic
MeeToo
iFruit
```

- Use the distinct transformation to remove duplicates from allMakesRDD. Collect and display the contents to confirm that duplicate elements were removed.

```
pyspark> newRDD = allMakesRDD.distinct()
```

```
>>> newRDD = allMakesRDD.distinct()
>>>
```

```
pyspark> for make in newRDD.collect(): print make
```

```
17/12/06 19:35:21 INFO scheduler.DAGScheduler: Job 12 finished: collect at <stdi
n>:1, took 0.162197 s
Sorrento
MeeToo
Titanic
Ronin
iFruit
>>>
```

Transforming Data Using RDDs

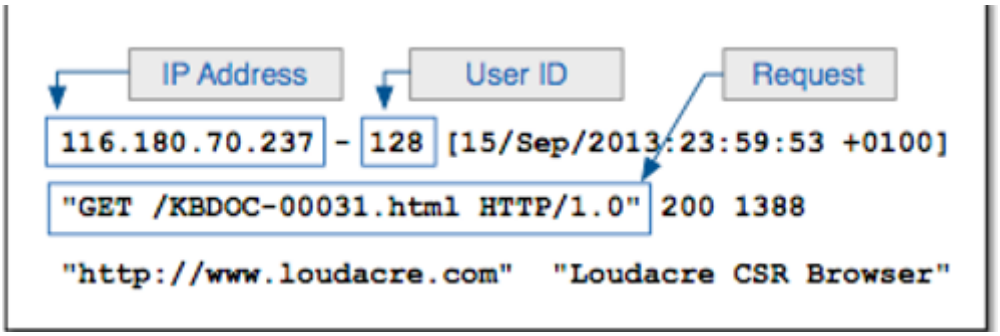
Data files (local) /home/cloudera/dataset/weblogs/

/home/cloudera/dataset/devicestatus.txt

In this exercise, transform data in RDDs. You will start reading a simple text file into a Resilient Distributed Dataset (RDD). Then you create an RDD based on Loudacre's website log data and practice transforming the data.

Explore the Loudacre Web Log Files

Format



- Copy the weblogs directory from the gateway filesystem to the loudacre HDFS directory.

```
$ hdfs dfs -put weblogs loudacre
```

- Create an RDD from the uploaded web logs data files in the loudacre/weblogs/ directory in HDFS.

```
pyspark> logsRDD = sc.textFile("loudacre/weblogs/")
```

```
>>> logsRDD = sc.textFile("loudacre/weblogs/")
17/12/06 20:00:02 INFO storage.MemoryStore: Block broadcast_18 stored as values
in memory (estimated size 353.1 KB, free 509.7 MB)
17/12/06 20:00:02 INFO storage.MemoryStore: Block broadcast_18_piece0 stored as
bytes in memory (estimated size 30.3 KB, free 509.6 MB)
17/12/06 20:00:02 INFO storage.BlockManagerInfo: Added broadcast_18_piece0 in me
mory on localhost:32933 (size: 30.3 KB, free: 511.0 MB)
17/12/06 20:00:02 INFO spark.SparkContext: Created broadcast 18 from textFile at
 NativeMethodAccessorImpl.java:-2
>>>
```

- Create an RDD containing only those lines that are requests for JPG files. Use the filter operation with a transformation function that takes a string RDD element and returns a boolean value.

```
pyspark> jpglogsRDD = logsRDD.filter(lambda line: ".jpg" in
line)
```

```
>>> jpglogsRDD = logsRDD.filter(lambda line: ".jpg" in line)
>>>
```

- Use take to return the first five lines of the data in jpglogsRDD. The return value is a list of strings (in Python)

```
pyspark> jpgLines = jpglogsRDD.take(5)
```

- Loop through and display the strings returned by take.

```
pyspark> for line in jpgLines: print line
```

```
>>> for line in jpgLines: print line
...
217.150.149.167 - 4712 [15/Sep/2013:23:56:06 +0100] "GET /ronin_s4.jpg HTTP/1.0"
 200 5552 "http://www.loudacre.com"  "Loudacre Mobile Browser MeeToo 1.0"
104.184.210.93 - 28402 [15/Sep/2013:23:42:53 +0100] "GET /titanic_2200.jpg HTTP/
1.0" 200 19466 "http://www.loudacre.com"  "Loudacre Mobile Browser MeeToo 2.0"
37.91.137.134 - 36171 [15/Sep/2013:23:39:33 +0100] "GET /ronin_novelty_note_3.jp
g HTTP/1.0" 200 7432 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit
3"
177.43.223.203 - 90653 [15/Sep/2013:23:31:17 +0100] "GET /ifruit_3.jpg HTTP/1.0"
 200 19578 "http://www.loudacre.com"  "Loudacre Mobile Browser Sorrento F31L"
19.250.65.76 - 44388 [15/Sep/2013:23:31:10 +0100] "GET /sorrento_f24l.jpg HTTP/1
.0" 200 5730 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit 3A"
>>>
```

- Now try using the map transformation to define a new RDD. Start with a simple map function that returns the length of each line in the log file. This results in an RDD of integers.

```
pyspark> lineLengthsRDD = logsRDD.map(lambda line: len(line))
```

```
>>> lineLengthsRDD = logsRDD.map(lambda line: len(line))
>>>
```

- Loop through and display the first five elements (integers) in the RDD.

```
pyspark> lineLengthsRDD.take(5)
```

```
17/12/06 20:04:54 INFO scheduler.DAGScheduler: Job 14 finished: runJob at Python
RDD.scala:393, took 0.026075 s
[151, 143, 154, 147, 160]
>>>
```

- Calculating line lengths is not very useful. Instead, try mapping each string in logsRDD by splitting the strings based on spaces. The result will be an RDD in which each element is a list of strings (in Python) . Each string represents a "field" in the web log line.

```
pyspark> lineFieldsRDD = logsRDD.map(lambda line: line.split('
'))
```

```
>>> lineFieldsRDD = logsRDD.map(lambda line: line.split(' '))
>>>
```

- Return the first five elements of lineFieldsRDD. The result will be a list of lists of strings (in Python)

```
pyspark> lineFields = lineFieldsRDD.take(5)
```

- Display the contents of the return from take. Unlike in examples above, which returned collections of simple values (strings and ints), this time you have a set of

compound values (arrays or lists containing strings). Therefore, to display them properly, you will need to loop through the arrays/lists in lineFields, and thenloop through each string in the array/list. (To make it easier to read the output, use ------- to separate each set of field values.)

```
pyspark> for fields in lineFields: print "-------"
           for field in fields: print field
```

```
>>> for fields in lineFields: print "-------"
...
-------
-------
-------
-------
-------
>>> for field in fields: print field
...
129.133.56.105
-
2489
[15/Sep/2013:23:58:34
+0100]
"GET
/KBDOC-00033.html
HTTP/1.0"
200
10590
"http://www.loudacre.com"

"Loudacre
Mobile
Browser
Sorrento
F00L"
>>>
```

- Now that you know how map works, create a new RDD containing just the IP addresses from each line in the log file. (The IP address is the first space-delimited field in each line.)

```
pyspark> ipsRDD = logsRDD.map(lambda line: line.split(' ')[0])
pyspark> for ip in ipsRDD.take(10): print ip
```

```
>>> ipsRDD = logsRDD.map(lambda line: line.split(' ')[0])
>>> for ip in ipsRDD.take(10): print ip
...
17/12/06 20:16:46 INFO spark.SparkContext: Starting job: runJob at PythonRDD.sca
la:393


17/12/06 20:16:46 INFO scheduler.DAGScheduler: Job 16 finished: runJob at Python
RDD.scala:393, took 0.099434 s
3.94.78.5
3.94.78.5
19.38.140.62
19.38.140.62
129.133.56.105
129.133.56.105
217.150.149.167
217.150.149.167
217.150.149.167
217.150.149.167
>>>
```

- Finally, save the list of IP addresses as a text file:

```
pyspark> ipsRDD.saveAsTextFile("loudacre/iplist")
```

```
>>> ipsRDD.saveAsTextFile("loudacre/iplist")
17/12/06 20:17:51 INFO output.FileOutputCommitter: File Output Committer Algorit
hm version is 1
17/12/06 20:17:51 INFO output.FileOutputCommitter: FileOutputCommitter skip clea
```

Note: If you re-run this command, you will not be able to save to the same directory because it already exists. Be sure to first delete the directory using either the hdfs command (in a separate terminal window) or the Hue file browser.

- In a seperate terminal window or the Hue file browser, list the contents of the loudacre/iplist folder. Review the contents of one of the files to confirm that they were created correctly.

```
[cloudera@quickstart ~]$ hdfs dfs -ls loudacre/iplist
Found 183 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-19 08:08 loudacre/iplist/_SUCCESS
-rw-r--r--   1 cloudera cloudera      49265 2018-04-19 08:08 loudacre/iplist/part-00000
-rw-r--r--   1 cloudera cloudera      45854 2018-04-19 08:08 loudacre/iplist/part-00001
-rw-r--r--   1 cloudera cloudera      50031 2018-04-19 08:08 loudacre/iplist/part-00002
-rw-r--r--   1 cloudera cloudera      45898 2018-04-19 08:08 loudacre/iplist/part-00003
-rw-r--r--   1 cloudera cloudera      48070 2018-04-19 08:08 loudacre/iplist/part-00004
-rw-r--r--   1 cloudera cloudera      46430 2018-04-19 08:08 loudacre/iplist/part-00005
-rw-r--r--   1 cloudera cloudera      46177 2018-04-19 08:08 loudacre/iplist/part-00006
-rw-r--r--   1 cloudera cloudera      50720 2018-04-19 08:08 loudacre/iplist/part-00007
-rw-r--r--   1 cloudera cloudera      47314 2018-04-19 08:08 loudacre/iplist/part-00008
-rw-r--r--   1 cloudera cloudera      46282 2018-04-19 08:08 loudacre/iplist/part-00009
-rw-r--r--   1 cloudera cloudera      45998 2018-04-19 08:08 loudacre/iplist/part-00010
-rw-r--r--   1 cloudera cloudera      49261 2018-04-19 08:08 loudacre/iplist/part-00011
-rw-r--r--   1 cloudera cloudera      47918 2018-04-19 08:08 loudacre/iplist/part-00012
-rw-r--r--   1 cloudera cloudera      46580 2018-04-19 08:08 loudacre/iplist/part-00013
-rw-r--r--   1 cloudera cloudera      45576 2018-04-19 08:08 loudacre/iplist/part-00014
-rw-r--r--   1 cloudera cloudera      49698 2018-04-19 08:08 loudacre/iplist/part-00015
-rw-r--r--   1 cloudera cloudera      45627 2018-04-19 08:08 loudacre/iplist/part-00016
-rw-r--r--   1 cloudera cloudera      49712 2018-04-19 08:08 loudacre/iplist/part-00017
```

Map weblog entries to IP address/user ID pairs

- Use RDD transformations to create a dataset consisting of the IP address and corresponding user ID for each request for an HTML file. (Filter for files with the .html extension; disregard requests for other file types.) The user ID is the third field in each log file line. Save the data into a comma-separated text file in the directory loudacre/userips_csv. Make sure the data is saved in the form of comma-separated strings:

```
165.32.101.206,8 100.219.90.44,102 182.4.148.56,173
246.241.6.175,45395 175.223.172.207,4115 …
```

```
userIPRDD=logsRDD.filter(lambda line: ".html" in
line).map(lambda line: line.split()[0]+","+line.split()[2])

userIPRDD.saveAsTextFile("loudacre/userips_csv")
```

```
>>> userIPRDD=logsRDD.filter(lambda line: ".html" in line).map(lambda line: line
.split()[0]+","+line.split()[2])
>>>
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls loudacre/userips_csv
Found 183 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-19 08:10 loudacre/userips_csv/_SUCCESS
-rw-r--r--   1 cloudera cloudera      29119 2018-04-19 08:09 loudacre/userips_csv/part-00000
-rw-r--r--   1 cloudera cloudera      27367 2018-04-19 08:09 loudacre/userips_csv/part-00001
-rw-r--r--   1 cloudera cloudera      29664 2018-04-19 08:09 loudacre/userips_csv/part-00002
-rw-r--r--   1 cloudera cloudera      27370 2018-04-19 08:09 loudacre/userips_csv/part-00003
-rw-r--r--   1 cloudera cloudera      28218 2018-04-19 08:09 loudacre/userips_csv/part-00004
-rw-r--r--   1 cloudera cloudera      27663 2018-04-19 08:09 loudacre/userips_csv/part-00005
-rw-r--r--   1 cloudera cloudera      27840 2018-04-19 08:09 loudacre/userips_csv/part-00006
-rw-r--r--   1 cloudera cloudera      30153 2018-04-19 08:09 loudacre/userips_csv/part-00007
-rw-r--r--   1 cloudera cloudera      28214 2018-04-19 08:09 loudacre/userips_csv/part-00008
-rw-r--r--   1 cloudera cloudera      27314 2018-04-19 08:09 loudacre/userips_csv/part-00009
-rw-r--r--   1 cloudera cloudera      27369 2018-04-19 08:09 loudacre/userips_csv/part-00010
-rw-r--r--   1 cloudera cloudera      29241 2018-04-19 08:09 loudacre/userips_csv/part-00011
-rw-r--r--   1 cloudera cloudera      28724 2018-04-19 08:09 loudacre/userips_csv/part-00012
-rw-r--r--   1 cloudera cloudera      27831 2018-04-19 08:09 loudacre/userips_csv/part-00013
-rw-r--r--   1 cloudera cloudera      27431 2018-04-19 08:09 loudacre/userips_csv/part-00014
-rw-r--r--   1 cloudera cloudera      29657 2018-04-19 08:09 loudacre/userips_csv/part-00015
-rw-r--r--   1 cloudera cloudera      27349 2018-04-19 08:09 loudacre/userips_csv/part-00016
-rw-r--r--   1 cloudera cloudera      29634 2018-04-19 08:09 loudacre/userips_csv/part-00017
```

Joining Data Using Pair RDDs

```
Data files (HDFS) loudacre/weblogs
/user/hive/warehouse/accounts
```

Explore Web Log Files Create a pair RDD based on data in the weblogs data files and use the pair RDD to explore the data.

Tip: In this exercise, you will be reducing and joining large datasets, which can take a lot of time and may result in memory errors resulting from the limited resources available in the course exercise environment. Perform these exercises with a subset of the the web log files by using a wildcard: textFile("/loudacre/weblogs/ *2.log") includes only filenames ending with 2.log.

```
# Create an RDD based on a subset of weblogs (those ending in
digit 2)

logsRDD = sc.textFile("loudacre/weblogs/*2.log")
```

```
>>> logsRDD = sc.textFile("loudacre/weblogs/*2.log")
17/12/08 07:48:41 INFO storage.MemoryStore: Block broadcast_11 stored as values
in memory (estimated size 91.0 KB, free 510.7 MB)
17/12/08 07:48:41 INFO storage.MemoryStore: Block broadcast_11_piece0 stored as
bytes in memory (estimated size 30.3 KB, free 510.7 MB)
17/12/08 07:48:41 INFO storage.BlockManagerInfo: Added broadcast_11_piece0 in me
mory on localhost:40520 (size: 30.3 KB, free: 511.0 MB)
17/12/08 07:48:41 INFO spark.SparkContext: Created broadcast 11 from textFile at
 NativeMethodAccessorImpl.java:-2
>>>
```

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

  a. Use map to create a pair RDD with the user ID as the key and the integer 1 as the value. (The user ID is the third field in each line.)

  b. Use reduceByKey to sum the values for each user ID.

```
# map each request (line) to a pair (userid, 1), then sum the
values

userReqsRDD = logsRDD.map(lambda line: line.split(' '))
.map(lambda words: (words[2],1)) .reduceByKey(lambda
count1,count2: count1 + count2)
```

```
>>> userReqsRDD = logsRDD.map(lambda line: line.split(' ')) .map(lambda words: (
words[2],1)).reduceByKey(lambda count1,count2: count1 + count2)
17/12/08 07:49:39 INFO mapred.FileInputFormat: Total input paths to process : 18
>>>
```

- Use countByKey to determine how many users visited the site for each frequency. That is, how many users visited once, twice, three times, and so on.
  a. Use map to reverse the key and value, like this:
     (5,userid) (7,userid) (2,userid) …

  b. Use the countByKey action to return a map of frequency: user-count pairs.

```
# Show the count frequencies

freqCountMap = userReqsRDD.map(lambda (userid,freq):
(freq,userid)).countByKey()

print freqCountMap
```

```
>>> freqCountMap = userReqsRDD.map(lambda (userid,freq): (freq,userid)).countByK
ey()
17/12/08 07:50:21 INFO spark.SparkContext: Starting job: countByKey at <stdin>:1
17/12/08 07:50:21 INFO scheduler.DAGScheduler: Registering RDD 24 (reduceByKey a
t <stdin>:1)
17/12/08 07:50:21 INFO scheduler.DAGScheduler: Got job 10 (countByKey at <stdin>
:1) with 18 output partitions
```

```
>>> print freqCountMap
defaultdict(<type 'int'>, {128: 9, 2: 7239, 3: 36, 4: 4155, 5: 26, 6: 2162, 7: 1
4, 8: 1409, 9: 14, 10: 878, 11: 12, 12: 549, 13: 7, 14: 308, 15: 8, 16: 155, 17:
 4, 146: 8, 19: 2, 20: 41, 21: 1, 22: 17, 150: 11, 152: 11, 132: 12, 154: 5, 27:
 1, 156: 5, 158: 6, 160: 8, 162: 5, 164: 3, 134: 9, 166: 3, 168: 4, 170: 3, 172:
 6, 174: 2, 24: 6, 176: 2, 136: 11, 178: 1, 188: 1, 138: 6, 190: 1, 130: 10, 140
: 14, 142: 8, 86: 1, 144: 7, 100: 1, 104: 1, 106: 1, 18: 76, 110: 4, 112: 1, 116
: 1, 118: 4, 120: 5, 148: 2, 122: 4, 124: 7, 126: 5})
>>>
```

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

Hint: Map to (userid,ipaddress) and then use groupByKey.
(userid,20.1.34.55) (userid,245.33.1.1)
(userid,65.50.196.141) …

groupByKey doesn't return an array (as may be implied by the use of square brackets here) but an array-like iterable type.

```
# Group IPs by user ID

userIPsRDD = logsRDD .map(lambda line:
line.split()).map(lambda words: (words[2],words[0]))
.groupByKey()
```

```
>>> userIPsRDD = logsRDD .map(lambda line: line.split()).map(lambda words: (word
s[2],words[0])) .groupByKey()
>>>
```

```
# print out the first 10 user ids, and their IP list

for (userid,ips) in userIPsRDD.take(10):

   print userid, ":"

   for ip in ips: print "\t",ip
```

```
>>> for (userid,ips) in userIPsRDD.take(10):
...    print userid, ":"
...    for ip in ips: print "\t",ip
...


3922 :
        195.220.211.104
        195.220.211.104
        138.217.174.182
        138.217.174.182
        138.217.174.182
        138.217.174.182
104959 :
        183.123.205.115
        183.123.205.115
90396 :
        191.120.254.24
        191.120.254.24
62733 :
        93.120.232.94
        93.120.232.94
        92.75.142.64
        92.75.142.64
30390 :
        235.242.157.100
        235.242.157.100
```

Join Web Log Data with Account Data Review the accounts data located in /user/hive/warehouse/accounts, which contains the data in the Hive accounts table. The first field in each line is the user ID, which corresponds to the user ID in the web server logs. The other fields include account details such as creation date, first and last name, and so on.

- Join the accounts data with the weblog data to produce a dataset keyed by user ID which contains the user account information and the number of website hits for that user.
  a. Create an RDD, based on the accounts data, consisting of key/value-array pairs: (userid,[values…])

```
# Map account data to (userid,[values....])

accountsData = "/user/cloudera/input/account"

accountsRDD = sc.textFile(accountsData).map(lambda s:
s.split(',')).map(lambda account: (account[0],account))
```

```
>>> accountsData = "/user/technocrafty/input/account"
>>>
```

```
>>> accountsRDD = sc.textFile(accountsData).map(lambda s: s.split(',')).map(lamb
da account: (account[0],account))
17/12/08 08:07:18 INFO storage.MemoryStore: Block broadcast_16 stored as values
in memory (estimated size 353.1 KB, free 510.4 MB)
17/12/08 08:07:18 INFO storage.MemoryStore: Block broadcast_16_piece0 stored as
bytes in memory (estimated size 30.3 KB, free 510.4 MB)
17/12/08 08:07:18 INFO storage.BlockManagerInfo: Added broadcast_16_piece0 in me
mory on localhost:40520 (size: 30.3 KB, free: 511.0 MB)
17/12/08 08:07:18 INFO spark.SparkContext: Created broadcast 16 from textFile at
 NativeMethodAccessorImpl.java:-2
>>>
```

b. Join the pair RDD with the set of user-id/hit-count pairs calculated in the first step.

```
# Join account data with userreqs then merge hit count into
valuelist

accountHitsRDD = accountsRDD.join(userReqsRDD)
```

```
>>> accountHitsRDD = accountsRDD.join(userReqsRDD)
17/12/08 08:07:44 INFO mapred.FileInputFormat: Total input paths to process : 5
>>>
```

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

```
# Display userid, hit count, first name, last name for the
first 5 elements

for (userid,(values,count)) in accountHitsRDD.take(5) :

    print  userid, count, values[3], values[4]
```

```
>>> for (userid,(values,count)) in accountHitsRDD.take(5) :
...     print  userid, count, values[3], values[4]
...
```

Exercise7: DataFrames and Spark SQL

```
Files and Data Used in This Exercise

Data files (local) /home/cloudera/dataset/devices.json
```

In this exercise, you will use the Spark shell to work with DataFrames.You will start the Spark shell and read a simple JSON file into a DataFrame.

Read and Display a JSON File

- Open a new terminal session (not the terminal running the Spark shell).
- Use the less command or an editor to view the simple text file you will be using by viewing (without editing) the file in a text editor in a separate window (not the Spark shell). The file is located at: /home/cloudera/dataset/devices.json.

  This file contains records for each of Loudacre's supported devices. For example:
  {"devnum":1,"release_dt":"2008-10-21T00:00:00.000-07:00", "make":"Sorrento","model":"F00L","dev_type":"phone"}
- Upload the data file to the /loudacre directory in HDFS:

```
$ hdfs dfs -mkdir loudacre

$ hdfs dfs -put devices.json loudacre
```

```
[cloudera@quickstart dataset]$ hdfs dfs -mkdir loudacre
[cloudera@quickstart dataset]$ hdfs dfs -put devices.json loudacre
[cloudera@quickstart dataset]$
```

- In the Spark shell, create a new DataFrame based on the devices.json file in HDFS.

```
scala> val devDF =
sqlContext.read.json("loudacre/devices.json")
```

```
scala> val devDF = sqlContext.read.json("loudacre/devices.json")
17/12/06 21:02:57 INFO json.JSONRelation: Listing hdfs://technocrafty.hortonwork
s.com:8020/user/technocrafty/loudacre/devices.json on driver


scala> val devDF = spark.read.json("loudacre/devices.json")
18/04/18 07:34:46 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
18/04/18 07:34:46 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
devDF: org.apache.spark.sql.DataFrame = [dev_type: string, devnum: bigint ... 3
more fields]

scala>
```

- Spark has not yet read the data in the file, but it has scanned the file to infer the schema. View the schema, and note that the column names match the record field names in the JSON file.

```
scala> devDF.printSchema
```

```
scala> devDF.printSchema
root
 |-- dev_type: string (nullable = true)
 |-- devnum: long (nullable = true)
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)
 |-- release_dt: string (nullable = true)


scala>
```

- Display the data in the DataFrame using the show function. If you don't pass an argument to show, Spark will display the first 20 rows in the DataFrame. For this step, display the first five rows. Note that the data is displayed in tabular form, using the column names defined in the schema.

```
> devDF.show(5)
```

```
scala> devDF.show(5)
+--------+------+--------+-----+--------------------+
|dev_type|devnum|    make|model|          release_dt|
+--------+------+--------+-----+--------------------+
|   phone|     1|Sorrento| F00L|2008-10-21T00:00:...|
|   phone|     2| Titanic| 2100|2010-04-19T00:00:...|
|   phone|     3|  MeeToo|  3.0|2011-02-18T00:00:...|
|   phone|     4|  MeeToo|  3.1|2011-09-21T00:00:...|
|   phone|     5|  iFruit|    1|2008-10-21T00:00:...|
+--------+------+--------+-----+--------------------+
only showing top 5 rows


scala>
```

- The show and printSchema operations are actions—that is, they return a value from the distributed DataFrame to the Spark driver. Both functions display the data in a nicely formatted table. These functions are intended for interactive use in the shell, but do not allow you work with the data that is returned. Try using the 'take' action instead, which returns an array (Scala) or list (Python) of Row objects. You can display the data by iterating through the collection.

```
scala> val rows = devDF.take(5)
```

```
scala> val rows = devDF.take(5)
rows: Array[org.apache.spark.sql.Row] = Array([phone,1,Sorrento,F00L,2008-10-21T
00:00:00.000-07:00], [phone,2,Titanic,2100,2010-04-19T00:00:00.000-07:00], [phon
e,3,MeeToo,3.0,2011-02-18T00:00:00.000-08:00], [phone,4,MeeToo,3.1,2011-09-21T00
:00:00.000-07:00], [phone,5,iFruit,1,2008-10-21T00:00:00.000-07:00])

scala>
```

```
scala> rows.foreach(println)
```

```
scala> rows.foreach(println)
[phone,1,Sorrento,F00L,2008-10-21T00:00:00.000-07:00]
[phone,2,Titanic,2100,2010-04-19T00:00:00.000-07:00]
[phone,3,MeeToo,3.0,2011-02-18T00:00:00.000-08:00]
[phone,4,MeeToo,3.1,2011-09-21T00:00:00.000-07:00]
[phone,5,iFruit,1,2008-10-21T00:00:00.000-07:00]

scala>
```

Query a DataFrame

- Use the count action to return the number of items in the DataFrame.

```
> devDF.count()
```

```
scala> devDF.count()
res3: Long = 50

scala>
```

- DataFrame transformations typically return another DataFrame. Try using a select transformation to return a DataFrame with only the make and model columns, then display its schema. Note that only the selected columns are in the schema.

```
scala> val makeModelDF = devDF.select("make","model")
```

```
scala> val makeModelDF = devDF.select("make","model")
makeModelDF: org.apache.spark.sql.DataFrame = [make: string, model: string]

scala>
```

```
scala> makeModelDF.printSchema
```

```
scala> makeModelDF.printSchema
root
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)

scala>
```

- A query is a series of one or more transformations followed by an action. Spark does not execute the query until you call the action operation. Display the first 20

lines of the final DataFrame in the series using the show action.

```
scala> makeModelDF.show
```

```
scala> makeModelDF.show
+--------+--------------+
|    make|         model|
+--------+--------------+
|Sorrento|          F00L|
| Titanic|          2100|
|  MeeToo|           3.0|
|  MeeToo|           3.1|
|  iFruit|             1|
|  iFruit|             3|
|  iFruit|             2|
|  iFruit|             5|
| Titanic|          1000|
|  MeeToo|           1.0|
|Sorrento|          F21L|
|  iFruit|             4|
|Sorrento|          F23L|
| Titanic|          2200|
|   Ronin|Novelty Note 1|
| Titanic|          2500|
|   Ronin|Novelty Note 3|
|   Ronin|Novelty Note 2|
|   Ronin|Novelty Note 4|
|  iFruit|            3A|
```

- Transformations in a query can be chained together. Execute a single command to show the results of a query using select and where. The resulting DataFrame will contain only the columns devnum, make, and model, and only the rows where the make is Ronin

```
scala> devDF.select("devnum","make","model").where("make = 'Ronin'").show
```

```
scala> devDF.select("devnum","make","model").where("make = 'Ronin'").show()
17/12/06 21:07:57 INFO storage.MemoryStore: Block broadcast_15 stored as values
in memory (estimated size 353.1 KB, free 509.2 MB)
```

```
scala> devDF.select("devnum","make","model").where("make = 'Ronin'").show
+------+-----+--------------+
|devnum| make|         model|
+------+-----+--------------+
|    15|Ronin|Novelty Note 1|
|    17|Ronin|Novelty Note 3|
|    18|Ronin|Novelty Note 2|
|    19|Ronin|Novelty Note 4|
|    46|Ronin|            S4|
|    47|Ronin|            S1|
|    48|Ronin|            S3|
|    49|Ronin|            S2|
+------+-----+--------------+

scala>
```

Working with DataFrames and Schemas

Files and Data Used in This Exercise:

```
Data files (HDFS) loudacre/devices.json

Hive Tables accounts
```

In this exercise, you will work with structured account and mobile device data using DataFrames. You will practice creating and saving DataFrames using different types of data sources and inferring and defining schemas.

Create a DataFrame Based on a Hive Table

- This exercise uses a DataFrame based on the accounts Hive table. Before you start working in Spark, visit Hue in your browser, and use the Impala Query Editor to review the schema and data of the accounts table in the default database.

- Create a new DataFrame using the Hive accounts table.

```
scala> val accountsDF = spark.read.table("accounts")
```

```
scala> val accountsDF = spark.read.table("accounts")
accountsDF: org.apache.spark.sql.DataFrame = [acct_num: int, acct_create_dt: tim
estamp ... 10 more fields]

scala>
```

- Print the schema and the first few rows of the DataFrame and note that the schema and data are the same as the Hive table.

```
scala> accountsDF.printSchema
```

```
scala> accountsDF.printSchema
root
 |-- acct_num: integer (nullable = true)
 |-- acct_create_dt: timestamp (nullable = true)
 |-- acct_close_dt: timestamp (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zipcode: string (nullable = true)
 |-- phone_number: string (nullable = true)
 |-- created: timestamp (nullable = true)
 |-- modified: timestamp (nullable = true)


scala>
```

- Create a new DataFrame with rows from the accounts data where the zip code is 94913, and save the result to CSV files in the loudacre/accounts_zip94913 HDFS directory. You can do this in a single command, as shown below, or with multiple commands

```
scala> accountsDF.where("zipcode = '94913'").
write.option("header","true").
csv("loudacre/accounts_zip94913")
```

```
scala> accountsDF.where("zipcode = '94913'").write.option("header","true").csv("
loudacre/accounts_zip94913")

scala>
```

- Use Hue or the command line, to view the loudacre/accounts_zip94913 directory in HDFS and the data in one of the saved files.

```
$ hdfs dfs -ls loudacre/accounts_zip94913
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls loudacre/accounts_zip94913
Found 6 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_z
ip94913/_SUCCESS
-rw-r--r--   1 cloudera cloudera        967 2018-04-18 07:41 loudacre/accounts_z
ip94913/part-00000-7193a9e6-53eb-4ead-8562-15d252a3ff87-c000.csv
-rw-r--r--   1 cloudera cloudera       1126 2018-04-18 07:41 loudacre/accounts_z
ip94913/part-00001-7193a9e6-53eb-4ead-8562-15d252a3ff87-c000.csv
-rw-r--r--   1 cloudera cloudera       1409 2018-04-18 07:41 loudacre/accounts_z
ip94913/part-00002-7193a9e6-53eb-4ead-8562-15d252a3ff87-c000.csv
-rw-r--r--   1 cloudera cloudera       1664 2018-04-18 07:41 loudacre/accounts_z
ip94913/part-00003-7193a9e6-53eb-4ead-8562-15d252a3ff87-c000.csv
-rw-r--r--   1 cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_z
ip94913/part-00004-7193a9e6-53eb-4ead-8562-15d252a3ff87-c000.csv
[cloudera@quickstart ~]$
```

Confirm that the CSV file includes a header line, and that only records for the selected zip code are included.

```
val test1DF =
spark.read.option("header","true").csv("loudacre/accounts_zip9
4913")

val test2DF =
spark.read.option("header","true").option("inferSchema","true"
).csv("loudacre/accounts_zip94913")



test1DF.printSchema

test2DF.printSchema
```

```
scala> val test1DF = spark.read.option("header","true").csv("loudacre/accounts_z
ip94913")
test1DF: org.apache.spark.sql.DataFrame = [acct_num: string, acct_create_dt: str
ing ... 10 more fields]

scala> val test2DF = spark.read.option("header","true").option("inferSchema","tr
ue").csv("loudacre/accounts_zip94913")
test2DF: org.apache.spark.sql.DataFrame = [acct_num: int, acct_create_dt: timest
amp ... 10 more fields]

scala>
```

```
scala> test1DF.printSchema
root
 |-- acct_num: string (nullable = true)
 |-- acct_create_dt: string (nullable = true)
 |-- acct_close_dt: string (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zipcode: string (nullable = true)
 |-- phone_number: string (nullable = true)
 |-- created: string (nullable = true)
 |-- modified: string (nullable = true)


scala>
```

```
scala> test2DF.printSchema
root
 |-- acct_num: integer (nullable = true)
 |-- acct_create_dt: timestamp (nullable = true)
 |-- acct_close_dt: timestamp (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zipcode: integer (nullable = true)
 |-- phone_number: long (nullable = true)
 |-- created: timestamp (nullable = true)
 |-- modified: timestamp (nullable = true)


scala>
```

Define a Schema for a DataFrame

- If you have not done so yet, review the data in the HDFS file loudacre/devices.json.
- Create a new DataFrame based on the devices.json file. (This command could take several seconds while it infers the schema.)

```
scala> val devDF = spark.read.json("loudacre/devices.json")
```

```
scala> val devDF = spark.read.json("loudacre/devices.json")
18/04/18 07:45:43 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
18/04/18 07:45:43 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
devDF: org.apache.spark.sql.DataFrame = [dev_type: string, devnum: bigint ... 3
more fields]

scala>
```

- View the schema of the devDF DataFrame. Note the column names and types that Spark inferred from the JSON file. Note that the release_dt column is of type string, whereas the data in the column actually represents a timestamp.

- Define a schema that correctly specifies the column types for this DataFrame. Start by importing the package with the definitions of necessary classes and types.

```
scala> import org.apache.spark.sql.types._
```

```
scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala>
```

- Next, create a collection of StructField objects, which represent column definitions. The release_dt column should be a timestamp.

```
scala> val devColumns = List(

StructField("devnum",LongType),
StructField("make",StringType),
StructField("model",StringType),
StructField("release_dt",TimestampType),
StructField("dev_type",StringType))
```

```
scala> val devColumns = List(
     | StructField("devnum",LongType),
     | StructField("make",StringType),
     | StructField("model",StringType),
     | StructField("release_dt",TimestampType),
     | StructField("dev_type",StringType))
devColumns: List[org.apache.spark.sql.types.StructField] = List(StructField(devn
um,LongType,true), StructField(make,StringType,true), StructField(model,StringTy
pe,true), StructField(release_dt,TimestampType,true), StructField(dev_type,Strin
gType,true))

scala>
```

- Create a schema (a StructType object) using the column definition list.

```
scala> val devSchema = StructType(devColumns)
```

```
scala> val devSchema = StructType(devColumns)
devSchema: org.apache.spark.sql.types.StructType = StructType(StructField(devnum
,LongType,true), StructField(make,StringType,true), StructField(model,StringType
,true), StructField(release_dt,TimestampType,true), StructField(dev_type,StringT
ype,true))

scala>
```

- Recreate the devDF DataFrame, this time using the new schema.

```
scala> val devDF = spark.read.
schema(devSchema).json("loudacre/devices.json")
```

```
scala> val devDF = spark.read.schema(devSchema).json("loudacre/devices.json")
18/04/18 07:48:50 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
devDF: org.apache.spark.sql.DataFrame = [devnum: bigint, make: string ... 3 more
 fields]

scala>
```

- View the schema and data of the new DataFrame, and confirm that the release_dt column type is now timestamp.

```
devDF.printSchema
```

```
scala> devDF.printSchema
root
 |-- devnum: long (nullable = true)
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)
 |-- release_dt: timestamp (nullable = true)
 |-- dev_type: string (nullable = true)

scala>
```

- Now that the device data uses the correct schema, write the data in Parquet format, which automatically embeds the schema. Save the Parquet data files into an HDFS directory called loudacre/devices_parquet.

```
devDF.write.parquet("loudacre/devices_parquet")
```

```
scala> devDF.write.parquet("loudacre/devices_parquet")

scala>
```

Analyzing Data with DataFrame Queries

Files and Data Used in This Exercise:

```
Data files (local) /home/cloudera/dataset/accountdevice Data
files (HDFS) loudacre/devices.json

Hive Tables accounts
```

In this exercise, you will analyze account and mobile device data using DataFrame queries. First, you will practice using column expressions in queries. You will analyze data in DataFrames by grouping and aggregating data, and by joining two DataFrames. Then you will query multiple sets of data to find out how many of each mobile device model is used in active accounts.

Query DataFrames Using Column Expressions

- Create a new DataFrame called accountsDF based on the Hive accounts table.

```
// Create a DataFrame based on the Hive accounts table

val accountsDF = spark.read.table("accounts")
```

```
scala> devDF.write.parquet("loudacre/devices_parquet")

scala> val accountsDF = spark.read.table("accounts")
accountsDF: org.apache.spark.sql.DataFrame = [acct_num: int, acct_create_dt: tim
estamp ... 10 more fields]

scala>
```

- Try a simple query with select, using both column reference syntaxes.

```
scala> accountsDF.select(accountsDF("first_name")).show
```

```
scala> accountsDF.select(accountsDF("first_name")).show
+----------+
|first_name|
+----------+
|    Donald|
|     Donna|
|    Dorthy|
|     Leila|
|     Anita|
|    Stevie|
|     David|
|   Dorothy|
|      Kara|
|     Diane|
|    Robert|
|    Marcia|
|    Andres|
|       Ann|
|    Joseph|
|     Sarah|
```

```
scala> accountsDF.select($"first_name").show
```

```
scala> accountsDF.select($"first_name").show
+----------+
|first_name|
+----------+
|    Donald|
|     Donna|
|    Dorthy|
|     Leila|
|     Anita|
|    Stevie|
|     David|
|   Dorothy|
|      Kara|
|     Diane|
|    Robert|
|    Marcia|
|    Andres|
|       Ann|
|    Joseph|
|     Sarah|
|      Lucy|
|    Roland|
|     Leona|
```

- To explore column expressions, create a column object to work with, based on the first_name column in the accountsDF DataFrame.

```
scala> val fnCol = accountsDF("first_name")
```

```
scala> val fnCol = accountsDF("first_name")
fnCol: org.apache.spark.sql.Column = first_name

scala>
```

- Note that the object type is Column. To see available methods and attributes, use tab completion—that is, enter fnCol. followed by TAB.

- New Column objects are created when you perform operations on existing columns. Create a new Column object based on a column expression that identifies users whose first name is Lucy using the equality operator on the fnCol object you created above.

```
scala> val lucyCol = (fnCol === "Lucy")
```

```
scala> val lucyCol = (fnCol === "Lucy")
lucyCol: org.apache.spark.sql.Column = (first_name = Lucy)

scala>
```

- Use the lucyCol column expression in a select statement. Because lucyCol is based on a boolean expression, the column values will be true or false depending on the value of the first_name column. Confirm that users named Lucy are identified with the value true.

```
scala> accountsDF.
select($"first_name",$"last_name",lucyCol).show
```

```
scala> accountsDF.  select($"first_name",$"last_name",lucyCol).show
+----------+---------+-------------------+
|first_name|last_name|(first_name = Lucy)|
+----------+---------+-------------------+
|    Donald|   Becton|              false|
|     Donna|    Jones|              false|
|    Dorthy| Chalmers|              false|
|     Leila|  Spencer|              false|
|     Anita| Laughlin|              false|
|    Stevie|   Bridge|              false|
|     David|   Eggers|              false|
|   Dorothy|  Koopman|              false|
|      Kara|     Kohl|              false|
|     Diane|   Nelson|              false|
|    Robert|   Fisher|              false|
|    Marcia|  Roberts|              false|
|    Andres|    Cruse|              false|
|       Ann|    Moore|              false|
|    Joseph|   Lackey|              false|
|     Sarah|   Duvall|              false|
|      Lucy|   Corley|               true|
|    Roland| Crawford|              false|
|     Leona|     Bray|              false|
```

- The where operation requires a boolean-based column expression. Use the lucyCol column expression in a where transformation and view the data in the resulting

DataFrame. Confirm that only users named Lucy are in the data.

```
> accountsDF.where(lucyCol).show(5)
```

```
scala> accountsDF.where(lucyCol).show(5)
+--------+------------------+------------------+----------+---------+---------
-----------+----------+-----+-------+------------+------------------+----------
---------+
|acct_num|     acct_create_dt|      acct_close_dt|first_name|last_name|
    address|      city|state|zipcode|phone_number|           created|
 modified|
+--------+------------------+------------------+----------+---------+---------
-----------+----------+-----+-------+------------+------------------+----------
---------+
|      17|2008-12-27 23:31:40|              null|      Lucy|   Corley|   4834 B
rown Street|Santa Rosa|   CA|  94980|  7076068290|2014-03-18 13:29:47|2014-03-18
 13:29:47|
|    1762|2009-03-19 23:43:00|              null|      Lucy|    Davis|2195 Rive
rside Drive|Sacramento|   CA|  95620|  9169959437|2014-03-18 13:29:50|2014-03-18
 13:29:50|
|    6551|2010-09-04 19:28:22|2014-02-14 19:19:16|      Lucy|  Casiano|2821 Wood
 Duck Drive|  Alhambra|   CA|  91810|  6261078791|2014-03-18 13:29:59|2014-03-18
 13:29:59|
|    6978|2010-09-22 07:13:16|              null|      Lucy|      Lee|4781 Shad
owmar Drive|   Salinas|   CA|  93961|  8315367970|2014-03-18 13:30:00|2014-03-18
 13:30:00|
```

- Column expressions do not need to be assigned to a variable. Try the same query without using the lucyCol variable.

```
scala> accountsDF.where(fnCol === "Lucy").show(5)
```


```
scala> accountsDF.where(fnCol === "Lucy").show(5)
+--------+------------------+------------------+----------+---------+---------
-----------+----------+-----+-------+------------+------------------+----------
---------+
|acct_num|     acct_create_dt|      acct_close_dt|first_name|last_name|
    address|      city|state|zipcode|phone_number|           created|
 modified|
+--------+------------------+------------------+----------+---------+---------
-----------+----------+-----+-------+------------+------------------+----------
---------+
|      17|2008-12-27 23:31:40|              null|      Lucy|   Corley|   4834 B
rown Street|Santa Rosa|   CA|  94980|  7076068290|2014-03-18 13:29:47|2014-03-18
 13:29:47|
|    1762|2009-03-19 23:43:00|              null|      Lucy|    Davis|2195 Rive
rside Drive|Sacramento|   CA|  95620|  9169959437|2014-03-18 13:29:50|2014-03-18
 13:29:50|
|    6551|2010-09-04 19:28:22|2014-02-14 19:19:16|      Lucy|  Casiano|2821 Wood
 Duck Drive|  Alhambra|   CA|  91810|  6261078791|2014-03-18 13:29:59|2014-03-18
 13:29:59|
|    6978|2010-09-22 07:13:16|              null|      Lucy|      Lee|4781 Shad
owmar Drive|   Salinas|   CA|  93961|  8315367970|2014-03-18 13:30:00|2014-03-18
 13:30:00|
```

- Column expressions are not limited to where operations like those above. They can be used in any transformation for which a simple column could be used, such as a select. Try selecting the city and state columns, and the first three characters of the phone_number column (in the U.S., the first three digits of a phone number are known as the area code). Use the substr operator on the phone_number column to extract the area code.

```
scala> accountsDF.select($"city", $"state",
$"phone_number".substr(1,3)).show(5)
```

```
scala> accountsDF.select($"city", $"state",$"phone_number".substr(1,3)).show(5)
+-------------+-----+-----------------------------+
|         city|state|substring(phone_number, 1, 3)|
+-------------+-----+-----------------------------+
|      Oakland|   CA|                          510|
|San Francisco|   CA|                          415|
|    San Mateo|   CA|                          650|
|    San Mateo|   CA|                          650|
|     Richmond|   CA|                          510|
+-------------+-----+-----------------------------+
only showing top 5 rows
```

- Notice that in the last step, the values returned by the query were correct, but the column name was substring(phone_number, 1, 3), which is long and hard to work with. Repeat the same query, using the alias operator to rename that column as area_code.

```
scala> accountsDF.select($"city", $"state",
$"phone_number".substr(1,3).alias("area_code")).show(5)
```

```
scala> accountsDF.select($"city", $"state",$"phone_number".substr(1,3).alias("ar
ea_code")).show(5)
+-------------+-----+---------+
|         city|state|area_code|
+-------------+-----+---------+
|      Oakland|   CA|      510|
|San Francisco|   CA|      415|
|    San Mateo|   CA|      650|
|    San Mateo|   CA|      650|
|     Richmond|   CA|      510|
+-------------+-----+---------+
only showing top 5 rows
```

- Perform a query that results in a DataFrame with just first_name and last_name columns, and only includes users whose first and last names both begin with the same two letters. (For example, the user Robert Roget would be included, because both his first and last names begin with "Ro".)

```
scala> accountsDF.select($"first_name",
$"last_name").where($"first_name".substr(1,2) ===
$"last_name".substr(1,2)).show(5)
```

```
scala> accountsDF.select($"first_name", $"last_name").where($"first_name".substr
(1,2) === $"last_name".substr(1,2)).show(5)
+----------+---------+
|first_name|last_name|
+----------+---------+
|   Johnnie|    Jones|
|    Robert|   Roller|
|   Michael|  Minnick|
| Rosemarie|Robertson|
|     Keith|   Kemble|
+----------+---------+
only showing top 5 rows
```

Group and Count Data by Name

- Query the accountsDF DataFrame using groupBy with count to find out the total number people sharing each last name. (Note that the count aggregation transformation returns a DataFrame, unlike the count DataFrame action, which returns a single value to the driver.)

```
scala> accountsDF.groupBy("last_name").count.show(5)
```

```
scala> accountsDF.groupBy("last_name").count.show(5)
+---------+-----+
|last_name|count|
+---------+-----+
| Francois|    6|
|   Bohner|    3|
|    Tyler|   50|
|   Maddox|   60|
| Abramson|    3|
+---------+-----+
only showing top 5 rows
```

- You can also group by multiple columns. Query accountsDF again, this time counting the number of people who share the same last and first name.

```
scala> accountsDF.
groupBy("last_name","first_name").count.show(5)
```

```
scala> accountsDF.groupBy("last_name","first_name").count.show(5)
+---------+----------+-----+
|last_name|first_name|count|
+---------+----------+-----+
| Williams|   Lillian|    6|
|Robertson| Rosemarie|    3|
|     Bell|    Joanna|    3|
|    Spano|    Amanda|    3|
|    Beane|     Marie|    3|
+---------+----------+-----+
only showing top 5 rows
```

To sort in descending order instead of ascending. In Scala this is straightforward, due to the ability to use unresolved column references, such as $"count" in this example:

```
scala> accountsDF.groupBy("last_name","first_name").count().
sort($"count".desc).show(5)
```

```
scala> accountsDF.groupBy("last_name","first_name").count().sort($"count".desc).
show(5)
+---------+----------+-----+
|last_name|first_name|count|
+---------+----------+-----+
|    Smith|     James|   38|
|  Johnson|     James|   37|
|    Smith|    Robert|   36|
|    Smith|     David|   32|
|    Brown|   Michael|   31|
+---------+----------+-----+
only showing top 5 rows
```

Count Active Devices

- The accountdevice CSV data files contain data lists all the devices used by all the accounts. Each row in the data set includes a row ID, an account ID, a device ID for the type of device, the date the device was activated

for the account, and the specific device's ID for that account. The CSV data files are in the /home/cloudera/dataset/accountdevice directory on the local file system. Review the data in the data set, then upload the directory and its contents to the HDFS directory loudacre/accountdevice.

```
// Load accountdevice data to HDFS in another terminal window

$ hdfs dfs -put accountdevice loudacre/
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put accountdevice loudacre/
[cloudera@quickstart dataset]$ hdfs dfs -ls loudacre/
Found 4 items
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 08:06 loudacre/accountdev
ice
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_z
ip94913
-rw-r--r--   1 cloudera cloudera       5483 2018-04-15 09:20 loudacre/devices.js
on
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:49 loudacre/devices_pa
rquet
[cloudera@quickstart dataset]$
```

- Create a DataFrame based on the accountdevice data files.

```
// Create a DataFrame from the account device data

val accountDeviceDF =
spark.read.option("header","true").option("inferSchema","true"
).csv("loudacre/accountdevice")
```

```
scala> val accountDeviceDF = spark.read.option("header","true").option("inferSch
ema","true").csv("loudacre/accountdevice")
18/04/18 08:07:06 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
accountDeviceDF: org.apache.spark.sql.DataFrame = [id: int, account_id: int ...
3 more fields]

scala>
```

```
// Create a DataFrame with only active accounts

val activeAccountsDF =
accountsDF.where(accountsDF("acct_close_dt").isNotNull)
```

```
scala> val activeAccountsDF = accountsDF.where(accountsDF("acct_close_dt").isNot
Null)
activeAccountsDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [acct
_num: int, acct_create_dt: timestamp ... 10 more fields]

scala>
```

```
// Create a DataFrame with a device model IDs for only devices

// used by unclosed accounts

val activeAcctDevsDF =
activeAccountsDF.join(accountDeviceDF,accountsDF("acct_num")
=== accountDeviceDF("account_id")).select("device_id")
```

```
scala> val activeAcctDevsDF =  activeAccountsDF.join(accountDeviceDF,accountsDF(
"acct_num") === accountDeviceDF("account_id")).select("device_id")
activeAcctDevsDF: org.apache.spark.sql.DataFrame = [device_id: int]

scala>
```

- Use the account device data and the DataFrames you created previously in this exercise to find the total number of each device models across all active accounts (that is, accounts that have not been closed). The new DataFrame should be sorted from most to least common model. Save the data as Parquet files in a directory called /loudacre/top_devices with the following columns:

```
// Sum up the total number of each device model

val sumDevicesDF =
activeAcctDevsDF.groupBy("device_id").count().withColumnRename
d("count","active_num")
```

```
scala> val sumDevicesDF = activeAcctDevsDF.groupBy("device_id").count().withColu
mnRenamed("count","active_num")
sumDevicesDF: org.apache.spark.sql.DataFrame = [device_id: int, active_num: bigi
nt]

scala>
```

```
// Order by count

val orderDevicesDF = sumDevicesDF.orderBy($"active_num".desc)
```

```
scala> val orderDevicesDF = sumDevicesDF.orderBy($"active_num".desc)
orderDevicesDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [device
_id: int, active_num: bigint]

scala>
```

```
// create a DataFrame based on the devices.json file

val devDF = spark.read.json("loudacre/devices.json")
```

```
scala> val devDF = spark.read.json("loudacre/devices.json")
18/04/18 08:09:47 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
18/04/18 08:09:47 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
devDF: org.apache.spark.sql.DataFrame = [dev_type: string, devnum: bigint ... 3
more fields]

scala>
```

```
// Join the list of device model totals with the list of
devices

// to get the make and model for each device

val joinDevicesDF =
orderDevicesDF.join(devDF,orderDevicesDF("device_id") ===
devDF("devnum"))
```

```
scala> val joinDevicesDF = orderDevicesDF.join(devDF,orderDevicesDF("device_id")
 === devDF("devnum"))
joinDevicesDF: org.apache.spark.sql.DataFrame = [device_id: int, active_num: big
int ... 5 more fields]

scala>
```

```
// Write out the data with the correct columns

// use overwrite mode so solution can be run multiple times

joinDevicesDF.select("device_id","make","model","active_num").
write.mode("overwrite").save("loudacre/top_devices")
```

```
scala> joinDevicesDF.select("device_id","make","model","active_num").write.mode(
"overwrite").save("loudacre/top_devices")

scala>
```

```
[cloudera@quickstart dataset]$ hdfs dfs -ls loudacre/
Found 5 items
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 08:06 loudacre/accountdevice
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:41 loudacre/accounts_zip94913
-rw-r--r--   1 cloudera cloudera       5483 2018-04-15 09:20 loudacre/devices.json
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 07:49 loudacre/devices_parquet
drwxr-xr-x   - cloudera cloudera          0 2018-04-18 08:11 loudacre/top_devices
[cloudera@quickstart dataset]$
```

```
[cloudera@quickstart dataset]$ hdfs dfs -ls loudacre/top_devices
Found 51 items
-rw-r--r--   1 cloudera cloudera          0 2018-04-18 08:11 loudacre/top_devices/_SUCCESS
-rw-r--r--   1 cloudera cloudera        919 2018-04-18 08:11 loudacre/top_devices/part-00000-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        919 2018-04-18 08:11 loudacre/top_devices/part-00001-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        914 2018-04-18 08:11 loudacre/top_devices/part-00002-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        914 2018-04-18 08:11 loudacre/top_devices/part-00003-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        894 2018-04-18 08:11 loudacre/top_devices/part-00004-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        904 2018-04-18 08:11 loudacre/top_devices/part-00005-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        919 2018-04-18 08:11 loudacre/top_devices/part-00006-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
-rw-r--r--   1 cloudera cloudera        919 2018-04-18 08:11 loudacre/top_devices/part-00007-9
1ea99da-19e4-48ad-9c54-709f372a9570-c000.snappy.parquet
```

Querying Tables and Views with SQL

Files and Data Used in This Exercise:

Data files (local) /home/cloudera/dataset/accountdevice

Hive Table accounts

In this exercise, you will use the Catalog API to explore Hive tables, and create DataFrames by executing SQL queries. Use the Catalog API to list the tables in the default Hive database, and view the schema of the accounts table. Perform queries on the accounts table, and review the resulting DataFrames. Create a temporary view based on the accountdevice CSV files, and use SQL to join that table with the accounts table.

Show tables and columns using the Catalog API

- View the list of current Hive tables and temporary views in the default database.

```
scala> spark.catalog.listTables.show
```

```
scala> spark.catalog.listTables.show
+--------+--------+-----------+---------+-----------+
|    name|database|description|tableType|isTemporary|
+--------+--------+-----------+---------+-----------+
|accounts| default|       null| EXTERNAL|      false|
+--------+--------+-----------+---------+-----------+

scala>
```

The list should include the accounts table.

- List the schema (column definitions) of the accounts table.

```
scala> spark.catalog.listColumns("accounts").show
```

```
scala> spark.catalog.listColumns("accounts").show
+-------------+-----------+---------+--------+-----------+--------+
|         name|description| dataType|nullable|isPartition|isBucket|
+-------------+-----------+---------+--------+-----------+--------+
|     acct_num|       null|      int|    true|      false|   false|
|acct_create_dt|       null|timestamp|    true|      false|   false|
| acct_close_dt|       null|timestamp|    true|      false|   false|
|   first_name|       null|   string|    true|      false|   false|
|    last_name|       null|   string|    true|      false|   false|
|      address|       null|   string|    true|      false|   false|
|         city|       null|   string|    true|      false|   false|
|        state|       null|   string|    true|      false|   false|
|      zipcode|       null|   string|    true|      false|   false|
| phone_number|       null|   string|    true|      false|   false|
|      created|       null|timestamp|    true|      false|   false|
|     modified|       null|timestamp|    true|      false|   false|
+-------------+-----------+---------+--------+-----------+--------+
```

- Create a new DataFrame based on the accounts table and confirm that its schema matches that of the column list above.

```
scala> val accountsDF = spark.read.table("accounts")

scala> accountsDF.printSchema
```

```
scala> val accountsDF = spark.read.table("accounts")
accountsDF: org.apache.spark.sql.DataFrame = [acct_num: int, acct_create_dt: tim
estamp ... 10 more fields]

scala> accountsDF.printSchema
root
 |-- acct_num: integer (nullable = true)
 |-- acct_create_dt: timestamp (nullable = true)
 |-- acct_close_dt: timestamp (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zipcode: string (nullable = true)
 |-- phone_number: string (nullable = true)
 |-- created: timestamp (nullable = true)
 |-- modified: timestamp (nullable = true)

scala>
```

Perform a SQL query on a table

- Create a new DataFrame by performing a simple SQL query on the accounts table. Confirm that the schema and data is correct.

```
scala> val firstLastDF = spark.sql("SELECT
first_name,last_name FROM accounts")
```

```
scala> val firstLastDF = spark.sql("SELECT first_name,last_name FROM accounts")
firstLastDF: org.apache.spark.sql.DataFrame = [first_name: string, last_name: st
ring]

scala>
```

```
scala> firstLastDF.printSchema
```

```
scala> firstLastDF.printSchema
root
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)


scala>
```

```
scala> firstLastDF.show(5)
```

```
scala> firstLastDF.show(5)
+----------+---------+
|first_name|last_name|
+----------+---------+
|    Donald|   Becton|
|     Donna|    Jones|
|    Dorthy| Chalmers|
|     Leila|  Spencer|
|     Anita| Laughlin|
+----------+---------+
only showing top 5 rows
```

```
scala> val firstLastDF2 =
accountsDF.select("first_name","last_name")
```

```
scala> val firstLastDF2 = accountsDF.select("first_name","last_name")
firstLastDF2: org.apache.spark.sql.DataFrame = [first_name: string, last_name: s
tring]

scala>
```

```
scala> firstLastDF2.printSchema
```

```
scala> firstLastDF2.printSchema
root
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)


scala>
```

```
scala> firstLastDF2.show(5)
```

```
scala> firstLastDF2.show(5)
+----------+---------+
|first_name|last_name|
+----------+---------+
|    Donald|   Becton|
|     Donna|    Jones|
|    Dorthy| Chalmers|
|     Leila|  Spencer|
|     Anita| Laughlin|
+----------+---------+
only showing top 5 rows
```

Create and query a view

- Create a DataFrame called accountDeviceDF based on the CSV files in loudacre/accountdevice. (Be sure to use the headers and inferred schema to determine the column names and types.)

```
scala> val accountDeviceDF =
spark.read.option("header","true").option("inferSchema","true"
).csv("loudacre/accountdevice")
```

```
scala> val accountDeviceDF = spark.read.option("header","true").option("inferSch
ema","true").csv("loudacre/accountdevice")
18/04/18 08:19:10 WARN streaming.FileStreamSink: Error while looking for metadat
a directory.
accountDeviceDF: org.apache.spark.sql.DataFrame = [id: int, account_id: int ...
3 more fields]

scala>
```

- Create a temporary view on the accountDeviceDF DataFrame called account_dev.

```
scala> accountDeviceDF.createOrReplaceTempView("account_dev")
```

```
scala> accountDeviceDF.createOrReplaceTempView("account_dev")

scala>
```

- Confirm the view was created correctly by listing the tables and views in the default database as you did

earlier. Notice that the account_dev table type is TEMPORARY.

```
scala> spark.catalog.listTables.show
```

```
scala> spark.catalog.listTables.show
+-----------+--------+-----------+---------+-----------+
|       name|database|description|tableType|isTemporary|
+-----------+--------+-----------+---------+-----------+
|   accounts| default|       null| EXTERNAL|      false|
|account_dev|    null|       null|TEMPORARY|       true|
+-----------+--------+-----------+---------+-----------+
```

- Using a SQL query, create a new DataFrame based on the first five rows of the account_dev table, and display the results.

```
scala> spark.sql("SELECT * FROM account_dev LIMIT 5").show()
```

```
scala> spark.sql("SELECT * FROM account_dev LIMIT 5").show()
+-----+----------+---------+---------------+--------------------+
|   id|account_id|device_id|activation_date|   account_device_id|
+-----+----------+---------+---------------+--------------------+
|48692|     32443|       29|  1393242509000|7351fed1-f344-4cd...|
|48693|     32444|        4|  1353649861000|6da22278-ff7a-461...|
|48694|     32445|        9|  1331819465000|cb993b85-6775-407...|
|48695|     32446|       43|  1336860950000|48ea2c09-a0df-4d1...|
|48696|     32446|       29|  1383650663000|4b49c0a6-d141-42e...|
+-----+----------+---------+---------------+--------------------+
```

Use SQL to join two tables

- Join the accounts and account_dev tables using the account ID, and display the results. (Note that the SQL string in the command below must be entered on a single line in the Spark shell.)

```
scala> val nameDevDF = spark.sql("SELECT acct_num, first_name,
last_name, account_device_id FROM accounts JOIN account_dev ON
acct_num = account_id")
```

```
scala> val nameDevDF = spark.sql("SELECT acct_num, first_name, last_name, accoun
t_device_id FROM accounts JOIN account_dev ON acct_num = account_id")
nameDevDF: org.apache.spark.sql.DataFrame = [acct_num: int, first_name: string .
.. 2 more fields]

scala>
```

```
scala> nameDevDF.show
```

```
scala> nameDevDF.show
+--------+----------+---------+--------------------+
|acct_num|first_name|last_name|   account_device_id|
+--------+----------+---------+--------------------+
|       1|    Donald|   Becton|20b671b4-2467-42d...|
|       1|    Donald|   Becton|7eb61253-55cd-430...|
|       2|     Donna|    Jones|658e3fac-8e85-40e...|
|       3|    Dorthy| Chalmers|fb674b1b-135e-408...|
|       4|     Leila|  Spencer|c0b0c922-7fd4-46b...|
|       5|     Anita| Laughlin|e054dafa-88b8-41e...|
|       5|     Anita| Laughlin|b72a7271-864e-46d...|
|       6|    Stevie|   Bridge|7d11b8e3-d67d-45f...|
|       7|     David|   Eggers|98428e3a-bfb0-487...|
|       8|   Dorothy|  Koopman|c49c4f82-9e22-4b3...|
|       8|   Dorothy|  Koopman|58830f32-8a63-4af...|
|       9|      Kara|     Kohl|5266e928-9304-4ee...|
|       9|      Kara|     Kohl|45fe71c6-c56f-435...|
|      10|     Diane|   Nelson|27bdf2f3-82a2-48d...|
|      10|     Diane|   Nelson|ece98119-a036-400...|
|      11|    Robert|   Fisher|678dc546-57ca-47f...|
|      12|    Marcia|  Roberts|71520a12-112e-448...|
|      12|    Marcia|  Roberts|2c3fb6f8-14bb-43f...|
|      13|    Andres|    Cruse|d7768cd0-b17b-4e0...|
```

- Save nameDevDF as a table called name_dev (with the file path as loudacre/name_dev).

```
scala>
nameDevDF.write.option("path","loudacre/name_dev").saveAsTable
("name_dev")
```

```
scala> nameDevDF.write.option("path","loudacre/name_dev").saveAsTable("name_dev"
)

scala>
```

- Use the Catalog API to confirm that the table was created correctly with the right schema.

```
scala> spark.catalog.listTables.show
```

```
scala> spark.catalog.listTables.show
+-----------+--------+-----------+---------+-----------+
|       name|database|description|tableType|isTemporary|
+-----------+--------+-----------+---------+-----------+
|   accounts| default|       null| EXTERNAL|      false|
|   name_dev| default|       null| EXTERNAL|      false|
|account_dev|    null|       null|TEMPORARY|       true|
+-----------+--------+-----------+---------+-----------+
```

```
scala> spark.sql("DESCRIBE name_dev").show
```

```
scala> spark.sql("DESCRIBE name_dev").show
+-----------------+---------+-------+
|         col_name|data_type|comment|
+-----------------+---------+-------+
|         acct_num|      int|   null|
|       first_name|   string|   null|
|        last_name|   string|   null|
|account_device_id|   string|   null|
+-----------------+---------+-------+
```

Exercise8: Spark Streaming

1. Spark Streaming Wordcount

Let's start with simple streaming example, to count the number of words in text data received from a data server listening on a TCP socket.

Create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

Create a Scala project with below inputs:

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext._
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds
import org.apache.spark.storage.StorageLevel;
import org.apache.spark.streaming.StreamingContext._
object SparkStreamReceiver {
  def main(args: Array[String]): Unit = {
    var port:Int = 9999;
    val conf = new SparkConf()
                .setMaster("local[*]")
                .setAppName("Simple spark streaming")

    val ssc = new StreamingContext(conf, Seconds(3));
    val dstream = ssc.socketTextStream("localhost", port,
StorageLevel.MEMORY_ONLY);
    val words = dstream.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val wordcounts = pairs.reduceByKey(_ + _)
    wordcounts.print();
    ssc.start();
    ssc.awaitTermination();
  }
```

```
}
```

Export the jar file i.e. StreamApp.jar

```
[cloudera@quickstart workspace]$ ls -ltr
total 28
drwxrwxr-x 5 cloudera cloudera 4096 Sep 13 06:23 Wordcount
-rw-rw-r-- 1 cloudera cloudera 6055 Sep 13 06:40 wordcount.jar
drwxrwxr-x 7 cloudera cloudera 4096 Sep 13 07:33 StreamApp
drwxrwxr-x 8 cloudera cloudera 4096 Sep 13 07:33 WordCount
-rw-rw-r-- 1 cloudera cloudera 6920 Sep 13 07:45 StreamApp.jar
[cloudera@quickstart workspace]$
```

Run the program using spark-submit

Run Netcat command and open another terminal to submit the job

```
nc -nlk 9999
```

```
[cloudera@quickstart workspace]$ nc -nlk 9999
hello world
hello hadoop
```

```
spark-submit --class "SparkStreamReceiver" StreamApp.jar
```

Any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following

```
16/09/13 08:19:03 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 3)
. 1312 bytes result sent to driver
16/09/13 08:19:03 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 3) in 10 ms on localhost (1/1)
16/09/13 08:19:03 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
16/09/13 08:19:03 INFO scheduler.DAGScheduler: ResultStage 4 (print at SparkStre
amReceiver.scala:24) finished in 0.012 s
16/09/13 08:19:03 INFO scheduler.DAGScheduler: Job 2 finished: print at SparkStr
eamReceiver.scala:24, took 0.033085 s
-------------------------------------------
Time: 1473779943000 ms
-------------------------------------------
(hello,2)
(world,1)
(hadoop,1)
```

2. To count number of requests for Knowledge Base articles

Files and Directories Used in This Exercise:

Exercise directory /home/cloudera/dataset/streaming-dstreams

Python stub stubs-python/StreamingLogs.py

Python solution solution-python/StreamingLogs.py

Test data (local) /home/cloudera/dataset/weblogs/

Test script streamtest.py

In this exercise, you will write a Spark Streaming application to count Knowledge Base article requests. This exercise has two parts. First, you will review the Spark Streaming documentation. Then you will write and test a Spark Streaming application to read streaming web server log data and count the number of requests for Knowledge Base articles.

Review the Spark Streaming Documentation

- View the Spark Streaming API by opening the Spark API documentation for either Scala or Python and then:

  For Scala:

  Scroll down and select the org.apache.spark.streaming package in the package pane on the left.

  Follow the links at the top of the package page to view the DStream and PairDStreamFunctions classes—these will show you the methods available on a DStream of regular RDDs and pair RDDs respectively.

For Python:

Go to the pyspark.streaming module.

Scroll down to the pyspark.streaming.DStream class and review the available methods.

- You may also wish to view the Spark Streaming Programming Guide (select Programming Guides > Spark Streaming on the Spark documentation main page).

Simulate Streaming Web Logs

To simulate a streaming data source, you will use the provided streamtest.py Python script, which waits for a connection on the host and port specified and, once it receives a connection, sends the contents of the file(s) specified to the client (which will be your Spark Streaming application). You can specify the speed (in lines per second) at which the data should be sent.

- Stream the Loudacre web log files at a rate of 20 lines per second using the provided test script.

```
$ /home/cloudera/dataset/streaming-dstreams

$ python streamtest.py localhost 9999 20 \
/home/cloudera/dataset/weblogs/*
```

```
[cloudera@quickstart streaming-dstreams]$ pwd
/home/cloudera/dataset/streaming-dstreams
[cloudera@quickstart streaming-dstreams]$ python streamtest.py localhost 9999 20
 /home/cloudera/dataset/weblogs/*
Waiting for connection on localhost : 9999
```

```
[technocrafty@technocrafty streaming-dstreams]$ python streamtest.py localhost 9999 20 /home/technocrafty/dataset/weblogs/*
Waiting for connection on localhost : 9999
Connection from ('127.0.0.1', 42104)
Streaming data from files ['/home/technocrafty/dataset/weblogs/2013-09-15.log', '/home/technocrafty/dataset/weblogs/2013-09-16.log', '/home/
technocrafty/dataset/weblogs/2013-09-17.log', '/home/technocrafty/dataset/weblogs/2013-09-18.log', '/home/technocrafty/dataset/weblogs/2013-
09-19.log', '/home/technocrafty/dataset/weblogs/2013-09-20.log', '/home/technocrafty/dataset/weblogs/2013-09-21.log', '/home/technocrafty/da
taset/weblogs/2013-09-22.log', '/home/technocrafty/dataset/weblogs/2013-09-23.log', '/home/technocrafty/dataset/weblogs/2013-09-24.log', '/h
ome/technocrafty/dataset/weblogs/2013-09-25.log', '/home/technocrafty/dataset/weblogs/2013-09-26.log', '/home/technocrafty/dataset/weblogs/2
013-09-27.log', '/home/technocrafty/dataset/weblogs/2013-09-28.log', '/home/technocrafty/dataset/weblogs/2013-09-29.log', '/home/technocraft
y/dataset/weblogs/2013-09-30.log', '/home/technocrafty/dataset/weblogs/2013-10-01.log', '/home/technocrafty/dataset/weblogs/2013-10-02.log',
 '/home/technocrafty/dataset/weblogs/2013-10-03.log', '/home/technocrafty/dataset/weblogs/2013-10-04.log', '/home/technocrafty/dataset/weblo
gs/2013-10-05.log', '/home/technocrafty/dataset/weblogs/2013-10-06.log', '/home/technocrafty/dataset/weblogs/2013-10-07.log', '/home/technoc
rafty/dataset/weblogs/2013-10-08.log', '/home/technocrafty/dataset/weblogs/2013-10-09.log', '/home/technocrafty/dataset/weblogs/2013-10-10.l
og', '/home/technocrafty/dataset/weblogs/2013-10-11.log', '/home/technocrafty/dataset/weblogs/2013-10-12.log', '/home/technocrafty/dataset/w
eblogs/2013-10-13.log', '/home/technocrafty/dataset/weblogs/2013-10-14.log', '/home/technocrafty/dataset/weblogs/2013-10-15.log', '/home/tec
hnocrafty/dataset/weblogs/2013-10-16.log', '/home/technocrafty/dataset/weblogs/2013-10-17.log', '/home/technocrafty/dataset/weblogs/2013-10-
18.log', '/home/technocrafty/dataset/weblogs/2013-10-19.log', '/home/technocrafty/dataset/weblogs/2013-10-20.log', '/home/technocrafty/datas
et/weblogs/2013-10-21.log', '/home/technocrafty/dataset/weblogs/2013-10-22.log', '/home/technocrafty/dataset/weblogs/2013-10-23.log', '/home
```

This script will exit after the client disconnects, so you will need to restart the script when you restart your Spark application.

Tip: This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each one by selecting Set Title… on the Terminal menu:

Set the title for this window to “Test Streaming.”

Write a Spark Streaming Application

- To help you get started writing a Spark Streaming application, stub files have been provided for you.

  For Python, start with the stub file StreamingLogs.py in the /home/cloudera/dataset/streaming-dstreams/stubs-python directory, which imports the necessary classes for the application.

- Define a Streaming context with a one-second batch duration
- Create a DStream by reading the data from the host and port provided as input parameters.
- Filter the DStream to only include lines containing the string KBDOC.
- To confirm that your application is correctly receiving the streaming web log data, display the first five records in the filtered DStream for each one-second batch. (In Scala, use the DStream print function; in Python, use pprint.)

- For each RDD in the filtered DStream, display the number of items—that is, the number of requests for KB articles.

  Tip: Python does not allow calling print within a lambda function, so create a named defined function to print.

- Save the filtered logs to text files in HDFS. Use the base directory name loudacre/streamlog/kblogs.

```
[technocrafty@technocrafty ~]$ hdfs dfs -ls loudacre/streamlog
Found 31 items
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725300000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725301000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725302000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725303000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725304000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725305000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725306000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725307000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725308000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725309000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725310000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725311000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725312000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725313000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725314000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725315000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725316000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725317000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725318000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 14:58 loudacre/streamlog/kblogs-1512725319000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725711000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725712000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725713000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725714000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725715000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725716000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725717000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725718000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725719000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725720000
drwxr-xr-x   - technocrafty hdfs          0 2017-12-08 15:05 loudacre/streamlog/kblogs-1512725721000
```

- Finally, start the Streaming context, and then call awaitTermination().

Test the Application

After a few batches of data have been processed and displayed, you can end the test.

- In a new terminal window, change to the correct directory for the language you are using for your application.

For Python, change to the exercise directory:

```
$ cd /home/cloudera/dataset/streaming-dstreams
```

- Use spark2-submit to run your application. The StreamingLogs application takes two parameters: the host name and the port number to which to connect the DStream. Specify the same host and port at which the test script you started earlier is listening.

  For Python:

```
$ spark2-submit stubs-python/StreamingLogs.py gateway 1234
```

Note: Use solution-python/StreamingLogs.py to run the solution application instead.

```
[technocrafty@technocrafty bin]$ spark-submit /home/technocrafty/dataset/streaming-dstreams/solution-python/StreamingLogs.py localhost 9999
Multiple versions of Spark are installed but SPARK_MAJOR_VERSION is not set
Spark1 will be picked by default
17/12/08 15:05:08 INFO spark.SparkContext: Running Spark version 1.6.3



-------------------------------------------
Time: 2017-12-08 15:05:11
-------------------------------------------
3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 14417 "http://www.loudacre.com"  "Loudacre Mobile Browse
r iFruit 1"
19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00277.html HTTP/1.0" 200 15517 "http://www.loudacre.com"  "Loudacre Mobile Bro
wser Ronin S1"
129.133.56.105 - 2489 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 10590 "http://www.loudacre.com"  "Loudacre Mobile Br
owser Sorrento F00L"

Number of KB requests: 3
-------------------------------------------
Time: 2017-12-08 15:05:12
-------------------------------------------
209.151.12.34 - 45922 [15/Sep/2013:23:55:09 +0100] "GET /KBDOC-00259.html HTTP/1.0" 200 19362 "http://www.loudacre.com"  "Loudacre Mobile Br
owser Sorrento F11L"
184.97.84.245 - 144 [15/Sep/2013:23:54:55 +0100] "GET /KBDOC-00052.html HTTP/1.0" 200 12499 "http://www.loudacre.com"  "Loudacre CSR Browser
"
233.60.251.2 - 33908 [15/Sep/2013:23:51:43 +0100] "GET /KBDOC-00292.html HTTP/1.0" 200 4779 "http://www.loudacre.com"  "Loudacre Mobile Brow
ser Ronin S2"
160.134.139.204 - 51340 [15/Sep/2013:23:50:11 +0100] "GET /KBDOC-00016.html HTTP/1.0" 200 1156 "http://www.loudacre.com"  "Loudacre Mobile B
rowser MeeToo 3.0"
19.209.18.222 - 13392 [15/Sep/2013:23:48:48 +0100] "GET /KBDOC-00083.html HTTP/1.0" 200 4398 "http://www.loudacre.com"  "Loudacre Mobile Bro
wser Titanic 2200"
...

Number of KB requests: 8
-------------------------------------------
Time: 2017-12-08 15:05:13
```

- After a few moments, the application will connect to the test script's simulated stream of web server log output. Confirm that for every batch of data received (every second), the application displays the first few Knowledge Base requests and the count of requests in the batch. Review the HDFS files the application saved in /loudacre/streamlog.

- Return to the terminal window in which you started the streamtest.py test script earlier. Stop the test script by typing Ctrl+C.

- Return to the terminal window in which your application is running. Stop your application by typing Ctrl+C. (You may see several error messages resulting from the interruption of the job in Spark; you may disregard these.)

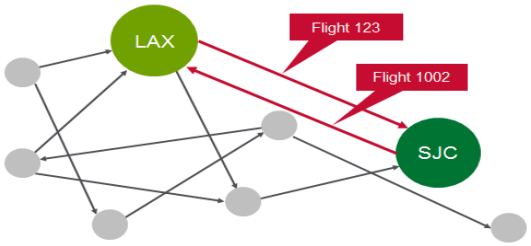
Exercise9: GraphX

Analyzing Flight Data

Graphx Property Graph

GraphX extends the Spark RDD with a Resilient Distributed Property Graph.

The property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. The parallel edges allow multiple relationships between the same vertices.



Scenario

As a starting simple example, we will analyze 3 flights, for each flight we have the following information:

Originating Airport	Destination Airport	Distance
SFO	ORD	1800 miles
ORD	DFW	800 miles
DFW	SFO	1400 miles

In this scenario, we are going to represent the airports as vertices and routes as edges. For our graph we will have three vertices, each representing an airport. The Vertices each have an Id and the airport code as a property:

Vertex Table for Airports

ID	Property
1	SFO
2	ORD
3	DFW

The Edges have the Source Id, the Destination Id and the distance as a property.

Edges Table for Routes

SrcId	DestId	Property
1	2	1800
2	3	800
3	1	1400

- Launch the Spark Interactive Shell

Define Vertices

First we will import the GraphX packages.

```
import org.apache.spark._

import org.apache.spark.rdd.RDD

// import classes required for using GraphX

import org.apache.spark.graphx._
```

```
scala> import org.apache.spark._
import org.apache.spark._

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala> import org.apache.spark.graphx._
import org.apache.spark.graphx._

scala>
```

We define airports as vertices. Vertices have an Id and can have properties or attributes associated with them. Each vertex consists of:

Vertex id → Id (Long)

Vertex Property → name (String)

Vertex Table for Airports

ID	Property(V)
1	SFO

We define an RDD with the above properties that is then used for the Vertexes.

```
// create  vertices RDD with ID and Name

val vertices=Array((1L, ("SFO")),(2L, ("ORD")),(3L,("DFW")))

val vRDD= sc.parallelize(vertices)

vRDD.take(1)

// Array((1,SFO))


// Defining a default vertex called nowhere

val nowhere = "nowhere"
```

```
scala> val vertices=Array((1L, ("SFO")),(2L, ("ORD")),(3L,("DFW")))
vertices: Array[(Long, String)] = Array((1,SFO), (2,ORD), (3,DFW))

scala> val vRDD= sc.parallelize(vertices)
vRDD: org.apache.spark.rdd.RDD[(Long, String)] = ParallelCollectionRDD[2] at parallelize at <console>:30

scala> vRDD.take(1)
18/04/17 20:38:26 INFO spark.SparkContext: Starting job: take at <console>:33
18/04/17 20:38:26 INFO scheduler.DAGScheduler: Got job 2 (take at <console>:33) with 1 output partitions

18/04/17 20:38:26 INFO scheduler.DAGScheduler: Job 3 finished: take at <console>:33, took 0.020053 s
res1: Array[(Long, String)] = Array((1,SFO))

scala>
```

```
scala> val nowhere = "nowhere"
nowhere: String = nowhere

scala>
```

Define Edges

Edges are the routes between airports. An edge must have a source, a destination, and can have properties. In our example, an edge consists of:

Edge origin id → src (Long)

Edge destination id → dest (Long)

Edge Property distance → distance (Long)

Edges Table for Routes

srcid	destid	Property(E)
1	12	1800

We define an RDD with the above properties that is then used for the Edges. The edge RDD has the form (src id, dest id, distance).

```
// create  routes  RDD with srcid, destid , distance

val edges =
Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))

val eRDD= sc.parallelize(edges)


eRDD.take(2)

// Array(Edge(1,2,1800), Edge(2,3,800))
```

```
scala> val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
edges: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800), Edge(3,1,1400))

scala> val eRDD= sc.parallelize(edges)
eRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[3] at parallelize at <con
sole>:30

scala> eRDD.take(2)
18/04/17 20:39:54 INFO spark.SparkContext: Starting job: take at <console>:33
18/04/17 20:39:54 INFO scheduler.DAGScheduler: Got job 4 (take at <console>:33) with 1 output partitions

18/04/17 20:39:54 INFO scheduler.DAGScheduler: Job 5 finished: take at <console>:33, took 0.011294 s
res2: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800))

scala>
```

Create Property Graph

To create a graph, you need to have a Vertex RDD, Edge RDD and a Default vertex.

```
// define the graph

val graph = Graph(vRDD,eRDD, nowhere)

// graph vertices

graph.vertices.collect.foreach(println)

// (2,ORD)

// (1,SFO)

// (3,DFW)
```

```
// graph edges
graph.edges.collect.foreach(println)

// Edge(1,2,1800)
// Edge(2,3,800)
// Edge(3,1,1400)
```

```
scala> val graph = Graph(vRDD,eRDD, nowhere)
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_5_piece0 on localhost:56186 in memory (size: 888
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 6
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_4_piece0 on localhost:56186 in memory (size: 888
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 5
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_3_piece0 on localhost:56186 in memory (size: 838
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 4
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_2_piece0 on localhost:56186 in memory (size: 838
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 3
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_1_piece0 on localhost:56186 in memory (size: 836
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 2
18/04/17 20:45:29 INFO storage.BlockManagerInfo: Removed broadcast_0_piece0 on localhost:56186 in memory (size: 836
.0 B, free: 511.1 MB)
18/04/17 20:45:29 INFO spark.ContextCleaner: Cleaned accumulator 1
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImpl@70d8f66

scala>
```

```
18/04/17 20:45:56 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all completed, from pool
18/04/17 20:45:56 INFO scheduler.DAGScheduler: ResultStage 8 (collect at <console>:41) finished in 0.039 s
18/04/17 20:45:56 INFO scheduler.DAGScheduler: Job 6 finished: collect at <console>:41, took 0.124166 s
(1,SFO)
(2,ORD)
(3,DFW)

scala>
```

```
18/04/17 20:46:20 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 9.0, whose tasks have all completed, from pool
18/04/17 20:46:20 INFO scheduler.DAGScheduler: ResultStage 9 (collect at EdgeRDDImpl.scala:51) finished in 0.018 s
18/04/17 20:46:20 INFO scheduler.DAGScheduler: Job 7 finished: collect at EdgeRDDImpl.scala:51, took 0.025542 s
Edge(1,2,1800)
Edge(2,3,800)
Edge(3,1,1400)

scala>
```

1. How many airports are there?

```
// How many airports?
val numairports = graph.numVertices
// Long = 3
```

```
18/04/17 20:46:39 INFO scheduler.DAGScheduler: ResultStage 12 (reduce at VertexRDDImpl.scala:90) finished in 0.015
s
18/04/17 20:46:39 INFO scheduler.DAGScheduler: Job 8 finished: reduce at VertexRDDImpl.scala:90, took 0.031259 s
numairports: Long = 3

scala>
```

2. How many routes are there?

```
// How many routes?

val numroutes = graph.numEdges

// Long = 3
```

```
18/04/17 20:47:02 INFO scheduler.DAGScheduler: ResultStage 13 (reduce at EdgeRDDImpl.scala:89) finished in 0.014 s
18/04/17 20:47:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 13.0, whose tasks have all completed, from pool

18/04/17 20:47:02 INFO scheduler.DAGScheduler: Job 9 finished: reduce at EdgeRDDImpl.scala:89, took 0.020304 s
numroutes: Long = 3

scala>
```

3. which routes > 1000 miles distance?

```
// routes >  1000 miles distance?

graph.edges.filter { case Edge(src, dst, prop) => prop > 1000
}.collect.foreach(println)

// Edge(1,2,1800)

// Edge(3,1,1400)
```

```
18/04/17 20:47:26 INFO scheduler.DAGScheduler: ResultStage 14 (collect at <console>:41) finished in 0.005 s
18/04/17 20:47:26 INFO scheduler.DAGScheduler: Job 10 finished: collect at <console>:41, took 0.014611 s
Edge(1,2,1800)
Edge(3,1,1400)

scala>
```

4. The EdgeTriplet class extends the Edge class by adding the srcAttr and dstAttr members which contain the source and destination properties respectively.

```
// triplets

graph.triplets.take(3).foreach(println)

((1,SFO),(2,ORD),1800)

((2,ORD),(3,DFW),800)

((3,DFW),(1,SFO),1400)
```

```
18/04/17 20:47:51 INFO scheduler.DAGScheduler: ResultStage 22 (take at <console>:41) finished in 0.020 s
18/04/17 20:47:51 INFO scheduler.DAGScheduler: Job 12 finished: take at <console>:41, took 0.025056 s
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)

scala>
```

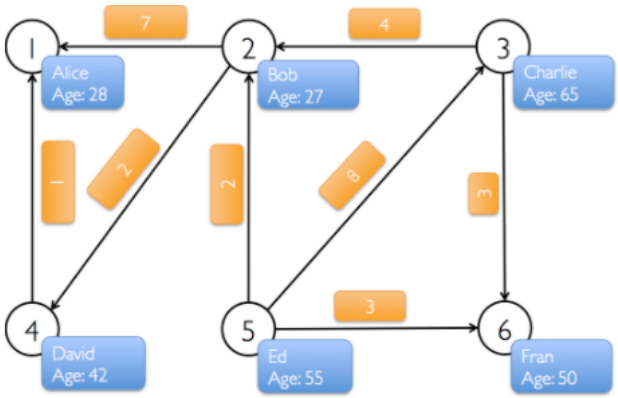
5. Sort and print out the longest distance routes

```
// print out longest routes

graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>

"Distance " + triplet.attr.toString + " from " +
triplet.srcAttr + " to " + triplet.dstAttr +
".").collect.foreach(println)


Distance 1800 from SFO to ORD.

Distance 1400 from DFW to SFO.

Distance 800 from ORD to DFW.
```

```
18/04/17 20:48:19 INFO scheduler.DAGScheduler: ResultStage 31 (collect at <console>:42) finished in 0.040 s
18/04/17 20:48:19 INFO scheduler.DAGScheduler: Job 14 finished: collect at <console>:42, took 0.074311 s
Distance 1800 from SFO to ORD.
Distance 1400 from DFW to SFO.
Distance 800 from ORD to DFW.

scala>
```

Analyze number of likes

We have a small social network with users and their ages modeled as vertices and likes modeled as directed edges.



Creating Property Graph

Use the Edge class. Edges have a srcId and a dstId corresponding to the source and destination vertex identifiers. In addition, the Edgeclass has an attr member which stores the edge property (in this case the number of likes).

```
val vertexArray = Array(
  (1L, ("Alice", 28)),
  (2L, ("Bob", 27)),
  (3L, ("Charlie", 65)),
  (4L, ("David", 42)),
  (5L, ("Ed", 55)),
  (6L, ("Fran", 50))
  )
val edgeArray = Array(
  Edge(2L, 1L, 7),
  Edge(2L, 4L, 2),
```

```
  Edge(3L, 2L, 4),
  Edge(3L, 6L, 3),
  Edge(4L, 1L, 1),
  Edge(5L, 2L, 2),
  Edge(5L, 3L, 8),
  Edge(5L, 6L, 3)
  )
```

```
scala> val vertexArray = Array(
     |   (1L, ("Alice", 28)),
     |   (2L, ("Bob", 27)),
     |   (3L, ("Charlie", 65)),
     |   (4L, ("David", 42)),
     |   (5L, ("Ed", 55)),
     |   (6L, ("Fran", 50))
     |   )
vertexArray: Array[(Long, (String, Int))] = Array((1,(Alice,28)), (2,(Bob,27)),
(3,(Charlie,65)), (4,(David,42)), (5,(Ed,55)), (6,(Fran,50)))

scala>
```

```
scala> val edgeArray = Array(
     |   Edge(2L, 1L, 7),
     |   Edge(2L, 4L, 2),
     |   Edge(3L, 2L, 4),
     |   Edge(3L, 6L, 3),
     |   Edge(4L, 1L, 1),
     |   Edge(5L, 2L, 2),
     |   Edge(5L, 3L, 8),
     |   Edge(5L, 6L, 3)
     |   )
edgeArray: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(2,1,7), Edge(2,
4,2), Edge(3,2,4), Edge(3,6,3), Edge(4,1,1), Edge(5,2,2), Edge(5,3,8), Edge(5,6,
3))
```

Using sc.parallelize to construct the following RDDs from the vertexArray and edgeArray variables.

```
val vertexRDD: RDD[(Long, (String, Int))] =
sc.parallelize(vertexArray)

val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
```

```
scala> val vertexRDD: RDD[(Long, (String, Int))] = sc.parallelize(vertexArray)
vertexRDD: org.apache.spark.rdd.RDD[(Long, (String, Int))] = ParallelCollectionR
DD[0] at parallelize at <console>:30

scala> val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
edgeRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelC
ollectionRDD[1] at parallelize at <console>:30

scala>
```

The basic property graph constructor takes an RDD of vertices (with type RDD[(VertexId, V)]) and an RDD of edges (with type RDD[Edge[E]]) and builds a graph (with type Graph[V, E]).

```
val graph: Graph[(String, Int), Int] = Graph(vertexRDD, edgeRDD)
```

```
scala> val graph: Graph[(String, Int), Int] = Graph(vertexRDD, edgeRDD)
graph: org.apache.spark.graphx.Graph[(String, Int),Int] = org.apache.spark.graph
x.impl.GraphImpl@22b0c1cf

scala>
```

The vertex property for this graph is a tuple (String, Int) corresponding to the User Name and Age and the edge property is just an Intcorresponding to the number of Likes in our hypothetical social network.

Graph Views

Use graph.vertices to display the names of the users that are at least 30 years old.

```
graph.vertices.filter { case (id, (name, age)) => age > 30
}.collect.foreach {
  case (id, (name, age)) => println(s"$name is $age")
}
```

```
scala> graph.vertices.filter { case (id, (name, age)) => age > 30 }.collect.fore
ach {
     |   case (id, (name, age)) => println(s"$name is $age")
     | }
David is 42
Fran is 50
Charlie is 65
Ed is 55

scala>
```

In addition to the vertex and edge views of the property graph, GraphX also exposes a triplet view. The triplet view logically joins the vertex and edge properties yielding an RDD[EdgeTriplet[VD, ED]] containing instances of the EdgeTriplet class.

The EdgeTriplet class extends the Edge class by adding the srcAttr and dstAttr members which contain the source and destination properties respectively.

Use the graph.triplets view to display who likes who.

```
for (triplet <- graph.triplets.collect) {

  println(s"${triplet.srcAttr._1} likes
${triplet.dstAttr._1}")

}
```

```
scala> for (triplet <- graph.triplets.collect) {
     |   println(s"${triplet.srcAttr._1} likes ${triplet.dstAttr._1}")
     | }
Bob likes Alice
Bob likes David
Charlie likes Bob
Charlie likes Fran
David likes Alice
Ed likes Bob
Ed likes Charlie
Ed likes Fran

scala>
```

If someone likes someone else more than 5 times than that relationship is getting pretty serious. For extra credit, find the lovers.

```
for (triplet <- graph.triplets.filter(t => t.attr >
5).collect) {

  println(s"${triplet.srcAttr._1} loves
${triplet.dstAttr._1}")

}
```

```
scala> for (triplet <- graph.triplets.filter(t => t.attr > 5).collect) {
     |   println(s"${triplet.srcAttr._1} loves ${triplet.dstAttr._1}")
     | }
Bob loves Alice
Ed loves Charlie

scala>
```

Compute the in-degree of each vertex

```
val inDegrees: VertexRDD[Int] = graph.inDegrees
```

In the above example the graph.inDegrees operators returned a VertexRDD[Int] (recall that this behaves like RDD[(VertexId, Int)]).

What if we wanted to incorporate the in and out degree of each vertex into the vertex property? To do this we will use a set of common graph operators.

First, we define a User class to better organize the vertex property and build a new graph with the user property

```
// Define a class to more clearly model the user property

case class User(name: String, age: Int, inDeg: Int, outDeg:
Int)

// Create a user Graph

val initialUserGraph: Graph[User, Int] = graph.mapVertices{
case (id, (name, age)) => User(name, age, 0, 0) }
```

```
scala> val inDegrees: VertexRDD[Int] = graph.inDegrees
inDegrees: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[37] at RDD at
VertexRDD.scala:57

scala> case class User(name: String, age: Int, inDeg: Int, outDeg: Int)
defined class User

scala> val initialUserGraph: Graph[User, Int] = graph.mapVertices{ case (id, (na
me, age)) => User(name, age, 0, 0) }
initialUserGraph: org.apache.spark.graphx.Graph[User,Int] = org.apache.spark.gra
phx.impl.GraphImpl@4e23ec20

scala>
```

Notice that we initialized each vertex with 0 in and out degree. Now we join the in and out degree information with each vertex building the new vertex property

```
// Fill in the degree information

val userGraph =
initialUserGraph.outerJoinVertices(initialUserGraph.inDegrees)
{

  case (id, u, inDegOpt) => User(u.name, u.age,
inDegOpt.getOrElse(0), u.outDeg)

}.outerJoinVertices(initialUserGraph.outDegrees) {

  case (id, u, outDegOpt) => User(u.name, u.age, u.inDeg,
outDegOpt.getOrElse(0))

}
```

```
scala> val userGraph = initialUserGraph.outerJoinVertices(initialUserGraph.inDeg
rees) {
     |   case (id, u, inDegOpt) => User(u.name, u.age, inDegOpt.getOrElse(0), u.
outDeg)
     | }.outerJoinVertices(initialUserGraph.outDegrees) {
     |   case (id, u, outDegOpt) => User(u.name, u.age, u.inDeg, outDegOpt.getOr
Else(0))
     | }
userGraph: org.apache.spark.graphx.Graph[User,Int] = org.apache.spark.graphx.imp
l.GraphImpl@9a82f1d

scala>
```

Use the degreeGraph to print the number of people who like each user:

```
for ((id, property) <- userGraph.vertices.collect) {
  println(s"User $id is called ${property.name} and is liked
by ${property.inDeg} people.")
}
```

```
scala> for ((id, property) <- userGraph.vertices.collect) {
     |   println(s"User $id is called ${property.name} and is liked by ${propert
y.inDeg} people.")
     | }
User 4 is called David and is liked by 1 people.
User 6 is called Fran and is liked by 2 people.
User 2 is called Bob and is liked by 2 people.
User 1 is called Alice and is liked by 2 people.
User 3 is called Charlie and is liked by 1 people.
User 5 is called Ed and is liked by 0 people.

scala>
```

Print the names of the users who are liked by the same number of people they like.

```
userGraph.vertices.filter {
  case (id, u) => u.inDeg == u.outDeg
}.collect.foreach {
  case (id, property) => println(property.name)
}
```

```
scala> userGraph.vertices.filter {
     |   case (id, u) => u.inDeg == u.outDeg
     | }.collect.foreach {
     |   case (id, property) => println(property.name)
     | }
David
Bob
```

Exercise10: Machine Learning with Spark ML and MLlib

Statistics and Transformations with Spark MLlib

Files and Data Used in This Exercise

Local data /home/cloudera/dataset/concrete.csv

HDFS path /user/cloudera/mldata

In this exercise, you will use Spark MLlib to transform and compute basic statistics on a data set.

Loading the Data into HDFS

The data that will be used for this exercise concerns various properties of concrete.

#	Field Name	Type	Description
0	cement	Double	Mass, in kg per cubic meter of mixture
1	blast_furnace_slag	Double	Mass, in kg per cubic meter of mixture
2	fly_ash	Double	Mass, in kg per cubic meter of mixture
3	water	Double	Mass, in kg per cubic meter of mixture
4	superplasticizer	Double	Mass, in kg per cubic meter of mixture
5	coarse_aggregate	Double	Mass, in kg per cubic meter of mixture
6	fine_aggregate	Double	Mass, in kg per cubic meter of mixture
7	age	Double	Age, in days
8	compressive_strength	Double	Strength, in megapascals (MPa)

1. Open a new terminal window

2. Add the input data file to the /user/cloudera/mldata directory in HDFS:

```
$ hdfs dfs -put concrete.csv /user/cloudera/mldata
```

```
[cloudera@quickstart dataset]$ hdfs dfs -mkdir mldata
[cloudera@quickstart dataset]$ hdfs dfs -put concrete.csv mldata
[cloudera@quickstart dataset]$
```

Calculating Statistics and Performing Transformations with Spark MLlib

- Open either the Scala Spark REPL:

- Begin by importing the necessary modules for this exercise. If you're using the Scala REPL, you'll use this code:

  If you're using the Scala REPL, you'll use this code:

```
import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.feature.ElementwiseProduct

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.mllib.stat.Statistics
```

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.feature.ElementwiseProduct
import org.apache.spark.mllib.feature.ElementwiseProduct

scala> import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.feature.StandardScaler

scala> import org.apache.spark.mllib.stat.Statistics
import org.apache.spark.mllib.stat.Statistics
```

- Load the concrete.csv file you just added to HDFS into an RDD named rawdata using the textFile method of the sc (SparkContext) object.

```
// TODO:

// Load the raw data into the rawdata RDD

val rawdata =
sc.textFile("/user/cloudera/mldata/concrete.csv")
```

```
scala> val rawdata = sc.textFile("/user/cloudera/mldata/concrete.csv")
rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/concrete.csv MapPartitionsRDD[1] at textFile at <console>:28

scala>
```

- Convert the rawdata RDD into an RDD of dense vectors called vecrdd. Be sure to convert the split values to a Double in Scala or a float in Python before casting to a dense vector.

```
// TODO:

// Convert the rawdata RDD to an RDD of dense vectors called vecrdd.

// Be sure to map the split values to a Double before casting to a dense vector.

val vecrdd = rawdata.map(x => x.split(",").map(_.toDouble)).map(x => Vectors.dense(x))
```

```
scala> val vecrdd = rawdata.map(x => x.split(",").map(_.toDouble)).map(x => Vectors.dense(x))
vecrdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPartitionsRDD[3] at map at <console>:30

scala>
```

- Create a dense vector of weights named weights to be used with an ElementWiseProduct transformer to weight each element in the RDD vecrdd. You can make the elements of the vector any floating-point values you would like, but ensure that the vector has a weight for each of the nine columns of the data set.

```
// TODO:

// Create a dense vector of weights, naming the vector "weights"

// Make sure that this vector has a weight for each of the 9 columns of the dataset.

val weights = Vectors.dense(Array(0.3, 0.1, 0.1, 0.1, 0.1, 0.1, 0.5, 0.5, 1.0))
```

```
scala> val weights = Vectors.dense(Array(0.3, 0.1, 0.1, 0.1, 0.1, 0.1, 0.5, 0.5,
 1.0))
weights: org.apache.spark.mllib.linalg.Vector = [0.3,0.1,0.1,0.1,0.1,0.1,0.5,0.5
,1.0]
```

- Create an instance of ElementWiseProduct, initialized with the weights vector and assign it to a variable called ep.

```
// TODO:

// Create an instance of "ElementwiseProduct"

// Initialize the instance with the weights vector and store it in a variable called "ep"

val ep = new ElementwiseProduct(weights)
```

```
scala> val ep = new ElementwiseProduct(weights)
ep: org.apache.spark.mllib.feature.ElementwiseProduct = org.apache.spark.mllib.f
eature.ElementwiseProduct@4063af6
```

- Use the transform method of the transformer ep to transform the RDD vecrdd and assign the resulting RDD to a variable called weighted. Then print the first line of both vecrdd and weighted to confirm that the values have been weighted properly.

```
// TODO

// Transform vecrdd and store the output in an RDD called "weighted".

// Then print the first row of vecrdd and the weighted.

val weighted = ep.transform(vecrdd)

weighted.take(1).foreach(println)

vecrdd.take(1).foreach(println)
```

```
scala> val weighted = ep.transform(vecrdd)
weighted: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPa
rtitionsRDD[4] at map at VectorTransformer.scala:52

scala>
```

```
scala> weighted.take(1).foreach(println)
17/12/06 22:10:29 INFO mapred.FileInputFormat: Total input paths to process : 1
17/12/06 22:10:29 INFO spark.SparkContext: Starting job: take at <console>:42
```

```
17/12/06 22:10:29 INFO scheduler.DAGScheduler: Job 0 finished: take at <console>
:42, took 0.447962 s
[162.0,0.0,0.0,16.2,0.25,104.0,338.0,14.0,79.99]
```

```
scala> vecrdd.take(1).foreach(println)
17/12/06 22:11:09 INFO spark.SparkContext: Starting job: take at <console>:36
17/12/06 22:11:09 INFO scheduler.DAGScheduler: Got job 1 (take at <console>:36)
with 1 output partitions
```

```
17/12/06 22:11:09 INFO scheduler.DAGScheduler: Job 1 finished: take at <console>
:36, took 0.055138 s
[540.0,0.0,0.0,162.0,2.5,1040.0,676.0,28.0,79.99]
```

- Call the colStats method of the Statistics object on the RDD vecrdd and store the resulting object into a variable called stats. Next, print out the mean, variance, and number of non-zero elements of the vecrdd RDD by referencing the appropriate attributes (Scala) or methods (Python) of the stats object.

```
// TODO:

// Compute basic statistics on the vecrdd using
Statistics.colStats()

// Print the mean, variance, and numNonZeros

val stats = Statistics.colStats(vecrdd)
```

```
scala> val stats = Statistics.colStats(vecrdd)
17/12/06 22:13:30 INFO spark.SparkContext: Starting job: treeAggregate at RowMat
rix.scala:405
17/12/06 22:13:30 INFO scheduler.DAGScheduler: Got job 2 (treeAggregate at RowMa
trix.scala:405) with 2 output partitions
```

```
println(stats.mean)
```

```
scala> println(stats.mean)
[281.1678640776697,73.89582524271839,54.188349514563086,181.56728155339817,6.204
660194174761,972.9189320388349,773.5804854368936,45.662135922330094,35.817961165
048544]
```

```
println(stats.variance)
```

```
scala> println(stats.variance)
[10921.580219923186,7444.124812486428,4095.6165405191177,456.00265052317604,35.6
8678098257336,6045.677357458954,6428.187791795222,3990.437729155462,279.08181449
800463]
```

```
println(stats.numNonzeros)
```

```
scala> println(stats.numNonzeros)
[1030.0,559.0,464.0,1030.0,651.0,1030.0,1030.0,1030.0,1030.0]
```

- Next, a StandardScaler transformer will be used to center and scale the data. Create an instance of StandardScaler, setting the withMean and withStd parameters to true. Store this in a variable called ss.

```
val ss = new StandardScaler(withMean=true, withStd=true)
```

- Call the fit method of ss on vecrdd to create a StandardScalerModel, and save the result into a variable called model.

```
val model = ss.fit(vecrdd)
```

- To transform the vecrdd, call the transform method of model on vecrdd and save the output to a new RDD called scaled.

```
val scaled = model.transform(vecrdd)
```

- Now call Statistics.colStats on the new RDD scaled to compute the stats on this RDD, and print out the mean, variance, and number of non-zeros. Has the data been scaled and centered?

```
val newstats = Statistics.colStats(scaled)

println(newstats.mean)

println(newstats.variance)

println(newstats. numNonzeros)
```

```
scala> println(newstats.mean)
[1.7763568394002505E-15,6.245004513516506E-16,4.440892098500626E-16,-5.606626274
3570405E-15,-6.106226635438361E-16,7.216449660063518E-16,-4.760081218080359E-15,
1.3877787807814457E-17,-2.7755575615628914E-16]

scala>
```

```
scala> println(newstats.variance)
[1.0000000000000007,1.000000000000001,1.0000000000000002,1.0000000000000013,0.99
99999999999987,0.9999999999999969,1.0000000000000004,1.0000000000000004,1.0]

scala>
```

```
localhost:43038 in memory (size: 2.8 KB, free: 511.1 MB)

[1030.0,1030.0,1030.0,1030.0,1030.0,1030.0,1030.0,1030.0,1030.0]

scala>
```

Performing Linear Regression with Spark MLlib

Files and Data Used in This Exercise

Data Files /home/cloudera/dataset/concrete_test.csv
/home/cloudera/dataset/concrete_train.csv

HDFS path /user/cloudera/mldata

In this exercise, you will create linear regression models using no regularization, L1 regularization, and L2 regularization using Spark MLlib.

Preparing the Data, the data* that will be used for this exercise concerns various properties of concrete. The target value to be predicted is the compressive strength, which is a continuous, floating point variable.

#	Field Name	Type	Description
0	cement	Double	Mass, in kg per cubic meter of mixture
1	blast_furnace_slag	Double	Mass, in kg per cubic meter of mixture
2	fly_ash	Double	Mass, in kg per cubic meter of mixture
3	water	Double	Mass, in kg per cubic meter of mixture
4	superplasticizer	Double	Mass, in kg per cubic meter of mixture
5	coarse_aggregate	Double	Mass, in kg per cubic meter of mixture
6	fine_aggregate	Double	Mass, in kg per cubic meter of mixture
7	age	Double	Age, in days
8	compressive_strength	Double	Strength, in megapascals (MPa)

- Open a new terminal window
- Add the two data files used in this exercise to the /user/cloudera/mldata directory in HDFS:

```
$ hdfs dfs -put concrete_test.csv concrete_train.csv mldata
```

```
[cloudera@quickstart dataset]$ hdfs dfs -put concrete.csv mldata
[cloudera@quickstart dataset]$ hdfs dfs -put concrete_test.csv concrete_train.cs
v mldata
[cloudera@quickstart dataset]$ hdfs dfs -ls mldata
Found 3 items
-rw-r--r--   1 cloudera cloudera      49615 2018-04-18 22:17 mldata/concrete.csv
-rw-r--r--   1 cloudera cloudera       6378 2018-04-18 22:21 mldata/concrete_tes
t.csv
-rw-r--r--   1 cloudera cloudera      43237 2018-04-18 22:21 mldata/concrete_tra
in.csv
[cloudera@quickstart dataset]$
```

Calculating Linear Regression with Spark MLlib

- Open either the Scala Spark REPL:
- Begin by importing the necessary modules for this exercise. If you're using the Scala REPL, you'll use this code:

```
import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.mllib.regression.
{LinearRegressionWithSGD, LassoWithSGD,
RidgeRegressionWithSGD, LabeledPoint}
```

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.feature.StandardScaler

scala> import org.apache.spark.mllib.regression. {LinearRegressionWithSGD, Lasso
WithSGD, RidgeRegressionWithSGD, LabeledPoint}
import org.apache.spark.mllib.regression.{LinearRegressionWithSGD, LassoWithSGD,
 RidgeRegressionWithSGD, LabeledPoint}

scala>
```

- Load the data from the delimited file concrete_train.csv from HDFS into an RDD called rawdata using the textFile method of the SparkContext object.

```
// TODO:

// Load the data from the delimited file "concrete_train.csv"
into the rdd rawdata

val rawdata =
sc.textFile("/user/cloudera/mldata/concrete_train.csv")
```

```
scala> val rawdata = sc.textFile("/user/cloudera/mldata/concrete_train.csv")
rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/concrete_train.csv MapPartitionsRDD[5] at textFile at <console>:28

scala>
```

- Next, map the rawdata RDD which contains strings to an RDD of dense vectors called vecrdd by splitting on commas, casting the values as Scala Double or Python float, then casting the resulting Scala array or Python list as a dense vector.

```
// TODO:

// Map the RDD of strings to an RDD of dense vectors by
splitting
```

```
// on commas, casting each value as floating point, then
casting as a dense vector

val vecrdd = rawdata.map(x =>
x.split(",").map(_.toDouble)).map(x => Vectors.dense(x))
```

```
scala> val vecrdd = rawdata.map(x => x.split(",").map(_.toDouble)).map(x => Vect
ors.dense(x))
vecrdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPart
itionsRDD[202] at map at <console>:32

scala>
```

- Create an instance of StandardScaler, called ss, setting withMean and withStd both to be true. Use the fit method of ss on vecrdd and save the resulting StandardScalerModel as model (this will be used in a later step).

```
// TODO:

// Create an instance of StandardScaler, setting withMean and
withStd both True.

// Use it to scale the vector RDD create above, and store in a
new RDD called scaled

// Note: Be sure to store the model created by the fit method
of the StandardScaler

// in a variable as it will be used in a later step

val ss = new StandardScaler(withMean=true, withStd=true)
```

```
scala> val ss = new StandardScaler(withMean=true, withStd=true)
ss: org.apache.spark.mllib.feature.StandardScaler = org.apache.spark.mllib.featu
re.StandardScaler@79b66662

scala>
```

```
val model = ss.fit(vecrdd)
```

```
scala> val model = ss.fit(vecrdd)
model: org.apache.spark.mllib.feature.StandardScalerModel = org.apache.spark.mll
ib.feature.StandardScalerModel@bbf8415

scala>
```

```
val scaled = model.transform(vecrdd)
```

```
scala> val scaled = model.transform(vecrdd)
scaled: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPart
itionsRDD[204] at map at VectorTransformer.scala:52

scala>
```

- Next, use the transform method of model to transform vecrdd and assign the resulting RDD to a variable called scaled.
- Map the scaled RDD to a new RDD of LabeledPoint called lprdd. The label should be the last element of the Vectors in vecrdd, and the features should be the remaining elements of the Vectors (all but the last).

```
// TODO:
// Create an RDD of labeledPoints using the last value in the
list
// as the feature and all but the last value in the list as
the features.
// Be sure to explicitly cast the features as a dense vector
before
// using in the labeledPoint initialization.
val lprdd = scaled.map{x =>
        val temp = x.toArray
        val label = temp.last
        val features = Vectors.dense(temp.dropRight(1))
        LabeledPoint(label, features)
}
```

```
scala> val lprdd = scaled.map{x =>
     |         val temp = x.toArray
     |         val label = temp.last
     |         val features = Vectors.dense(temp.dropRight(1))
     |         LabeledPoint(label, features)
     | }
lprdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]
= MapPartitionsRDD[205] at map at <console>:40

scala>
```

- Create three linear regression models, one with no regularization (named lr_model), one with L1 regularization (named lr_l1_model), and one with L2 regularization (named lr_l2_model). Set the regParam to 0.2 for both regularized models.

```
// TODO:

// Create three LinearRegressionWithSGD models

// by calling the train method with different parameters on
the RDD of labeled points.

// The first should have no regularization, the second should
use l1 regularization,

// and the third should use l2 regularization.

// For both l1 and l2 regularization, set the regParam to 0.2


val lr_model = LinearRegressionWithSGD.train(lprdd,
numIterations = 10)
```

```
scala> val lr_model = LinearRegressionWithSGD.train(lprdd, numIterations = 10)
warning: there was one deprecation warning; re-run with -deprecation for details
17/12/07 20:35:33 WARN LinearRegressionWithSGD: The input data is not directly c
ached, which may hurt performance if its parent RDDs are also uncached.
17/12/07 20:35:33 WARN BLAS: Failed to load implementation from: com.github.fomm
il.netlib.NativeSystemBLAS
17/12/07 20:35:33 WARN BLAS: Failed to load implementation from: com.github.fomm
il.netlib.NativeRefBLAS
17/12/07 20:35:34 WARN LinearRegressionWithSGD: The input data was not directly
cached, which may hurt performance if its parent RDDs are also uncached.
lr_model: org.apache.spark.mllib.regression.LinearRegressionModel = org.apache.s
park.mllib.regression.LinearRegressionModel: intercept = 0.0, numFeatures = 8

scala>
```

```
val lr_l1_model = LassoWithSGD.train(lprdd, numIterations = 10, regParam = 0.2, stepSize = 1.0)
```

```
scala> val lr_l1_model = LassoWithSGD.train(lprdd, numIterations = 10, regParam
= 0.2, stepSize = 1.0)
warning: there was one deprecation warning; re-run with -deprecation for details
17/12/07 20:36:29 WARN LassoWithSGD: The input data is not directly cached, whic
h may hurt performance if its parent RDDs are also uncached.
17/12/07 20:36:29 WARN LassoWithSGD: The input data was not directly cached, whi
ch may hurt performance if its parent RDDs are also uncached.
lr_l1_model: org.apache.spark.mllib.regression.LassoModel = org.apache.spark.mll
ib.regression.LassoModel: intercept = 0.0, numFeatures = 8

scala>
```

```
val lr_l2_model = RidgeRegressionWithSGD.train(lprdd, numIterations = 10, regParam = 0.2, stepSize = 1.0)
```

```
scala> val lr_l2_model = RidgeRegressionWithSGD.train(lprdd, numIterations = 10,
 regParam = 0.2, stepSize = 1.0)
warning: there was one deprecation warning; re-run with -deprecation for details
17/12/07 20:36:51 WARN RidgeRegressionWithSGD: The input data is not directly ca
ched, which may hurt performance if its parent RDDs are also uncached.
17/12/07 20:36:51 WARN RidgeRegressionWithSGD: The input data was not directly c
ached, which may hurt performance if its parent RDDs are also uncached.
lr_l2_model: org.apache.spark.mllib.regression.RidgeRegressionModel = org.apache
.spark.mllib.regression.RidgeRegressionModel: intercept = 0.0, numFeatures = 8

scala>
```

- Create an RDD of LabeledPoint named test_lprdd from the data in the concrete_test.csv file. Use the same methods that were used to create lprdd, but instead of training a new StandardScalerModel, use the previously created StandardScalerModel called model just to transform this new test data set (in other words, you will not need to call the fit method during this step).

```
// TODO:

// Create a new RDD of labeled points called "test_lprdd" from the "concrete_test.csv" data

// using the same exact method used to create the "lprdd" RDD above.
```

```
// Be sure to use the same StandardScaler model used to scale the training data,

// i.e. do not create a new instance of StandardScaler for this step.


val test_rawdata = sc.textFile("/user/cloudera/mldata/concrete_test.csv")
```

```
scala> val test_rawdata = sc.textFile("/user/cloudera/mldata/concrete_test.csv")
test_rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/concrete_test.csv MapPartitionsRDD[7] at textFile at <console>:28

scala>
```

```
val test_vecrdd = test_rawdata.map(x => x.split(",").map(_.toDouble)).map(x => Vectors.dense(x))
```

```
scala> val test_vecrdd = test_rawdata.map(x => x.split(",").map(_.toDouble)).map
(x => Vectors.dense(x))
test_vecrdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = Ma
pPartitionsRDD[275] at map at <console>:32

scala>
```

```
val test_scaled = model.transform(test_vecrdd)
```

```
scala> val test_scaled = model.transform(test_vecrdd)
test_scaled: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = Ma
pPartitionsRDD[276] at map at VectorTransformer.scala:52

scala>
```

```
val test_lprdd = test_scaled.map{x =>
        val temp = x.toArray
        val label = temp.last
        val features = Vectors.dense(temp.dropRight(1))
        LabeledPoint(label, features)
}
```

```
scala> val test_lprdd = test_scaled.map{x =>
     |         val temp = x.toArray
     |         val label = temp.last
     |         val features = Vectors.dense(temp.dropRight(1))
     |         LabeledPoint(label, features)
     | }
test_lprdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPo
int] = MapPartitionsRDD[277] at map at <console>:44
```

- Finally, use the predict method of the three linear regression models created previously to create predictions on the features of the test_lprdd data. Note here that you'll have to map the test_lprdd to only the features before using as input to the predict method of the models. Save the predictions to three new RDDs called pred_lr, pred_lr_l1, and pred_lr_l2. Invoke the collect method on each of the prediction RDDs to see the results.

```
// TODO:

// Create RDDs of predictions for all three models

val pred_lr = lr_model.predict(test_lprdd.map(x =>
x.features))
```

```
scala> val pred_lr = lr_model.predict(test_lprdd.map(x => x.features))
pred_lr: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[279] at mapPartitio
ns at GeneralizedLinearAlgorithm.scala:70

scala>
```

```
pred_lr.collect
```

```
scala> pred_lr.collect
res38: Array[Double] = Array(0.13396512737030666, -0.32824386638219344, 0.205827
65432480285, -0.685201423278049, -0.582426128088622, -0.6492038317139167, 0.2531
2623528405365, 0.6236983592157614, 0.8541416040249497, -0.6883886459101022, -0.3
325367038920255, -0.4661648152866765, 0.14553887774657073, 0.30699801139994554,
0.37341960847975697, -0.07721653430942847, -0.140562949811266, -0.58379989310386
6, -0.989847557394648, -0.2670688697779736, -0.043138564327351, -0.8750249892009
23, -0.1532038552091455, -0.0653383057397072, -0.49477102827835917, -0.150511415
75816243, 0.06475864524943042, -0.02277750071547277, -0.019014793698203025, -0.3
9527328118172644, -0.056738767962598025, -0.4806226210021596, -0.414603697139698
8, 0.21577025494874671, -0.42562753488695826, -0.30366967125987254, -0.930140...
scala>
```

```
val pred_lr_l1 = lr_l1_model.predict(test_lprdd.map(x =>
x.features))
```

```
scala> val pred_lr_l1 = lr_l1_model.predict(test_lprdd.map(x => x.features))
pred_lr_l1: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[281] at mapParti
tions at GeneralizedLinearAlgorithm.scala:70

scala>
```

```
pred_lr_l1.collect
```

```
scala> pred_lr_l1.collect
res39: Array[Double] = Array(-0.1569046188138531, -0.30530310124995724, 0.118402
23152802574, -0.24667834341424222, -0.24667834341424222, -0.25829350705509124, 0
.038526806281356754, 0.07988356491184549, 0.07761776099324665, -0.46927606706839
85, -0.33900401598108504, -0.060333482457667356, -0.037652094807439376, 0.212135
27586990294, 0.07761776099324665, -0.24667834341424222, -0.04416336251538108, -0
.4984453739623286, -0.30303729733135837, 0.07761776099324665, -0.256027703136492
4, -0.24667834341424222, -0.12120069574774162, -0.19711680125292258, -0.39083136
206100355, -0.2599870228779805, -0.19711680125292258, -0.24214673557704453, -0.2
93974081656963, -0.39309716597960237, -0.3050169572021024, -0.37270493071221283,
 -0.31634597679509663, -0.2868905258533117, -0.28207277396825925, -0.31181436...
scala>
```

```
val pred_lr_l2 = lr_l2_model.predict(test_lprdd.map(x =>
x.features))
```

```
scala> val pred_lr_l2 = lr_l2_model.predict(test_lprdd.map(x => x.features))
pred_lr_l2: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[283] at mapParti
tions at GeneralizedLinearAlgorithm.scala:70

scala>
```

```
pred_lr_l2.collect
```

```
scala> pred_lr_l2.collect
res40: Array[Double] = Array(0.11600971594865228, -0.2760791792729472, 0.2170131
3329063243, -0.5661674251978508, -0.45439763925570276, -0.5403201764490984, 0.22
612350058480013, 0.5201465784472996, 0.6873858755238433, -0.5923667417471264, -0
.29592866938241313, -0.3525565629834626, 0.1264917600377837, 0.28469310730937536
, 0.3275495477473256, -0.07054221125941208, -0.08992793973812124, -0.52067297648
85198, -0.8067398971090892, -0.1841894633804831, -0.05198116825240465, -0.708047
6842648914, -0.1214670658158461, -0.04278830537866968, -0.41814041514928973, -0.
13461457652642767, 0.05554190296172329, -0.021142930877250776, -0.04301729783927
38, -0.3599933960367445, -0.06333806403068723, -0.43898158934861564, -0.34096093
810814504, 0.14085851125428756, -0.3612270209400117, -0.2560502366770091, -0....
scala>
```

Applying Transformations with Spark ML

Files and Data Used in This Exercise

```
Local data /home/cloudera/dataset/concrete.csv

HDFS path /user/cloudera/mldata
```

In this exercise, you will create a DataFrame and use Spark ML to perform transformations on it.

- Open either the Scala Spark REPL:
- Begin by importing the necessary modules for this exercise.

  If you're using the Scala REPL, you'll use this code:

```
import org.apache.spark.ml.linalg.Vectors

import org.apache.spark.ml.feature.{StandardScaler,
PolynomialExpansion}

import org.apache.spark.ml.Pipeline

import org.apache.spark.sql.Row
```

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.ml.feature.{StandardScaler, PolynomialExpansion}
import org.apache.spark.ml.feature.{StandardScaler, PolynomialExpansion}

scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala>
```

- Next, load the concrete.csv file from HDFS into an RDD named rawdata using the textFile method of the SparkContext instance.

```
// TODO:

// Load the raw data from concrete.csv into the "rawdata" RDD

val rawdata =
sc.textFile("/user/cloudera/mldata/concrete.csv")
```

```
scala> val rawdata = sc.textFile("/user/cloudera/mldata/concrete.csv")
rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/concrete.csv MapPartitionsRDD[1] at textFile at <console>:28
```

- Convert the rawdata RDD to an RDD called splits by splitting on commas and then casting each value as a Double in Scala or a float in Python.

```
// TODO:

// Split the rawdata rdd on commas, convert the elements to
Double

// and save in an rdd called "splits"

val splits = rawdata.map(x => x.split(",").map(_.toDouble))
```

```
scala> val splits = rawdata.map(x => x.split(",").map(_.toDouble))
splits: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[286] at map a
t <console>:36

scala>
```

- Use the splits RDD to create a Spark DataFrame called df with two columns: label and features. The label column should contain only the last element of the Scala array/Python list contained in splits. The features column should contain dense vectors, with all but the last element of the Scala array/Python list contained in splits.

  In Scala:

  First create a case class called line that will then be used to map the elements of the splits RDD:

```
// Create a case class called line that will

// be used to create a DataFrame from splits.

// The case class should have two elements,

// "label" which should be a Double and "features"

// which should be a dense vector

case class line(label: Double, features:
org.apache.spark.ml.linalg.Vector)
```

```
scala> case class line(label: Double, features: org.apache.spark.mllib.linalg.Ve
ctor)
defined class line
```

  Next, use this case class to create a DataFrame with columns label and features. Make sure to explicitly cast the features as a dense vector before using the line case class. Recall that the toDF method of an RDD[case class] will convert the RDD to a DataFrame.

```
// TODO:

// Use the newly created case class above in a map to map the
last
```

```
// element of the arrays in "splits" to the "label" field, and
the remaining

// elements to the "features" field. Make sure to cast the
features to a dense

// vector before using in the case class.

val df = splits.map{x =>

    val label = x.last

    val features = Vectors.dense(x.dropRight(1))

    line(label, features)

}.toDF()
```

```
scala> val df = splits.map{x =>
     |     val label = x.last
     |     val features = Vectors.dense(x.dropRight(1))
     |     line(label, features)
     | }.toDF()
df: org.apache.spark.sql.DataFrame = [label: double, features: vector]

scala>
```

- Create an instance of StandardScaler called ss, setting the withStd and withMean parameters to true, the inputCol parameter to features, and the outputCol parameter to scaledfeatures. Recall that these parameters are set in Scala using associated set methods, and in Python as arguments to the StandardScaler constructor.

```
// TODO:

// Create an instance of "StandardScaler" named "ss" to scale

// the "features" column and output this to a new column

// called "scaledfeatures"

val ss = new
StandardScaler().setWithStd(true).setWithMean(true).setInputCo
l("features").setOutputCol("scaledfeatures")
```

```
scala> val ss = new StandardScaler().setWithStd(true).setWithMean(true).setInput
Col("features").setOutputCol("scaledfeatures")
ss: org.apache.spark.ml.feature.StandardScaler = stdScal_a5b8c94e92a3
```

- Create an instance of PolynomialExpansion called pe, with the degree parameter set to 2, inputCol set to scaledfeatures and outputCol set to expandedfeatures.

```
// TODO:

// Create an instance of "PolynomialExpansion" named "pe" to
transform

// the "scaledfeatures" column and output this to a new column
called

// "expandedfeatures" (set "degree" = 2 when initializing).

val pe = new
PolynomialExpansion().setDegree(2).setInputCol("scaledfeatures
").setOutputCol("expandedfeatures")
```

```
scala> val pe = new PolynomialExpansion().setDegree(2).setInputCol("scaledfeatur
es").setOutputCol("expandedfeatures")
pe: org.apache.spark.ml.feature.PolynomialExpansion = poly_91a99b0c6782

scala>
```

- Create an instance of Pipeline called pl, and set the stages parameter to be a Scala Array or Python list containing ss followed by pe.

```
// TODO:

// Create a Pipeline estimator

// Use "ss" as the first stage and "pe" as the second stage.

val pl = new Pipeline().setStages(Array(ss,pe))
```

```
scala> val pl = new Pipeline().setStages(Array(ss,pe))
pl: org.apache.spark.ml.Pipeline = pipeline_eabe2e99abbe

scala>
```

- Call the fit method of the Pipeline transformer pl with the DataFrame df as an argument, and save the resulting PipelineModel into a variable called model.

```
// TODO:
// Call the "fit" method of the "pl" estimator on "df" to create a
// "PipelineModel" and save it in a variable called "model"
val model = pl.fit(df)
```

```
scala> val model = pl.fit(df)
model: org.apache.spark.ml.PipelineModel = pipeline_eabe2e99abbe

scala>
```

- Call the transform method of model with df as an argument, and save the resulting DataFrame as transformed. Finally, print the columns attribute of the transformed DataFrame to confirm that the transformed columns exist.

```
// TODO:
// Create a new DataFrame called "transformed" by calling the "transform"
// method of "model" on "df" and then print the columns using the "columns"
// attribute of the "transformed" dataframe.
val transformed = model.transform(df)
println(transformed.columns)
```

```
scala> val transformed = model.transform(df)
transformed: org.apache.spark.sql.DataFrame = [label: double, features: vector .
.. 2 more fields]
```

```
scala> transformed.columns
res2: Array[String] = Array(label, features, scaledfeatures, expandedfeatures)

scala> transformed.take(4)
res3: Array[org.apache.spark.sql.Row] = Array([79.99,[540.0,0.0,0.0,162.0,2.5,10
40.0,676.0,28.0],[2.476711702426232,-0.856471824489096,-0.8467325968146491,-0.91
63192506346087,-0.6201470629660243,0.8627351348572722,-1.2170787967116363,-0.279
5972873837829],[2.476711702426232,6.134100856935045,-0.856471824489096,-2.121233
79051049,0.7335439861436808,-0.8467325968146491,-2.097112531356594,0.72520261204
82326,0.7169560905084791,-0.9163192506346087,-2.269458611205171,0.78480162040550
46,0.7758773786010955,0.8396409690835709,-0.6201470629660243,-1.5359254880732098
,0.5311384864700651,0.5250987330321994,0.5682526920202808,0.3845823797053861,0.8
627351348572722,2.1367462045952794,-0.7389083350020541,-0.7305059611009346,-0.79
05408122685638,-0.5350226599993342,0.7443119129171956,-1.2170787967116363,-3....
scala>
```

Using Decision Tree Classifiers with Spark ML

Files and Data Used in This Exercise

Local data /home/cloudera/dataset/cars_train.csv
/user/cloudera/dataset/cars_test.csv

HDFS path /user/cloudera/mldata

In this exercise, you will use Spark ML to build a decision tree classifier.

Preparing the Data During this exercise, you will build a Spark ML Pipeline to encode categorical string variables as integers before using them to build a model. The data used for this exercise concerns various properties of cars, and whether these cars were classified as a good value. The target value to be predicted is acceptability, which is a categorical variable representing whether a car is considered acceptable for purchase. All other feature variables are also categorical.

#	Field	Data Type	Description
0	buying	String	Based on selling price
1	maint	String	Based on cost to maintain the vehicle
2	doors	String	Number of doors
3	persons	String	Passenger capacity
4	lug_boot	String	Based on luggage boot size
5	safety	String	Based on estimated safety of the vehicle
6	acceptability	String	Based on overall acceptability of the vehicle

- Open a new terminal window and change to the project data directory:

```
$ cd /user/cloudera/dataset
```

- Add the two data files used in this exercise to the /user/cloudera/mldata directory in HDFS:

```
$ hdfs dfs -put cars_train.csv cars_test.csv mldata
```

Building the Model

- Open either the Scala Spark REPL:
- Begin by importing the necessary modules for this exercise. If you're using the Scala REPL, you'll use this code:

```
import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.ml.Pipeline import
org.apache.spark.sql.Row

import org.apache.spark.ml.feature.{StringIndexer,
VectorAssembler}

import
org.apache.spark.ml.classification.DecisionTreeClassifier
```

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.ml.Pipeline import org.apache.spark.sql.Row
<console>:1: error: ';' expected but 'import' found.
import org.apache.spark.ml.Pipeline import org.apache.spark.sql.Row
                                  ^

scala> import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

scala> import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.classification.DecisionTreeClassifier

scala>
```

- Create an RDD called rawdata that contains the contents of the cars_train.csv file you uploaded to HDFS in an earlier step.

```
// Load the data from the delimited file 'cars_train.csv' into the rdd rawdata

val rawdata = sc.textFile("/user/cloudera/mldata/cars_train.csv")
```

```
scala> val rawdata = sc.textFile("/user/cloudera/mldata/cars_train.csv")
rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/cars_train.csv MapPartitionsRDD[9] at textFile at <console>:28

scala>
```

- Map rawdata to a new RDD of Scala arrays or Python lists called lrdd by splitting on commas.

```
// Map the RDD of strings to an RDD of Arrays by splitting on commas

val lrdd = rawdata.map(x => x.split(","))
```

```
scala> val lrdd = rawdata.map(x => x.split(","))
lrdd: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[290] at map at
<console>:39

scala>
```

- Next, create an RDD representing each row of input data.

In Scala:

You will need to create a new case class for each row of the input data, and then create an RDD of these new classes called rowrdd. All types should be strings.

```
// Create a new case class for each row of the input data,
// then create an RDD of these new classes called rowrdd.
// Recall that the rownames are buying, maint, doors, persons, lugboots, safety, label.
// All types should be strings.
// Also recall that the syntax for specifying a case class is
// e.g.,
// case class line(buying: String, maint: String, ...)
case class line(buying: String, maint: String, doors: String, persons: String, lugboots: String, safety: String, label: String)
```

```
scala> case class line(buying: String, maint: String, doors: String, persons: St
ring, lugboots: String, safety: String, label: String)
defined class line
```

```
val rowrdd = lrdd.map(x =>
    line(x(0), x(1), x(2), x(3), x(4), x(5), x(6))
)
```

```
scala> val rowrdd = lrdd.map(x =>
     |     line(x(0), x(1), x(2), x(3), x(4), x(5), x(6))
     | )
rowrdd: org.apache.spark.rdd.RDD[line] = MapPartitionsRDD[291] at map at <consol
e>:43

scala>
```

- Convert the rowrdd from the previous step to a DataFrame called train_df.

```
// Convert rowrdd to a DataFrame, called "train_df"

val train_df = rowrdd.toDF()
```

```
scala> val train_df = rowrdd.toDF()
train_df: org.apache.spark.sql.DataFrame = [buying: string, maint: string ... 5
more fields]

scala>
```

- Create a new StringIndexer transformer for each of the columns, with the output column name the same as the input columns name with "_ix" appended to it (for example, this_col becomes this_col_ix). Also create a Scala array or Python list called indexedcols that contains all the indexed column names ending in "_ix" except the label_ix column. Save these seven transformers as variables called si1, si2, si3, and so on.

```
// TODO:

// Create 7 distinct StringIndexer transformers with the
outputCol

// parameter set to be the name of the input column appended
with the string "_ix"

//

// e.g.,:

// val MyStringIndexer = new
StringIndexer.setInputCol("thiscol").setOutputCol("thiscol_ix"
)

//

// Additionally, store the names of the transformed feature
columns

// (excluding the "label_ix" column) in an Array named
"indexedcols"

val indexedcols = Array("buying_ix", "maint_ix", "doors_ix",
"persons_ix", "lugboots_ix", "safety_ix")
```

```
scala> val indexedcols = Array("buying_ix", "maint_ix", "doors_ix", "persons_ix'
, "lugboots_ix", "safety_ix")
indexedcols: Array[String] = Array(buying_ix, maint_ix, doors_ix, persons_ix, lu
gboots_ix, safety_ix)

scala>
```

```
val si1 = new
StringIndexer().setInputCol("buying").setOutputCol("buying_ix"
)

val si2 = new
StringIndexer().setInputCol("maint").setOutputCol("maint_ix")

val si3 = new
StringIndexer().setInputCol("doors").setOutputCol("doors_ix")

val si4 = new
StringIndexer().setInputCol("persons").setOutputCol("persons_i
x")

val si5 = new
StringIndexer().setInputCol("lugboots").setOutputCol("lugboots
_ix")

val si6 = new
StringIndexer().setInputCol("safety").setOutputCol("safety_ix"
)

val si7 = new
StringIndexer().setInputCol("label").setOutputCol("label_ix")
```

```
scala> val si1 = new StringIndexer().setInputCol("buying").setOutputCol("buying_
ix")
si1: org.apache.spark.ml.feature.StringIndexer = strIdx_072d6d1acb17

scala> val si2 = new StringIndexer().setInputCol("maint").setOutputCol("maint_ix
")
si2: org.apache.spark.ml.feature.StringIndexer = strIdx_c0a17747c7a0

scala> val si3 = new StringIndexer().setInputCol("doors").setOutputCol("doors_ix
")
si3: org.apache.spark.ml.feature.StringIndexer = strIdx_ec5d81acf09c

scala> val si4 = new StringIndexer().setInputCol("persons").setOutputCol("person
s_ix")
si4: org.apache.spark.ml.feature.StringIndexer = strIdx_5062b9d429d1

scala> val si5 = new StringIndexer().setInputCol("lugboots").setOutputCol("lugbo
ots_ix")
si5: org.apache.spark.ml.feature.StringIndexer = strIdx_d601d090a279

scala> val si6 = new StringIndexer().setInputCol("safety").setOutputCol("safety_
ix")
si6: org.apache.spark.ml.feature.StringIndexer = strIdx_b62e1d150aa1
```

- Next, create a VectorAssembler transformer to convert each of the columns in the Scala array or Python list indexedcols into a single column called features. Name the new VectorAssembler as va.

```
// TODO:

// Create a VectorAssembler transformer to combine all of the
indexed

// categorical features into a vector. Provide the
"indexedcols" Array created above

// as the inputCols parameter, and name the outputCol
"features".

val va = new
VectorAssembler().setInputCols(indexedcols).setOutputCol("feat
ures")
```

```
scala> val va = new VectorAssembler().setInputCols(indexedcols).setOutputCol("fe
atures")
va: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_40c5da4a7db3

scala>
```

- Create a DecisionTreeClassifier setting the label column to the indexed label column label_ix and the features column to the newly created features column from the VectorAssembler. Then create a Scala array or Python list called steps containing all the StringIndexer transformers, then the VectorAssembler, then the DecisionTreeClassifier.

```
// TODO:

// Create a DecisionTreeClassifier, setting the label column
to your

// indexed label column ("label_ix") and the features column
to the

// newly created column from the VectorAssembler above
("features").

// Store the new StringIndexer transformers, the
VectorAssembler,

// as well as the DecisionTreeClassifier in an Array called
"steps"
```

```
val clf = new DecisionTreeClassifier().setLabelCol("label_ix")
```

```
scala> val clf = new DecisionTreeClassifier().setLabelCol("label_ix")
clf: org.apache.spark.ml.classification.DecisionTreeClassifier = dtc_3eefddfd4c4
7
```

```
val steps = Array(si1, si2, si3, si4, si5, si6, si7, va, clf)
```

```
scala> val steps = Array(si1, si2, si3, si4, si5, si6, si7, va, clf)
steps: Array[org.apache.spark.ml.PipelineStage with org.apache.spark.ml.util.Def
aultParamsWritable{def copy(extra: org.apache.spark.ml.param.ParamMap): org.apac
he.spark.ml.PipelineStage with org.apache.spark.ml.util.DefaultParamsWritable{de
f copy(extra: org.apache.spark.ml.param.ParamMap): org.apache.spark.ml.PipelineS
tage with org.apache.spark.ml.util.DefaultParamsWritable}}] = Array(strIdx_072d6
d1acb17, strIdx_c0a17747c7a0, strIdx_ec5d81acf09c, strIdx_5062b9d429d1, strIdx_d
601d090a279, strIdx_b62e1d150aa1, strIdx_033c5b6abd1b, vecAssembler_40c5da4a7db3
, dtc_3eefddfd4c47)

scala>
```

- Create a new Spark ML Pipeline called pl setting the stages parameter with the steps Scala array or Python list created above.

```
// TODO:
// Create a ML pipeline named "pl" using the steps list to set
the stages parameter
val pl = new Pipeline().setStages(steps)
```

```
scala> val pl = new Pipeline().setStages(steps)
pl: org.apache.spark.ml.Pipeline = pipeline_af8f1dada8f4

scala>
```

- Create a PipelineModel named plmodel by calling the fit method of the pl created in the previous step on train_df.

```
// TODO:
// Run the fit method of the pipeline on the DataFrame
// "train_df" to create a pipeline model, and save the
// model in a new variable called "plmodel"
```

```
val plmodel = pl.fit(train_df)
```

```
scala> val plmodel = pl.fit(train_df)
plmodel: org.apache.spark.ml.PipelineModel = pipeline_af8f1dada8f4

scala>
```

- Create a new DataFrame called test_df from the cars_test.csv dataset. Use the exact same method used to create the train_df DataFrame.

```
// TODO:

// Create a new DataFrame called "test_df" from the
cars_test.csv data.

// using the same exact method used to create the "train_df"
DataFrame above.

val test_df =
sc.textFile("/user/cloudera/mldata/cars_test.csv").map(x =>
x.split(',')).map(x =>

    line(x(0), x(1), x(2), x(3), x(4), x(5), x(6))

    ).toDF()
```

```
scala> val test_df = sc.textFile("/user/cloudera/mldata/cars_test.csv").map(x => x.split(',')).map(x => line(x(0), x(1), x(2), x(3), x(4), x(5), x(6))
     | ).toDF()
test_df: org.apache.spark.sql.DataFrame = [buying: string, maint: string ... 5 more fields]

scala>
```

- Call the transform method of plmodel on the test_df DataFrame to create a new DataFrame with predictions. Save the new DataFrame as predictions. How many of the first 15 values in the prediction column match the values in the label_ix column?

```
// TODO:

// Run the transform method of the pipeline model created
above

// on the "test_df" DataFrame to create a new DataFrame called
"predictions"

val predictions = plmodel.transform(test_df)
```

```
scala> val predictions = plmodel.transform(test_df)
predictions: org.apache.spark.sql.DataFrame = [buying: string, maint: string ...
 16 more fields]

scala>
```

You can display the data needed to answer the question using the Python code below, which displays the output that follows it. As you can see, there are twelve matching values (and three non-matching values, shown here in bold):

```
predictions.select(predictions['label_ix'],
predictions['prediction']).show(15)
```

Scala code is:

```
// Compare the first 15 values in the "prediction" column with
those

// rows' values in the "label_ix" column

predictions.select("label_ix", "prediction").show(15)
```

```
scala> predictions.select("label_ix", "prediction").show(15)
+--------+----------+
|label_ix|prediction|
+--------+----------+
|     0.0|       0.0|
|     1.0|       1.0|
|     2.0|       1.0|
|     0.0|       0.0|
|     1.0|       1.0|
|     1.0|       1.0|
|     0.0|       0.0|
|     1.0|       1.0|
|     2.0|       1.0|
|     0.0|       0.0|
|     1.0|       1.0|
|     2.0|       1.0|
|     0.0|       0.0|
|     0.0|       0.0|
|     0.0|       0.0|
+--------+----------+
only showing top 15 rows


scala>
```

Clustering with k-Means in Spark ML

```
Files and Data Used in This Exercise

Local data /home/cloudera/dataset/wine.csv

HDFS path /user/cloudera/mldata
```

In this exercise, you will use Spark ML to implement clustering with the k-means algorithm.

The data used in this exercise contains the following physical properties of wines, which you will use to cluster the data into a specific number of groups:

#	Field	Data Type	Description
0	alcohol	Double	Alcohol content of wine
1	malic_acid	String	Malic acid content of wine
2	ash	Double	Ash content of wine
3	alcalinity	Double	Alcalinity of wine ash
4	magnesium	Double	Magnesium content of wine
5	tot_phenols	Double	Total phenol content of wine
6	flavanoids	Double	Flavanoid content of wine
7	non_flav_phenols	Double	Non-flavanoid phenol content of wine
8	proanthocyanins	Double	Proanthocyanin content of wine
9	color	Double	Color intensity of wine
10	hue	Double	Hue of wine
11	od280_od315	Double	OD280/OD315 measure of wine (the ratio between absorbances at 280 and 315 nm)
12	proline	Double	Proline content of wine

- Open a new terminal window and change to the project data directory:

```
$ cd /home/cloudera/dataset
```

- Add the data file used in this exercise to the /user/cloudera/mldata directory in HDFS:

```
$ hdfs dfs -put wine.csv mldata
```

- Open either the Scala Spark REPL:
- Begin by importing the necessary modules for this exercise. If you're using the Scala REPL, you'll use this code:

```
import org.apache.spark.ml.linalg.Vectors

import org.apache.spark.ml.feature.StandardScaler

import org.apache.spark.ml.Pipeline

import org.apache.spark.sql.Row

import org.apache.spark.ml.clustering.KMeans
```

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.ml.feature.StandardScaler
import org.apache.spark.ml.feature.StandardScaler

scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.clustering.KMeans

scala>
```

- Create an RDD called rawdata containing the contents of the wine.csv file you previously added to HDFS.

```
Load the data from the delimited file "wine.csv" into the rdd
rawdata

val rawdata = sc.textFile("/user/cloudera/mldata/wine.csv")
```

```
scala> val rawdata = sc.textFile("/user/cloudera/mldata/wine.csv")
rawdata: org.apache.spark.rdd.RDD[String] = /user/cloudera/mldata/wine.csv MapPartitionsRDD[15] at textFile at <console>:28

scala>
```

- Map rawdata to a new RDD of dense vectors called vecrdd.

```
// Create an RDD of dense vectors by splitting on commas,
casting to float,

// then casting to a dense vector

val vecrdd = rawdata.map(x =>
x.split(",").map(_.toDouble)).map(x => Vectors.dense(x))
```

```
scala> val vecrdd = rawdata.map(x => x.split(",").map(_.toDouble)).map(x => Vect
ors.dense(x))
vecrdd: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPart
itionsRDD[386] at map at <console>:44
```

Convert vecrdd to a Spark DataFrame called df with a single column, called features. If you're using Scala, this will require creating a case class for the Vector elements of vecrdd (this is not necessary if you're using Python):

```
// Create a case class called line which contains a single
element called features

case class line(features:org.apache.spark.ml.linalg.Vector)
```

```
scala> case class line(features:org.apache.spark.ml.linalg.Vector)
defined class line

scala>
```

```
// Convert to a DataFrame with a single column called
"features"

// using the case class create above.

val df = vecrdd.map(x => line(x)).toDF()
```

```
scala> val df = vecrdd.map(x => line(x)).toDF()
df: org.apache.spark.sql.DataFrame = [features: vector]

scala>
```

- Create a StandardScaler transformer called ss that will scale and center the data, setting the input column to features and the output column to scaled.

```
// Create a StandardScaler transformer that will

// scale and center the data, taking as input the

// features column and setting the output column

// to be called "scaled"

val ss = new
StandardScaler().setWithMean(true).setWithStd(true).setInputCo
l("features").setOutputCol("scaled")
```

```
scala> val ss = new StandardScaler().setWithMean(true).setWithStd(true).setInput
Col("features").setOutputCol("scaled")
ss: org.apache.spark.ml.feature.StandardScaler = stdScal_cb963e0ffeac

scala>
```

- Create a KMeans estimator called km, setting the features column to scaled and setting the k parameter to be three.

```
// Create a KMeans estimator, setting the featuresCol

// to 'scaled' and setting 3 clusters

val km = new KMeans().setK(3).setFeaturesCol("scaled")
```

```
scala> val km = new KMeans().setK(3).setFeaturesCol("scaled")
km: org.apache.spark.ml.clustering.KMeans = kmeans_b3d552ecb090

scala>
```

- Create a new Pipeline called pl with two stages, ss and km.

```
// Create a pipeline with two stages: the StandardScaler
```

```
// transformer and the KMeans estimator created above

val pl = new Pipeline().setStages(Array(ss,km))
```

```
scala> val pl = new Pipeline().setStages(Array(ss,km))
pl: org.apache.spark.ml.Pipeline = pipeline_da718ff52861

scala>
```

- Using pl, chain the fit and transform methods to produce a new DataFrame called clusters that contains the prediction for each of the data points. This should be done in a single line.

```
// Chain methods of the pipeline (and the resulting pipeline model)

// to create an RDD of predictions called clustered using only one line.

val clustered = pl.fit(df).transform(df)
```

```
scala> val clustered = pl.fit(df).transform(df)
17/12/08 11:00:19 WARN BLAS: Failed to load implementation from: com.github.fomm
il.netlib.NativeSystemBLAS
17/12/08 11:00:19 WARN BLAS: Failed to load implementation from: com.github.fomm
il.netlib.NativeRefBLAS
clustered: org.apache.spark.sql.DataFrame = [features: vector, scaled: vector ..
. 1 more field]

scala>
```

- The data used for this exercise is composed of three distinct groups, which are distributed as follows:

    a. The first 59 entries belong to group one
    b. The next 71 entries belong to group two
    c. The last 48 entries belong to group three

Collect all the predictions to the driver and then print them to the screen. Is the above grouping captured by the k-means clustering algorithm?

```
// Select only the prediction column, collect all
```

```
// predictions to the driver and print them to screen

clustered.select("prediction").collect().foreach(println)
```

```
scala> clustered.select("prediction").collect().foreach(println)
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
```