

Contents

Exercise 1: Setting up the Environment	1
Exercise 2: Practicing HDFS Commands.....	4
Exercise 3: Running MapReduce Job	7
Exercise 4: Launching Spark	23
Exploring Data using RDD operations:	26
Transformations.....	26
Action.....	31
PairRDD	35
Developing with Spark	42
REPL	42
Additional Exercise	43
Analyze Movie Lens Dataset using RDD	43
Exercise 5: Dataframe and Spark SQL.....	49
Infer schema by Reflection (case class).....	49
Programmatically	54
UDF in Spark Dataframe	55
Exercise 6: Spark Streaming.....	60

Exercise 1: Setting up the Environment

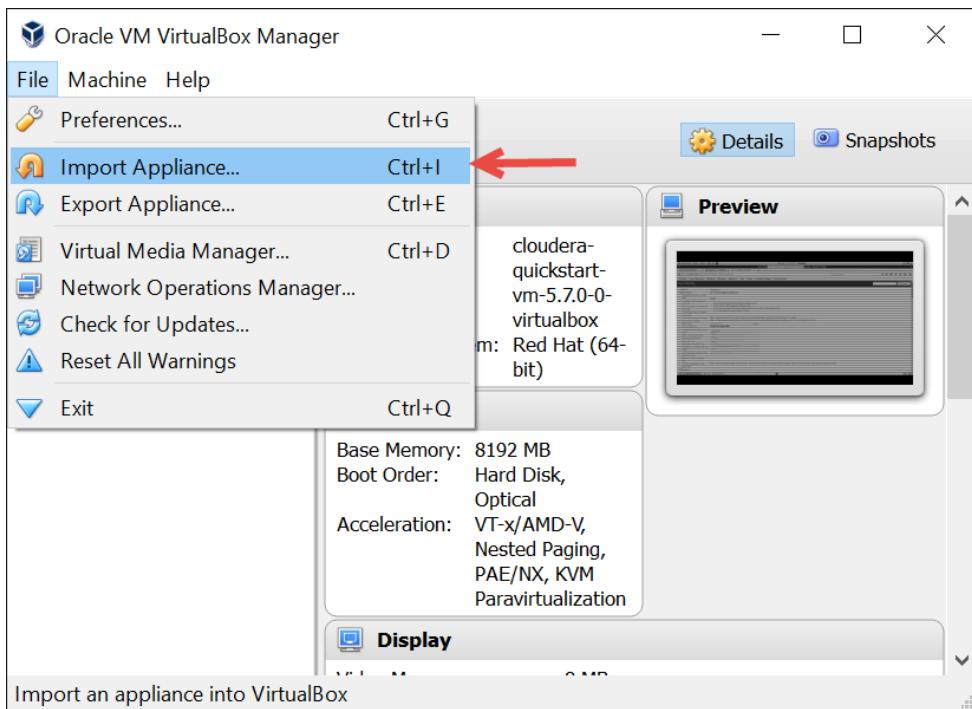
Step1: Installing Oracle VM virtual box from Training Bundle.

- Under ‘Training Bundle’ folder, navigate to “Software” folder.
- Find the executable file named ‘VirtualBox-5.0.16-105871-Win’ and complete the installation.

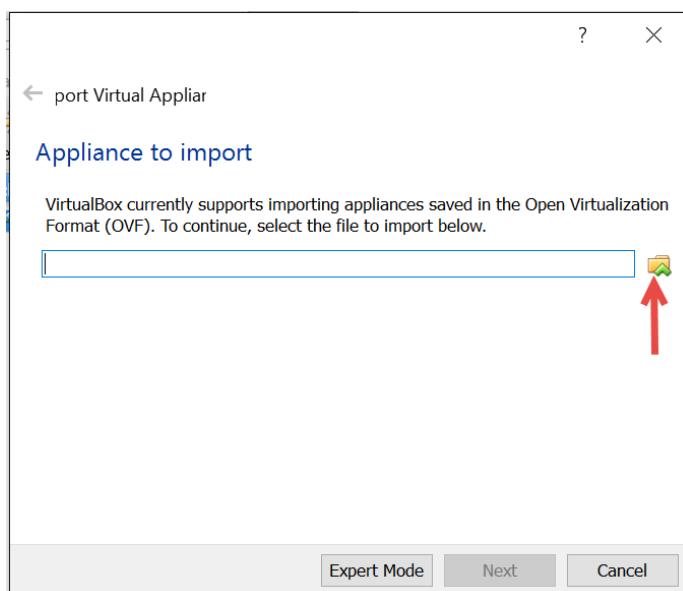
Step2: Extracting the Image

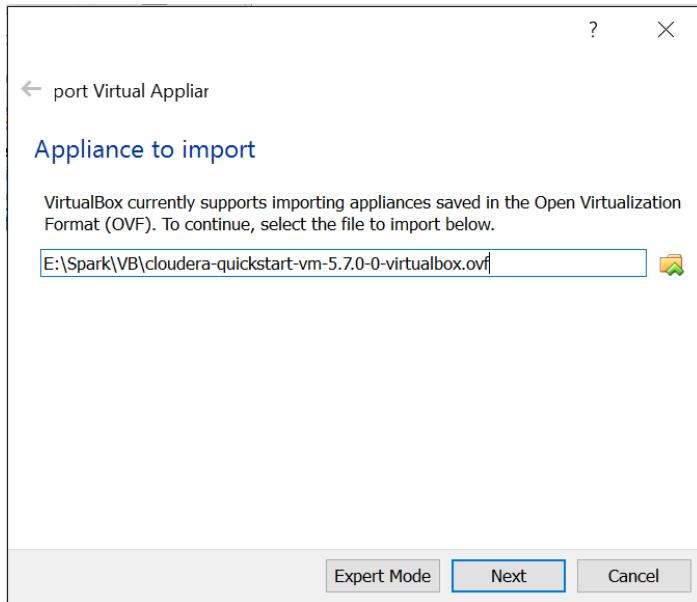
- From the same training bundle, locate the folder named VM image> ‘cloudera-quickstart-vm-5.10.0-0-vmware’
- Load the image to get started with the machine.

Step3: Importing the file



- Browse to the location under “Training Bundle” and select the OVF file.





- This will prepare your machine.
- After import, change the number of processors to two before starting the machine.

- Verify all the running services with jps command

```
[cloudera@quickstart ~]$ sudo jps
6403 RESTServer
5709 SecondaryNameNode
5980 NodeManager
6878 RunJar
8543
10329 ZeppelinServer
9721 org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
5841 Bootstrap
5366 DataNode
5454 JournalNode
8584
7516 HistoryServer
8518 Bootstrap
5573 NameNode
7484 Bootstrap
5303 QuorumPeerMain
6659 RunJar
5897 JobHistoryServer
18565 Jps
6184 ResourceManager
6521 ThriftServer
8191 Bootstrap
[cloudera@quickstart ~]$
```

Exercise 2: Practicing HDFS Commands

Task 1: Using Hadoop Command Line Interface

Navigate to Application > System Tools > Terminal

OR

Use shortcut to open the terminal.



➤ Operation on Files and directories

To check the content of root directory in HDFS

```
hdfs dfs -ls /
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 5 items
drwxrwxrwx  - hdfs  supergroup          0 2016-04-05 23:56 /benchmarks
drwxr-xr-x  - hbase  supergroup          0 2016-09-04 10:14 /hbase
drwxrwxrwt  - hdfs  supergroup          0 2016-08-03 01:56 /tmp
drwxr-xr-x  - hdfs  supergroup          0 2016-08-02 08:08 /user
drwxr-xr-x  - hdfs  supergroup          0 2016-04-05 23:58 /var
[cloudera@quickstart ~]$ █
```

View the content of /user directory

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

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user
Found 9 items
drwxr-xr-x  - cloudera  cloudera          0 2016-08-03 01:54 /user/cloudera
drwxr-xr-x  - hdfs      supergroup        0 2016-08-02 08:08 /user/hdfs
drwxr-xr-x  - mapred    hadoop           0 2016-04-05 23:56 /user/history
drwxrwxrwx  - hive      supergroup        0 2016-04-05 23:58 /user/hive
drwxrwxrwx  - hue       supergroup        0 2016-04-05 23:57 /user/hue
drwxrwxrwx  - jenkins   supergroup        0 2016-04-05 23:56 /user/jenkins
drwxrwxrwx  - oozie    supergroup        0 2016-04-05 23:57 /user/oozie
drwxrwxrwx  - root     supergroup        0 2016-04-05 23:57 /user/root
drwxr-xr-x  - hdfs      supergroup        0 2016-04-05 23:58 /user/spark
```



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 ~]$ █
```



The directory structure of Hadoop filesystem is different from local filesystem i.e. Linux filesystem. To know the difference, list the files on your local filesystem.

Create a new directory named “hadoop” in HDFS

```
hdfs dfs -mkdir hadoop
```

Create a sample.txt file on local and upload the file into newly created directory

```
hdfs dfs -put sample.txt hadoop/sample.txt
```

View the content of new file

```
hdfs dfs -cat hadoop/sample.txt
```



In HDFS, any non-absolute path is considered relative to your home directory i.e. /user/cloudera. Unlike Linux filesystem concept of “current” or “present working directory”.

Download a file from HDFS to your local filesystem, specify HDFS path and local path to achieve the same.

```
hdfs dfs -get /user/cloudera/sample.text /home/cloudera
```

Remove the directory along with its content (perform this step after completing Task 2)

```
hdfs dfs -rm -r hadoop
```

List all the shell commands supported by Hadoop

```
hdfs dfs
```

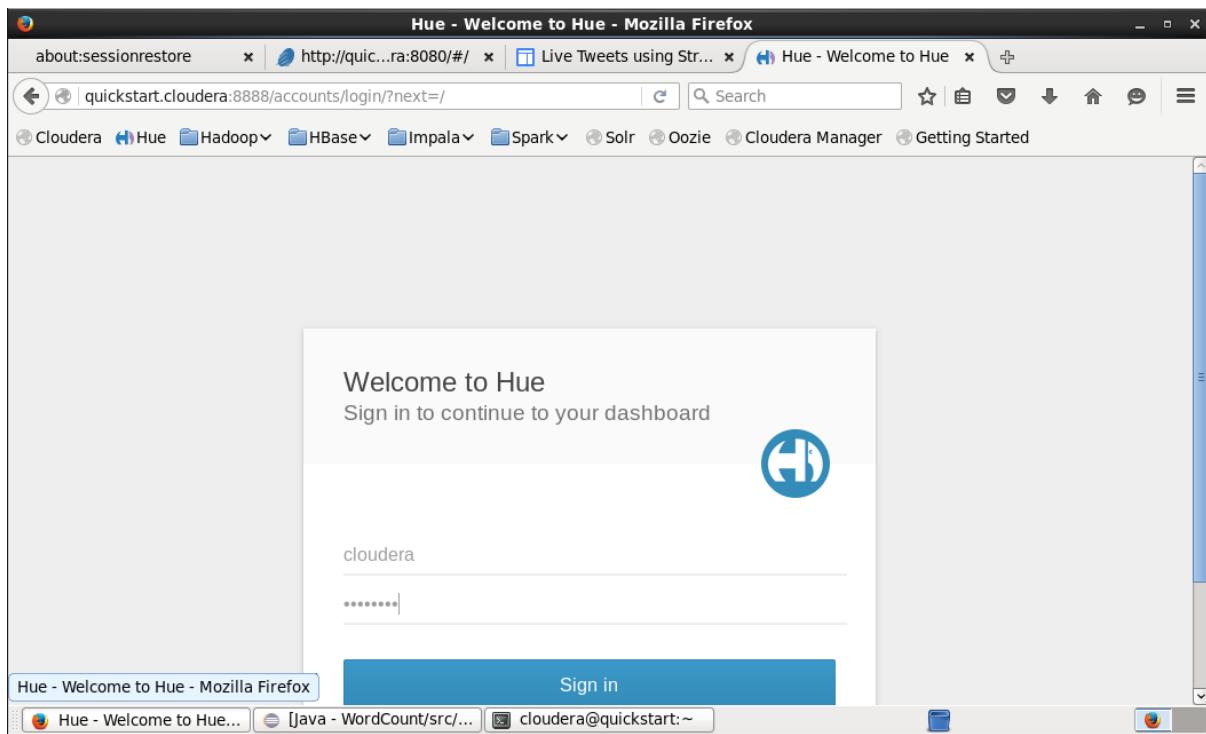
Task2: Using HUE File browser

1. Open a web browser on your VM, and navigate to bookmark tab and select **HUE**. This can be alternatively launched by specifying <http://quickstart.cloudera:8888>
2. Enter username as ‘cloudera’ and password ‘cloudera’.
3. To access HDFS, click on **Manage HDFS** in the HUE menu bar
4. By default, the content of home directory i.e. /user/cloudera will be visible.
5. Point out to the directory named “hadoop” created above and view the file ‘sample.txt’
6. Click on **Upload button** and select the appropriate format of the file to be uploaded i.e. plain file or zipped file



The zipped file will be automatically unzipped after upload.

7. Select **Files> Select Files**, and browser to location of data file on your local filesystem
8. Choose the file and click the **Open button**.
9. The selected file will be displayed in the directory /user/cloudera/
10. Click on the checkbox next to file’s icon and then tab on **Actions** button to know the possible actions that can be performed on the file.
11. Once you have practised above steps, you can remove the files by clicking on **“Move to Trash”** button.

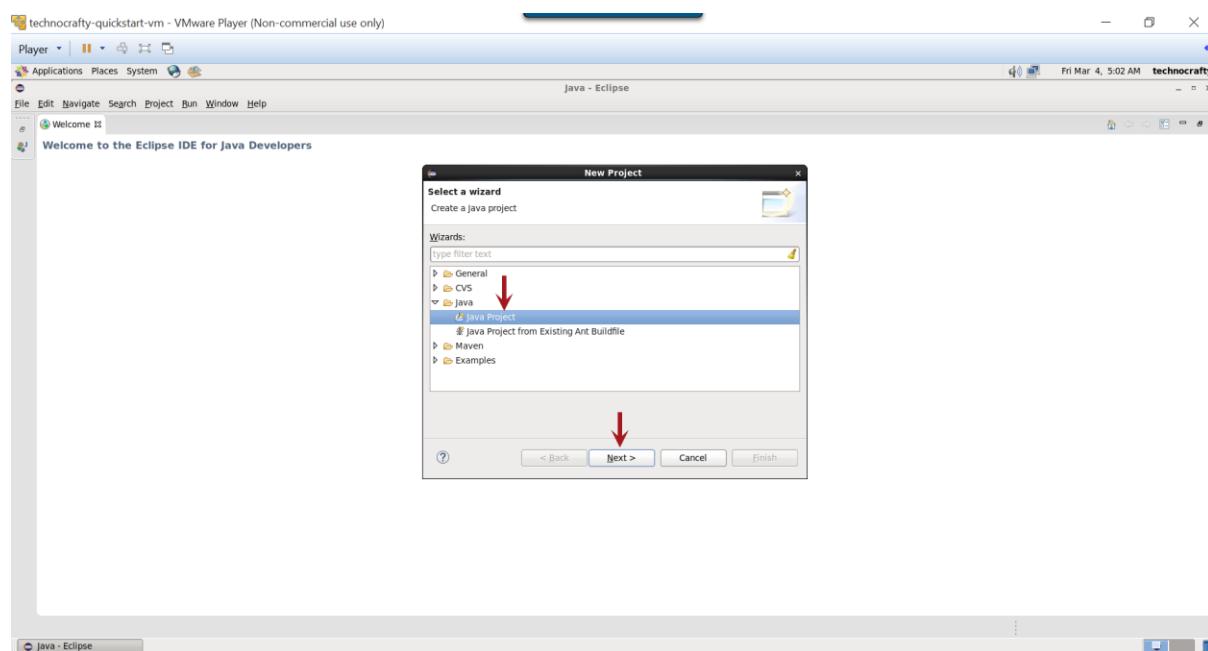
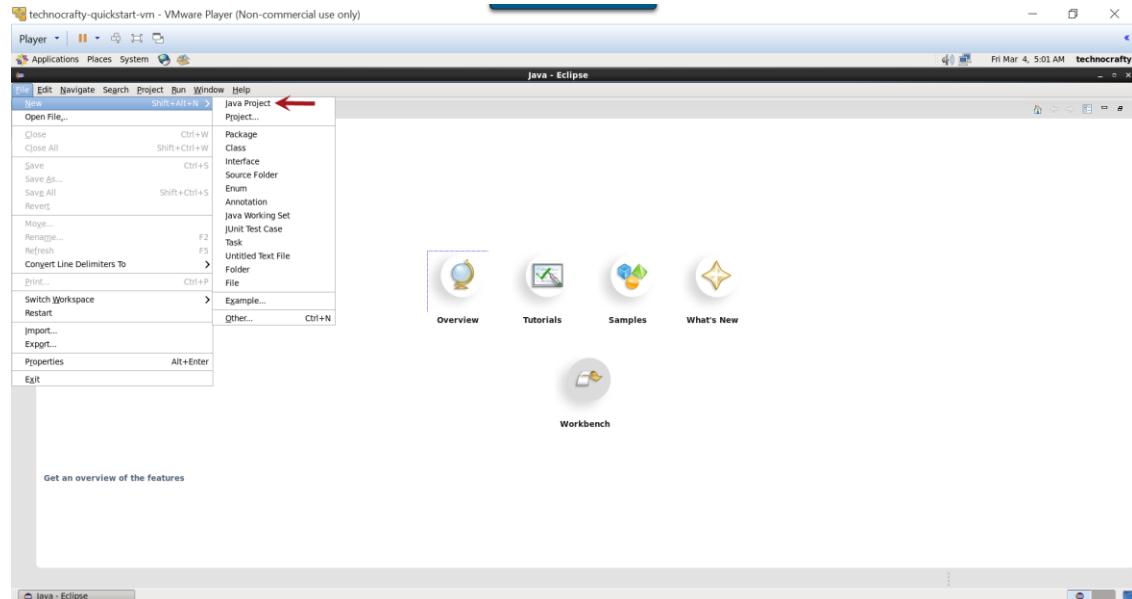


Exercise 3: Running MapReduce Job

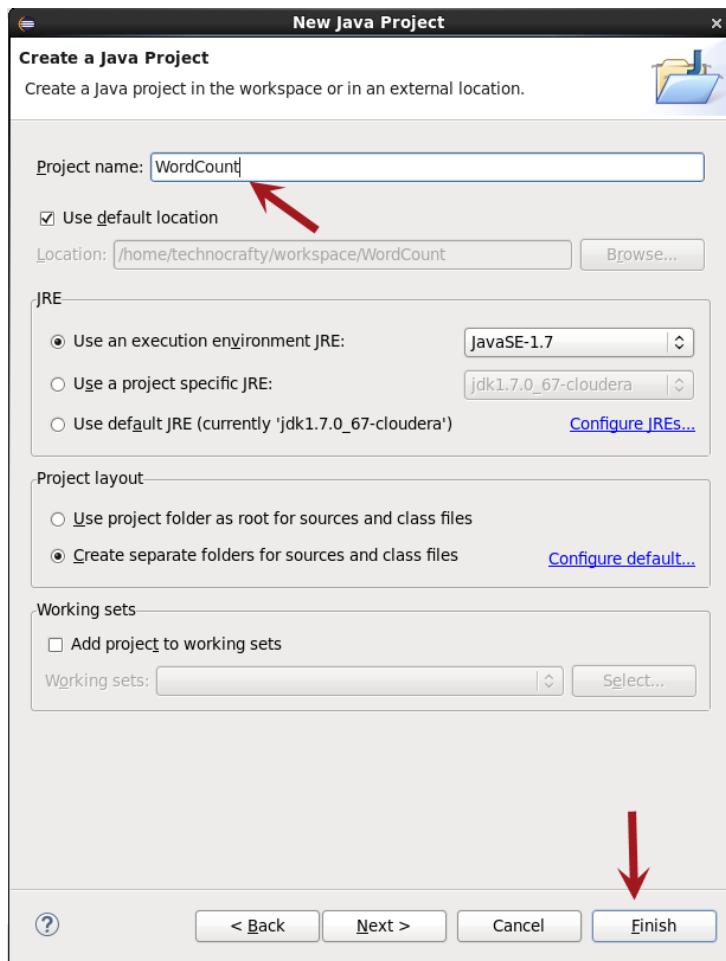
Prerequisites: Create an Eclipse WordCount Project

Step1: Launch Eclipse, Select the workspace location, and click on “OK”.

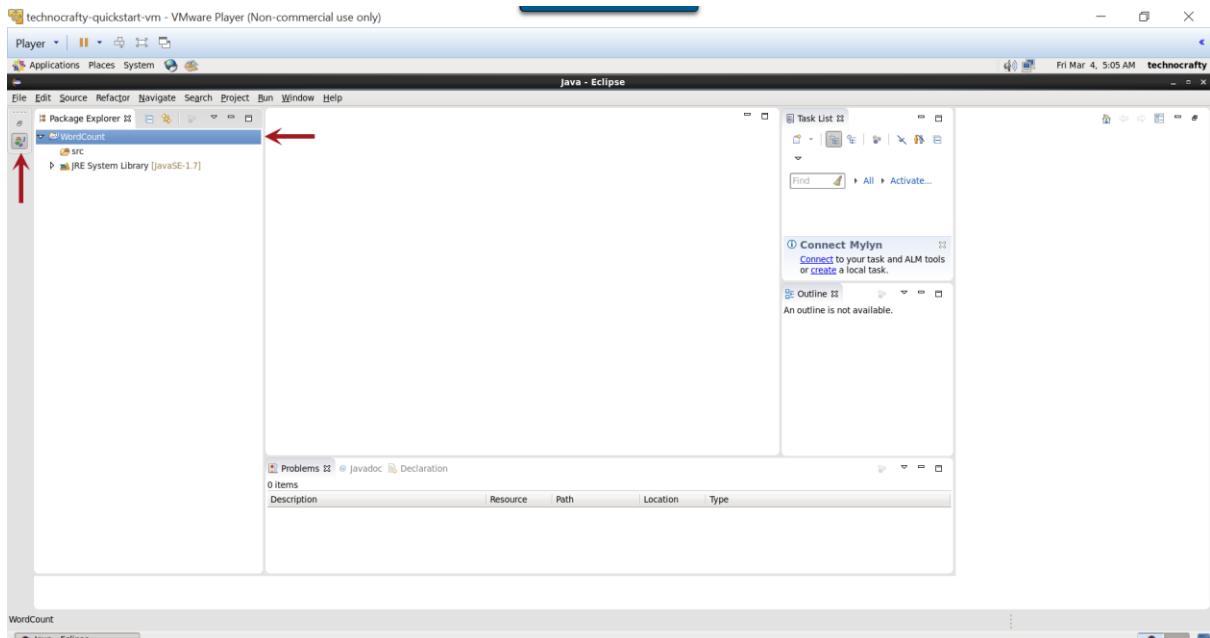
Step2: Select **File** > **New** > **Java Project** > **Next**



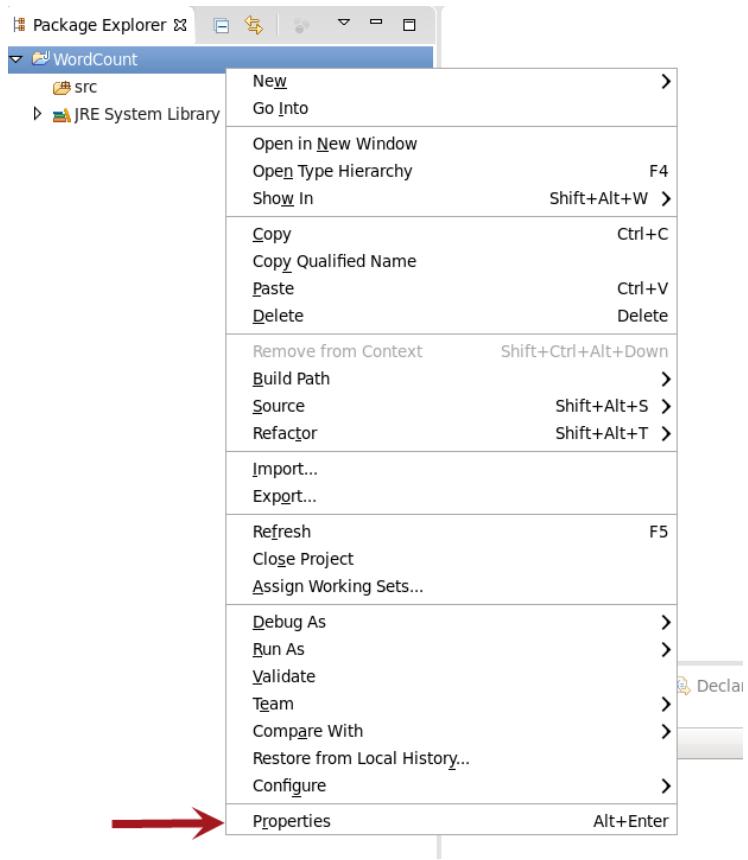
Step3: Name the project as "Wordcount" and click **"Finish"**



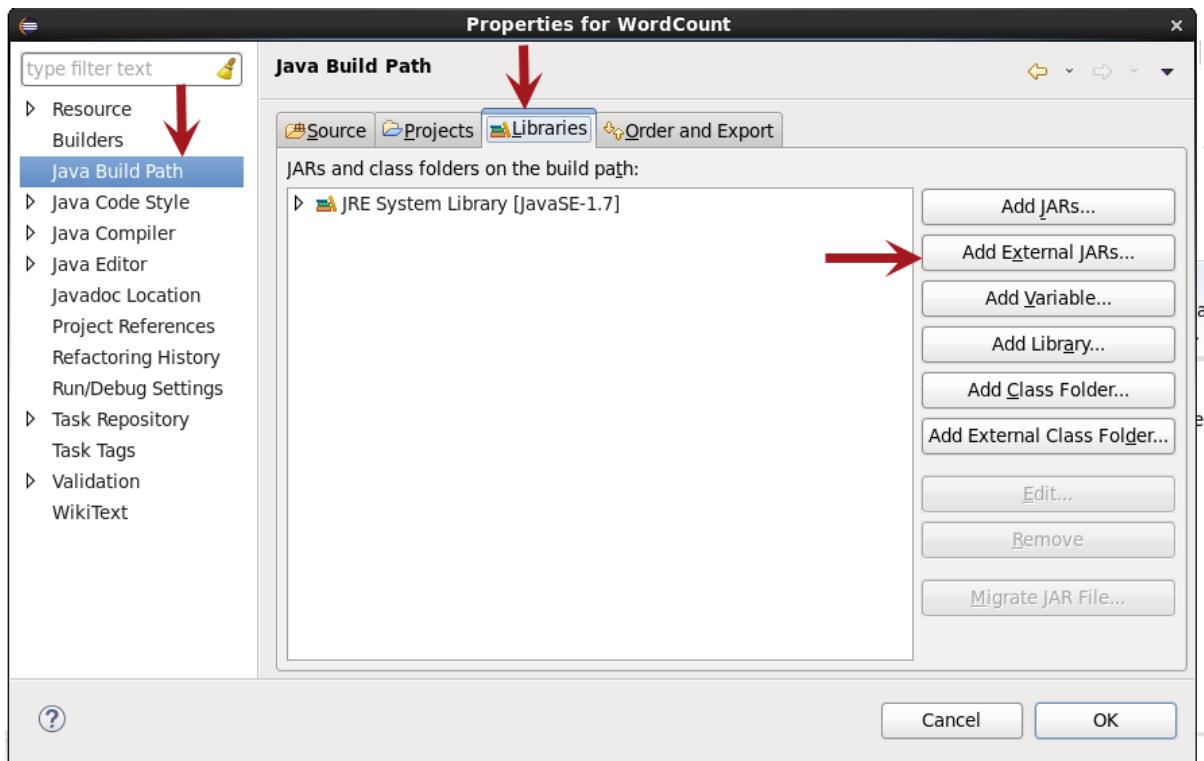
Step4: Expand the “Package Explorer” tab and locate your new project i.e. “WordCount”.



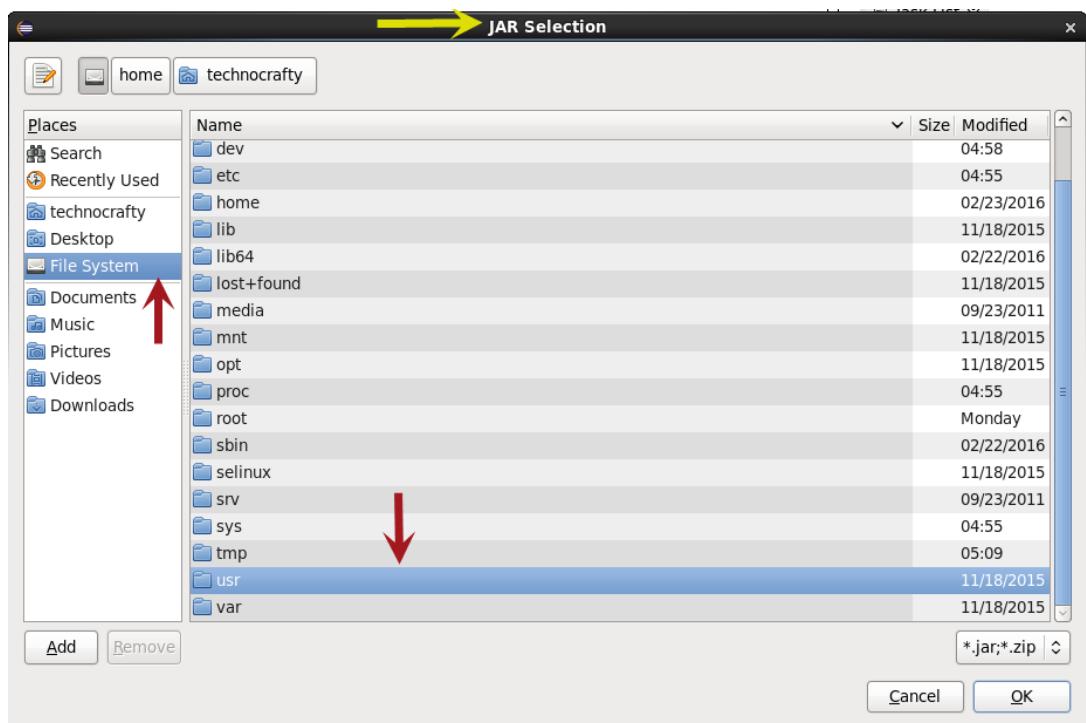
Step5: Export Hadoop Libraries by Right Click on Wordcount project and selecting Properties

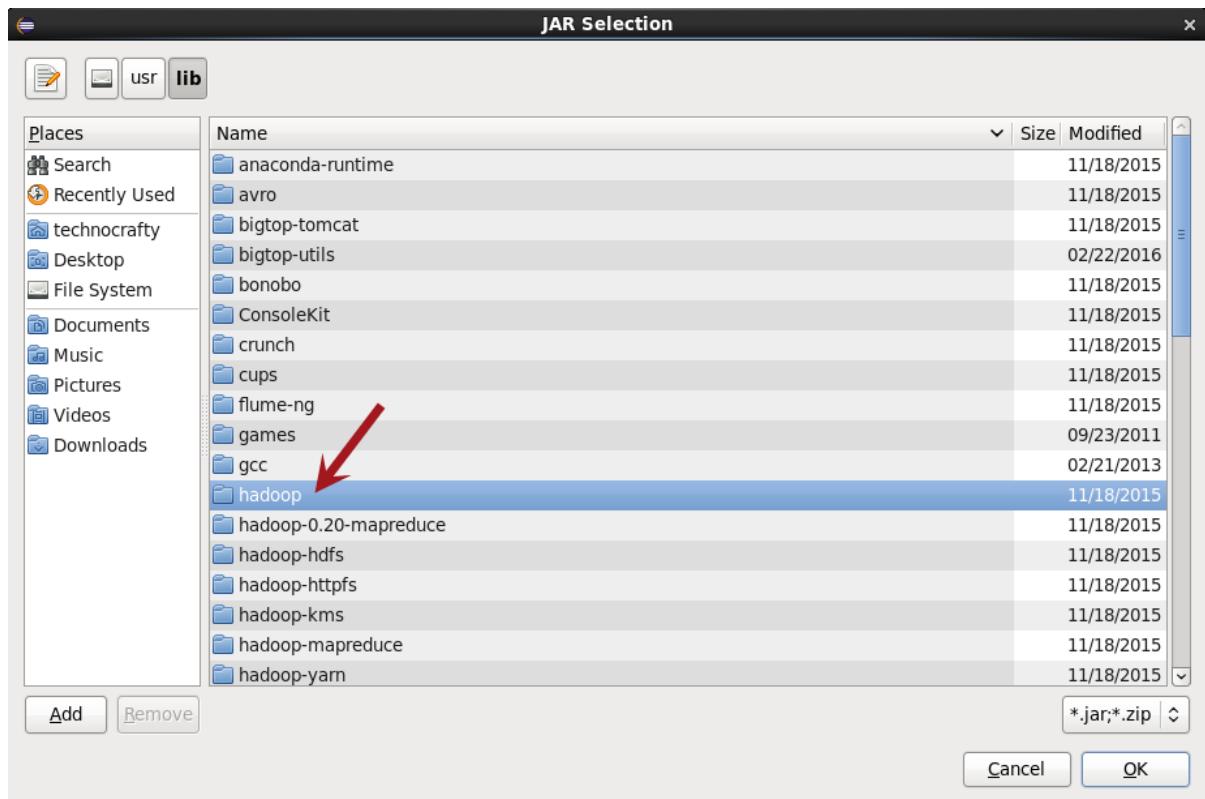
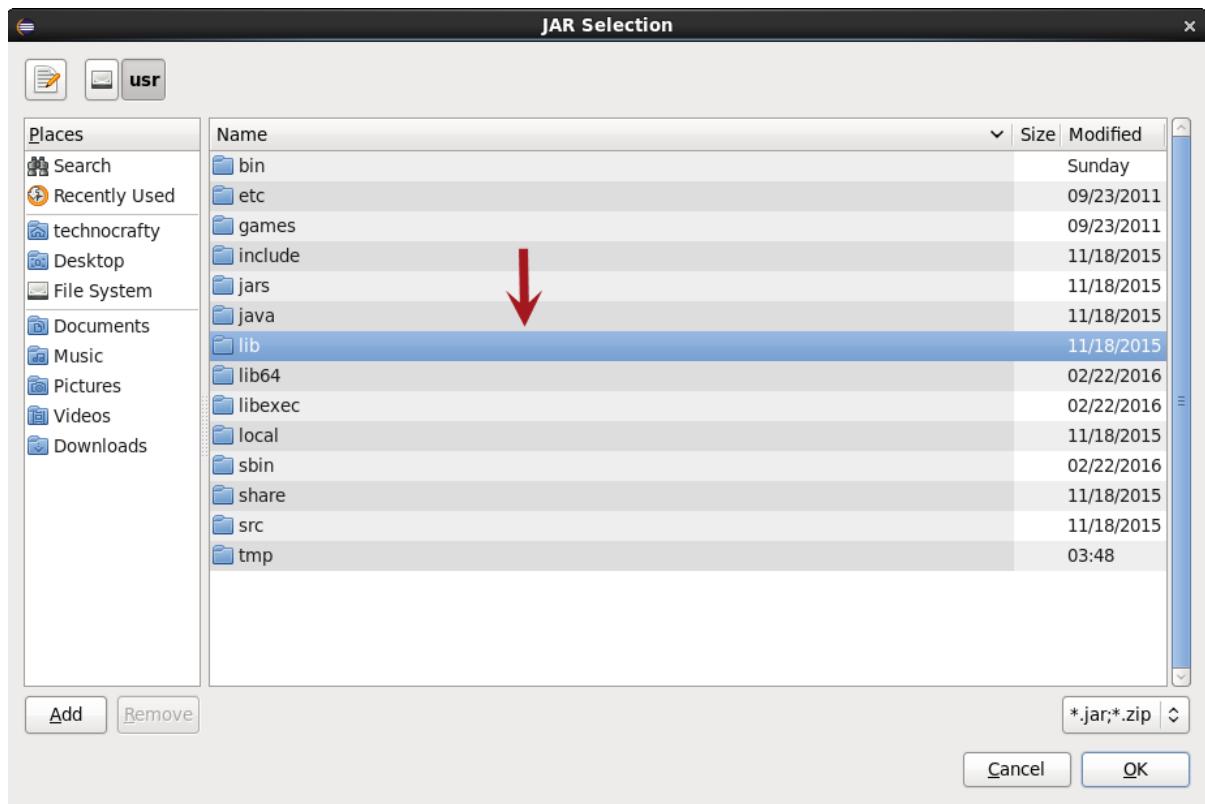


Step6: Click on Java Build Path from left panel and select “Libraries” tab from right panel view

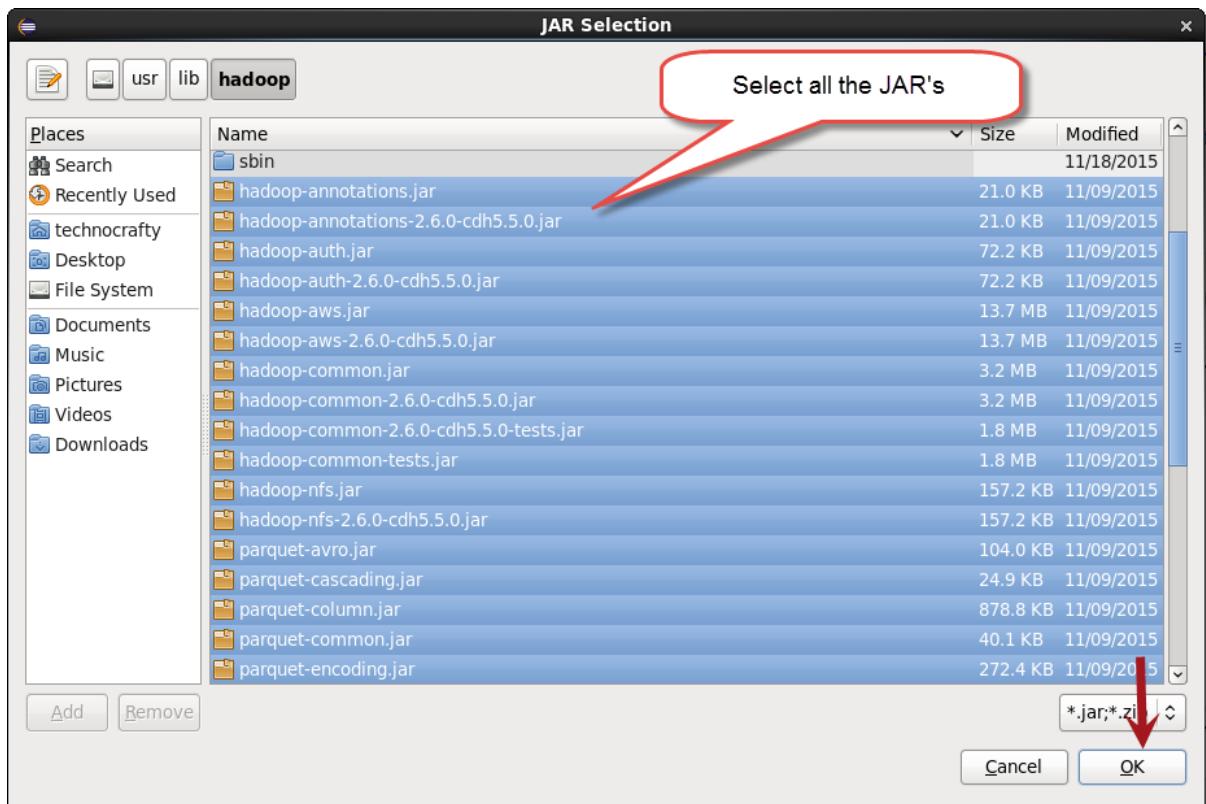


Step7: Select “Add External JAR...”, then navigate to File System > usr > lib > Hadoop

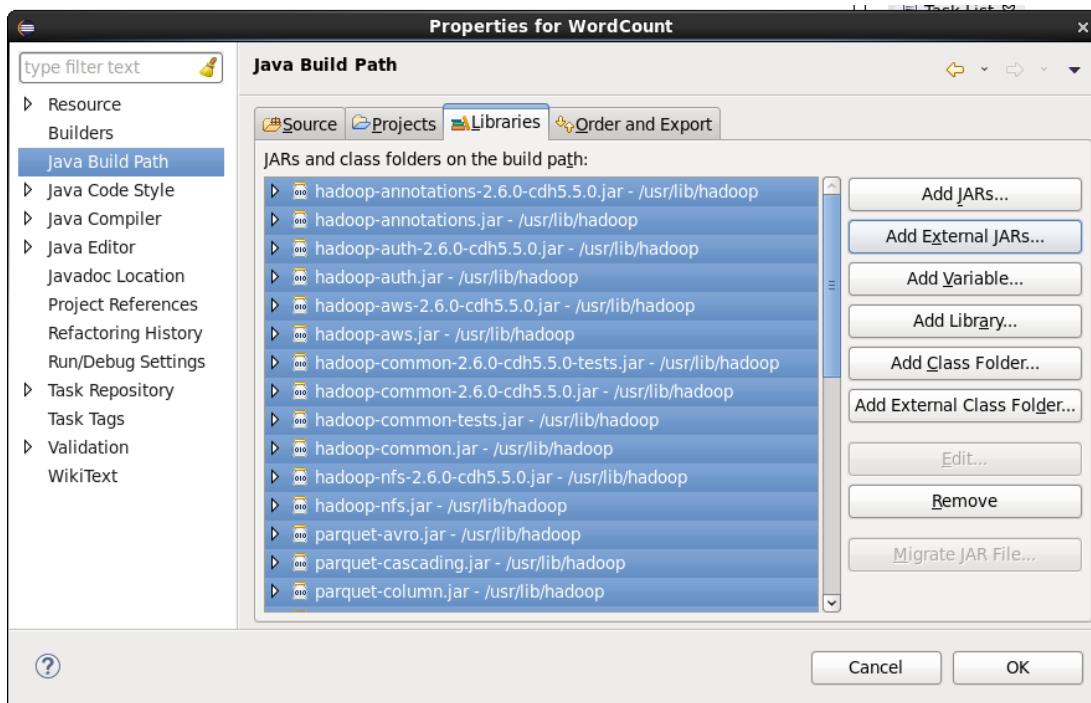




Step8: Select all the JAR available under this folder

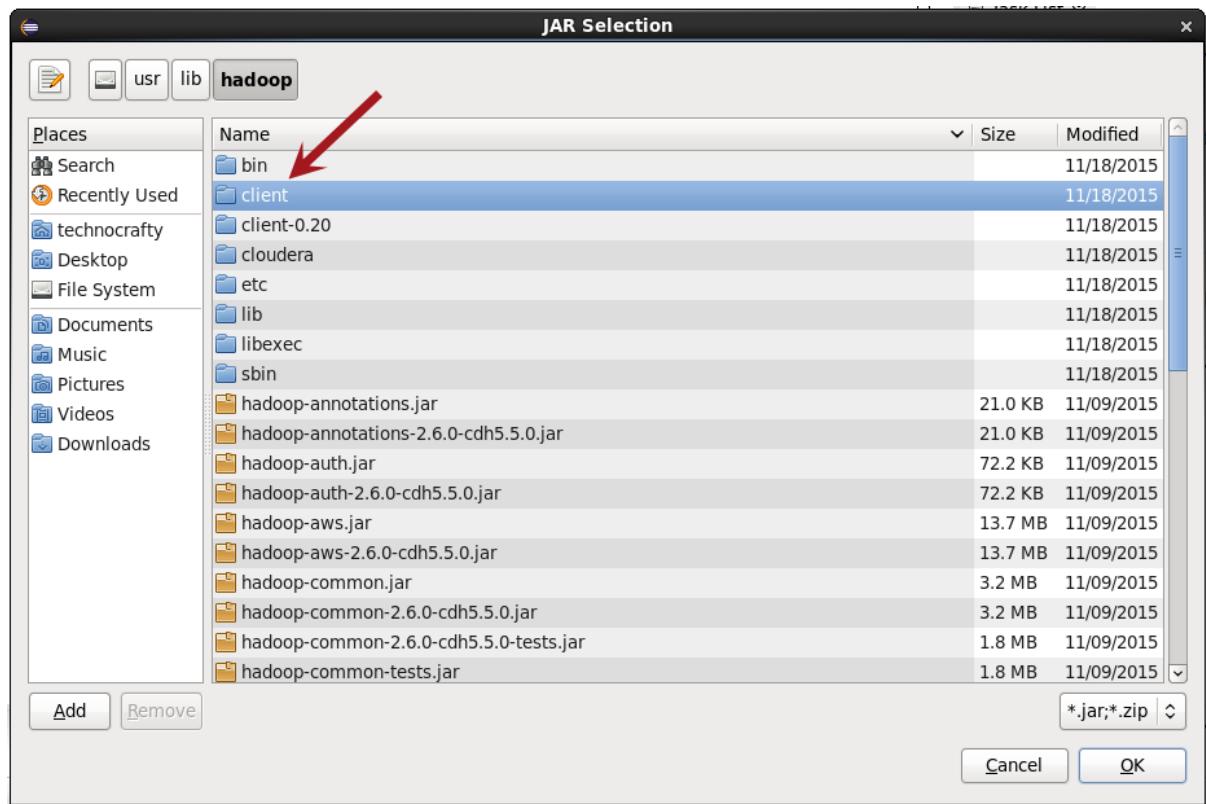


Click on "OK"



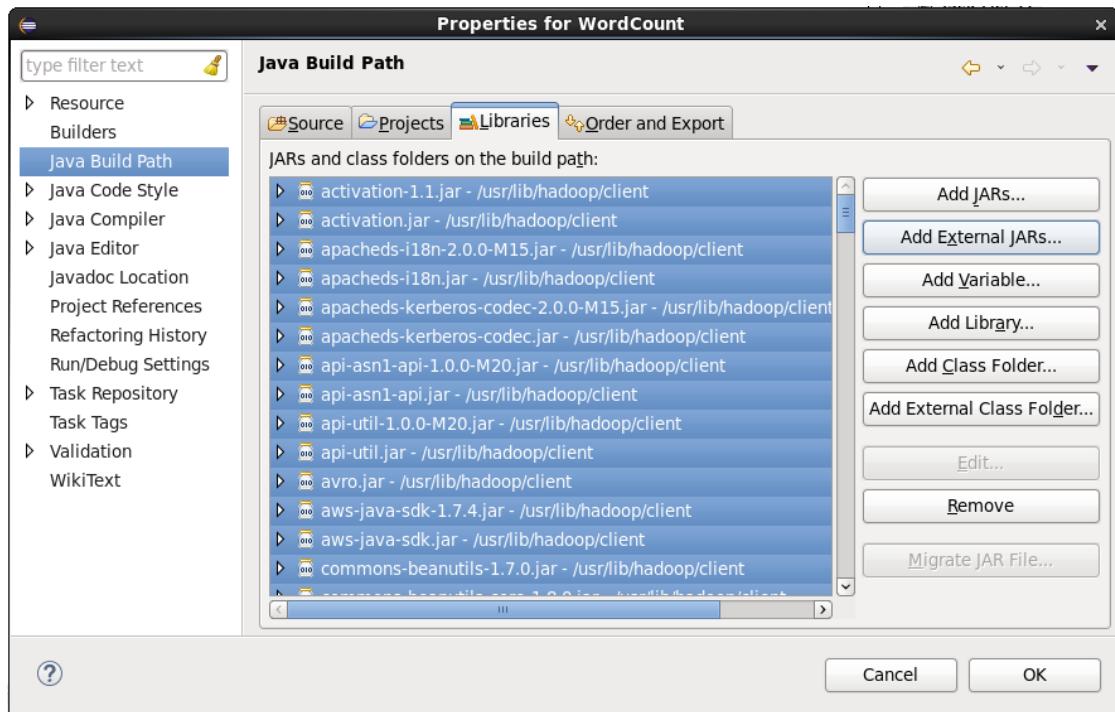
Step 9: Perform same steps for /usr/lib/hadoop-mapreduce and /usr/lib/Hadoop-yarn

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

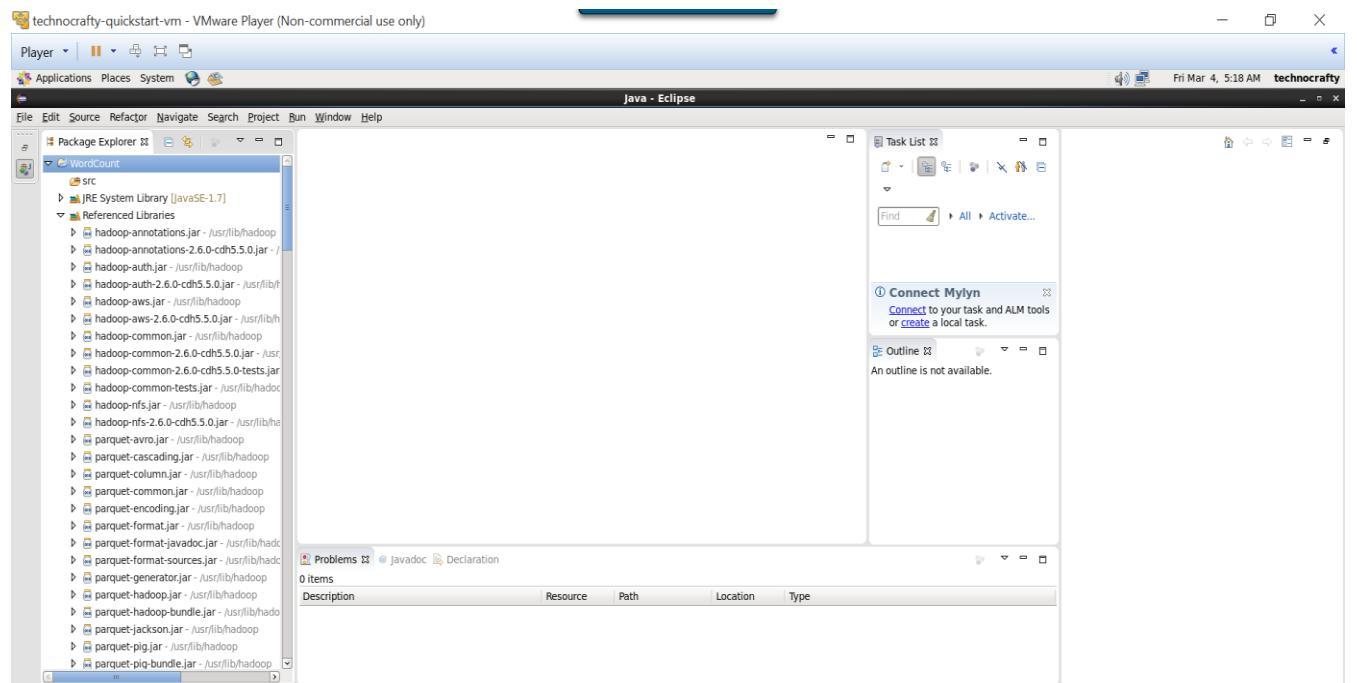


Select the files and click **OK**

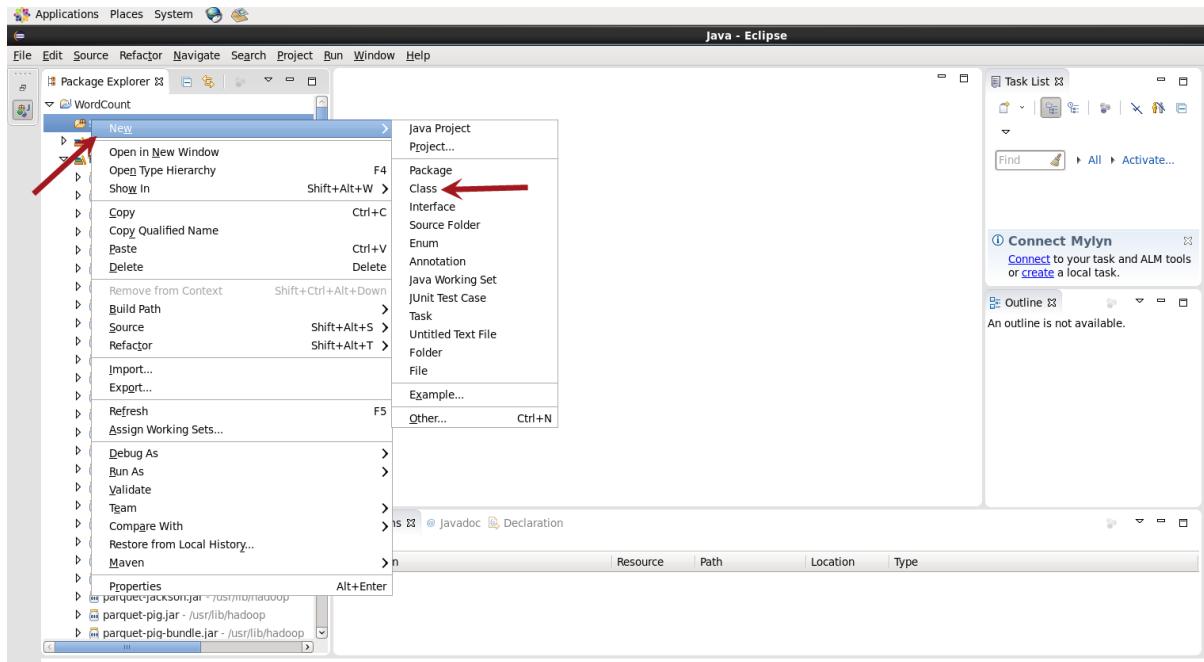




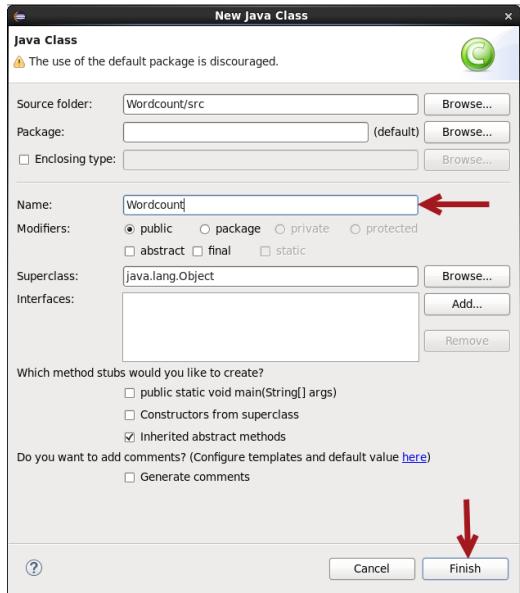
Step11: Finally, you will find all the JAR files under “Referenced Libraries”



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

```

public class WordCount {
}

```

Step15: MapReduce Wordcount Program



Step 16: Save the program

```

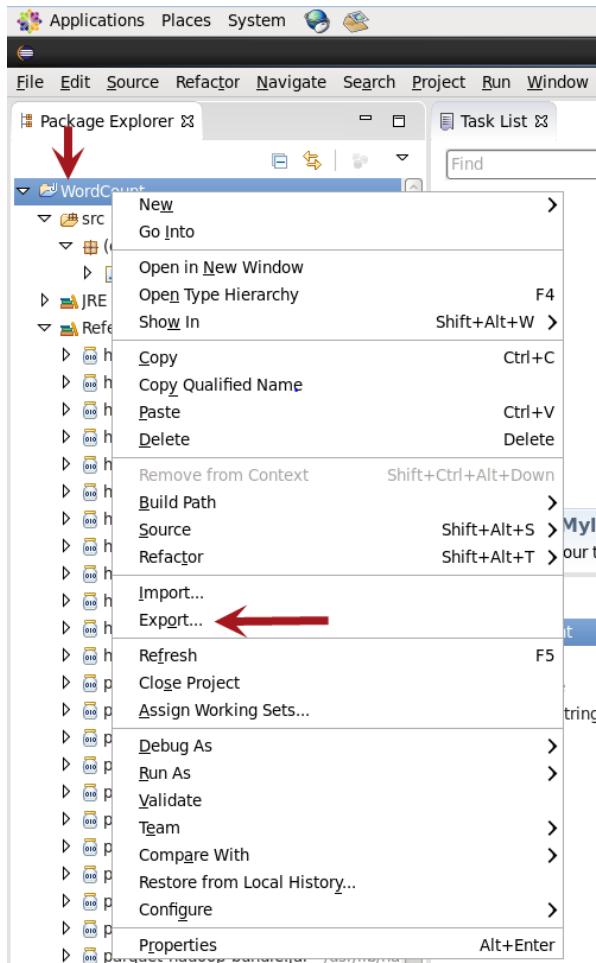
import java.io.IOException;
public class WordCount extends Configured implements Tool {
    public static void main(String args[]) throws Exception {
        int res = ToolRunner.run(new WordCount(), args);
        System.exit(res);
    }
    public int run(String[] args) throws Exception {
        Path inputPath = new Path(args[0]);
        Path outputPath = new Path(args[1]);
        Configuration conf = getConf();
        Job job = new Job(conf, this.getClass().toString());
        FileInputFormat.setInputPaths(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputPath);
        job.setJobName("WordCount");
        job.setJarByClass(WordCount.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
    }
}

```

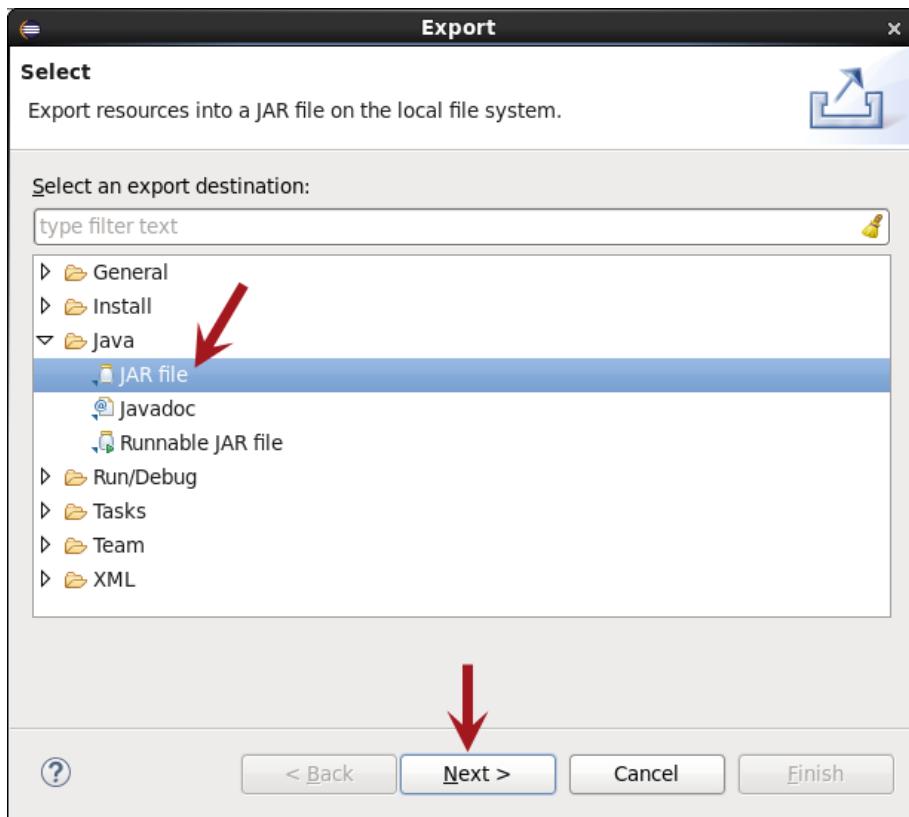


Ensure that there is no compilation error, before exporting the JAR file.

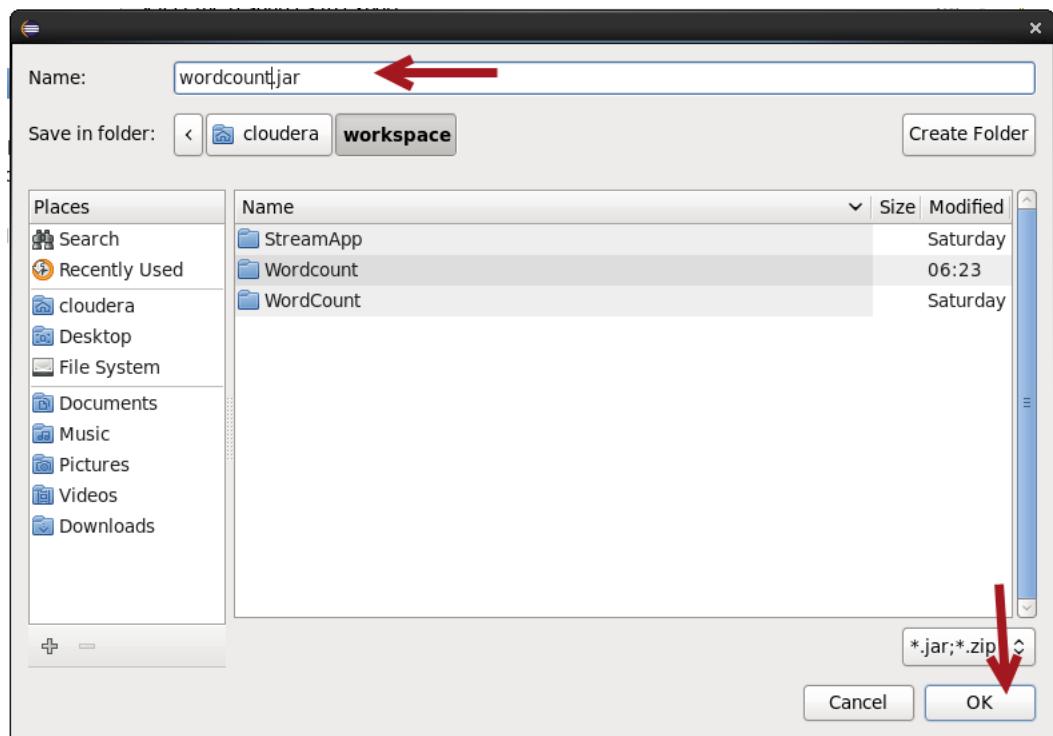
Step17: Exporting the JAR file, for that Right Click on WordCount project and select “Export”



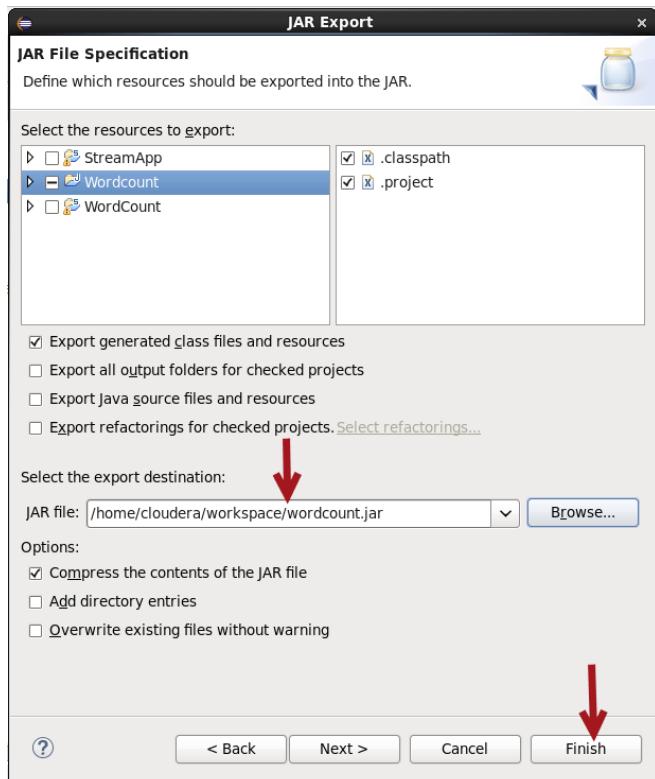
Step 18: In the next panel, expand Java and select JAR file



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 file at the target location from the terminal.

```
[cloudera@quickstart workspace]$ ls -ltr
total 20
drwxrwxr-x 7 cloudera cloudera 4096 Sep 10 01:43 StreamApp
drwxrwxr-x 8 cloudera cloudera 4096 Sep 10 01:43 WordCount
drwxrwxr-x 5 cloudera cloudera 4096 Sep 13 06:23 Wordcount
-rw-rw-r-- 1 cloudera cloudera 6055 Sep 13 06:40 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 story.txt

```
[cloudera@quickstart ~]$ cd datasets/
[cloudera@quickstart datasets]$ ls -ltr story.txt
-rw-rw-r-- 1 cloudera cloudera 2249 Sep  6 19:28 story.txt
[cloudera@quickstart datasets]$ pwd
/home/cloudera/datasets
[cloudera@quickstart datasets]$
```

Step2: Create a directory named “data” in the HDFS and import “story.txt” file.

```
hdfs dfs -mkdir data  
hdfs dfs -put story.txt data
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls data/story.txt  
-rw-r--r-- 1 cloudera cloudera 2249 2016-09-06 19:29 data/story.txt  
[cloudera@quickstart ~]$ █
```

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

```
hadoop jar wordcount.jar Wordcount data/story.txt output_map
```

```
[cloudera@quickstart workspace]$ pwd  
/home/cloudera/workspace  
[cloudera@quickstart workspace]$ jar -tf wordcount.jar  
META-INF/MANIFEST.MF  
.project  
Wordcount$Map.class  
Wordcount$Reduce.class  
Wordcount.class  
.classpath  
[cloudera@quickstart workspace]$ hadoop jar wordcount.jar Wordcount data/story.t  
xt output_map
```

To check the
classes

```
16/09/13 06:51:07 INFO mapreduce.Job: map 100% reduce 100%  
16/09/13 06:51:08 INFO mapreduce.Job: Job job_1473496582710_0001 completed  
successfully  
16/09/13 06:51:08 INFO mapreduce.Job: Counters: 49  
    File System Counters  
        FILE: Number of bytes read=2906  
        FILE: Number of bytes written=234627  
        FILE: Number of read operations=0  
        FILE: Number of large read operations=0  
        FILE: Number of write operations=0  
        HDFS: Number of bytes read=2374  
        HDFS: Number of bytes written=2035  
        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)=10457  
        Total time spent by all reduces in occupied slots (ms)=11396  
        Total time spent by all map tasks (ms)=10457  
        Total time spent by all reduce tasks (ms)=11396  
        Total vcore-seconds taken by all map tasks=10457  
        Total vcore-seconds taken by all reduce tasks=11396
```

```

Total megabyte-seconds taken by all map tasks=10707968
Total megabyte-seconds taken by all reduce tasks=11669504
Map-Reduce Framework
  Map input records=17
  Map output records=354
  Map output bytes=3656
  Map output materialized bytes=2906
  Input split bytes=125
  Combine input records=354
  Combine output records=217
  Reduce input groups=217
  Reduce shuffle bytes=2906
  Reduce input records=217
  Reduce output records=217
  Spilled Records=434
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=246
  CPU time spent (ms)=1530
  Physical memory (bytes) snapshot=385163264
  Virtual memory (bytes) snapshot=3007664128
  Total committed heap usage (bytes)=354357248
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=2249
File Output Format Counters
  Bytes Written=2035
[clooudera@quickstart workspace]$

```

Check the output file:

```

[clooudera@quickstart workspace]$ hdfs dfs -ls output_map
Found 2 items
-rw-r--r--  1 cloudera cloudera      0 2016-09-13 06:51 output_map/_SUCCESS
-rw-r--r--  1 cloudera cloudera 2035 2016-09-13 06:51 output_map/part-r-0
0000

```

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

The screenshot shows the Cloudera Manager interface for the 'All Applications' section. The URL in the browser is 'quickstart.cloudera:8088/cluster'. The main content area is titled 'All Applications' and features the Hadoop logo. On the left, there's a sidebar with 'Cluster Metrics' and a table showing cluster status (e.g., NEW, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED). Below that is a 'Scheduler' section and a 'Tools' section. A red arrow points from the URL bar to the table below. Another red arrow points to the specific application entry in the table.

ID	User	Name	Application Type	Queue	StartTime	FinishTime
application_1473496582710_0001	cloudera	Wordcount	MAPREDUCE	root.cloudera	Tue Sep 13 06:50:20 -0700 2016	Tue Sep 13 06:51:06 -0700 2016

Exercise 4: 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

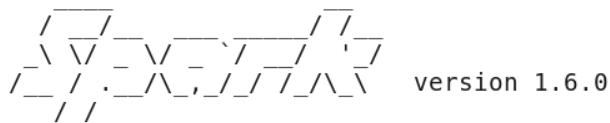
```
scala> sc.
accumable          accumableCollection
accumulator        addFile
addJar            addSparkListener
appName           applicationAttemptId
applicationId    asInstanceOf
binaryFiles       binaryRecords
broadcast         cancelAllJobs
cancelJobGroup   clearCallSite
clearFiles        clearJars
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.

- Invoke python spark shell by using ‘pyspark’

```
[cloudera@quickstart ~]$ pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
Welcome to
```



```
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> █
```

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 story.txt using **sc.textFile** method.

```
val textFile = sc.textFile("file:/home/cloudera/datasets/story.txt")
```

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)] = ZippedPartitionsRDD[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))

```

Using Python Shell:

```

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
>>> █

```

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)
```

```
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
```

Using python shell:

```
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'
>>> 
```

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 localhost (3/3)
16/08/30 07:30:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 15.0, whose tasks have all completed, from 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), (20,120), (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), (52,152), (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), (74,174), (75,175), (76,176), (77,177), (78...)
```

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 localhost (2/2)
16/08/30 07:31:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, from 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, from 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 local host (2/2)
16/08/30 07:34:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 18.0, whose tasks have all completed, from 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, from 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, from 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, from 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, from 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, from 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, from 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, from 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 to 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, from 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, from 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, from 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 = 10.60533333333332
```

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, from 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 to 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,CompactBuffer(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 localhost (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))

```

fold

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 employeeData = List(("Jack",1000.0),("Bob",2000.0),("Carl",7000.0))

val employeeRDD = sc.makeRDD(employeeData)

val dummyEmployee = ("dummy",0.0)

val maxSalaryEmployee = employeeRDD.fold(dummyEmployee)((acc,employee) => {
  if(acc._2 < employee._2) employee else acc})

println("employee with maximum salary is"+maxSalaryEmployee)

scala> println("employee with maximum salary is"+maxSalaryEmployee)
employee with maximum salary is(Carl,7000.0)
```

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(
  ("cs", ("jack",1000.0)),
  ("cs", ("bron",1200.0)),
  ("phy", ("sam",2200.0)),
  ("phy", ("ronaldo",500.0))

)

val employeeRDD = sc.makeRDD(deptEmployees)

val maxByDept = employeeRDD.foldByKey(("dummy",0.0))((acc,element)=>
if(acc._2 > element._2) acc else element)

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

scala> println("maximum salaries in each dept" + maxByDept.collect().toList)
maximum salaries in each deptList((phy,(sam,2200.0)), (cs,(bron,1200.0)))
```

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 to 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 to 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", "turkey", "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, from 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 an 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","turkey","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, from 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 an 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","turkey","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, from 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 localhost (3/3)
16/08/30 09:14:29 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 56.0, whose tasks have all completed, from 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 particular word in it

```
val textFile = sc.textFile("file:/home/cloudera/datasets/Joyce.txt")  
  
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
```

Additional Exercise

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 are 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

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

```
val movies = sc.textFile("file:/home/cloudera/datasets/movies.dat")
val users = sc.textFile("file:/home/cloudera/datasets/users.dat")
val ratings =
sc.textFile("file:/home/cloudera/datasets/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 tasks 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, 974709030)), (1485,Array(1721, 1485, 5, 974706178)), (3247,Array(1721, 3247, 5, 974706026)))
```

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 tasks 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(_.value._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:

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

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?

Exercise 6: Dataframe and Spark SQL

Two ways to create Spark Dataframes:

1. Create from an existing RDD
2. Create from other data sources

Creating DataFrames from Existing RDD:

1. Infer schema by Reflection
 - a. Convert RDD containing case classes.
 - b. Use when schema is known.
2. Construct schema programmatically
 - a. Schema is dynamic
 - b. Number of fields in case class is more than 22 fields.

Infer schema by Reflection (case class)

Steps:

1. Import necessary classes
2. Create RDD
3. Define case class
4. Convert RDD into RDD of case objects.
5. Implicitly convert resulting RDD of case objects into DataFrames.
6. Register DataFrame as table.

Exercise: Analyze SFPD Crime Dataset using Spark Dataframes.

1. Upload the file ‘crime_incidents.csv’ from datasets directory on local filesystem to HDFS.

2. Create a base RDD from the file ‘crime_incidents’.

```
val rdd =  
sc.textFile("file:/home/cloudera/datasets/crime_incidents.csv")  
rdd.take(5).foreach(println)
```

```
IncidntNum,Category,Descript,DayOfWeek,Date,Time,PdDistrict,Resolution,Address,X  
,Y,Location  
050436712,ASSAULT,BATTERY,Wednesday,04/20/2005 12:00:00 AM,04:00,MISSION,NONE,18  
TH ST / CASTRO ST,-122.435002864271,37.7608878061245,"(37.7608878061245, -122.43  
5002864271)"  
080049078,LARCENY/THEFT,GRAND THEFT FROM A BUILDING,Sunday,01/13/2008 12:00:00 A  
M,18:00,PARK,NONE,1100 Block of CLAYTON ST,-122.446837820235,37.7622550270122,"(  
37.7622550270122, -122.446837820235)"  
130366639,ASSAULT,AGGRAVATED ASSAULT WITH A KNIFE,Sunday,05/05/2013 12:00:00 AM,  
04:10,INGLESIDE,"ARREST, BOOKED",0 Block of SGTJOHNVYOUNG LN,-122.444707063455,3  
7.7249307267936,"(37.7249307267936, -122.444707063455)"  
030810835,DRIVING UNDER THE INFLUENCE,DRIVING WHILE UNDER THE INFLUENCE OF ALCOH  
OL,Tuesday,07/08/2003 12:00:00 AM,01:00,SOUTHERN,"ARREST, BOOKED",MASON ST / TUR  
K ST,-122.408953598286,37.7832878735491,"(37.7832878735491, -122.408953598286)"
```

3. Cache the RDD since we are going to process it several times

```
rdd.cache  
rdd.count
```

```
16/09/01 02:32:12 INFO scheduler.DAGScheduler: Job 1 finished: count at <console  
>:24, took 13.574773 s  
res2: Long = 1888568
```

4. Create a filtered RDD by removing the headers

```
val data = rdd.filter(line => !line.startsWith("IncidntNum"))  
data.take(5).foreach(println)
```

```

16/09/01 02:33:04 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
050436712,ASSAULT,BATTERY,Wednesday,04/20/2005 12:00:00 AM,04:00,MISSION,NONE,18TH ST / CASTRO ST,-122.435002864271,37.7608878061245,"(37.7608878061245, -122.435002864271)"
080049078,LARCENY/THEFT,GRAND THEFT FROM A BUILDING,Sunday,01/13/2008 12:00:00 AM,18:00,PARK,NONE,1100 Block of CLAYTON ST,-122.446837820235,37.7622550270122,"(37.7622550270122, -122.446837820235)"
130366639,ASSAULT,AGGRAVATED ASSAULT WITH A KNIFE,Sunday,05/05/2013 12:00:00 AM,04:10,INGLESIDE,"ARREST, BOOKED",0 Block of SGTJOHNVYOUNG LN,-122.444707063455,37.7249307267936,"(37.7249307267936, -122.444707063455)"
030810835,DRIVING UNDER THE INFLUENCE,DRIVING WHILE UNDER THE INFLUENCE OF ALCOHOL,Tuesday,07/08/2003 12:00:00 AM,01:00,SOUTHERN,"ARREST, BOOKED",MASON ST / TURK ST,-122.408953598286,37.7832878735491,"(37.7832878735491, -122.408953598286)"
130839567,OTHER OFFENSES,TRAFFIC VIOLATION ARREST,Friday,10/04/2013 12:00:00 AM,20:53,TENDERLOIN,"ARREST, BOOKED",TURK ST / LEAVENWORTH ST,-122.414056291891,37.7827931071006,"(37.7827931071006, -122.414056291891)"

```

- Parse the data using regex. Regex allows to handle fields that contain the field separators comma

```

val pattern = """^([^\"]*)\"|(?=<=,|^)([^,]*)(?=>,|$)""".r
val dataParsed = data.map(pattern.findAllIn(_).toArray)
dataParsed.take(5).foreach(rec => println(rec.mkString(", ")))

```

```

16/09/01 02:34:42 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
050436712,ASSAULT,BATTERY,Wednesday,04/20/2005 12:00:00 AM,04:00,MISSION,NONE,18TH ST / CASTRO ST,-122.435002864271,37.7608878061245,"(37.7608878061245, -122.435002864271)"
080049078,LARCENY/THEFT,GRAND THEFT FROM A BUILDING,Sunday,01/13/2008 12:00:00 AM,18:00,PARK,NONE,1100 Block of CLAYTON ST,-122.446837820235,37.7622550270122,"(37.7622550270122, -122.446837820235)"
130366639,ASSAULT,AGGRAVATED ASSAULT WITH A KNIFE,Sunday,05/05/2013 12:00:00 AM,04:10,INGLESIDE,"ARREST, BOOKED",0 Block of SGTJOHNVYOUNG LN,-122.444707063455,37.7249307267936,"(37.7249307267936, -122.444707063455)"
030810835,DRIVING UNDER THE INFLUENCE,DRIVING WHILE UNDER THE INFLUENCE OF ALCOHOL,Tuesday,07/08/2003 12:00:00 AM,01:00,SOUTHERN,"ARREST, BOOKED",MASON ST / TURK ST,-122.408953598286,37.7832878735491,"(37.7832878735491, -122.408953598286)"
130839567,OTHER OFFENSES,TRAFFIC VIOLATION ARREST,Friday,10/04/2013 12:00:00 AM,20:53,TENDERLOIN,"ARREST, BOOKED",TURK ST / LEAVENWORTH ST,-122.414056291891,37.7827931071006,"(37.7827931071006, -122.414056291891)"

```

Create Dataframe using case class objects

- Create a case class that we will encapsulate the incident records into

```

case class Incident(
    IncidntNum:String,
    Category:String,

```

```

    Description:String,
    DayOfWeek:String,
    DateOfIncident:String,
    TimeOfIncident:String,
    PdDistrict:String,
    Resolution:String,
    Address:String,
    X:String,
    Y:String,
    Location:String)

```

7. Create a new rdd from dataParsed by converting the records into case class objects

```

val incidentCaseObjects = dataParsed.map(rec => Incident(rec(0),
rec(1), rec(2), rec(3), rec(4), rec(5), rec(6), rec(7), rec(8),
rec(9), rec(10), rec(11)))

incidentCaseObjects.take(5).foreach(println)

```

8. Convert the rdd of the case class objects into a dataframe

```

val incidentsDF = incidentCaseObjects.toDF

incidentsDF.show

```

IncidntNum	Category	Description[DayofWeek]	DateOfIncident[TimeOfIncident]	PdDistrict	Resolution	Address	X	Y	Location
059436712 ASSAULT BATTERY Wednesday 04/28/2005 12:00:... 04:00 MISSION NONE 18TH ST / CASTRO ST -122.435902864271 37.7668878061245 (*37.7668878061245)									
080049078 LARCENY/THEFT GRAND THEFT FROM ... Sunday 01/13/2008 12:00:... 18:00 PARK NONE 1100 Block of CLA... -122.446837820235 37.7622550270122 (*37.7622550270122)									
130366639 ASSAULT AGGRAVATED ASSAUL... Sunday 05/05/2013 12:00:... 04:10 INGLESIDE "ARREST, BOOKED" 0 Block of SGTJON... -122.444707063455 37.7249307267936 (*37.7249307267936)									
030818035 DRIVING UNDER THE... DRIVING WHILE UND... Tuesday 07/08/2003 12:00:... 01:00 SOUTHERN "ARREST, BOOKED" MASON ST / TURK ST -122.408953598206 37.7633878735491 (*37.7633878735491)									
130839567 OTHER OFFENSES TRAFFIC VIOLATION... Friday 10/04/2013 12:00:... 20:53 TENDERLOIN "ARREST, BOOKED" TURK ST / LEAVENW... -122.414056291891 37.7627931871061 (*37.7627931871061)									
078833898 BURGLARY BURGLARY OF APAR... Tuesday 08/14/2007 12:00:... 07:00 NORTHERN NONE 3100 Block of FRA... -122.426739542420 37.8034674969672 (*37.8034674969672)									
080233102 DRUG/NARCOTIC POSSESSION OF MAR... Tuesday 03/04/2008 12:00:... 14:23 INGLESIDE "ARREST, CITED" MISSION ST / PERS... -122.43597721703 37.7231288306727 (*37.7231288306727)									
060711095 OTHER OFFENSES DRIVERS LICENSE,... Wednesday 07/05/2006 12:00:... 15:50 INGLESIDE "ARREST, CITED" 2300 Block of SAN... -122.447241159611 37.7205577971255 (*37.7205577971255)									
040062593 LARCENY/THEFT GRAND THEFT FROM ... Wednesday 12/10/2003 12:00:... 09:30 INGLESIDE NONE 0 Block of MOFFET... -122.432787775641 37.737566745272 (*37.737566745272)									
110651822 NON-CRIMINAL STAY AWAY OR COU... Monday 01/17/2011 12:00:... 15:35 INGLESIDE NONE 600 Block of CAMP... -122.408761072232 37.7159008951041 (*37.7159008951041)									
060027838 LARCENY/THEFT GRAND THEFT FROM ... Saturday 01/07/2006 12:00:... 22:00 NORTHERN NONE 100 Block of HML... -122.428815262697 37.7872368476925 (*37.7872368476925)									
110929398 LARCENY/THEFT PETTY THEFT BICYCLE Sunday 11/13/2011 12:00:... 18:00 MISSION NONE 3000 Block of 22N... -122.416014728293 37.755546425920...									

9. Use Dataframe API to run analytics on dataframe

```
incidentsDF.groupBy($"Category").count().sort($"count".desc).show
```

Category	count
LARCENY/THEFT	385774
OTHER OFFENSES	269250
NON-CRIMINAL	200942
ASSAULT	165324
VEHICLE THEFT	114258
DRUG/NARCOTIC	111436
VANDALISM	96350
WARRANTS	89782
BURGLARY	78968
SUSPICIOUS OCC	67788
MISSING PERSON	55584
ROBBERY	48713
FRAUD	36004
FORGERY/COUNTERFE...	21804

10. Cache the dataframe and compare the size of the dataframe in memory with that of the rdd cached earlier

```
incidentsDF.cache  
incidentsDF.count
```

```
res91: incidentsDF.type = [IncidentNum: string, Category: string, Description: string, DayOfWeek: string, DateOfIncident:
```

11. Register the data frame as temporary table in Spark SQL

```
incidentsDF.registerTempTable("incidents")
```

12. Run SQL query on top of newly created table

```
sqlContext.sql("select * from incidents").show
```

[IncidentNum]	Category	Description	DayOfWeek	DateOfIncident	TimeOfIncident	PdDistrict	Resolution	Address	X	Y	Location
050436712	ASSAULT	BATTERY	Wednesday	04/29/2005	12:00:...	04:00	MISSION	NONE	18TH ST / CASTRO ST	-122.435902864271	(37.7608878961245
080049078	LARCENY/THEFT	GRAND THEFT FROM ...	Sunday	01/13/2008	12:00:...	18:00	PARK	NONE 1100 Block of CLA...	-122.44683782035	(37.7622559270122	
130366639	ASSAULT	AGGRAVATED ASSAUL...	Sunday	05/05/2012	12:00:...	04:18	INGLESIDE	"ARREST, BOOKED" @ Block of SGTJOH...	-122.444707063455	(37.7249307267936	
030810835	DRIVING UNDER THE ...	DRIVING WHILE UND...	Tuesday	07/09/2003	12:00:...	01:00	SOUTHERN	"ARREST, BOOKED" MASON ST / TURK ST	-122.408953598286	(37.783387873549	
130839567	OTHER OFFENSES	TRAFFIC VIOLATION...	Friday	10/04/2012	12:00:...	20:53	TENDERLOIN	"ARREST, BOOKED" TURK ST / LEAVENN...	-122.414056291891	(37.782793107100...	
070838580	BURGLARY	BURGLARY OF APAR...	Tuesday	08/14/2007	12:00:...	07:00	NORTHERN	NONE 3100 Block of FRA...	-122.426730544229	(37.8034674969672	
080023102	DRUG/NARCOTIC	POSSESSION OF MAR...	Tuesday	03/04/2008	12:00:...	14:23	INGLESIDE	"ARREST, CITED" MISSION ST / PERS...	-122.43597721703	(37.7231288306727	
060711895	OTHER OFFENSES	DRIVERS LICENSE, ...	Wednesday	07/05/2006	12:00:...	15:58	INGLESIDE	"ARREST, CITED" 1200 Block of SAN...	-122.447241150611	(37.720157797125	
040602593	LARCENY/THEFT	GRAND THEFT FROM ...	Wednesday	12/10/2003	12:00:...	09:30	INGLESIDE	NONE 8 Block of MOFFIT...	-122.432787775164	(37.737156674527	
110051822	NON-CRIMINAL	STAY AWAY OR COU...	Monday	01/17/2011	12:00:...	15:35	INGLESIDE	NONE 600 Block of CAMP...	-122.408761072232	(37.7159000951041	
060027038	LARCENY/THEFT	GRAND THEFT FROM ...	Saturday	01/07/2006	12:00:...	22:00	NORTHERN	NONE 100 Block of HEM...	-122.420815202607	(37.7872368476925	

Programmatically

Steps:

1. Create an RDD of Rows from the original RDD
2. Create a schema represented by a StructType matching the structure of Rows in the RDD created in step1.
3. Create DataFrame by applying schema to Row RDD.

Create Dataframe Dynamically (...without case class objects)

1. Create a schema programmatically

```
import org.apache.spark.sql.types.{StructField, StructType,
StringType}

val schema = StructType(Array(
  StructField("IncidentNum", StringType, false),
  StructField("Category", StringType, false),
  StructField("Description", StringType, false),
  StructField("DayOfWeek", StringType, false),
  StructField("DateOfIncident", StringType, false),
  StructField("TimeOfIncident", StringType, false),
  StructField("PdDistrict", StringType, false),
  StructField("Resolution", StringType, false),
  StructField("Address", StringType, false),
  StructField("X", StringType, false),
  StructField("Y", StringType, false),
  StructField("Location", StringType, false)
))
```

2. Convert the data to RDD of Row objects

```
import org.apache.spark.sql.Row

val incidentRowObjects = dataParsed.map(rec => Row(rec(0), rec(1),
rec(2), rec(3), rec(4), rec(5), rec(6), rec(7), rec(8), rec(9),
rec(10), rec(11)))
```

3. Create the dataframe from RDD of Row objects and dynamic schema

```
val incidentsDF = sqlContext.createDataFrame(incidentRowObjects,
schema)
```

4. Use dataframe api and further, register the data frame as Spark SQL table and run SQL statements on top of it

```
incidentsDF.show
```

[IncidentNum]	Category	Description	DayOfWeek	DateOfIncident	TimeOfIncident	PdDistrict	Resolution	Address	X	Y	Location
0504367121	ASSAULT	BATTERY	Wednesday	04/20/2008 12:08...	04:00	MISSION	NONE	18TH ST / CASTRO ST	122.435903864271	37.7608878061245	(37.7608878061245...
080409078	LARCENY/THEFT	GRAND THEFT FROM ...	Sunday	01/13/2008 12:08...	18:00	PARK	NONE	1100 Block of CLA...	122.44683782035	37.7622559270122	(37.7622559270122...
130366639	ASSAULT	AGGRAVATED ASSAUL...	Sunday	05/05/2013 12:08...	04:10	INGLESIDE	"ARREST, BOOKED"	10 Block of SGTJOH...	122.444707063455	37.7249307267936	(37.7249307267936...
030810835	DRIVING UNDER THE ...	DRIVING WHILE UND...	Tuesday	07/09/2003 12:08...	01:00	SOUTHERN	"ARREST, BOOKED"	MASON ST / TURK ST	122.4890535082861	37.7932878735491	(37.7932878735491...
1308395671	OTHER OFFENSES	TRAFFIC VIOLATION...	Friday	10/04/2013 12:08...	20:53	TENDERLOIN	"ARREST, BOOKED"	TURK ST / LEAVENN...	122.414056291091	37.7927931071006	(37.7927931071006...
070833580	BURGLARY	"BURGLARY OF APAR...	Tuesday	08/14/2007 12:08...	07:00	NORTHERN	NONE	3100 Block of FRA...	122.426730544229	37.8034674969672	(37.8034674969672...
080233102	DRUG/NARCOTIC	POSSESSION OF MAR...	Tuesday	03/04/2008 12:08...	14:23	INGLESIDE	"ARREST, CITED"	MISSION ST / PERS...	122.43597721703	37.7231288396727	(37.7231288396727...
060711895	OTHER OFFENSES	"DRIVERS LICENSE...	Wednesday	07/05/2006 12:08...	15:50	INGLESIDE	"ARREST, CITED"	2300 Block of SAN...	122.4472411596111	37.7201577971255	(37.7201577971255...
04062593	LARCENY/THEFT	GRAND THEFT FROM ...	Wednesday	12/10/2007 12:08...	09:30	INGLESIDE	NONE	10 Block of MOFFIT...	122.432787775164	37.7371566745527	(37.7371566745527...
110051822	NON-CRIMINAL	"STAY AWAY OR COU...	Monday	01/17/2011 12:08...	15:35	INGLESIDE	NONE	600 Block of CAMP...	122.40876107222	37.7159008951041	(37.7159008951041...
060027038	LARCENY/THEFT	GRAND THEFT FROM ...	Saturday	01/07/2006 12:08...	22:00	NORTHERN	NONE	100 Block of HEMI...	122.420815282609	37.7872360476925	(37.7872360476925...

UDF in Spark Dataframe

There are two type:

1. UDF for Dataframe API
2. UDF for SQL expressions

UDF for Dataframe API

1. Parse string DateOfIncident field into a time stamp

```

import java.text.SimpleDateFormat
import java.util.Locale
import java.sql.Timestamp // Spark SQL does not support
java.util.Date

def sf = new SimpleDateFormat( "MM/dd/yyyy HH:mm:ss aaa",
Locale.ENGLISH) //Sample value: 10/04/2013 12:00:00 AM
sf.setLenient(true)

def parseDatetimeString(dateStr: String): Timestamp = {
    new Timestamp(sf.parse(dateStr).getTime)
}

```

2. Register the UDF for Dataframe API

```

import org.apache.spark.sql.functions.udf
def toTimestamp = udf(parseDatetimeString _)

```

3. Use the UDF you just created

```

incidentsDF.select($"DateOfIncident",
toTimestamp($"DateOfIncident")).show(10, false)

```

DateOfIncident	UDF(DateOfIncident)
04/20/2005 12:00:00 AM	2005-04-20 12:00:00.0
01/13/2008 12:00:00 AM	2008-01-13 12:00:00.0
05/05/2013 12:00:00 AM	2013-05-05 12:00:00.0
07/08/2003 12:00:00 AM	2003-07-08 12:00:00.0
10/04/2013 12:00:00 AM	2013-10-04 12:00:00.0
08/14/2007 12:00:00 AM	2007-08-14 12:00:00.0
03/04/2008 12:00:00 AM	2008-03-04 12:00:00.0
07/05/2006 12:00:00 AM	2006-07-05 12:00:00.0
12/10/2003 12:00:00 AM	2003-12-10 12:00:00.0
01/17/2011 12:00:00 AM	2011-01-17 12:00:00.0

only showing top 10 rows

UDF for SQL Expression

1. Register UDF for SQL expression

```
sqlContext.udf.register("toTimestamp", parseDatetimeString_)
```

2. Use UDF in SQL statements that you just created

```
sqlContext.sql("select t.IncidentNum, t.DateOfIncident,
toTimestamp(DateOfIncident) `Timestamp` from incidents t").show(10,
false)
```

Save the Dataframe to filesystem

1. Default format is 'parquet'

```
incidentsDF.write.save("/user/cloudera/datasets/incidents")
```

2. To save in 'JSON' format

```
incidentsDF.write.format("json").save
file:/home/cloudera/datasets/incidents_json")
```

Creating Dataframe from certain file format

If you want to use the spark-shell you can provide the list of packages to import dynamically in your shell with "--packages" like

```
spark-shell --packages com.databricks:spark-csv_2.10:1.4.0
```

```
val incidents = sqlContext.read.format("com.databricks.spark.csv").
options(Map("header" -> "true", "inferSchema" -> "true")) .
load("file:/home/cloudera/datasets/crime_incidents.csv")
incidents.show
```

IncidentNum	DateOfIncident	Timestamp
50,436,712	04/20/2005 12:00:00 AM	2005-04-20 12:00:00.0
80,049,078	01/13/2008 12:00:00 AM	2008-01-13 12:00:00.0
130,366,639	05/05/2013 12:00:00 AM	2013-05-05 12:00:00.0
30,810,835	07/08/2003 12:00:00 AM	2003-07-08 12:00:00.0
130,839,567	10/04/2013 12:00:00 AM	2013-10-04 12:00:00.0
70,838,580	08/14/2007 12:00:00 AM	2007-08-14 12:00:00.0
80,233,102	03/04/2008 12:00:00 AM	2008-03-04 12:00:00.0
60,711,805	07/05/2006 12:00:00 AM	2006-07-05 12:00:00.0

Additional Exercise: Creating Dataframe from csv file format

```
// Getting Started with DataFrames!
val df = sqlContext.read.format("com.databricks.spark.csv").
options(Map("header" -> "true", "inferSchema" -> "true")).
load("file:/home/cloudera/datasets/CitiGroup2006_2008")

// Get first 5 rows
df.head(5) // returns an Array
println("\n")
for(line <- df.head(10)){
    println(line)
}

// Get column names
df.columns

// Find out DataTypes
// Print Schema
df.printSchema()

// Describe DataFrame Numerical Columns
df.describe()

// Select columns .transform().action()
df.select("Volume").show()
```

```

// Multiple Columns
df.select($"Date",$"Close").show(2)

// Creating New Columns
val df2 = df.withColumn("HighPlusLow",df("High")-df("Low"))

// Show result
df2.columns
df2.printSchema()

// Recheck Head
df2.head(5)

// Renaming Columns (and selecting some more)
df2.select(df2("HighPlusLow").as("HPL"),df2("Close")).show()

```

Hands on Assignment

1. Use the Netflix_2011_2016.csv file to Answer and complete the commented tasks below!
2. Load the Netflix Stock CSV File, have Spark infer the data types.
3. What are the column names?
4. What does the Schema look like?
5. Print out the first 5 columns.
6. Use describe() to learn about the DataFrame.
7. Create a new dataframe with a column called HV Ratio that
8. is the ratio of the High Price versus volume of stock traded for a day.
9. What day had the Peak High in Price?
10. What is the mean of the Close column?
11. What is the max and min of the Volume column?
12. How many days was the Close lower than \$ 600?
13. What percentage of the time was the High greater than \$500 ?
14. What is the Pearson correlation between High and Volume?
15. What is the max High per year?
16. What is the average Close for each Calender Month?

Exercise 7: Spark Streaming

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
█
```

```
cd /usr/lib/spark/bin
```

```
spark-submit --class SparkStreamReceiver --master local[3] --verbose
/home/cloudera/Desktop/Spark/Code/StreamApp/target/StreamApp-0.0.1-
SNAPSHOT.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)
```