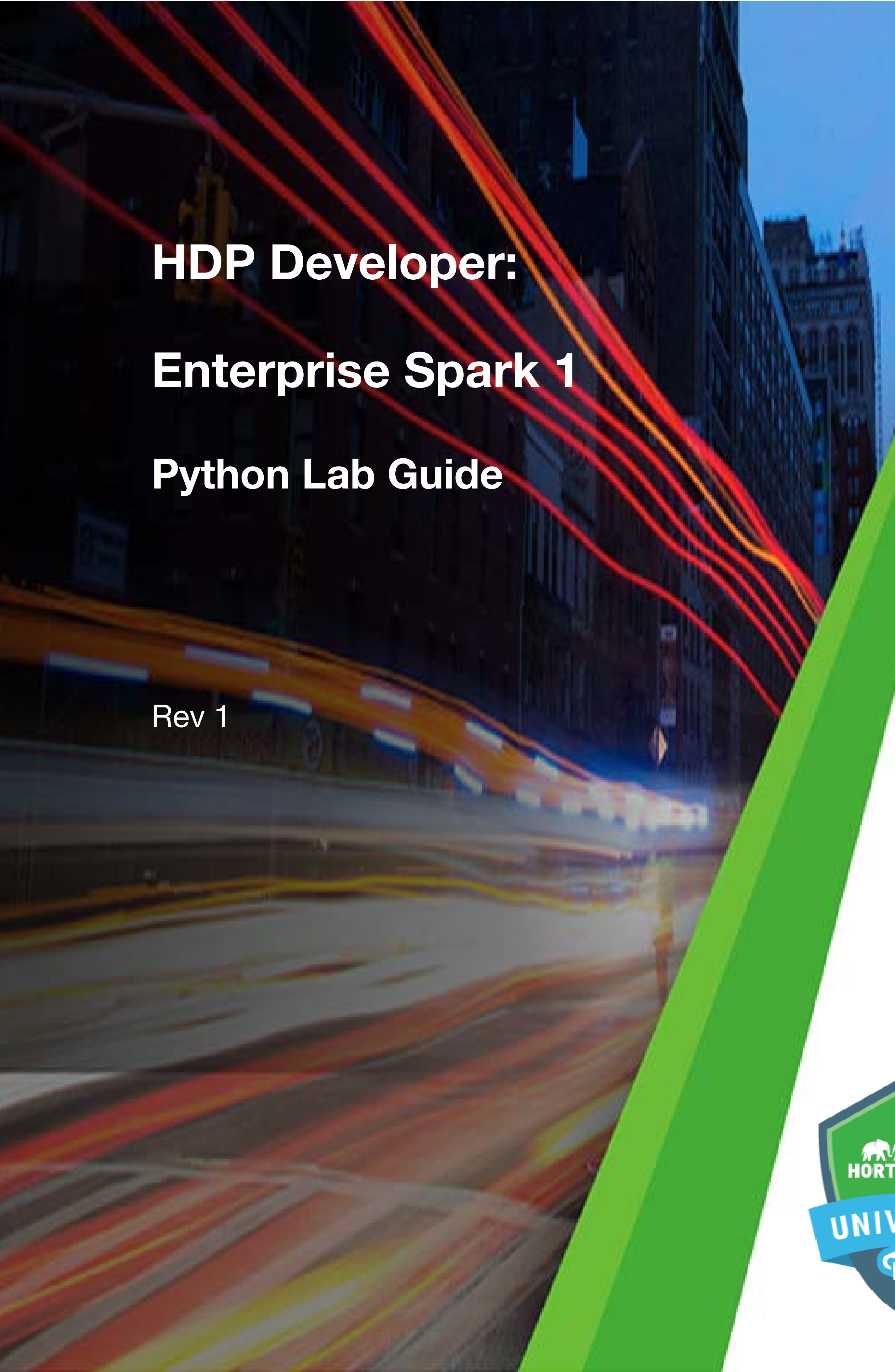


HDP Developer:

Enterprise Spark 1

Python Lab Guide

Rev 1

This instructional program, including all material provided herein, is supplied without any guarantees from Hortonworks, Inc. Hortonworks, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

Become a ***Hortonworks Certified Professional*** and establish your credentials:

- HDP Certified Developer: for Hadoop developers using frameworks like Pig, Hive, Sqoop and Flume.

- HDP Certified Administrator: for Hadoop administrators who deploy and manage Hadoop clusters.

- HDP Certified Developer: Java: for Hadoop developers who design, develop and architect Hadoop-based solutions written in the Java programming language.

- HDP Certified Developer: Spark: for Hadoop developers who write and deploy applications for the Spark framework.

How to Register: Visit www.examslocal.com and search for “Hortonworks” to register for an exam. The cost of each exam is $250 USD, and you can take the exam anytime, anywhere using your own computer. For more details, including a list of exam objectives and instructions on how to attempt our practice exams, visit http://hortonworks.com/training/certification/

Earn Digital Badges: Hortonworks Certified Professionals receive a digital badge for each certification earned. Display your badges proudly on your résumé, LinkedIn profile, email signature, etc.

Self Paced Learning Library

On Demand Learning

Hortonworks University Self-Paced Learning Library is an on-demand dynamic repository of content that is accessed using a Hortonworks University account. Learners can view lessons anywhere, at any time, and complete lessons at their own pace. Lessons can be stopped and started, as needed, and completion is tracked via the Hortonworks University Learning Management System.

Hortonworks University courses are designed and developed by Hadoop experts and provide an immersive and valuable real world experience. In our scenario-based training courses, we offer unmatched depth and expertise. We prepare you to be an expert with highly valued, practical skills and prepare you to successfully complete Hortonworks Technical Certifications.

Target Audience: Hortonworks University Self-Paced Learning Library is designed for those new to Hadoop, as well as architects, developers, analysts, data scientists, and IT decision makers. It is essentially for anyone who desires to learn more about Apache Hadoop and the Hortonworks Data Platform.

Duration: Access to the Hortonworks University Self-Paced Learning Library is provided for a 12-month period per individual named user. The subscription includes access to over 400 hours of learning lessons.

The online library accelerates time to Hadoop competency. In addition, the content is constantly being expanded with new material, on an ongoing basis.

Visit: http://hortonworks.com/training/class/hortonworks-university-self-paced-learning-library/

Table of Contents

Lab 0: Pre-lab Setup

About This Lab

Objective:
Set up the lab environment and confirm functionality

File Locations:
N/A

Successful Outcome:
User will set up the HDP cluster and verify login

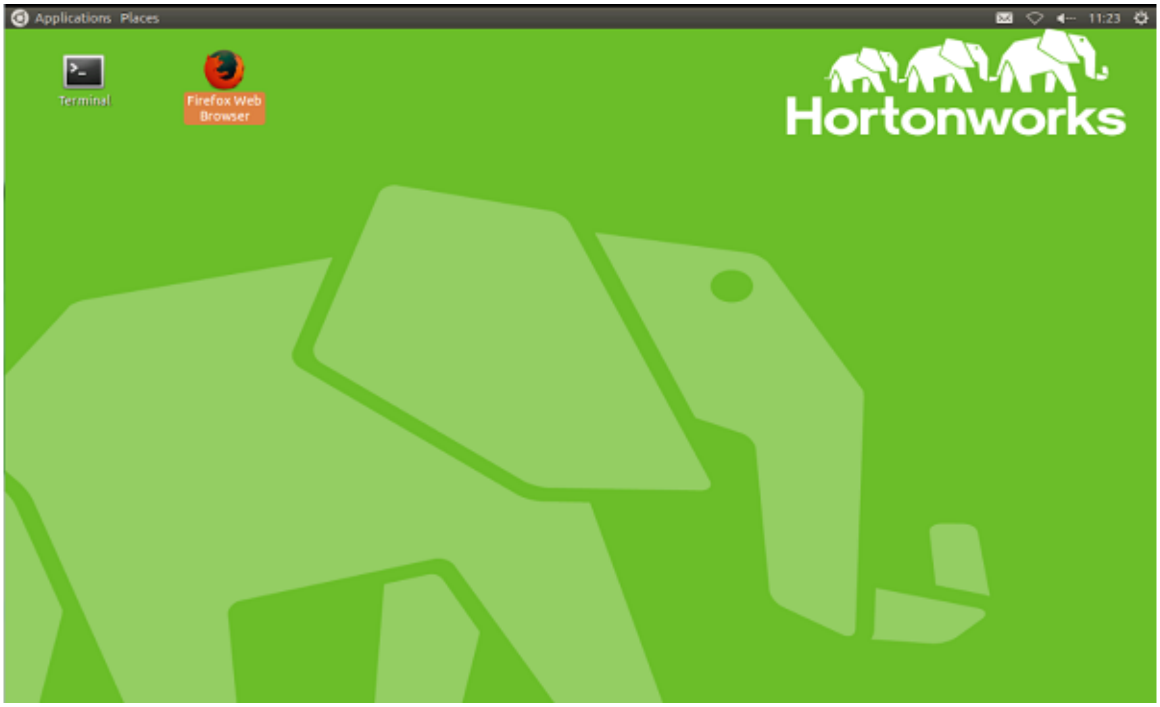
Before You Begin:
Connect to the lab Environment

Lab Steps

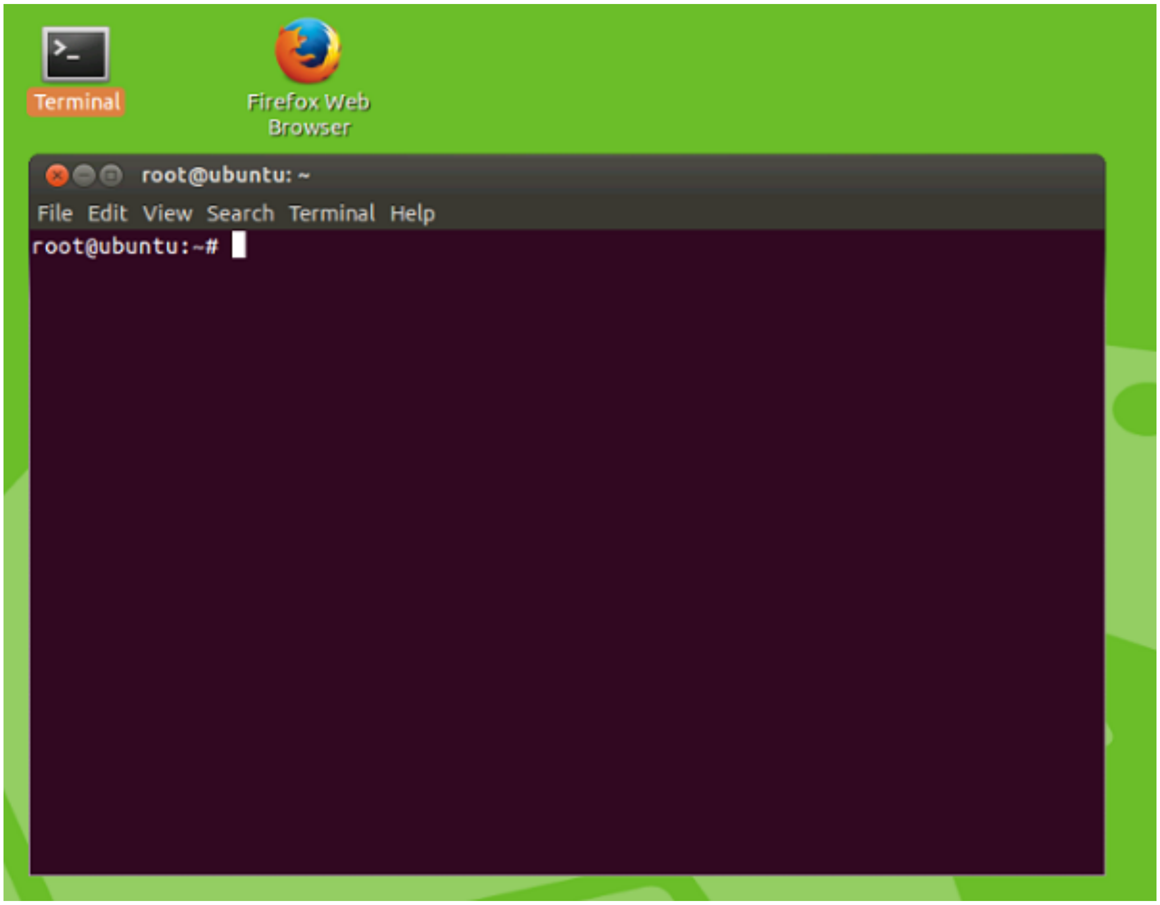
Perform the following steps:

1. **Start the HDP cluster.**

    a. Connect to the lab environment.



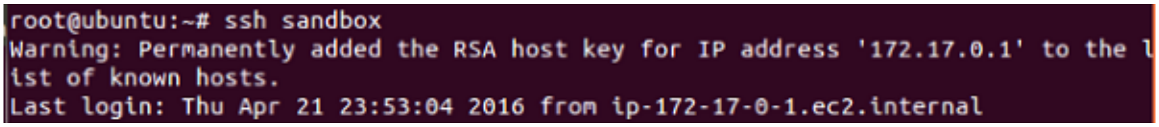
b. Double-click on the Terminal icon on the desktop.



```
root@ubuntu:~#
```

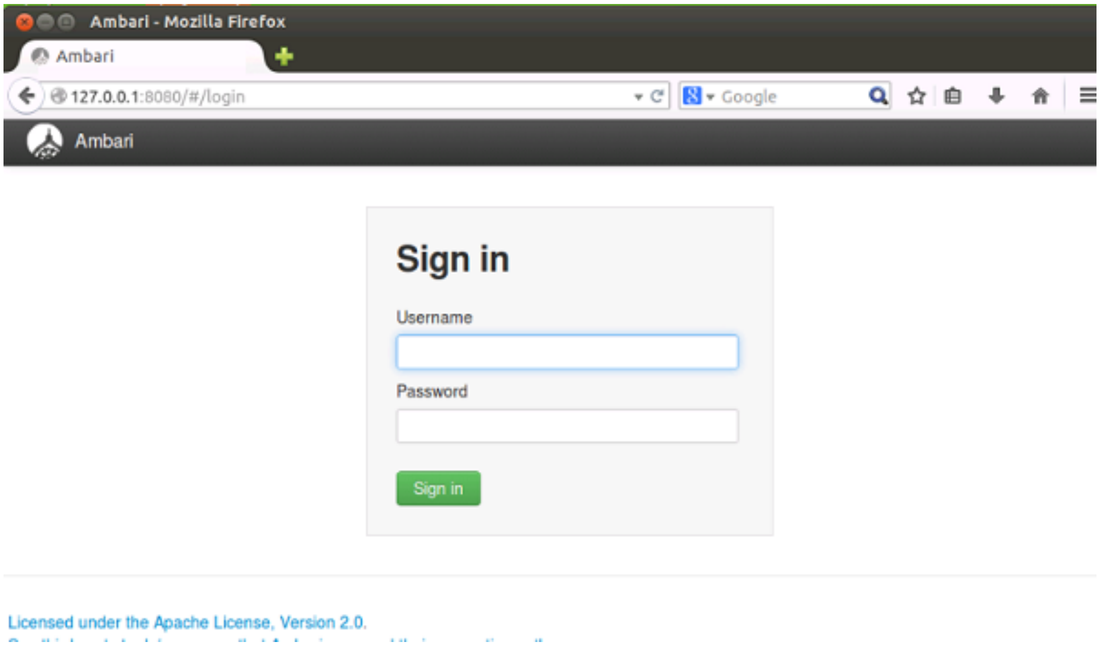
c. Use SSH to connect to the Docker container – named "sandbox" – that has been a single-node HDP cluster installation configured.

```
# ssh sandbox
```



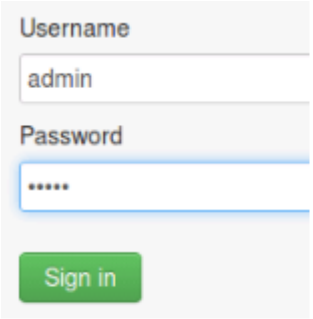
```
root@ubuntu:~# ssh sandbox
Warning: Permanently added the RSA host key for IP address '172.17.0.1' to the l
ist of known hosts.
Last login: Thu Apr 21 23:53:04 2016 from ip-172-17-0-1.ec2.internal
```

2. **Verify and, if necessary, start HDP cluster services.**
    a. Open a Firefox web browser and log into the Ambari Web UI using http://sandbox:8080.



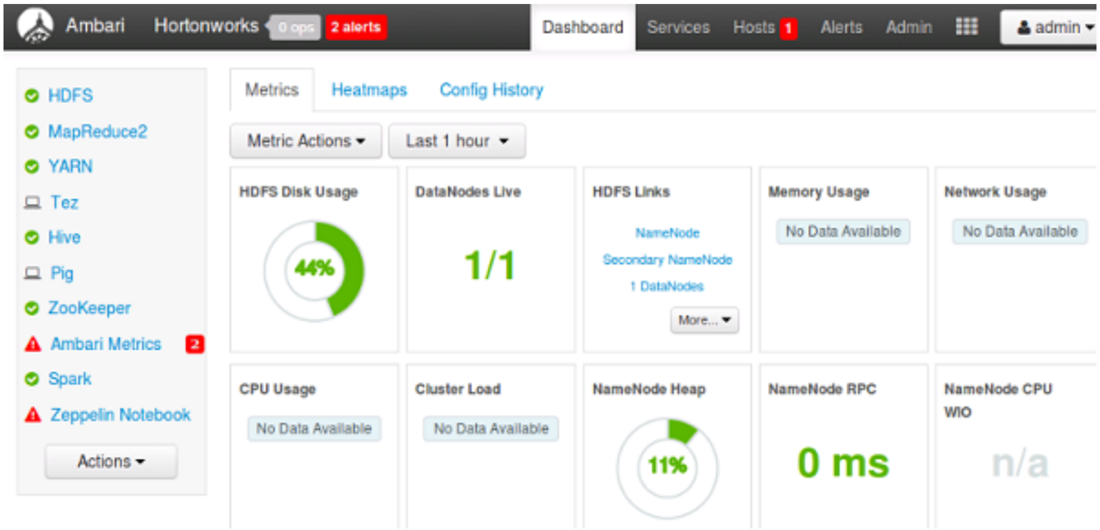
Licensed under the Apache License, Version 2.0.

    b. Supply a username and password of admin and admin, then click the Sign in button to get to the Ambari Web UI dashboard.

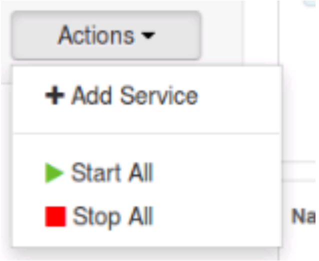


Username: admin

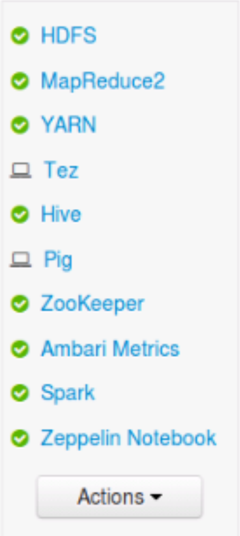
Password: •••••



c. All services should be running. If not, start any stopped services by clicking on the Actions button at the bottom left and selecting Start All.



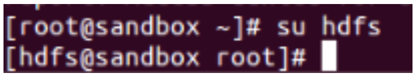
d. If a restart was necessary, give the services a couple of minutes to start. One or more of them may initially report failure, but after waiting will go green. When everything has settled, your dashboard list of services should look similar to this:



3. **Confirm HDFS (Hadoop Distributed File System) access from the command line.**

   a. Go back to the terminal window that is connected to the sandbox Docker container (reopen and reconnect if necessary) and switch users so that you can run HDFS administrative commands.

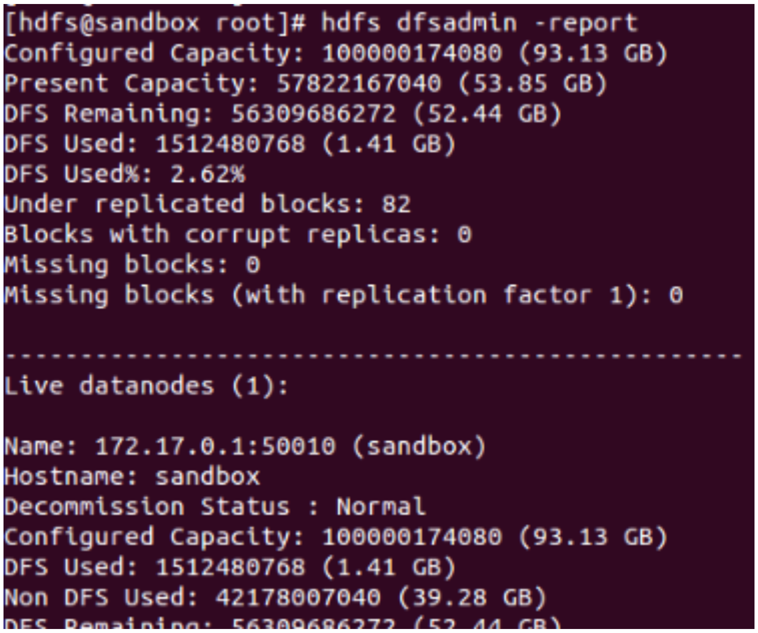
```
# su hdfs
```



```
[root@sandbox ~]# su hdfs
[hdfs@sandbox root]# 
```

   b. To verify HDFS connectivity, run the hdfs dfsadmin -report command. Verify that it provides output similar to the screenshot provided.

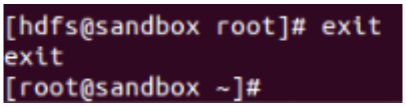
```
# hdfs dfsadmin -report
```



```
[hdfs@sandbox root]# hdfs dfsadmin -report
Configured Capacity: 100000174080 (93.13 GB)
Present Capacity: 57822167040 (53.85 GB)
DFS Remaining: 56309686272 (52.44 GB)
DFS Used: 1512480768 (1.41 GB)
DFS Used%: 2.62%
Under replicated blocks: 82
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (1):

Name: 172.17.0.1:50010 (sandbox)
Hostname: sandbox
Decommission Status : Normal
Configured Capacity: 100000174080 (93.13 GB)
DFS Used: 1512480768 (1.41 GB)
Non DFS Used: 42178007040 (39.28 GB)
DFS Remaining: 56309686272 (52.44 GB)
```

   c. Exit the HDFS administrative user and go back to being the root user.

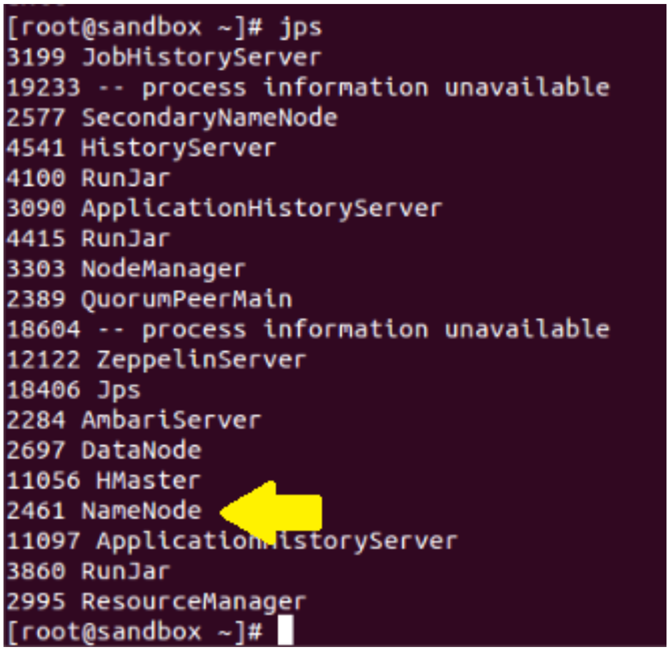
```
# exit
```



```
[hdfs@sandbox root]# exit
exit
[root@sandbox ~]#
```

   d. Run the `jps` command and verify that a process called NameNode is running.

```
# jps
```



```
[root@sandbox ~]# jps
3199 JobHistoryServer
19233 -- process information unavailable
2577 SecondaryNameNode
4541 HistoryServer
4100 RunJar
3090 ApplicationHistoryServer
4415 RunJar
3303 NodeManager
2389 QuorumPeerMain
18604 -- process information unavailable
12122 ZeppelinServer
18406 Jps
2284 AmbariServer
2697 DataNode
11056 HMaster
2461 NameNode
11097 ApplicationHistoryServer
3860 RunJar
2995 ResourceManager
[root@sandbox ~]#
```

Result

You have successfully connected to your lab environment, used SSH to connect to the HDP cluster Docker container, started Ambari and all HDP services, and verified connection to HDFS and operation of the NameNode process.

Lab 1: Using HDFS Commands

About This Lab

Objective:
View, add, manipulate, and remove files and directories to and from HDFS using `hdfs dfs` commands.

File Locations:
`/root/spark/data/`

Successful Outcome:
You will have added, manipulated, and deleted several files and folders in HDFS

Before You Begin:
You should be logged in to your lab environment

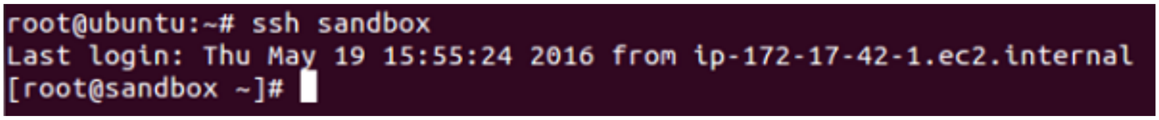
Lab Steps

Perform the following steps:

1. **View the hdfs dfs command.**

    a. Open a Terminal window and use ssh to connect to the sandbox virtual machine.

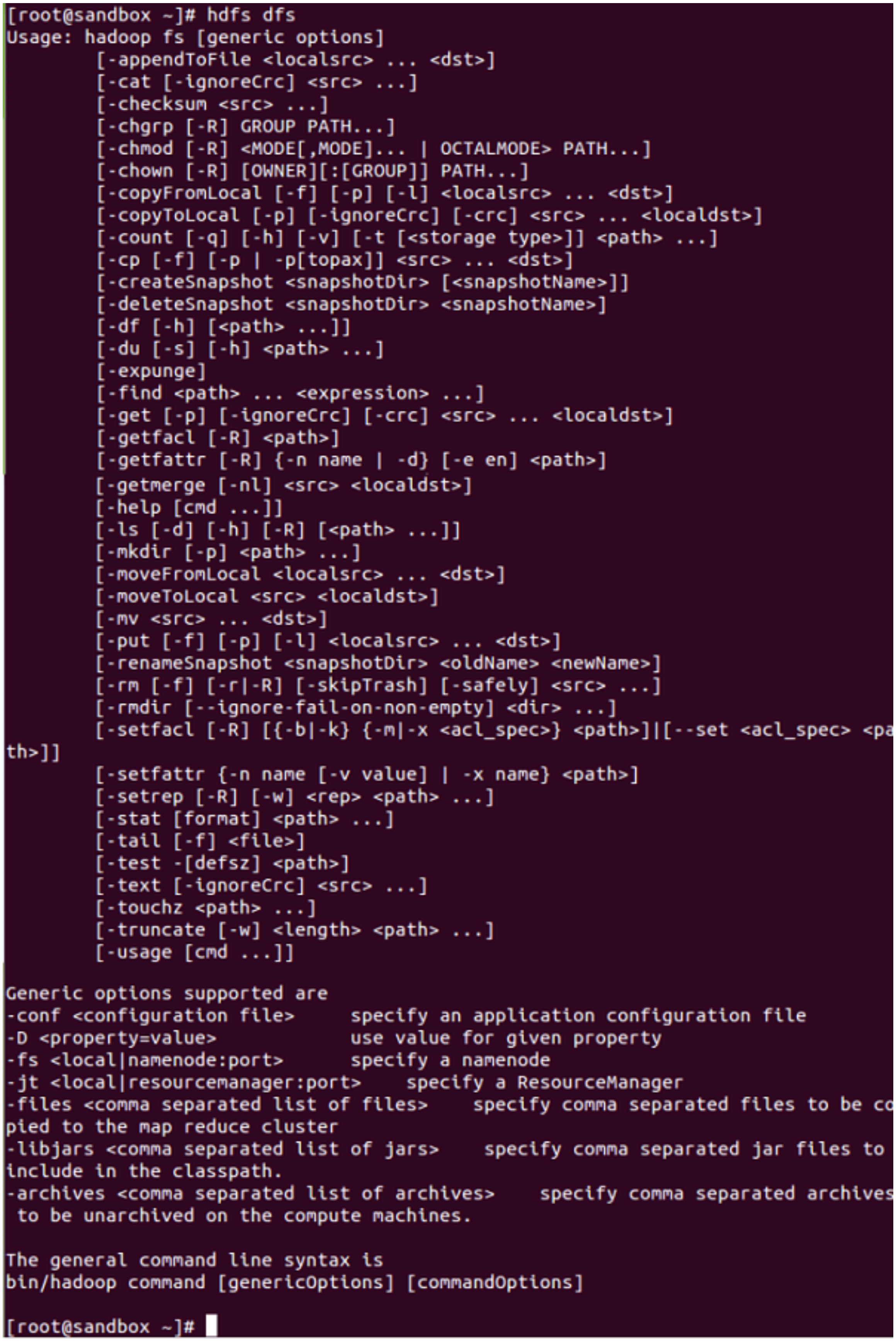
```
# ssh sandbox
```



```
root@ubuntu:~# ssh sandbox
Last login: Thu May 19 15:55:24 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]#
```

    b. From the command line, enter the hdfs dfs command with no arguments to view its usage.

```
# hdfs dfs
```

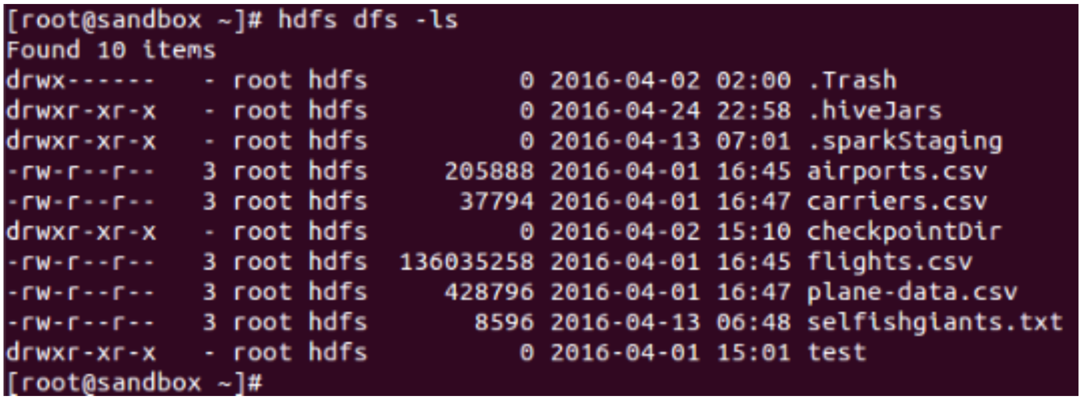


```
[root@sandbox ~]# hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] [-l] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <pa
th>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
        [-tail [-f] <file>]
        [-test -[defsz] <path>]
        [-text [-ignoreCrc] <src> ...]
        [-touchz <path> ...]
        [-truncate [-w] <length> <path> ...]
        [-usage [cmd ...]]

Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|resourcemanager:port>    specify a ResourceManager
-files <comma separated list of files>    specify comma separated files to be co
pied to the map reduce cluster
-libjars <comma separated list of jars>    specify comma separated jar files to
include in the classpath.
-archives <comma separated list of archives>    specify comma separated archives
 to be unarchived on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]

[root@sandbox ~]# 
```

2. Create directories in HDFS.

a. Enter the `hdfs dfs -ls` command with no directory specified to view the contents of the current user's home directory in HDFS. Since you are logged in as the user root, the typical home directory location will be `/user/root`.

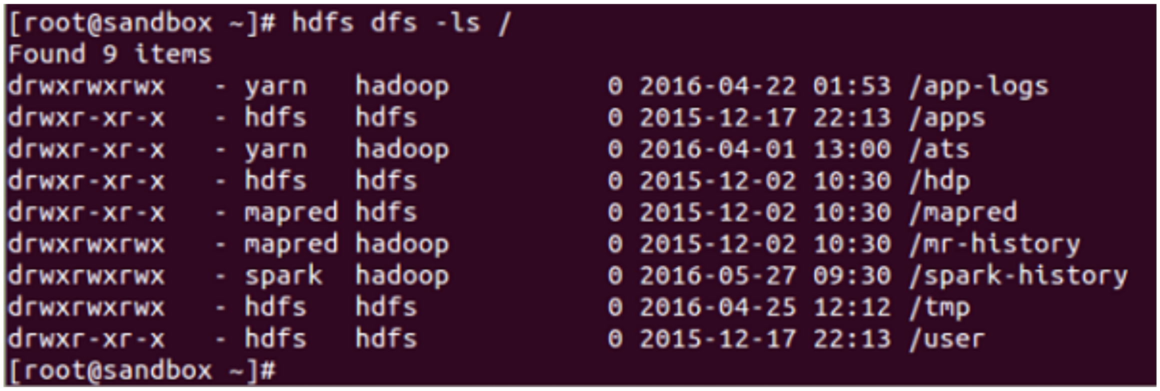
```
# hdfs dfs –ls
```



```
[root@sandbox ~]# hdfs dfs -ls
Found 10 items
drwx------   - root hdfs          0 2016-04-02 02:00 .Trash
drwxr-xr-x   - root hdfs          0 2016-04-24 22:58 .hiveJars
drwxr-xr-x   - root hdfs          0 2016-04-13 07:01 .sparkStaging
-rw-r--r--   3 root hdfs     205888 2016-04-01 16:45 airports.csv
-rw-r--r--   3 root hdfs      37794 2016-04-01 16:47 carriers.csv
drwxr-xr-x   - root hdfs          0 2016-04-02 15:10 checkpointDir
-rw-r--r--   3 root hdfs  136035258 2016-04-01 16:45 flights.csv
-rw-r--r--   3 root hdfs     428796 2016-04-01 16:47 plane-data.csv
-rw-r--r--   3 root hdfs       8596 2016-04-13 06:48 selfishgiants.txt
drwxr-xr-x   - root hdfs          0 2016-04-01 15:01 test
[root@sandbox ~]#
```

b. Run the command again, but this time specify the root folder for all of HDFS.

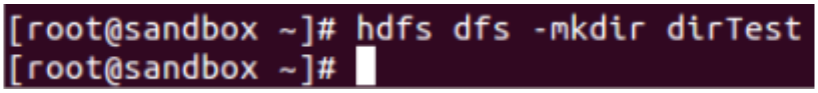
```
# hdfs dfs –ls /
```



```
[root@sandbox ~]# hdfs dfs -ls /
Found 9 items
drwxrwxrwx   - yarn   hadoop          0 2016-04-22 01:53 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-12-17 22:13 /apps
drwxr-xr-x   - yarn   hadoop          0 2016-04-01 13:00 /ats
drwxr-xr-x   - hdfs   hdfs            0 2015-12-02 10:30 /hdp
drwxr-xr-x   - mapred hdfs            0 2015-12-02 10:30 /mapred
drwxrwxrwx   - mapred hadoop          0 2015-12-02 10:30 /mr-history
drwxrwxrwx   - spark  hadoop          0 2016-05-27 09:30 /spark-history
drwxrwxrwx   - hdfs   hdfs            0 2016-04-25 12:12 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2015-12-17 22:13 /user
[root@sandbox ~]#
```

c. Create a directory named `dirTest` in the current user's home directory in HDFS.

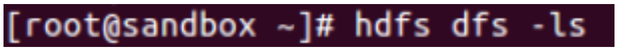
```
# hdfs dfs -mkdir dirTest
```



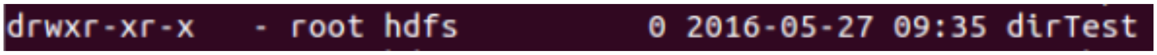
```
[root@sandbox ~]# hdfs dfs -mkdir dirTest
[root@sandbox ~]# █
```

d. Verify the folder was created successfully.

```
# hdfs dfs -mkdir dirTest
```



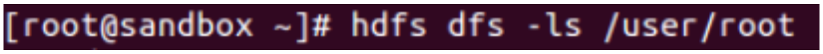
```
[root@sandbox ~]# hdfs dfs -ls
```



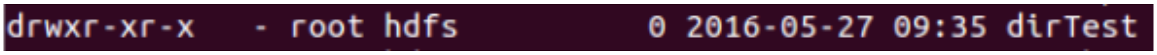
```
drwxr-xr-x   - root hdfs          0 2016-05-27 09:35 dirTest
```

e. Verify that this directory was created in the user's home directory.

```
# hdfs dfs -ls /user/root
```



```
[root@sandbox ~]# hdfs dfs -ls /user/root
```

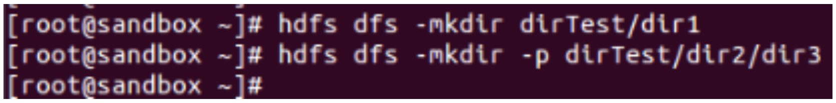


```
drwxr-xr-x   - root hdfs          0 2016-05-27 09:35 dirTest
```

NOTE:
There is no difference between performing the `-ls` command when you specify no directories and when you specify the user's home directory. All commands will be executed in the user's home directory unless otherwise specified.

f. Use `-mkdir` to create subdirectory `dir1` in the `dirTest` directory. Then run the command again with the `-p` option to create an additional subdirectory, `dir2`, which also contains its own subdirectory, `dir3`.

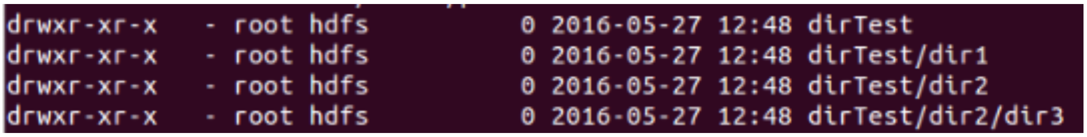
```
# hdfs dfs -mkdir dirTest/dir1

# hdfs dfs -mkdir -p dirTest/dir2/dir3
```



```
[root@sandbox ~]# hdfs dfs -mkdir dirTest/dir1
[root@sandbox ~]# hdfs dfs -mkdir -p dirTest/dir2/dir3
[root@sandbox ~]#
```

g. Run the `hdfs dfs -ls -R` command to recursively view the contents of the user's home directory, and verify that all three directories from the previous step were successfully created.

```
# hdfs dfs -ls -R
```

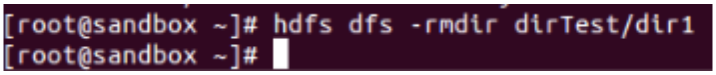


```
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest/dir1
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest/dir2
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest/dir2/dir3
```

3. Delete directories in HDFS.

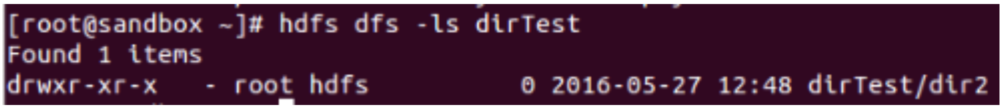
a. Delete the `dir1` directory and verify it no longer exists.

```
# hdfs dfs -rmdir dirTest/dir1
```



```
[root@sandbox ~]# hdfs dfs -rmdir dirTest/dir1
[root@sandbox ~]# 
```

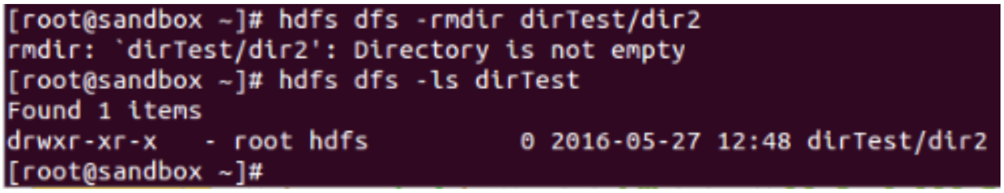
```
# hdfs dfs -ls dirTest
```



```
[root@sandbox ~]# hdfs dfs -ls dirTest
Found 1 items
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest/dir2
```

b. This command works because the directory is empty. Run the command again, and this time try to delete the dir2 directory and note the error message. Then verify that the directory still exists.

```
# hdfs dfs -rmdir dirTest/dir2

# hdfs dfs -ls dirTest
```



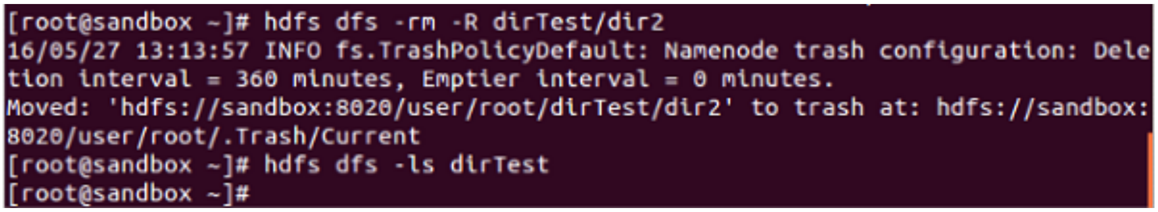
```
[root@sandbox ~]# hdfs dfs -rmdir dirTest/dir2
rmdir: `dirTest/dir2': Directory is not empty
[root@sandbox ~]# hdfs dfs -ls dirTest
Found 1 items
drwxr-xr-x   - root hdfs          0 2016-05-27 12:48 dirTest/dir2
[root@sandbox ~]#
```

c. To delete a directory and all of its contents, use `hdfs dfs -rm -R <directory path>`.

WARNING:
Be very careful not to run this without specifying a directory, as the default behavior would be to delete the user's home directory and all contents (in our case, the `/user/root` directory and everything it contains).

Use this command to delete the dir2 directory and its contents, and verify that the directory has been deleted.

```
# hdfs dfs –rm -R dirTest/dir2

# hdfs dfs –ls dirTest
```

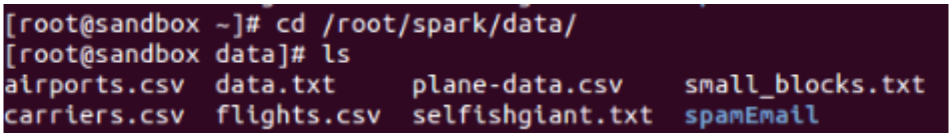


```
[root@sandbox ~]# hdfs dfs -rm -R dirTest/dir2
16/05/27 13:13:57 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele
tion interval = 360 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://sandbox:8020/user/root/dirTest/dir2' to trash at: hdfs://sandbox:
8020/user/root/.Trash/Current
[root@sandbox ~]# hdfs dfs -ls dirTest
[root@sandbox ~]#
```

4. **Upload, copy, and delete HDFS files.**

   a. The sandbox container image should be preloaded with some test files. Change directories to /root/spark/data/ and view the contents of this directory.

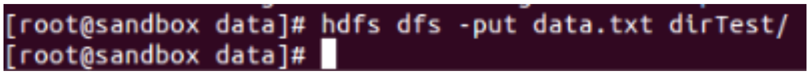
```
# cd /root/spark/data/

# ls
```



```
[root@sandbox ~]# cd /root/spark/data/
[root@sandbox data]# ls
airports.csv  data.txt     plane-data.csv    small_blocks.txt
carriers.csv  flights.csv  selfishgiant.txt  spamEmail
```

   b. Put the data.txt file into the `dirTest` directory in HDFS.

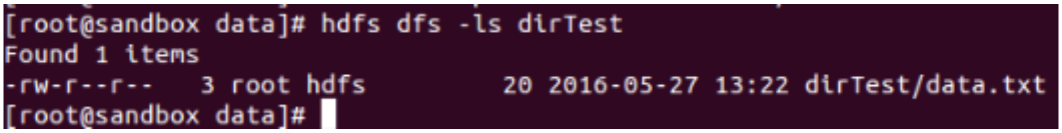
```
# hdfs dfs -put data.txt dirTest/
```



```
[root@sandbox data]# hdfs dfs -put data.txt dirTest/
[root@sandbox data]# 
```

   c. Verify the file was uploaded successfully.

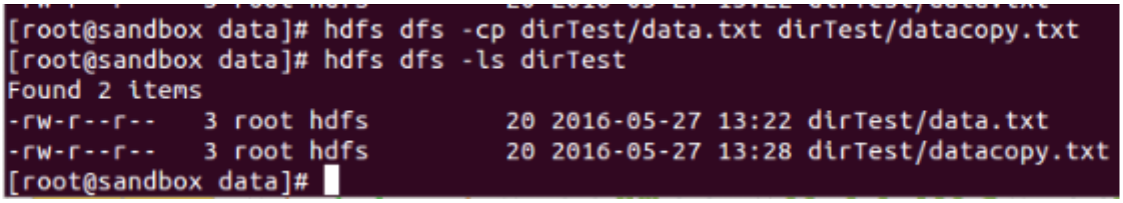
```
# hdfs dfs -ls dirTest
```



```
[root@sandbox data]# hdfs dfs -ls dirTest
Found 1 items
-rw-r--r--   3 root hdfs         20 2016-05-27 13:22 dirTest/data.txt
[root@sandbox data]# 
```

d. Create a copy of the `data.txt` file named `datacopy.txt` and verify the operation was successful.

```
# hdfs dfs -cp dirTest/data.txt dirTest/datacopy.txt

# hdfs dfs –ls dirTest
```

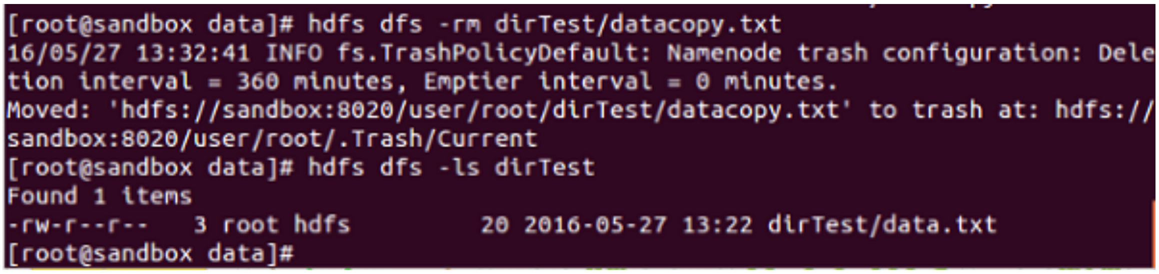


```
[root@sandbox data]# hdfs dfs -cp dirTest/data.txt dirTest/datacopy.txt
[root@sandbox data]# hdfs dfs -ls dirTest
Found 2 items
-rw-r--r--   3 root hdfs         20 2016-05-27 13:22 dirTest/data.txt
-rw-r--r--   3 root hdfs         20 2016-05-27 13:28 dirTest/datacopy.txt
[root@sandbox data]#
```

QUESTION:
What do you think would have happened if the dirTest directory had not been explicitly specified as the location for the `datacopy.txt` file?

e. Now delete the `datacopy.txt` file and verify it has been removed.

```
# hdfs dfs -rm dirTest/datacopy.txt

# hdfs dfs –ls dirTest
```

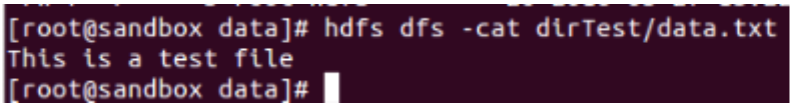


```
[root@sandbox data]# hdfs dfs -rm dirTest/datacopy.txt
16/05/27 13:32:41 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele
tion interval = 360 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://sandbox:8020/user/root/dirTest/datacopy.txt' to trash at: hdfs://
sandbox:8020/user/root/.Trash/Current
[root@sandbox data]# hdfs dfs -ls dirTest
Found 1 items
-rw-r--r--   3 root hdfs         20 2016-05-27 13:22 dirTest/data.txt
[root@sandbox data]#
```

5. **View, download, and download merged files in HDFS.**

a. View the contents of the `data.txt` file in HDFS.

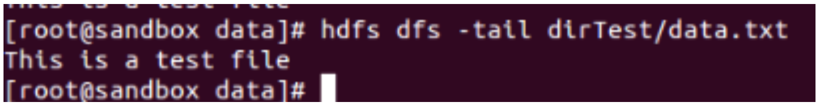
```
# hdfs dfs -cat dirTest/data.txt
```



```
[root@sandbox data]# hdfs dfs -cat dirTest/data.txt
This is a test file
[root@sandbox data]#
```

OR

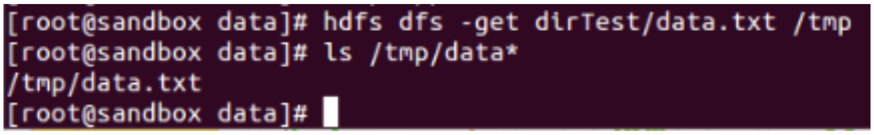
```
# hdfs dfs -tail dirTest/data.txt
```



```
[root@sandbox data]# hdfs dfs -tail dirTest/data.txt
This is a test file
[root@sandbox data]#
```

b. **Download the `data.txt` file from HDFS to the `/tmp` directory on the local file system and verify the operation was successful.**

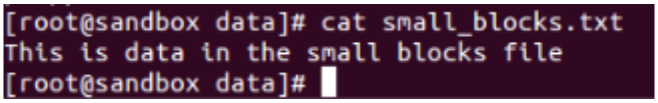
```
# hdfs dfs -get dirTest/data.txt /tmp

# ls /tmp/data*
```



```
[root@sandbox data]# hdfs dfs -get dirTest/data.txt /tmp
[root@sandbox data]# ls /tmp/data*
/tmp/data.txt
[root@sandbox data]# █
```

c. **View the contents of the `small_blocks.txt` file on the local file system. It should be in the current directory.**

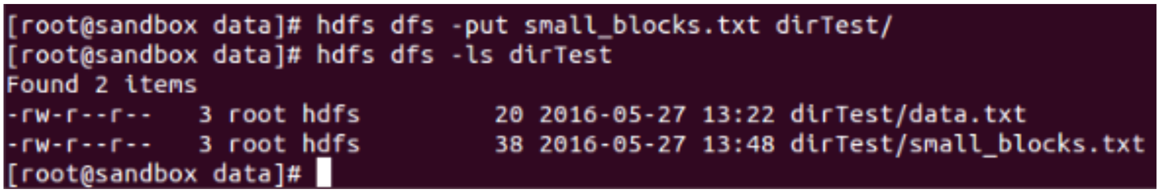
```
# cat small_blocks.txt
```



```
[root@sandbox data]# cat small_blocks.txt
This is data in the small blocks file
[root@sandbox data]# █
```

d. **Upload the `small_blocks.txt` into the `dirTest` folder in HDFS and verify that you now have two files in `dirTest`.**

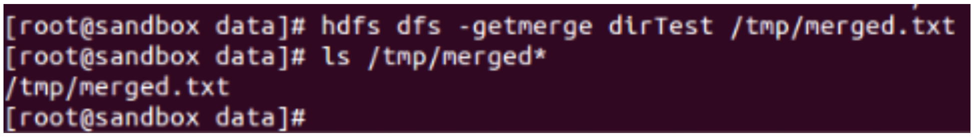
```
# hdfs dfs -put small_blocks.txt dirTest/

# hdfs dfs -ls dirTest
```



```
[root@sandbox data]# hdfs dfs -put small_blocks.txt dirTest/
[root@sandbox data]# hdfs dfs -ls dirTest
Found 2 items
-rw-r--r--   3 root hdfs         20 2016-05-27 13:22 dirTest/data.txt
-rw-r--r--   3 root hdfs         38 2016-05-27 13:48 dirTest/small_blocks.txt
[root@sandbox data]# █
```

e. **Merge and download all of the contents of the `dirTest` directory in HDFS to a file named `merged.txt` in the `/tmp` directory on the local file system. Verify that the `merged.txt` file was successfully created.**

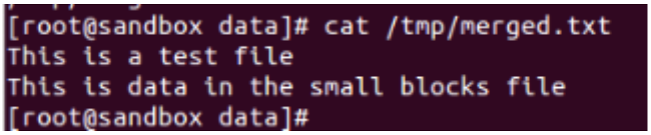
```
# hdfs dfs -getmerge dirTest /tmp/merged.txt

# ls /tmp/merged*
```



```
[root@sandbox data]# hdfs dfs -getmerge dirTest /tmp/merged.txt
[root@sandbox data]# ls /tmp/merged*
/tmp/merged.txt
[root@sandbox data]#
```

View the contents of the `merged.txt` file to confirm that it contains the contents of both files that were in the `dirTest` directory.

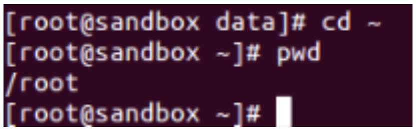
```
# cat /tmp/merged.txt
```



```
[root@sandbox data]# cat /tmp/merged.txt
This is a test file
This is data in the small blocks file
[root@sandbox data]#
```

f. Change directories back to the root user's home directory.

```
# cd ~

# pwd
```



```
[root@sandbox data]# cd ~
[root@sandbox ~]# pwd
/root
[root@sandbox ~]#
```

Result

You have successfully created, manipulated, and deleted files and directories in HDFS.

Lab 2: Introduction to Spark REPLs and Zeppelin

About This Lab

Objective:
Access and browse Spark REPLs and Zeppelin

File Locations:
N/A

Successful Outcome:
Use Spark REPLs and browse Zeppelin

Before You Begin:
Complete the Pre-Lab and confirm cluster operation

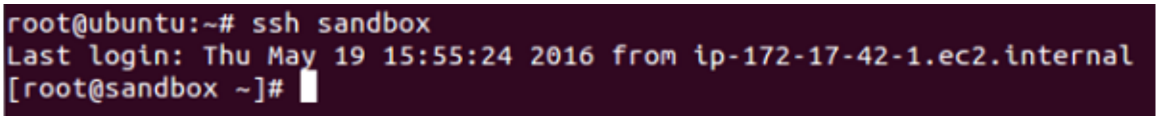
Lab Steps

Perform the following steps:

1. **Access the Spark REPLs.**

   a. Open a Terminal window and use ssh to connect to the sandbox virtual machine.

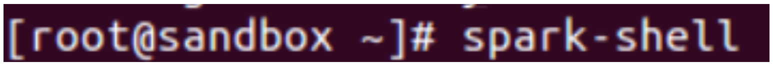
```
# ssh sandbox
```



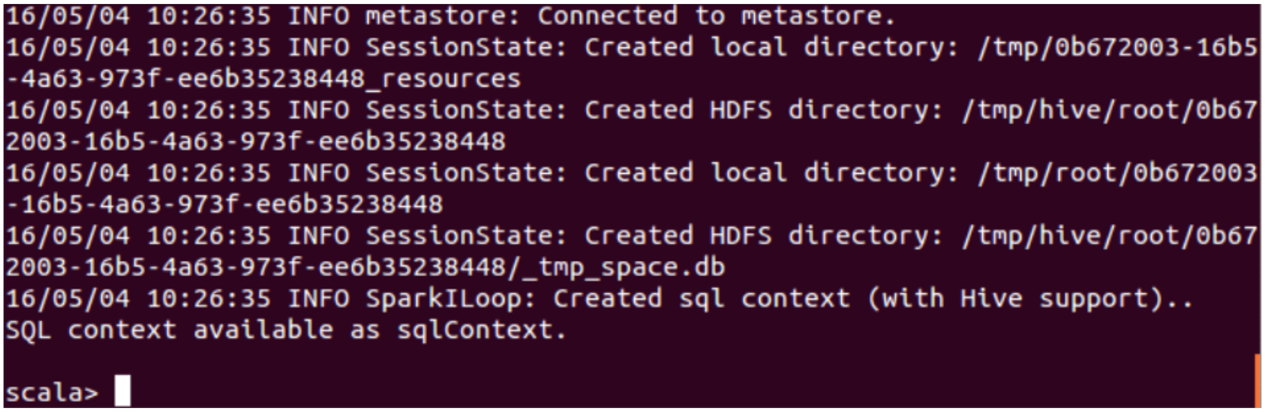
```
root@ubuntu:~# ssh sandbox
Last login: Thu May 19 15:55:24 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]#
```

   b. Run the Spark REPL for Scala.

```
# spark-shell
```



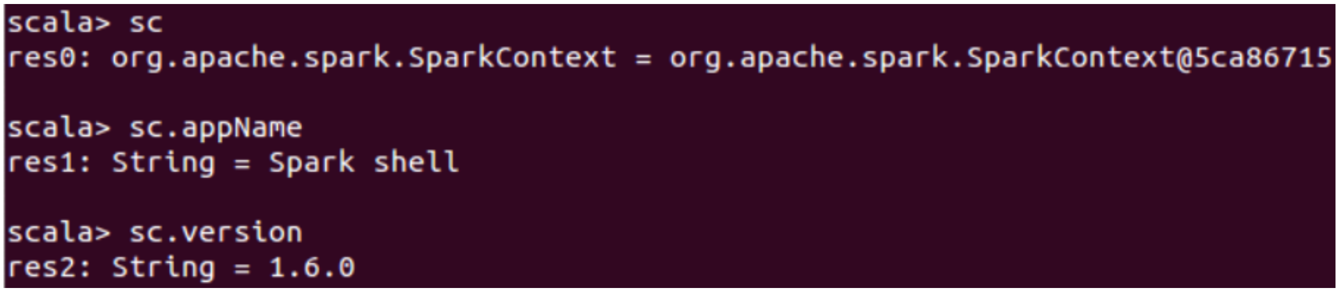
```
[root@sandbox ~]# spark-shell
```



```
16/05/04 10:26:35 INFO metastore: Connected to metastore.
16/05/04 10:26:35 INFO SessionState: Created local directory: /tmp/0b672003-16b5
-4a63-973f-ee6b35238448_resources
16/05/04 10:26:35 INFO SessionState: Created HDFS directory: /tmp/hive/root/0b67
2003-16b5-4a63-973f-ee6b35238448
16/05/04 10:26:35 INFO SessionState: Created local directory: /tmp/root/0b672003
-16b5-4a63-973f-ee6b35238448
16/05/04 10:26:35 INFO SessionState: Created HDFS directory: /tmp/hive/root/0b67
2003-16b5-4a63-973f-ee6b35238448/_tmp_space.db
16/05/04 10:26:35 INFO SparkILoop: Created sql context (with Hive support)..
SQL context available as sqlContext.

scala>
```

c. View the values for the `SparkContext`, `appname`, and `version`.

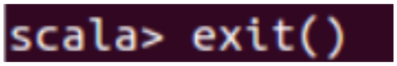
```
scala> sc

scala> sc.appname

scala> sc.version
```



```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@5ca86715

scala> sc.appName
res1: String = Spark shell

scala> sc.version
res2: String = 1.6.0
```

d. Exit the Spark Scala REPL.

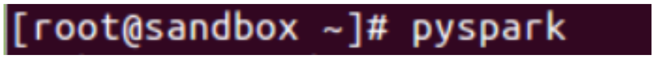
```
scala> exit()
```



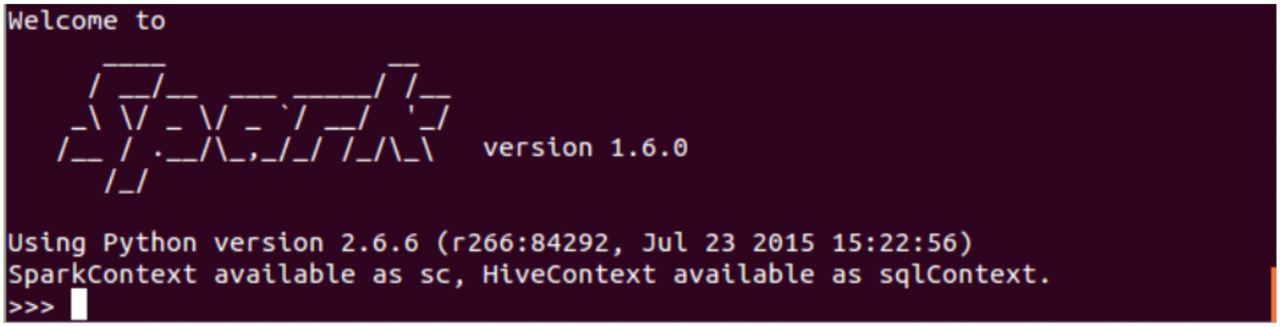
```
scala> exit()
```

e. Run the Spark REPL for Python.

```
# pyspark
```



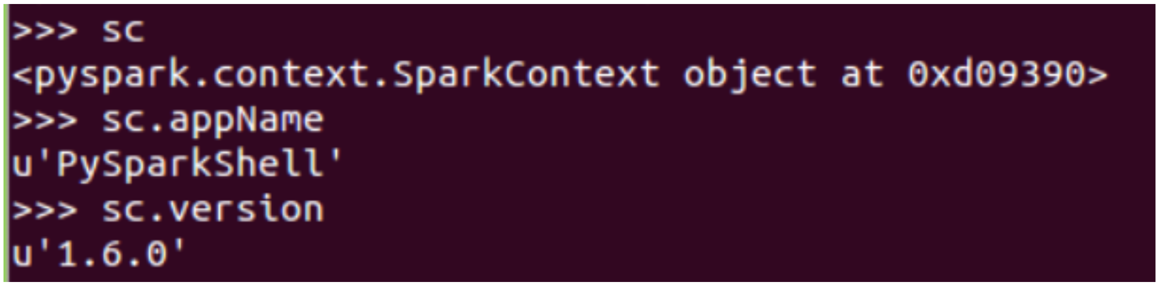
```
[root@sandbox ~]# pyspark
```



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

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

f. View the values for the `SparkContext`, `appName`, and `version`.

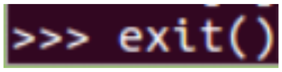
```
>>> sc

>>> sc.appName

>>> sc.version
```



```
>>> sc
<pyspark.context.SparkContext object at 0xd09390>
>>> sc.appName
u'PySparkShell'
>>> sc.version
u'1.6.0'
```

g. Exit the Spark Python REPL.

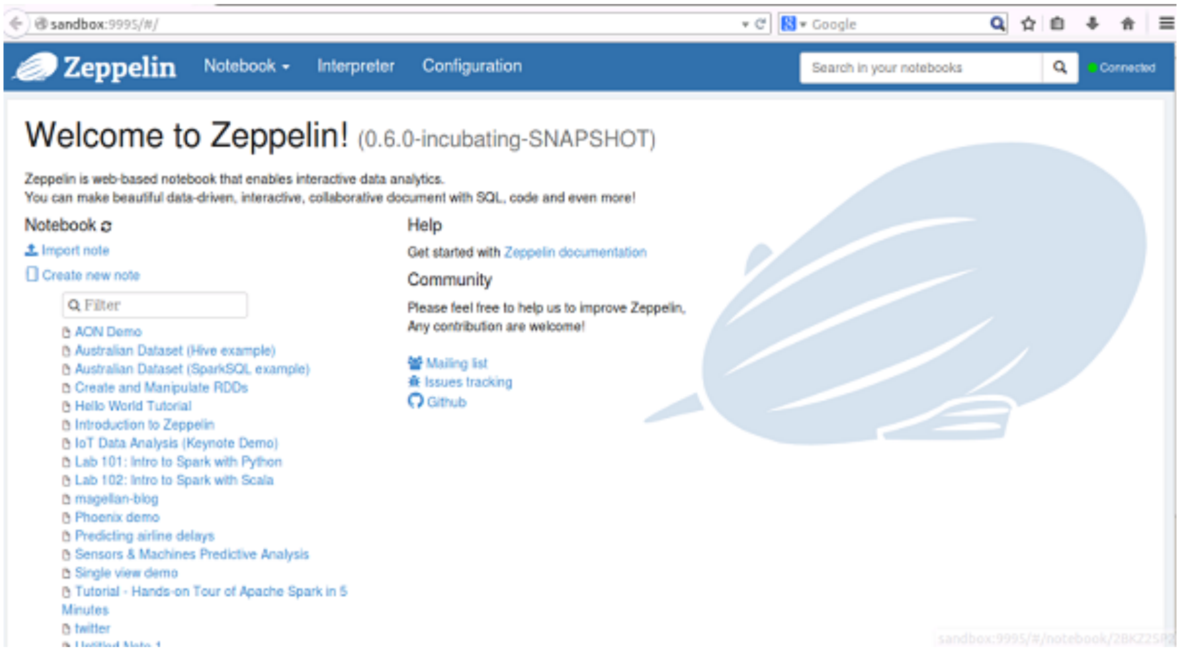
```
>>> exit()
```



```
>>> exit()
```

2. Access and browse Zeppelin.

a. Open the Firefox browser and enter the following URL to view the Zeppelin UI: http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Create and Manipulate RDDs
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1

Help

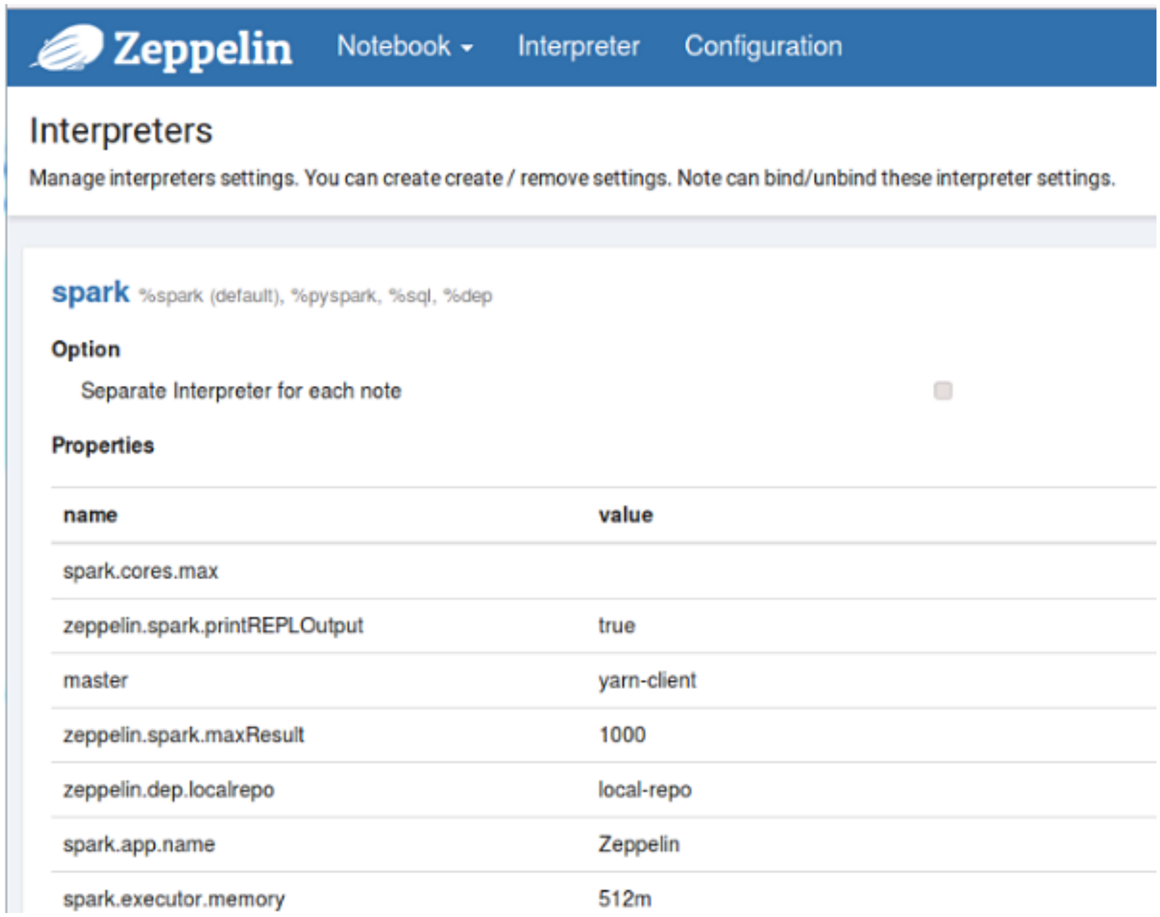
Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

b. Click Interpreter in the top menu and note that Zepplin's default interpreter is set to Spark and has a number of default settings configured.



Interpreters

Manage interpreters settings. You can create create / remove settings. Note can bind/unbind these interpreter settings.

spark %spark (default), %pyspark, %sql, %dep

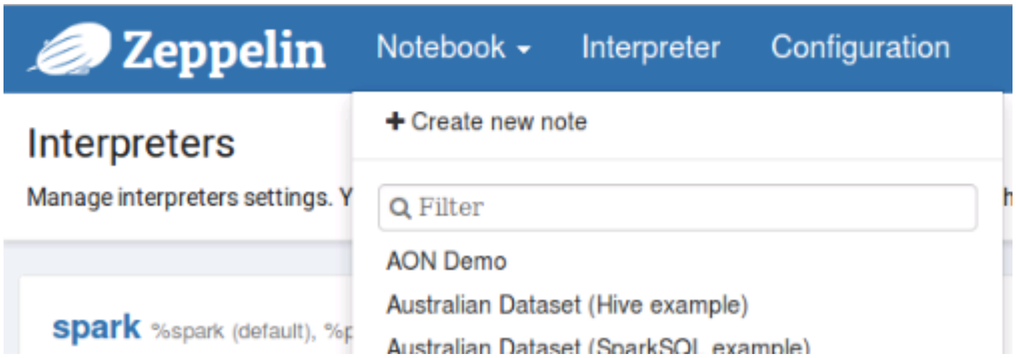
Option

Separate Interpreter for each note □

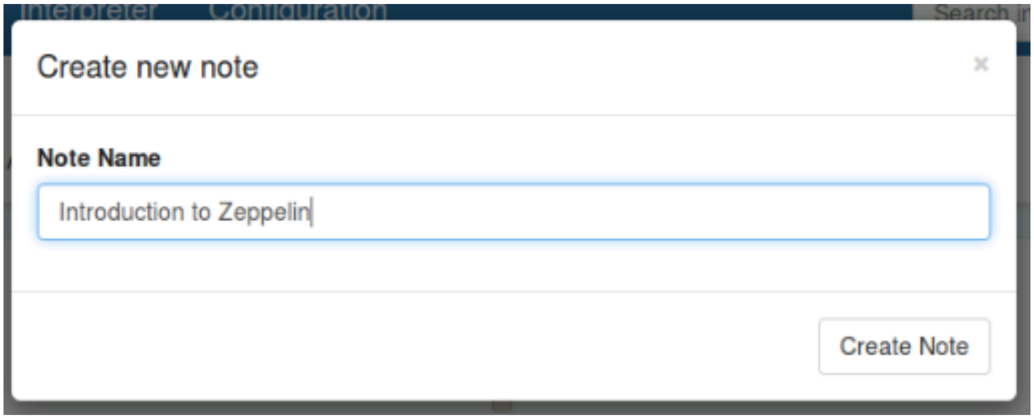
Properties

name	value
spark.cores.max	
zeppelin.spark.printREPLOutput	true
master	yarn-client
zeppelin.spark.maxResult	1000
zeppelin.dep.localrepo	local-repo
spark.app.name	Zeppelin
spark.executor.memory	512m

c. Click on Notebook in the top menu and select Create new note from the resulting drop down options.

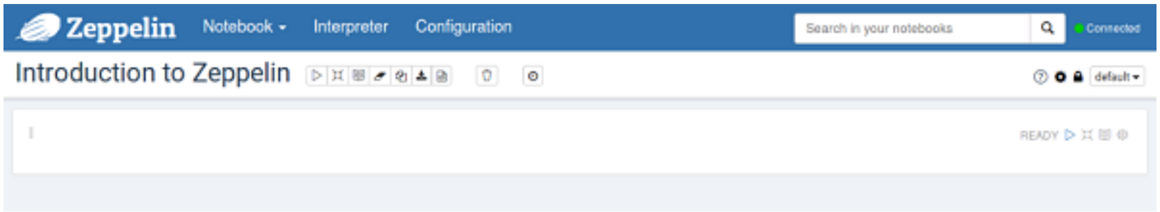

Zeppelin
Notebook ▾
Interpreter
Configuration
Interpreters
Manage interpreters settings. Y
spark %spark (default), %p
+ Create new note
Filter
AON Demo
Australian Dataset (Hive example)
Australian Dataset (SparkSQL example)

d. Name this note Introduction to Zeppelin and click Create Note.



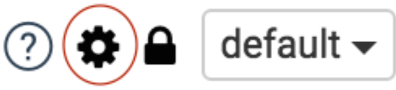
Create new note

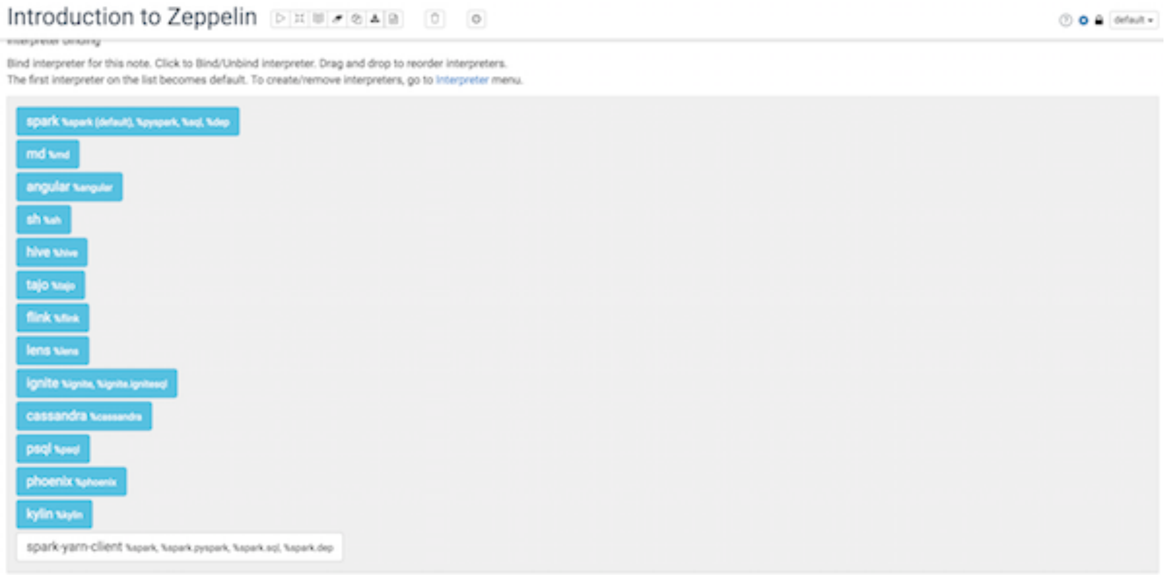
Note Name: Introduction to Zeppelin



Introduction to Zeppelin

e. At the top right click on the gear icon to change interpreter binding. Your administrator has enabled an interpreter called "`spark yarn-client`" which is configured for the HDP cluster you are using. Drag it to the top of the list of interpreters, and click the Save button.



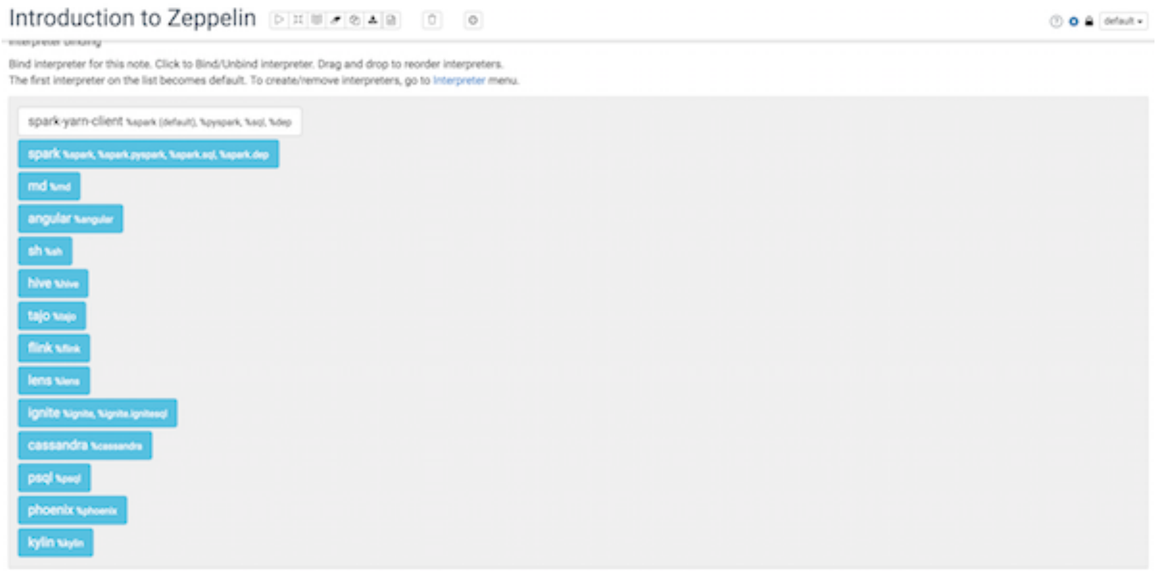


Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

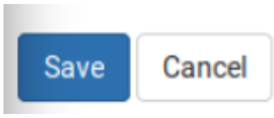


Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

- spark-yarn-client %spark (default), %pyspark, %sql, %dep
- spark %spark, %spark.pyspark, %spark.sql, %spark.dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin

The first interpreter on the list is treated as the default interpreter. Scroll down to find the Save button.

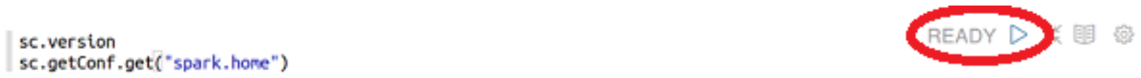


f. Find the values for Spark version and the Spark home directory. When you type the commands, run them either by pressing the Shift + Enter keys, or by clicking on the Play icon to the right of the word Ready.

NOTE:

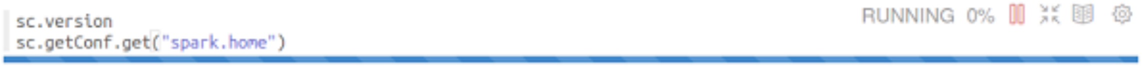
The first time this is run, it may take a few minutes to complete. Future commands will run much faster, including this one if repeated.

```
sc.version
sc.getConf.get("spark.home")
```



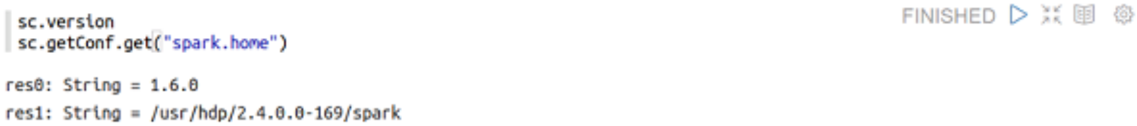
```
sc.version
sc.getConf.get("spark.home")
```

While processing, Zeppelin will display a status of RUNNING. It will also display a Pause icon should it become necessary.



```
sc.version
sc.getConf.get("spark.home")
```

The output may vary slightly from the screenshot below, but should look something like this when processing is completed:



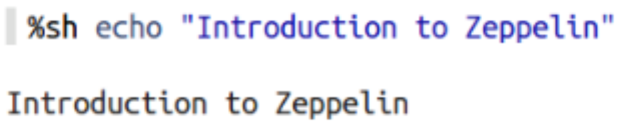
```
sc.version
sc.getConf.get("spark.home")

res0: String = 1.6.0
res1: String = /usr/hdp/2.4.0.0-169/spark
```

g. Zeppelin can be instructed to use multiple languages in an interactive fashion within the same notebook. Simply specify the desired language prior to the command.

Run the following commands to demonstrate this flexibility using Shell, Python, Scala, Markdown, and Spark SQL. Execute each command by clicking on the Play icon or pressing Shift + Enter when you are finished typing.

Shell:

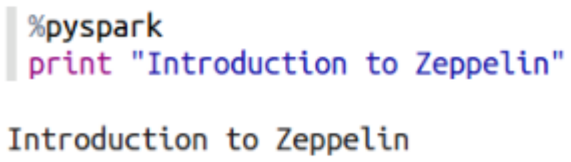
```
%sh echo "Introduction to Zeppelin"
```



```
%sh echo "Introduction to Zeppelin"

Introduction to Zeppelin
```

Python:

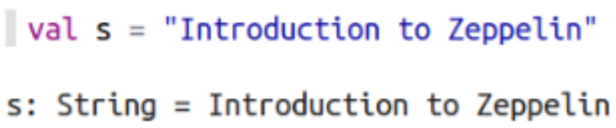
```
%pyspark
print "Introduction to Zeppelin"
```



```
%pyspark
print "Introduction to Zeppelin"

Introduction to Zeppelin
```

Scala (default, so no need to specify prior to running command):

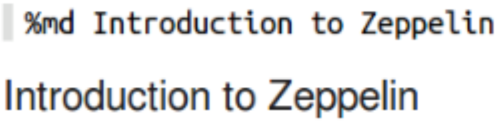
```
val s = "Introduction to Zeppelin"
```



```
val s = "Introduction to Zeppelin"

s: String = Introduction to Zeppelin
```

Markdown:

```
%md Introduction to Zeppelin
```

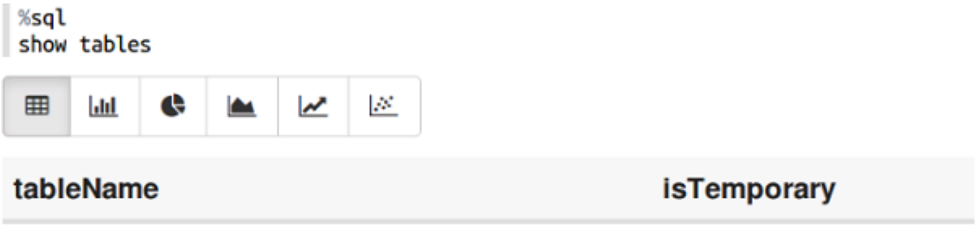


```
%md Introduction to Zeppelin
```

Introduction to Zeppelin

Spark SQL:

```
%sql
show tables
```



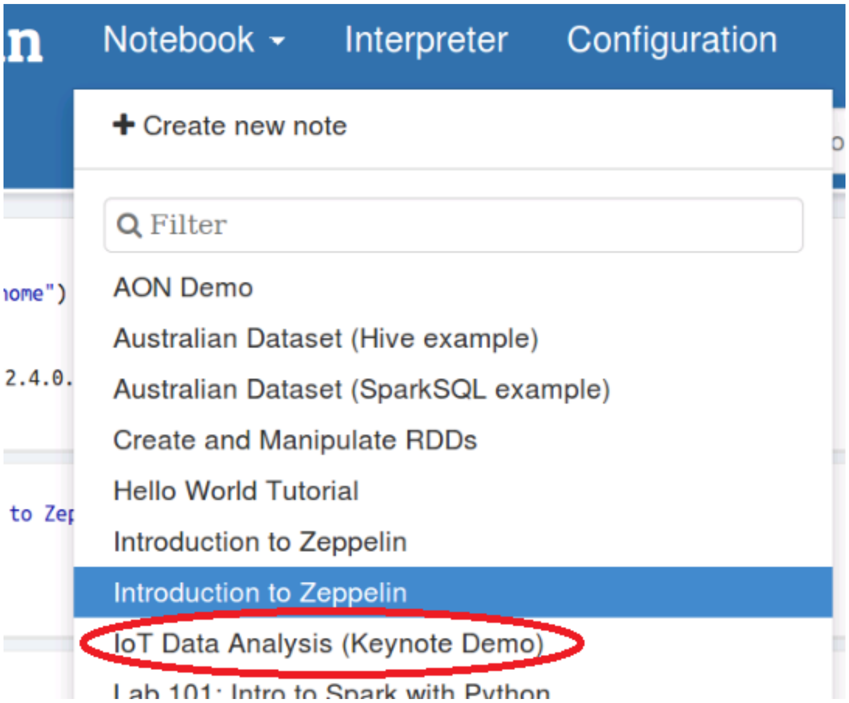
```
%sql
show tables
```

tableName	isTemporary

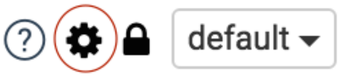
3. **Use a preconfigured notebook to browse Zeppelin's capabilities.**

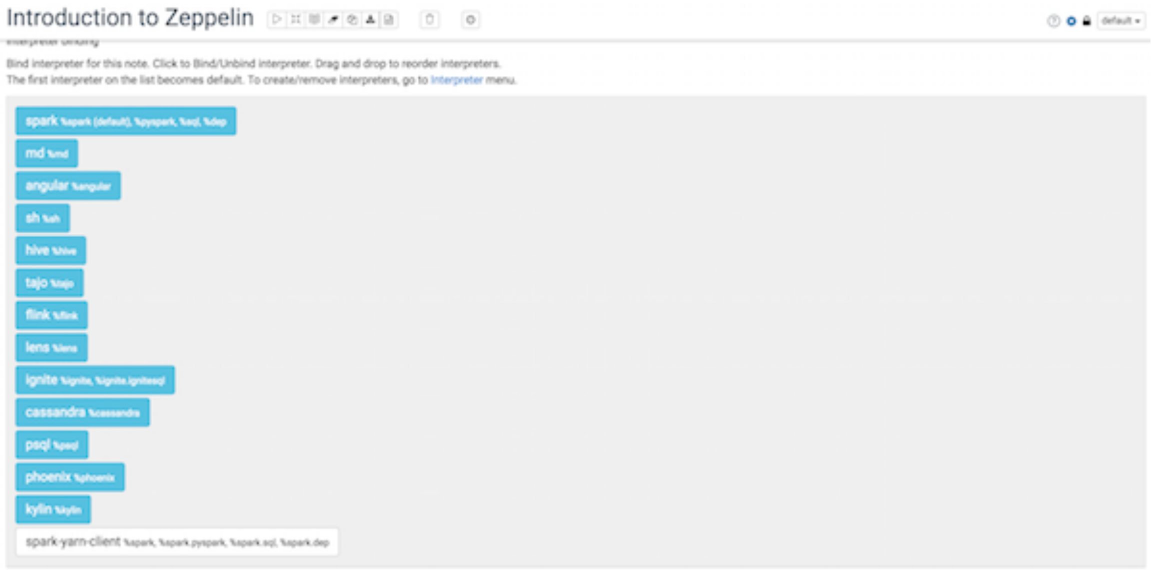
   a. Zeppelin has four major functions: data ingestion, data discovery, data analytics, and data visualization. One of the easiest ways to explore these functions is with a preconfigured notebook, many of which are available by default.

   Click on Notebook at the top of the browser window and find and select the notebook labeled IoT Data Analysis (Keynote Demo) in the resulting drop-down menu.


Notebook
Interpreter
Configuration
Create new note
Filter
AON Demo
Australian Dataset (Hive example)
Australian Dataset (SparkSQL example)
Create and Manipulate RDDs
Hello World Tutorial
Introduction to Zeppelin
Introduction to Zeppelin
IoT Data Analysis (Keynote Demo)
Lab 101: Intro to Spark with Python

b. At the top right click on the gear icon to change interpreter binding.





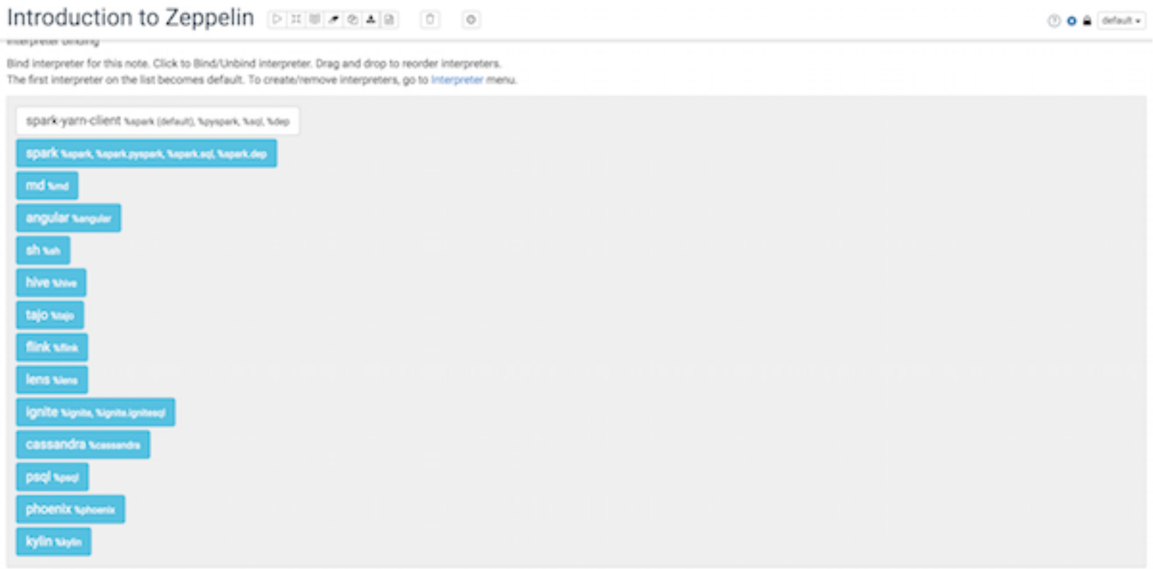
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

- spark %spark (default), %pyspark, %sql, %dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin
- spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

Drag the spark-yarn-client to the top and click save.



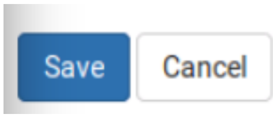
Introduction to Zeppelin

Interpreter binding

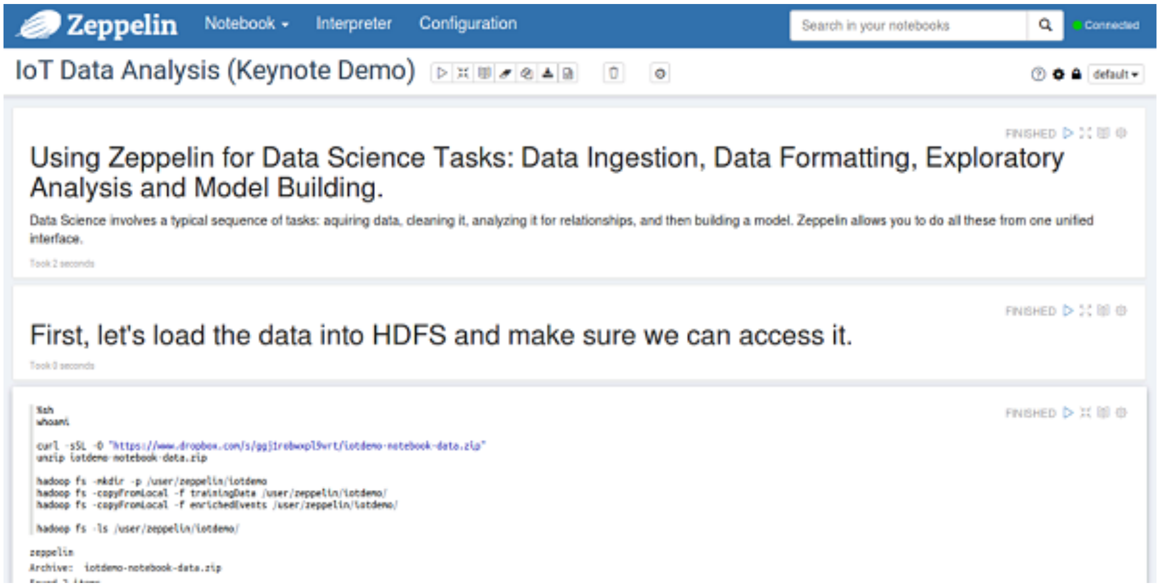
Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

- spark-yarn-client %spark (default), %pyspark, %sql, %dep
- spark %spark, %spark.pyspark, %spark.sql, %spark.dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin

The first interpreter on the list becomes default.



c. For the purposes of this lab, all necessary code has already been entered for you in the saved notebook. All you have to do is scroll to the appropriate section and click the Play icon or press Shift + Enter.



IoT Data Analysis (Keynote Demo)

Using Zeppelin for Data Science Tasks: Data Ingestion, Data Formatting, Exploratory Analysis and Model Building.

Data Science involves a typical sequence of tasks: aquiring data, cleaning it, analyzing it for relationships, and then building a model. Zeppelin allows you to do all these from one unified interface.

Took 2 seconds

First, let's load the data into HDFS and make sure we can access it.

Took 0 seconds

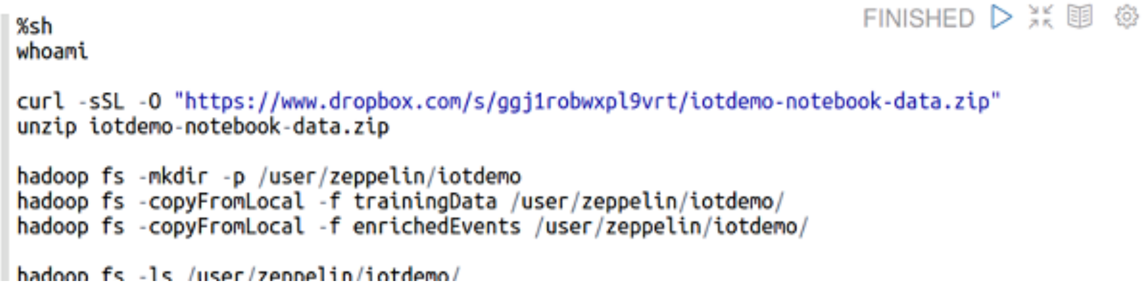
```
%sh
whoami

curl -sSL -O "https://www.dropbox.com/s/ggj1robwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip

hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/

hadoop fs -ls /user/zeppelin/iotdemo/
```

```
zeppelin
Archive:  iotdemo-notebook-data.zip
```

d. The first major block of code ingests data from an online source into HDFS and then displays those files using the shell scripting interpreter. Find and run that code.

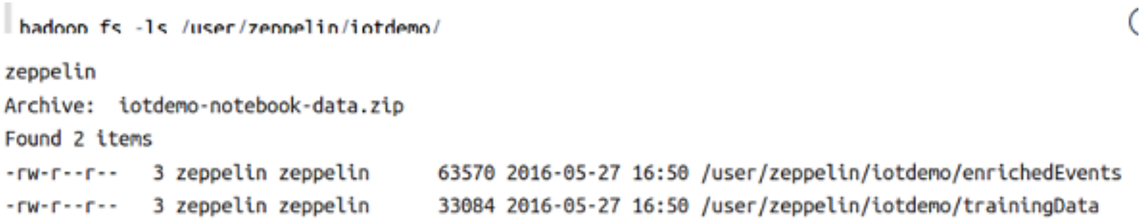
NOTE:
that the label to the left of the Play icon says FINISHED, but this will not prohibit you from running the code again on this machine.

This notebook uses a deprecated command, `hadoop fs`, rather than the more updated `hdfs dfs` command we used in the previous lab. This should not affect the functionality of the demo.



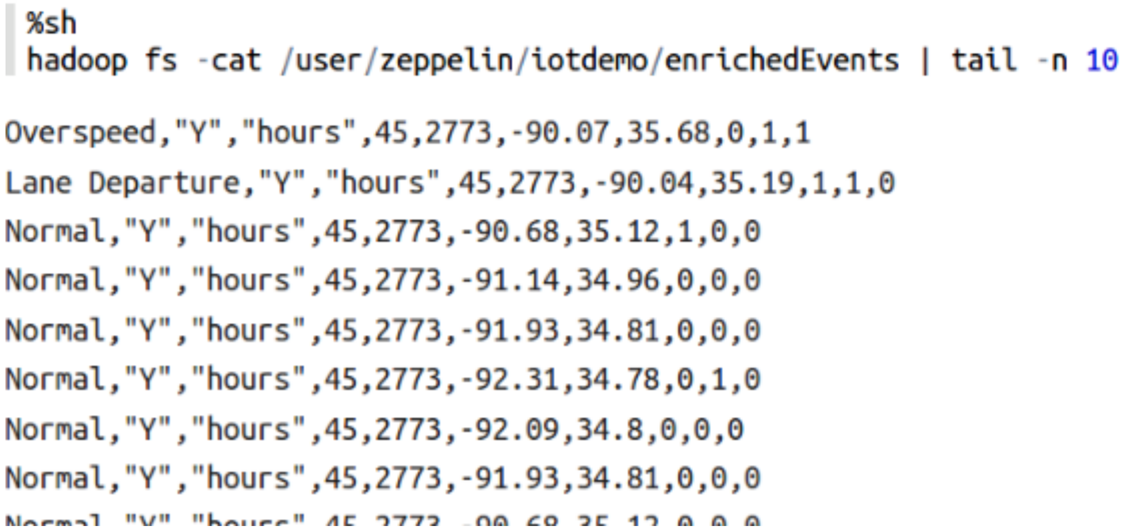
```
%sh
whoami

curl -sSL -O "https://www.dropbox.com/s/ggj1robwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip

hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/

hadoop fs -ls /user/zeppelin/iotdemo/
```

When the code has finished, the output at the bottom should look like this:



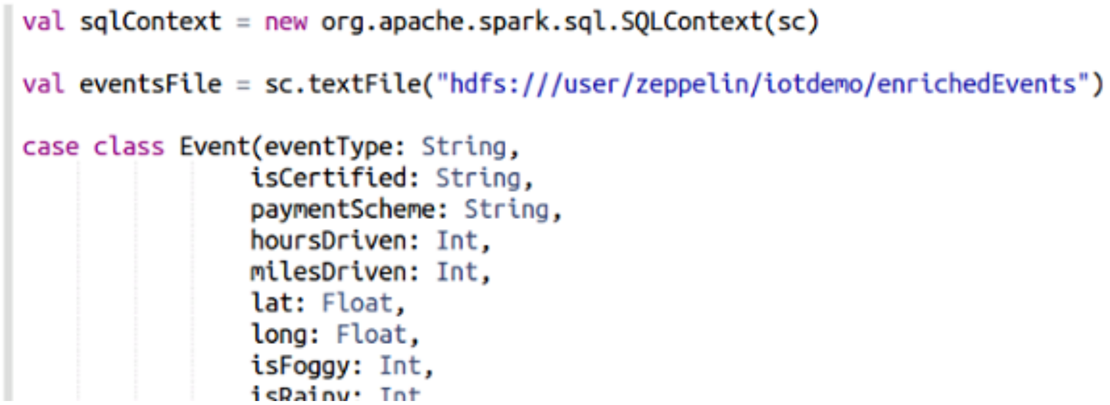
```
hadoop fs -ls /user/zeppelin/iotdemo/

zeppelin
Archive:  iotdemo-notebook-data.zip
Found 2 items
-rw-r--r--   3 zeppelin zeppelin      63570 2016-05-27 16:50 /user/zeppelin/iotdemo/enrichedEvents
-rw-r--r--   3 zeppelin zeppelin      33084 2016-05-27 16:50 /user/zeppelin/iotdemo/trainingData
```

e. The next section of the notebook once again uses the shell scripting interpreter to view some of the raw data in one of the downloaded files. Scroll down and run this code, then view its output.



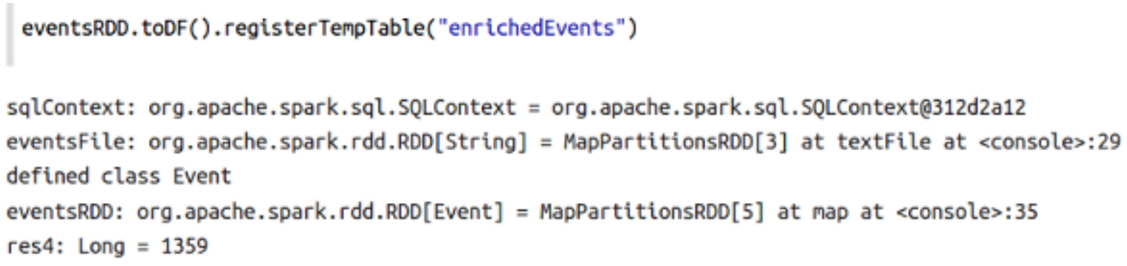
```
%sh
hadoop fs -cat /user/zeppelin/iotdemo/enrichedEvents | tail -n 10

Overspeed,"Y","hours",45,2773,-90.07,35.68,0,1,1
Lane Departure,"Y","hours",45,2773,-90.04,35.19,1,1,0
Normal,"Y","hours",45,2773,-90.68,35.12,1,0,0
Normal,"Y","hours",45,2773,-91.14,34.96,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
Normal,"Y","hours",45,2773,-92.31,34.78,0,1,0
Normal,"Y","hours",45,2773,-92.09,34.8,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
Normal,"Y","hours",45,2773,-90.68,35.12,0,0,0
```

f. The next section of the notebook performs actions necessary to import and use this data with Spark SQL. You may note that the status to the left of the Play icon is shown as ERROR. This is due to the fact that the file being manipulated did not exist at the time the notebook was opened on this system. Run this code and view the output.



```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val eventsFile = sc.textFile("hdfs:///user/zeppelin/iotdemo/enrichedEvents")

case class Event(eventType: String,
                 isCertified: String,
                 paymentScheme: String,
                 hoursDriven: Int,
                 milesDriven: Int,
                 lat: Float,
                 long: Float,
                 isFoggy: Int,
                 isRainy: Int
```

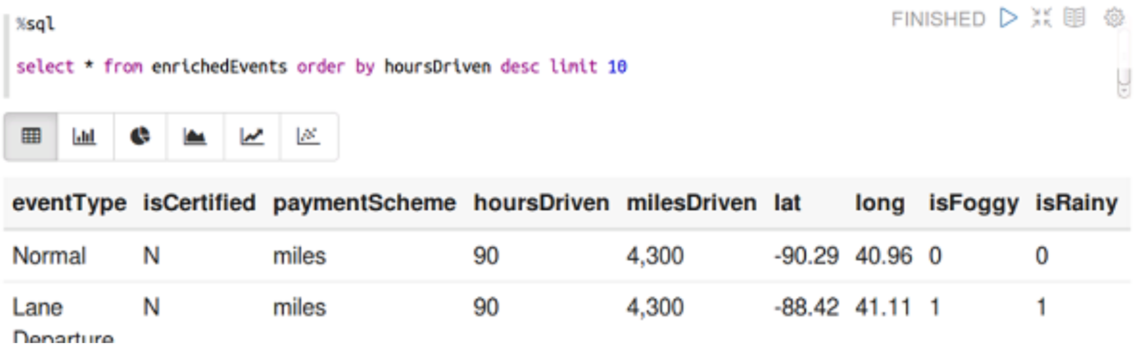
The output should look like this:



```
eventsRDD.toDF().registerTempTable("enrichedEvents")
```

```
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@312d2a12
eventsFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at <console>:29
defined class Event
eventsRDD: org.apache.spark.rdd.RDD[Event] = MapPartitionsRDD[5] at map at <console>:35
res4: Long = 1359
```

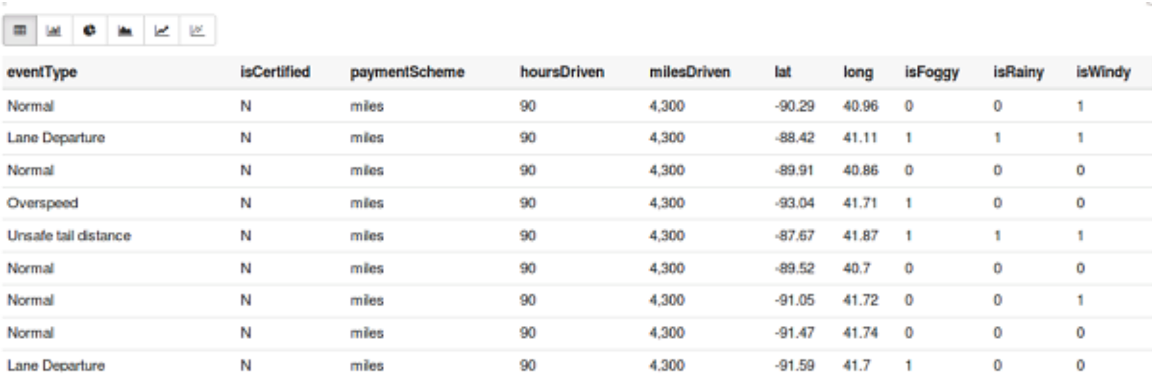
g. The next block of code utilizes Spark SQL to view this data. Run this code and examine the output.



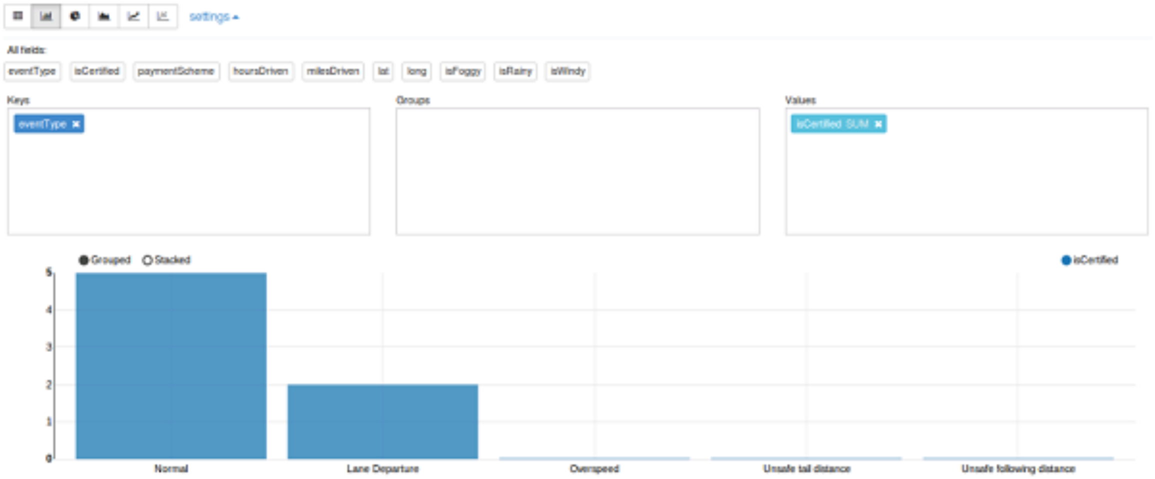
```
%sql
select * from enrichedEvents order by hoursDriven desc limit 10
```

eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy
Normal	N	miles	90	4,300	-90.29	40.96	0	0
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1

h. Note that at the top of the results there are six buttons that allow you to display the results using six different visualizations. Click on each one to view the differences between them.



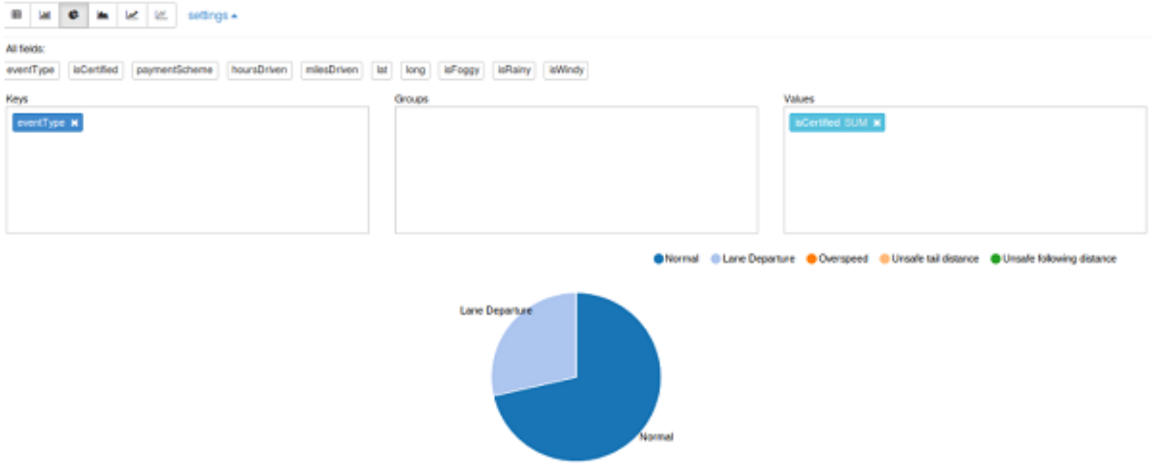
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy	isWindy
Normal	N	miles	90	4,300	-90.29	40.96	0	0	1
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1	1
Normal	N	miles	90	4,300	-89.91	40.86	0	0	0
Overspeed	N	miles	90	4,300	-93.04	41.71	1	0	0
Unsafe tail distance	N	miles	90	4,300	-87.67	41.87	1	1	1
Normal	N	miles	90	4,300	-89.52	40.7	0	0	0
Normal	N	miles	90	4,300	-91.05	41.72	0	0	1
Normal	N	miles	90	4,300	-91.47	41.74	0	0	0
Lane Departure	N	miles	90	4,300	-91.59	41.7	1	0	0



Keys: eventType

Groups:

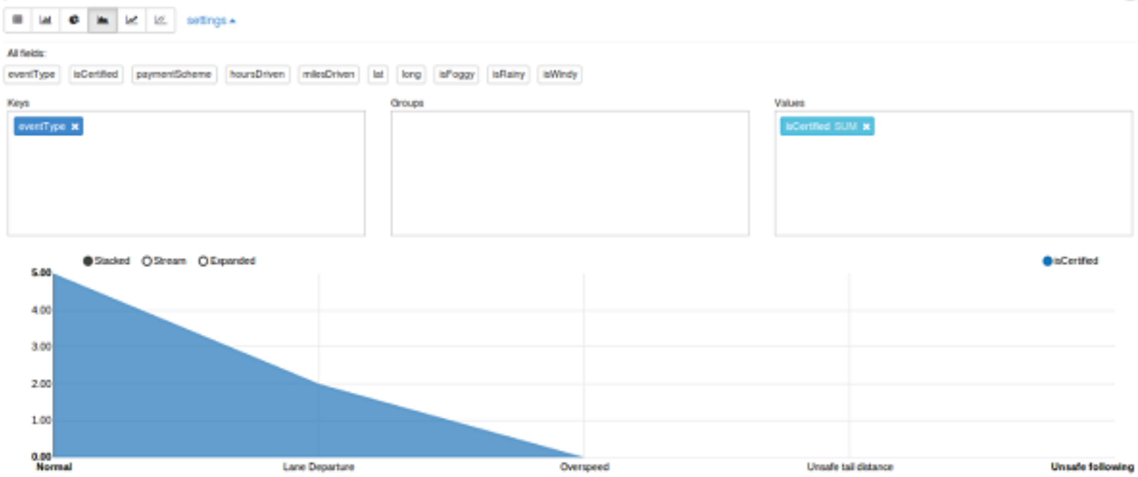
Values: isCertified SUM



Keys: eventType

Groups:

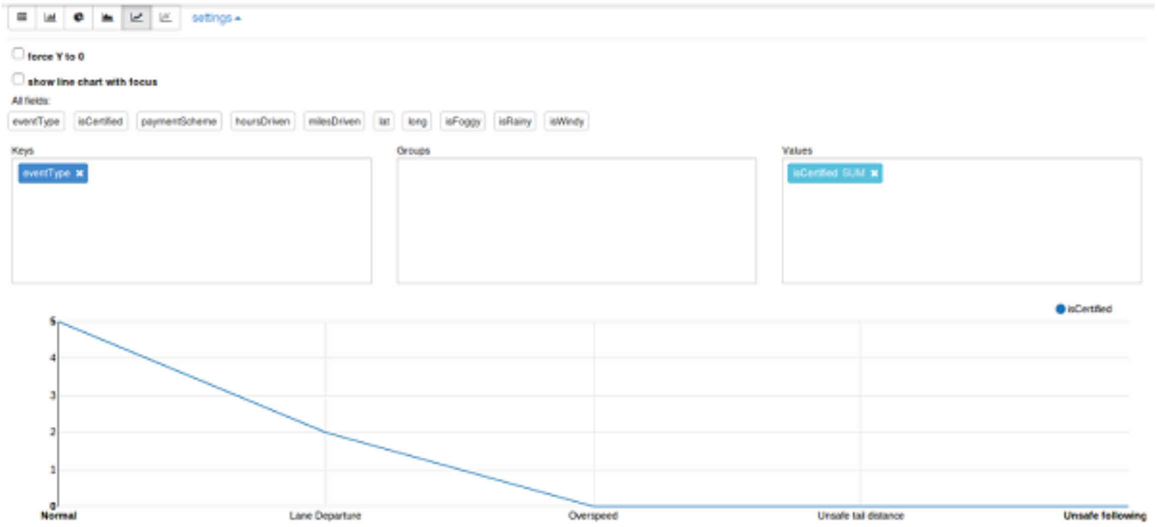
Values: isCertified SUM

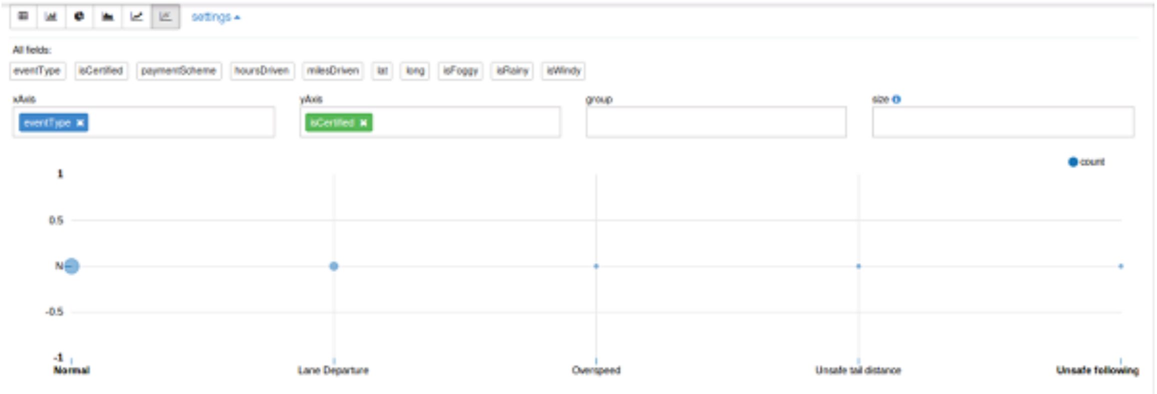


Keys: eventType

Groups:

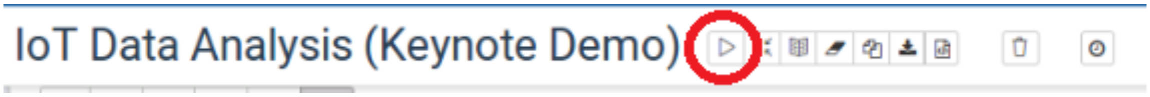
Values: isCertified SUM





TIP:

In this lab you ran each section of code, known as a paragraph, individually. The entire notebook could have been played at once, however, by clicking the Play icon labeled Run all paragraphs directly to the right of the notebook title at the top of the browser.



Result

You have accessed the Spark REPLs for both Scala and Python, created a Zeppelin notebook and demonstrated Zeppelin's ability to interpret multiple languages, and used a pre-built Zeppelin notebook to briefly explore Zeppelin's ability to ingest, view, analyze, and visualize data.

Lab 3: Create and Manipulate RDDs (Python)

About This Lab

Objective:
Create and Manipulate RDDs using Python and Zeppelin

File Locations:
`/home/zeppelin/spark/data/`

Successful Outcome:
Perform basic RDD transformations and actions using Zeppelin.

Before You Begin:
Complete the Pre-Lab

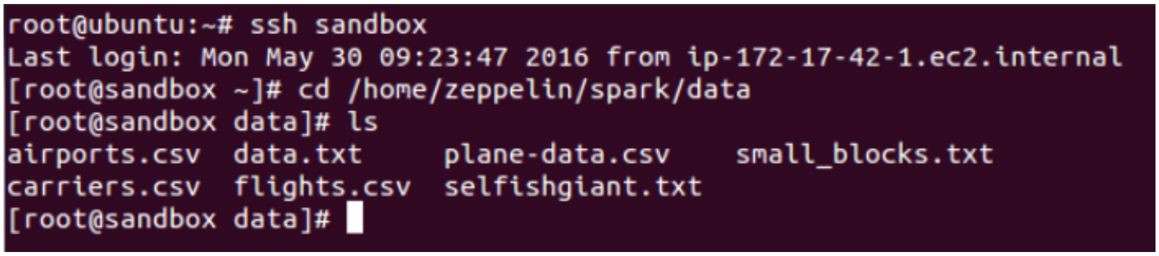
Lab Steps

Perform the following steps:

1. **View the raw data for this lab.**

   a. In a new terminal window, ssh to sandbox and change directories to `/home/zeppelin/spark/data`. View the files in this directory.

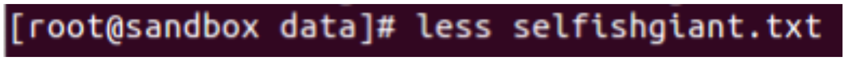
```
# ssh sandbox

# cd /home/zeppelin/spark/data/

# ls
```



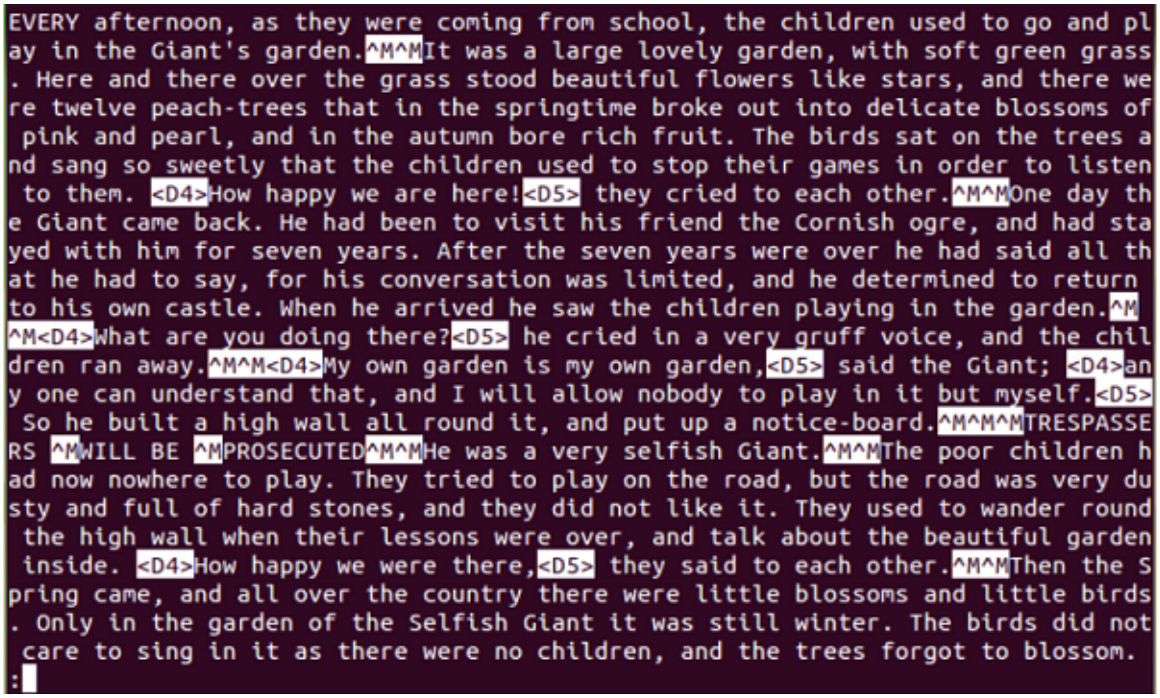
```
root@ubuntu:~# ssh sandbox
Last login: Mon May 30 09:23:47 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]# cd /home/zeppelin/spark/data
[root@sandbox data]# ls
airports.csv  data.txt     plane-data.csv    small_blocks.txt
carriers.csv  flights.csv  selfishgiant.txt
[root@sandbox data]# 
```

   b. Use `less` to view the “selfishgiant.txt” data file. Press `q` to quit when you are finished reviewing.

```
# less selfishgiant.txt
```



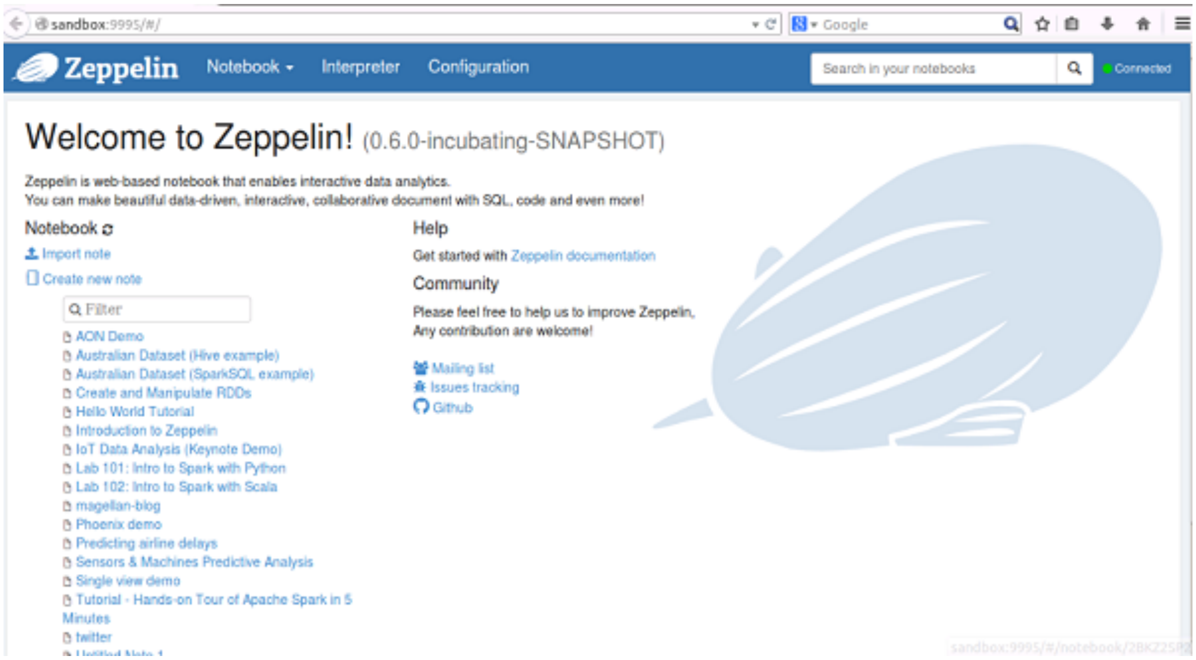
```
[root@sandbox data]# less selfishgiant.txt
```



```
EVERY afternoon, as they were coming from school, the children used to go and pl
ay in the Giant's garden.^M^MIt was a large lovely garden, with soft green grass
. Here and there over the grass stood beautiful flowers like stars, and there we
re twelve peach-trees that in the springtime broke out into delicate blossoms of
 pink and pearl, and in the autumn bore rich fruit. The birds sat on the trees a
nd sang so sweetly that the children used to stop their games in order to listen
 to them. <D4>How happy we are here!<D5> they cried to each other.^M^MOne day th
e Giant came back. He had been to visit his friend the Cornish ogre, and had sta
yed with him for seven years. After the seven years were over he had said all th
at he had to say, for his conversation was limited, and he determined to return
to his own castle. When he arrived he saw the children playing in the garden.^M
^M<D4>What are you doing there?<D5> he cried in a very gruff voice, and the chil
dren ran away.^M^M<D4>My own garden is my own garden,<D5> said the Giant; <D4>an
y one can understand that, and I will allow nobody to play in it but myself.<D5>
 So he built a high wall all round it, and put up a notice-board.^M^M^MTRESPASSE
RS ^MWILL BE ^MPROSECUTED^M^MHe was a very selfish Giant.^M^MThe poor children h
ad now nowhere to play. They tried to play on the road, but the road was very du
sty and full of hard stones, and they did not like it. They used to wander round
 the high wall when their lessons were over, and talk about the beautiful garden
 inside. <D4>How happy we were there,<D5> they said to each other.^M^MThen the S
pring came, and all over the country there were little blossoms and little birds
. Only in the garden of the Selfish Giant it was still winter. The birds did not
 care to sing in it as there were no children, and the trees forgot to blossom.
:
```

2. **Perform basic RDD manipulations using the Zeppelin notebook.**

   a. Open the Firefox browser and enter the following URL to view the Zeppelin UI: `http://sandbox:9995/`



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

- Import note
- Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Create and Manipulate RDDs
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1

Help

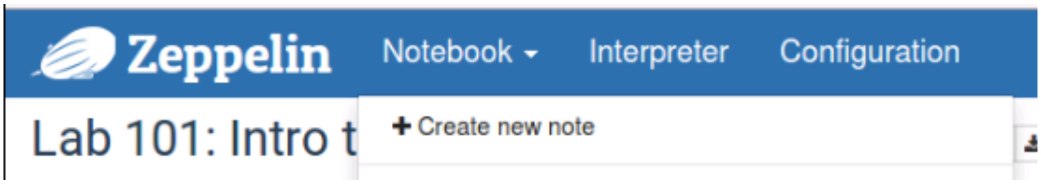
Get started with Zeppelin documentation

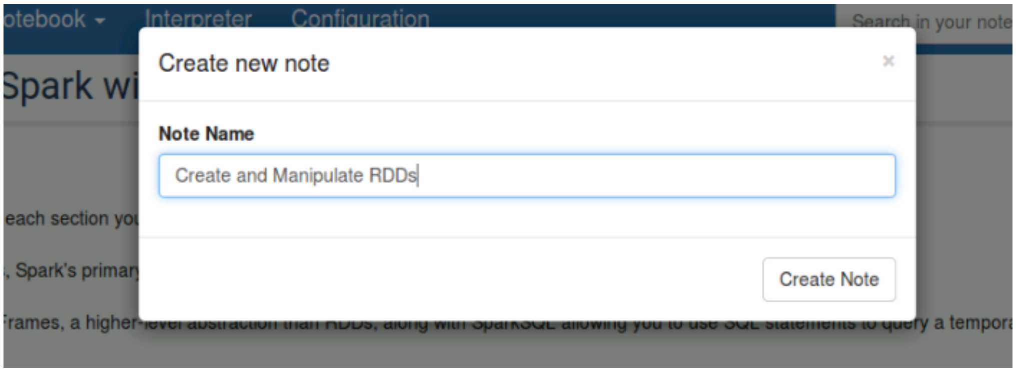
Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

- Mailing list
- Issues tracking
- Github

b. Click on Notebook and select Create new note on the drop down. Name this note Create and Manipulate RDDs.

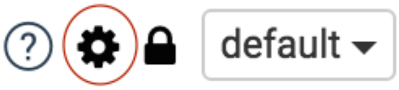


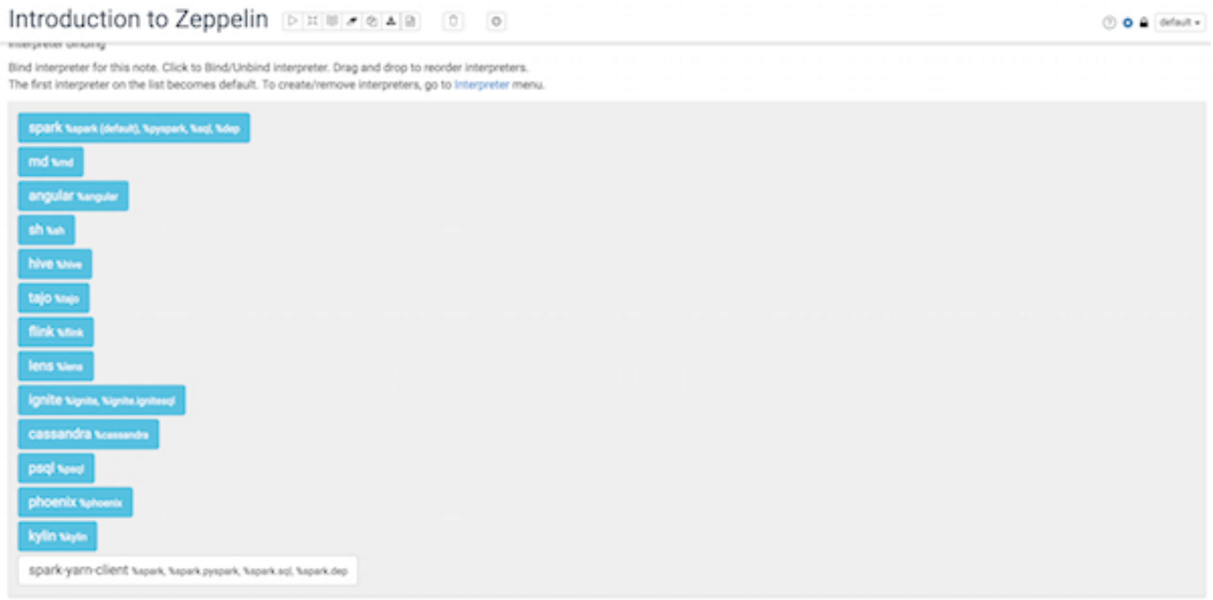


Create and Manipulate RDDs

each section yo
, Spark's primar
rames, a higher-level abstraction than RDDs, along with SparkSQL allowing you to use SQL statements to query a tempora

c. At the top right click on the gear icon to change interpreter binding.

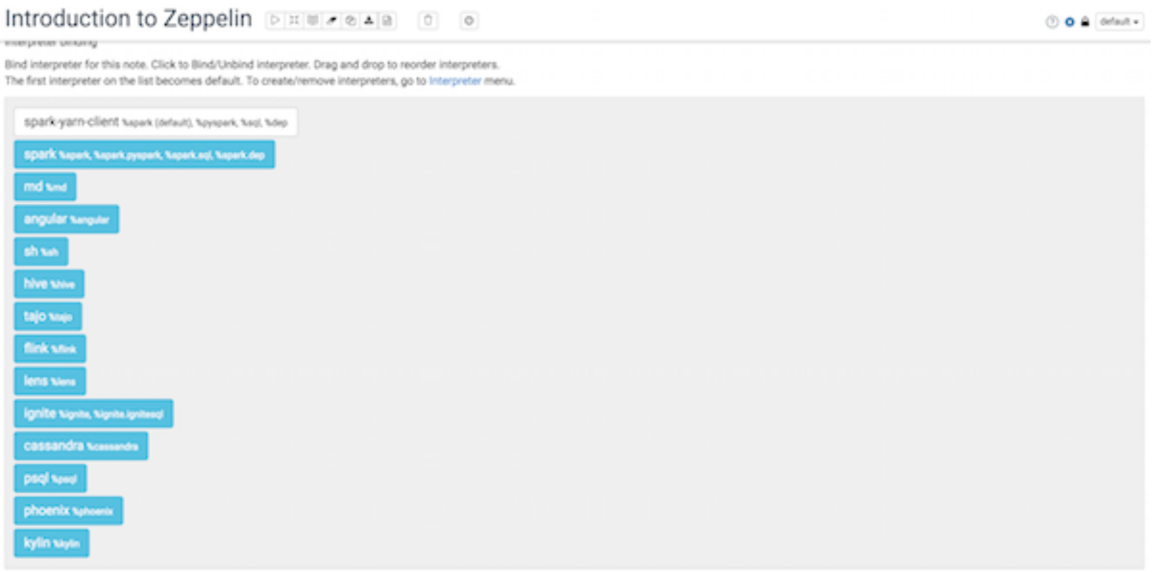




Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

Drag the spark-yarn-client to the top and click save.



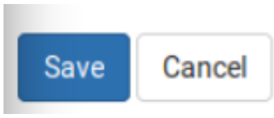
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

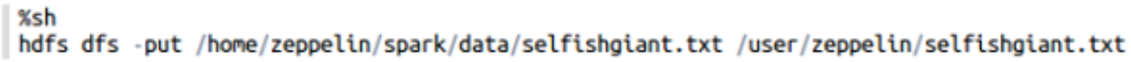
spark-yarn-client %spark (default), %pyspark, %sql, %dep
spark %spark, %spark.pyspark, %spark.sql, %spark.dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin

The first interpreter on the list becomes default.



d. Place the selfishgiant.txt file into the Zeppelin user's home directory on HDFS, `/user/zeppelin`. (There are no line breaks in the code below after `%sh`. Please refer to the screenshot.)

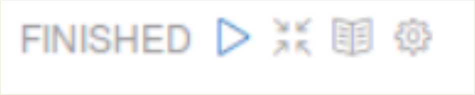
```
%sh
hdfs dfs -put /home/zeppelin/spark/data/selfishgiant.txt
/user/zeppelin/selfishgiant.txt
```



```
%sh
hdfs dfs -put /home/zeppelin/spark/data/selfishgiant.txt /user/zeppelin/selfishgiant.txt
```

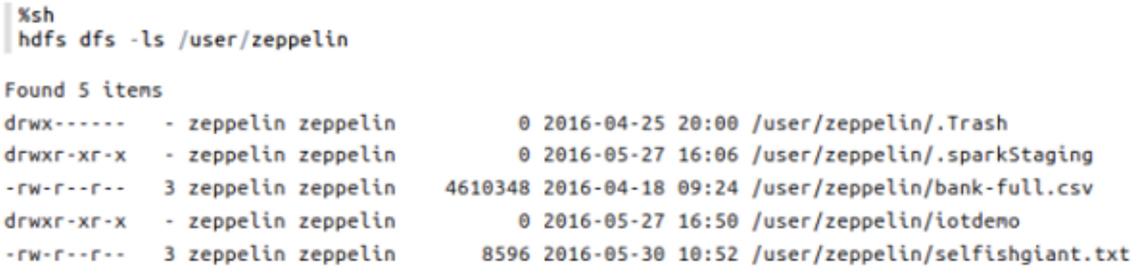
REMINDER:

After entering a command, press **Shift + Enter** keys or press the **Play** button on the right side of the paragraph to execute the commands. The text to the left of the **Play** button should change from READY to FINISHED when it is complete.



e. Verify the file was uploaded successfully.

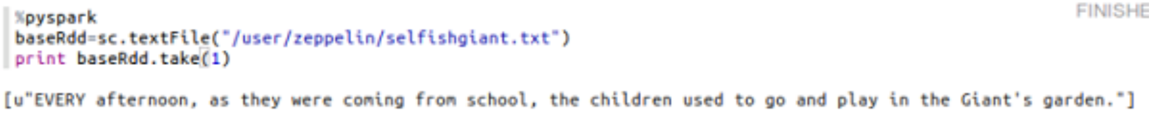
```
%sh
hdfs dfs -ls /user/zeppelin
```



```
%sh
hdfs dfs -ls /user/zeppelin

Found 5 items
drwx------   - zeppelin zeppelin          0 2016-04-25 20:00 /user/zeppelin/.Trash
drwxr-xr-x   - zeppelin zeppelin          0 2016-05-27 16:06 /user/zeppelin/.sparkStaging
-rw-r--r--   3 zeppelin zeppelin    4610348 2016-04-18 09:24 /user/zeppelin/bank-full.csv
drwxr-xr-x   - zeppelin zeppelin          0 2016-05-27 16:50 /user/zeppelin/iotdemo
-rw-r--r--   3 zeppelin zeppelin       8596 2016-05-30 10:52 /user/zeppelin/selfishgiant.txt
```

f. Create an RDD named `baseRdd` using this file. Verify the RDD exists by using the `take()` function to print the first line of the file.

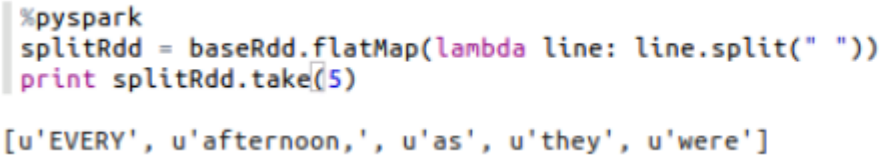
```
%pyspark
baseRdd=sc.textFile(“/user/zeppelin/selfishgiant.txt”)
print baseRdd.take(1)
```



```
%pyspark
baseRdd=sc.textFile("/user/zeppelin/selfishgiant.txt")
print baseRdd.take(1)

[u"EVERY afternoon, as they were coming from school, the children used to go and play in the Giant's garden."]
```

g. Each line of the file is currently a string. Transform the lines into arrays of individual elements (words) stored in a new RDD named `splitRdd`, then take a look at the first five elements.

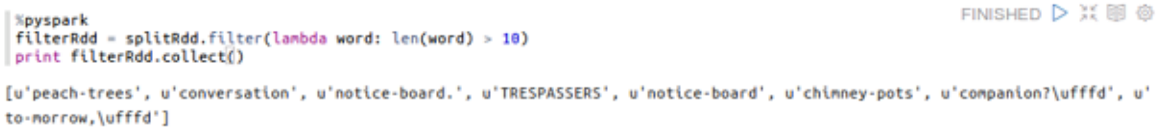
```
%pyspark
splitRdd = baseRdd.flatMap(lambda line: line.split(" "))
print splitRdd.take(5)
```



```
%pyspark
splitRdd = baseRdd.flatMap(lambda line: line.split(" "))
print splitRdd.take(5)

[u'EVERY', u'afternoon,', u'as', u'they', u'were']
```

h. Create a new RDD named `filterRdd` that only contains words in `splitRdd` that are longer than 10 characters. Use `collect()` to view the entire output.

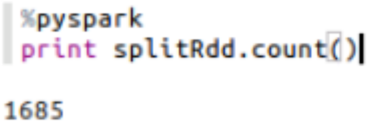
```
%pyspark
filterRdd = splitRdd.filter(lambda word: len(word) > 10)
print filterRdd.collect()
```



```
%pyspark
filterRdd = splitRdd.filter(lambda word: len(word) > 10)
print filterRdd.collect()

[u'peach-trees', u'conversation', u'notice-board.', u'TRESPASSERS', u'notice-board', u'chimney-pots', u'companion?\ufffd', u'
to-morrow,\ufffd']
```

i. Display a count of the total number of words in `splitRdd`.

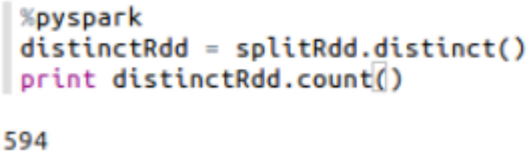
```
%pyspark
print splitRdd.count()
```



```
%pyspark
print splitRdd.count()

1685
```

j. Create an RDD named `distinctRdd` that eliminates any duplicate words in `splitRdd`. Then display a count of the number of distinct words in the RDD.

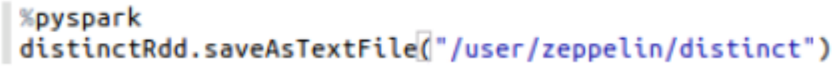
```
%pyspark
distinctRdd = splitRdd.distinct()
print distinctRdd.count()
```



```
%pyspark
distinctRdd = splitRdd.distinct()
print distinctRdd.count()

594
```

k. Save the contents of `distinctRDD` to text in HDFS. Put the contents in a folder named "`distinct`" for future reference.

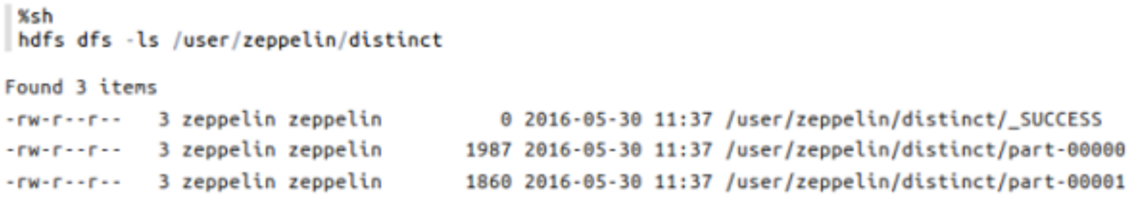
```
%pyspark
distinctRdd.saveAsTextFile("/user/zeppelin/distinct")
```



```
%pyspark
distinctRdd.saveAsTextFile("/user/zeppelin/distinct")
```

l. Verify the contents of the RDD were written to HDFS.

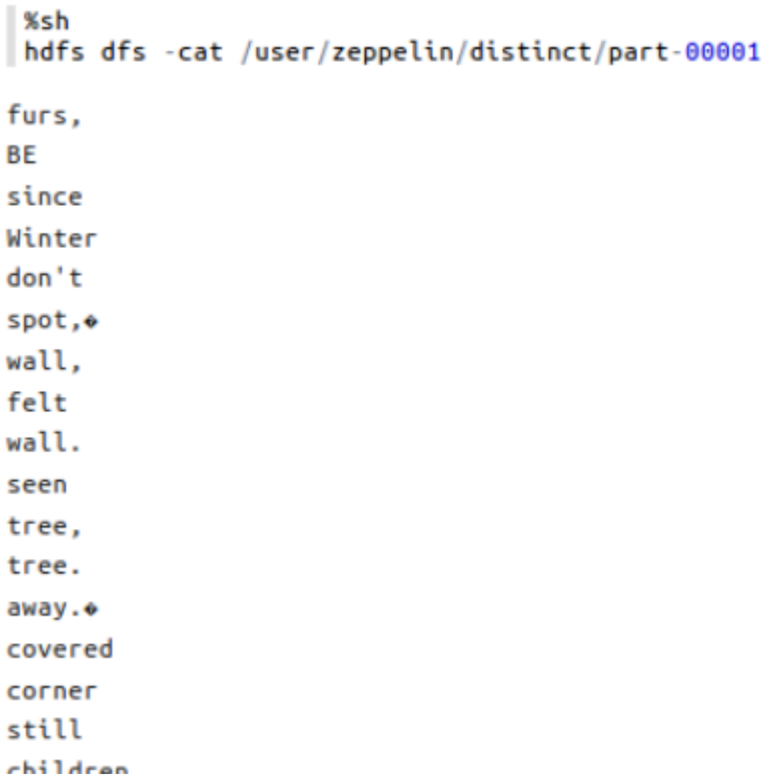
```
%sh
hdfs dfs -ls /user/zeppelin/distinct
```



```
%sh
hdfs dfs -ls /user/zeppelin/distinct

Found 3 items
-rw-r--r--   3 zeppelin zeppelin          0 2016-05-30 11:37 /user/zeppelin/distinct/_SUCCESS
-rw-r--r--   3 zeppelin zeppelin       1987 2016-05-30 11:37 /user/zeppelin/distinct/part-00000
-rw-r--r--   3 zeppelin zeppelin       1860 2016-05-30 11:37 /user/zeppelin/distinct/part-00001
```

m. View the contents of one of the `part-*` files and verify that an array of unique words has been generated and saved.

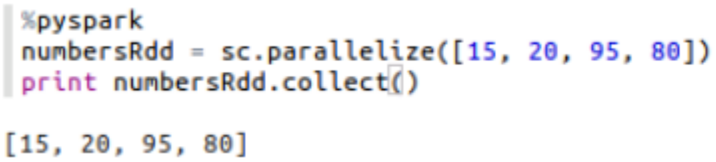
```
%sh
hdfs dfs -cat /user/zeppelin/distinct/part-00001
```



```
%sh
hdfs dfs -cat /user/zeppelin/distinct/part-00001

furs,
BE
since
Winter
don't
spot,�
wall,
felt
wall.
seen
tree,
tree.
away.�
covered
corner
still
children
```

n. Create an RDD named `numbersRdd` that contains an array of the following numbers: 15, 20, 95, and 80. View the contents of the RDD to verify it was successfully created.

```
%pyspark
numbersRdd = sc.parallelize([15, 20, 95, 80])
print numbersRdd.collect()
```

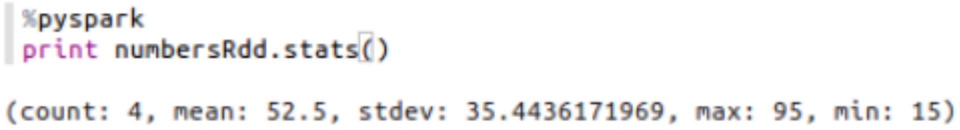


```
%pyspark
numbersRdd = sc.parallelize([15, 20, 95, 80])
print numbersRdd.collect()
```

```
[15, 20, 95, 80]
```

o. Display a count of the elements in `numbersRdd`, as well as the mean, standard deviation, maximimum, and minimum values.

```
%pyspark
print numbersRdd.stats()
```

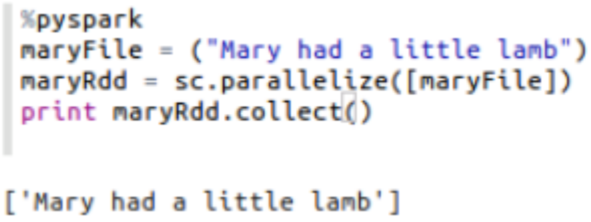


```
%pyspark
print numbersRdd.stats()
```

```
(count: 4, mean: 52.5, stdev: 35.4436171969, max: 95, min: 15)
```

p. Create a variable named `maryFile` that contains the string “Mary had a little lamb” and then convert that variable into an RDD named `maryRdd`. View the RDD contents when finished.

```
%pyspark
maryFile = (“Mary had a little lamb”)
maryRdd = sc.parallelize([maryFile])
print maryRdd.collect()
```

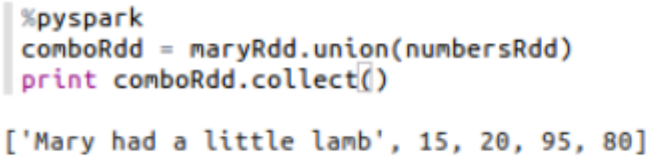


```
%pyspark
maryFile = ("Mary had a little lamb")
maryRdd = sc.parallelize([maryFile])
print maryRdd.collect()
```

```
['Mary had a little lamb']
```

q. **Create a new RDD named `comboRdd` that creates a union between `maryRdd` and `numbersRdd`. Then view the combined RDD.**

```
%pyspark
comboRdd = maryRdd.union(numbersRdd)
print comboRdd.collect()
```



```
%pyspark
comboRdd = maryRdd.union(numbersRdd)
print comboRdd.collect()

['Mary had a little lamb', 15, 20, 95, 80]
```

Result

You have created several RDDs and performed various transactions and actions using the Zeppelin notebook.

Lab 4: Create and Manipulate Pair RDDs (Python)

About This Lab

Objective:
Create pair RDD's and use various functions to transform these RDD's using Python in Zeppelin.

File Locations:
`/home/zeppelin/spark/data/`

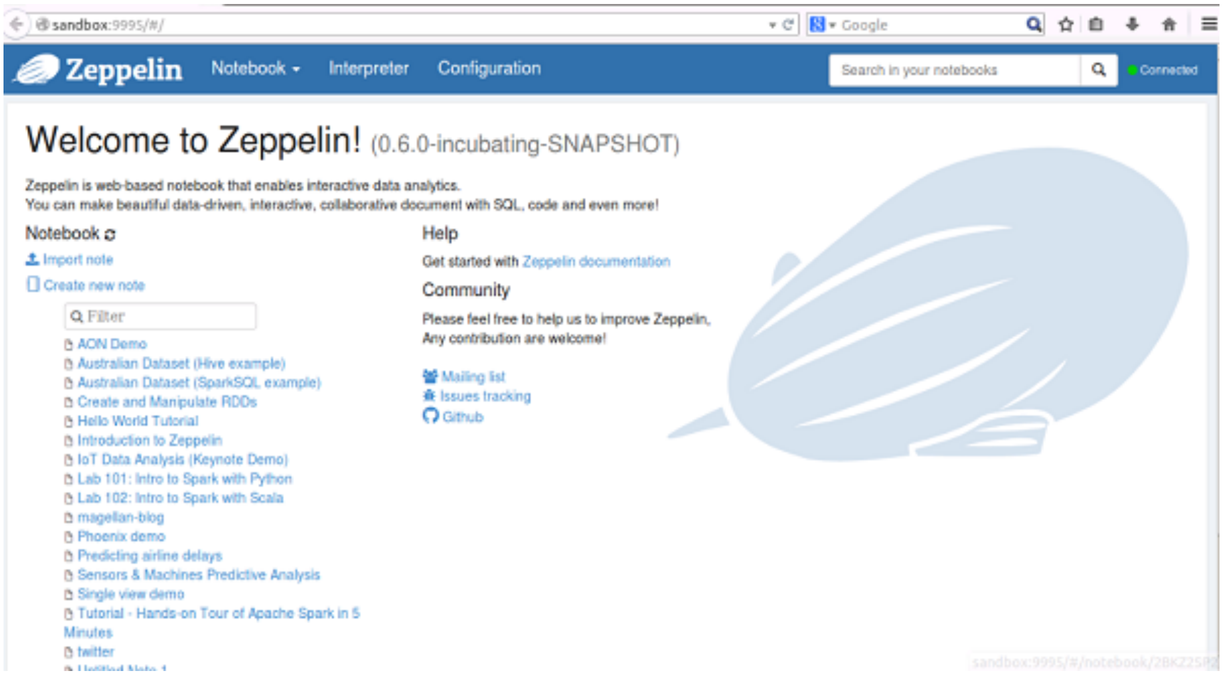
Successful Outcome:
REQUIRED: Create pair RDDs and perform various operations.
OPTIONAL: Complete challenge labs performing more complex operations.

Lab Steps

Perform the following steps:

1. **Create a Pair RDD note in Zeppelin.**

   a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.
   http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

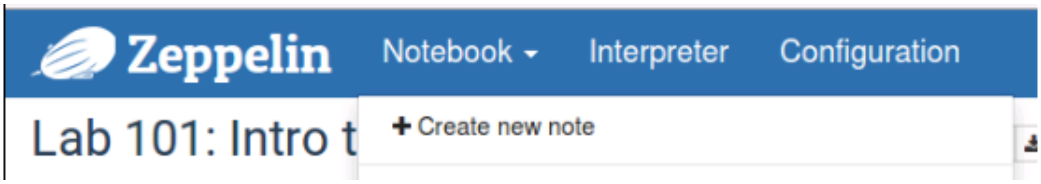
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

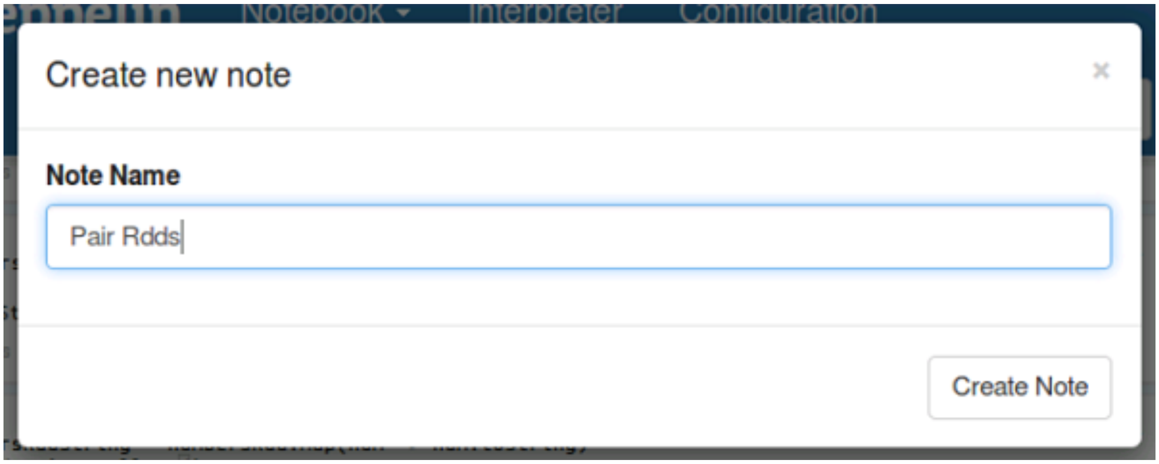
Get started with Zeppelin documentation

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

sandbox:9995/#/
Google
Zeppelin
Notebook
Interpreter
Configuration
Search in your notebooks
Connected
Notebook
Import note
Create new note
Filter
AON Demo
Australian Dataset (Hive example)
Australian Dataset (SparkSQL example)
Create and Manipulate RDDs
Hello World Tutorial
Introduction to Zeppelin
IoT Data Analysis (Keynote Demo)
Lab 101: Intro to Spark with Python
Lab 102: Intro to Spark with Scala
magellan-blog
Phoenix demo
Predicting airline delays
Sensors & Machines Predictive Analysis
Single view demo
Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
twitter
Help
Community
Mailing list
Issues tracking
Github

b. Click on Notebook and select Create new note on the drop down. Name this note Pair RDDs.

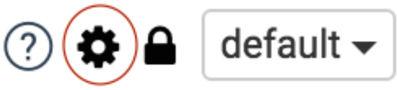


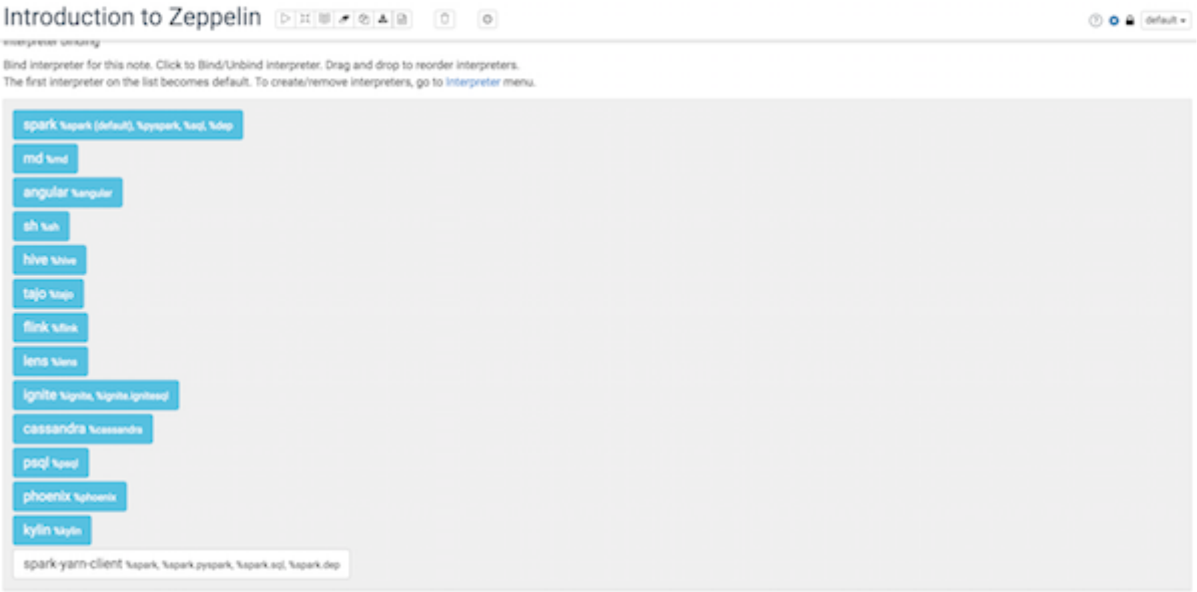

Create new note

Note Name

Pair Rdds

c. At the top right click on the gear icon to change interpreter binding.



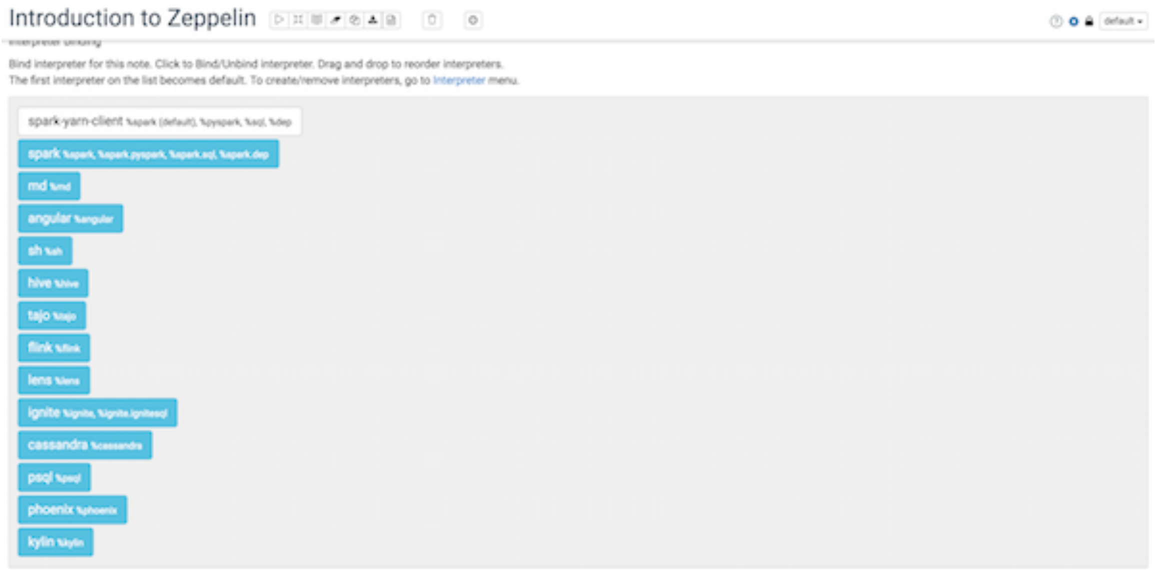

Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

default
spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

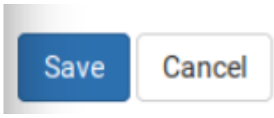
Drag the spark-yarn-client to the top and click save.



Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

The first interpreter on the list becomes default.

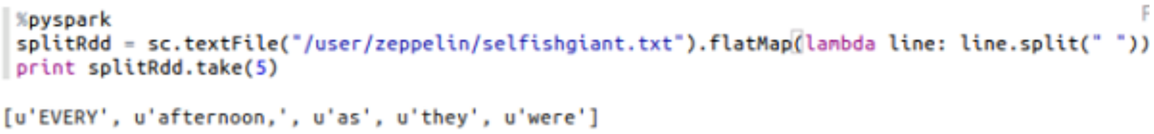


2. **Create a Pair RDD from a text file using map().**

   a. Recreate the RDD splitRDD using the selfishgiant.txt file by importing it to an RDD as a text file and then flatting it into individual word elements. Then view the first 5 words to confirm the RDD exists and is correctly formatted.

      In the code below, there are no line breaks between splitRdd and (" ")). Please refer to the screenshot.

```
%pyspark

splitRdd = sc.textFile("/user/zeppelin/selfishgiant.txt").flatMap(lambda line:
line.split(" "))

print splitRdd.take(5)
```



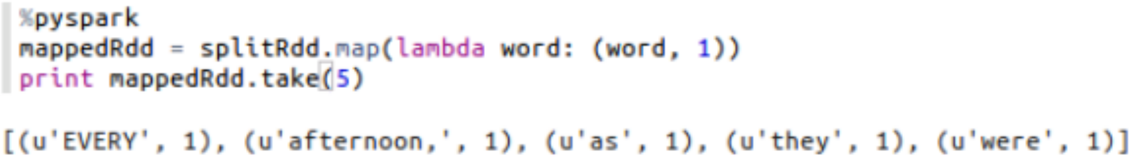
```
%pyspark
splitRdd = sc.textFile("/user/zeppelin/selfishgiant.txt").flatMap(lambda line: line.split(" "))
print splitRdd.take(5)

[u'EVERY', u'afternoon,', u'as', u'they', u'were']
```

NOTE:

In the previous lab, this RDD creation was performed over two steps, creating an intermediary RDD named baseRdd. The creation of the intermediary is not necessary unless it needs to be used in a future step.

b. Use `map()` to create an RDD named `mappedRdd` that converts each element into a key-value pair with a value of 1. View the first five elements to confirm successful operation.

```
%pyspark

mappedRdd = splitRdd.map(lambda word: (word, 1))

print mappedRdd.take(5)
```



```
%pyspark
mappedRdd = splitRdd.map(lambda word: (word, 1))
print mappedRdd.take(5)

[(u'EVERY', 1), (u'afternoon,', 1), (u'as', 1), (u'they', 1), (u'were', 1)]
```

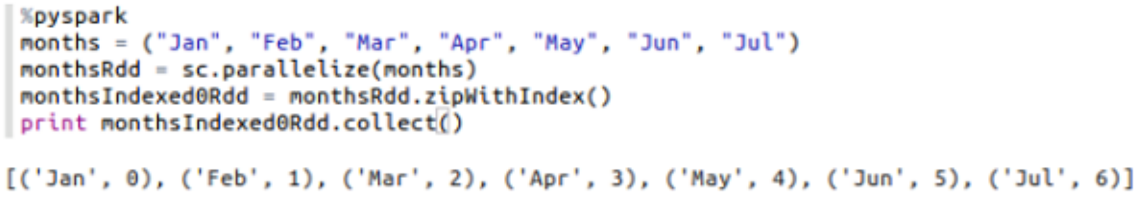
3. Create Pair RDDs using zip functions and perform simple transformations.

a. Create a variable named `months` that contains the values `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, and `Jul` as a list of string values. Convert this to an RDD named `monthsRdd`. Then create another RDD named `monthsIndexed0Rdd` using `zipWithIndex()` to create a Pair RDD that automatically assigns a value to each element based on its position in the list.

REMINDER:

The first element will be assigned a value of "0" using this function.

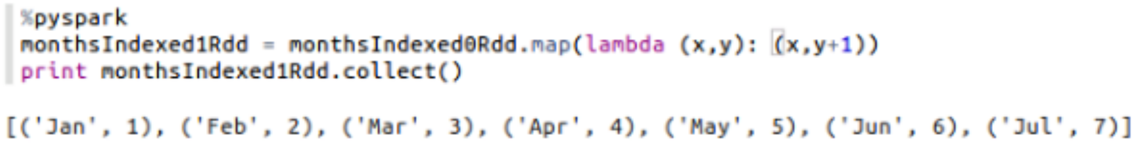
```
%pyspark

months =("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul")

monthsRdd = sc.parallelize(months)

monthsIndexed0Rdd = monthsRdd.zipWithIndex()

print monthsIndexed0Rdd.collect()
```



```
%pyspark
months = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul")
monthsRdd = sc.parallelize(months)
monthsIndexed0Rdd = monthsRdd.zipWithIndex()
print monthsIndexed0Rdd.collect()

[('Jan', 0), ('Feb', 1), ('Mar', 2), ('Apr', 3), ('May', 4), ('Jun', 5), ('Jul', 6)]
```

b. Use `map()` to convert the value for each month to the actual month number and store this in a new RDD named `monthsIndexed1Rdd`. For reference, `Jan` should have a value of 1, `Feb` should have a value of 2, and so on. View the new RDD to confirm success.

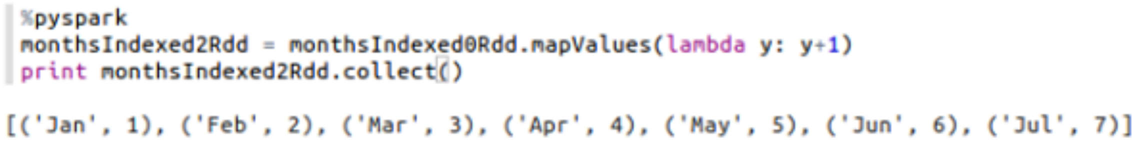
```
%pyspark

monthsIndexed1Rdd = monthsIndexed0Rdd.map(lambda (x,y): (x,y+1))

print monthsIndexed1Rdd.collect()
```



```
%pyspark
monthsIndexed1Rdd = monthsIndexed0Rdd.map(lambda (x,y): (x,y+1))
print monthsIndexed1Rdd.collect()

[('Jan', 1), ('Feb', 2), ('Mar', 3), ('Apr', 4), ('May', 5), ('Jun', 6), ('Jul', 7)]
```

c. Create a new RDD named `monthsIndexed2Rdd` that performs the same operation on `monthsIndexed0Rdd` as in the previous step but uses `mapValues()` instead of `map()` to perform the operation. View the new RDD and confirm it looks identical to the output of `monthsIndexed1Rdd`.

```
%pyspark

monthsIndexed2Rdd = monthsIndexed0Rdd.mapValues(lambda y: y+1)

print monthsIndexed2Rdd.collect()
```



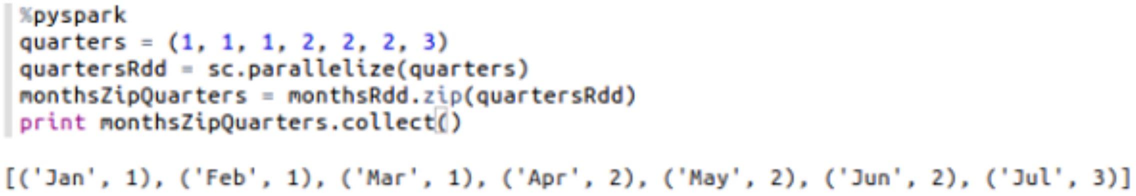
```
%pyspark
monthsIndexed2Rdd = monthsIndexed0Rdd.mapValues(lambda y: y+1)
print monthsIndexed2Rdd.collect()

[('Jan', 1), ('Feb', 2), ('Mar', 3), ('Apr', 4), ('May', 5), ('Jun', 6), ('Jul', 7)]
```

NOTE:

No difference exists between the two previous lab steps from Spark's perspective. The `mapValues` function simply performs a `map()` and returns the key without modification, while performing the function you define on the value.

d. **Create a variable named `quarters` that contains the following seven values: `1, 1, 1, 2, 2, 2,` and `3`. Convert the variable into an RDD named `quartersRdd`. Then create an RDD named `monthsZipQuarters` and use `zip()` to create a Pair RDD that assigns each value from `quartersRdd` to a month in `monthsRdd`. Finally, view the output and make sure that each month was assigned to the correct quarter in the final RDD.**

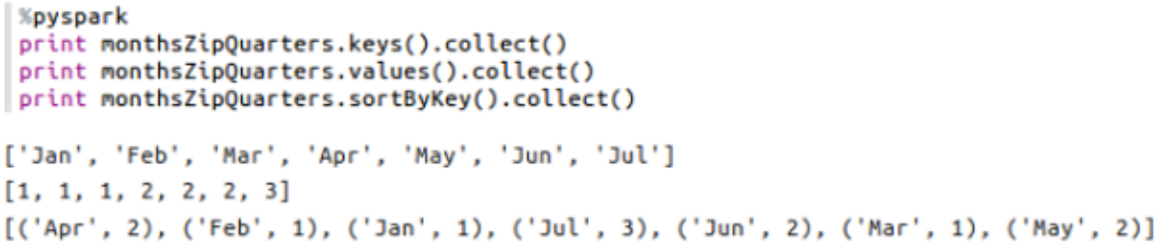
```
%pyspark

quarters = (1, 1, 1, 2, 2, 2, 3)

quartersRdd = sc.parallelize(quarters)

monthsZipQuarters = monthsRdd.zip(quartersRdd)

print monthsZipQuarters.collect()
```



```
%pyspark
quarters = (1, 1, 1, 2, 2, 2, 3)
quartersRdd = sc.parallelize(quarters)
monthsZipQuarters = monthsRdd.zip(quartersRdd)
print monthsZipQuarters.collect()

[('Jan', 1), ('Feb', 1), ('Mar', 1), ('Apr', 2), ('May', 2), ('Jun', 2), ('Jul', 3)]
```

e. **Perform the following operations on `monthsZipQuarters` without creating new RDDs: view the keys only, view the values only, and view the contents of the RDD sorted alphabetically by key.**

```
%pyspark

print monthsZipQuarters.keys().collect()

print monthsZipQuarters.values().collect()

print monthsZipQuarters.sortByKey().collect()
```

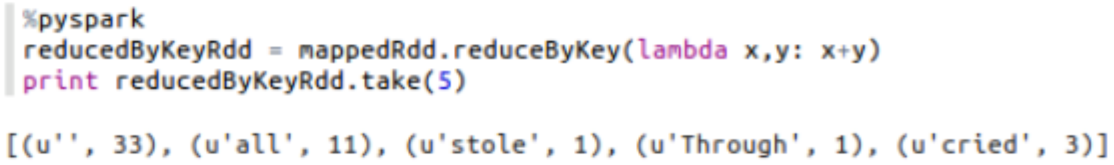


```
%pyspark
print monthsZipQuarters.keys().collect()
print monthsZipQuarters.values().collect()
print monthsZipQuarters.sortByKey().collect()

['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul']
[1, 1, 1, 2, 2, 2, 3]
[('Apr', 2), ('Feb', 1), ('Jan', 1), ('Jul', 3), ('Jun', 2), ('Mar', 1), ('May', 2)]
```

4. **Count the number of times words appear in a Pair RDD and manipulate the output.**

   a. Use the `mappedRDD` created in a previous step and create a new RDD named `reducedByKeyRdd` that reduces the file so that each word appears only once but has a value equal to the number of times it appeared in the original RDD. View the first five elements of the new RDD to confirm successful operation.

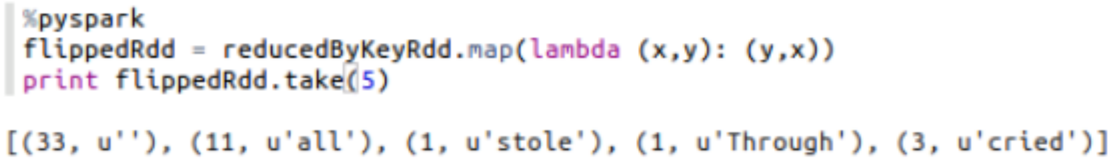
```
%pyspark

reducedByKeyRdd = mappedRdd.reduceByKey(lambda x,y: x+y)

print reducedByKeyRdd.take(5)
```



```
%pyspark
reducedByKeyRdd = mappedRdd.reduceByKey(lambda x,y: x+y)
print reducedByKeyRdd.take(5)

[(u'', 33), (u'all', 11), (u'stole', 1), (u'Through', 1), (u'cried', 3)]
```

   b. Use `map()` to create a new RDD named `flippedRdd` that switches your keys and values so that the current keys become the values, and the values become the keys. View the first five elements of the new RDD to confirm successful operation.

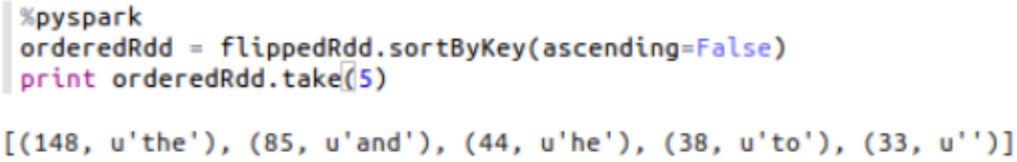
```
%pyspark

flippedRdd = reducedByKeyRdd.map(lambda (x,y): (y,x))

print flippedRdd.take(5)
```



```
%pyspark
flippedRdd = reducedByKeyRdd.map(lambda (x,y): (y,x))
print flippedRdd.take(5)

[(33, u''), (11, u'all'), (1, u'stole'), (1, u'Through'), (3, u'cried')]
```

   c. Create a new RDD named `orderedRdd` that manipulates `flippedRDD` and arranges the words in descending order by number of times they appear. View the first five elements of the new RDD to confirm successful operation.

```
%pyspark

orderedRdd = flippedRdd.sortByKey(ascending=False)

print orderedRdd.take(5)
```



```
%pyspark
orderedRdd = flippedRdd.sortByKey(ascending=False)
print orderedRdd.take(5)

[(148, u'the'), (85, u'and'), (44, u'he'), (38, u'to'), (33, u'')]
```

Result

You have successfully created and manipulated Pair RDD's using various functions.

Challenge Labs

The labs below work with Pair RDDs to perform real-world operations. In some cases, the solutions to the lab utilize programming techniques not explicitly described in the course lecture. These techniques, however, should be clear and easy to understand by carefully following the instructions. If you have questions and are in an instructor-supported class, please ask for assistance as needed.

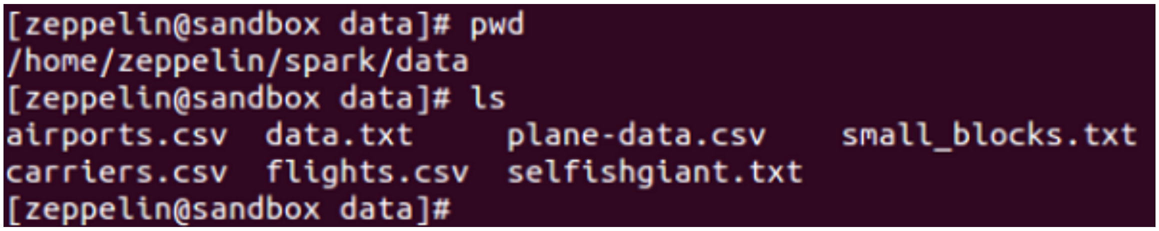
You may want to start by creating a new notebook named Pair RDD Challenge Labs, but this is up to you.

Perform the following steps:

1. **Determine the airlines with the greatest number of flights.**

    a. Go back to a terminal window that has used SSH to connect to the sandbox Docker environment and change to the /home/zeppelin/spark/data directory if necessary. View the contents of this directory and confirm the existence of three files: airports.csv, plane-data.csv, and flights.csv.

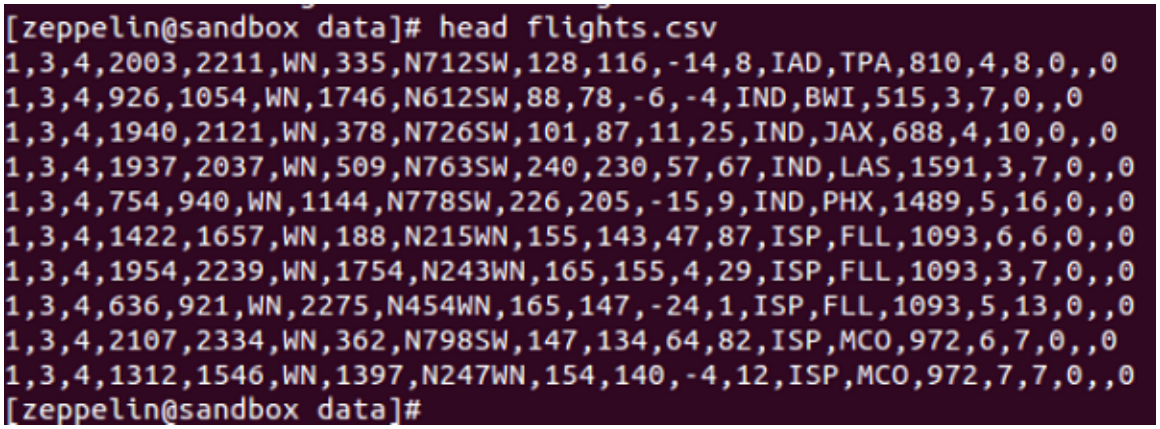
```
# ls
```



```
[zeppelin@sandbox data]# pwd
/home/zeppelin/spark/data
[zeppelin@sandbox data]# ls
airports.csv  data.txt     plane-data.csv    small_blocks.txt
carriers.csv  flights.csv  selfishgiant.txt
[zeppelin@sandbox data]#
```

b. Use `head` to view the first few lines of the flights.csv file.

```
# head flights.csv
```



```
[zeppelin@sandbox data]# head flights.csv
1,3,4,2003,2211,WN,335,N712SW,128,116,-14,8,IAD,TPA,810,4,8,0,,0
1,3,4,926,1054,WN,1746,N612SW,88,78,-6,-4,IND,BWI,515,3,7,0,,0
1,3,4,1940,2121,WN,378,N726SW,101,87,11,25,IND,JAX,688,4,10,0,,0
1,3,4,1937,2037,WN,509,N763SW,240,230,57,67,IND,LAS,1591,3,7,0,,0
1,3,4,754,940,WN,1144,N778SW,226,205,-15,9,IND,PHX,1489,5,16,0,,0
1,3,4,1422,1657,WN,188,N215WN,155,143,47,87,ISP,FLL,1093,6,6,0,,0
1,3,4,1954,2239,WN,1754,N243WN,165,155,4,29,ISP,FLL,1093,3,7,0,,0
1,3,4,636,921,WN,2275,N454WN,165,147,-24,1,ISP,FLL,1093,5,13,0,,0
1,3,4,2107,2334,WN,362,N798SW,147,134,64,82,ISP,MCO,972,6,7,0,,0
1,3,4,1312,1546,WN,1397,N247WN,154,140,-4,12,ISP,MCO,972,7,7,0,,0
[zeppelin@sandbox data]#
```

Each column in the file can be interpreted using the guide below. The first comma-separated value in each line (index number 0) represents the month, the second value represents the day of the month, and so on. Of note for our purposes: the sixth value (index number 5) represents the carrier for each flight.

Field	Index	Example data
Month	0	1
DayofMonth	1	3
DayOfWeek	2	4
DepTime	3	1738
ArrTime	4	1841
UniqueCarrier	**5**	**WN**
FlightNum	6	3948
TailNum	7	N467WN
ElapsedTime	8	63
AirTime	9	49
ArrDelay	10	1
DepDelay	11	8
Origin	12	JAX
Dest	13	FLL
Distance	14	318
TaxiIn	15	6
TaxiOut	16	8
Cancelled	17	0
CancellationCode	18	
Diverted	19	0

c. Use Zeppelin to import this file into the `/user/zeppelin` folder in HDFS.

```
%sh

hdfs dfs -put /home/zeppelin/spark/data/flights.csv /user/zeppelin/flights.csv
```

```
%sh
hdfs dfs -put /home/zeppelin/spark/data/flights.csv /user/zeppelin/flights.csv
```

QUESTION:

Why do this in Zeppelin instead of from the command line?

ANSWER:

When the tasks are performed in a Zeppelin notebook, the entire series of actions can be exported and then imported and replayed on another system. This will be discussed in more detail in a later lab exercise.

d. Create an RDD named `carrierRdd` by performing the following transformations:

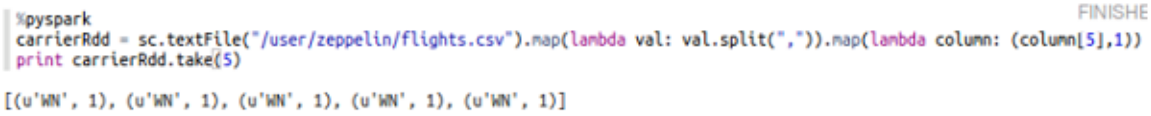
1. Import the text file from HDFS using `sc.textFile()`.

2. Split the lines into an array of individual elements using `map()`
(Hint: The elements are comma-separated rather than space-separated as in previous examples.)

3. Use `map()` to create a key-value pair from only the elements in the sixth column (index number 5) - which can be specified by appending `[5]` to the anonymous function value – and assign each instance a value of 1.

4. View the first five elements to confirm successful operation.

```
%pyspark

carrierRdd = sc.textFile(“/user/zeppelin/flights.csv”).map(lambda val:
val.split(“,”)).map(lambda column: (column[5],1))

print carrierRdd.take(5)
```



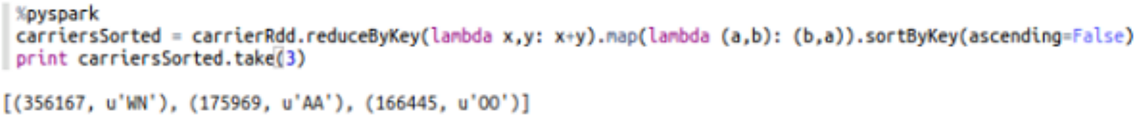
```
%pyspark
carrierRdd = sc.textFile("/user/zeppelin/flights.csv").map(lambda val: val.split(",")).map(lambda column: (column[5],1))
print carrierRdd.take(5)

[(u'WN', 1), (u'WN', 1), (u'WN', 1), (u'WN', 1), (u'WN', 1)]
```

NOTE:

As in a previous example, these operations to create carrierRdd could have been performed in stages, using intermediate RDDs at each transformation step. We do not need the data in these intermediate forms, however, so chaining together multiple transformations to get to the final output works fine.

e. Perform a reduce and sort the results, then display the top three carrier codes by number of flights based on this data.

```
%pyspark

carriersSorted = carrierRdd.reduceByKey(lambda x,y: x+y).map(lambda (a,b):
(b,a)).sortByKey(ascending=False)

print carriersSorted.take(3)
```

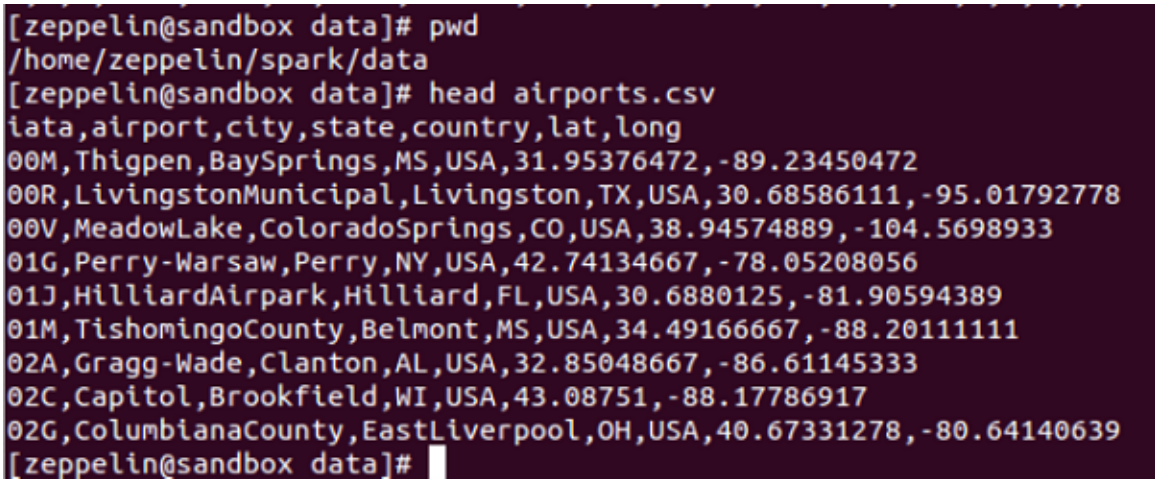


```
%pyspark
carriersSorted = carrierRdd.reduceByKey(lambda x,y: x+y).map(lambda (a,b): (b,a)).sortByKey(ascending=False)
print carriersSorted.take(3)

[(356167, u'WN'), (175969, u'AA'), (166445, u'OO')]
```

2. Determine the most common routes between two cities.

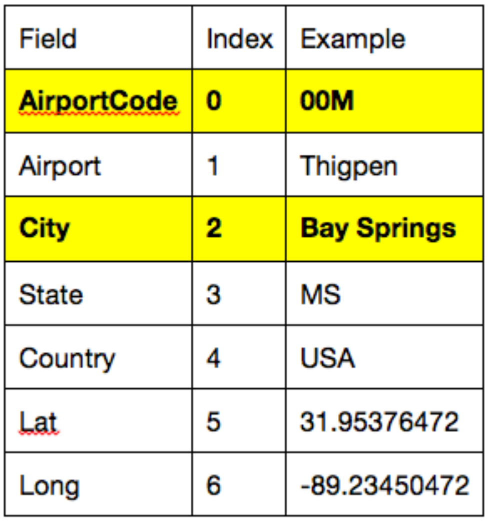
a. The next exercise uses the flights.csv file from the previous lab, as well as the airports.csv file. Go back to the terminal window and take a look at the first few lines of the airports.csv file.

```
# head airports.csv
```



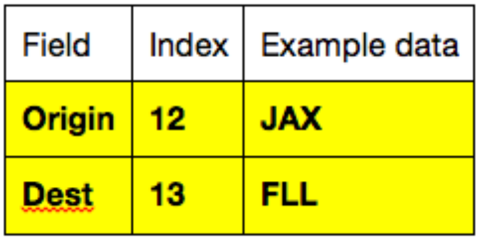
```
[zeppelin@sandbox data]# pwd
/home/zeppelin/spark/data
[zeppelin@sandbox data]# head airports.csv
iata,airport,city,state,country,lat,long
00M,Thigpen,BaySprings,MS,USA,31.95376472,-89.23450472
00R,LivingstonMunicipal,Livingston,TX,USA,30.68586111,-95.01792778
00V,MeadowLake,ColoradoSprings,CO,USA,38.94574889,-104.5698933
01G,Perry-Warsaw,Perry,NY,USA,42.74134667,-78.05208056
01J,HilliardAirpark,Hilliard,FL,USA,30.6880125,-81.90594389
01M,TishomingoCounty,Belmont,MS,USA,34.49166667,-88.20111111
02A,Gragg-Wade,Clanton,AL,USA,32.85048667,-86.61145333
02C,Capitol,Brookfield,WI,USA,43.08751,-88.17786917
02G,ColumbianaCounty,EastLiverpool,OH,USA,40.67331278,-80.64140639
[zeppelin@sandbox data]#
```

Each column in the file can be interpreted using the guide below. The first comma-separated value in each line (index number 0) represents the airport code, the second value represents the airport name, and so on. Of note for our purposes: the airport code (index number 0) and the airport city (index number 2).



Field	Index	Example
AirportCode	**0**	**00M**
Airport	1	Thigpen
City	**2**	**Bay Springs**
State	3	MS
Country	4	USA
Lat	5	31.95376472
Long	6	-89.23450472

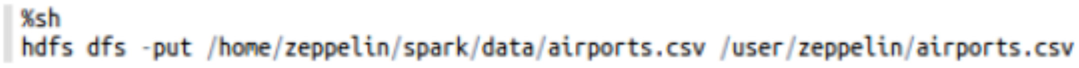
From the flights.csv file used earlier, columns 13 and 14 (index values 12 and 13) will be used in this exercise.



Field	Index	Example data
Origin	**12**	**JAX**
Dest	**13**	**FLL**

b. Use Zeppelin to import the airports.csv file into the `/user/zeppelin` folder in HDFS.

```
%sh

hdfs dfs -put /home/zeppelin/spark/data/airports.csv
/user/zeppelin/airports.csv
```



```
%sh
hdfs dfs -put /home/zeppelin/spark/data/airports.csv /user/zeppelin/airports.csv
```

c. Create an RDD named `cityRdd` by performing the following transformations:

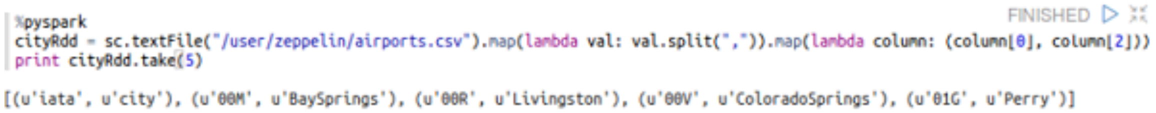
1. Import the text file from HDFS using `sc.textFile()`.

2. Split the lines into an array of individual elements using `map()`
(Hint: Once again, the elements are comma-separated rather than space-separated.)

3. Use `map()` to pull out only the airport code and city elements in the first and third columns (index numbers 0 and 2).

4. View the first five elements to confirm successful operation.

```
%pyspark

cityRdd = sc.textFile(“/user/zeppelin/airports.csv”).map(lambda val:
val.split(“,”)).map(lambda column: (column[0], column[2]))


print cityRdd.take(5)
```



```
%pyspark
cityRdd = sc.textFile("/user/zeppelin/airports.csv").map(lambda val: val.split(",")).map(lambda column: (column[0], column[2]))
print cityRdd.take(5)

[(u'iata', u'city'), (u'00M', u'BaySprings'), (u'00R', u'Livingston'), (u'00V', u'ColoradoSprings'), (u'01G', u'Perry')]
```

d. Create an RDD named `flightOrigDestRdd` by performing the following transformations:

1. Import the text file from HDFS using `sc.textFile()`.

2. Split the lines into an array of individual elements using `map()`.

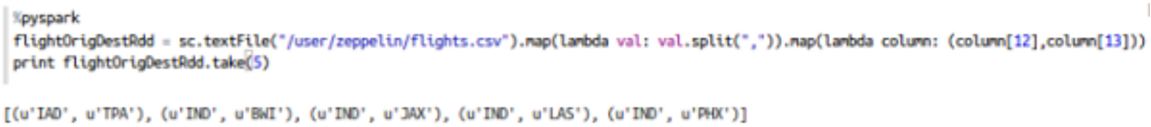
3. Use `map()` to pull out only the origin and destination elements in the 13th and 14th columns (index numbers 12 and 13).

4. View the first five elements to confirm successful operation.

NOTE:

Some of this code can be copied and pasted from a previous paragraph in the Zeppelin notebook.

```
%pyspark

flightOrigDestRdd = sc.textFile(“/user/zeppelin/flights.csv”).map(lambda val:
val.split(“,”)).map(lambda column: (column[12],column[13]))

print flightOrigDestRdd.take(5)
```



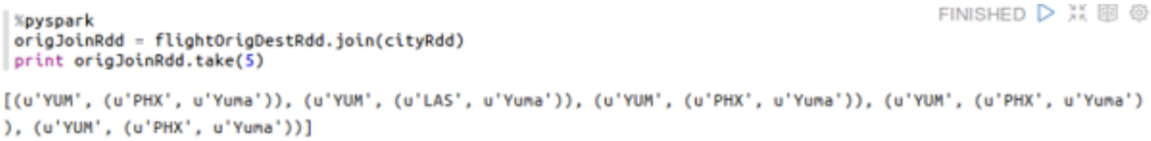
```
%pyspark
flightOrigDestRdd = sc.textFile("/user/zeppelin/flights.csv").map(lambda val: val.split(",")).map(lambda column: (column[12],column[13]))
print flightOrigDestRdd.take(5)

[(u'IAD', u'TPA'), (u'IND', u'BWI'), (u'IND', u'JAX'), (u'IND', u'LAS'), (u'IND', u'PHX')]
```

e. Use `join()` to join `flightOrigDestRdd` and `cityRdd` into a third RDD named `origJoinRdd`.

This operation will result in an RDD that contains the origin code as the key, with a value of (destination code, origin city). This is half of the operation needed to get origin and destination cities.

View the first five elements to confirm successful operation.

```
%pyspark

origJoinRdd = flightOrigDestRdd.join(cityRdd)

print origJoinRdd.take(5)
```



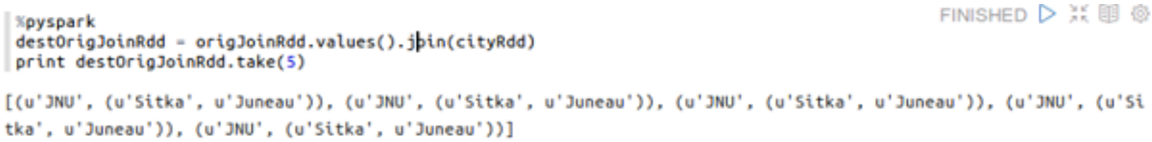
```
%pyspark
origJoinRdd = flightOrigDestRdd.join(cityRdd)
print origJoinRdd.take(5)

[(u'YUM', (u'PHX', u'Yuma')), (u'YUM', (u'LAS', u'Yuma')), (u'YUM', (u'PHX', u'Yuma')), (u'YUM', (u'PHX', u'Yuma')
), (u'YUM', (u'PHX', u'Yuma'))]
```

f. Next use `join()` again to create an RDD named `destOrigJoinRdd` using `origJoinRdd` as a source and joining it `cityRdd` once again. Before performing the join operation, use `values()` to filter out the origin code (which is no longer needed) and pull out only the destination code and city name from the previous transformation.

   This operation will result in an RDD that contains the destination code as the key, with a value of (origin city, destination city).

   View the first five elements to confirm successful operation.

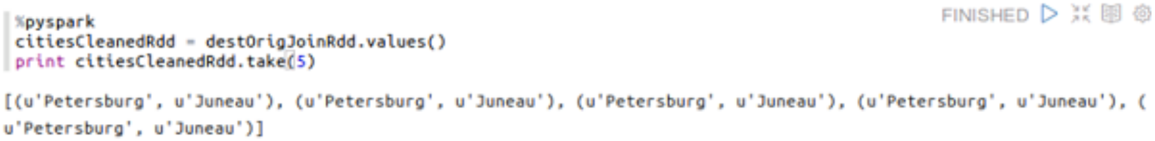
```
%pyspark

destOrigJoinRdd = origJoinRdd.values().join(cityRdd)

print destOrigJoinRdd.take(5)
```



```
%pyspark
destOrigJoinRdd = origJoinRdd.values().join(cityRdd)
print destOrigJoinRdd.take(5)

[(u'JNU', (u'Sitka', u'Juneau')), (u'JNU', (u'Sitka', u'Juneau')), (u'JNU', (u'Sitka', u'Juneau')), (u'JNU', (u'Si
tka', u'Juneau')), (u'JNU', (u'Sitka', u'Juneau'))]
```

g. Create another RDD named `citiesCleanedRdd` that contains only the values of the `destOrigJoinRdd` (in other words, just the origin and destination city names). View the first five elements to confirm successful operation.

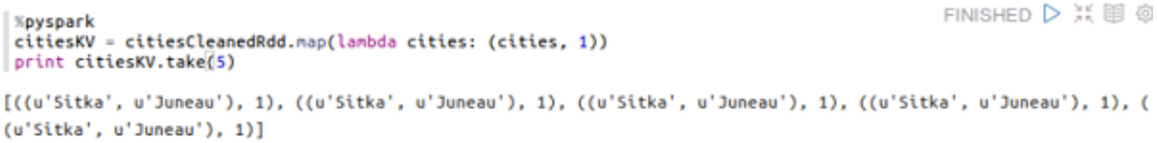
```
%pyspark

citiesCleanedRdd = destOrigJoinRdd.values()

print citiesCleanedRdd.take(5)
```



```
%pyspark
citiesCleanedRdd = destOrigJoinRdd.values()
print citiesCleanedRdd.take(5)

[(u'Petersburg', u'Juneau'), (u'Petersburg', u'Juneau'), (u'Petersburg', u'Juneau'), (u'Petersburg', u'Juneau'), (
u'Petersburg', u'Juneau')]
```

h. **Use `map()` to convert the key-value pairs in `citiesCleanedRdd` into keys for a new RDD named `citiesKV`, and give each key a value of 1. View the first five elements to confirm successful operation.**

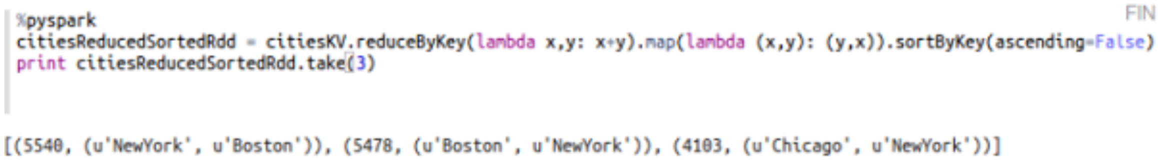
```
%pyspark

citiesKV = citiesCleanedRdd.map(lambda cities: (cities, 1))

print citiesKV.take(5)
```



```
%pyspark
citiesKV = citiesCleanedRdd.map(lambda cities: (cities, 1))
print citiesKV.take(5)

[((u'Sitka', u'Juneau'), 1), ((u'Sitka', u'Juneau'), 1), ((u'Sitka', u'Juneau'), 1), ((u'Sitka', u'Juneau'), 1), (
(u'Sitka', u'Juneau'), 1)]
```

i. **Create an RDD named `citiesReducedSortedRdd` that reduces by key, swaps the keys and values, and then sorts by key in descending order. View the first three elements to confirm successful operation.**

```
%pyspark

citiesReducedSortedRdd = citiesKV.reduceByKey(lambda x,y: x+y).map(lambda
(x,y): (y,x)).sortByKey(ascending=False)

print citiesReducedSortedRdd.take(3)
```



```
%pyspark
citiesReducedSortedRdd = citiesKV.reduceByKey(lambda x,y: x+y).map(lambda (x,y): (y,x)).sortByKey(ascending=False)
print citiesReducedSortedRdd.take(3)

[(5540, (u'NewYork', u'Boston')), (5478, (u'Boston', u'NewYork')), (4103, (u'Chicago', u'NewYork'))]
```

NOTE:

The top three origin city / destination combinations are New York to Boston, Boston to New York, and Chicago to New York.

3. Find the longest departure delays for any airline that experienced a delay of 15 minutes or more.

This exercise once again uses the flights.csv file. This time we use the unique carrier code in column 6 (index value 5) and the departure delay value in minutes, which is in column 12 (index value 11).

Field	Index	Example data
UniqueCarrier	**5**	**WN**
DepDelay	**11**	**8**

b. Create an RDD named `delayRdd` by performing the following transformations:

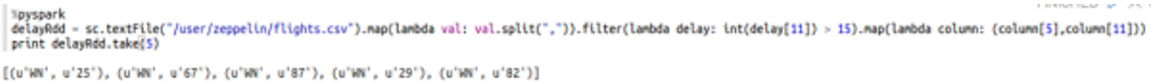
1. Import the flights.csv file from HDFS using `sc.textFile()`.

2. Split the lines into an array of individual elements using `map()`.

3. Use `filter()` to remove any lines for which the value of column 12 (index value 11) is less than 15. Because the `sc.textFile()` operation reads in all values as strings, you will need to cast the values in column 12 as integers prior to performing the `filter()` evaluation.

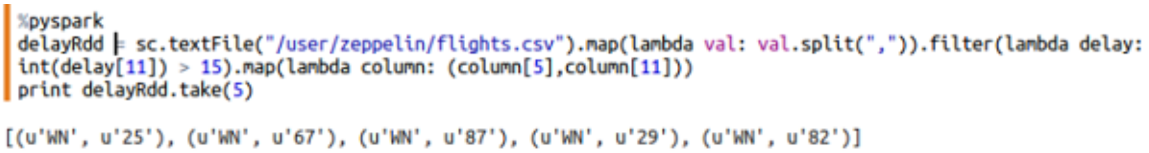
4. Use `map()` to pull out only the carrier code and departure delay elements in the 6th and 12th columns (index numbers 5 and 11).

5. View the first five elements to confirm successful operation.



```
%pyspark
delayRdd = sc.textFile("/user/zeppelin/flights.csv").map(lambda val: val.split(",")).filter(lambda delay: int(delay[11]) > 15).map(lambda column: (column[5],column[11]))
print delayRdd.take(5)

[(u'WN', u'25'), (u'WN', u'67'), (u'WN', u'87'), (u'WN', u'29'), (u'WN', u'82')]
```

For sake or readability, here is another screenshot of the above code with lines wrapped so that the code can be viewed in a larger font.



```
%pyspark
delayRdd = sc.textFile("/user/zeppelin/flights.csv").map(lambda val: val.split(",")).filter(lambda delay:
int(delay[11]) > 15).map(lambda column: (column[5],column[11]))
print delayRdd.take(5)

[(u'WN', u'25'), (u'WN', u'67'), (u'WN', u'87'), (u'WN', u'29'), (u'WN', u'82')]
```

c. Create an RDD named `delayMaxRdd` that reduces the elements in `delayRdd` and returns only the longest delay per airline. For this exercise, it is not necessary to sort the values from largest to smallest.

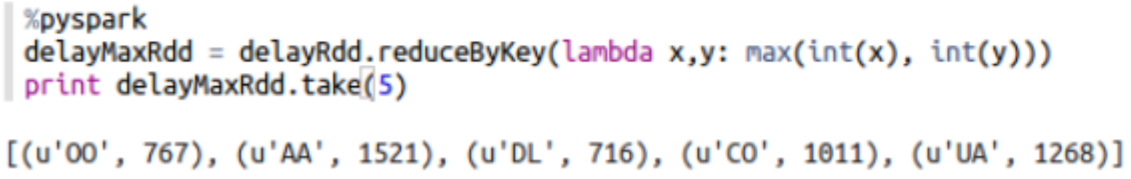
   Display five values to confirm successful operation.

NOTE:

The reduce operation will need to compare all values for the same key and only keep the largest value in the final output.

The values in `delayRdd` are strings, so to compare the values they will first need to be cast as integers, similar to the `filter()` operation performed in the first step of this exercise.

```
%pyspark

delayMaxRdd = delayRdd.reduceByKey(lambda x,y: max(int(x), int(y)))

print delayMaxRdd.take(5)
```

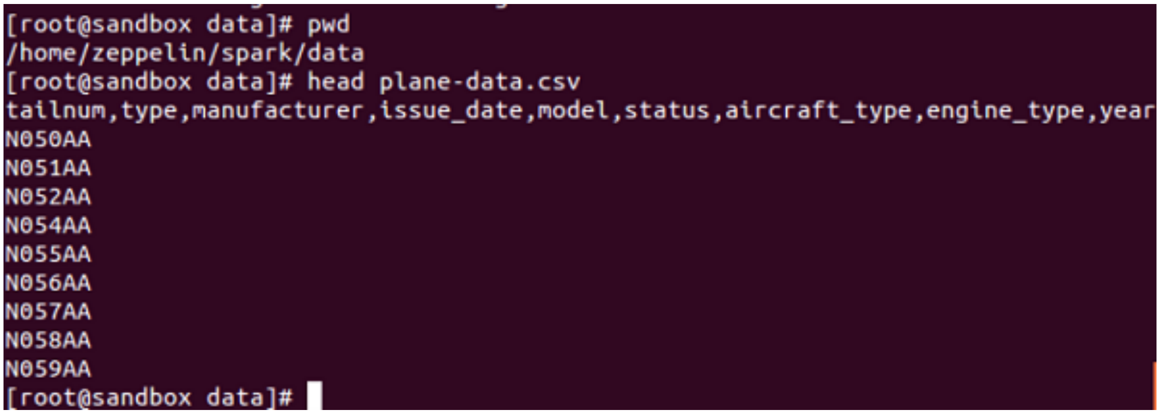


```
%pyspark
delayMaxRdd = delayRdd.reduceByKey(lambda x,y: max(int(x), int(y)))
print delayMaxRdd.take(5)

[(u'OO', 767), (u'AA', 1521), (u'DL', 716), (u'CO', 1011), (u'UA', 1268)]
```

4. **Remove records than contain incomplete data from a file.**

   a. The next exercise uses the plane-data.csv. Go back to the terminal window and take a look at the first few lines of the plane-data.csv file.

```
# head plane-data.csv
```

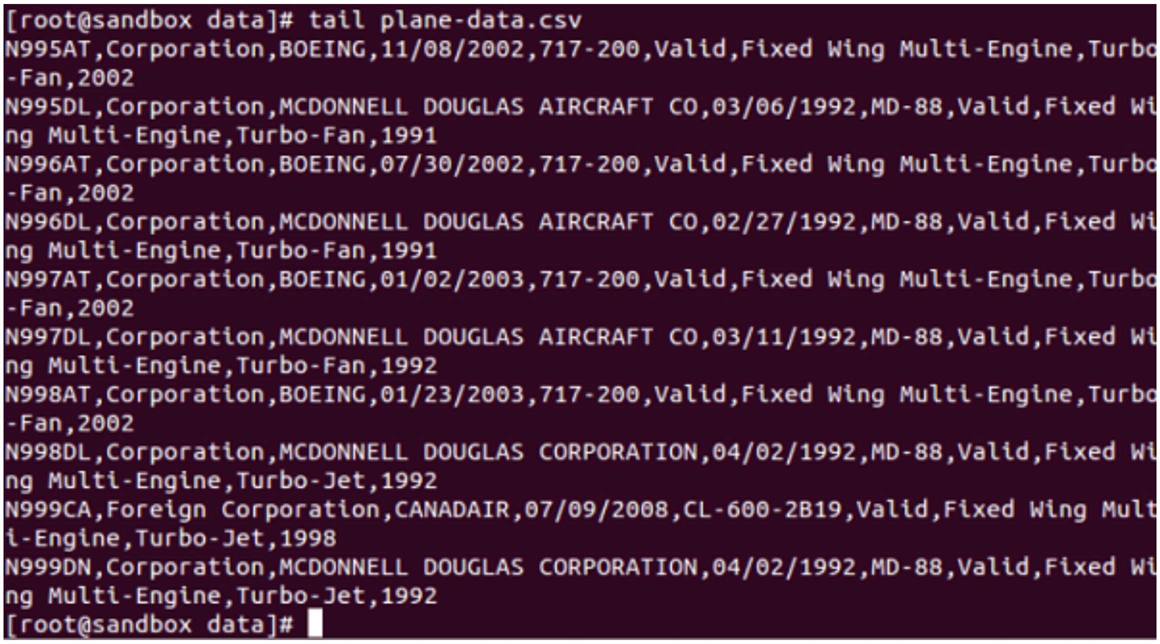


```
[root@sandbox data]# pwd
/home/zeppelin/spark/data
[root@sandbox data]# head plane-data.csv
tailnum,type,manufacturer,issue_date,model,status,aircraft_type,engine_type,year
N050AA
N051AA
N052AA
N054AA
N055AA
N056AA
N057AA
N058AA
N059AA
[root@sandbox data]#
```

Note that in the screenshot above, this file contains the column header names, followed by the column values. In this case, the first few records only have values for the first column, and the rest of the values are blank.

To see what complete records should look like, take a look at the last few lines of the file.

```
# tail plane-data.csv
```



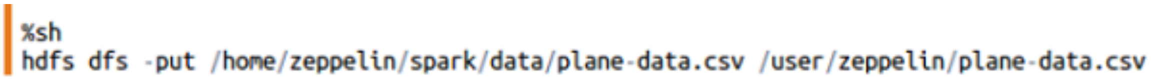
```
[root@sandbox data]# tail plane-data.csv
N995AT,Corporation,BOEING,11/08/2002,717-200,Valid,Fixed Wing Multi-Engine,Turbo
-Fan,2002
N995DL,Corporation,MCDONNELL DOUGLAS AIRCRAFT CO,03/06/1992,MD-88,Valid,Fixed Wi
ng Multi-Engine,Turbo-Fan,1991
N996AT,Corporation,BOEING,07/30/2002,717-200,Valid,Fixed Wing Multi-Engine,Turbo
-Fan,2002
N996DL,Corporation,MCDONNELL DOUGLAS AIRCRAFT CO,02/27/1992,MD-88,Valid,Fixed Wi
ng Multi-Engine,Turbo-Fan,1991
N997AT,Corporation,BOEING,01/02/2003,717-200,Valid,Fixed Wing Multi-Engine,Turbo
-Fan,2002
N997DL,Corporation,MCDONNELL DOUGLAS AIRCRAFT CO,03/11/1992,MD-88,Valid,Fixed Wi
ng Multi-Engine,Turbo-Fan,1992
N998AT,Corporation,BOEING,01/23/2003,717-200,Valid,Fixed Wing Multi-Engine,Turbo
-Fan,2002
N998DL,Corporation,MCDONNELL DOUGLAS CORPORATION,04/02/1992,MD-88,Valid,Fixed Wi
ng Multi-Engine,Turbo-Jet,1992
N999CA,Foreign Corporation,CANADAIR,07/09/2008,CL-600-2B19,Valid,Fixed Wing Mult
i-Engine,Turbo-Jet,1998
N999DN,Corporation,MCDONNELL DOUGLAS CORPORATION,04/02/1992,MD-88,Valid,Fixed Wi
ng Multi-Engine,Turbo-Jet,1992
[root@sandbox data]#
```

Each column in the file can be interpreted using the guide below. Note that there are nine possible column values for each record (index 0 through 8).

Field	Index	Example
Tailnum	0	N10156
Type	1	Corporation
Manufacturer	2	EMBRAER
Issue_date	3	02/13/2004
Model	4	EMB-145XR
Status	5	Valid
Aircraft_type	6	Fixed Wing Multi-Engine
Engine_type	7	Turbo-Fan
Year	8	2004

b. Use Zeppelin to import the plane-data.csv file into the `/user/zeppelin` folder in HDFS.

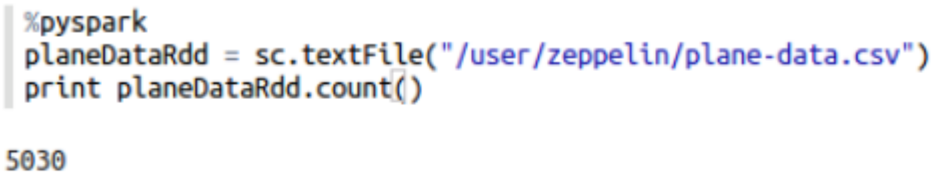
```
%sh

hdfs dfs -put /home/zeppelin/spark/data/plane-data.csv /user/zeppelin/plane-
data.csv
```



```
%sh
hdfs dfs -put /home/zeppelin/spark/data/plane-data.csv /user/zeppelin/plane-data.csv
```

c. Create an RDD named `planeDataRdd` from the plane-data.csv file. Before performing any transformations, use `count()` to display the number of lines in the RDD.

```
%pyspark

planeDataRdd = sc.textFile(“/user/zeppelin/plane-data.csv”)

print planeDataRdd.count()
```



```
%pyspark
planeDataRdd = sc.textFile("/user/zeppelin/plane-data.csv")
print planeDataRdd.count()

5030
```

d. Create an RDD named `cleanedPlaneDataRdd` by performing the following transformations:

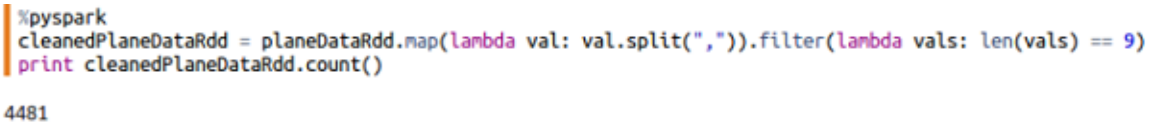
1. Start with `planeDataRdd` from the previous step.

2. Split the lines into an array of individual elements using `map()`. (Hint: The elements are comma-separated.)

3. Use `filter()` to remove any lines that do not have a length of exactly 9 elements.

4. Use `count()` to display the number of lines in the new RDD and confirm that the data set contains fewer lines than before.

```
%pyspark

cleanedPlaneDataRdd = planeDataRdd.map(lambda val:
val.split(“,”)).filter(lambda elements: len(elements) == 9)

print cleanedPlaneDataRdd.count()
```



```
%pyspark
cleanedPlaneDataRdd = planeDataRdd.map(lambda val: val.split(",")).filter(lambda vals: len(vals) == 9)
print cleanedPlaneDataRdd.count()

4481
```

Bonus Challenge Labs

The lab exercises below are for advanced students only. Instructor support and solutions will *not* be provided for these exercises. Some of the coding skills required to complete exercise 6 have not been covered in this class.

Perform the following steps:

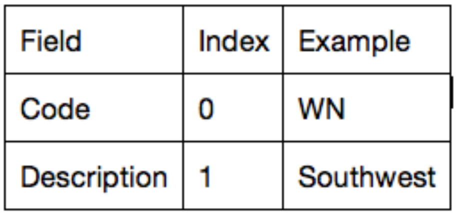
1. **Extend CHALLENGE LABS exercise 4 by finding the top three most common airplane models for flights over 1500 miles.**

   Both flights.csv and plane-data.csv will be used to solve this exercise.

2. **Extend CHALLENGE LABS exercises 1 and 3 by returning the names of the airlines rather than their carrier codes.**

   To perform this extension, another file in the `/home/zeppelin/spark/data` directory must be used: carriers.csv.

   The data in this file contains two columns, as indicated below:



Field	Index	Example
Code	0	WN
Description	1	Southwest

BE AWARE:

This data contains additional challenges. The first row of the data contains column headers, just like plane-data.csv did. However, in addition, in some cases the description of the airline includes a comma that is not meant to separate values. For example, the airline with code 09Q is has a description of Swift Air, LLC. The comma is part of the business name.

Good luck!

Lab 5: Basic Spark Streaming (Python)

About This Lab

Objective:
Set up basic Spark Streaming operations using the REPL

File Locations:
/root/spark/data/

Successful Outcome:
Stream data from HDFS directories and TCP sockets using Spark Streaming

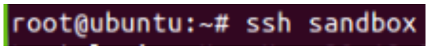
Lab Steps

Perform the following steps:

1. **Use an HDFS directory as a streaming source.**

   a. Open a terminal window and SSH into sandbox.

```
# ssh sandbox
```



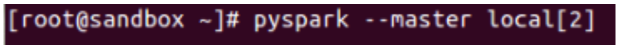
```
root@ubuntu:~# ssh sandbox
```

   b. Create an HDFS directory for streaming output.

```
# hdfs dfs -mkdir /user/root/test/stream
```

   c. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

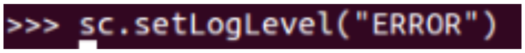
```
# pyspark --master local[2]
```



```
[root@sandbox ~]# pyspark --master local[2]
```

   d. Set the log level to ERROR to avoid screen clutter while running the streaming application.

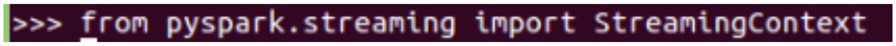
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

   e. Import the streaming library.

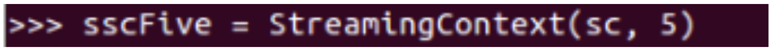
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

f. Create a streaming context with a five-second batch duration.

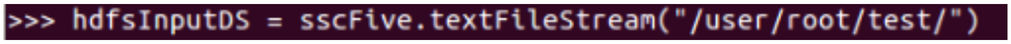
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

g. Create a DStream using `textFileStream()` to monitor the local HDFS directory `/user/root/test/`.

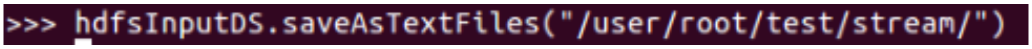
```
>>> hdfsInputDS = sscFive.textFileStream("/user/root/test/")
```



```
>>> hdfsInputDS = sscFive.textFileStream("/user/root/test/")
```

h. Use `saveAsTextFiles()` to save the outputs to `/user/root/test/stream`.

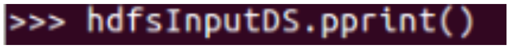
```
>>> hdfsInputDS.saveAsTextFiles("/user/root/test/stream/")
```



```
>>> hdfsInputDS.saveAsTextFiles("/user/root/test/stream/")
```

i. Print out the output to the terminal window.

```
>>> hdfsInputDS.pprint()
```

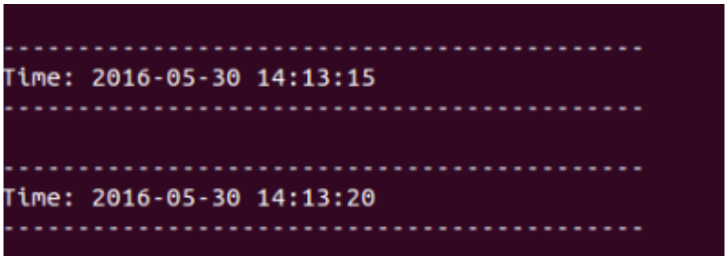


```
>>> hdfsInputDS.pprint()
```

j. Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.

```
>>> sscFive.start()
```

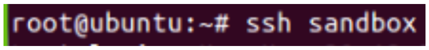
```
>>> sscFive.start()
```



```
-------------------------------------------
Time: 2016-05-30 14:13:15
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 14:13:20
-------------------------------------------
```

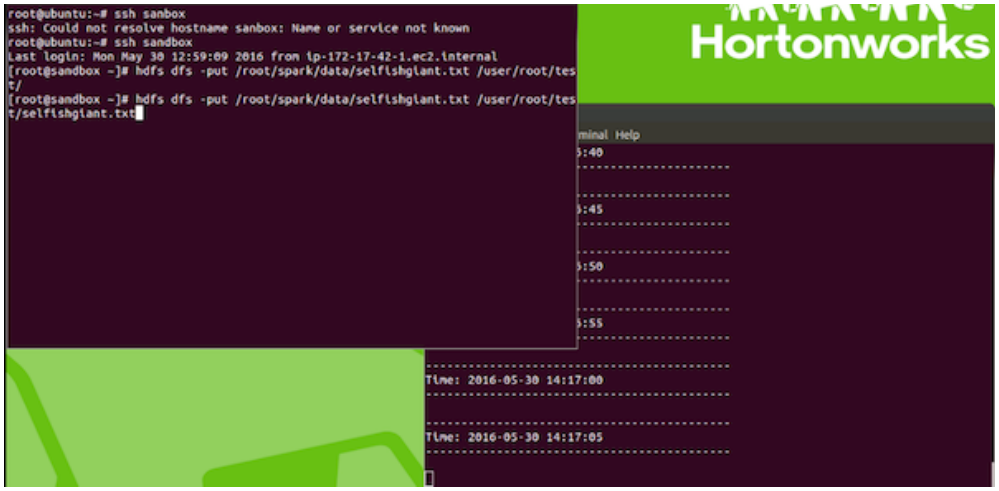
k. Open a new terminal window, SSH to sandbox, and place the input file selfishgiant.txt from `/root/spark/data` into the folder. Observe what happens a few seconds later in the streaming terminal window.

```
# ssh sandbox

# hdfs dfs -put /root/spark/data/selfishgiant.txt /user/root/test/
```



```
root@ubuntu:~# ssh sandbox
```

```
[root@sandbox ~]# hdfs dfs -put /root/spark/data/selfishgiant.txt /user/root/tes
t/selfishgiant.txt
```



```
root@ubuntu:~# ssh sanbox
ssh: Could not resolve hostname sanbox: Name or service not known
root@ubuntu:~# ssh sandbox
Last login: Mon May 30 12:59:09 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]# hdfs dfs -put /root/spark/data/selfishgiant.txt /user/root/tes
t/
[root@sandbox ~]# hdfs dfs -put /root/spark/data/selfishgiant.txt /user/root/tes
t/selfishgiant.txt
```

```
5:40
-------------------------------------------

-------------------------------------------
5:45
-------------------------------------------

-------------------------------------------
5:50
-------------------------------------------

-------------------------------------------
5:55
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 14:17:00
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 14:17:05
-------------------------------------------
```

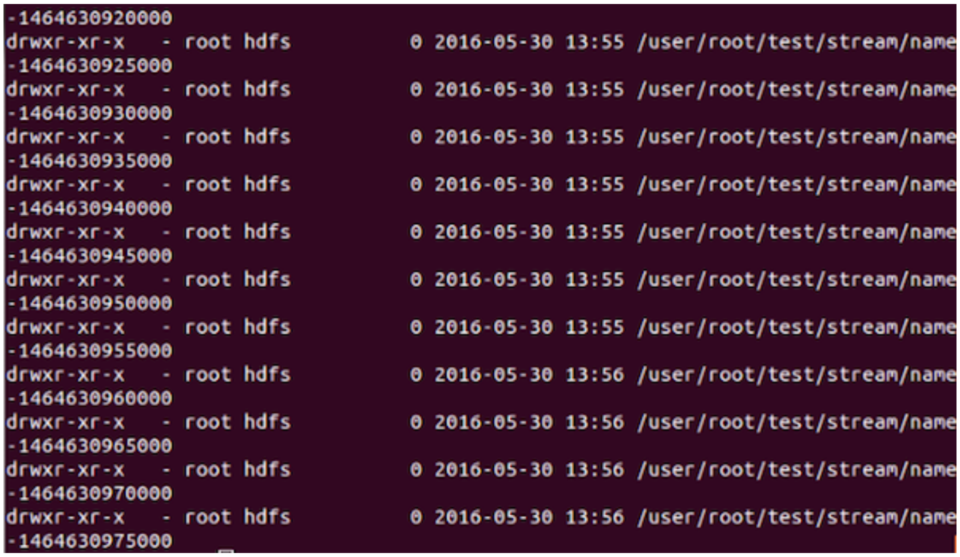
NOTE:

You are free to upload additional files to see more streaming take place if you want.

l. Once you observe data being streamed on-screen in the first terminal window, use the second terminal window to list the contents of the `/user/root/test/stream/` directory on HDFS.

```
#hdfs dfs -ls /user/root/test/stream/
```

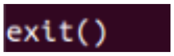
```
[root@sandbox ~]# hdfs dfs -ls /user/root/test/stream
```



```
-1464630920000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630925000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630930000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630935000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630940000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630945000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630950000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630955000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630960000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630965000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630970000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630975000
```

m. In the first terminal window, stop the stream and exit the REPL. If the stream refreshes while you are typing, that will not affect the input. Simply continue to type the command and press enter.

```
sc.stop()
exit()
```



```
exit()
```

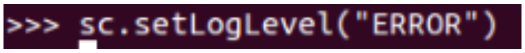
2. **Use a TCP socket as a streaming source.**

a. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

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

b. Set the log level to ERROR to avoid screen clutter while running the streaming application.

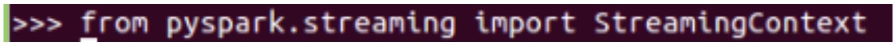
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

c. Import the streaming library.

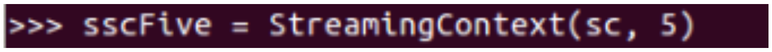
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

d. **Create a streaming context with a five-second batch duration.**

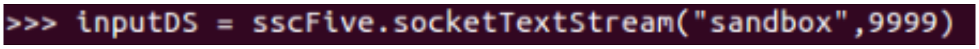
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

e. **Create a DStream using** `socketTestStream()` **to the system named "sandbox" on port 9999.**

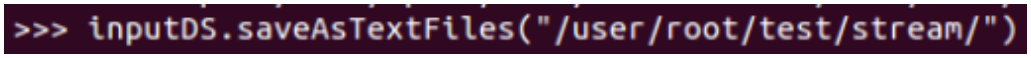
```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```



```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```

f. **Use** `saveAsTextFiles()` **to save the outputs to** `/user/root/test/stream`**.**

```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
```



```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
```

g. **Print out the output to the terminal window.**

```
>>> inputDS.pprint()
```

```
>>> inputDS.pprint()
```

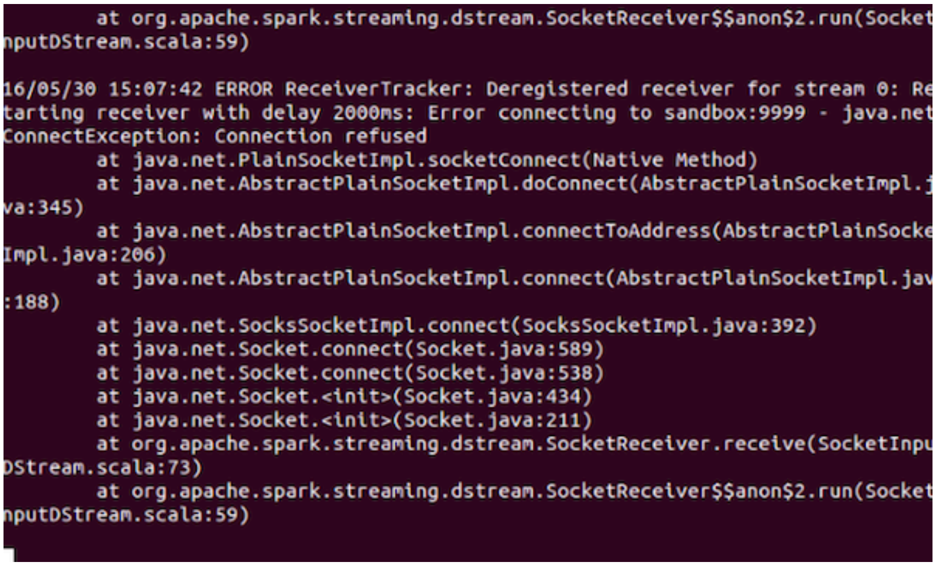
h. **Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.**

```
>>> sscFive.start()
```

```
>>> sscFive.start()
```

NOTE:

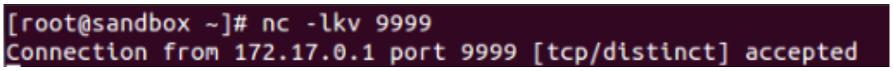
An error will appear when the application starts because the application is waiting for an input connection.



```
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)

16/05/30 15:07:42 ERROR ReceiverTracker: Deregistered receiver for stream 0: Re
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.j
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocke
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.jav
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInpu
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)
```

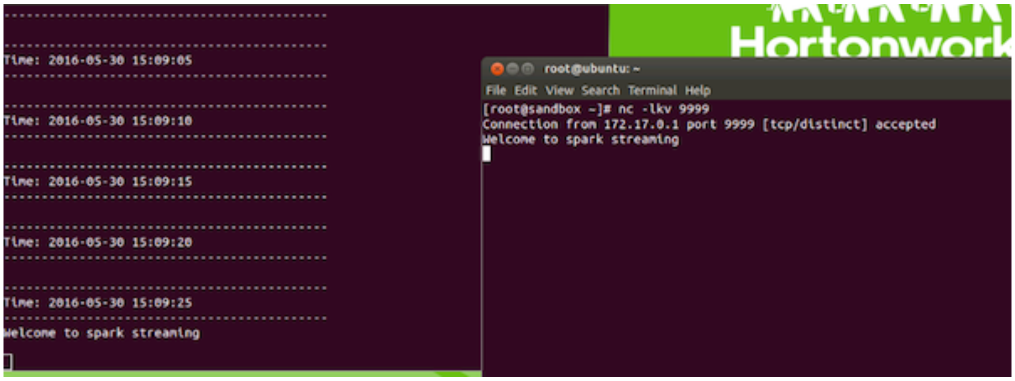
i. In the second terminal window use the `netcat` utility to create a connection to port 9999.

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

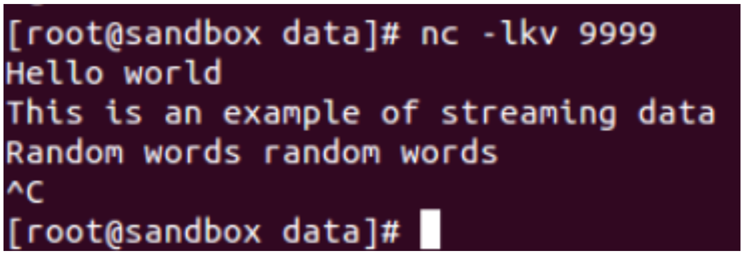
j. Start typing words separated by space, hit Enter occasionally to submit them. Observe what happens in the streaming terminal window a few seconds after hitting Enter.



```
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:05
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:10
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:15
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:20
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:25
-------------------------------------------
Welcome to spark streaming
```

```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
Welcome to spark streaming
```

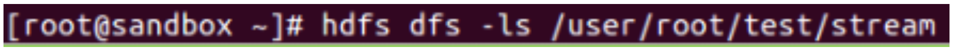
k. Once you observe data being streamed on-screen in the first terminal window, use Ctrl + C (or Cmd + C if using a Mac) to exit netcat in the second terminal window.



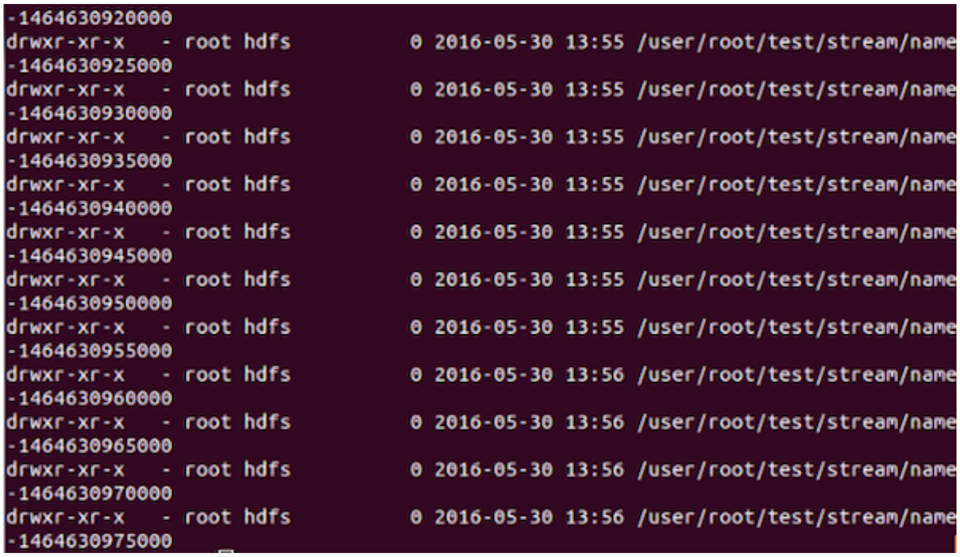
```
[root@sandbox data]# nc -lkv 9999
Hello world
This is an example of streaming data
Random words random words
^C
[root@sandbox data]# 
```

l. Use the second terminal window to list the contents of the `/user/root/test/stream/` directory on HDFS. Note the time stamps on the files.

```
#hdfs dfs -ls /user/root/test/stream/
```



```
[root@sandbox ~]# hdfs dfs -ls /user/root/test/stream
```



```
-1464630920000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630925000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630930000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630935000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630940000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630945000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630950000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:55 /user/root/test/stream/name
-1464630955000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630960000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630965000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630970000
drwxr-xr-x   - root hdfs          0 2016-05-30 13:56 /user/root/test/stream/name
-1464630975000
```

m. In the first terminal window, stop the stream and exit the REPL.

```
sc.stop()
exit()
```

Result

You have created data streams from HDFS and TCP socket sources, observed the stream in real-time, and observed text files created from those streams for long-term storage and future use.

Lab 6: Basic Spark Streaming Transformations (Python)

About This Lab

Objective:
Learn to use basic Spark Streaming transformations on data streams

File Locations:
`/root/spark/data/`

Successful Outcome:
Perform several basic transformations on streaming data

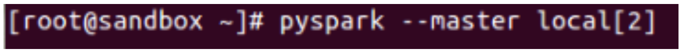
Lab Steps

Perform the following steps:

1. **Perform a Spark Streaming transformations using flatmap().**

   a. Open a terminal, connect to the sandbox cluster using SSH, and start a new instance of the REPL that is configured to use two CPU cores.

```
# ssh sandbox

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



```
[root@sandbox ~]# pyspark --master local[2]
```

   b. Create a data stream the performs the following operations:

   1. Sets the log level to `"ERROR"`

   2. Imports the `StreamingContext` class

   3. Creates an instance of that class named `sscFive` with a five-second time window

   4. Creates a socket text DStream named `inputDS` that listens to "sandbox" on port 9999

   5. Saves the DStream to text files in the `/user/root/test/stream/` directory.

   6. Creates a DStream named `flatMapDS` that uses `flatMap()` to break lines into individual elements separated by spaces

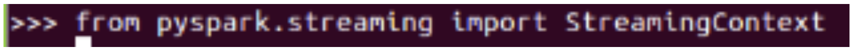
   7. Prints the contents of `flatMapDS` to the screen

8. Starts the application

```
>>> sc.setLogLevel("ERROR")
```

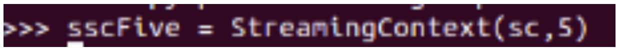
```
>>> sc.setLogLevel("ERROR")
```

```
>>> from pyspark.streaming import StreamingContext
```



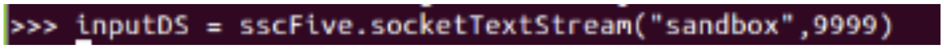
```
>>> from pyspark.streaming import StreamingContext
```

```
>>> sscFive = StreamingContext(sc, 5)
```



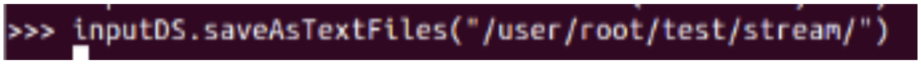
```
>>> sscFive = StreamingContext(sc,5)
```

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```



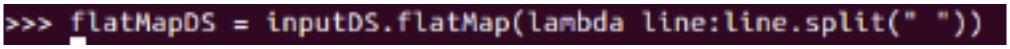
```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```

```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
```



```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
```

```
>>> flatMapDS = inputDS.flatMap(lambda line: line.split(" "))
```

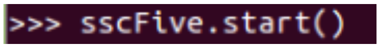


```
>>> flatMapDS = inputDS.flatMap(lambda line:line.split(" "))
```

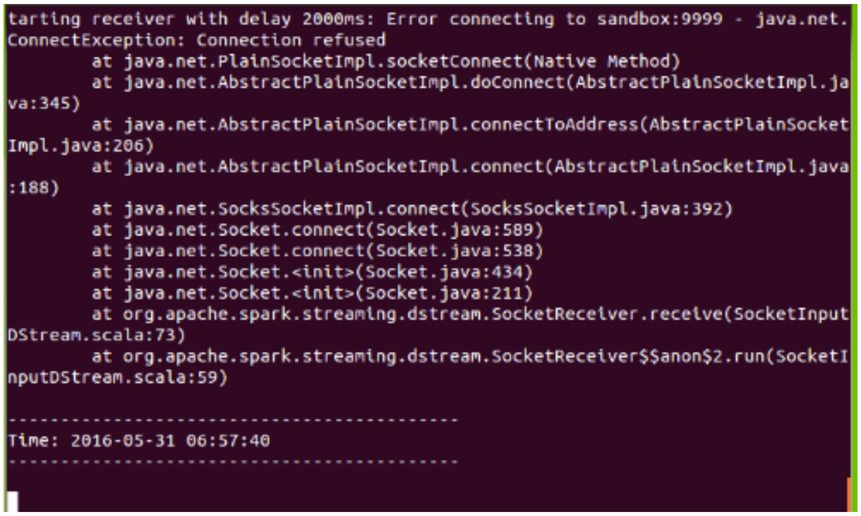
```
>>> flatMapDS.pprint()
```

```
>>> flatmapDS.pprint()
```

```
>>> sscFive.start()
```



```
>>> sscFive.start()
```



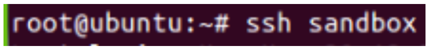
```
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net.
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.ja
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocket
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInput
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(SocketI
nputDStream.scala:59)

-------------------------------------------
Time: 2016-05-31 06:57:40
-------------------------------------------
```

NOTE:

You will see an error when it starts because it is waiting for an input connection.

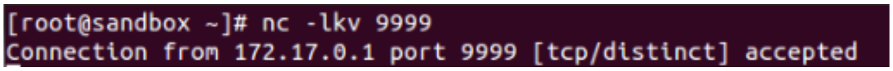
c. Open a new terminal window, connect to the sandbox cluster, and connect to port 9999 using the netcat utility. Make sure both terminal windows are visible on-screen.

```
# ssh sandbox
```



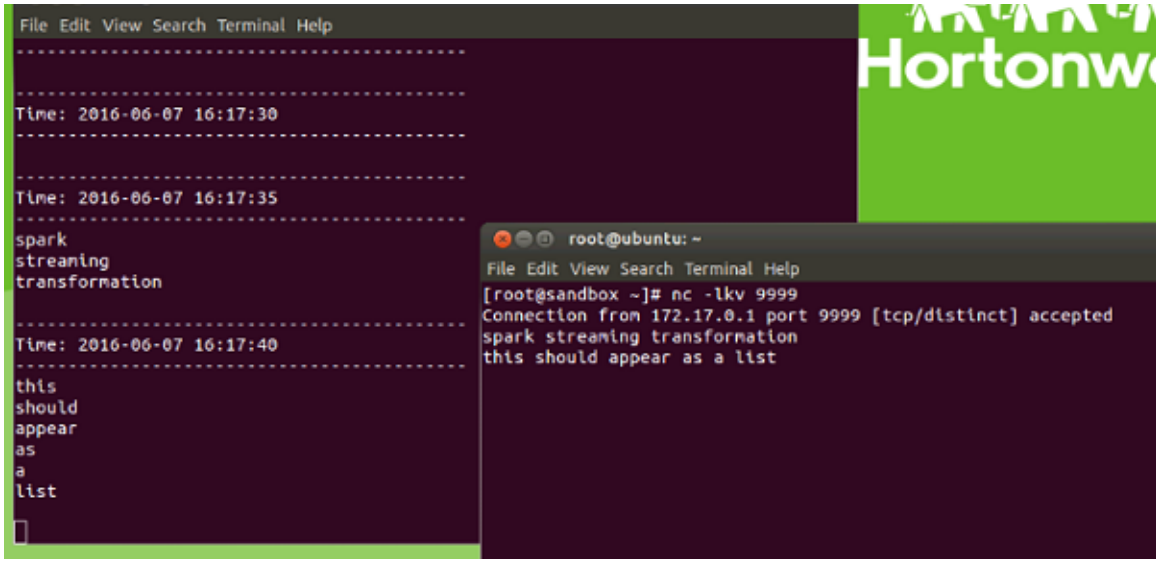
```
root@ubuntu:~# ssh sandbox
```

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

d. In the netcat terminal, start typing words separated by spaces. Hit the Enter key occasionally to submit them to the stream. Observe how the words appear in the streaming window.


File Edit View Search Terminal Help
Hortonw
root@ubuntu: ~
File Edit View Search Terminal Help

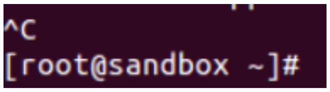
```
-------------------------------------------

-------------------------------------------
Time: 2016-06-07 16:17:30
-------------------------------------------

-------------------------------------------
Time: 2016-06-07 16:17:35
-------------------------------------------
spark
streaming
transformation

-------------------------------------------
Time: 2016-06-07 16:17:40
-------------------------------------------
this
should
appear
as
a
list
```

```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
spark streaming transformation
this should appear as a list
```

e. In the streaming window, stop the stream and exit the REPL.

```
sc.stop()
exit()
```

f. In the netcat window, exit the socket by entering Ctrl + C (or CMD + C if using a Mac) on your keyboard.

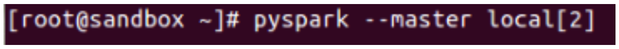


```
^C
[root@sandbox ~]#
```

2. **Perform a Spark Streaming word count transformations using reduceByKey().**

   a. In the streaming window, start a new instance of the REPL that is configured to use two CPU cores.

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



```
[root@sandbox ~]# pyspark --master local[2]
```

   b. Create a data stream the performs the following operations:

      1. Sets the log level to "ERROR"

      2. Imports the StreamingContext class

      3. Creates an instance of that class named sscFive with a five-second time window

      4. Creates a socket text DStream named inputDS that listens to "sandbox" on port 9999

      5. Saves the DStream to text files in the /user/root/test/stream/ directory.

      6. Creates a DStream named wc that uses flatMap(), map(), and reduceByKey() to count the number of times a word appears in a stream

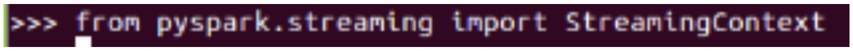
      7. Prints the contents of wc to the screen

      8. Starts the application

```
>>> sc.setLogLevel("ERROR")
```

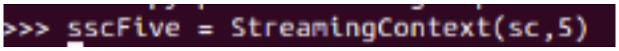
```
>>> sc.setLogLevel("ERROR")
```

```
>>> from pyspark.streaming import StreamingContext
```



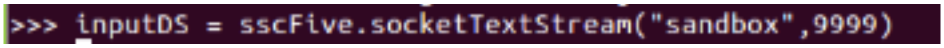
```
>>> from pyspark.streaming import StreamingContext
```

```
>>> sscFive = StreamingContext(sc, 5)
```



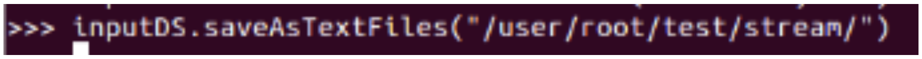
```
>>> sscFive = StreamingContext(sc,5)
```

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```

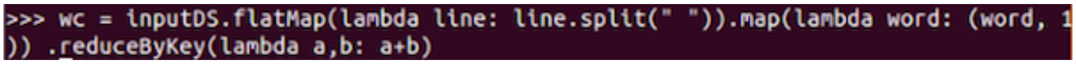


```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```

```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
```

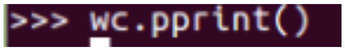


```
>>> inputDS.saveAsTextFiles("/user/root/test/stream/")
>>> wc = inputDS.flatMap(lambda line: line.split(" ")).map(lambda  word:
(word,1)).reduceByKey(lambda a,b: a+b)
```



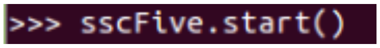
```
>>> wc = inputDS.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1
)) .reduceByKey(lambda a,b: a+b)
```

```
>>> wc.pprint()
```

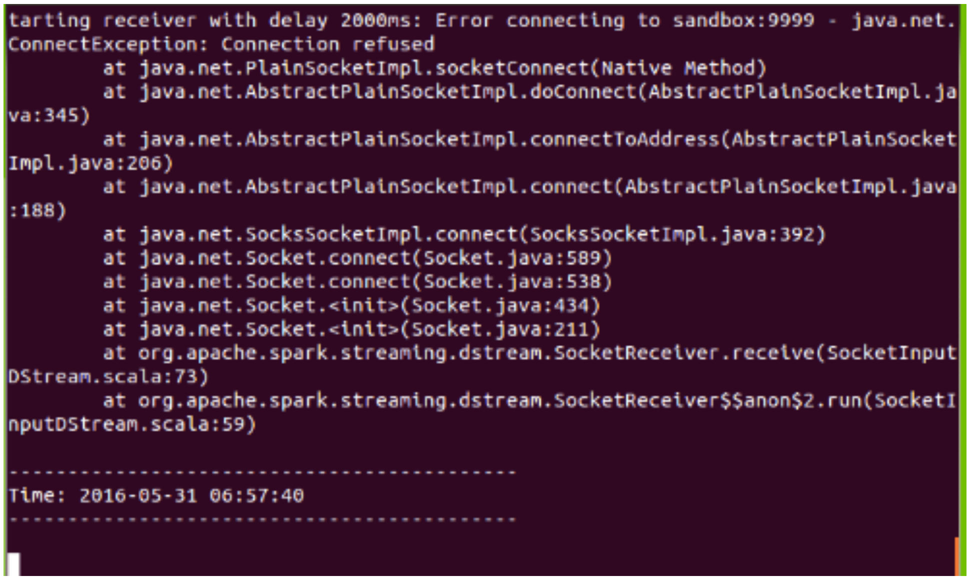


```
>>> wc.pprint()
```

```
>>> sscFive.start()
```



```
>>> sscFive.start()
```



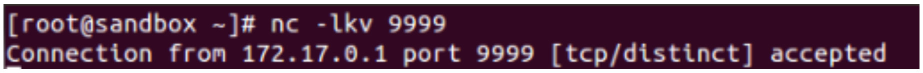
```
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net.
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.ja
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocket
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInput
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(SocketI
nputDStream.scala:59)

-------------------------------------------
Time: 2016-05-31 06:57:40
-------------------------------------------
```

NOTE:

You will see an error when it starts because it is waiting for an input connection.

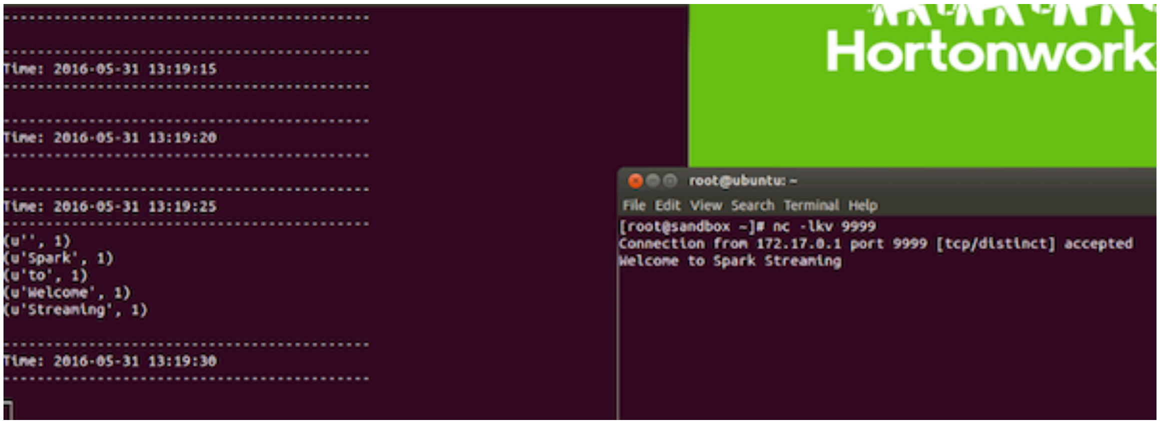
c. In the netcat window from the previous lab section, reconnect to port 9999 using the netcat utility. Make sure both terminal windows are visible on-screen.

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

d. In the netcat terminal, start typing words separated by spaces, making sure to repeat some of the words as you type. Hit the Enter key occasionally to submit them to the stream. Observe how the words appear in the streaming window.



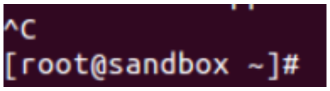
```
-------------------------------------------

-------------------------------------------
Time: 2016-05-31 13:19:15
-------------------------------------------

-------------------------------------------
Time: 2016-05-31 13:19:20
-------------------------------------------

-------------------------------------------
Time: 2016-05-31 13:19:25
-------------------------------------------
(u'', 1)
(u'Spark', 1)
(u'to', 1)
(u'Welcome', 1)
(u'Streaming', 1)

-------------------------------------------
Time: 2016-05-31 13:19:30
-------------------------------------------
```

```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
Welcome to Spark Streaming
```

e. In the streaming window, stop the stream and exit the REPL.

```
sc.stop()
exit()
```

f. In the netcat window, exit the socket by entering Ctrl + C (or CMD + C if using a Mac) on your keyboard.

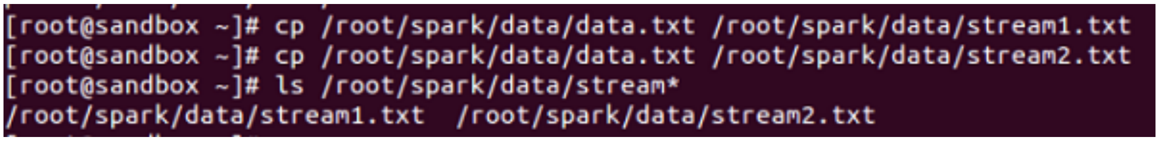


```
^C
[root@sandbox ~]#
```

3. **Perform a Spark Streaming transformations using union().**

a. In the streaming window, create two copies of the /root/spark/data/data.txt file named stream1.txt and stream2.txt and confirm the operation was successful.

```
# cp /root/spark/data/data.txt /root/spark/data/stream1.txt

# cp /root/spark/data/data.txt /root/spark/data/stream2.txt

# ls /root/spark/data/stream*
```

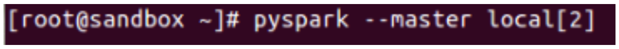


```
[root@sandbox ~]# cp /root/spark/data/data.txt /root/spark/data/stream1.txt
[root@sandbox ~]# cp /root/spark/data/data.txt /root/spark/data/stream2.txt
[root@sandbox ~]# ls /root/spark/data/stream*
/root/spark/data/stream1.txt  /root/spark/data/stream2.txt
```

You can view the contents of the file if you want. As a reminder, these files contain a single line of text: “This is a test file”

b. In the streaming window, start a new instance of the REPL that is once again configured to use two CPU cores.

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



```
[root@sandbox ~]# pyspark --master local[2]
```

c. Create a data stream the performs the following operations:

1. Sets the log level to “ERROR”

2. Imports the `StreamingContext` class

3. Creates an instance of that class named `sscFive` with a five-second time window

4. Creates two text file DStreams named `inputDS1` and `inputDS2` that both listen to the `/user/root/test/` directory on HDFS.

5. Creates a DStream named `combined` that uses `union()` to combine the two streams into a single DStream

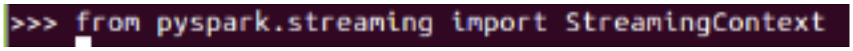
6. Prints the contents of `combined` to the screen

7. Starts the application

```
>>> sc.setLogLevel("ERROR")
```

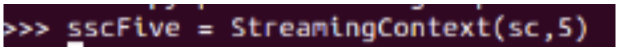
```
>>> sc.setLogLevel("ERROR")
```

```
>>> from pyspark.streaming import StreamingContext
```



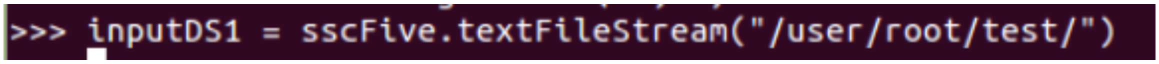
```
>>> from pyspark.streaming import StreamingContext
```

```
>>> sscFive = StreamingContext(sc, 5)
```



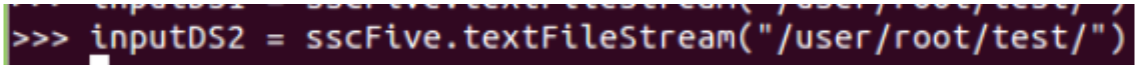
```
>>> sscFive = StreamingContext(sc,5)
```

```
>>> inputDS1 = sscFive.textFileStream(“/user/root/test/”)
```

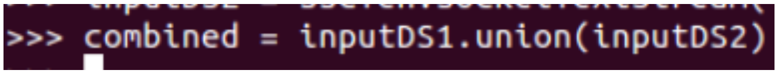


```
>>> inputDS1 = sscFive.textFileStream("/user/root/test/")
```

```
>>> inputDS2 = sscFive.textFileStream(“/user/root/test/”)
```

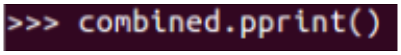


```
>>> inputDS2 = sscFive.textFileStream("/user/root/test/")
>>> combined = inputDS1.union(inputDS2)
```



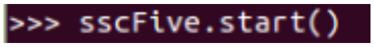
```
>>> combined = inputDS1.union(inputDS2)
```

```
>>> combined.pprint()
```



```
>>> combined.pprint()
```

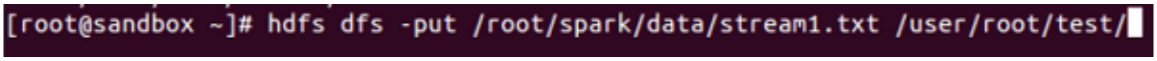
```
>>> sscFive.start()
```



```
>>> sscFive.start()
```

d. Go to the netcat terminal window (which we'll refer to now as the input1 window) from the previous lab section and type the command to upload the small_blocks.txt file from the local `/root/spark/data/` directory to the `/user/root/test/` directory on HDFS, but **DO NOT PRESS THE ENTER KEY**.

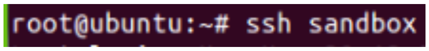
```
# hdfs dfs -put /root/spark/data/stream1.txt /user/root/test/
```



```
[root@sandbox ~]# hdfs dfs -put /root/spark/data/stream1.txt /user/root/test/
```

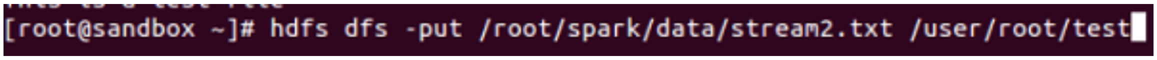
e. Open a third terminal window (we'll refer to this as the input2 window), connect to the sandbox cluster, and type the same command as in the step above, but once again **DO NOT PRESS THE ENTER KEY**. Make sure both terminal windows are visible on-screen.

```
# ssh sandbox
```



```
root@ubuntu:~# ssh sandbox
```

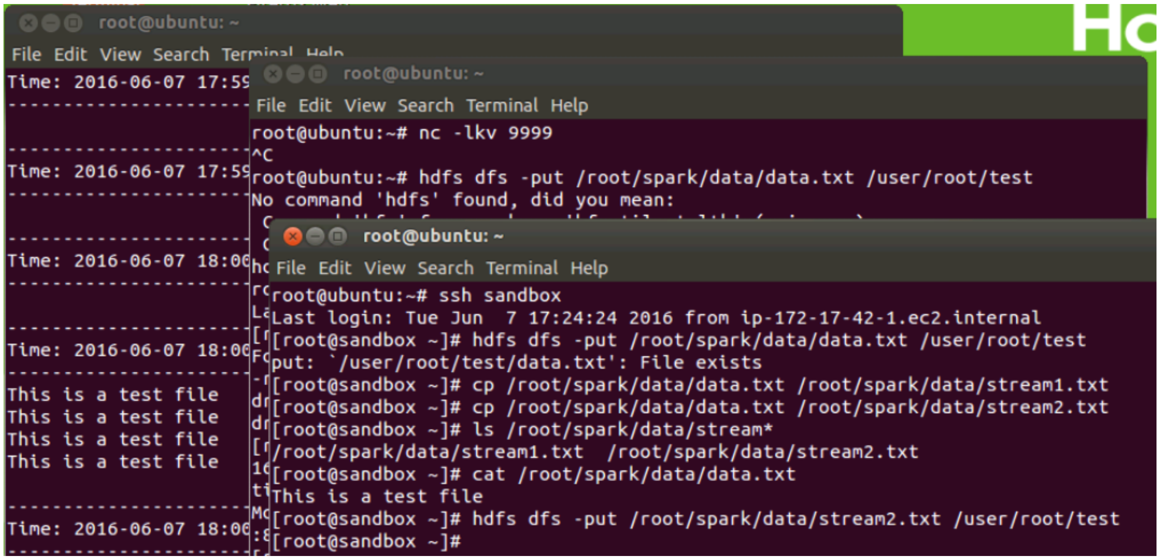
```
# ssh sandbox
```



```
[root@sandbox ~]# hdfs dfs -put /root/spark/data/stream2.txt /user/root/test
```

f. Wait for a screen refresh in the streaming window, then immediately go to the input1 and input2 windows and press the Enter key.

Assuming you perform both actions within a 5-second collection window, the streaming window should display the contents of files as a combined data stream, as displayed in the screenshot below. The content of the text files (which in our case should be the same line of text) should each print multiple times because both streams were monitoring the same HDFS directory.



```
Time: 2016-06-07 17:59
---------------------

---------------------
Time: 2016-06-07 17:59
---------------------

---------------------
Time: 2016-06-07 18:00
---------------------

---------------------
Time: 2016-06-07 18:00
---------------------
This is a test file
This is a test file
This is a test file
This is a test file

---------------------
Time: 2016-06-07 18:00
---------------------
```

```
root@ubuntu:~# nc -lkv 9999
^C
root@ubuntu:~# hdfs dfs -put /root/spark/data/data.txt /user/root/test
No command 'hdfs' found, did you mean:
```

```
root@ubuntu:~# ssh sandbox
Last login: Tue Jun  7 17:24:24 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]# hdfs dfs -put /root/spark/data/data.txt /user/root/test
put: `/user/root/test/data.txt': File exists
[root@sandbox ~]# cp /root/spark/data/data.txt /root/spark/data/stream1.txt
[root@sandbox ~]# cp /root/spark/data/data.txt /root/spark/data/stream2.txt
[root@sandbox ~]# ls /root/spark/data/stream*
/root/spark/data/stream1.txt  /root/spark/data/stream2.txt
[root@sandbox ~]# cat /root/spark/data/data.txt
This is a test file
[root@sandbox ~]# hdfs dfs -put /root/spark/data/stream2.txt /user/root/test
[root@sandbox ~]#
```

If your timing is off the first time, simply try again with a couple of additional copies that have unique file names like `streaming3.txt` and `streaming4.txt`.

g. In the streaming window, stop the stream and exit the REPL.

```
sc.stop()
exit()
```

Result

You have successfully used several basic transformations on DStreams.

Lab 7: Spark Streaming Window Transformations (Python)

About This Lab

Objective:
Use Spark Streaming Window Transformations

File Locations:
NA

Successful Outcome:
Perform several Spark Streaming Window Transformations

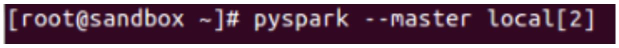
Lab Steps

Perform the following steps:

1. Create a streaming window using a TCP socket.

   a. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

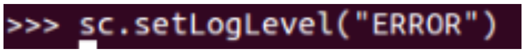
```
# pyspark --master local[2]
```



```
[root@sandbox ~]# pyspark --master local[2]
```

   b. Set the log level to `ERROR` to avoid screen clutter while running the streaming application.

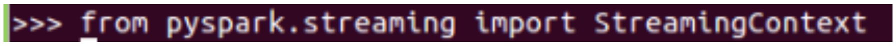
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

   c. Import the streaming library.

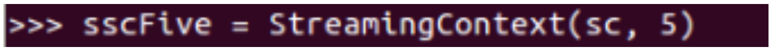
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

   d. Create a streaming context with a five-second batch duration.

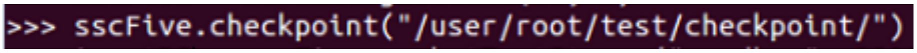
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

e. Set the checkpoint directory.

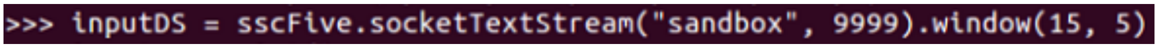
```
>>> sscFive.checkpoint(“/user/root/test/checkpoint/”)
```



```
>>> sscFive.checkpoint("/user/root/test/checkpoint/")
```

f. Create a DStream using `socketTestStream()` to the system named “sandbox” on port 9999 and set it up as a window function with a 15-second collection period (window length) and a 5-second collection interval.

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999).window(15, 5)
```



```
>>> inputDS = sscFive.socketTextStream("sandbox", 9999).window(15, 5)
```

g. Print out the output to the terminal window.

```
>>> inputDS.pprint()
```

```
>>> inputDS.pprint()
```

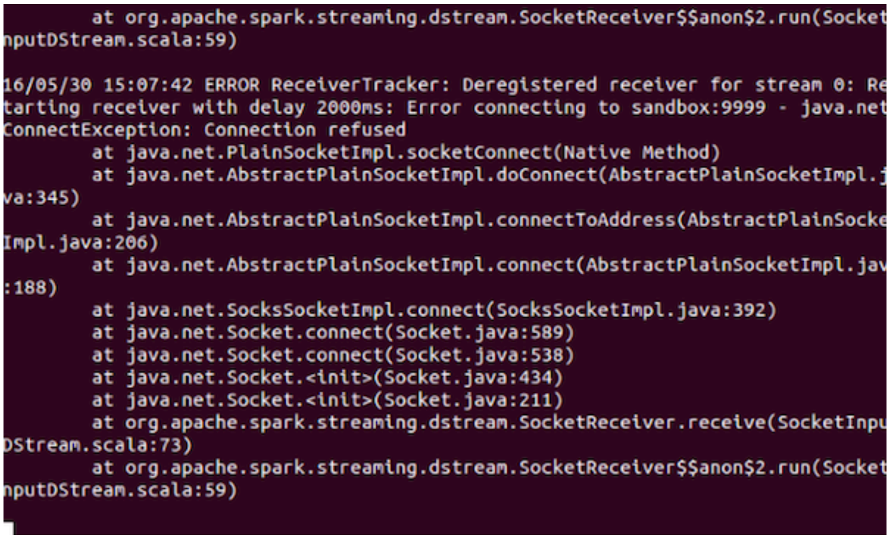
h. Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.

```
>>> sscFive.start()
```

```
>>> sscFive.start()
```

NOTE:

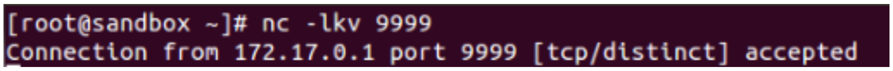
An error will appear when the application starts because the application is waiting for an input connection.



```
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)

16/05/30 15:07:42 ERROR ReceiverTracker: Deregistered receiver for stream 0: Re
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.j
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocke
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.jav
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInpu
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)
```

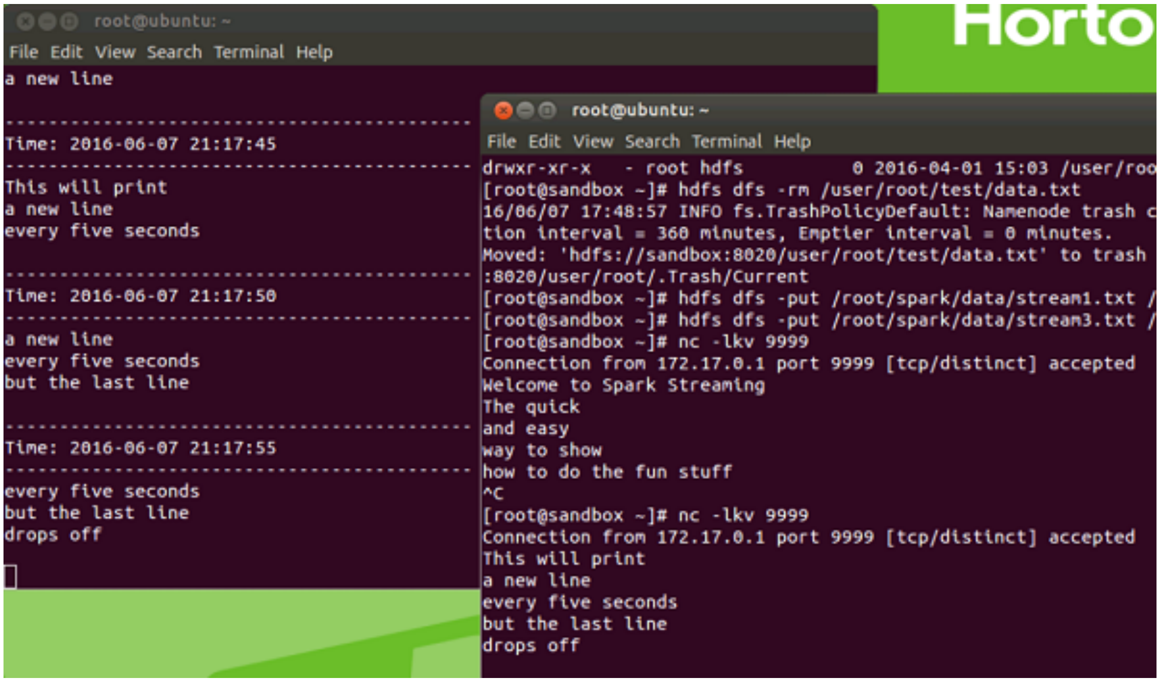
i. In the second terminal window use the netcat utility to create a connection to port 9999.

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

j. Start typing words separated by spaces, hit ***Enter*** occasionally to submit them. Observe what happens in the streaming terminal window a few seconds after hitting ***Enter***.



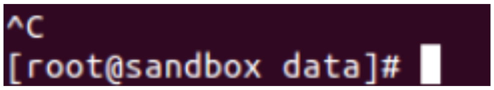
root@ubuntu: ~
File Edit View Search Terminal Help
Horto

```
a new line

-------------------------------------------
Time: 2016-06-07 21:17:45
-------------------------------------------
This will print
a new line
every five seconds

-------------------------------------------
Time: 2016-06-07 21:17:50
-------------------------------------------
a new line
every five seconds
but the last line

-------------------------------------------
Time: 2016-06-07 21:17:55
-------------------------------------------
every five seconds
but the last line
drops off
```

root@ubuntu: ~
File Edit View Search Terminal Help

```
drwxr-xr-x   - root hdfs          0 2016-04-01 15:03 /user/roo
[root@sandbox ~]# hdfs dfs -rm /user/root/test/data.txt
16/06/07 17:48:57 INFO fs.TrashPolicyDefault: Namenode trash c
tion interval = 360 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://sandbox:8020/user/root/test/data.txt' to trash
:8020/user/root/.Trash/Current
[root@sandbox ~]# hdfs dfs -put /root/spark/data/stream1.txt /
[root@sandbox ~]# hdfs dfs -put /root/spark/data/stream3.txt /
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
Welcome to Spark Streaming
The quick
and easy
way to show
how to do the fun stuff
^C
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
This will print
a new line
every five seconds
but the last line
drops off
```

k. Once you observe data being streamed on-screen in the first terminal window, use ***Ctrl + C*** (or ***Cmd + C*** if using a Mac) to exit netcat in the second terminal window.



```
^C
[root@sandbox data]#
```

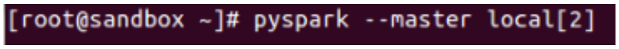
l. In the first terminal window, stop the stream and exit the REPL. If the stream refreshes while you are typing, that will not affect the input. Simply continue to type the command and press **Enter**.

```
sc.stop()
exit()
```

2. Create a streaming window that counts words in a DStream using a TCP socket.

   a. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

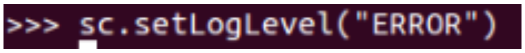
```
# pyspark --master local[2]
```



```
[root@sandbox ~]# pyspark --master local[2]
```

   b. Set the log level to `ERROR` to avoid screen clutter while running the streaming application.

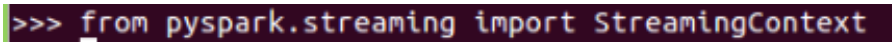
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

   c. Import the streaming library.

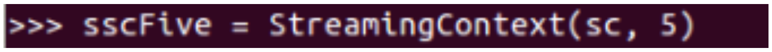
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

   d. Create a streaming context with a five-second batch duration.

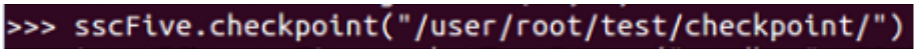
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

   e. Set the checkpoint directory.

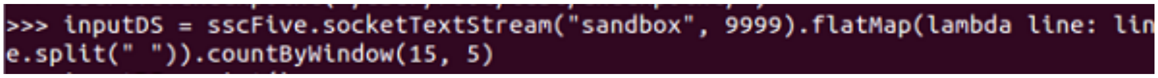
```
>>> sscFive.checkpoint(“/user/root/test/checkpoint/”)
```



```
>>> sscFive.checkpoint("/user/root/test/checkpoint/")
```

   f. Create a DStream using `socketTestStream()` to the system named “sandbox” on port 9999. Convert the lines of text it will accept into individual elements using `flatMap()`. Then use `countByWindow()` with a 15-second collection period (window length) and a 5-second collection interval to count the number of words typed over the last 15 seconds as a running total.

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999).flatMap(lambda line:
line.split(“ “)).countByWindow(15, 5)
```



```
>>> inputDS = sscFive.socketTextStream("sandbox", 9999).flatMap(lambda line: lin
e.split(" ")).countByWindow(15, 5)
```

QUESTION:

What do you think would happen if the `flatMap` function were removed from the line of code above?

g. Print out the output to the terminal window.

```
>>> inputDS.pprint()
```

```
>>> inputDS.pprint()
```

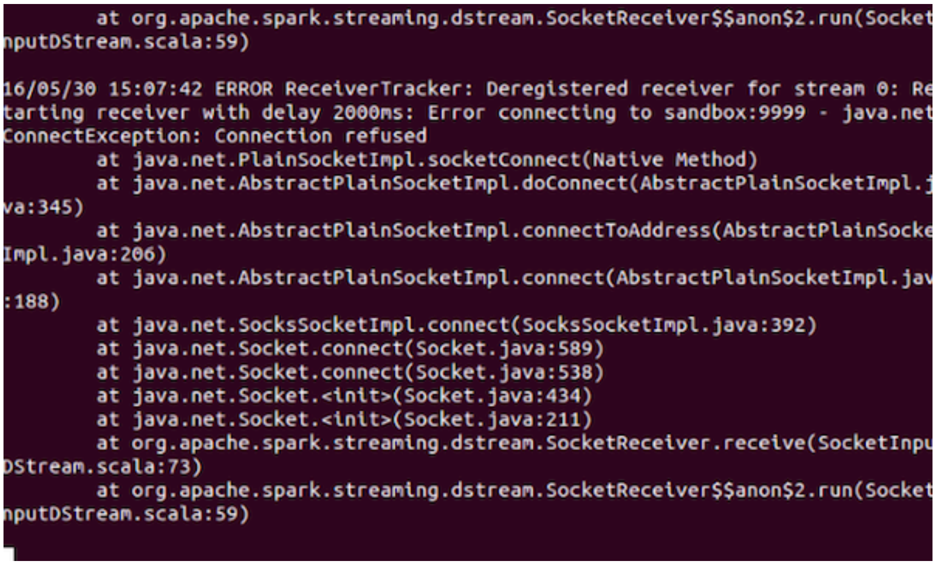
h. Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.

```
>>> sscFive.start()
```

```
>>> sscFive.start()
```

NOTE:

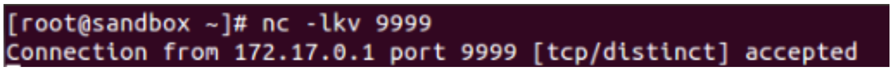
An error will appear when the application starts because the application is waiting for an input connection.



```
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)

16/05/30 15:07:42 ERROR ReceiverTracker: Deregistered receiver for stream 0: Re
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.j
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocke
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.jav
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInpu
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)
```

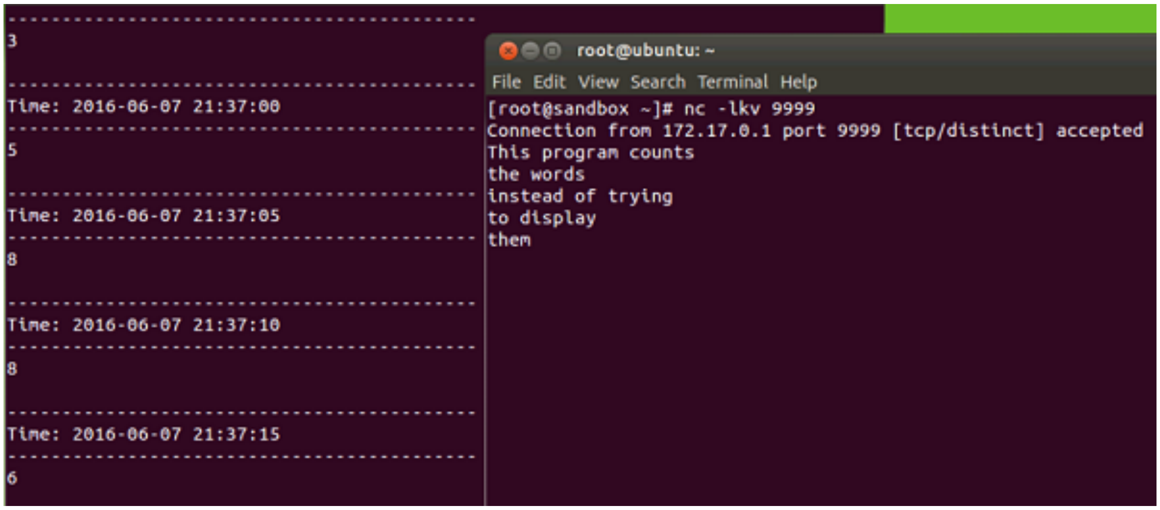
i. In the second terminal window use the netcat utility to create a connection to port 9999.

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

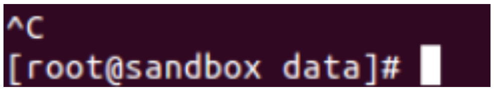
j. Start typing words separated by space, hit ***Enter*** occasionally to submit them. Observe what happens in the streaming terminal window a few seconds after hitting ***Enter***.



```
-------------------------------------------
3

-------------------------------------------
Time: 2016-06-07 21:37:00
-------------------------------------------
5

-------------------------------------------
Time: 2016-06-07 21:37:05
-------------------------------------------
8

-------------------------------------------
Time: 2016-06-07 21:37:10
-------------------------------------------
8

-------------------------------------------
Time: 2016-06-07 21:37:15
-------------------------------------------
6
```

```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
This program counts
the words
instead of trying
to display
them
```

k. Once you observe data being streamed on-screen in the first terminal window, use ***Ctrl + C*** (or ***Cmd + C*** if using a Mac) to exit netcat in the second terminal window.



```
^C
[root@sandbox data]#
```

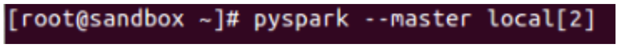
l. In the first terminal window, stop the stream and exit the REPL. If the stream refreshes while you are typing, that will not affect the input. Simply continue to type the command and press **Enter**.

```
sc.stop()
exit()
```

3 . Create a streaming window that counts instances of words in a DStream using a TCP socket.

a. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

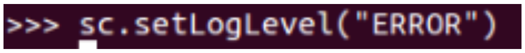
```
# pyspark --master local[2]
```



```
[root@sandbox ~]# pyspark --master local[2]
```

b. Set the log level to `ERROR` to avoid screen clutter while running the streaming application.

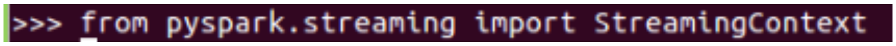
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

c. Import the streaming library.

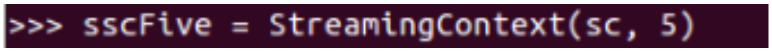
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

d. Create a streaming context with a five-second batch duration.

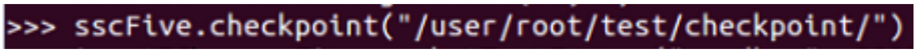
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

e. Set the checkpoint directory.

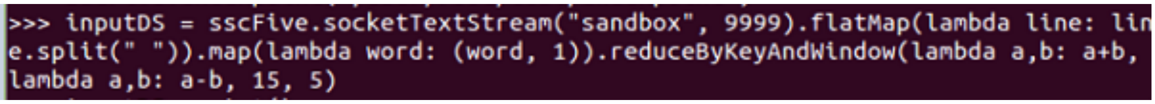
```
>>> sscFive.checkpoint(“/user/root/test/checkpoint/”)
```



```
>>> sscFive.checkpoint("/user/root/test/checkpoint/")
```

f. Create a DStream using `socketTestStream()` to the system named “sandbox” on port 9999. Convert the lines of text it will accept into individual elements using `flatMap()`. Then use `map()` to create key-value pairs out of the individual elements. Finally, use `reduceByKeyAndWindow()` with a 15-second collection period (window length) and a 5-second collection interval to count the number of times a word has been typed over the last 15 seconds as a running total.

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999).flatMap(lambda line:
line.split(“ “)).map(lambda word: (word, 1)). reduceByKeyAndWindow(lambda a,b:
a+b, lambda a,b: a-b, 15, 5)
```



```
>>> inputDS = sscFive.socketTextStream("sandbox", 9999).flatMap(lambda line: lin
e.split(" ")).map(lambda word: (word, 1)).reduceByKeyAndWindow(lambda a,b: a+b,
lambda a,b: a-b, 15, 5)
```

g. Print out the output to the terminal window.

```
>>> inputDS.pprint()
```

```
>>> inputDS.pprint()
```

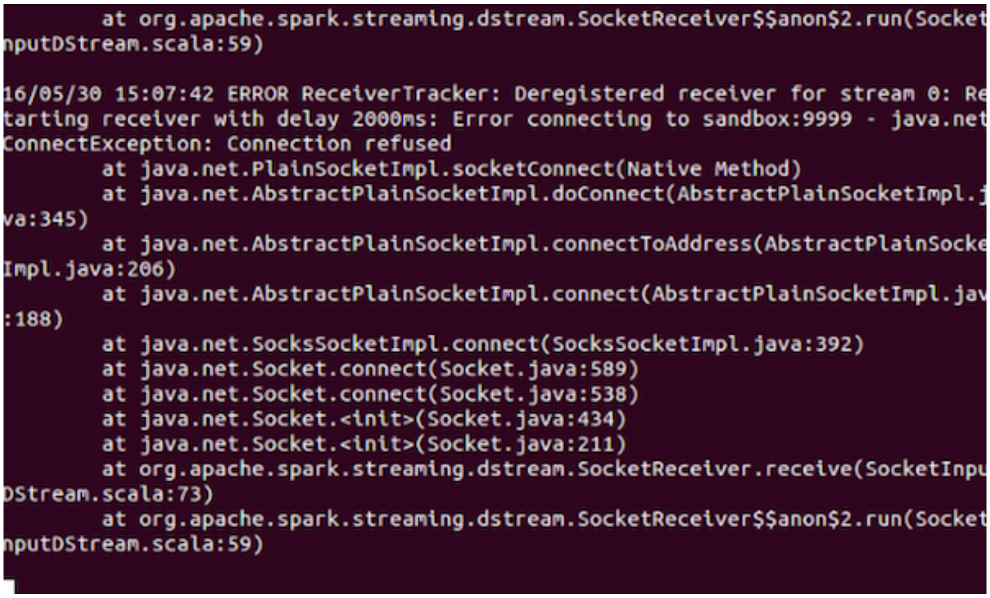
h. Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.

```
>>> sscFive.start()
```

```
>>> sscFive.start()
```

NOTE:

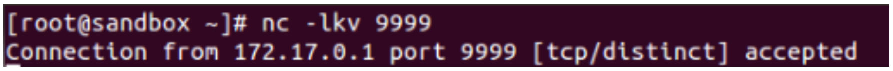
An error will appear when the application starts because the application is waiting for an input connection.



```
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)

16/05/30 15:07:42 ERROR ReceiverTracker: Deregistered receiver for stream 0: Re
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.j
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocke
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.jav
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInpu
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)
```

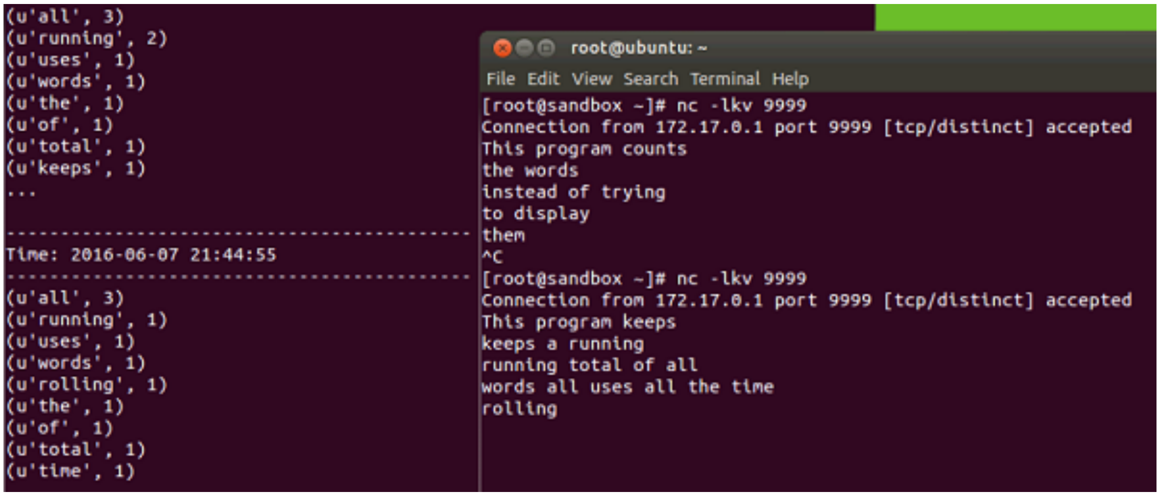
i. In the second terminal window use the netcat utility to create a connection to port 9999.

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

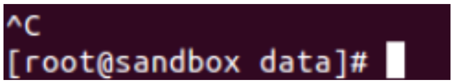
j. Start typing words separated by space, hit ***Enter*** occasionally to submit them. Make sure to repeat words every so often between lines. Observe what happens in the streaming terminal window a few seconds after hitting ***Enter***.



```
(u'all', 3)
(u'running', 2)
(u'uses', 1)
(u'words', 1)
(u'the', 1)
(u'of', 1)
(u'total', 1)
(u'keeps', 1)
...

-------------------------------------------
Time: 2016-06-07 21:44:55
-------------------------------------------
(u'all', 3)
(u'running', 1)
(u'uses', 1)
(u'words', 1)
(u'rolling', 1)
(u'the', 1)
(u'of', 1)
(u'total', 1)
(u'time', 1)
```

```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
This program counts
the words
instead of trying
to display
them
^C
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
This program keeps
keeps a running
running total of all
words all uses all the time
rolling
```

k. Once you observe data being streamed on-screen in the first terminal window, use ***Ctrl + C*** (or ***Cmd + C*** if using a Mac) to exit netcat in the second terminal window.



```
^C
[root@sandbox data]#
```

l. In the first terminal window, stop the stream and exit the REPL. If the stream refreshes while you are typing, that will not affect the input. Simply continue to type the command and press **Enter**.

```
sc.stop()
exit()
```

Result

You have successfully performed various Spark Streaming Window Transformations.

Lab 8: Create and Save DataFrames and Tables (Python)

About This Lab

Objective:
Create and save DataFrames and tables

Files Locations:
NA

Successful Outcome:
Use various methods to create and save DataFrames and tables

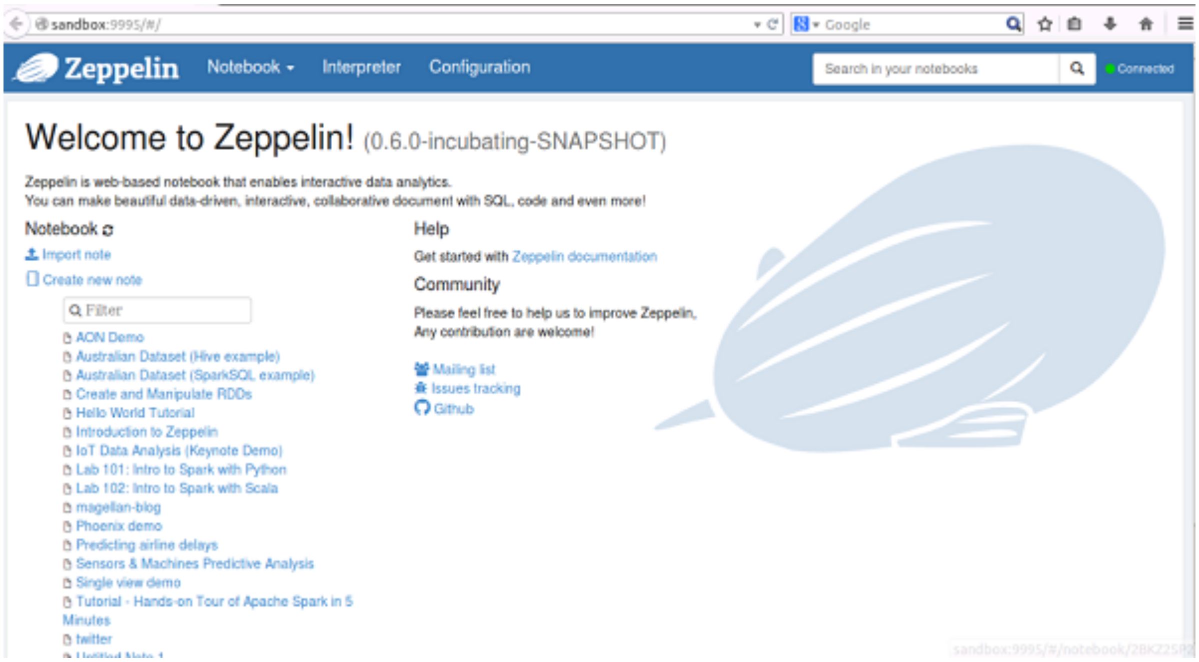
Lab Steps

Perform the following steps:

1. **Create and save DataFrames and tables.**

    a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.

    http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Create and Manipulate RDDs
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1

Help

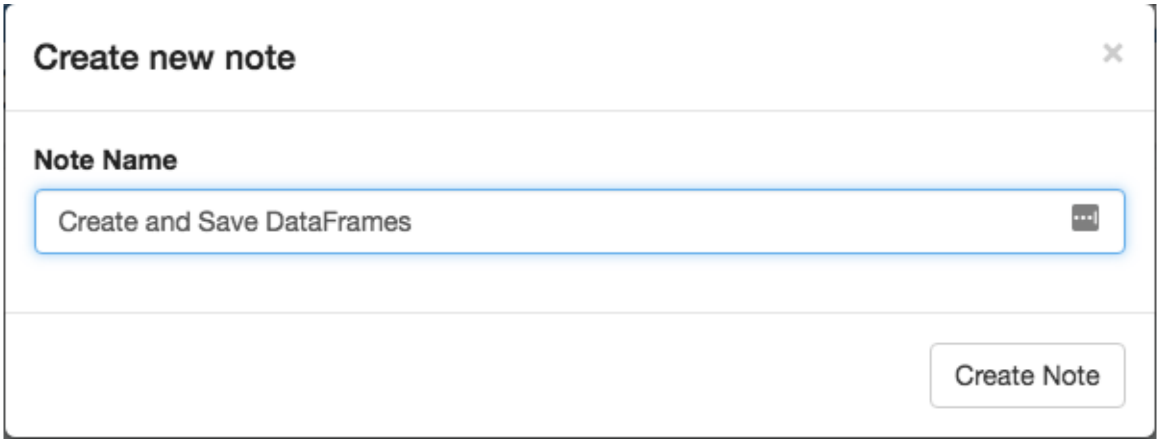
Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

b. Click Create new note. Name this note Create and Save DataFrames.



Create new note

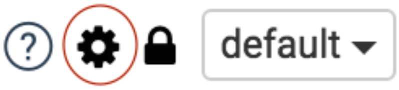
Note Name

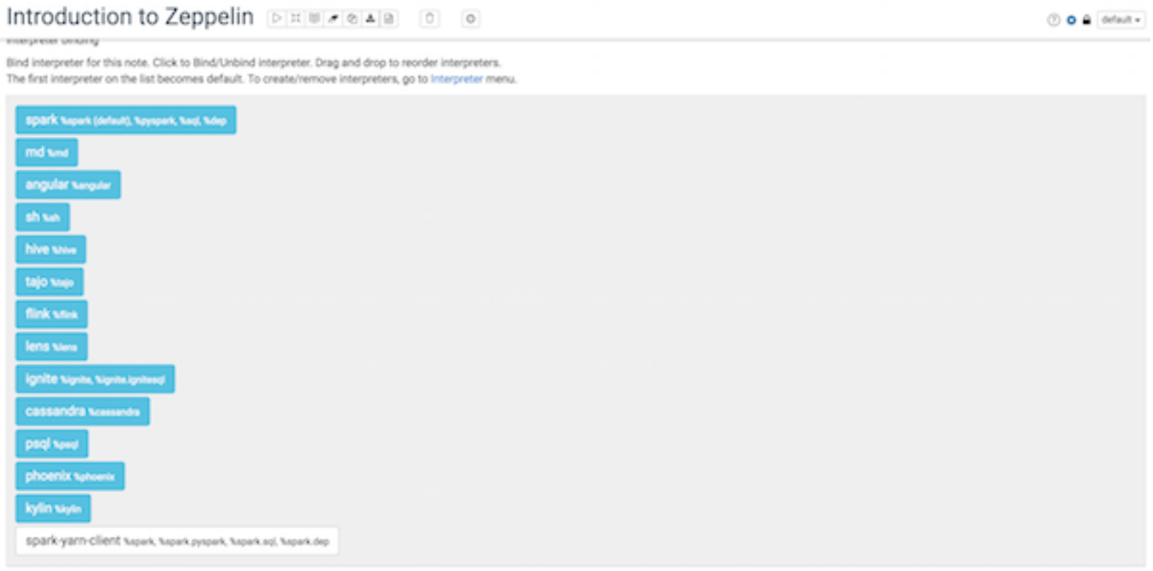
Create and Save DataFrames

NOTE:

Make sure to set the interpreter to spark-yarn-client as in previous labs.

c. At the top right click on the gear icon to change interpreter binding.





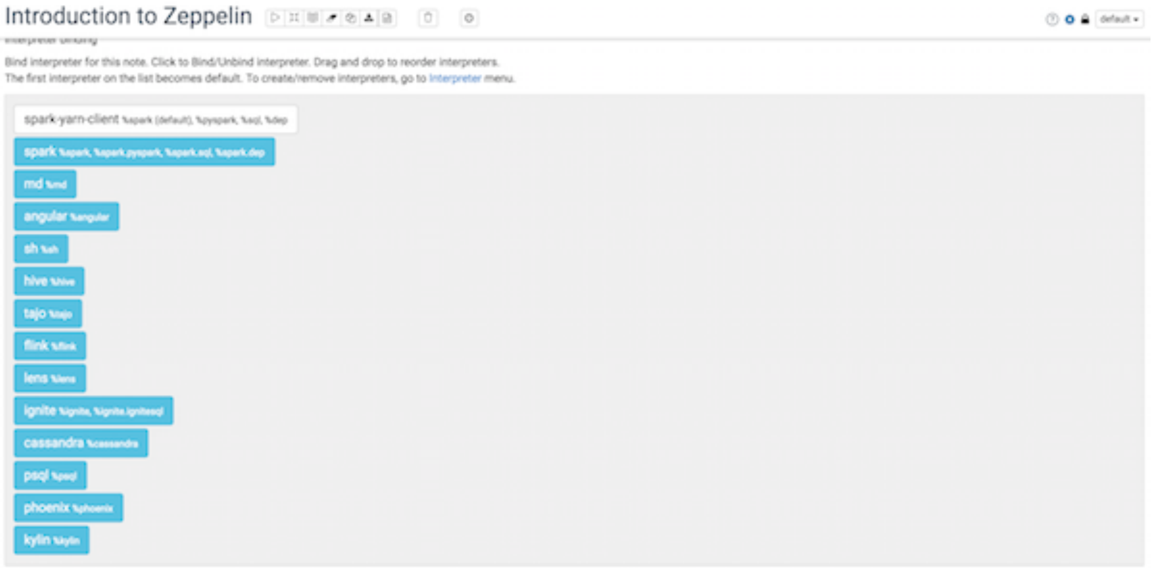
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

Drag the spark-yarn-client to the top and click save.



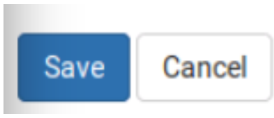
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark-yarn-client %spark (default), %pyspark, %sql, %dep
spark %spark, %spark.pyspark, %spark.sql, %spark.dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin

The first interpreter on the list becomes default.

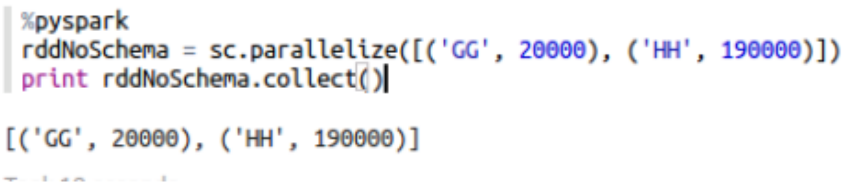


d. Create an RDD named `rddNoSchema` that consists of a comma-separated list of organized comma-separated lists, as specified below:

The first entry in each sub-list should be a two-letter code (`GG` and `HH`). The second entry in each sub-list should be numeric values of 20,000 and 190,000 respectively.

View the resulting RDD to confirm success.

```
%pyspark

rddNoSchema = sc.parallelize([(‘GG’, 20000), (‘HH’, 190000)])

print rddNoSchema.collect()
```

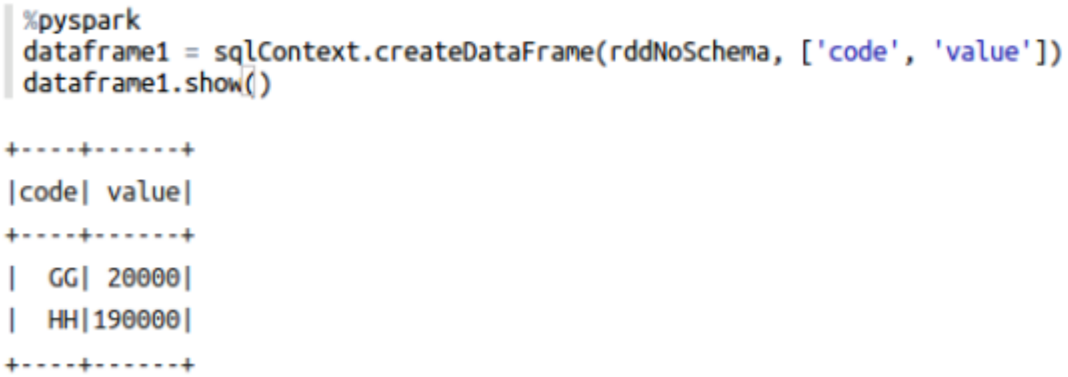


```
%pyspark
rddNoSchema = sc.parallelize([('GG', 20000), ('HH', 190000)])
print rddNoSchema.collect()
```

```
[('GG', 20000), ('HH', 190000)]
```

e. Use `createDataFrame()` to convert this RDD into a DataFrame named `dataframe1` and apply a schema where the first entry in each sub-list is assigned to the code category and the second entry in each sub-list is assigned to the value category. View the DataFrame to confirm success.

```
%pyspark

dataframe1 = sqlContext.createDataFrame(rddNoSchema, [‘code’, ‘value’])

dataframe1.show()
```



```
%pyspark
dataframe1 = sqlContext.createDataFrame(rddNoSchema, ['code', 'value'])
dataframe1.show()
```

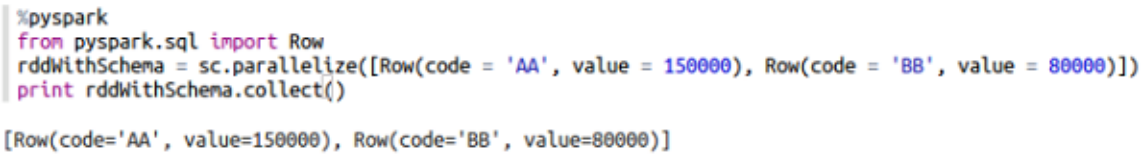
```
+----+------+
|code| value|
+----+------+
|  GG| 20000|
|  HH|190000|
+----+------+
```

f. Create an RDD named `rddWithSchema` that utilizes Row objects organized so that each element has a schema value.

The first entry in each Row should be a two-letter code (`AA` and `BB`) that are assigned a schema value of `code`. The second entry in each Row should be numeric values of 150,000 and 80,000 respectively that are assigned a schema value of `value`.

View the RDD to confirm success.

```
%pyspark

from pyspark.sql import Row

rddWithSchema = sc.parallelize([Row(code = ‘AA’, value = 150000), Row(code = 
‘BB’, value = 80000)])

print rddWithSchema.collect()
```

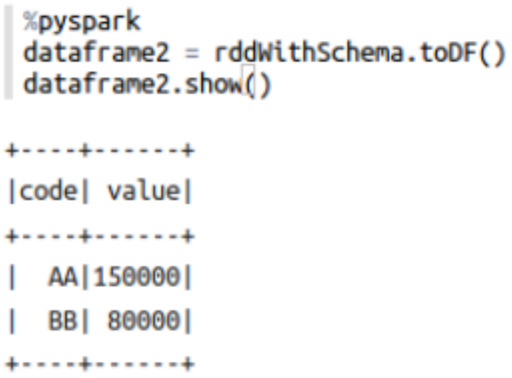


```
%pyspark
from pyspark.sql import Row
rddWithSchema = sc.parallelize([Row(code = 'AA', value = 150000), Row(code = 'BB', value = 80000)])
print rddWithSchema.collect()
```

```
[Row(code='AA', value=150000), Row(code='BB', value=80000)]
```

g. **Use `toDF()` to convert this RDD to a new DataFrame named `dataframe2`. View the DataFrame to confirm success.**

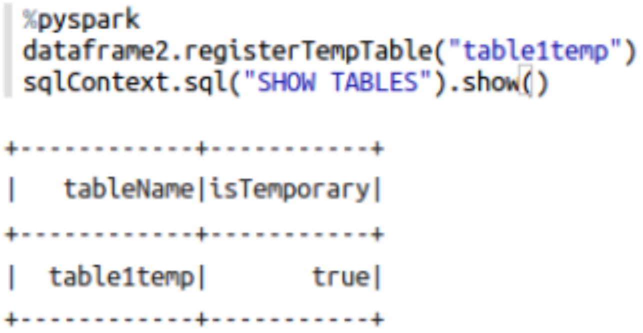
```
%pyspark

dataframe2 = rddWithSchema.toDF()

dataframe2.show()
```



```
%pyspark
dataframe2 = rddWithSchema.toDF()
dataframe2.show()

+----+------+
|code| value|
+----+------+
|  AA|150000|
|  BB| 80000|
+----+------+
```

h. **Register `dataframe2` as a temporary table named `table1temp`. Then issue a SQL command using the DataFrames API to show the tables visible to the context.**

```
%pyspark

dataframe2.registerTempTable(“table1temp”)

sqlContext.sql(“SHOW TABLES”).show()
```



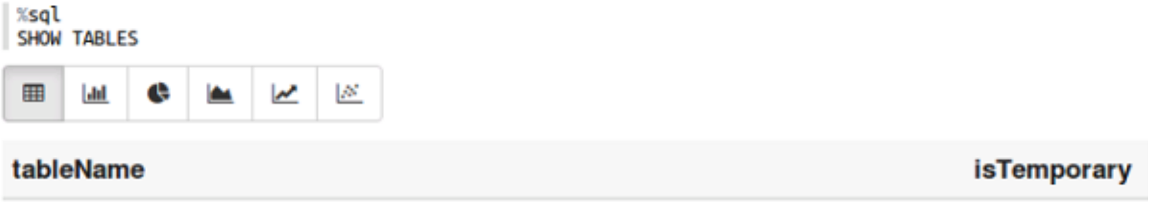
```
%pyspark
dataframe2.registerTempTable("table1temp")
sqlContext.sql("SHOW TABLES").show()

+-----------+-----------+
|  tableName|isTemporary|
+-----------+-----------+
| table1temp|       true|
+-----------+-----------+
```

i. In the next paragraph, issue a Spark SQL command to `SHOW TABLES`. Does `table1temp` show up? If so, why? If not, why not?

NOTE:

Your output may also contain tables created when you ran demos in previous labs.

```
%sql

SHOW TABLES
```



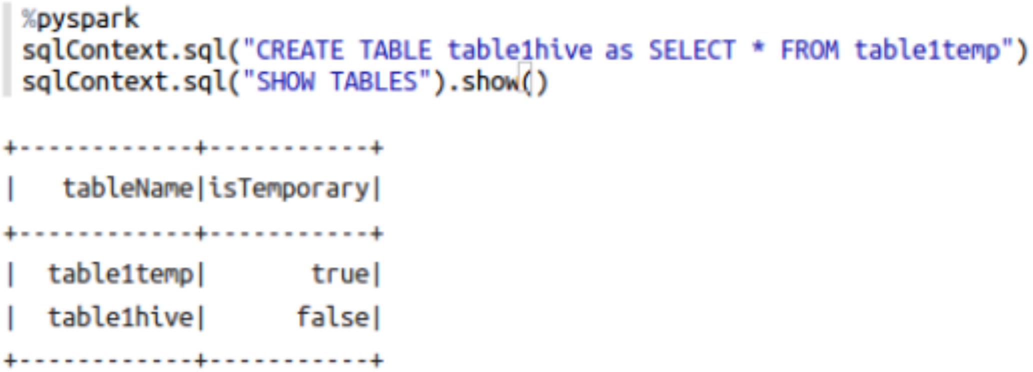
```
%sql
SHOW TABLES
```

tableName	isTemporary

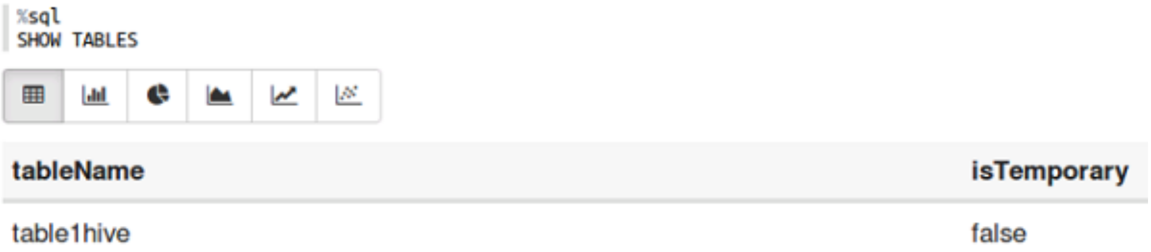
j. Issue a HiveQL `CREATE TABLE` command from within the DataFrames API and create a permanent version of `table1temp` named `table1hive`. Use `SHOW TABLES` both from the DataFrames API, and then in a new paragraph from Spark SQL, to confirm this table is visible across contexts.

```
%pyspark

sqlContext.sql("CREATE TABLE table1hive AS SELECT * FROM table1temp")

sqlContext.sql("SHOW TABLES").show()
```

```
%sql

SHOW TABLES
```



```
%pyspark
sqlContext.sql("CREATE TABLE table1hive as SELECT * FROM table1temp")
sqlContext.sql("SHOW TABLES").show()

+----------+-----------+
| tableName|isTemporary|
+----------+-----------+
|table1temp|       true|
|table1hive|      false|
+----------+-----------+
```

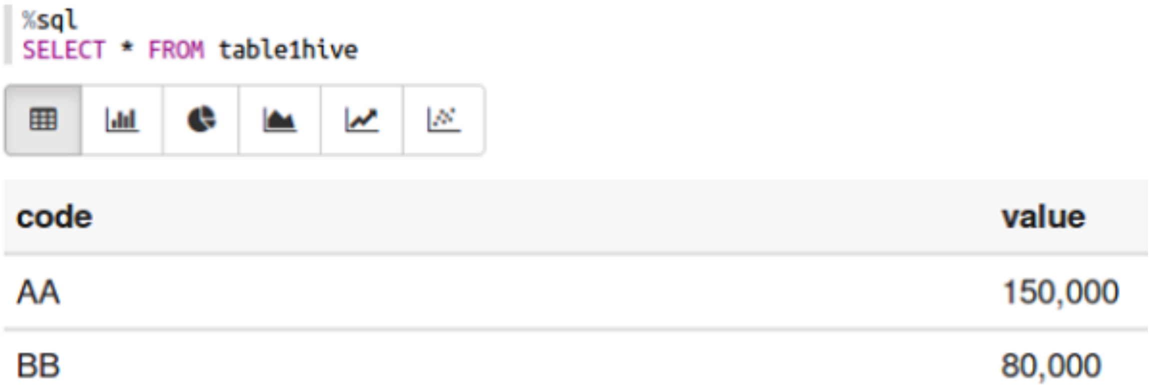


```
%sql
SHOW TABLES
```

tableName	isTemporary
table1hive	false

k. Use Spark SQL to view the contents of table1hive.

```
%sql

SELECT * FROM table1hive
```

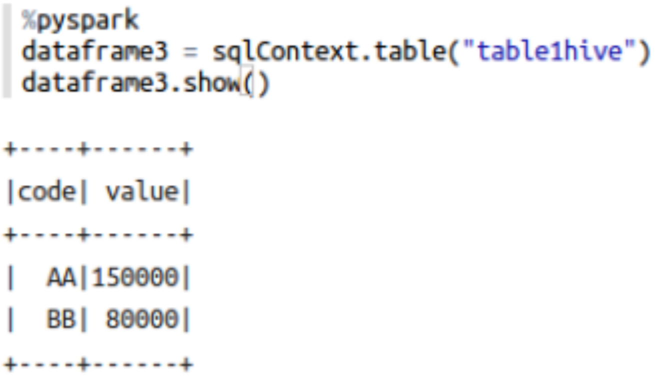


```
%sql
SELECT * FROM table1hive
```

code	value
AA	150,000
BB	80,000

l. Convert this Hive table into a DataFrame named `dataframe3`. View the new DataFrame to confirm success.

```
%pyspark

dataframe3 = sqlContext.table(“table1hive”)

dataframe3.show()
```



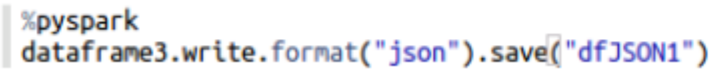
```
%pyspark
dataframe3 = sqlContext.table("table1hive")
dataframe3.show()
```

```
+----+------+
|code| value|
+----+------+
|  AA|150000|
|  BB| 80000|
+----+------+
```

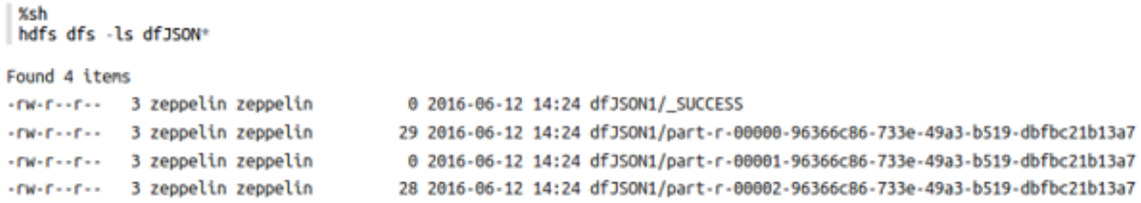
m. Save `dataframe3` to HDFS in JSON format to a folder named `dfJSON1`. In a new paragraph, list all contents of your HDFS home directory to confirm the DataFrame was successfully written.

```
%pyspark

dataframe3.write.format(“json”).save(“dfJSON1”)
```

```
%sh

hdfs dfs -ls dfJSON*
```



```
%pyspark
dataframe3.write.format("json").save("dfJSON1")
```



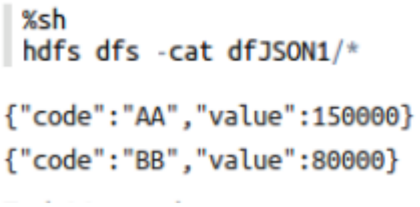
```
%sh
hdfs dfs -ls dfJSON*

Found 4 items
-rw-r--r--   3 zeppelin zeppelin          0 2016-06-12 14:24 dfJSON1/_SUCCESS
-rw-r--r--   3 zeppelin zeppelin         29 2016-06-12 14:24 dfJSON1/part-r-00000-96366c86-733e-49a3-b519-dbfbc21b13a7
-rw-r--r--   3 zeppelin zeppelin          0 2016-06-12 14:24 dfJSON1/part-r-00001-96366c86-733e-49a3-b519-dbfbc21b13a7
-rw-r--r--   3 zeppelin zeppelin         28 2016-06-12 14:24 dfJSON1/part-r-00002-96366c86-733e-49a3-b519-dbfbc21b13a7
```

NOTE:

The JSON file is stored in several `part-*` files in the folder name you specified. If you wanted to copy this file to your local file system for distribution outside the cluster, you could use `hdfs dfs -getmerge` to combine it as a single file on your local file system.

n. View the combined contents of the files in the `dfJSON1` folder on HDFS.

```
%sh

hdfs dfs -cat dfJSON1/*
```



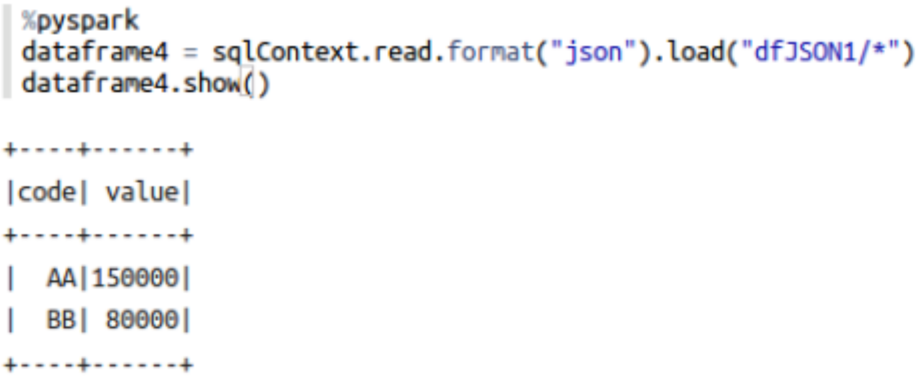
```
%sh
hdfs dfs -cat dfJSON1/*

{"code":"AA","value":150000}
{"code":"BB","value":80000}
```

NOTE:

The JSON format is not what you might typically see when looking at JSON files. For DataFrame creation, each row of information must be self-contained, and thus the formatting you see here is a requirement for converting JSON files to DataFrames. This same content coded in more typical JSON fashion would error out upon attempting to read it as a DataFrame.

o. Create a new DataFrame named `dataframe4` from the contents of this folder on HDFS. View the new DataFrame to confirm success.

```
%pyspark

dataframe4 = sqlContext.read.format(“json”).load(“dfJSON1/*”)

dataframe4.show()
```



```
%pyspark
dataframe4 = sqlContext.read.format("json").load("dfJSON1/*")
dataframe4.show()

+----+------+
|code| value|
+----+------+
|  AA|150000|
|  BB| 80000|
+----+------+
```

Result

You have used several methods to create and save DataFrames and tables.

Lab 9: Working with DataFrames (Python)

About This Lab

Objective:
Learn to use the DataFrames API.

File Locations:
NA

Successful Outcome:
Manipulate DataFrames using the DataFrames API

Lab Steps

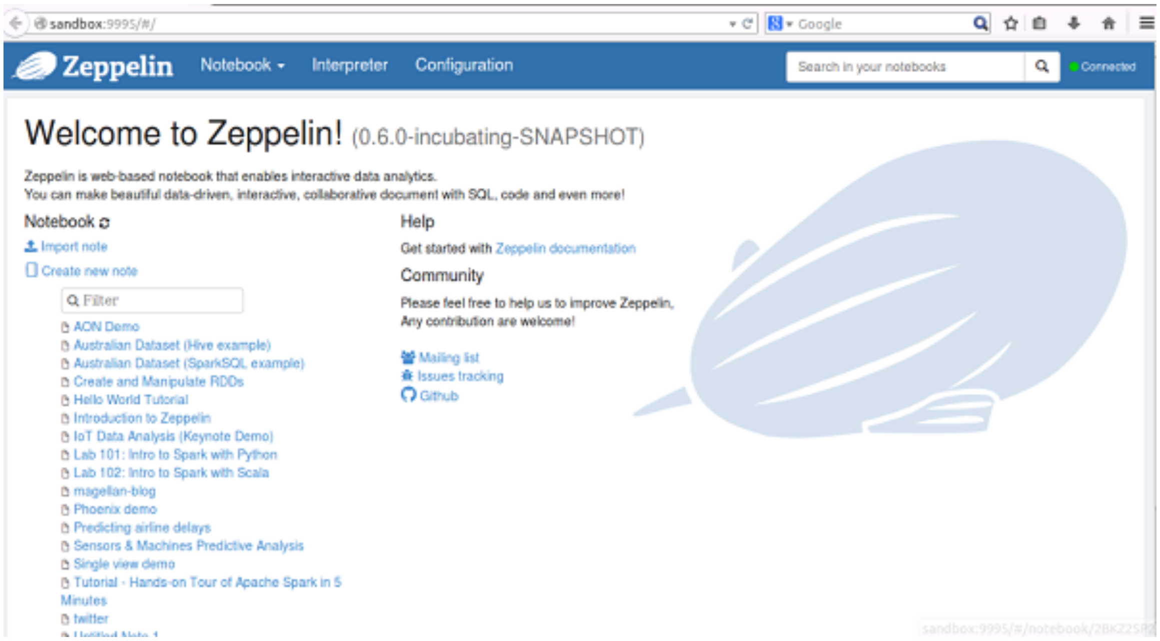
Perform the following steps:

Manipulate DataFrames using the DataFrames API

NOTE:

This lab intentionally makes use of one or more functions not discussed in the student book. The new functions are very similar in nature to functions already discussed in Core RDD programming and should make sense to the student. In addition, some functions are used in ways not discussed in the student book as well. This is to encourage exploration and experimentation, in addition to learning new ways to do things.

a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.
http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note
Create new note
Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Create and Manipulate RDDs
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1

Help

Get started with Zeppelin documentation

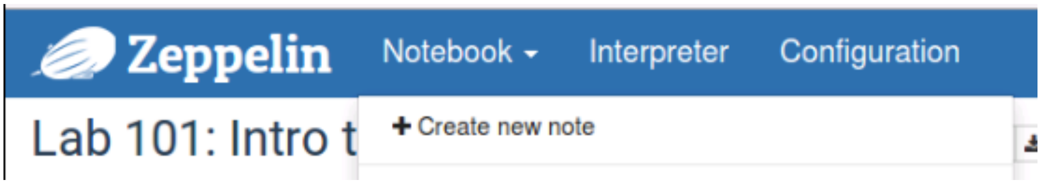
Community

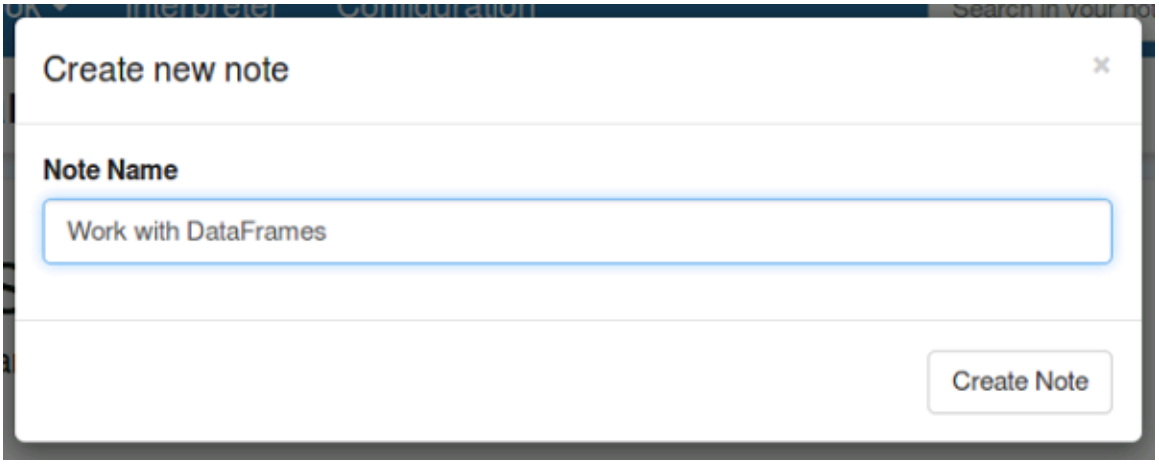
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

sandbox:9995/#/notebook/2BKZ2SR

b. Click on Notebook and select Create new note on the drop down. Name this note Work with DataFrames.

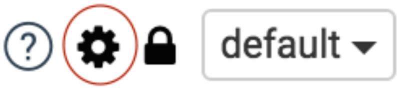


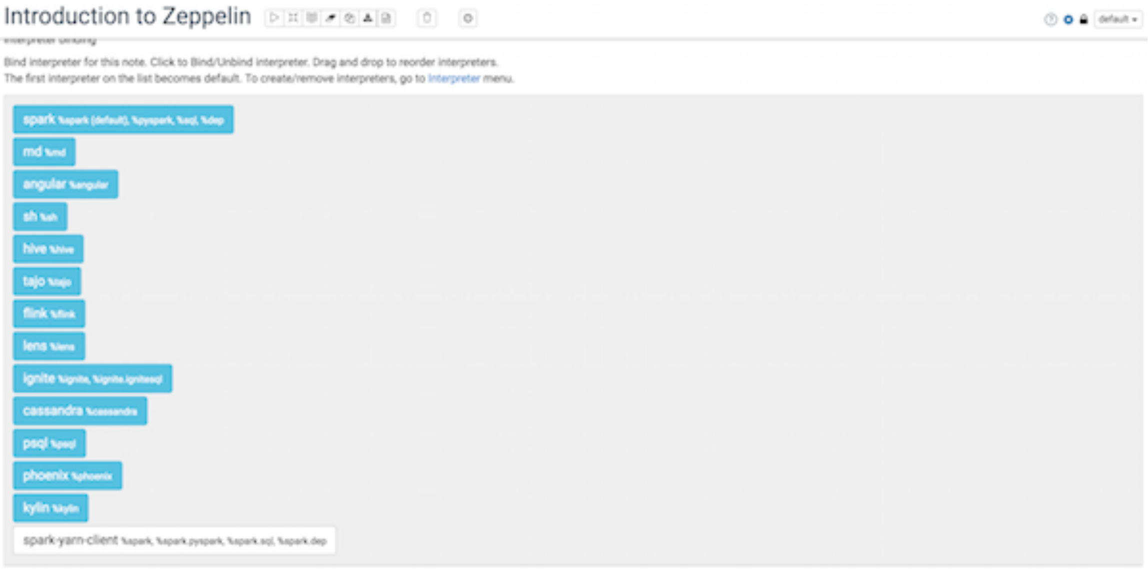


Note Name

Work with DataFrames

c. At the top right click on the gear icon to change interpreter binding.

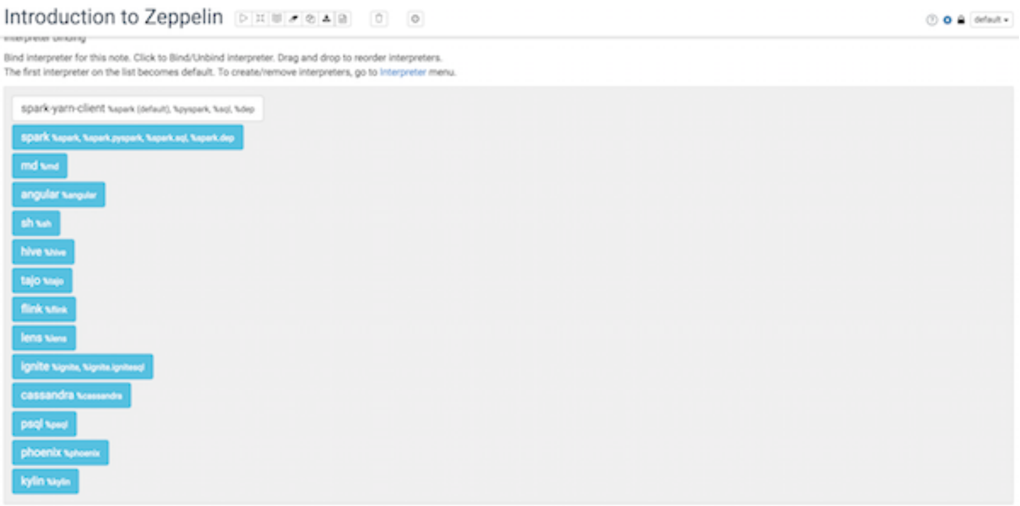




Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

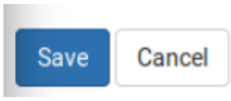
- spark %spark (default), %pyspark, %sql, %dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin
- spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

Drag the spark-yarn-client to the top and click save.



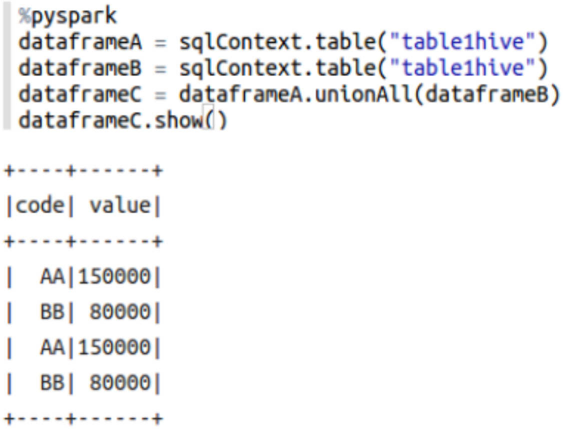
Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

The first interpreter on the list becomes default.



d. Create two DataFrames named `dataframeA` and `dataframeB` from the Hive table named `table1hive` created in the previous lab. Then use `unionAll()` to combine the rows of these two tables into a new DataFrame named `dataframeC`. Then show the contents of dataframeC to confirm success.

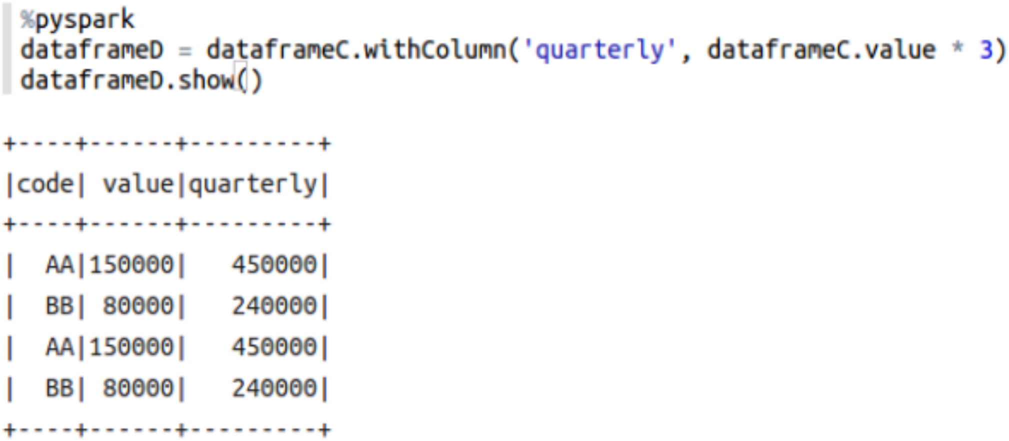
```
%pyspark

dataframeA = sqlContext.table(“table1hive”)

dataframeB = sqlContext.table(“table1hive”)

dataframeC = dataframeA.unionAll(dataframeB)

dataframeC.show()
```



```
%pyspark
dataframeA = sqlContext.table("table1hive")
dataframeB = sqlContext.table("table1hive")
dataframeC = dataframeA.unionAll(dataframeB)
dataframeC.show()

+----+------+
|code| value|
+----+------+
|  AA|150000|
|  BB| 80000|
|  AA|150000|
|  BB| 80000|
+----+------+
```

e. **Create a DataFrame named `dataframeD` that adds a column named `quarterly` that contains the contents of the `value` column multiplied by three. View the new DataFrame to confirm success.**

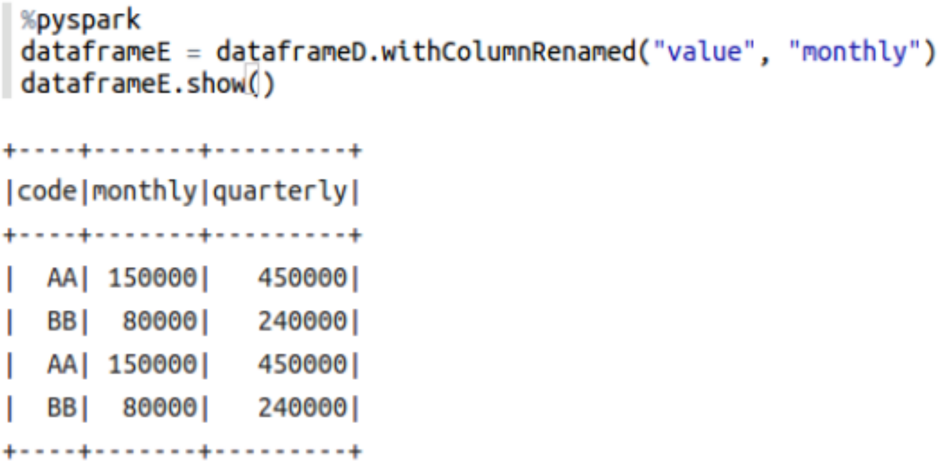
```
%pyspark

dataframeD = dataframeC.withColumn(‘quarterly’, dataframeC.value * 3)

dataframeD.show()
```



```
%pyspark
dataframeD = dataframeC.withColumn('quarterly', dataframeC.value * 3)
dataframeD.show()

+----+------+---------+
|code| value|quarterly|
+----+------+---------+
|  AA|150000|   450000|
|  BB| 80000|   240000|
|  AA|150000|   450000|
|  BB| 80000|   240000|
+----+------+---------+
```

f. **Create a DataFrame named `dataframeE` that renames the `value` column to `monthly`. View the new DataFrame to confirm success.**

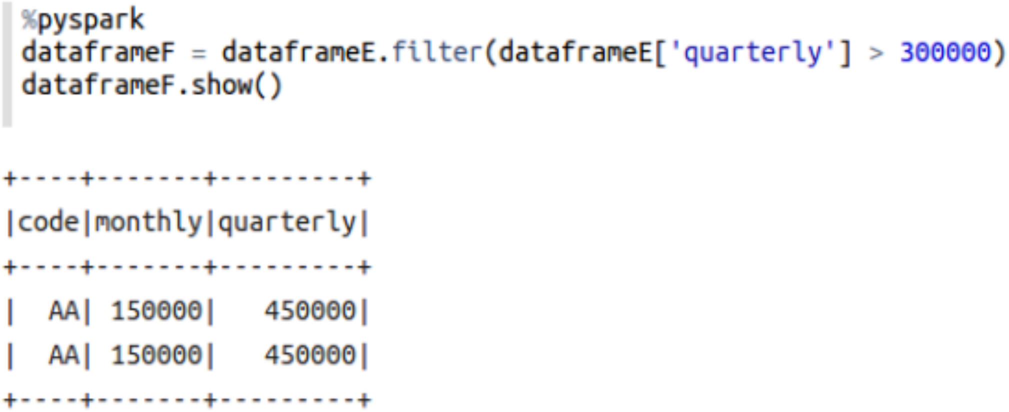
```
%pyspark

dataframeE = dataframeD.withColumnRenamed(“value”, “monthly”)

dataframeE.show()
```



```
%pyspark
dataframeE = dataframeD.withColumnRenamed("value", "monthly")
dataframeE.show()

+----+-------+---------+
|code|monthly|quarterly|
+----+-------+---------+
|  AA| 150000|   450000|
|  BB|  80000|   240000|
|  AA| 150000|   450000|
|  BB|  80000|   240000|
+----+-------+---------+
```

g. **Create a DataFrame named `dataframeF` that contains only those rows from `dataframeE` where the `quarterly` value is greater than 300,000. View the new DataFrame to confirm success.**

```
%pyspark

dataframeF = dataframeE.filter(dataframeE[‘quarterly’] > 300000)

dataframeF.show()
```

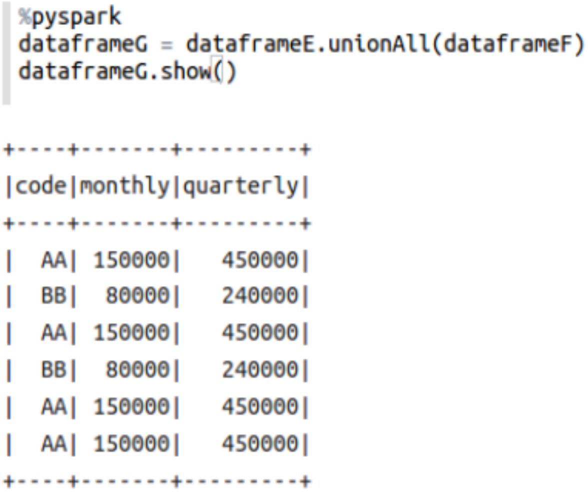


```
%pyspark
dataframeF = dataframeE.filter(dataframeE['quarterly'] > 300000)
dataframeF.show()
```

```
+----+-------+---------+
|code|monthly|quarterly|
+----+-------+---------+
|  AA| 150000|   450000|
|  AA| 150000|   450000|
+----+-------+---------+
```

h. **Create a new DataFrame named `dataframeG` that adds the rows of `dataframeE` to `dataframeF` so that there are six rows total. View the new DataFrame to confirm success.**

```
%pyspark

dataframeG = dataframeE.unionAll(dataframeF)

dataframeG.show()
```



```
%pyspark
dataframeG = dataframeE.unionAll(dataframeF)
dataframeG.show()
```

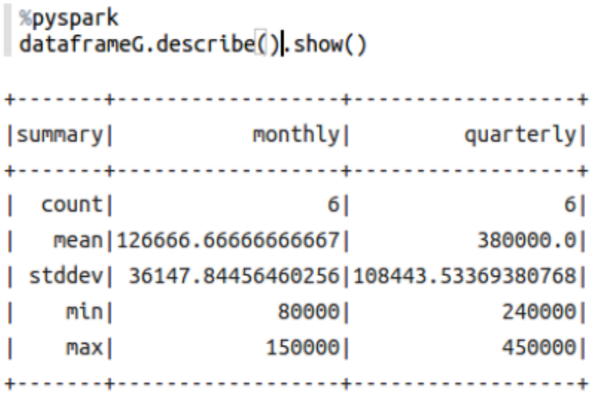
```
+----+-------+---------+
|code|monthly|quarterly|
+----+-------+---------+
|  AA| 150000|   450000|
|  BB|  80000|   240000|
|  AA| 150000|   450000|
|  BB|  80000|   240000|
|  AA| 150000|   450000|
|  AA| 150000|   450000|
+----+-------+---------+
```

i. Use `describe()` on `dataframeG` without supplying a column name and show the results.

QUESTION:

What happens?

```
%pyspark

dataframeG.describe().show()
```



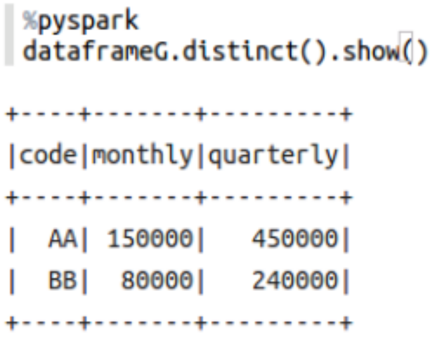
```
%pyspark
dataframeG.describe().show()

+-------+-----------------+-----------------+
|summary|          monthly|        quarterly|
+-------+-----------------+-----------------+
|  count|                6|                6|
|   mean|126666.66666666667|         380000.0|
| stddev| 36147.84456460256|108443.53369380768|
|    min|            80000|           240000|
|    max|           150000|           450000|
+-------+-----------------+-----------------+
```

ANSWER:

All columns with numeric values have statistics displayed.

j. Show only unique rows from `DataFrameG`.

```
%pyspark

dataframeG.distinct().show()
```



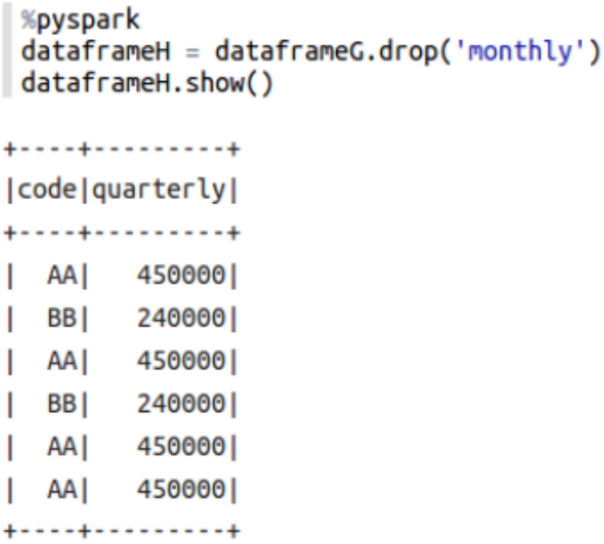
```
%pyspark
dataframeG.distinct().show()

+----+-------+---------+
|code|monthly|quarterly|
+----+-------+---------+
|  AA| 150000|   450000|
|  BB|  80000|   240000|
+----+-------+---------+
```

k. **Use `drop()` to create a new DataFrame named `dataframeH` that contains only the `code` and `quarterly` columns. View the new DataFrame to confirm success.**

QUESTION:

What other function described in the student book, could you have used to accomplish the same task? What would the code have been?

```
%pyspark

dataframeH = dataframeG.drop(‘monthly’)

dataframeH.show()
```

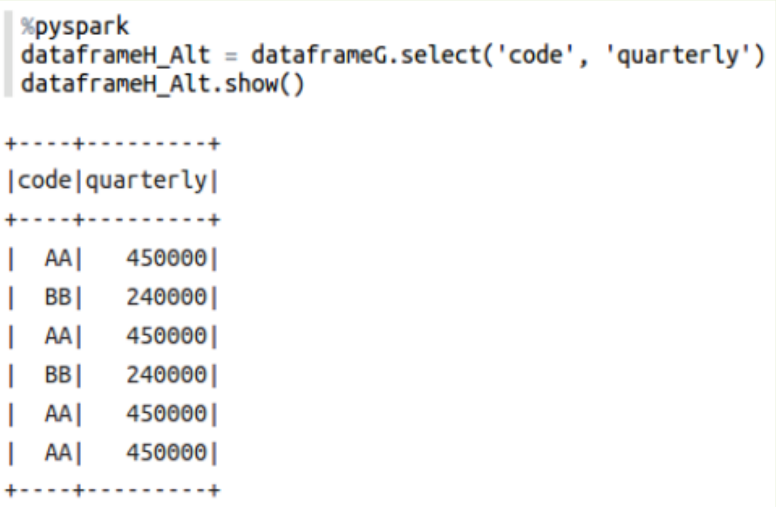


```
%pyspark
dataframeH = dataframeG.drop('monthly')
dataframeH.show()

+----+---------+
|code|quarterly|
+----+---------+
|  AA|   450000|
|  BB|   240000|
|  AA|   450000|
|  BB|   240000|
|  AA|   450000|
|  AA|   450000|
+----+---------+
```

ANSWER:

The same thing could have been accomplished using the following code:

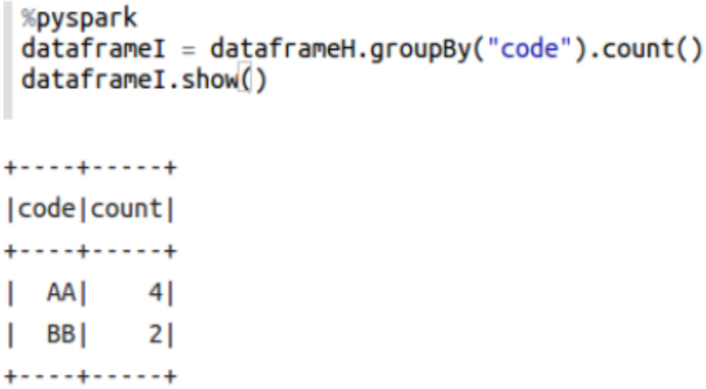
```
dataframeH = dataframeG.select(‘code’, ‘quarterly’)
```



```
%pyspark
dataframeH_Alt = dataframeG.select('code', 'quarterly')
dataframeH_Alt.show()

+----+---------+
|code|quarterly|
+----+---------+
|  AA|   450000|
|  BB|   240000|
|  AA|   450000|
|  BB|   240000|
|  AA|   450000|
|  AA|   450000|
+----+---------+
```

l. Create a new DataFrame named `dataframeI` that contains each unique element in the code column and a count of the number of times each code appears `dataframeH`. View the new DataFrame to confirm success.

```
%pyspark

dataframeI = dataframeH.groupBy(“code”).count()

dataframeI.show()
```



```
%pyspark
dataframeI = dataframeH.groupBy("code").count()
dataframeI.show()

+----+-----+
|code|count|
+----+-----+
|  AA|    4|
|  BB|    2|
+----+-----+
```

Result

You have successfully used the DataFrames API to manipulate DataFrames.

Lab 10: Data Visualization, Reporting & Collaboration using Zeppelin (Scala)

About This Lab

Objective:
Learn to use Zeppelin to perform data visualizations, collaborate, and integrate visualizations into reports.

Files Locations:
NA

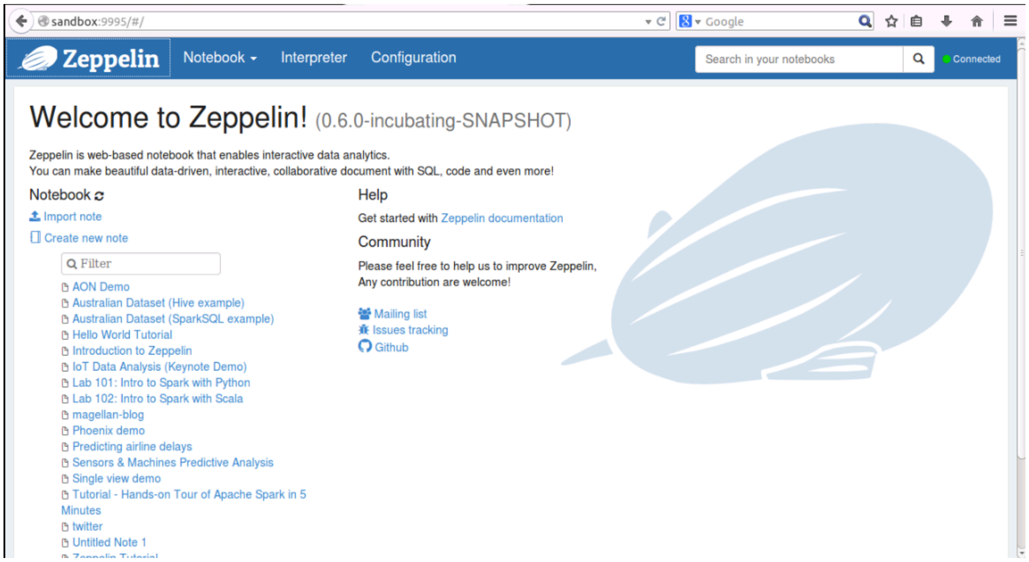
Successful Outcome:
Use Zeppelin to perform data visualization, collaboration, and reporting tasks.

Lab Steps

Perform the following steps:

1. Create data visualizations from a file of banking data.

   a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.
   http://sandbox:9995/



sandbox:9995/#/

Zeppelin Notebook Interpreter Configuration Search in your notebooks Connected

Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1
- Zeppelin Tutorial

Help

Get started with Zeppelin documentation

Community

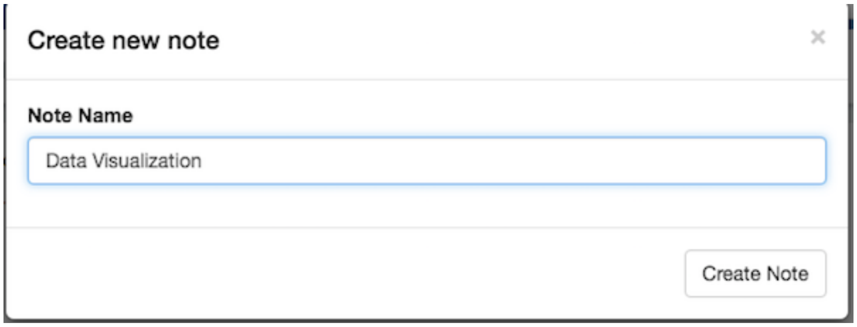
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

NOTE:

Zepplin’s current main backend processing engine is Apache Spark.

b. Create a new note named Data Visualization.

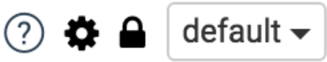


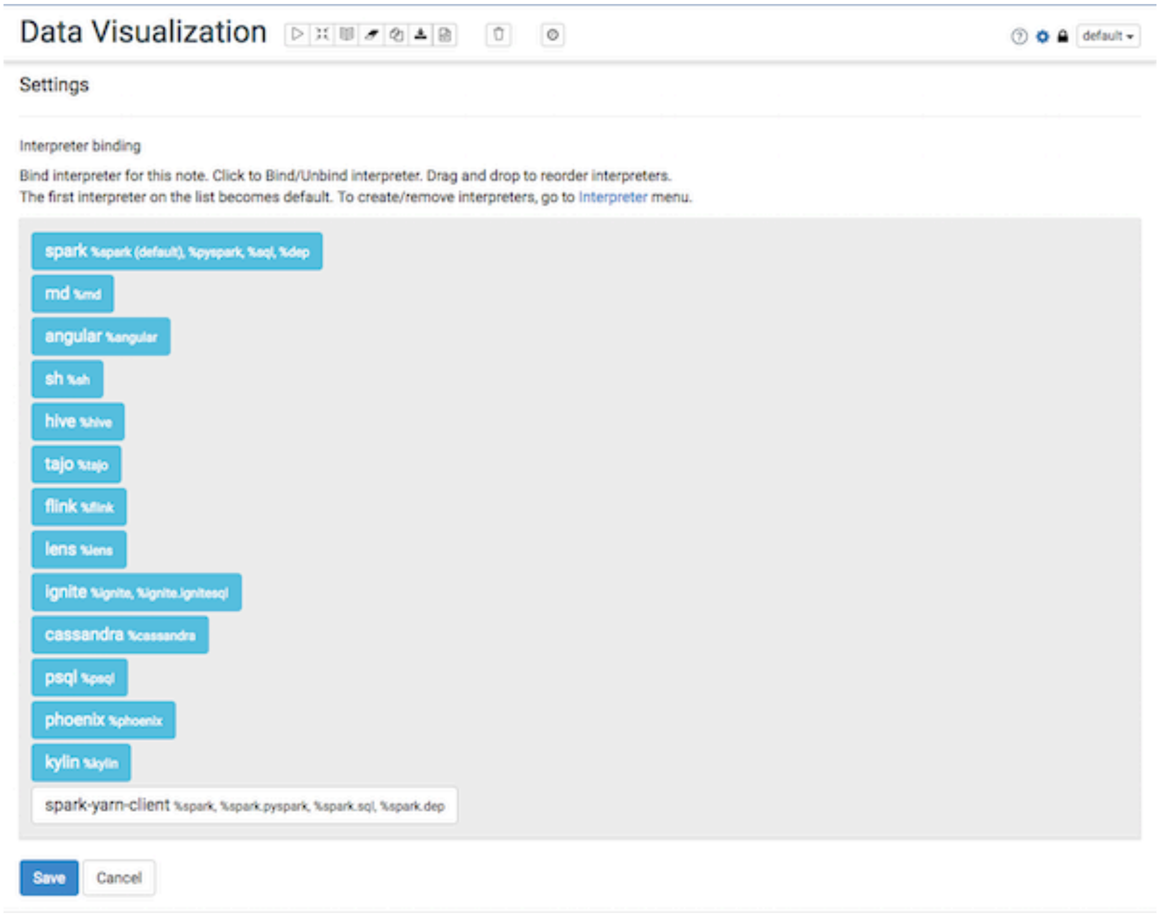
Create new note

Note Name

Data Visualization

c. Set the interpreter for this note to spark-yarn-client.





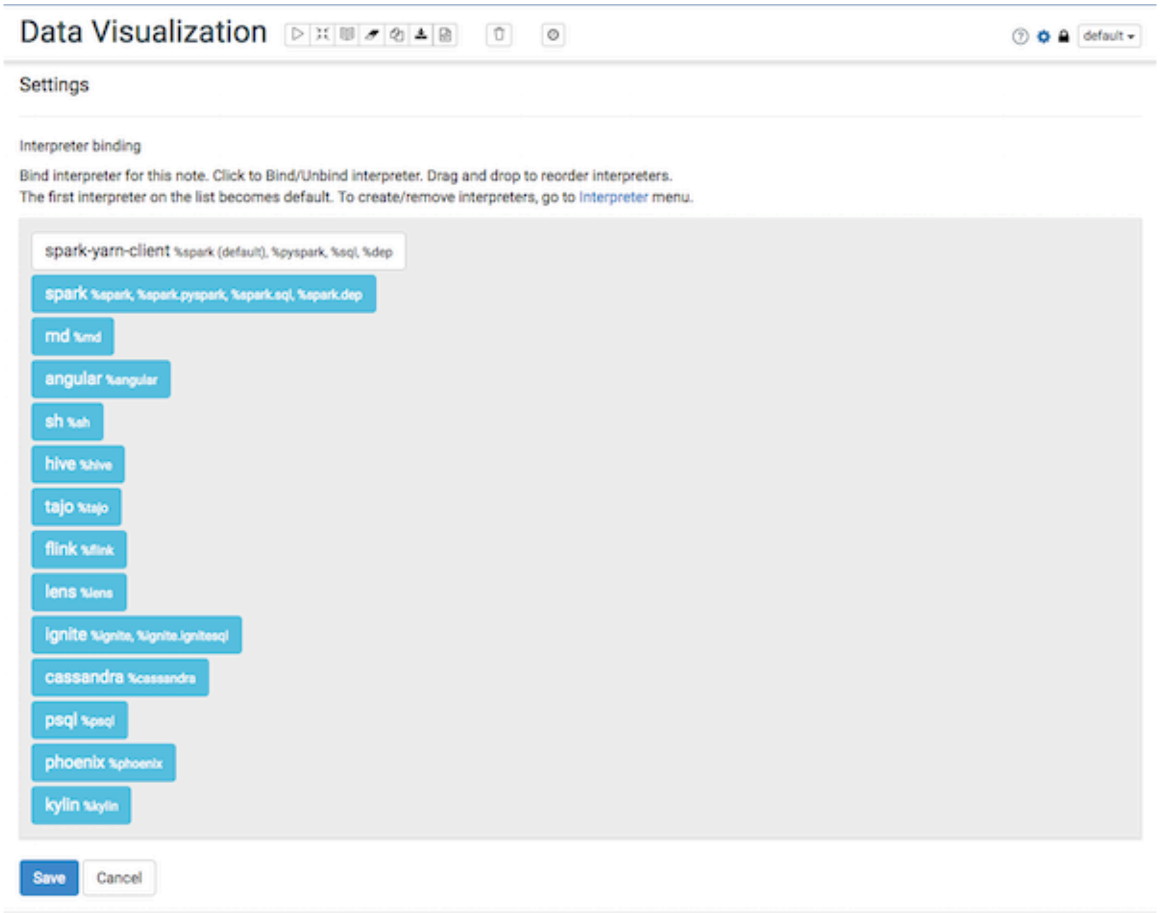
Data Visualization

Settings

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

- spark %spark (default), %pyspark, %sql, %dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin
- spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep



Settings

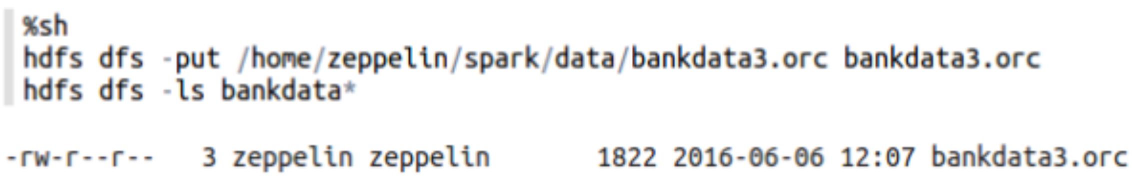
Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark-yarn-client %spark (default), %pyspark, %sql, %dep
spark %spark, %spark.pyspark, %spark.sql, %spark.dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin

d. Upload the bankdata3.orc file from the `/home/zeppelin/spark/data` directory on your local file system to your HDFS home directory. Confirm the file was uploaded successfully.

```
%sh

hdfs dfs -put /home/zeppelin/spark/data/bankdata3.orc bankdata3.orc

hdfs dfs -ls bankdata*
```



```
%sh
hdfs dfs -put /home/zeppelin/spark/data/bankdata3.orc bankdata3.orc
hdfs dfs -ls bankdata*

-rw-r--r--   3 zeppelin zeppelin       1822 2016-06-06 12:07 bankdata3.orc
```

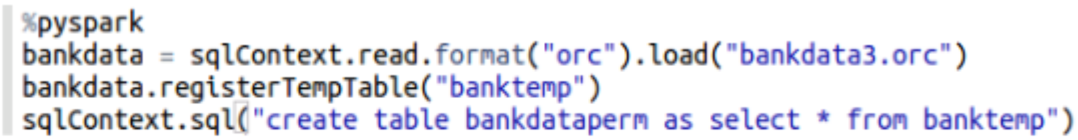
NOTE:

This data is a cleaned subset of a publicly available machine learning dataset. The original dataset can be found at the following link:

http://archive.ics.uci.edu/ml/machine-learning-databases/00222/

e. Use the bankdata3.orc file to create a DataFrame named `bankdata`, a temporary table named `banktemp`, and a Hive table named `bankdataperm`.

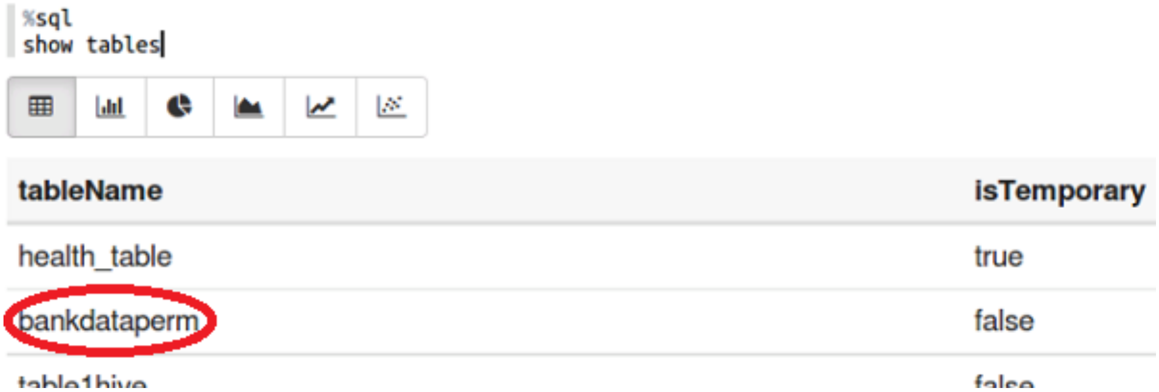
```
%pyspark

bankdata = sqlContext.read.format(“orc”).load(“bankdata3.orc”)

bankdata.registerTempTable(“banktemp”)

sqlContext.sql(“create table bankdataperm as select * from banktemp”)
```



```
%pyspark
bankdata = sqlContext.read.format("orc").load("bankdata3.orc")
bankdata.registerTempTable("banktemp")
sqlContext.sql("create table bankdataperm as select * from banktemp")
```

f. Use SQL to show the tables available and confirm that `bankdataperm` is available.

```
%sql

show tables
```

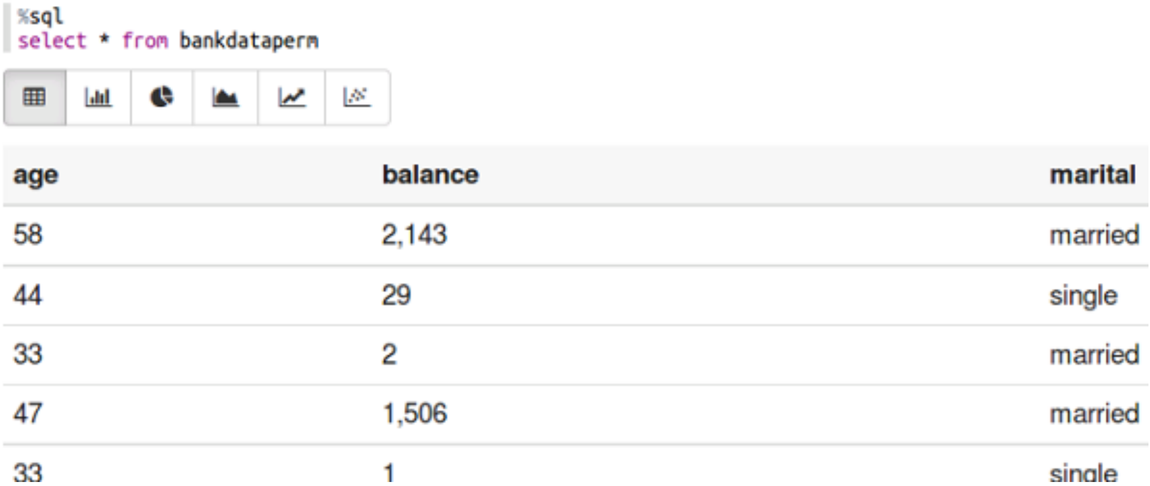


```
%sql
show tables
```

tableName	isTemporary
health_table	true
bankdataperm	false
table1hive	false

g. Use SQL to select and display all rows and columns from `bankdataperm`.

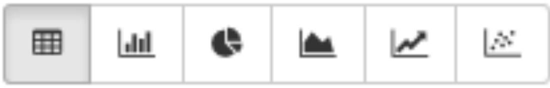
```
%sql

select * from bankdataperm
```

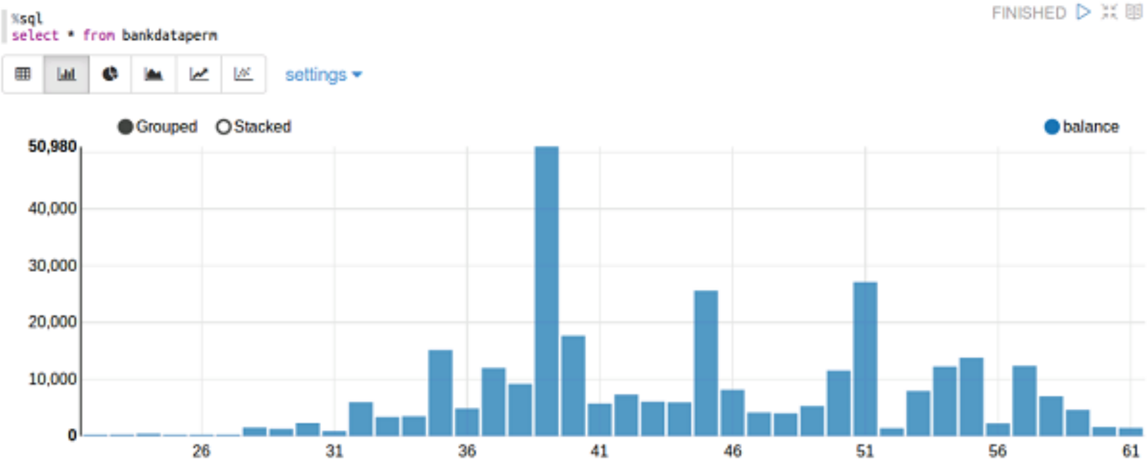


```
%sql
select * from bankdataperm
```

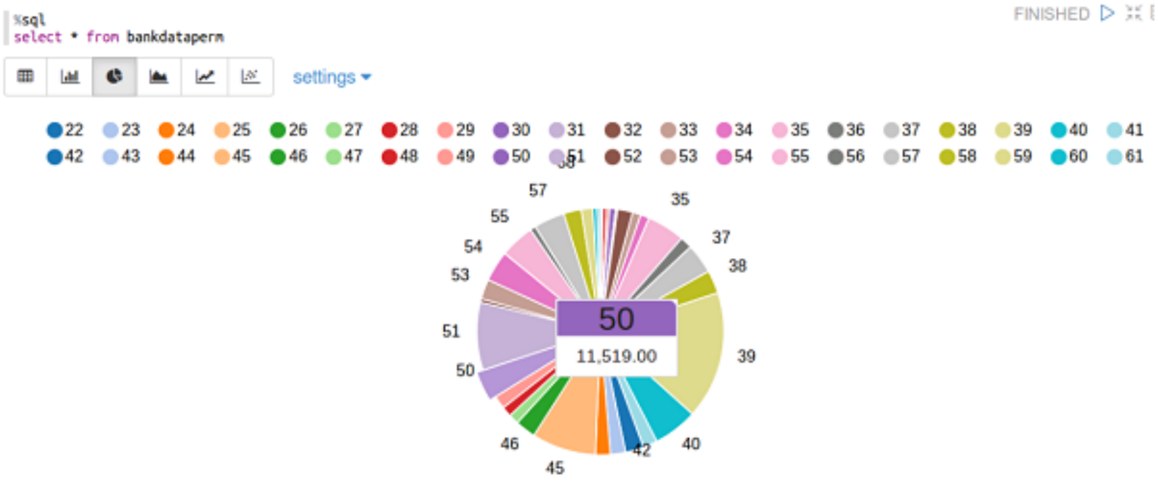
age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,506	married
33	1	single

h. Quickly browse through the five data visualizations available by default in Zeppelin. For most of this lab, we will work with the bar chart view.

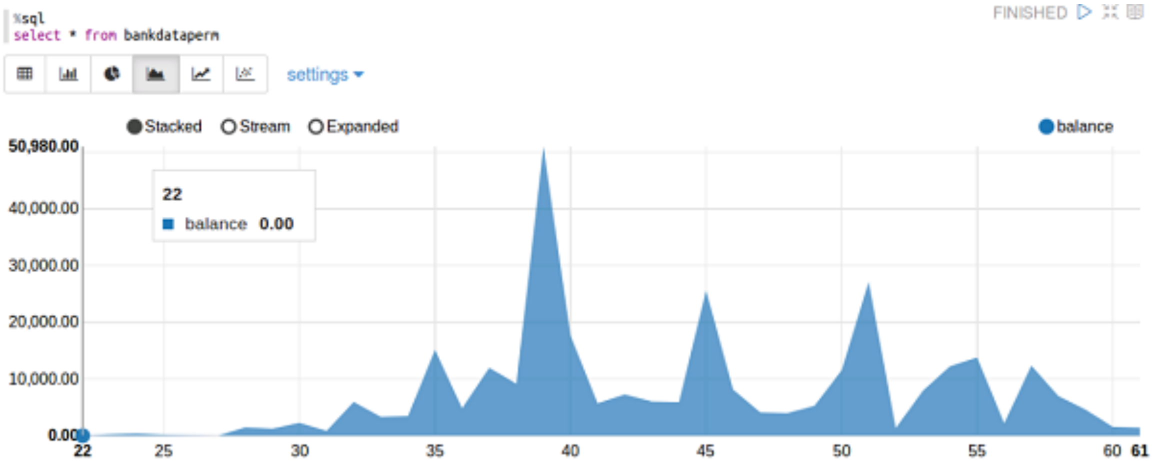




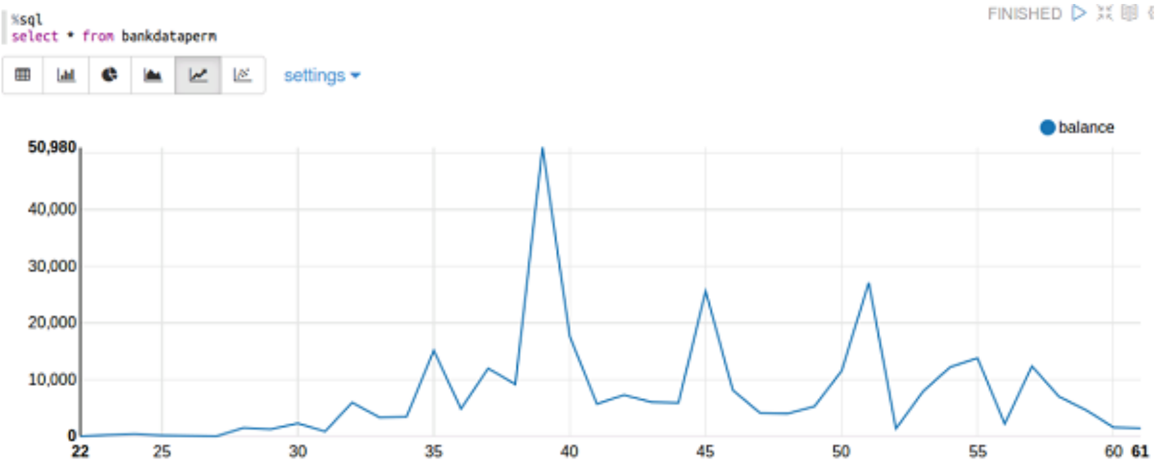
```
%sql
select * from bankdataperm
```



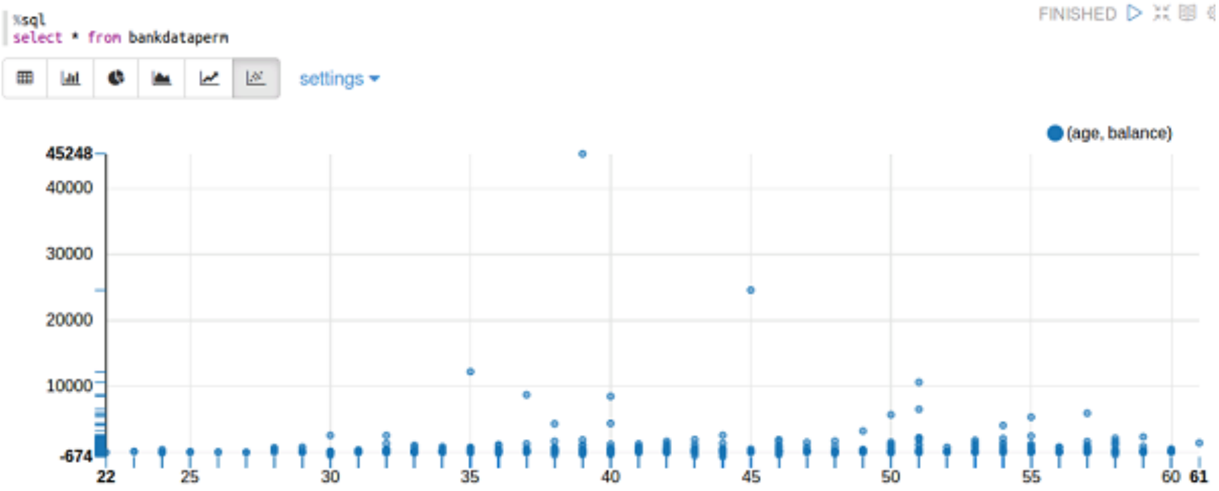
```
%sql
select * from bankdataperm
```



```
%sql
select * from bankdataperm
```



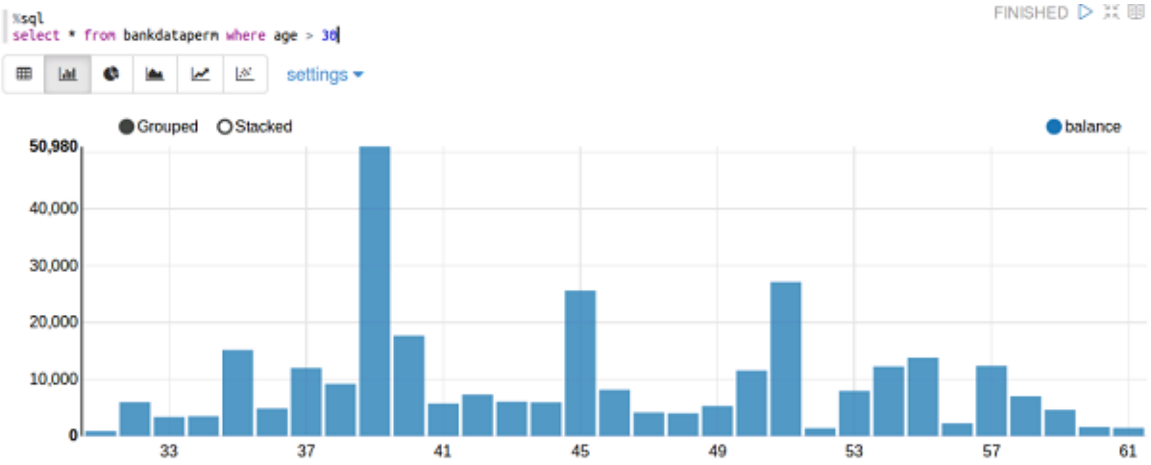
```
%sql
select * from bankdataperm
```



```
%sql
select * from bankdataperm
```

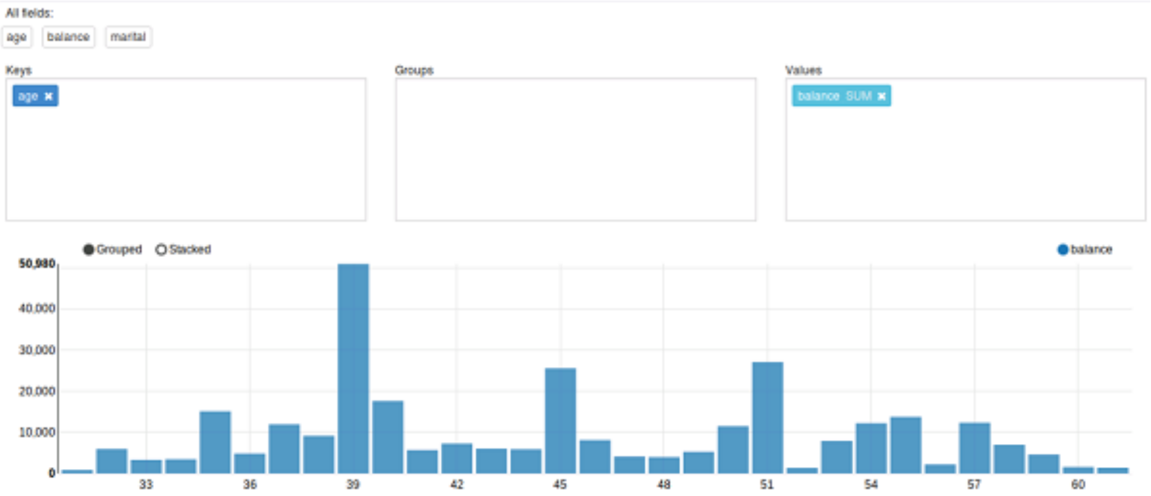
i. Go back to the bar chart view. Then, edit your SQL query so that it only shows data for individuals over the age of 30. Run the query and note the change in the chart.

```
%sql

select * from bankdataperm where age > 30
```



```
%sql
select * from bankdataperm where age > 30
```

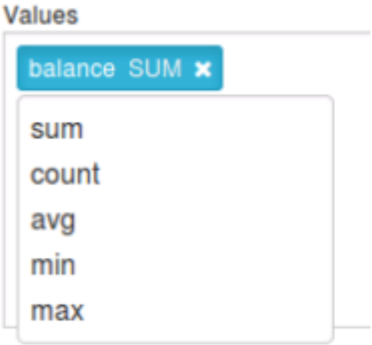
j. Click on the settings link and notice that Zeppelin has selected the age column as the key column and is showing the sum of the balances for all individuals in each age bracket. Display the average balance instead of the sum of balances.

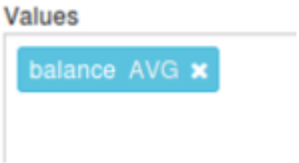

All fields: age balance marital

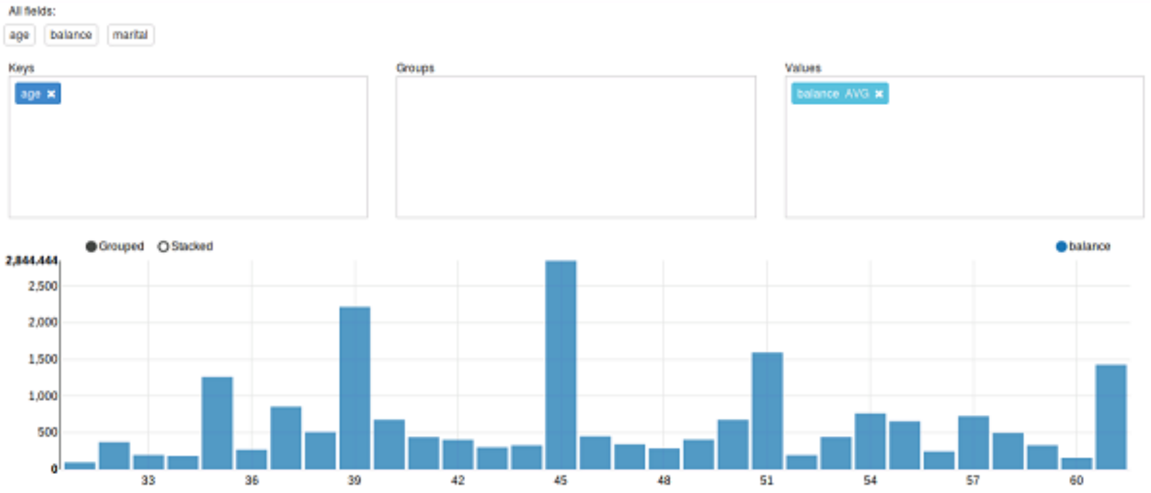
Keys: age

Groups:

Values: balance SUM


Values: balance SUM


Values: balance AVG



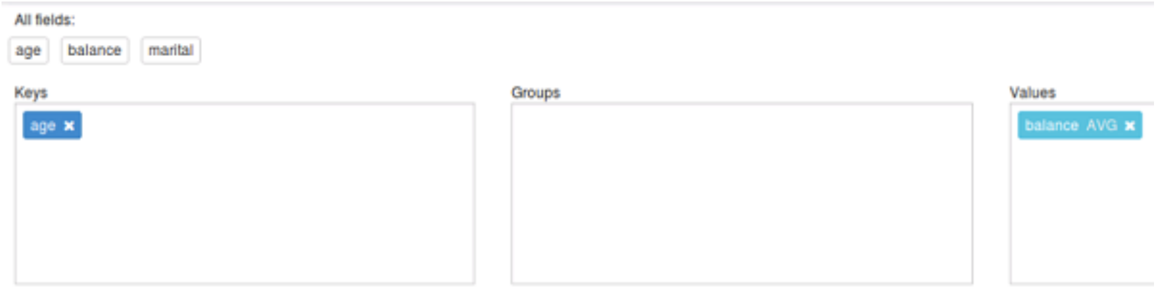
All fields:
age balance marital

Keys: age ✖

Groups:

Values: balance AVG ✖

k. Click and drag the available `marital` field into the `Groups` category to modify the visualization so that data is shown not only by age, but also grouped by marital status. When you are finished, click the settings link again to close the pivot chart options.

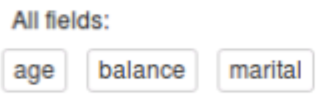


All fields:
age balance marital

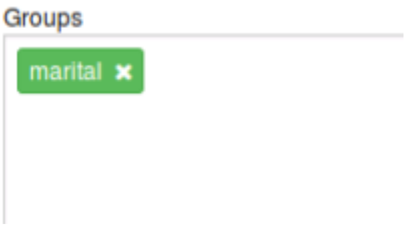
Keys: age ✖

Groups:

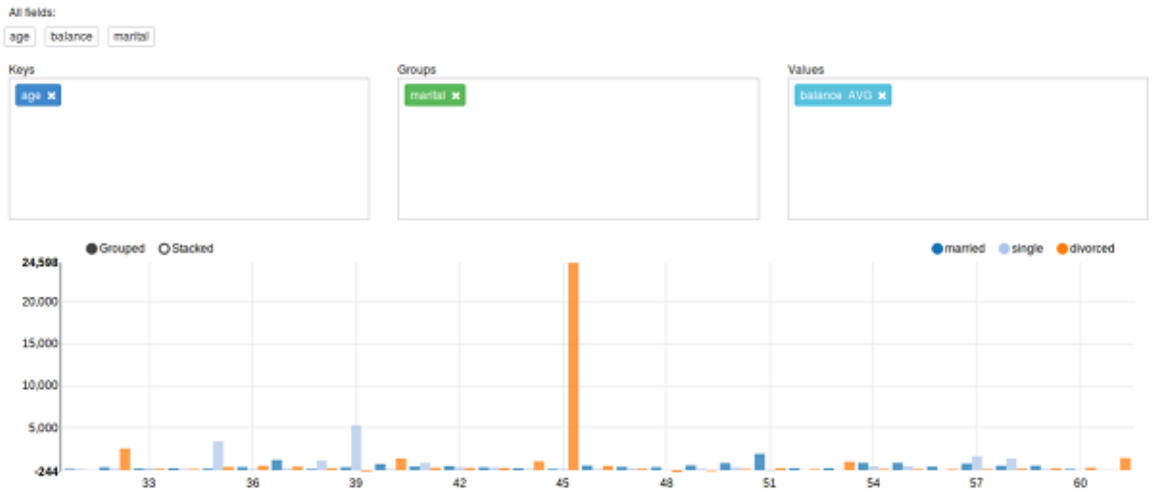
Values: balance AVG ✖



All fields:
age balance marital

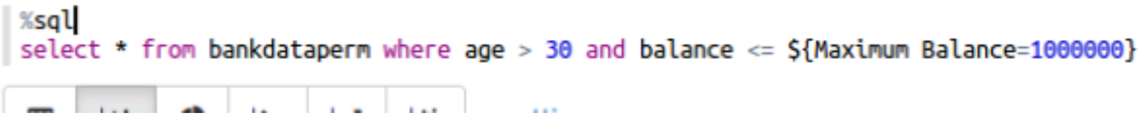


Groups: marital ✖

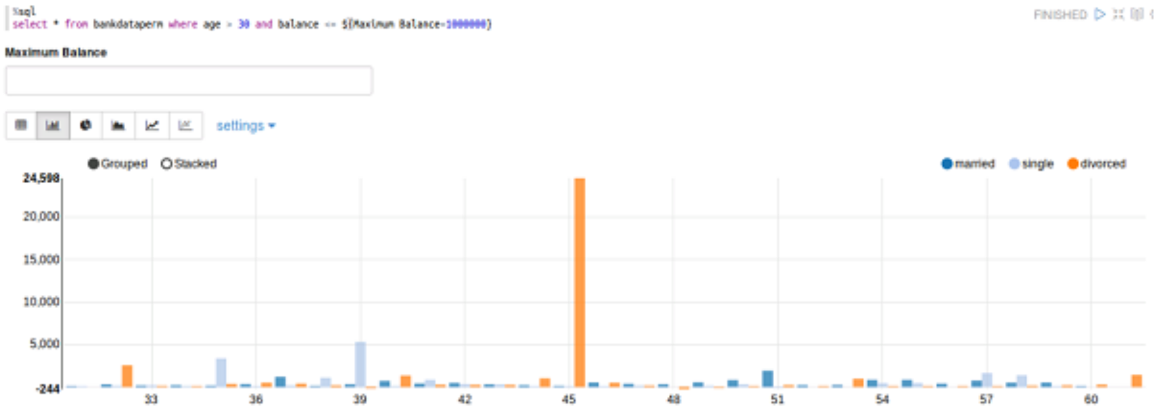


I. It appears that we have what appears to be a single outlier that is skewing the data fairly significantly. We can easily see that the vast majority of average balances are well below $5,000. Add a dynamic form to the SQL query that allows you to filter out data where the maximum balance for any individual exceeds a certain threshold, but set the default to 1,000,000 so that it doesn't immediately modify the chart. Rerun the query with this new code, then use this dynamic form to adjust the maximum balance to $10,000 and $5,000 and note the effects on the visualization.

```
%sql

select * from bankdataperm where age > 30 and balance <= ${Maximum
Balance=1000000}
```

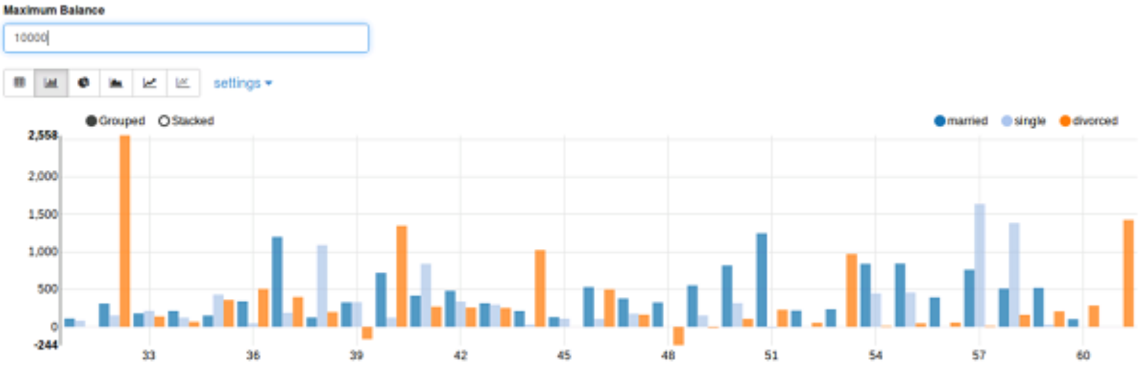


```
%sql
select * from bankdataperm where age > 30 and balance <= ${Maximum Balance=1000000}
```



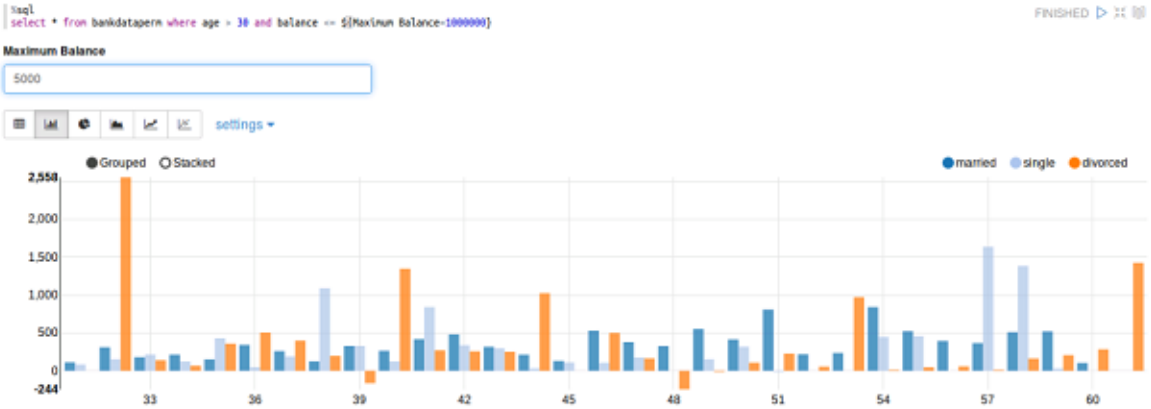
```
%sql
select * from bankdataperm where age > 30 and balance <= ${Maximum Balance=1000000}
```

Maximum Balance



Maximum Balance

10000



```
%sql
select * from bankdatapern where age > 30 and balance <= ${Maximum Balance=1000000}
```

Maximum Balance

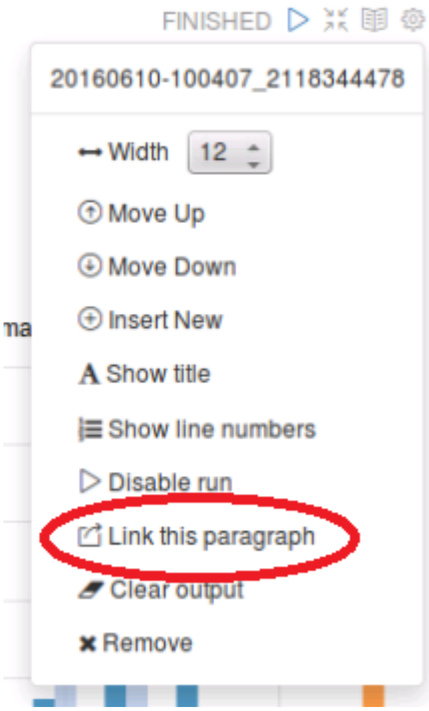
5000

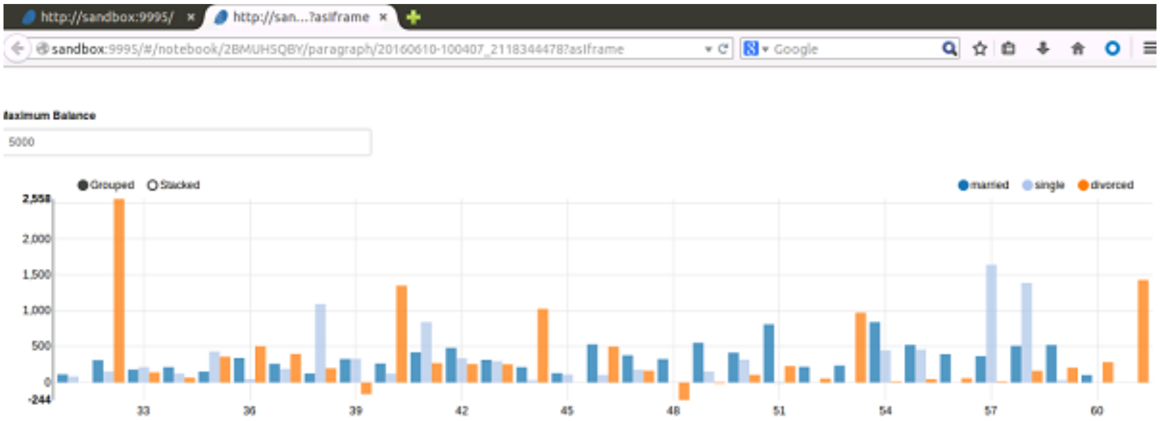
QUESTIONS:

Why do you think changing the maximum balance from $10,000 to $5,000 had so little effect on the chart?

What group (married, single, or divorced) had the most change based on changing the maximum balance?

m. Create a URL that allows you to share this chart with others without giving them access to the code or the Zeppelin note. Use the linked page to change the maximum balance to $2,500, then return to your note and observe the effects the change had at the source.

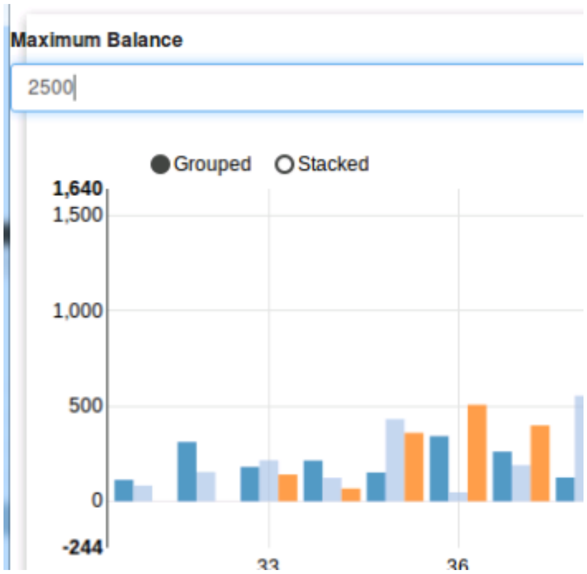




sandbox:9995/#/notebook/2BMUHSQBY/paragraph/20160610-100407_2118344478?asIframe

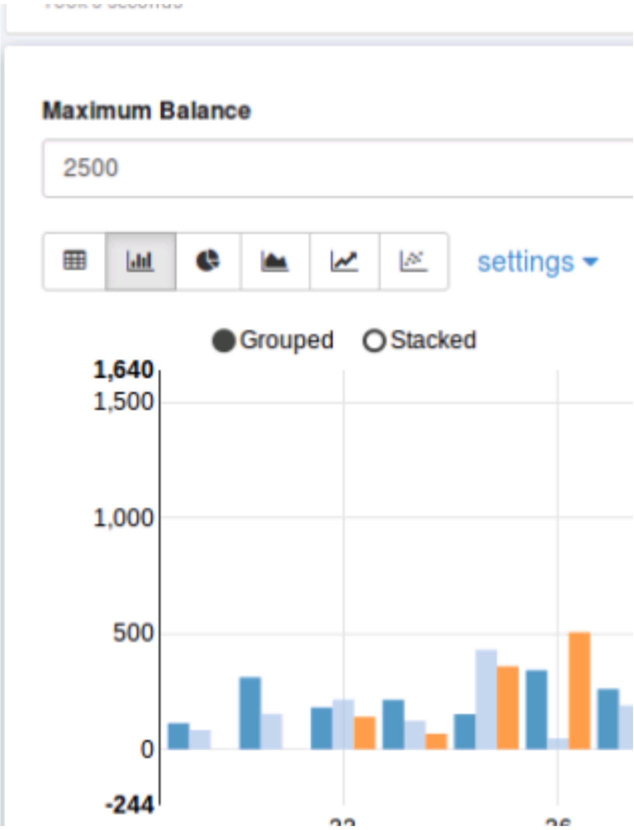
Maximum Balance

5000



Maximum Balance

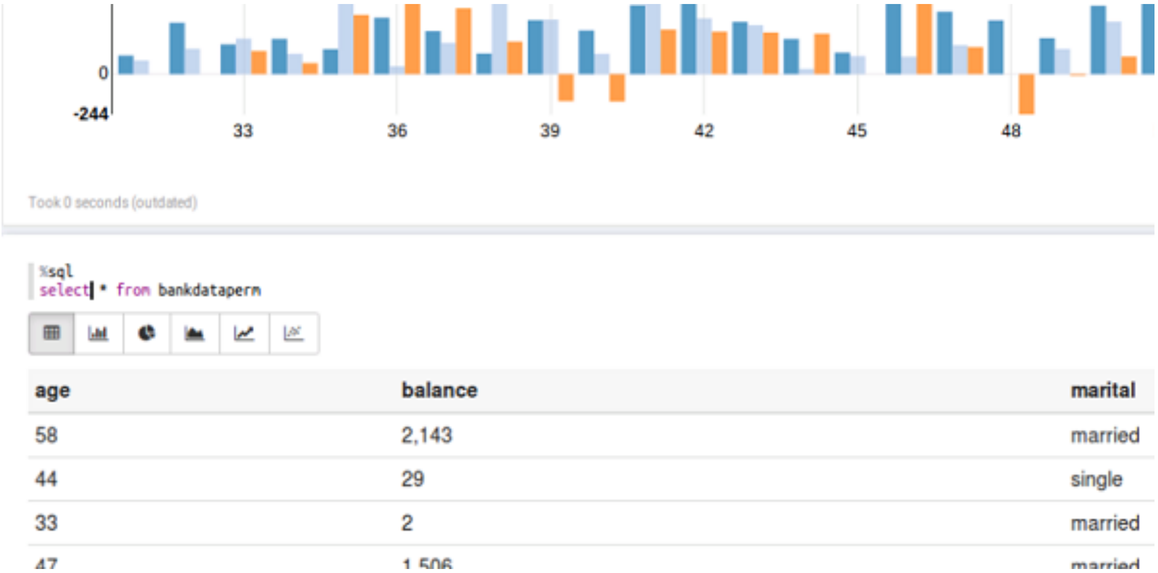
2500



Maximum Balance

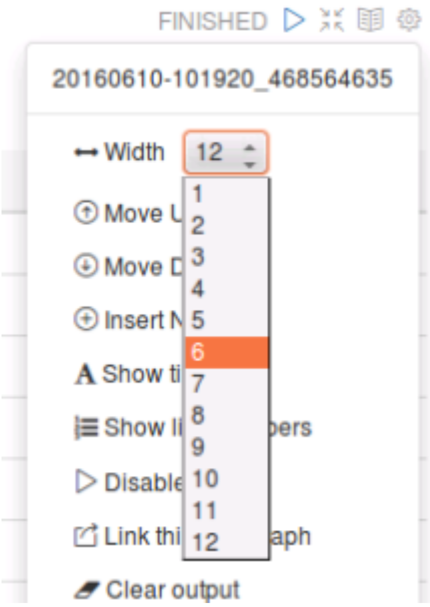
2500

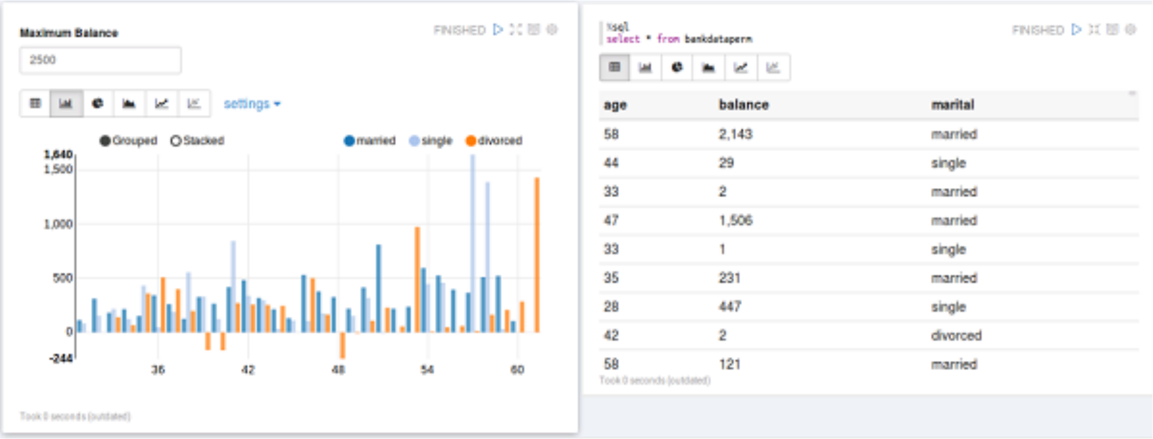
n. In the paragraph below this one, run the SQL command to read all data from `bankdataperm`. Then adjust the width of the two paragraphs so that they both appear on the same line.



```
%sql
select * from bankdataperm
```

age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,506	married



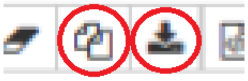


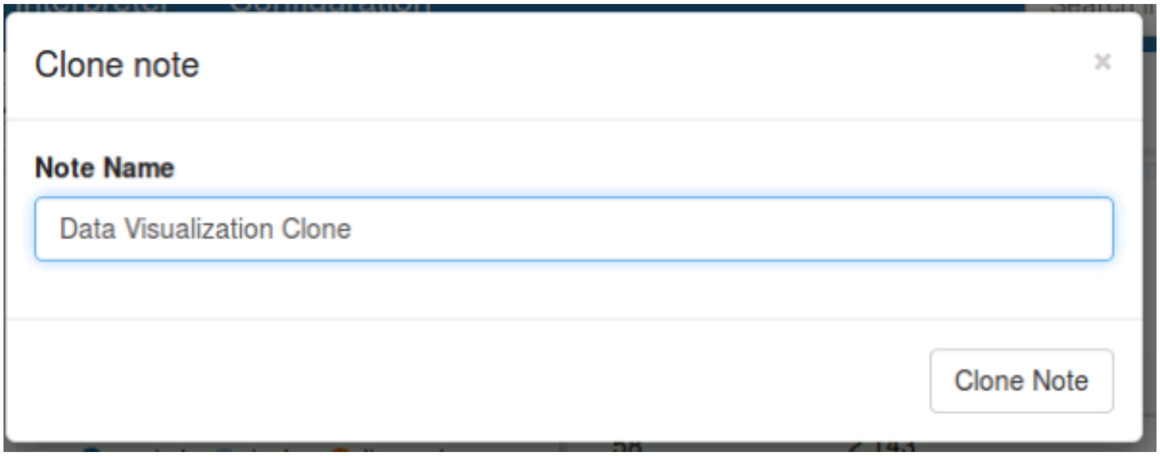
```
%sql
select * from bankdatapern
```

age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,506	married
33	1	single
35	231	married
28	447	single
42	2	divorced
58	121	married

Took 0 seconds (outdated)

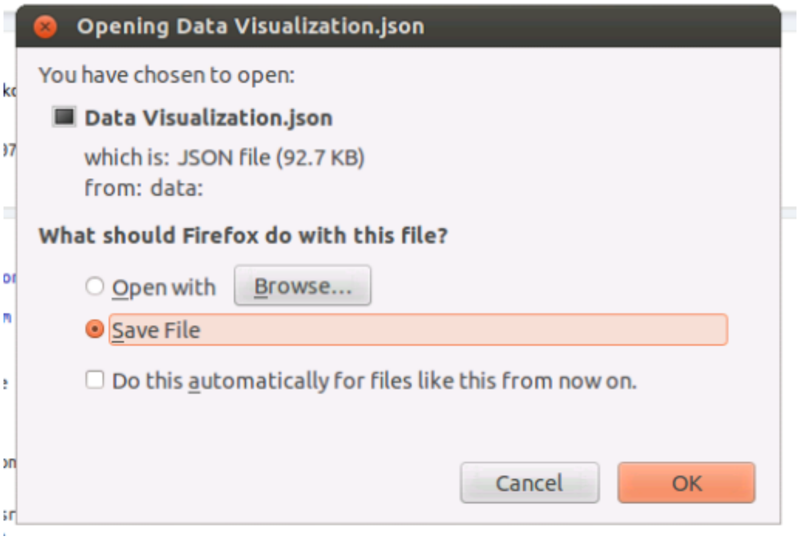
o. We are now ready to prepare this note for sharing. Create a clone copy of this note named `Data Visualization Clone`. Also export a copy of the note.





Clone note

Note Name: Data Visualization Clone



Opening Data Visualization.json

You have chosen to open:

Data Visualization.json

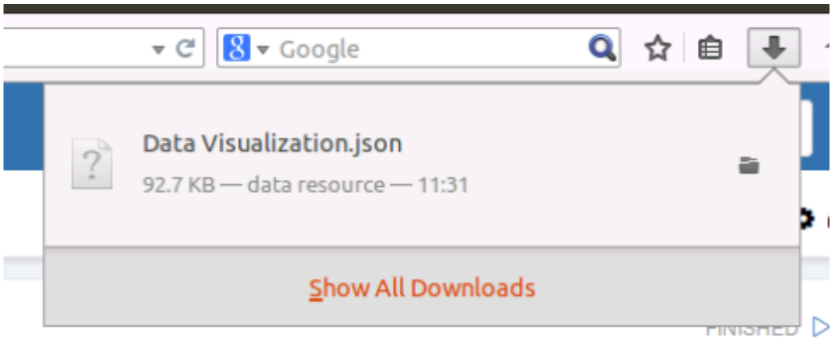
which is: JSON file (92.7 KB)
from: data:

What should Firefox do with this file?

○ Open with Browse...

● Save File

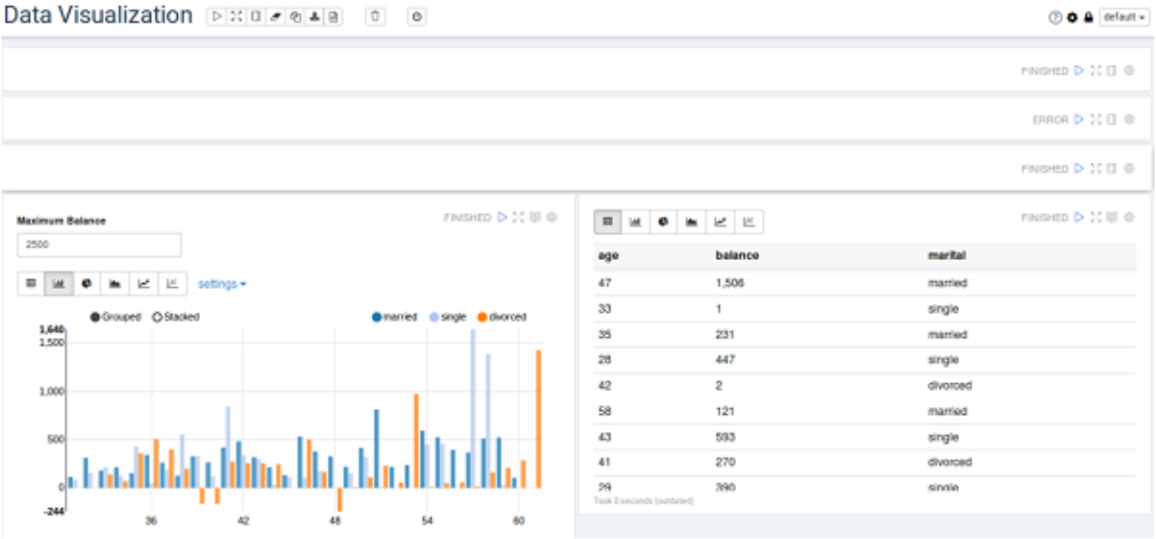
□ Do this automatically for files like this from now on.



Data Visualization.json

92.7 KB — data resource — 11:31

p. On the Data Visualization note we are going to share, hide the code for all paragraphs. Then hide the output for every paragraph except for the two that are on the same line.

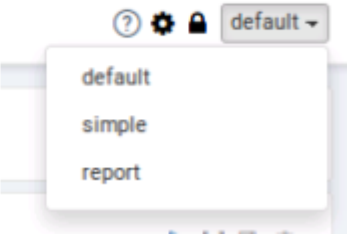


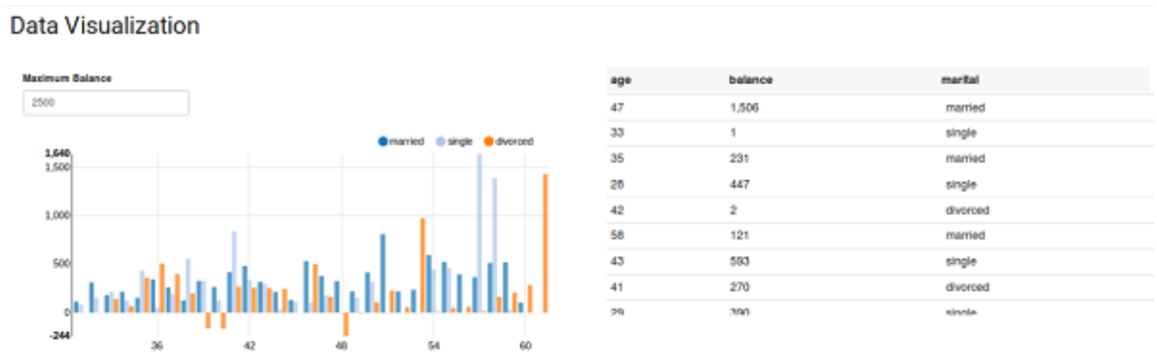
Data Visualization

Maximum Balance: 2500

age	balance	marital
47	1,506	married
33	1	single
35	231	married
28	447	single
42	2	divorced
58	121	married
43	593	single
41	270	divorced
29	390	single

q. Next, convert this from the default view to report view. Now the URL to this note is ready to share with your stakeholders.

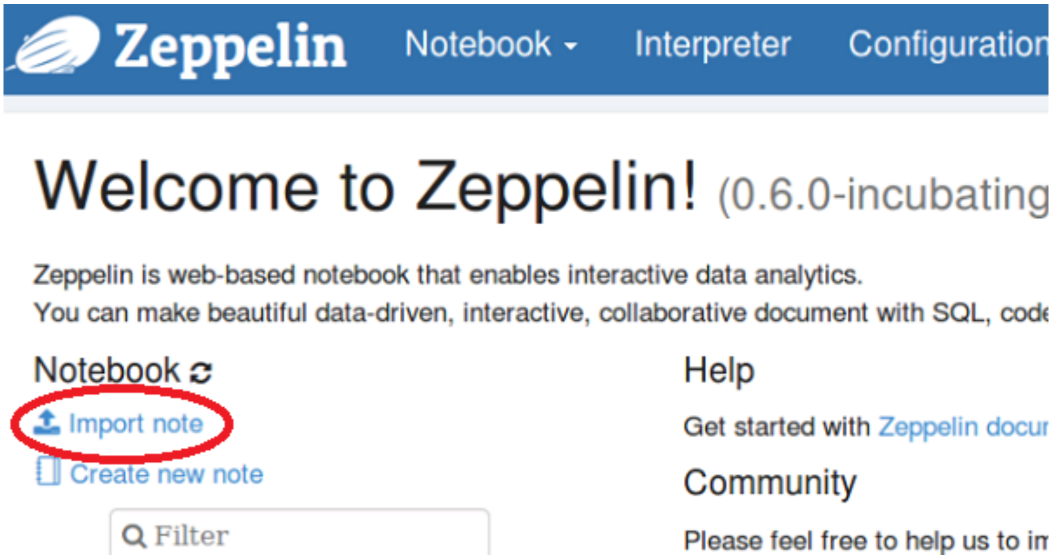



Data Visualization

Maximum Balance: 2500

age	balance	marital
47	1,506	married
33	1	single
35	231	married
28	447	single
42	2	divorced
58	121	married
43	593	single
41	270	divorced
29	390	single

r. Import the copy of this note you made earlier and name the new note Data Visualization Imported. Confirm that the copy contains all original code and formatting.



Welcome to Zeppelin! (0.6.0-incubating

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code

Notebook

Import note

Create new note

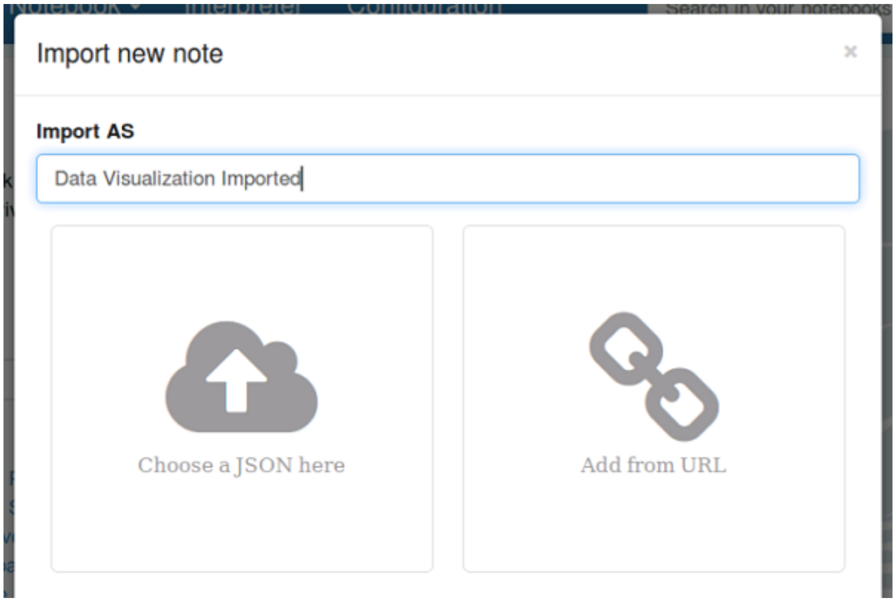
Filter

Help

Get started with Zeppelin docur

Community

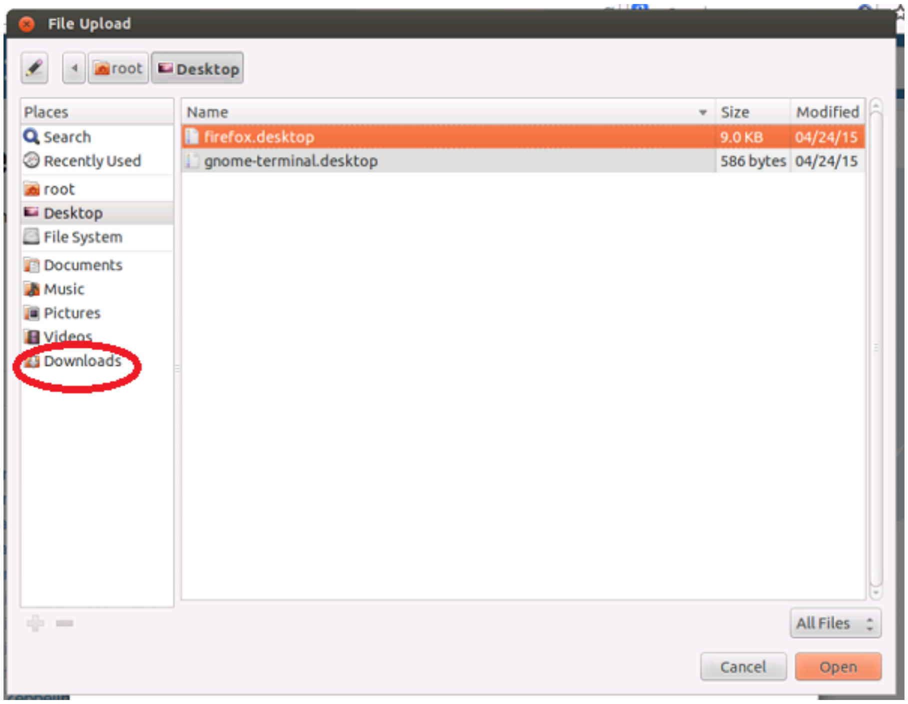
Please feel free to help us to in



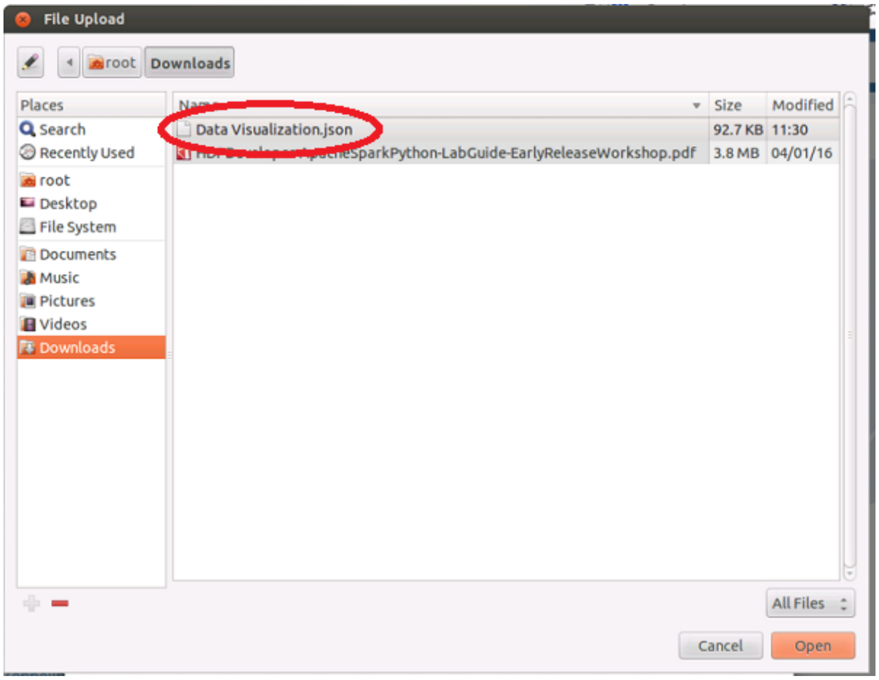
Import new note

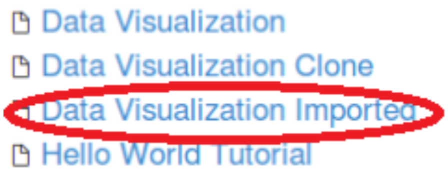
Import AS

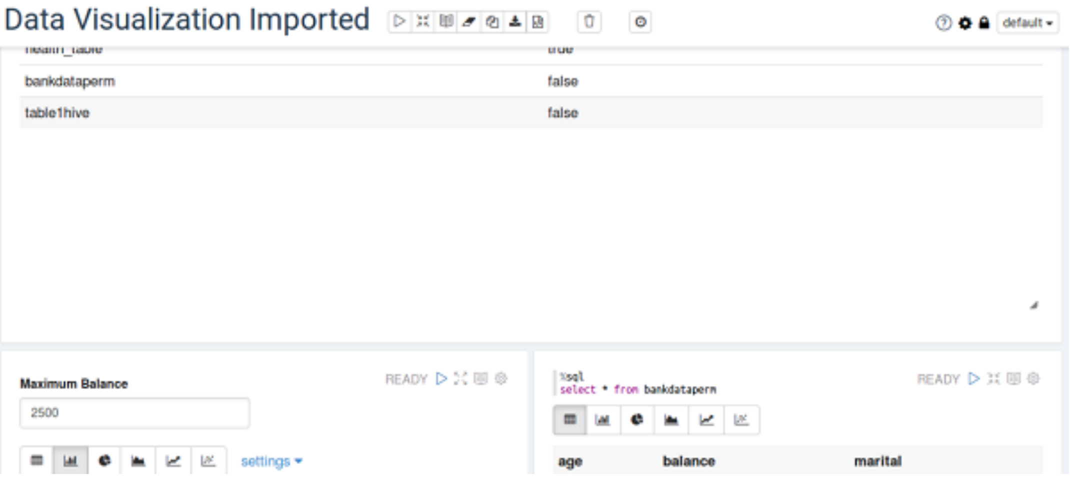
Data Visualization Imported



Name	Size	Modified
firefox.desktop	9.0 KB	04/24/15
gnome-terminal.desktop	586 bytes	04/24/15







health_table	true
bankdataperm	false
table1hive	false

```
%sql
select * from bankdataperm
```

age	balance	marital

Result

You have successfully created and manipulated Zeppelin visualizations, made them available for collaboration, and used Zeppelin to create a shareable report.

Lab 11: Job Monitoring (Python)

About This Lab

Objective:
Monitor Spark jobs using the Spark Application UI

Files Locations:
NA

Successful Outcome:
Monitor Spark jobs

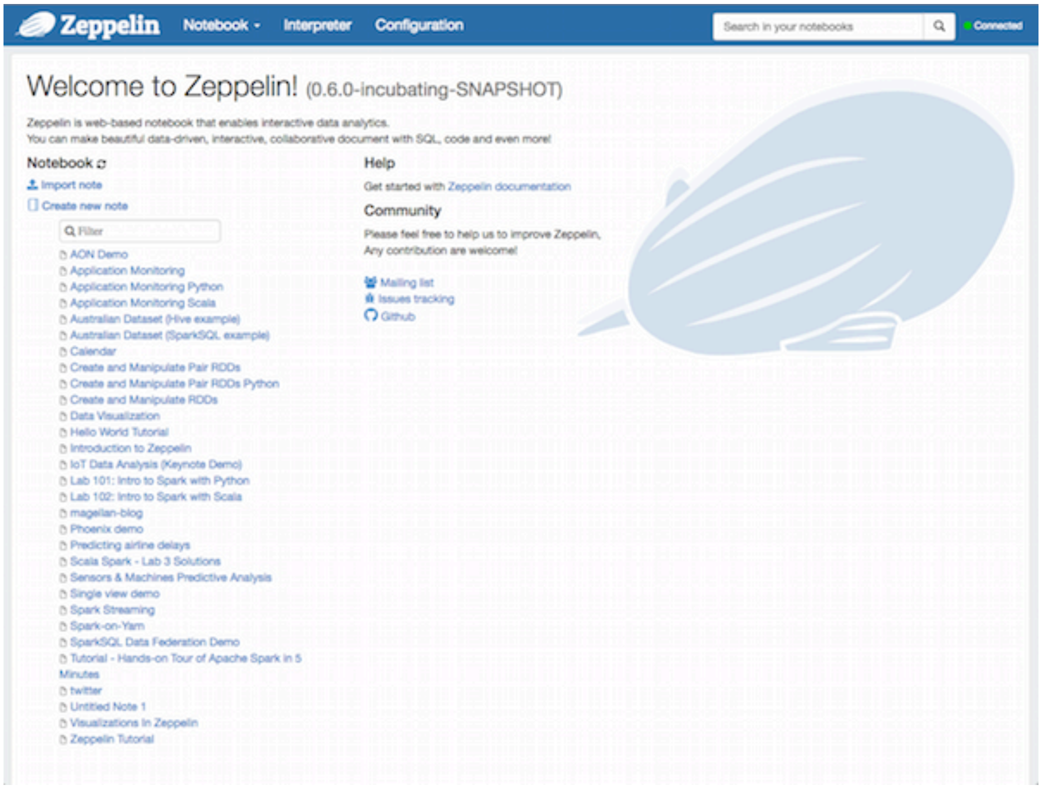
Lab Steps

Perform the following steps:

1. **Monitor a core RDD programming job.**

   a. Open the Firefox browser and access your Zeppelin notebook.

   http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

- Import note
- Create new note

Filter

- AON Demo
- Application Monitoring
- Application Monitoring Python
- Application Monitoring Scala
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Calendar
- Create and Manipulate Pair RDDs
- Create and Manipulate Pair RDDs Python
- Create and Manipulate RDDs
- Data Visualization
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Scala Spark - Lab 3 Solutions
- Sensors & Machines Predictive Analysis
- Single view demo
- Spark Streaming
- Spark-on-Yarn
- SparkSQL Data Federation Demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1
- Visualizations In Zeppelin
- Zeppelin Tutorial

Help

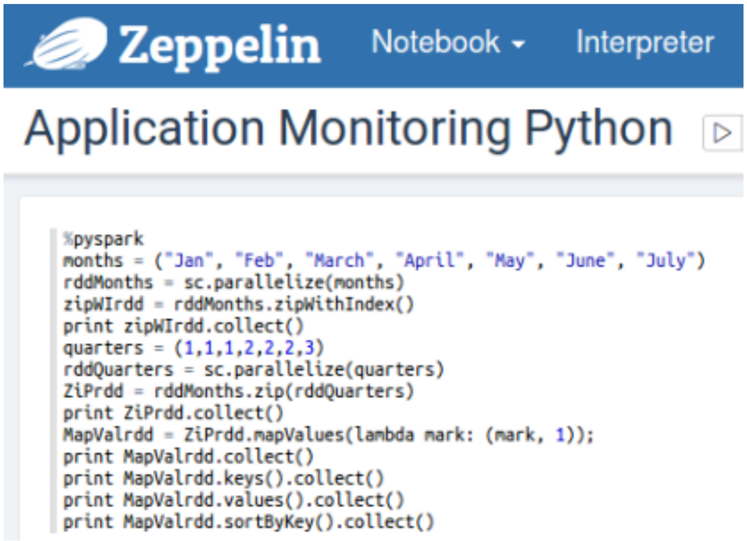
Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

- Mailing list
- Issues tracking
- Github

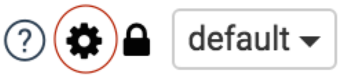
b. From the home page, select the Application Monitoring Python Note. This note has prebuilt code that we will run to generate Spark job activity.

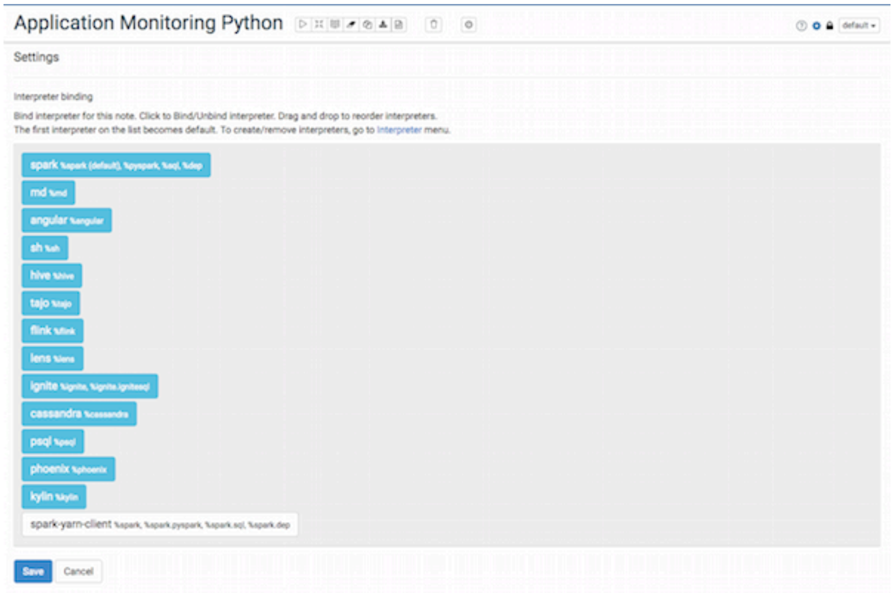


Application Monitoring Python

```
%pyspark
months = ("Jan", "Feb", "March", "April", "May", "June", "July")
rddMonths = sc.parallelize(months)
zipWIrdd = rddMonths.zipWithIndex()
print zipWIrdd.collect()
quarters = (1,1,1,2,2,2,3)
rddQuarters = sc.parallelize(quarters)
ZiPrdd = rddMonths.zip(rddQuarters)
print ZiPrdd.collect()
MapValrdd = ZiPrdd.mapValues(lambda mark: (mark, 1));
print MapValrdd.collect()
print MapValrdd.keys().collect()
print MapValrdd.values().collect()
print MapValrdd.sortByKey().collect()
```

c. At the top right click on the gear icon to change interpreter binding. Your administrator has enabled an interpreter called "**spark yarn-client**" which is configured for the HDP cluster you are using. Drag it to the top of the list of interpreters, and click the Save button.



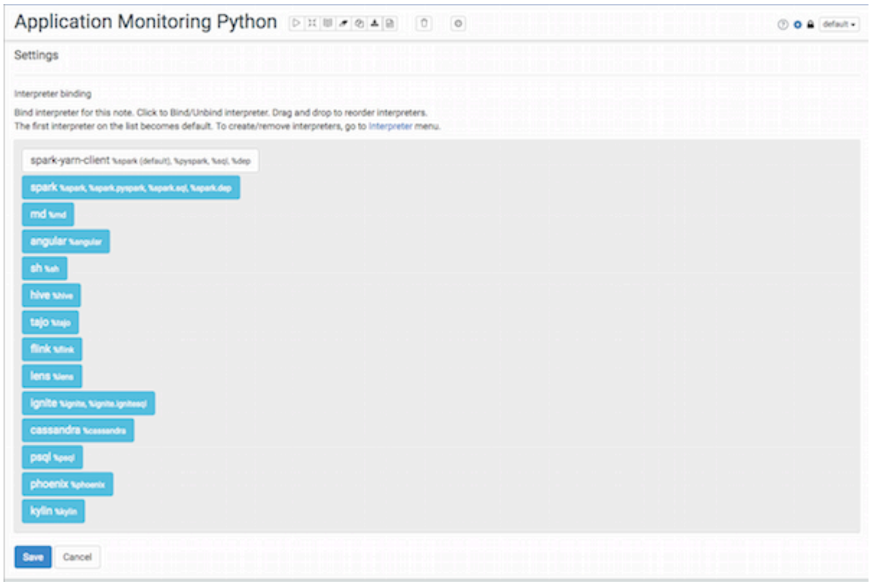


Settings

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep



Application Monitoring Python

Settings

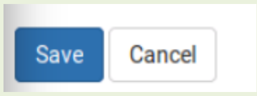
Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

- spark-yarn-client %spark (default), %pyspark, %sql, %dep
- spark %spark, %spark.pyspark, %spark.sql, %spark.dep
- md %md
- angular %angular
- sh %sh
- hive %hive
- tajo %tajo
- flink %flink
- lens %lens
- ignite %ignite, %ignite.ignitesql
- cassandra %cassandra
- psql %psql
- phoenix %phoenix
- kylin %kylin

NOTE:

The first interpreter on the list is treated as the default interpreter. Scroll down to find the Save button.

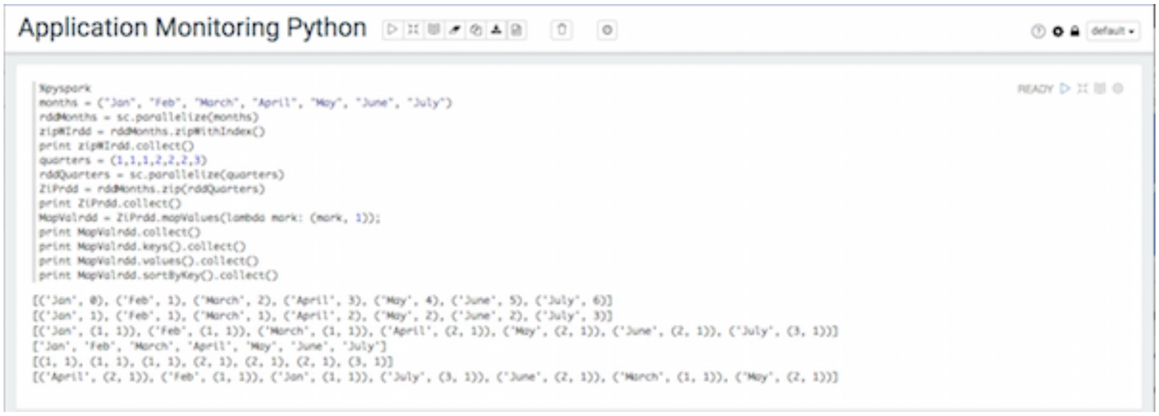


d. Now run the code by hitting Play button ▷ or by pressing ***Shift + Enter***.

NOTE:

The below code is for reference purposes and has already been placed in the note.

```
%pyspark
months = ("Jan", "Feb", "March", "April", "May", "June", "July")
rddMonths = sc.parallelize(months)
zipWIrdd = rddMonths.zipWithIndex()
print zipWIrdd.collect()
quarters = (1,1,1,2,2,2,3)
rddQuarters = sc.parallelize(quarters)
ZiPrdd = rddMonths.zip(rddQuarters)
print ZiPrdd.collect()
MapValrdd = ZiPrdd.mapValues(lambda mark: (mark, 1));
print MapValrdd.collect()
print MapValrdd.keys().collect()
print MapValrdd.values().collect()
print MapValrdd.sortByKey().collect()
```

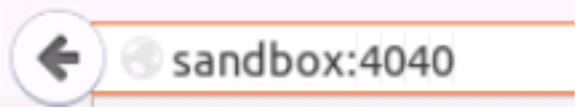


Application Monitoring Python

```
%pyspark
months = ("Jan", "Feb", "March", "April", "May", "June", "July")
rddMonths = sc.parallelize(months)
zipWIrdd = rddMonths.zipWithIndex()
print zipWIrdd.collect()
quarters = (1,1,1,2,2,2,3)
rddQuarters = sc.parallelize(quarters)
ZiPrdd = rddMonths.zip(rddQuarters)
print ZiPrdd.collect()
MapValrdd = ZiPrdd.mapValues(lambda mark: (mark, 1));
print MapValrdd.collect()
print MapValrdd.keys().collect()
print MapValrdd.values().collect()
print MapValrdd.sortByKey().collect()
```

```
[('Jan', 0), ('Feb', 1), ('March', 2), ('April', 3), ('May', 4), ('June', 5), ('July', 6)]
[('Jan', 1), ('Feb', 1), ('March', 1), ('April', 2), ('May', 2), ('June', 2), ('July', 3)]
[('Jan', (1, 1)), ('Feb', (1, 1)), ('March', (1, 1)), ('April', (2, 1)), ('May', (2, 1)), ('June', (2, 1)), ('July', (3, 1))]
['Jan', 'Feb', 'March', 'April', 'May', 'June', 'July']
[(1, 1), (1, 1), (1, 1), (2, 1), (2, 1), (2, 1), (3, 1)]
[('April', (2, 1)), ('Feb', (1, 1)), ('Jan', (1, 1)), ('July', (3, 1)), ('June', (2, 1)), ('March', (1, 1)), ('May', (2, 1))]
```

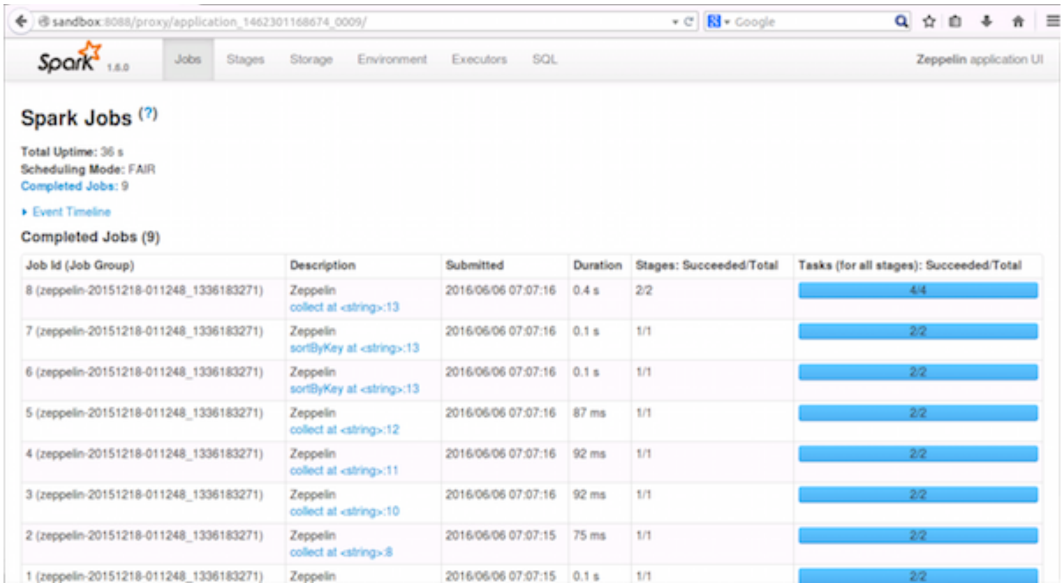
e. Open a new tab on the Firefox browser and enter the following URL to view the Spark Application UI:



sandbox:4040

NOTE:

http://sandbox:4040/ will work only once the job is submitted.



sandbox:8088/proxy/application_1462301168674_0009/

Spark Jobs (?)

Total Uptime: 36 s
Scheduling Mode: FAIR
Completed Jobs: 9

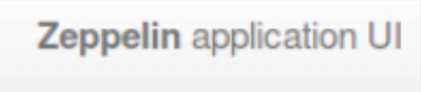
▸ Event Timeline

Completed Jobs (9)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:13	2016/06/06 07:07:16	0.4 s	2/2	4/4
7 (zeppelin-20151218-011248_1336183271)	Zeppelin sortByKey at <string>:13	2016/06/06 07:07:16	0.1 s	1/1	2/2
6 (zeppelin-20151218-011248_1336183271)	Zeppelin sortByKey at <string>:13	2016/06/06 07:07:16	0.1 s	1/1	2/2
5 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:12	2016/06/06 07:07:16	87 ms	1/1	2/2
4 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:11	2016/06/06 07:07:16	92 ms	1/1	2/2
3 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:10	2016/06/06 07:07:16	92 ms	1/1	2/2
2 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:8	2016/06/06 07:07:15	75 ms	1/1	2/2
1 (zeppelin-20151218-011248_1336183271)	Zeppelin	2016/06/06 07:07:15	0.1 s	1/1	2/2

NOTE:

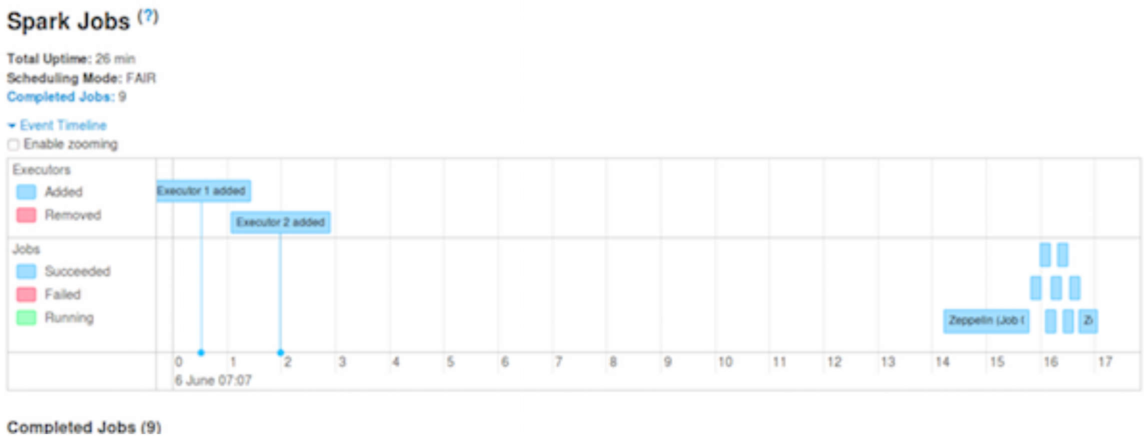
the URL http://sandbox:4040/ has been redirected to http://sandbox:8088/proxy/application_ID. Port 8088 belongs to the job history server for various applications that run on YARN. Here our application is “Zeppelin application UI” as noted in the top-right corner of the window.



Zeppelin application UI

f. SPARK APPLICATION UI SCAVENGER HUNT!

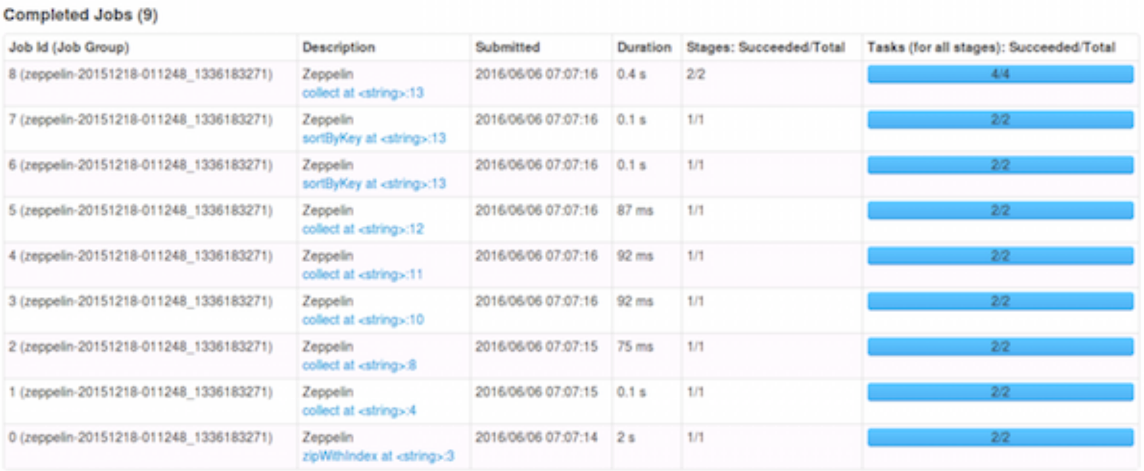
Look at the various aspects of the jobs that were run as part of the code being executed in the step above. Try to locate the following screens (the details of your environment may differ from the details shown):



Spark Jobs (?)

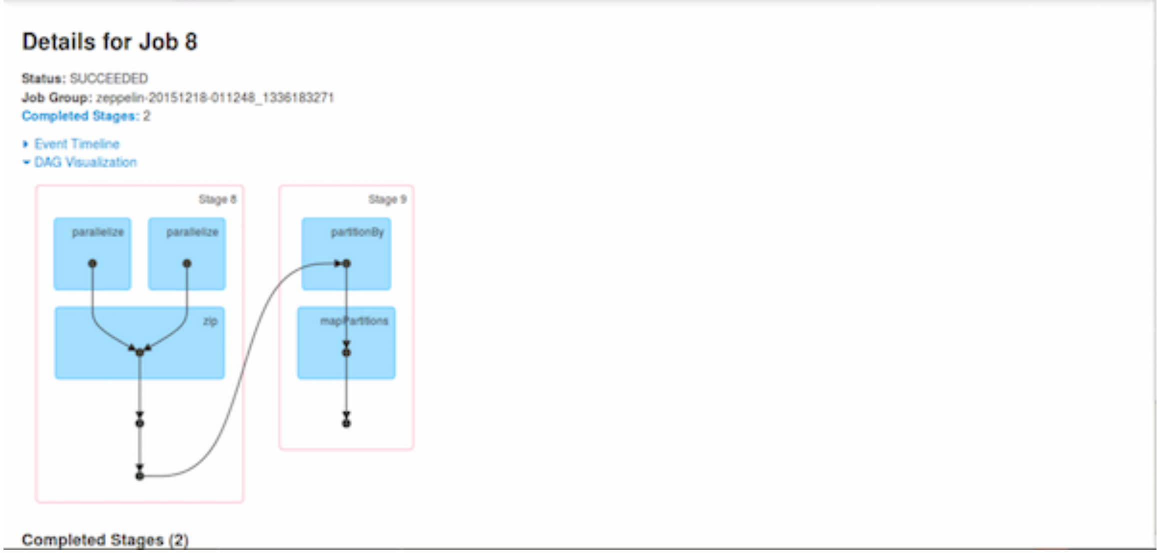
Total Uptime: 26 min
Scheduling Mode: FAIR
Completed Jobs: 9

Completed Jobs (9)



Completed Jobs (9)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:13	2016/06/06 07:07:16	0.4 s	2/2	4/4
7 (zeppelin-20151218-011248_1336183271)	Zeppelin sortByKey at <string>:13	2016/06/06 07:07:16	0.1 s	1/1	2/2
6 (zeppelin-20151218-011248_1336183271)	Zeppelin sortByKey at <string>:13	2016/06/06 07:07:16	0.1 s	1/1	2/2
5 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:12	2016/06/06 07:07:16	87 ms	1/1	2/2
4 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:11	2016/06/06 07:07:16	92 ms	1/1	2/2
3 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:10	2016/06/06 07:07:16	92 ms	1/1	2/2
2 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:8	2016/06/06 07:07:15	75 ms	1/1	2/2
1 (zeppelin-20151218-011248_1336183271)	Zeppelin collect at <string>:4	2016/06/06 07:07:15	0.1 s	1/1	2/2
0 (zeppelin-20151218-011248_1336183271)	Zeppelin zipWithIndex at <string>:3	2016/06/06 07:07:14	2 s	1/1	2/2

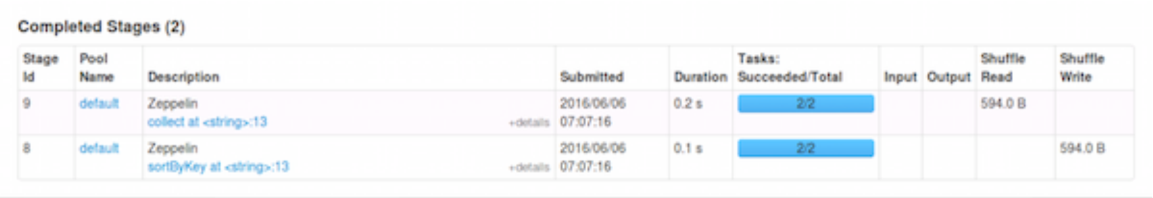


Details for Job 8

Status: SUCCEEDED
Job Group: zeppelin-20151218-011248_1336183271
Completed Stages: 2

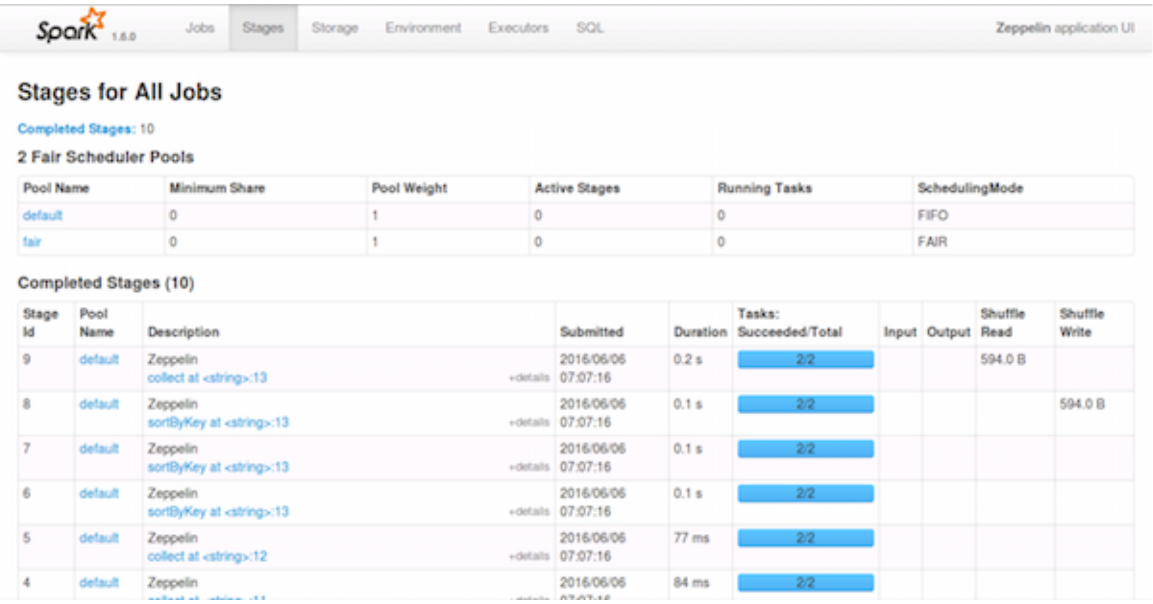
▸ Event Timeline
▾ DAG Visualization

Completed Stages (2)



Completed Stages (2)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	default	Zeppelin collect at <string>:13 +details	2016/06/06 07:07:16	0.2 s	2/2			594.0 B	
8	default	Zeppelin sortByKey at <string>:13 +details	2016/06/06 07:07:16	0.1 s	2/2				594.0 B



Stages for All Jobs

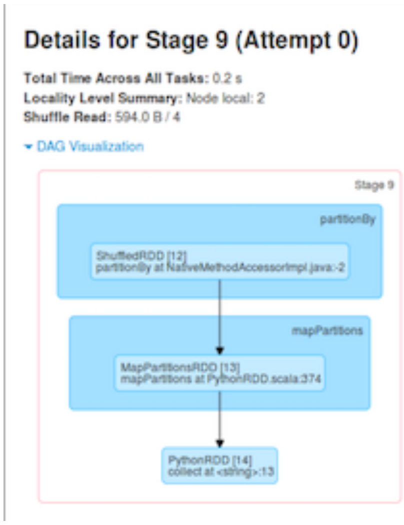
Completed Stages: 10

2 Fair Scheduler Pools

Pool Name	Minimum Share	Pool Weight	Active Stages	Running Tasks	SchedulingMode
default	0	1	0	0	FIFO
fair	0	1	0	0	FAIR

Completed Stages (10)

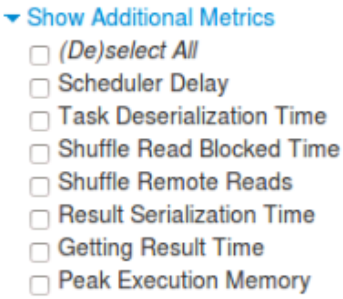
Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	default	Zeppelin collect at <string>:13 +details	2016/06/06 07:07:16	0.2 s	2/2			594.0 B	
8	default	Zeppelin sortByKey at <string>:13 +details	2016/06/06 07:07:16	0.1 s	2/2				594.0 B
7	default	Zeppelin sortByKey at <string>:13 +details	2016/06/06 07:07:16	0.1 s	2/2				
6	default	Zeppelin sortByKey at <string>:13 +details	2016/06/06 07:07:16	0.1 s	2/2				
5	default	Zeppelin collect at <string>:12 +details	2016/06/06 07:07:16	77 ms	2/2				
4	default	Zeppelin collect at <string>:11 +details	2016/06/06 07:07:16	84 ms	2/2				



Details for Stage 9 (Attempt 0)

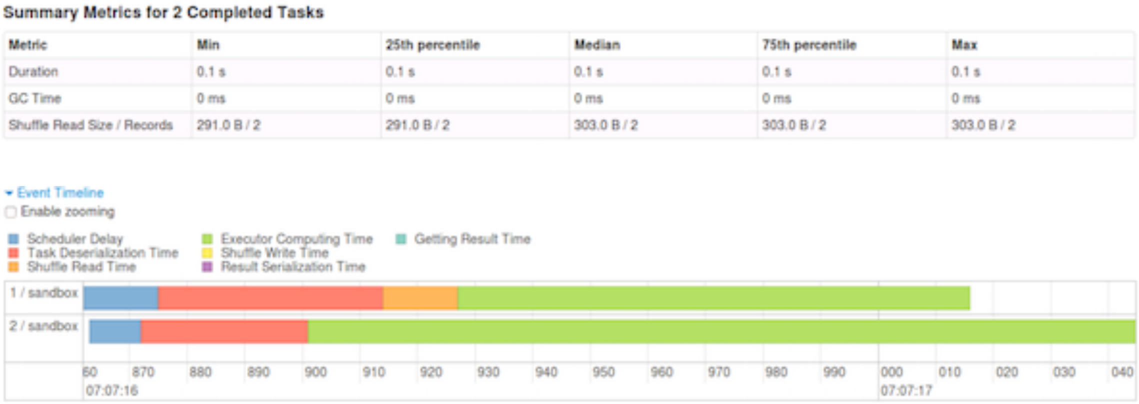
Total Time Across All Tasks: 0.2 s
Locality Level Summary: Node local: 2
Shuffle Read: 594.0 B / 4

When you get to the Show Additional Metrics link, try reading about and selecting additional metrics and view the information they provide. How might this be useful in troubleshooting application performance problems?



Show Additional Metrics

- □ *(De)select All*
- □ Scheduler Delay
- □ Task Deserialization Time
- □ Shuffle Read Blocked Time
- □ Shuffle Remote Reads
- □ Result Serialization Time
- □ Getting Result Time
- □ Peak Execution Memory



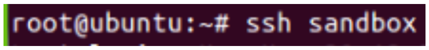
Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.1 s	0.1 s	0.1 s	0.1 s	0.1 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	291.0 B / 2	291.0 B / 2	303.0 B / 2	303.0 B / 2	303.0 B / 2

2. Monitor a Spark Streaming job.

a. Open a terminal window and SSH into sandbox.

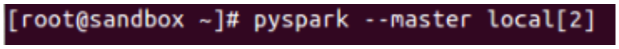
```
# ssh sandbox
```



```
root@ubuntu:~# ssh sandbox
```

b. Start a new REPL specifying the local machine as the master and allocate two cores for the streaming application.

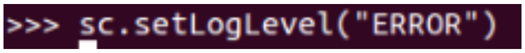
```
# pyspark --master local[2]
```



```
[root@sandbox ~]# pyspark --master local[2]
```

c. Set the log level to ERROR to avoid screen clutter while running the streaming application.

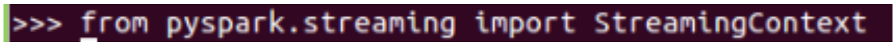
```
>>> sc.setLogLevel("ERROR")
```



```
>>> sc.setLogLevel("ERROR")
```

d. Import the streaming library.

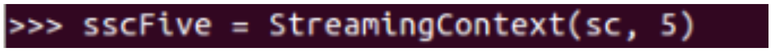
```
>>> from pyspark.streaming import StreamingContext
```



```
>>> from pyspark.streaming import StreamingContext
```

e. Create a streaming context with a five-second batch duration.

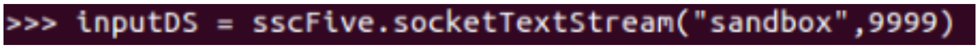
```
>>> sscFive = StreamingContext(sc, 5)
```



```
>>> sscFive = StreamingContext(sc, 5)
```

f. Create a DStream using `socketTestStream()` to the system named “sandbox” on port 9999.

```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```



```
>>> inputDS = sscFive.socketTextStream("sandbox",9999)
```

g. Print out the output to the terminal window.

```
>>> inputDS.pprint()
```

```
>>> inputDS.pprint()
```

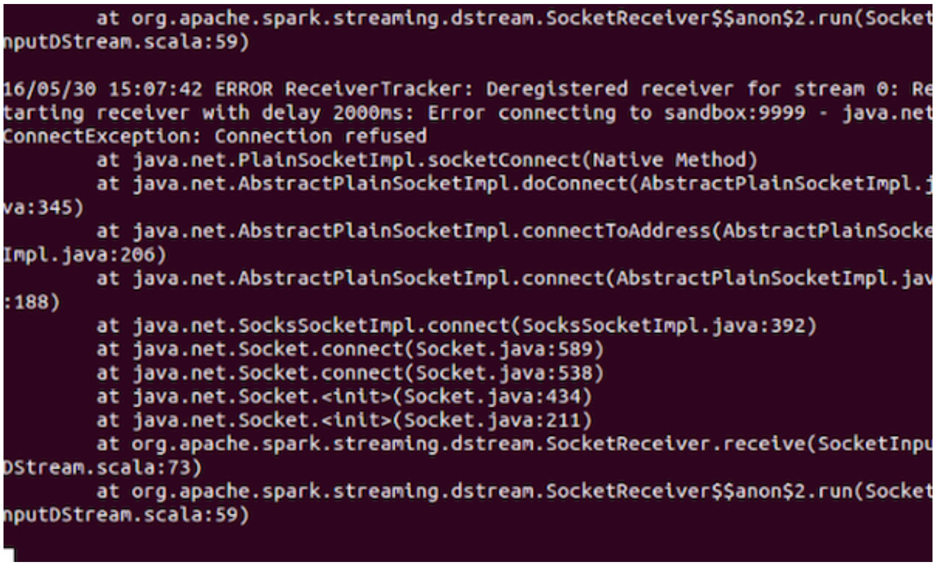
h. Start the streaming application. Note that only new files will be streamed, so any files that existed at application launch will not be streamed.

```
>>> sscFive.start()
```

```
>>> sscFive.start()
```

NOTE:

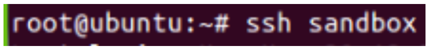
An error will appear when the application starts because the application is waiting for an input connection.



```
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)

16/05/30 15:07:42 ERROR ReceiverTracker: Deregistered receiver for stream 0: Re
tarting receiver with delay 2000ms: Error connecting to sandbox:9999 - java.net
ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.j
va:345)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocke
Impl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.jav
:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:211)
        at org.apache.spark.streaming.dstream.SocketReceiver.receive(SocketInpu
DStream.scala:73)
        at org.apache.spark.streaming.dstream.SocketReceiver$$anon$2.run(Socket
nputDStream.scala:59)
```

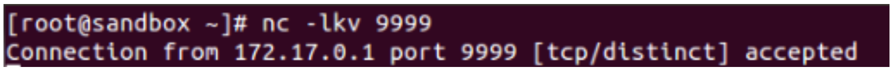
i. In a second terminal window SSH to sandbox and use the netcat utility to create a connection to port 9999.

```
# ssh sandbox
```



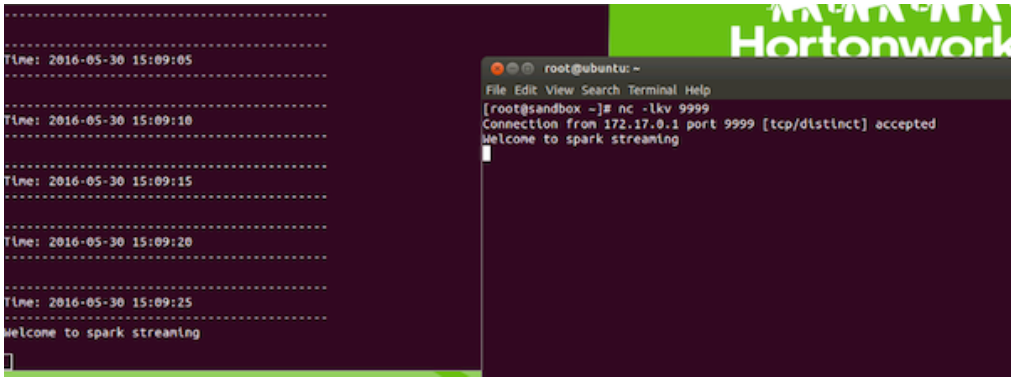
```
root@ubuntu:~# ssh sandbox
```

```
# nc -lkv 9999
```



```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
```

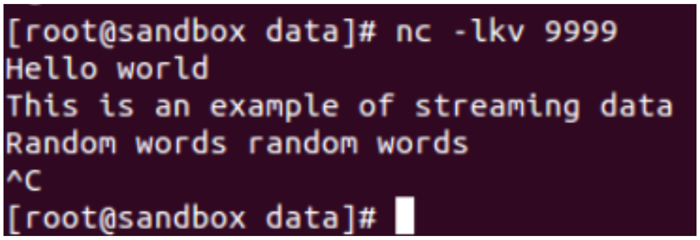
j. Start typing words separated by space, hit ***Enter*** occasionally to submit them. Observe what happens in the streaming terminal window a few seconds after hitting ***Enter***.



```
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:05
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:10
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:15
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:20
-------------------------------------------

-------------------------------------------
Time: 2016-05-30 15:09:25
-------------------------------------------
Welcome to spark streaming
```

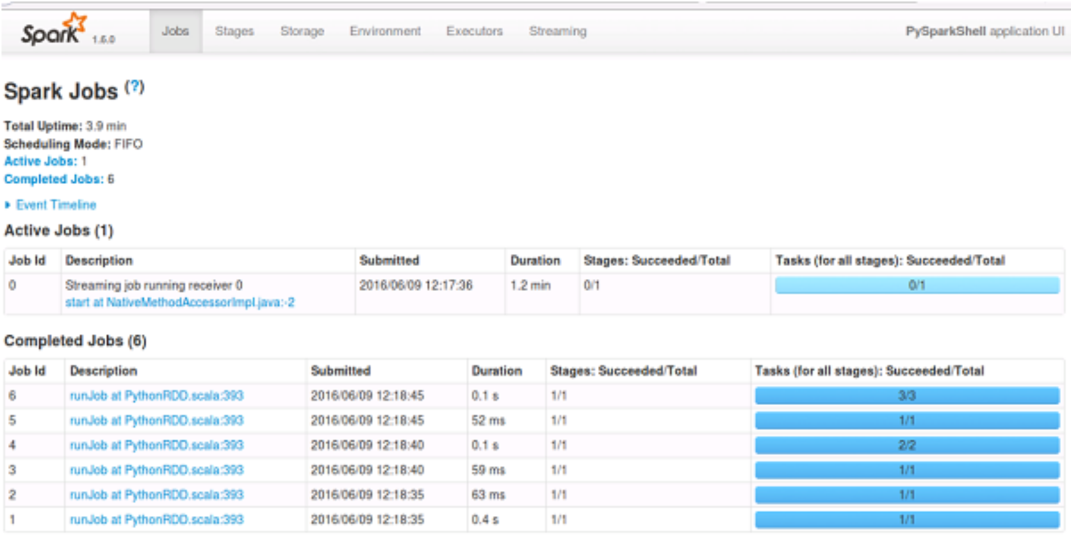
```
[root@sandbox ~]# nc -lkv 9999
Connection from 172.17.0.1 port 9999 [tcp/distinct] accepted
Welcome to spark streaming
```

k. Once you observe data being streamed on-screen in the first terminal window, use ***Ctrl + C*** (or ***Cmd + C*** if using a Mac) to exit netcat in the second terminal window.



```
[root@sandbox data]# nc -lkv 9999
Hello world
This is an example of streaming data
Random words random words
^C
[root@sandbox data]#
```

l. Since this is a new SparkContext instance, a new Spark Applications UI should now be available. Open a new FireFox tab and browse to the Streaming Application UI URL from before, but replace port 4040 with 4041:



Spark Jobs (?)

Total Uptime: 3.9 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 6

▸ Event Timeline

Active Jobs (1)

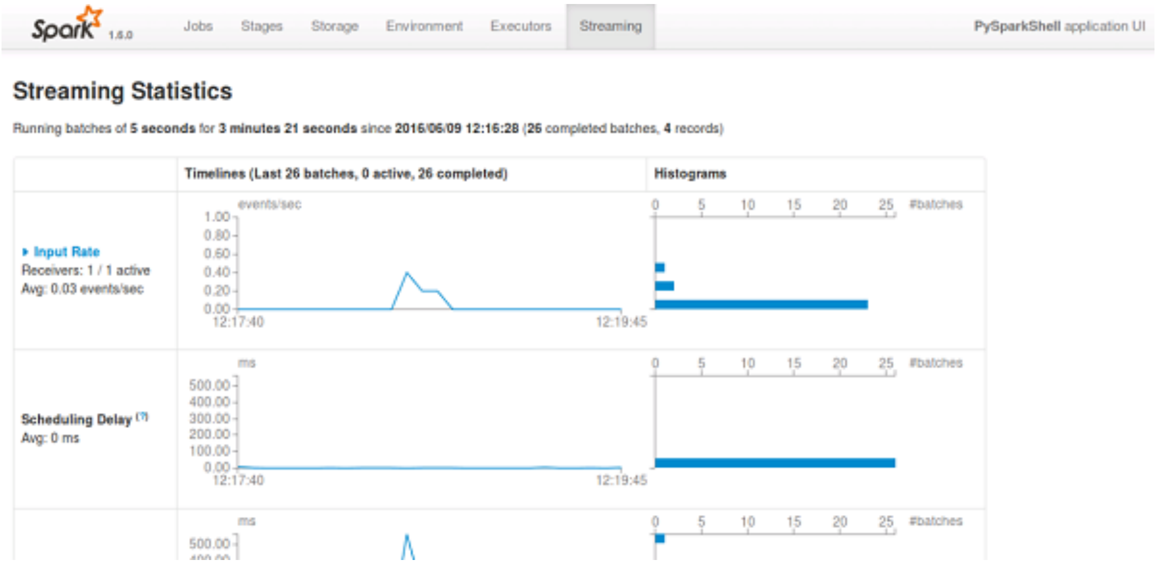
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	Streaming job running receiver 0 start at NativeMethodAccessorImpl.java:-2	2016/06/09 12:17:36	1.2 min	0/1	0/1

Completed Jobs (6)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	runJob at PythonRDD.scala:393	2016/06/09 12:18:45	0.1 s	1/1	3/3
5	runJob at PythonRDD.scala:393	2016/06/09 12:18:45	52 ms	1/1	1/1
4	runJob at PythonRDD.scala:393	2016/06/09 12:18:40	0.1 s	1/1	2/2
3	runJob at PythonRDD.scala:393	2016/06/09 12:18:40	59 ms	1/1	1/1
2	runJob at PythonRDD.scala:393	2016/06/09 12:18:35	63 ms	1/1	1/1
1	runJob at PythonRDD.scala:393	2016/06/09 12:18:35	0.4 s	1/1	1/1

m. SCAVENGER HUNT PART II

Look at the various aspects of the streaming jobs that were run as part of the code being executed in the steps above. Try to locate the following screens (the details of your environment may differ from the details shown):



Streaming Statistics

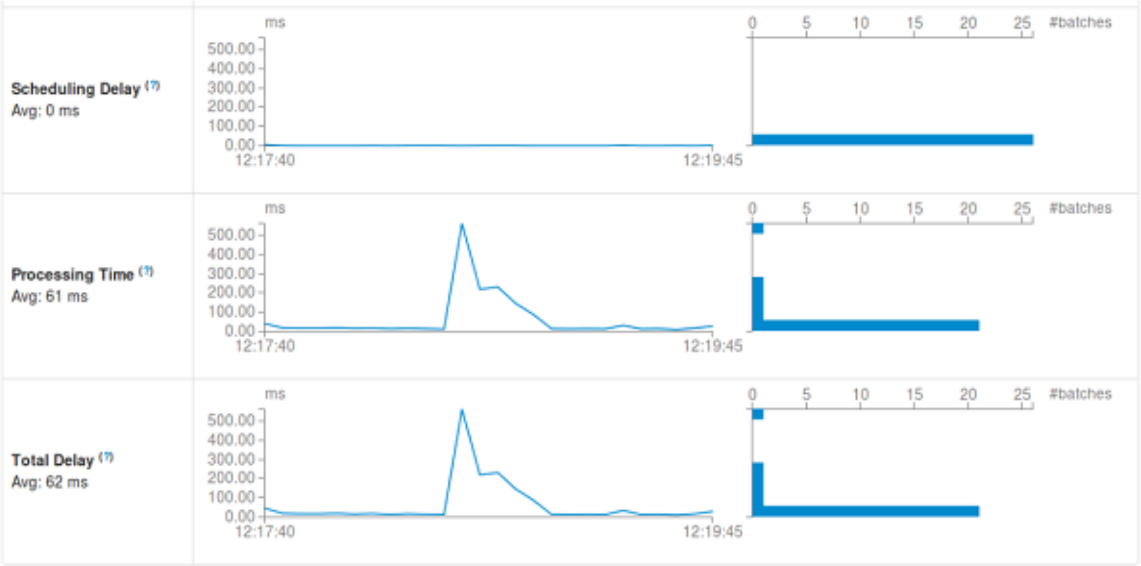
Running batches of **5 seconds** for **3 minutes 21 seconds** since **2016/06/09 12:16:28** (**26** completed batches, **4** records)

Timelines (Last 26 batches, 0 active, 26 completed)

Histograms

Input Rate
Receivers: 1 / 1 active
Avg: 0.03 events/sec

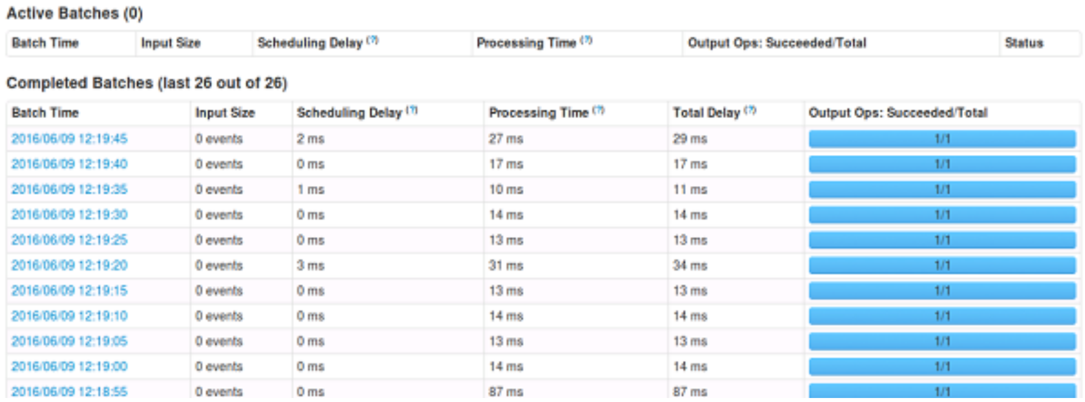
Scheduling Delay (?)
Avg: 0 ms



Scheduling Delay (?)
Avg: 0 ms

Processing Time (?)
Avg: 61 ms

Total Delay (?)
Avg: 62 ms

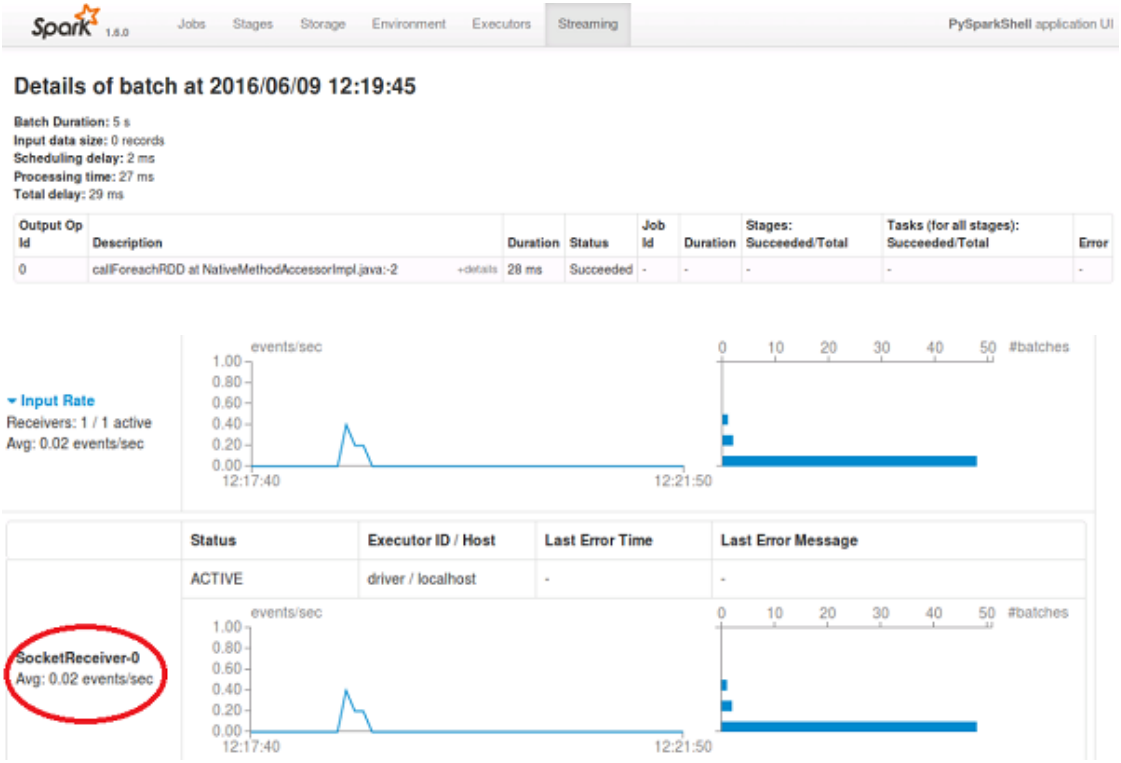


Active Batches (0)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status

Completed Batches (last 26 out of 26)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2016/06/09 12:19:45	0 events	2 ms	27 ms	29 ms	1/1
2016/06/09 12:19:40	0 events	0 ms	17 ms	17 ms	1/1
2016/06/09 12:19:35	0 events	1 ms	10 ms	11 ms	1/1
2016/06/09 12:19:30	0 events	0 ms	14 ms	14 ms	1/1
2016/06/09 12:19:25	0 events	0 ms	13 ms	13 ms	1/1
2016/06/09 12:19:20	0 events	3 ms	31 ms	34 ms	1/1
2016/06/09 12:19:15	0 events	0 ms	13 ms	13 ms	1/1
2016/06/09 12:19:10	0 events	0 ms	14 ms	14 ms	1/1
2016/06/09 12:19:05	0 events	0 ms	13 ms	13 ms	1/1
2016/06/09 12:19:00	0 events	0 ms	14 ms	14 ms	1/1
2016/06/09 12:18:55	0 events	0 ms	87 ms	87 ms	1/1



Details of batch at 2016/06/09 12:19:45

Batch Duration: 5 s
Input data size: 0 records
Scheduling delay: 2 ms
Processing time: 27 ms
Total delay: 29 ms

Output Op Id	Description	Duration	Status	Job Id	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total	Error
0	callForeachRDD at NativeMethodAccessorImpl.java:-2 +details	28 ms	Succeeded	-	-	-	-	-

▾ Input Rate
Receivers: 1 / 1 active
Avg: 0.02 events/sec

	Status	Executor ID / Host	Last Error Time	Last Error Message
	ACTIVE	driver / localhost	-	-

SocketReceiver-0
Avg: 0.02 events/sec

n. When you have located all of the required sections, go back to the first terminal window, stop the stream and exit the REPL. If the stream refreshes while you are typing, that will not affect the input. Simply continue to type the command and press **Enter**.

```
sc.stop()
exit()
```

Result

You have successfully monitored Spark core programming and Spark Streaming jobs using the Spark Application UI.

Lab12: Performance Tuning (Python)

About This Lab

Objective:
Practice performance tuning techniques

File Locations:
`/home/zeppelin/spark/data/`

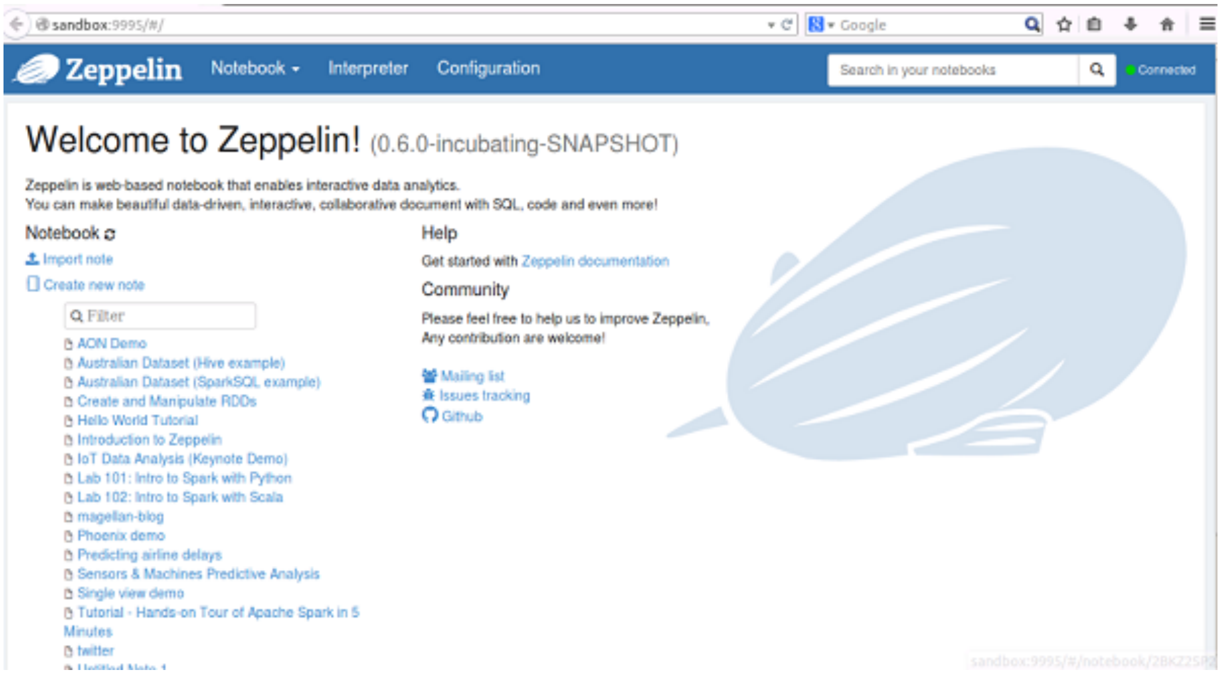
Successful Outcome:
Code performance tuning techniques from the lesson

Lab Steps

Perform the following steps:

1. **Practice using performance tuning techniques.**

   a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.
   http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

- Import note
- Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Create and Manipulate RDDs
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter

Help

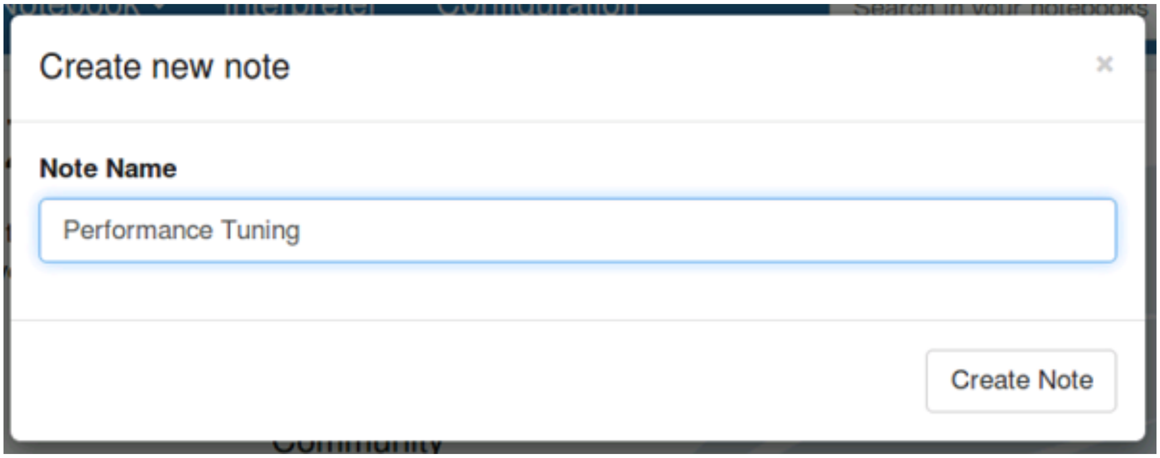
Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin.
Any contribution are welcome!

- Mailing list
- Issues tracking
- Github

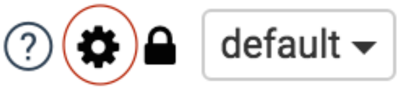
b. Click on Create new note. Name this note Performance Tuning.

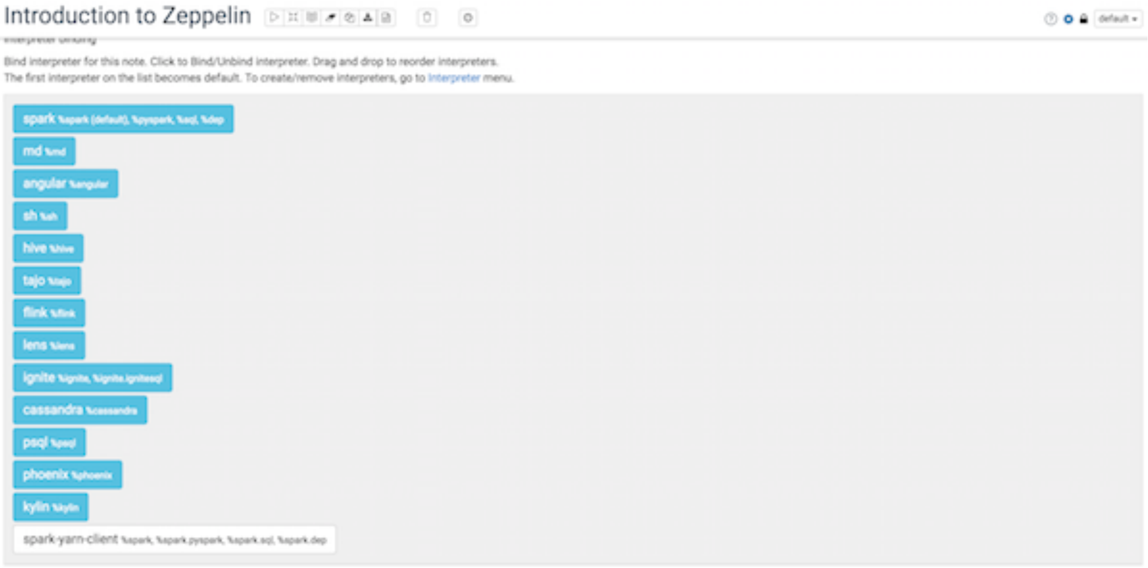


Create new note

Note Name: Performance Tuning

c. At the top right click on the gear icon to change interpreter binding.





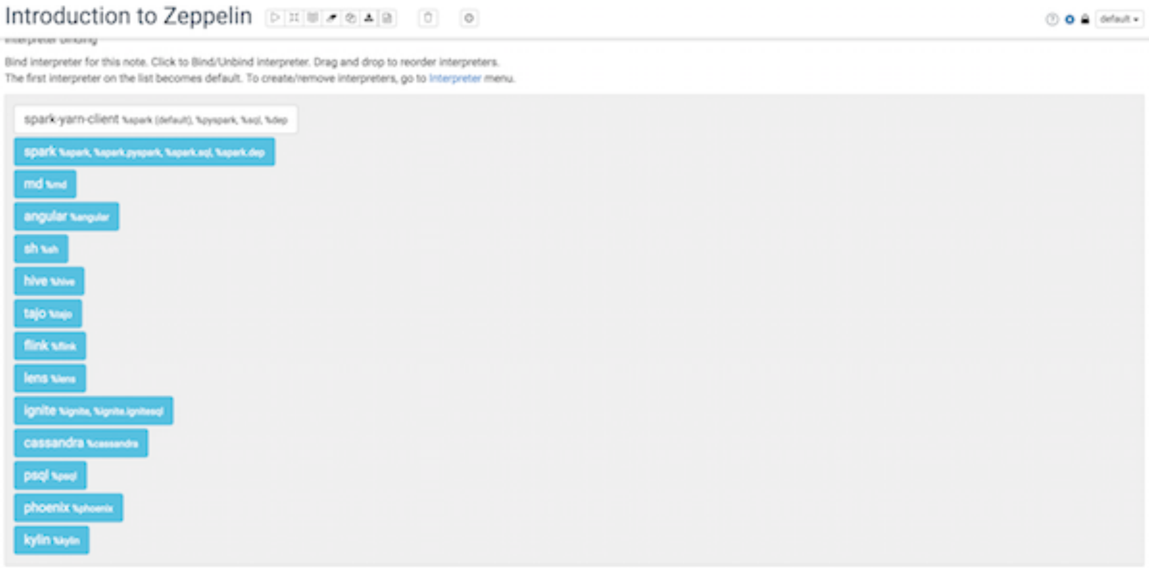
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

default
spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep

Drag the spark-yarn-client to the top and click save.



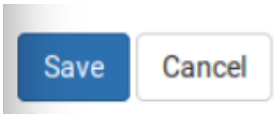
Introduction to Zeppelin

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

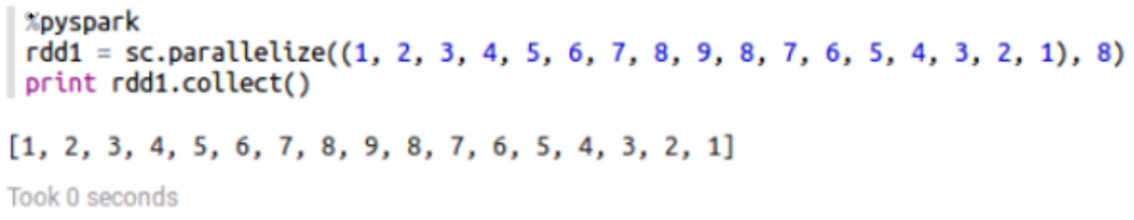
default
spark-yarn-client %spark (default), %pyspark, %sql, %dep
spark %spark, %spark.pyspark, %spark.sql, %spark.dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin

The first interpreter on the list becomes default.



d. Create an RDD named `rdd1` that contains a list of numbers one through nine, then back down to one again (17 elements total) and set it to eight partitions. Use `print` to confirm the RDD was created successfully.

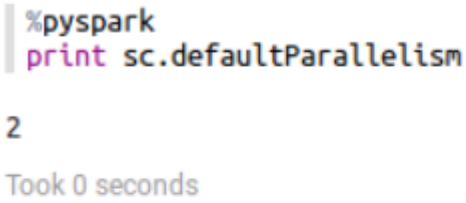
```
%pyspark

rdd1 = sc.parallelize((1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1), 8)

print rdd1.collect()
```



```
%pyspark
rdd1 = sc.parallelize((1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1), 8)
print rdd1.collect()

[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Took 0 seconds
```

e. View the default parallelism settings for your environment, and then verify that `rdd1` was partitioned with eight partitions instead of the default number.

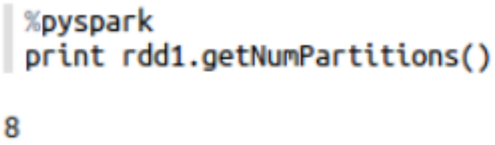
```
%pyspark

print sc.defaultParallelism
```



```
%pyspark
print sc.defaultParallelism

2
```

Took 0 seconds

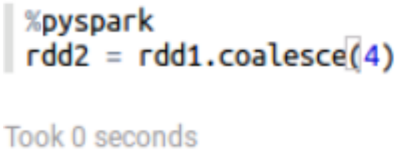
```
%pyspark

print rdd1.getNumPartitions()
```



```
%pyspark
print rdd1.getNumPartitions()

8
```

f. Create an RDD named `rdd2` that is a copy of `rdd1` but uses only four partitions. Verify that `rdd2` has only four partitions.

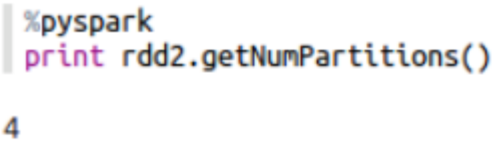
```
%pyspark

rdd2 = rdd1.coalesce(4)
```



```
%pyspark
rdd2 = rdd1.coalesce(4)
```

Took 0 seconds

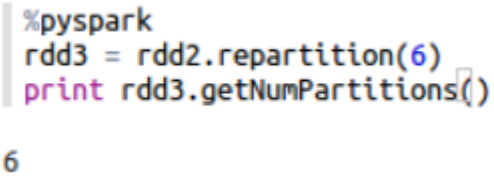
```
%pyspark

print rdd2.getNumPartitions()
```



```
%pyspark
print rdd2.getNumPartitions()

4
```

g. Create an RDD named `rdd3` that is a copy of `rdd2` but expands the number of partitions from four to six. Verify that `rdd3` has six partitions.

```
%pyspark

rdd3 = rdd2.repartition(6)

print rdd3.getNumPartitions()
```

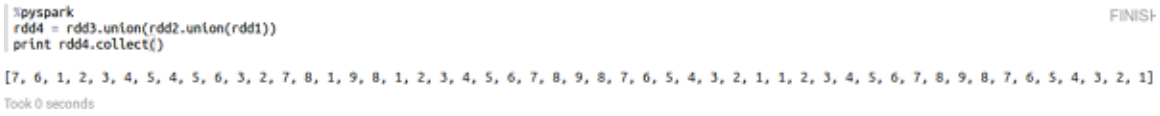


```
%pyspark
rdd3 = rdd2.repartition(6)
print rdd3.getNumPartitions()
```

```
6
```

h. Create an RDD named `rdd4` that contains a larger set of data by combining `rdd3`, `rdd2`, and `rdd1`. The view this list of 51 numbers.

```
%pyspark

rdd4 = rdd3.union(rdd2.union(rdd1))

print rdd4.collect()
```

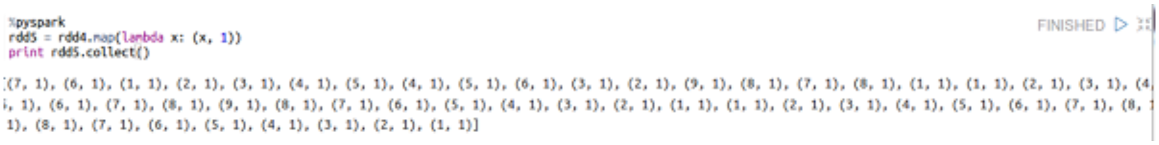


```
%pyspark
rdd4 = rdd3.union(rdd2.union(rdd1))
print rdd4.collect()
```

```
[7, 6, 1, 2, 3, 4, 5, 4, 5, 6, 3, 2, 7, 8, 1, 9, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Took 0 seconds
```

i. Create an RDD named `rdd5` that turns this list into a Pair RDD using the existing numbers as keys and assign each key a value of one. View `rdd5` to confirm successful operation.

```
%pyspark

rdd5 = rdd4.map(lambda x: (x, 1))

print rdd5.collect()
```

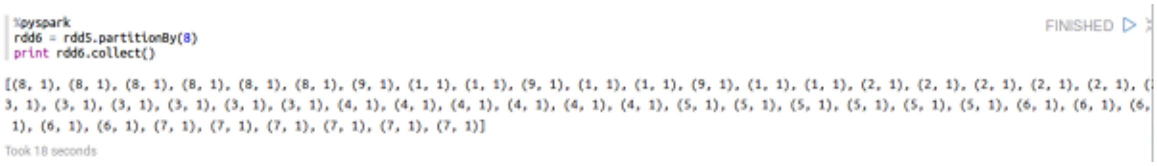


```
%pyspark
rdd5 = rdd4.map(lambda x: (x, 1))
print rdd5.collect()
```

```
[(7, 1), (6, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (4, 1), (5, 1), (6, 1), (3, 1), (2, 1), (9, 1), (8, 1), (7, 1), (8, 1), (1, 1), (1, 1), (2, 1), (3, 1), (4,
5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (8, 1), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1), (2, 1), (1, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1
1), (8, 1), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1), (2, 1), (1, 1)]
```

j. **Create an RDD named `rdd6` that uses `partitionBy()` to create eight hashed partitions from `rdd5`. View `rdd6` to confirm successful operation.**

```
%pyspark

rdd6 = rdd5.partitionBy(8)

print rdd6.collect()
```

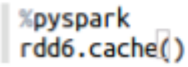


```
%pyspark
rdd6 = rdd5.partitionBy(8)
print rdd6.collect()
```

```
[(8, 1), (8, 1), (8, 1), (8, 1), (8, 1), (8, 1), (9, 1), (1, 1), (1, 1), (9, 1), (1, 1), (1, 1), (9, 1), (1, 1), (1, 1), (2, 1), (2, 1), (2, 1), (2, 1), (2, 1), (
3, 1), (3, 1), (3, 1), (3, 1), (3, 1), (3, 1), (4, 1), (4, 1), (4, 1), (4, 1), (4, 1), (4, 1), (5, 1), (5, 1), (5, 1), (5, 1), (5, 1), (5, 1), (6, 1), (6, 1), (6,
 1), (6, 1), (6, 1), (7, 1), (7, 1), (7, 1), (7, 1), (7, 1), (7, 1)]
Took 18 seconds
```

k. **Cache `rdd6` in memory so that it will be quickly available should we want to use the hash partitioning in a future operation.**

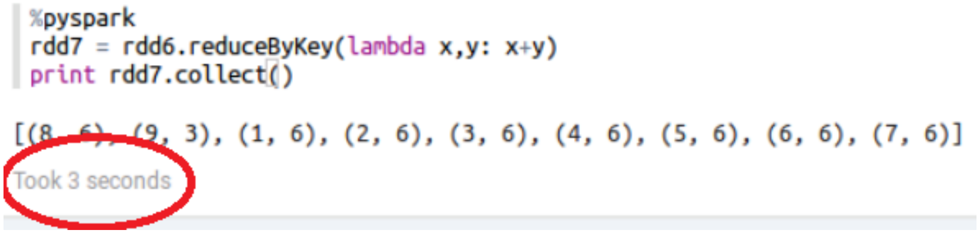
```
%pyspark

rdd6.cache()
```



```
%pyspark
rdd6.cache()
```

l. **Create a new RDD named `rdd7` that reduces `rdd6` by key. View the results, and pay attention to the time it took to generate it.**

```
%pyspark

rdd7 = rdd6.reduceByKey(lambda x,y: x+y)

print rdd7.collect()
```

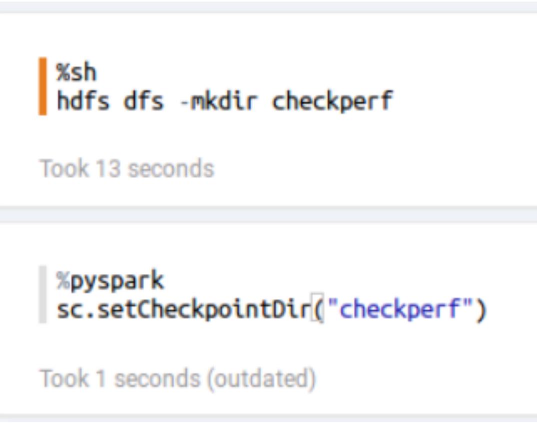


```
%pyspark
rdd7 = rdd6.reduceByKey(lambda x,y: x+y)
print rdd7.collect()

[(8, 6), (9, 3), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6)]
Took 3 seconds
```

m. Create a directory named `checkperf` in your HDFS home directory, then configure it as your checkpoint directory for Spark applications.

```
%sh

hdfs dfs -mkdir checkperf
```

```
%pyspark

sc.setCheckpointDir(“checkperf”)
```



```
%sh
hdfs dfs -mkdir checkperf
```

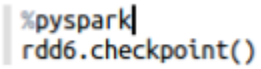
Took 13 seconds

```
%pyspark
sc.setCheckpointDir("checkperf")
```

Took 1 seconds (outdated)

n. Checkpoint `rdd6` so that future operations can use it as the starting point for lineage-tracking purposes.

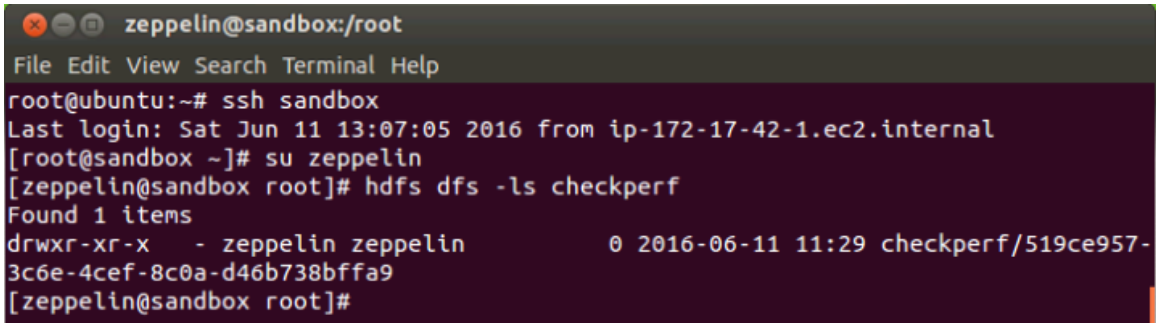
```
%pyspark

rdd6.checkpoint()
```



```
%pyspark
rdd6.checkpoint()
```

o. Open a terminal window and connect to sandbox using SSH. Switch to the zeppelin user. Then view the contents of the checkperf directory and confirm that a checkpoint file exists. Then exit the zeppelin user back to root.

```
# ssh sandbox

# su zeppelin

# hdfs dfs -ls checkperf

# exit
```



```
root@ubuntu:~# ssh sandbox
Last login: Sat Jun 11 13:07:05 2016 from ip-172-17-42-1.ec2.internal
[root@sandbox ~]# su zeppelin
[zeppelin@sandbox root]# hdfs dfs -ls checkperf
Found 1 items
drwxr-xr-x   - zeppelin zeppelin          0 2016-06-11 11:29 checkperf/519ce957-
3c6e-4cef-8c0a-d46b738bffa9
[zeppelin@sandbox root]#
```

p. Use broadcast variables to perform an operation. Code the following:

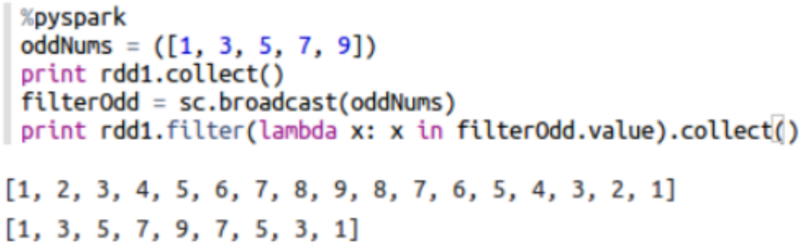
1. Create a variable named oddNums that contains a list of odd numbers 1-9.

2. Print the contents of rdd1 used at the beginning of the lab.

3. Create a broadcast variable named filterOdd that contains the values in oddNums.

4. Print the results of a filter operation where only numbers that appear in the filterOdd broadcast variable show up in the output.

```
%pyspark

oddNums = ([1, 3, 5, 7, 9])

print rdd1.collect()

filterOdd = sc.broadcast(oddNums)

print rdd1.filter(lambda x: x in filterOdd.value).collect()
```



```
%pyspark
oddNums = ([1, 3, 5, 7, 9])
print rdd1.collect()
filterOdd = sc.broadcast(oddNums)
print rdd1.filter(lambda x: x in filterOdd.value).collect()

[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[1, 3, 5, 7, 9, 7, 5, 3, 1]
```

Result

You have used several of the performance tuning tools and practices discussed in the lesson.

Lab13: Build and Submit Applications to YARN (Python)

About This Lab

Objective:
Apply programming knowledge into stand-alone applications submitted to a YARN cluster

File Locations:
NA

Successful Outcome:
Build and submit a cluster-mode application to YARN

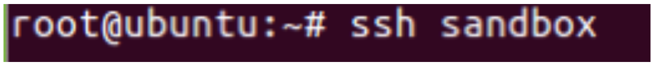
Lab Steps

Perform the following steps:

1. **Build and Submit a Spark RDD application**

   a. Open a terminal and use SSH to connect to sandbox:

```
# ssh sandbox
```



```
root@ubuntu:~# ssh sandbox
```

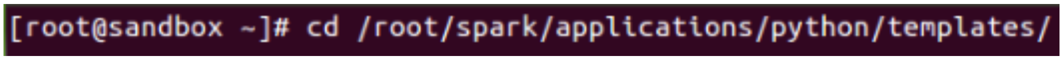
   b. **OPTIONAL:**
   If you have a favorite Linux text editor already, you may use it for the rest of the lab. If you are not already familiar with a Linux text editor, we recommend that you download and install `nano` – a small, simple to use editor that will be used for the commands and screenshots in this lab.

```
yum -y install nano
```

```
[root@sandbox ~]# yum -y install nano
```

   c. Navigate to `/root/spark/data/applications/python/templates/` and view the SparkRDD.py file.

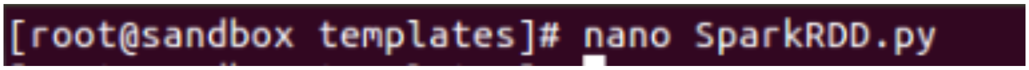
```
# cd /root/spark/applications/python/templates/
```



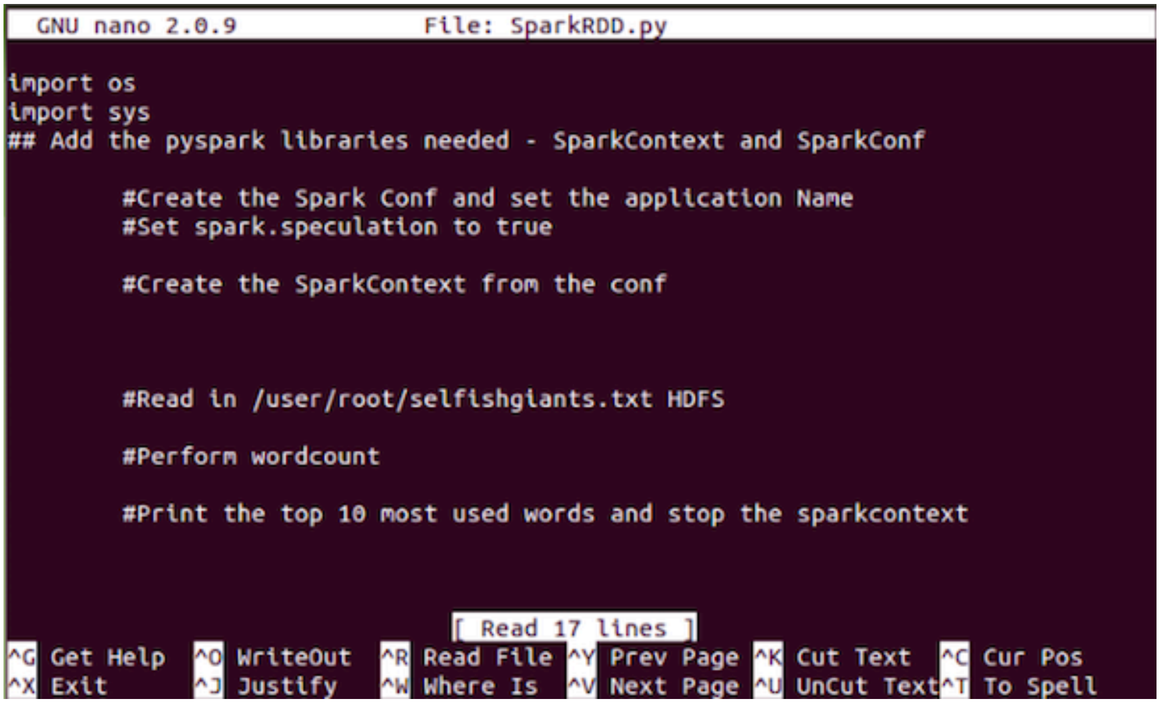
```
[root@sandbox ~]# cd /root/spark/applications/python/templates/
```

```
# nano SparkRDD.py
```

(Again, `vi` or another editor can also be used based on your preference.)

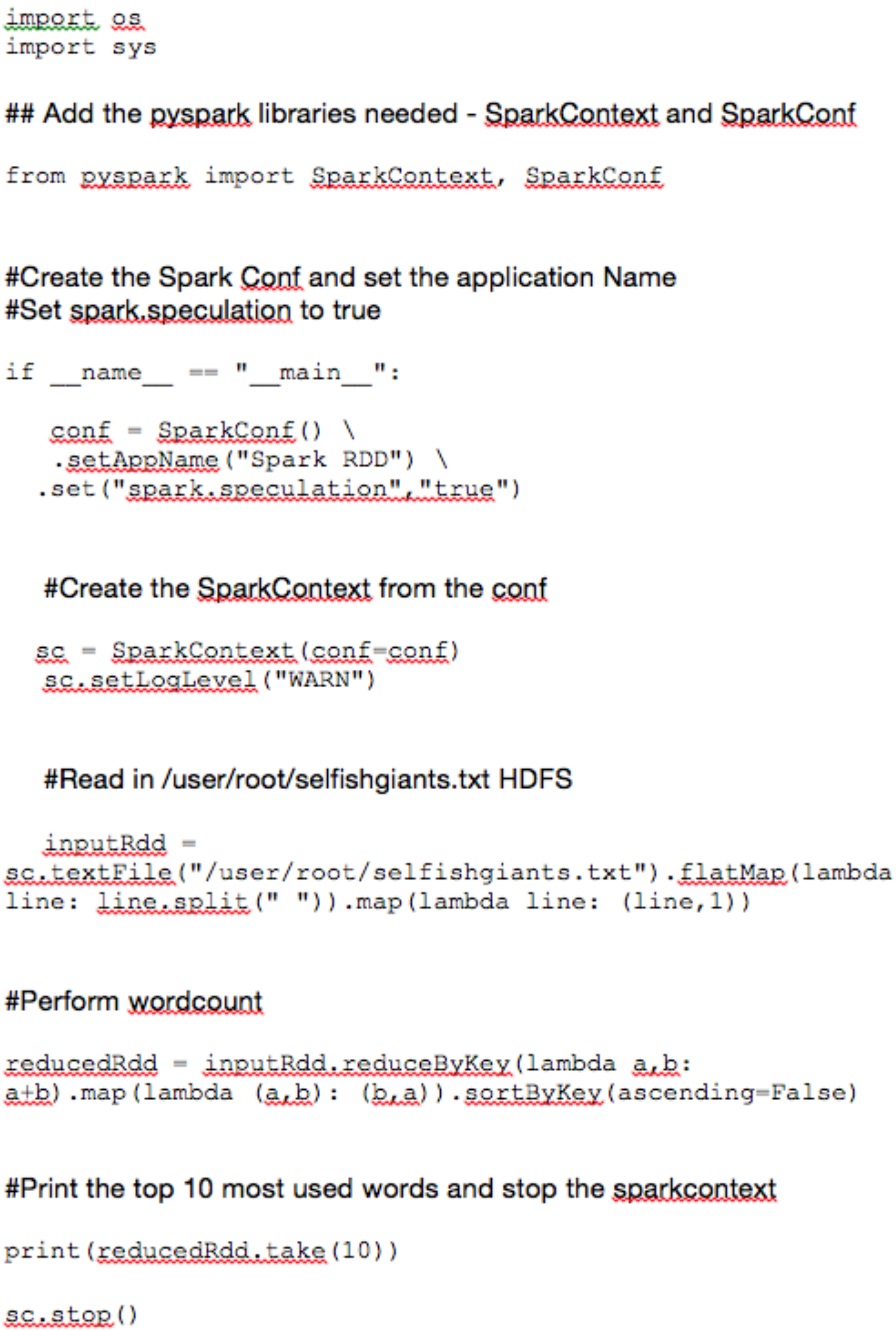


```
[root@sandbox templates]# nano SparkRDD.py
```



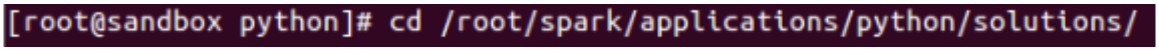
```
  GNU nano 2.0.9             File: SparkRDD.py

import os
import sys
## Add the pyspark libraries needed - SparkContext and SparkConf

        #Create the Spark Conf and set the application Name
        #Set spark.speculation to true

        #Create the SparkContext from the conf



        #Read in /user/root/selfishgiants.txt HDFS

        #Perform wordcount

        #Print the top 10 most used words and stop the sparkcontext



                                [ Read 17 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

d. The objective is to build an application based on this template and the comments posted on this template. You may try to do this on your own, or use the solution steps below:

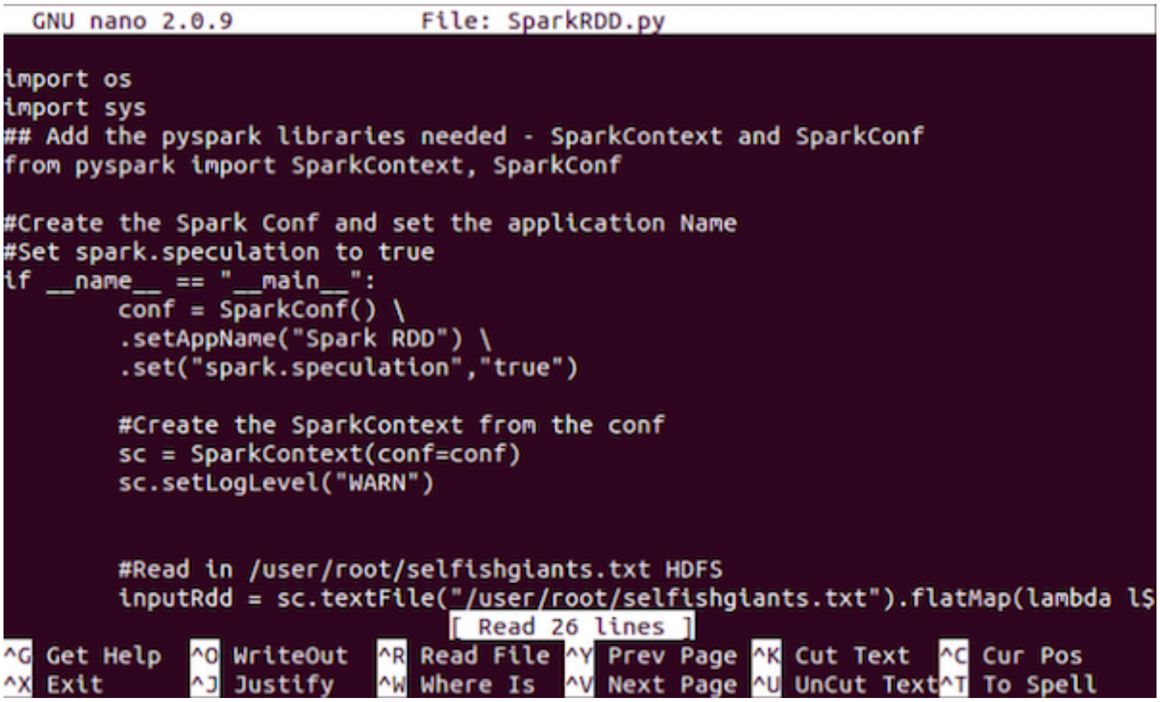


```
import os
import sys

## Add the pyspark libraries needed - SparkContext and SparkConf

from pyspark import SparkContext, SparkConf


#Create the Spark Conf and set the application Name
#Set spark.speculation to true

if __name__ == "__main__":

    conf = SparkConf() \
    .setAppName("Spark RDD") \
   .set("spark.speculation","true")


    #Create the SparkContext from the conf

   sc = SparkContext(conf=conf)
    sc.setLogLevel("WARN")


    #Read in /user/root/selfishgiants.txt HDFS

    inputRdd =
sc.textFile("/user/root/selfishgiants.txt").flatMap(lambda
line: line.split(" ")).map(lambda line: (line,1))


#Perform wordcount

reducedRdd = inputRdd.reduceByKey(lambda a,b:
a+b).map(lambda (a,b): (b,a)).sortByKey(ascending=False)


#Print the top 10 most used words and stop the sparkcontext

print(reducedRdd.take(10))

sc.stop()
```

The solution file is also available at: `/root/spark/applications/python/solutions/`
SolutionFileName: `SparkRDD.py`



```
[root@sandbox python]# cd /root/spark/applications/python/solutions/
```

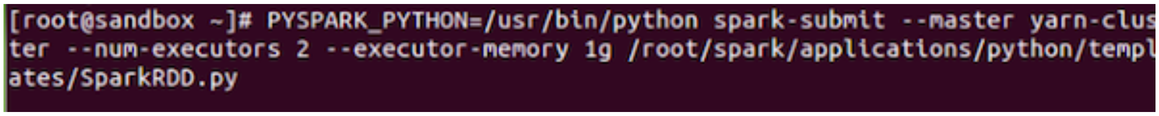


```
  GNU nano 2.0.9              File: SparkRDD.py

import os
import sys
## Add the pyspark libraries needed - SparkContext and SparkConf
from pyspark import SparkContext, SparkConf

#Create the Spark Conf and set the application Name
#Set spark.speculation to true
if __name__ == "__main__":
        conf = SparkConf() \
        .setAppName("Spark RDD") \
        .set("spark.speculation","true")

        #Create the SparkContext from the conf
        sc = SparkContext(conf=conf)
        sc.setLogLevel("WARN")


        #Read in /user/root/selfishgiants.txt HDFS
        inputRdd = sc.textFile("/user/root/selfishgiants.txt").flatMap(lambda l$
                               [ Read 26 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

e. Exit the text editor and save your changes (in `nano`, press `Ctrl + X` to exit and press `Y` to save your changes.

f. Run the application from the terminal.

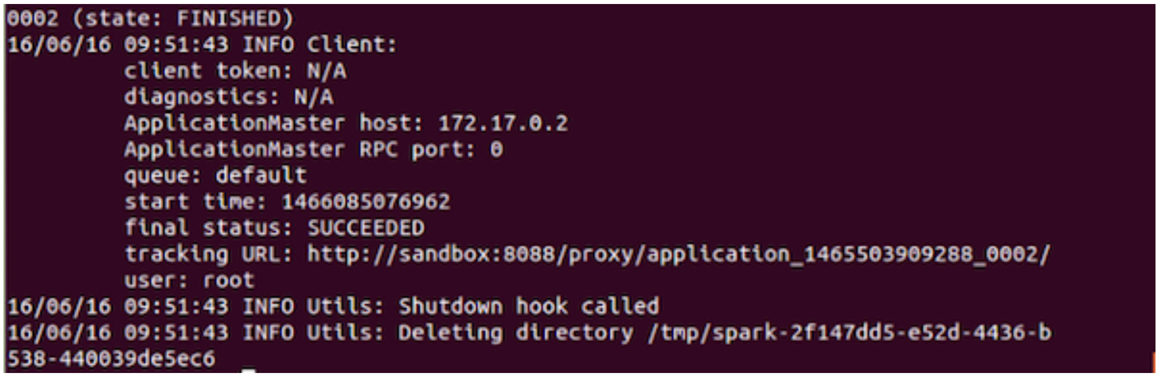
```
PYSPARK_PYTHON=/usr/bin/python spark-submit --master yarn-cluster --num-executors 2 --executor-memory 1g
/root/spark/applications/python/templates/SparkRDD.py
```



```
[root@sandbox ~]# PYSPARK_PYTHON=/usr/bin/python spark-submit --master yarn-clus
ter --num-executors 2 --executor-memory 1g /root/spark/applications/python/templ
ates/SparkRDD.py
```

NOTE:

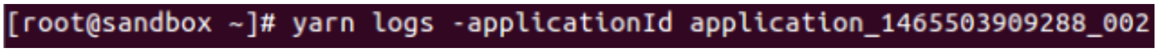
This application will now use YARN as the resource manager with number of executors as 2 and 1g of memory.



```
0002 (state: FINISHED)
16/06/16 09:51:43 INFO Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 172.17.0.2
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1466085076962
         final status: SUCCEEDED
         tracking URL: http://sandbox:8088/proxy/application_1465503909288_0002/
         user: root
16/06/16 09:51:43 INFO Utils: Shutdown hook called
16/06/16 09:51:43 INFO Utils: Deleting directory /tmp/spark-2f147dd5-e52d-4436-b
538-440039de5ec6
```

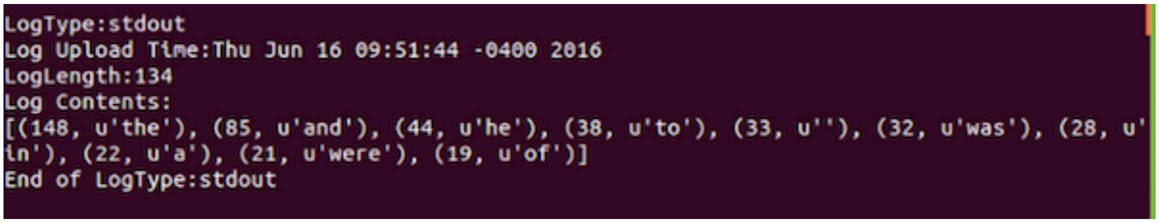
Copy the application ID at the end when the application stops.
The output of the program can be seen using the following command:

```
yarn logs -applicationId <id>
```



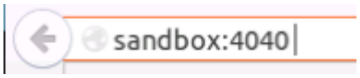
```
[root@sandbox ~]# yarn logs -applicationId application_1465503909288_002
```

Scroll up to see the output



```
LogType:stdout
Log Upload Time:Thu Jun 16 09:51:44 -0400 2016
LogLength:134
Log Contents:
[(148, u'the'), (85, u'and'), (44, u'he'), (38, u'to'), (33, u''), (32, u'was'), (28, u'
in'), (22, u'a'), (21, u'were'), (19, u'of')]
End of LogType:stdout
```

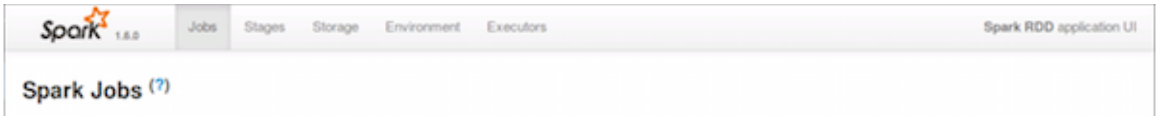
Monitor the submitted Job. Open a new tab on the Firefox browser and browse to:
http://sandbox:4040/



sandbox:4040

IMPORTANT:

The UI below will only be available while the job is running. If you are unable to see the UI, run the application again and quickly switch to the provided link.



Result

You have successfully built and submitted a Spark applications to a YARN cluster.

Lab 14: Machine Learning Walkthrough

About This Lab

Objective:
Observe and run code examples that demonstrate machine learning processes.

File Locations:
NA

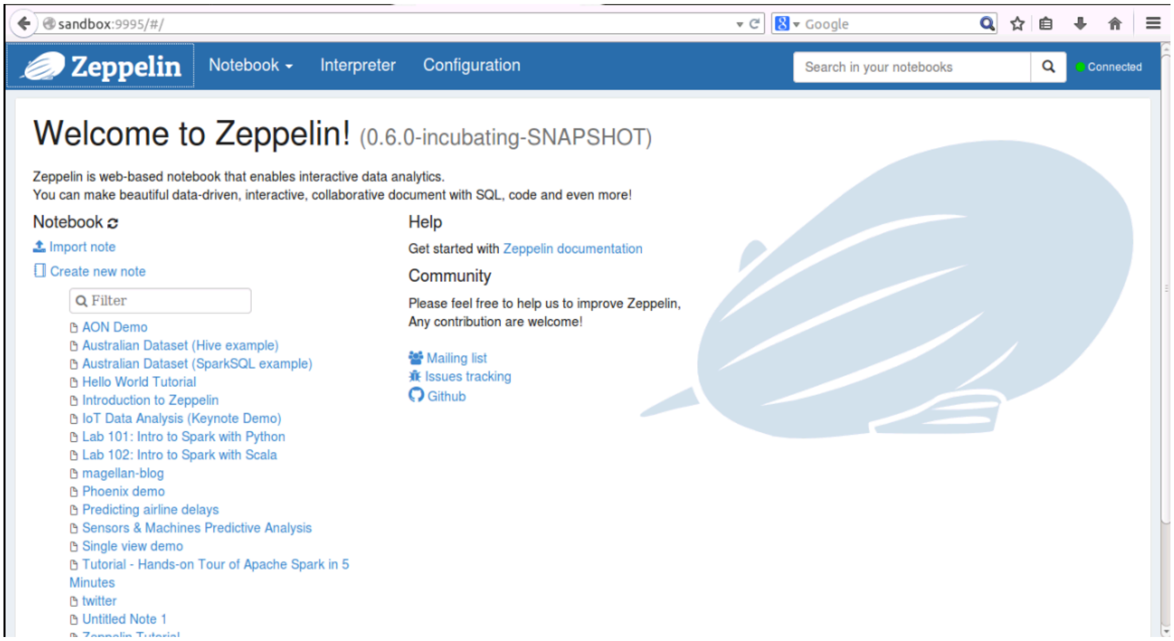
Successful Outcome:
Import a preconfigured note that contains machine learning code samples, read through the note, and run those examples.

Lab Steps

Perform the following steps:

1. **Import the note, read through it, and run code examples.**

   a. Open the Firefox browser and enter the following URL to view the Zeppelin UI.
   http://sandbox:9995/



Welcome to Zeppelin! (0.6.0-incubating-SNAPSHOT)

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

- Import note
- Create new note

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- Hello World Tutorial
- Introduction to Zeppelin
- IoT Data Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- magellan-blog
- Phoenix demo
- Predicting airline delays
- Sensors & Machines Predictive Analysis
- Single view demo
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1
- Zeppelin Tutorial

Help

Get started with Zeppelin documentation

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

- Mailing list
- Issues tracking
- Github

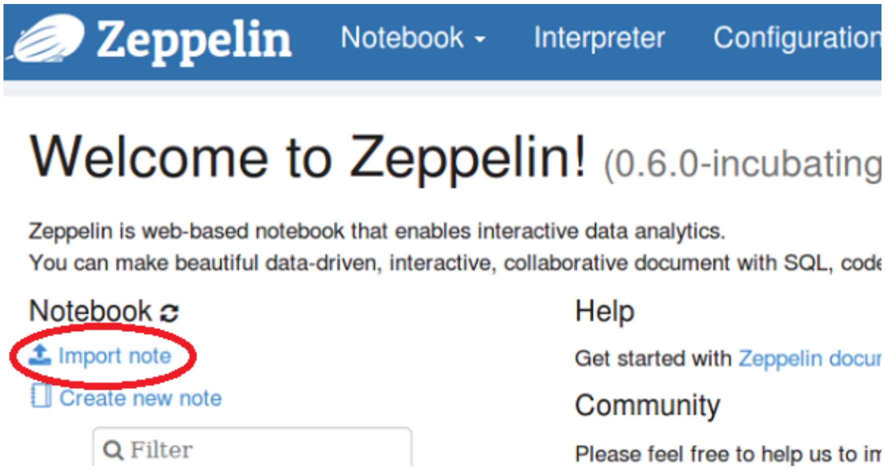
NOTE:

Zepplin’s current main backend processing engine is Apache Spark.

b. Import a copy of the note at the following URL:

https://raw.githubusercontent.com/hortonworks-gallery/zeppelin-notebooks/master/2BNDT63TY/note.json

Name this note Machine Learning Lab. It should appear in the list of available notes on the Zeppelin home page.



Welcome to Zeppelin! (0.6.0-incubating

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code

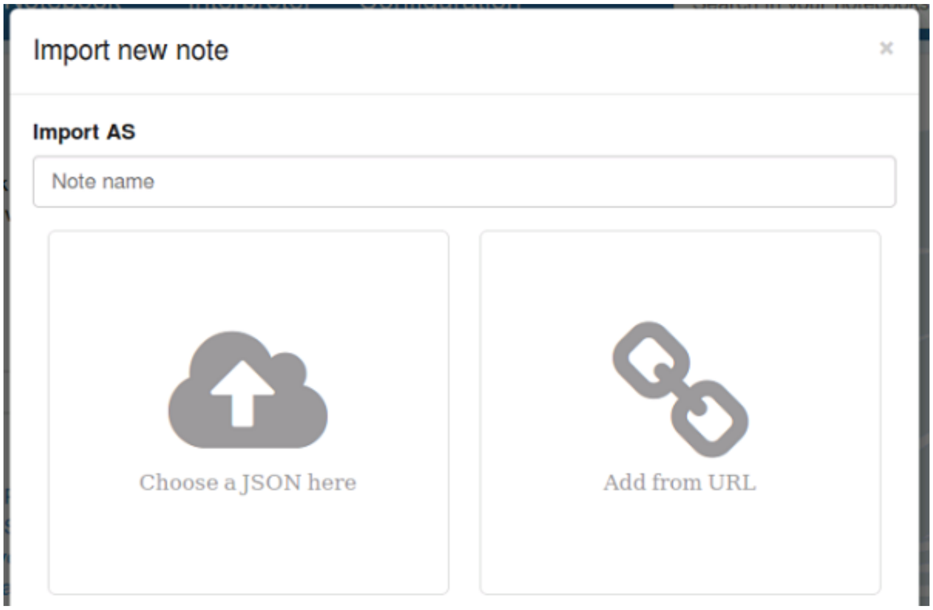
Notebook

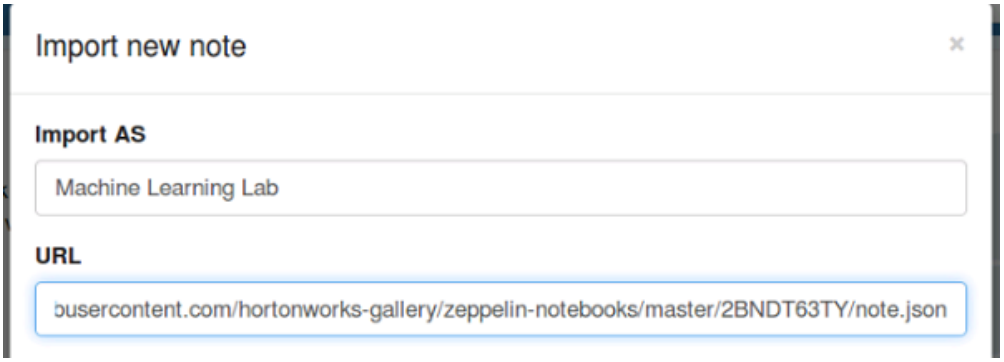
Help

Get started with Zeppelin docur

Community

Please feel free to help us to in





Import new note

Import AS

Machine Learning Lab

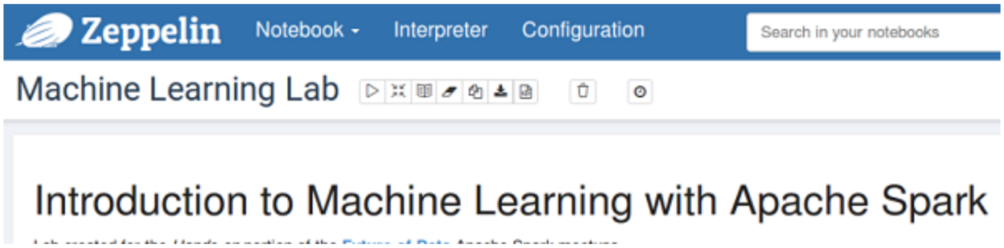
URL

busercontent.com/hortonworks-gallery/zeppelin-notebooks/master/2BNDT63TY/note.json

NOTE:

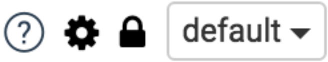
If for some reason the URL is not working, your instructor should know the location of a JSON copy of this note that can be imported instead of importing it from an Internet link.

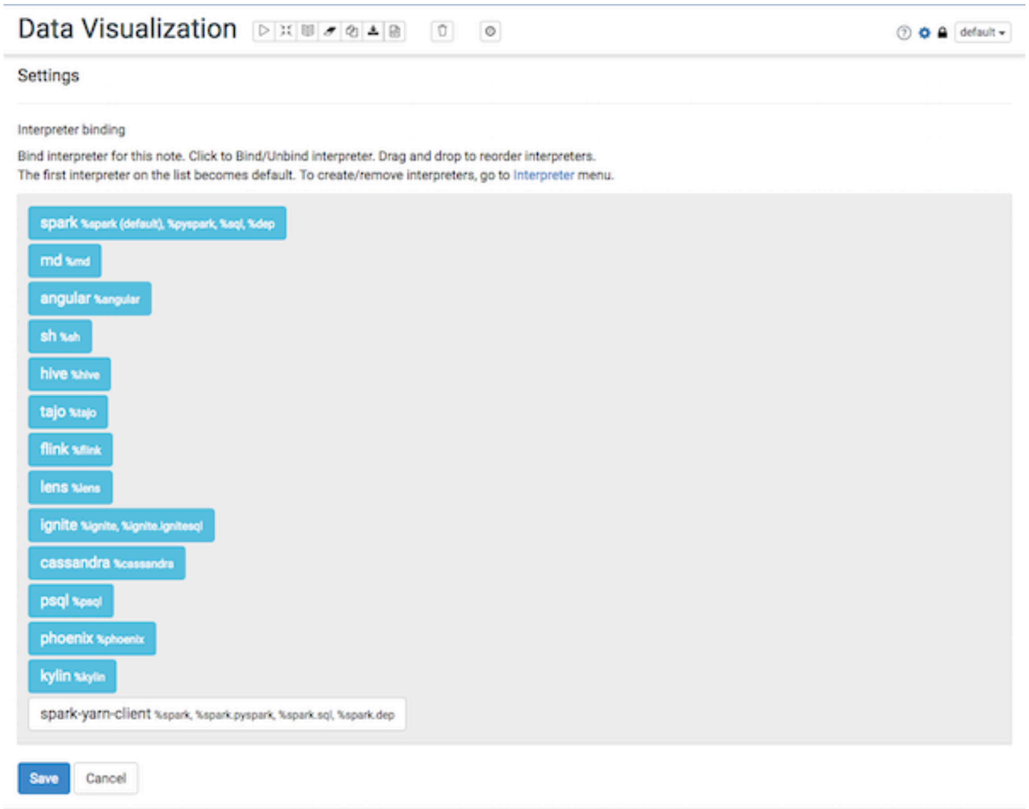
c. Open the new note and set the interpreter to spark-yarn-client.



Machine Learning Lab

Introduction to Machine Learning with Apache Spark





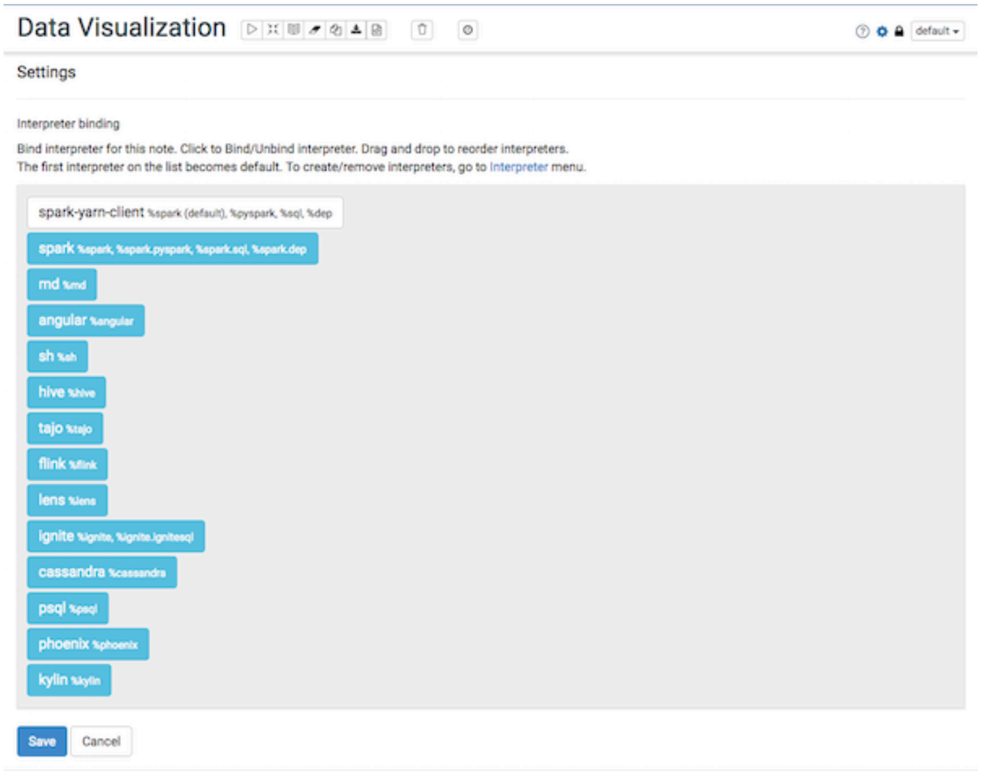
Data Visualization

Settings

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark %spark (default), %pyspark, %sql, %dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin
spark-yarn-client %spark, %spark.pyspark, %spark.sql, %spark.dep



Data Visualization

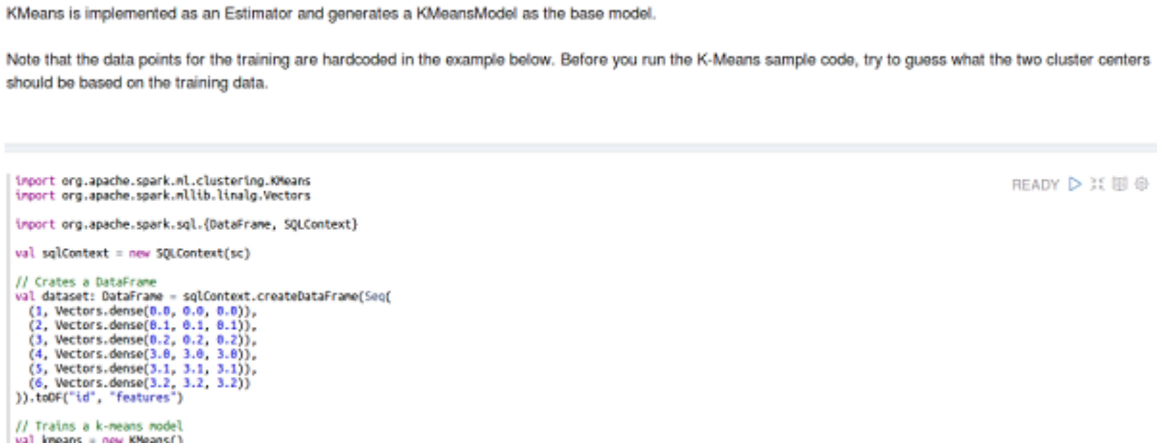
Settings

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters.
The first interpreter on the list becomes default. To create/remove interpreters, go to Interpreter menu.

spark-yarn-client %spark (default), %pyspark, %sql, %dep
spark %spark, %spark.pyspark, %spark.sql, %spark.dep
md %md
angular %angular
sh %sh
hive %hive
tajo %tajo
flink %flink
lens %lens
ignite %ignite, %ignite.ignitesql
cassandra %cassandra
psql %psql
phoenix %phoenix
kylin %kylin

d. Read through the note. A fair number of paragraphs are there for context and instructions. When you come to the first paragraph that displays code, run the code in that paragraph and view the results.

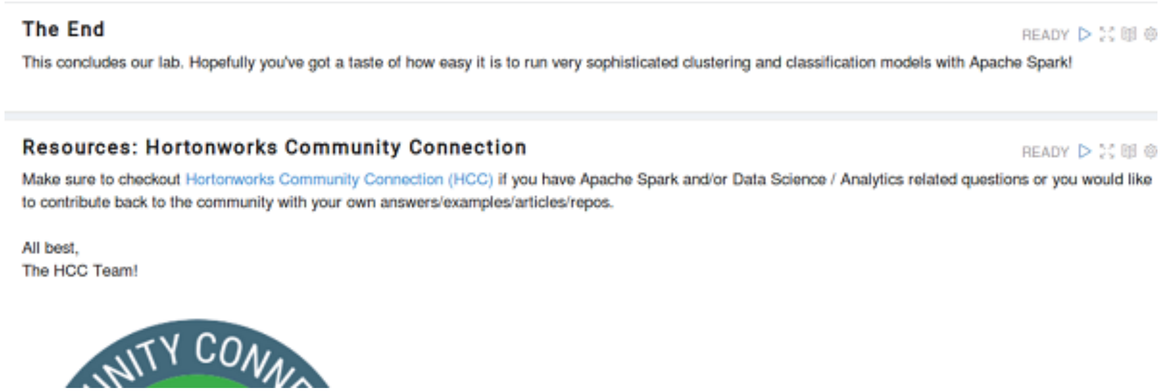


KMeans is implemented as an Estimator and generates a KMeansModel as the base model.

Note that the data points for the training are hardcoded in the example below. Before you run the K-Means sample code, try to guess what the two cluster centers should be based on the training data.

```
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.sql.{DataFrame, SQLContext}

val sqlContext = new SQLContext(sc)

// Crates a DataFrame
val dataset: DataFrame = sqlContext.createDataFrame(Seq(
  (1, Vectors.dense(0.0, 0.0, 0.0)),
  (2, Vectors.dense(0.1, 0.1, 0.1)),
  (3, Vectors.dense(0.2, 0.2, 0.2)),
  (4, Vectors.dense(3.0, 3.0, 3.0)),
  (5, Vectors.dense(3.1, 3.1, 3.1)),
  (6, Vectors.dense(3.2, 3.2, 3.2))
)).toDF("id", "features")

// Trains a k-means model
val kmeans = new KMeans()
```

e. Continue down the note, reading the descriptions and explanations and running the code as instructed, until you reach the end of the note.



The End

This concludes our lab. Hopefully you've got a taste of how easy it is to run very sophisticated clustering and classification models with Apache Spark!

Resources: Hortonworks Community Connection

Make sure to checkout Hortonworks Community Connection (HCC) if you have Apache Spark and/or Data Science / Analytics related questions or you would like to contribute back to the community with your own answers/examples/articles/repos.

All best,
The HCC Team!

Result

You have walked through a preconfigured Zeppelin note that contained multiple examples of machine learning code.

HORTONWORKS®

UNIVERSITY

Learn from the company focused solely on Hadoop.

What Makes Us Different?

1. Our courses are designed by the **leaders and committers** of Hadoop
2. We provide an **immersive** experience in **real-world** scenarios
3. We prepare you to **be an expert** with highly valued, **fresh skills**
4. Our courses are available **near you**, or accessible **online**

Hortonworks University courses are designed by the leaders and committers of Apache Hadoop. We provide immersive, real-world experience in scenario-based training. Courses offer unmatched depth and expertise available in both the classroom or online from anywhere in the world. We prepare you to be an expert with highly valued skills and for Certification.