

CDSW Workshop - Student Guide

This lab will give you hands-on experience with Cloudera Data Science Workbench (CDSW).

In particular it will ensure that you experience:

- Self-service
 - Security
 - Creation of Projects
 - Integration with Git
 - Ability to install and use custom packages
 - Basic Python Visualizations
 - Connectivity to Hadoop filesystem via PySpark
 - R connectivity to Hadoop cluster via Sparklyr
 - Demonstration of remote connectivity to a running program
 - Using Experiments to batch execute workloads where the code, input parameters, and output artifacts are versioned.
-

[Accessing CDSW](#)

[Lab 1 - Self-Registration](#)

[Lab 2 - Hadoop Authentication](#)

[Lab 3 - Creating your first project](#)

[Lab 4 - Visualization and Sharing](#)

[Lab 5 - Hadoop Integration - Run a PySpark job](#)

[Lab 6 - Pushing the Boundaries with TensorFlow](#)

[Lab 7 - R](#)

[SparklyR](#)

[Shiny](#)

[Lab 8 - Scala](#)

[Lab 9 - Experiments](#)

[Lab 10 - Models](#)

[Lab 11 - Scheduling Jobs](#)

[Appendix](#)

[Using your own CDSW cluster](#)

[Troubleshooting](#)

[DNS Issue](#)

[Spark R backend might have failed](#)

[ImportError: No module named google.protobuf](#)

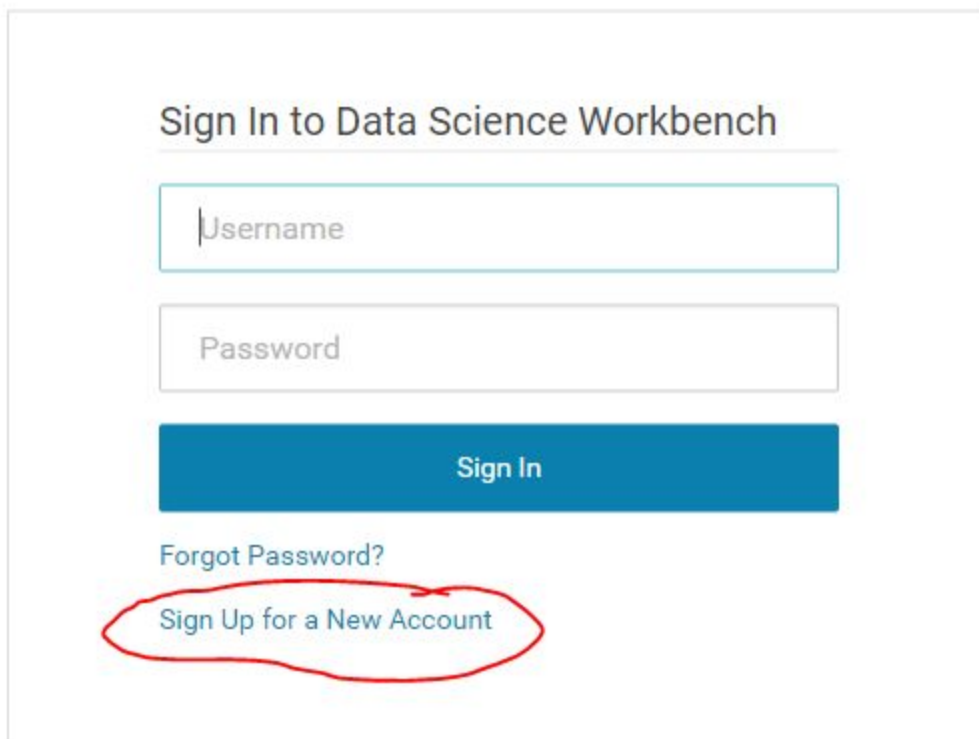
Accessing CDSW

You will need to use a browser to connect to the CDSW system. The CDSW URL for this workshop is:
<http://cdsw.cloudera-partner.com/>

Lab 1 - Self-Registration

For this system we've turned on self-registration, but you can restrict registration to invitation only if you'd prefer. And users can be integrated with LDAP, providing a single username/password consistent with other parts of the organization.

1. Go to CDSW at the URL as described above in "Accessing CDSW"
2. Register yourself by clicking the link "[Sign-Up for a New Account](#)". Use **userXX**, where **XX** is the number assigned to you by your instructor .
Use a simple password you can remember - 'cloudera' will do!



Sign In to Data Science Workbench

Username

Password

Sign In

Forgot Password?

Sign Up for a New Account

Sign Up for Data Science Workbench

user99

user99

Your personal profile will be located at:

<http://cdsw.cloudera-partner.com/user99>

user99@admin.com

.....|

Sign Up

Already have an account? [Sign In!](#)

Question: How will this help your IT group manage the explosive growth of Data Scientists?

Question: How do you sign out of your account? - Do it now, then sign back in.

Question: Does your laptop have enough RAM to run different types and loads for Data Science jobs you would like to utilize? Look at the main Dashboard. How much RAM and vCPUs are setup in our lab environment for the group to use?

Lab 2 - Hadoop Authentication

Only do this lab if you know your cluster is kerberized AND you know your kerberos credentials.

At Cloudera we believe all clusters should be secured with Kerberos. However configuring a regular Jupyter notebook to gain access to a secure cluster is very difficult. Here we show you how easy it is with CDSW:

1. Navigate to **Settings** → **Hadoop Authentication**

The screenshot shows the Cloudera Data Science Workbench (CDSW) interface. On the left sidebar, the 'Settings' menu item is circled in red. The main content area is titled 'User Settings' and has four tabs: 'Profile', 'SSH Keys', 'Hadoop Authentication' (which is circled in red), and 'API Keys'. Under the 'Hadoop Authentication' tab, there is a 'Kerberos' section. It contains a text input field for the 'Principal' with the value 'user99@HADOOP.LOCAL'. Below this is a 'Credentials' section with two tabs: 'Password' and 'Keytab'. The 'Password' tab is active, showing a text input field for the password (masked with dots) and an 'Authenticate' button. At the bottom of the 'Kerberos' section, there is a link 'Show Kerberos configuration'.

2. The Principal and password you'll use vary depending on how you're taking this lab:
 - a. If using the Cloudera provided online training then use Principal **userXX@HADOOP.LOCAL** and **Cloudera1** as your password, where **XX** is the number assigned to you by the instructor.
 - b. If using your own lab then your administrator will need to provide this to you.
3. Hit **Authenticate**:

The screenshot shows the Cloudera user interface. At the top, there's a header with 'cloudera' logo, 'user99', 'Settings', and 'Hadoop Authentication'. On the left is a dark sidebar with icons and labels for 'Projects', 'Sessions', 'Experiments', 'Models', 'Jobs', and 'Settings'. The main content area is titled 'User Settings' and has four tabs: 'Profile', 'SSH Keys', 'Hadoop Authentication' (which is selected and underlined), and 'API Keys'. Under the 'Hadoop Authentication' tab, there's a section titled 'Kerberos'. Inside this section, it says 'Kerberos authentication' in green. Below that, it shows a checkmark and the text 'Currently authenticated as user99@HADOOP.LOCAL', where the email address is highlighted in yellow. At the bottom of the Kerberos section, there is a blue 'Sign out' button and a link 'Show Kerberos configuration'.

And that's it! - you're now ready to work on a secured cluster.

In a normal production setting each of you would have separate authentication credentials as we are doing. It's also possible to have users access Hadoop with the same credentials but that's not a best practice and often violates company and compliance policies.

Question: Will IT finally allow your Data Scientists access to the secured data in Hadoop?

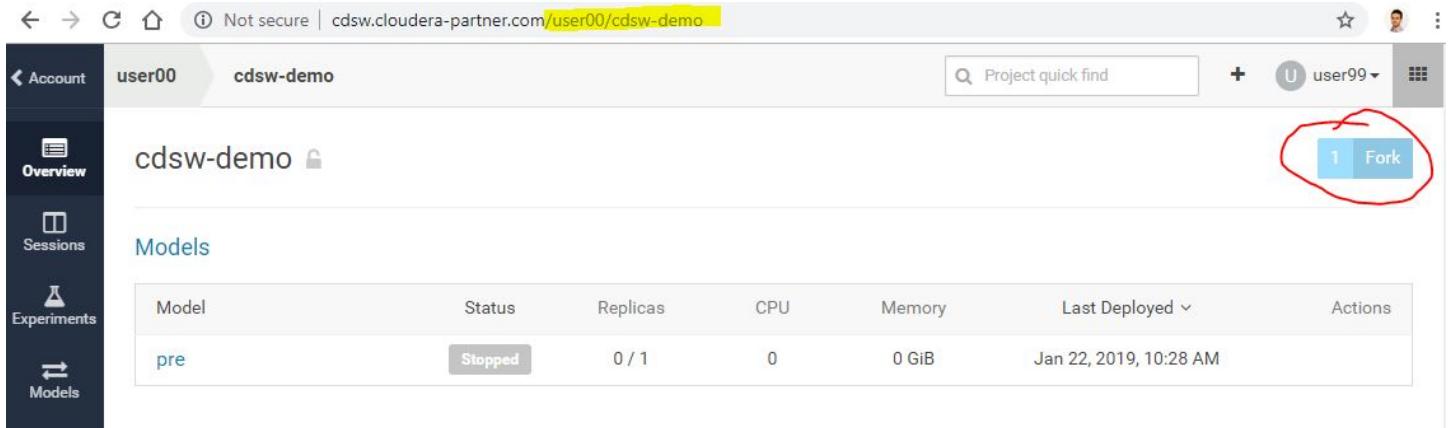
Question: How do you think you might use the public SSH key that's made available in this Settings section?

Question: If an alternate solution hardcoded Hadoop credentials in the application code, how big of a problem do you think that would be to maintain and stay compliant within your company and our at your customer?

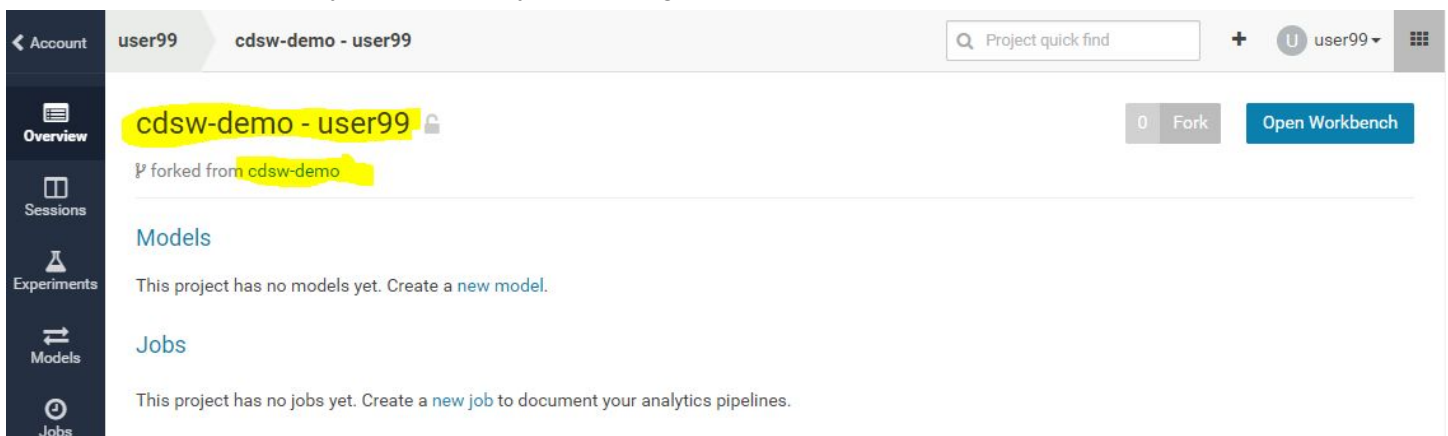
Lab 3 - Creating your first project


In this lab we're going to create a project by forking from the **cdsw-demo** project setup by the leader user **user00** (and into which that user has already installed all the R libraries - this is a typical workflow for CDSW: a central project that has many of the dependencies installed, and which people fork to work on their own code). We'll install some python packages too for our python code:

1. Go to the **user00**'s project page by replacing your user name in the URL with user00:
<http://cdsw.cloudera-partner.com/user00>
2. Select the project provided (**cdsw-demo** in below screenshot:

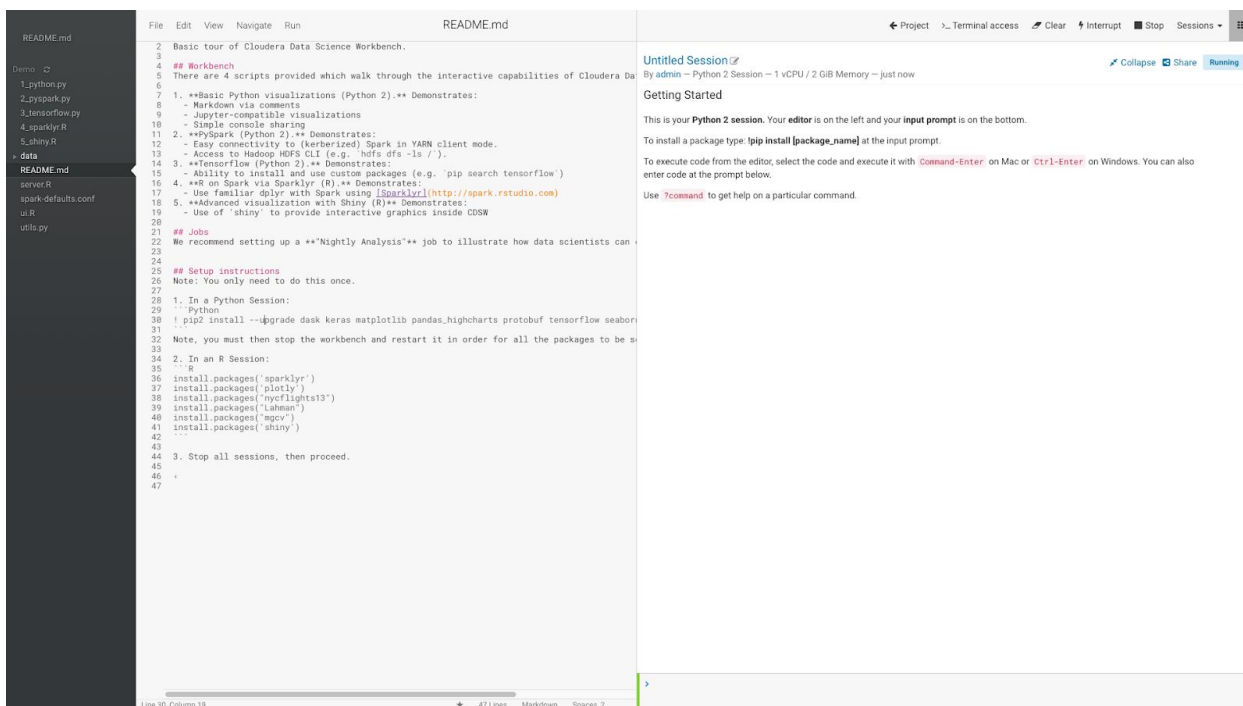
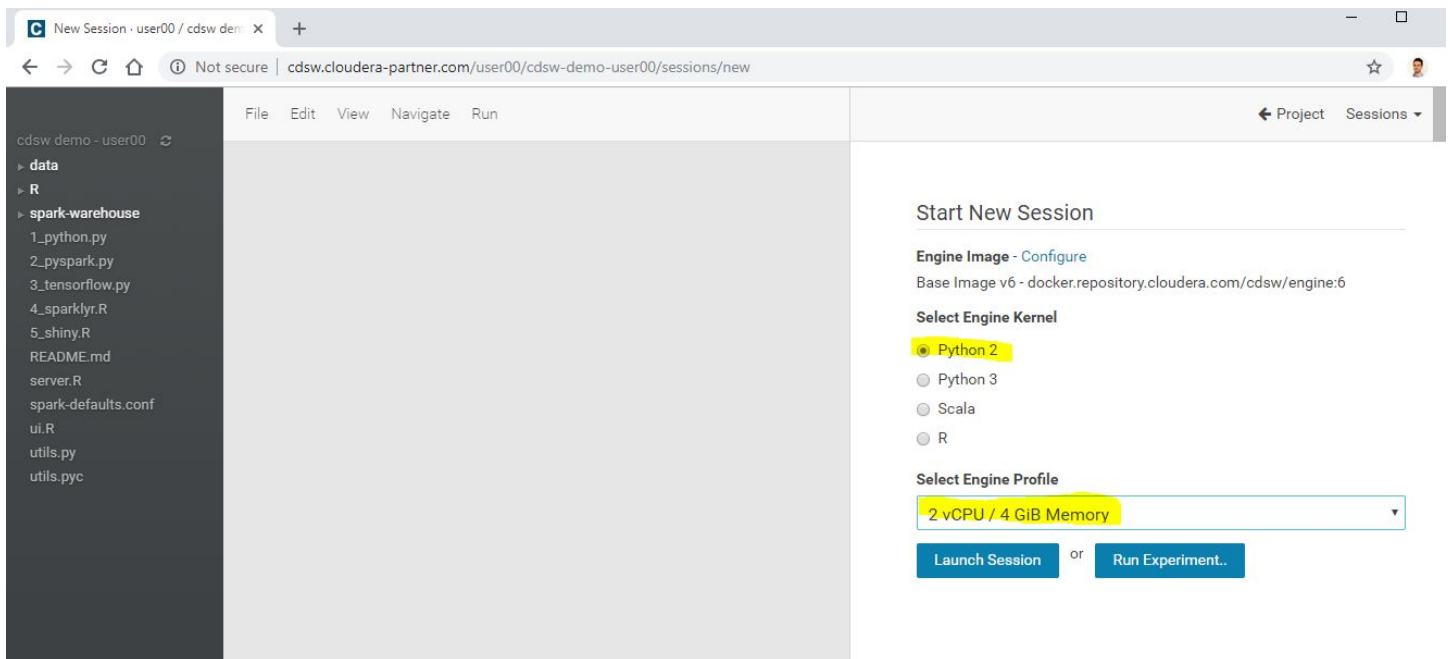


3. Hit the **Fork** button at the top right and you'll be given a choice of where to fork this project (there'd be more choices if you were a member of one or more teams). Select **@userXX**
4. When its finished you'll see that you're being shown the forked version




5. So we've created a project by forking from another project. Now we're going to add the python packages, just to show that anyone can add whatever packages they need to their own project:
6. Select **Open Workbench** (blue button, top right) and you'll be taken to the workbench. This is the main area that you'll use when working in a project with CDSW.
7. On the far left is a file browser (note the little 'refresh' icon at the top: )
8. In the middle is an editor open on the file that you selected - in this case the README.md file.
9. On the right is a Session Start tile - in this case it's waiting for you to select an engine to run (so far your project has file space but no compute sessions).

10. Select the **Python 2 kernel**; select the 2 vCPU / 4 GB Engine (and use this size engine for all your Python sessions in this workshop) and then the **Launch Session** button. This will startup a Python engine and the right hand side will become two tiles. The top one is an output tile and the bottom one (with a red, then green left hand border) is a shell input window.



11. Rename your session from 'Untitled Session' to something more interesting (I usually start with a language indicator and then other stuff - 'P2: package install', for example). Do this by clicking on the

edit icon  just to the right of the 'Untitled Session' title. We won't remind you but we do expect you to do this whenever you open up a new session from now on. Your session heading should now look something like this:

P2: install

By admin – Python 2 Session – 1 vCPU / 2 GiB Memory – just now

12. Now to show you some cool stuff ...

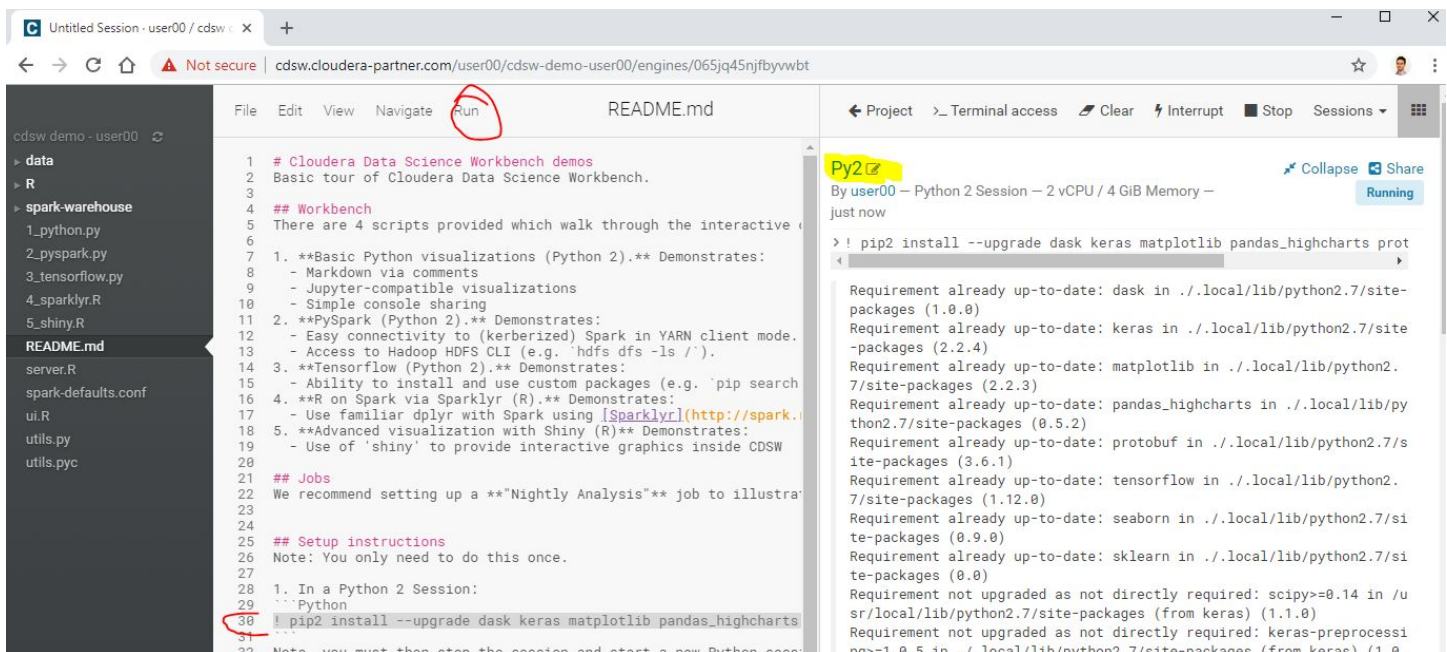
13. Look at this line in the README.md:

```
14. !pip2 install --upgrade pip dask keras matplotlib pandas_highcharts protobuf  
tensorflow seaborn sklearn
```

15. Question: What does this do? If you know Python it should be obvious; if you don't, then let me tell you: this is an execution of pip (a Python package manager), which will tell the system upgrade the listed packages

16. Put your cursor at that line and select 'Run Line(s)'

17. The cursor on the left should turn to red and, after a few seconds, you should see output like this:



The screenshot shows the Cloudera Data Science Workbench interface. On the left is a file explorer with a tree view containing folders like 'data', 'R', 'spark-warehouse', and files like '1_python.py', '2_pyspark.py', '3_tensorflow.py', '4_sparklyr.R', '5_shiny.R', 'README.md', 'server.R', 'spark-defaults.conf', 'ui.R', 'utils.py', and 'utils.pyc'. The 'README.md' file is selected and its content is displayed in the main editor. The code in the README.md includes sections for 'Workbench', 'PySpark', 'Tensorflow', and 'Jobs'. A red circle highlights the line: `! pip2 install --upgrade dask keras matplotlib pandas_highcharts protobuf tensorflow seaborn sklearn`. On the right, a terminal window titled 'Py2' shows the output of the command. The output indicates that several packages (dask, keras, matplotlib, pandas_highcharts, protobuf, tensorflow, seaborn, sklearn) are already up-to-date or have been successfully upgraded. The terminal window also shows the session title 'By user00 – Python 2 Session – 2 vCPU / 4 GiB Memory – just now' and a 'Running' status.

18. What's happening here is that the code in the file is being executed in the console (did you notice the left hand edge turned red?) and now you can see the output in the right hand screen.

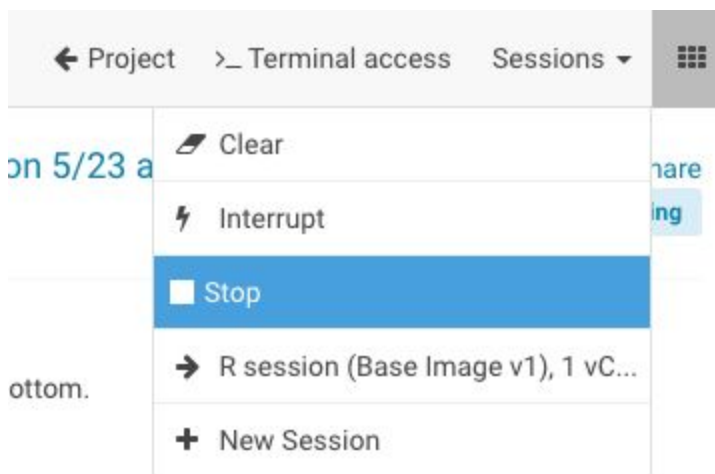
19. Question: What do you think the exclamation is for?

20. Question: What happens if you type **magic**? What happens if you type just **?** Try it!

21. Once the vertical bar turns green then stop this session. The Stop button is either on the top menu bar (when there's sufficient room for it):

← Project >_ Terminal access  Clear  Interrupt  Stop Sessions ▾

22. Or it's in the the Session drop down (when there's little room for buttons):



ter on Mac or **Ctrl+Enter** on Windows. You can also

23. The instructor already installed the R packages using and R session in the user00's project we forked. If the R packages hadn't already been installed you could've installed those yourself in a R session, using the following lines from the Readme - but those packages are already installed, so you don't need to do this:

```
34 2. In an R Session:
35 ```R
36 install.packages('sparklyr')
37 install.packages('plotly')
38 install.packages("nycflights13")
39 install.packages("Lahman")
40 install.packages("mgcv")
41 install.packages('shiny')
42 ```
```

Note that this workbench is independent of any other project's workbench. You want to try out different and conflicting libraries? Have at it! Just start another project, open the workbench, pick the libraries you want, install them and off you go. Just like on your laptop, but this is managed, secure, stable, won't get stolen from a taxi, is always available and easily shared!

All of this isolation is achieved by mounting filesystems (one or more per project) into docker containers (one per engine). The details don't matter, except to note that now you can really go mad and try out lots of different and conflicting permutations without having to go down the complex path of virtual environments etc. It's all been done for you!

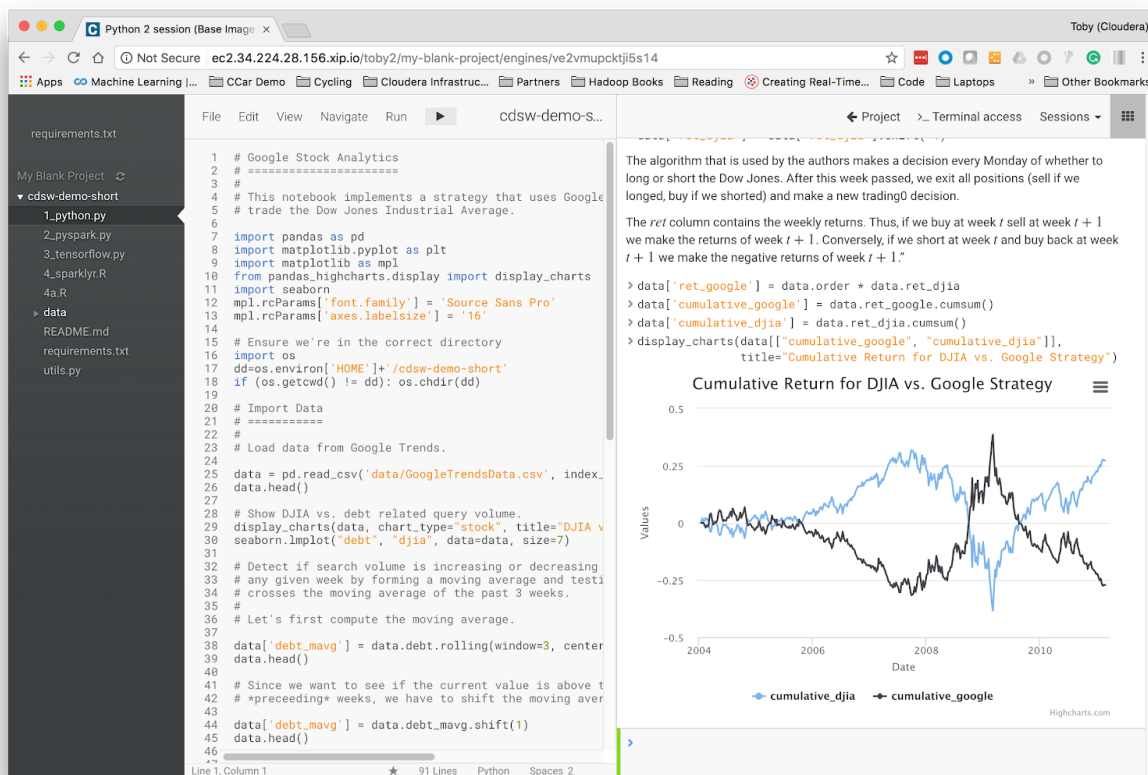
Question: How many beers will the IT guys buy you when they realize that you now have a secure, auditable and managed system where they don't have to pick and choose the libraries that you install?

Lab 4 - Visualization and Sharing

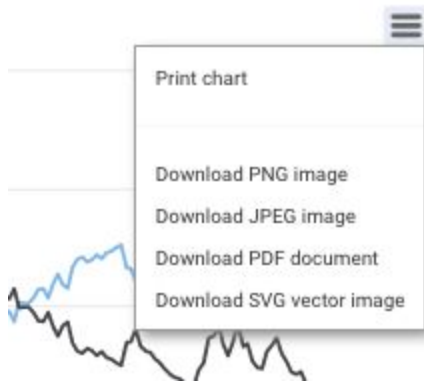
Check that you have NO sessions running - if any sessions are running then stop them now. You've setup your environment and those sessions can be safely disposed of.

Data Science is often about visualizing ideas, and then sharing them to persuade others to take action. CDSW lets you use the visualization tools you'd use naturally, and adds a neat twist to the whole idea of sharing. Let's get started:

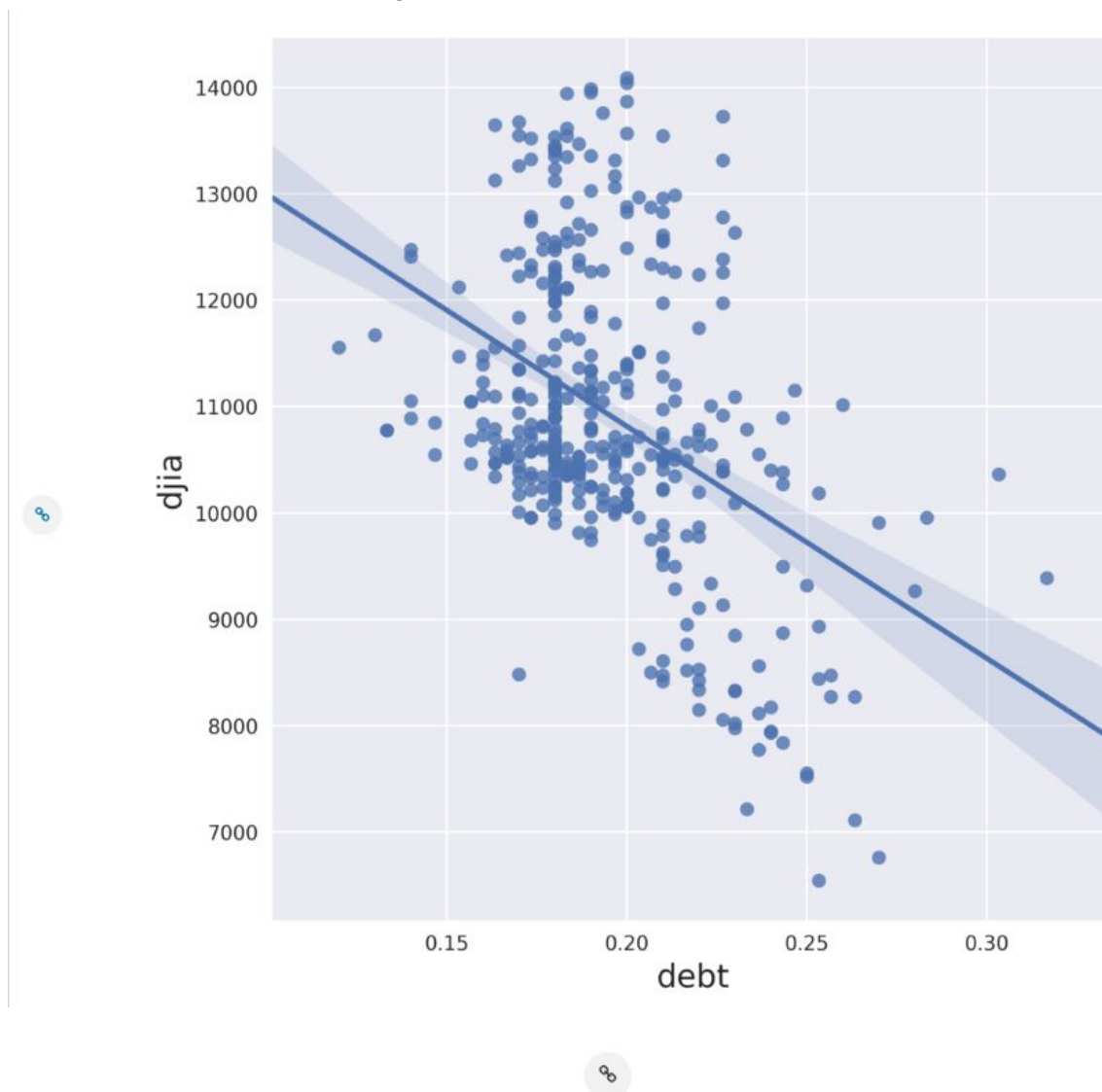
1. Start up a new **Python 3** session (2vCPU, 4GiB) in the same manner you did before.
2. Select **1_python.py** in the file browser
3. Run the entire file (multiple ways of doing that - try to figure out more than one way. It should be pretty obvious!). You should end up with some nice graphs in the output window:



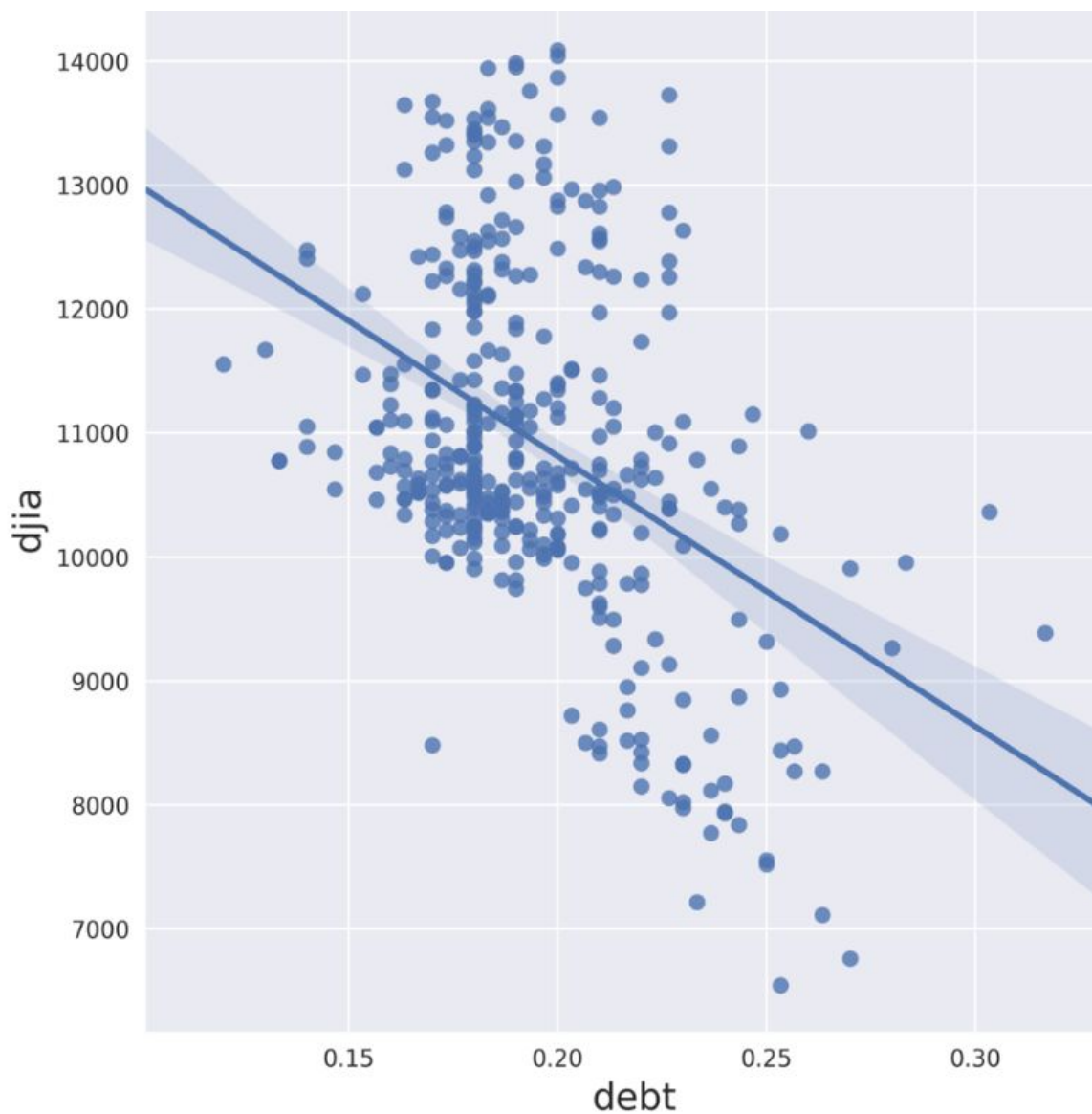
4. In the top right of that graphic is a little 3 bar icon - click on it and you get these handy options:



5. You can see that CDSW is very similar to a notebook, supporting the same visualization tools. However, unlike a notebook, it doesn't use cells: instead it uses markup in the source file, and an output window. Furthermore, that window has some interesting properties ...
6. Scroll up to find this diagram:



7. On the left is a little chain link button:
8. Click on it and you'll see beneath the chart some html that can be used to embed that chart into a website:



Copy and paste to embed this cell in your website!



width (optional)

height (optional)

```
<div id='sense-widget-gvy7q0dy' class='sense-embed-container' style='width:100%;'><script id='embedding-script-35'  
src='http://consoles.ec2.34.211.6.111.xip.io/js/sense-embed.js' data-cell-  
url='http://livelog.ec2.34.211.6.111.xip.io/topics/s0u1n6if2njemzy2_output/35' data-auth-  
token='eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRoIjpb7InJlYWQiOlsic2B1MW42aWYybmlbXp5Ml8qIl0sIndyaXRlIjpbInMwdTFuNm1mI  
o8' data-assets-cdn-root='http://consoles.ec2.34.211.6.111.xip.io/0/1/s0u1n6if2njemzy2/' data-widget='sense-widget-  
gvy7q0dy' data-width='\" data-height='\"></script></div>
```

9. Scroll to the top of the window and you'll see this on the far right:

Collapse


Share

Running

10. Hit the 'collapse' link and see the difference in the output window.


11. Question: What difference did you see? How might this be used? Is it useful?

12. Notebooks have great output, but how do you share what they show you? CDSW solves this by simply providing a link to the output that you can send to anyone and they can see the output. Try it:
13. Select the 'Share' link from either a Public or Private project:

 This session is **public** since it's part of a public project. Generate a share link for a clean presentation.

Generate Share Link

14. And then 'Generate Share Link' (your URL will be similar, but different from this one):

 Anyone who has the link **can view** this session's results until revoked.

☒ Hide code and text

Revoke Share Link

15. Cut and paste that link and put it into some other browser (best to be a completely different browser, or on a incognito window, than the one you're logged in with, but not that important)
16. You should see that you have access to almost the same output window (this new one doesn't have this share link!)

Question: How would you use this sharing capability to keep your co-workers informed about what you're doing?

Question: How would you change your presentation to take advantage of the collapse/expand capabilities of this sharing system?

So we've demonstrated how CDSW is like a notebook, but is perhaps more powerful, and has great sharing capability. Let's go on to see about integration with Hadoop!

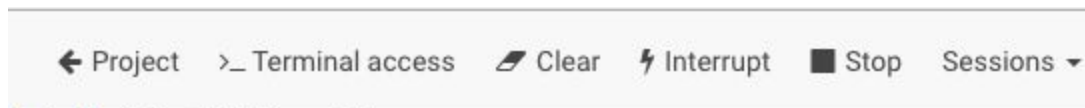
Lab 5 - Hadoop Integration - Run a PySpark job

In this lesson we'll see two mechanisms for integrating with Hadoop:

1. Filesystem - storing data in Hadoop itself using HDFS
2. Computation - executing code on the Hadoop cluster via Spark

Stop all sessions you have currently running. Start a new **Python2** session.

1. Select the **2_pyspark.py** file.
2. Look at line 34:
3. `!hdfs dfs -put -f $HOME/data/kmeans_data.txt /user/$HADOOP_USER_NAME`
4. **Question:** What is this line doing? Where is the data coming from? Where is it going to?
5. You can see that the original data (stored under the data directory of our project) is copied into HDFS. The data is tiny (go look at it with the file browser!), but the principle is what applies - simple integration with HDFS.
6. Execute the `2_pyspark.py` file in your already running Python session
7. **Question:** What did it do?
8. **Question:** What kind of thing is the variable 'data'? (try typing 'data' into the console and seeing what gets printed out.)
9. Open a terminal using the 'terminal' icon in the top right:



10. Execute `'hdfs dfs -ls'` to see the data file in the hadoop file system (or, to show off, execute `'! hdfs dfs -ls'` in the python console to do the same thing!)

If everything went correctly you'll see that we demonstrate:

- Natural integration with HDFS - it's just a path to a file!
- Natural parallel computation across the cluster using Spark

Question: If you had 20TB and 50 nodes with 32 processors per node how much faster do you think a KMeans algorithm would take compared to running it on your laptop? 10X? 100X?

Question: What shell is running in that terminal? What else can you do in the terminal? Does it remind you of a terminal access to your Linux laptop?

Lab 6 - Pushing the Boundaries with TensorFlow

So you've just heard that the latest thing from Google is [Tensorflow](#) and you're keen to get started. You're going to need to install some custom packages and, more importantly, connect a program providing a web interface so that you can view the results. Your IT department isn't going to help you - you're going to want to do this experiment on your own.

Fortunately CDSW enables you to install custom libraries. This cluster might be managed by IT but you can still get your libraries in there to do the work you need ...

We've done the hard work for you - take a look:

1. Stop all sessions. Start a new **Python 3** session.
2. Select **3_tensorflow.py**
3. Lines 2 through 10 show the imports of the various libraries (we installed them in Lab 3)
4. Run 3_tensorflow.py in your current Python session - the input handwritten number images are shown, along with some images of feature maps for particular numbers
5. In the output on the right hand side, at the bottom, is a blue link 'Open Tensorboard'. Selecting that link opens an application, running within your CDSW engine, to examine the Tensorboard system. Note that because we are using a dynamic DNS service for these labs it might take as long as a minute before that URL is useful - what's happening is that the DNS records are being updated in real time to point that URL to the right place, so you might have to wait until those records are updated.

Note - if everyone is using a different ID you shouldn't see any errors. If attendees are using the same ID you're likely to see some output with errors like this:

Explore using Tensorboard

```
> utils.start_tensorboard(logs_path, iframe=False)
TensorBoard 0.1.6 at http://gif90wc4hfiym2r7:8080 (Press CTRL+C to quit)
Starting Tensorboard at http://gif90wc4hfiym2r7.cds.w.104.197.97.113.nip.io...
```

[Open Tensorboard](#)

```
Traceback (most recent call last):
  File "/usr/local/lib/python2.7/SocketServer.py", line 599, in process_request_thread
    self.finish_request(request, client_address)
  File "/usr/local/lib/python2.7/SocketServer.py", line 334, in finish_request
    self.RequestHandlerClass(request, client_address, self)
  File "/usr/local/lib/python2.7/SocketServer.py", line 657, in __init__
    self.finish()
  File "/usr/local/lib/python2.7/SocketServer.py", line 716, in finish
    self.wfile.close()
  File "/usr/local/lib/python2.7/socket.py", line 283, in close
    self.flush()
  File "/usr/local/lib/python2.7/socket.py", line 307, in flush
    self._sock.sendall(view[write_offset:write_offset+buffer_size])
error: [Errno 32] Broken pipe
```

We're not certain, but we believe that this is because the TensorBoard application is being run multiple times by the same service user on the CDSW Gateway node and is an issue in the TensorBoard code.

Key takeaways: You were able to install and use custom third party libraries, as well as run an application that you can connect to from an external application

Lab 7 - R

SparklyR

We've focused on python integration, but just to show we can do similar things with R, let's take a look at the R programs and execute them. Let's start with connecting to and running Spark from an R session.

Start up an R session (2vCPU, 4GiB engine).

1. Switch to the **R Session**.
2. Select (and run) **4_sparklyr.R**
3. Can you figure out some of the things it's doing? If you know R, and if you know sparklyr, then you can get detailed; if you don't know R then simply 'collapse' the output and see if you can make sense of the analysis without looking at any code ... hopefully you can!

Shiny

R has a great interactive experience using the shiny package. In this lab we'll create an interactive histogram and you can work with it to find out the frequency distribution of the period between Yellowstone Geyser eruptions!

Run the **5_shiny.R** application

This will start up a shiny server in the local context (line 16), and connect it to the server/ui code (in server.R and ui.R, respectively).

Your users are then presented with a nice little application where they can experiment with changing some of the parameters and graphing features of R's base graphics histogram plot!

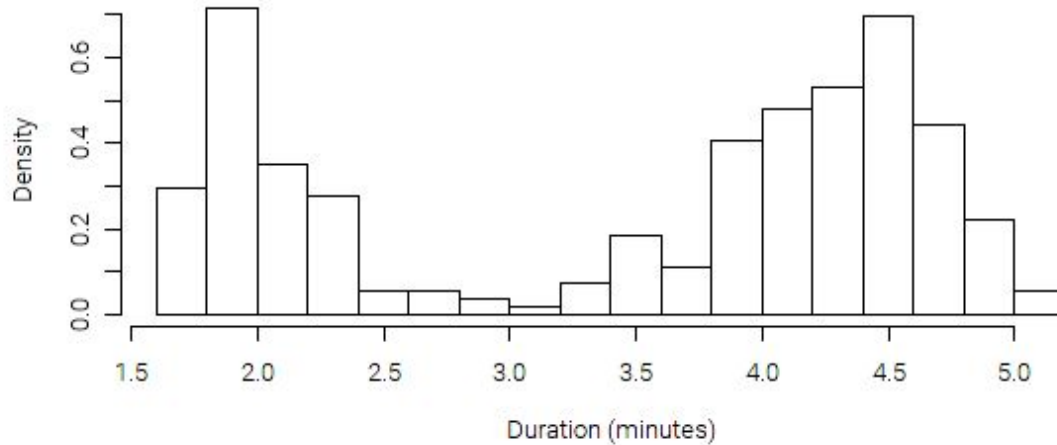
Number of bins in histogram (approximate):

20

☐ Show individual observations

☐ Show density estimate

Geyser eruption duration



Try generating a share link and opening up the share in another browser window - amazingly enough each browser share is independent, allowing your users to share the same underlying experience, but with their individual data inputs!

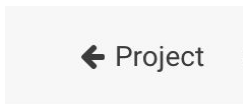
STOP all your R Sessions now - this will help free up resources for everyone

Question: Is this kind of interactivity with data likely to change the way your business users understand and appreciate the work of the Data Scientists?

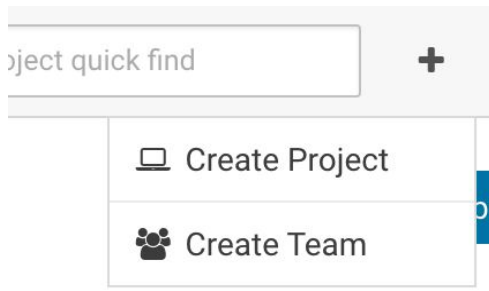
Lab 8 - Scala

In this lab we show how you can use the 'Template' mechanism to get started with a simple Scala example:

1. Navigate to the project space by selecting "project":



2. Create a new project by hitting the '+' button on the top right and selecting 'create project':



3. In the Create new Project window that comes up provide a name for your new project ('Scala', for example), and then choose the Scala template:

Create a New Project

Project Name

Project Visibility

☒ **Private** - Only added collaborators can view the project.

☐ **Public** - All authenticated users can view this project.

Initial Setup

[Blank](#) [Template](#) [Local](#) [Git](#)

Templates include example code to help you get started.

Create Project

4. Create the project. You'll see the File Browser view onto the project.
5. Hit '**Open Workbench**' in the top right and let's go run some Scala code:
6. Start a **Scala** session
7. The Scala example project includes its own data set that needs to be moved into HDFS, since that is where the scala code expects to find it. Open a Terminal by clicking on the **>_Terminal access** button

and execute the following shell commands to do this. Note how you have ready access to your own project's data (purely local to you) and the secure (and massive) HDFS cluster:

- \$ hdfs dfs -put data /tmp/datauserXX (replace XX with your user number)

The screenshot shows the Cloudera Data Science Workbench interface. On the left, a file explorer shows a project structure with files like `data`, `examples`, `spark-warehouse`, and `auction-analysis.scala`. The main editor displays a Scala file `auction-analysis.scala` with code for loading data from HDFS and creating an RDD. A terminal window is open in the foreground, showing the project workspace at `/home/cdsw` and the Kerberos principal `user99@HAD00P.LOCAL`. The terminal also lists the runtimes: R (3.4.1), Python 2 (2.7.11), Python 3 (3.6.1), and Java (1.8.0_121). The terminal output shows the execution of `hdfs dfs -put data /tmp/datauser99` and `hdfs dfs -ls /tmp/*`, resulting in a list of files and directories in the `/tmp` directory. A red arrow points to the terminal output.

- Execute one or more of the various scala files in the Workbench

Question: How will you use templates when demonstrating CDSW to your friends and colleagues?

STOP your Scala session now

Lab 9 - Experiments

It's often the case that data scientists need the ability to iterate and repeat similar experiments in parallel and on demand, as they rely on differences in output and scores to tune parameters until they obtain the best fit for the problem at hand. Such a training workflow requires versioning of the file system, input parameters, and output of each training run.

This lab will give you practice running non interactive batch execution scripts called Experiments that are versioned across input parameters, project files, and output associated with a specific versioning of the project files retaining run-level artifacts and metadata.

1. Open up your cdsw-demo project and open the workbench.
2. Select file 6_experiment.py and inspect the code. What do you think it does?
3. From the Workbench launch a 2 vCPU/4 GiB memory Python3 session and enter the in the command prompt to run the file and verify it is working:

```
!python 6_experiment.py 11 1
```

The screenshot displays the CDSW Workbench interface. On the left, a file explorer shows the project structure with files like `cdsw-build.sh`, `1_python.py`, `2_pyspark.py`, `3_tensorflow.py`, `4_sparklyr.R`, `5_shiny.R`, and `6_experiment.py` (which is selected). The main editor shows the code for `6_experiment.py`:

```
1 import sys
2 import cdsw
3
4 args = len(sys.argv) - 1
5 sum = 0
6 x = 1
7
8 while (args >= x):
9     print ("Argument %i: %s" % (x, sys.argv[x]))
10    sum = sum + int(sys.argv[x])
11    x = x + 1
12
13 print ("Sum of the numbers is: %i." % sum)
14 cdsw.track_metric("Sum", sum)
15
```

On the right, a terminal window titled "Untitled Session" shows the execution of the command `!python3 6_experiment.py 11 1`. The output is:

```
> import sys
> import cdsw
> args = len(sys.argv) - 1
> sum = 0
> x = 1
> while (args >= x):
>     print ("Argument %i: %s" % (x, sys.argv[x]))
>     sum = sum + int(sys.argv[x])
>     x = x + 1
> print ("Sum of the numbers is: %i." % sum)
Sum of the numbers is: 0.
> cdsw.track_metric("Sum", sum)
⚠ This output feature is only available within an Experiment.
  For more details, please see the documentation.
> !python3 6_experiment.py 11 1
Argument 1: 11
Argument 2: 1
Sum of the numbers is: 12.
```

You now have a working python file which takes parameters that we can use to run some Experiments.

4. Stop your python session. (Sessions->Stop)
5. There are multiple ways in CDSW to run your Experiments:
 - a. Go back to the main window for your project to run an Experiment.
 - b. From the Workbench you can also select **Run->Run Experiment**.

6. Fill out the screen with your add.py file and sample numbers for the Arguments to sum. Use the 2vCPU/4GiB RAM Python3 engine available for this example.

Hit the **Start Run** button to start the Experiment:

Run New Experiment

Script

6_experiment.py

Arguments ?

44 22

Engine Kernel

☐ Python 2

☒ Python 3

☐ Scala

☐ R

Engine Profile

2 vCPU / 4 GiB Memory

Comment

batch_experiment 1

Cancel

Start Run

7. You will see the Experiment status in the main **Experiments** screen show various lifecycle status; Queued, Building, Scheduling, Running and then Success. The output will be added as a column called “Sum” from your Experiment code shown below. You could also save the output into a file as well or instead.

Account

user00

cdsw-demo

Experiments

Project quick find

+

U user00

Overview

Sessions

Experiments

Models

Experiments

1 metrics

Run Experiment

Run	Script	Arguments	Kernel	Comment	Submitter	Created At	Sum	Status	Duration	Actions
1	6_experiment.py	44 22	python3	batch_experiment 1	user00	2/20/19 9:32 AM	66	Success	0 mins	

8. While it's going through the various status, go ahead and select the Run job (job 1 in above screen shot). If you can, look at the build status before the job completes to see the build process or run another job if you would like to see the build status in process.

Account

Overview

Sessions

Experiments

Models

Jobs

Files

Team

Settings

user00

cdsw-demo

Experiments

1

Build

Run-1

New Experiment

Overview

Session

Build

Sending build context to Docker daemon26 MB

Step 1/5 : FROM docker.repository.cloudera.com/cdsw/engine:6
----> c87ea1f60a1a

Step 2/5 : COPY sources /home/cdsw
----> a2c2f9d7eb09

Removing intermediate container 4363203075c0

Step 3/5 : WORKDIR /home/cdsw
----> 9bec6596daf0

Removing intermediate container 7c5ea40cb69d

Step 4/5 : RUN /home/cdsw/cdsw-build.sh && chown -R 8536:8536 /home/cdsw
----> Running in 5af7aef03dd9

Collecting scikit-learn
Downloading
https://files.pythonhosted.org/packages/0d/3a/b92670f5c368c20329ecc4c25f0.20.2-cp36-cp36m-manylinux1_x86_64.whl (5.4MB)
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python3.6...

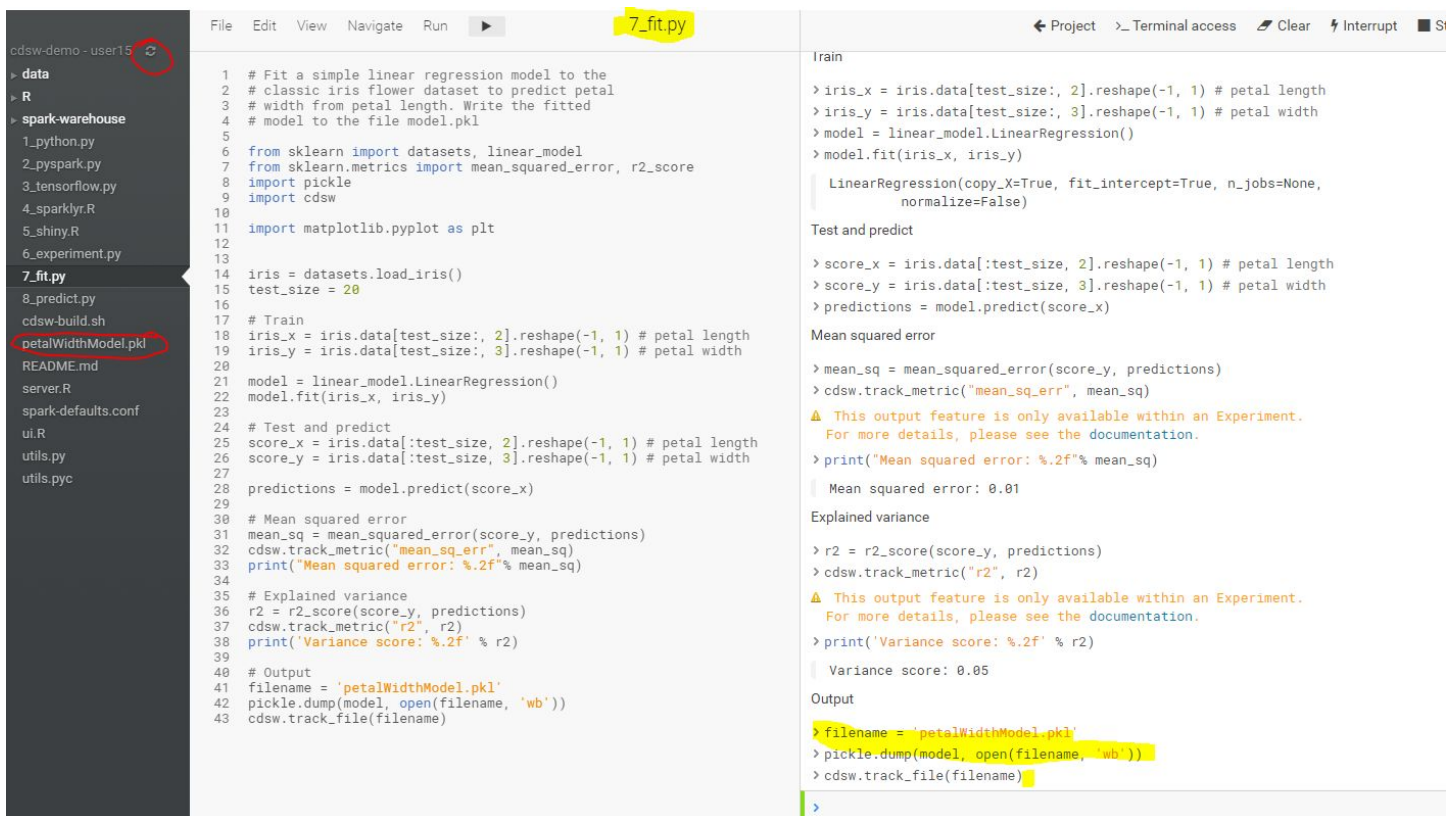
Can you think of examples of batch experiments with different input parameters and project files that would be good to have multiple parallel runs and corresponding output to compare.

Lab 10 - Models

Data scientists often develop models using a variety of Python/R open source packages. The challenge lies in actually exposing those models to stakeholders who can test the model. The data science team also needs a reliable way to retain history of the models they build and ensure that they can rebuild a specific version if needed

This lab will give you practice packaging a trained model into an immutable artifact and deploying models as REST APIs.

1. Open Workbench and start a **Python 3** session using the 2 vCPU/ 4 GiB RAM engine.
2. Open the file **7_fit.py** and run it. You will see the code created a model called “petalWidthModel.pkl”:



```
File Edit View Navigate Run 7_fit.py Project Terminal access Clear Interrupt SI

cdsw-demo - user15
> data
> R
> spark-warehouse
1_python.py
2_pyspark.py
3_tensorflow.py
4_sparklyr.R
5_shiny.R
6_experiment.py
7_fit.py
8_predict.py
cdsw-build.sh
petalWidthModel.pkl
README.md
server.R
spark-defaults.conf
ui.R
utils.py
utils.pyc

1 # Fit a simple linear regression model to the
2 # classic iris flower dataset to predict petal
3 # width from petal length. Write the fitted
4 # model to the file model.pkl
5
6 from sklearn import datasets, linear_model
7 from sklearn.metrics import mean_squared_error, r2_score
8 import pickle
9 import cdsw
10
11 import matplotlib.pyplot as plt
12
13
14 iris = datasets.load_iris()
15 test_size = 20
16
17 # Train
18 iris_x = iris.data[test_size:, 2].reshape(-1, 1) # petal length
19 iris_y = iris.data[test_size:, 3].reshape(-1, 1) # petal width
20
21 model = linear_model.LinearRegression()
22 model.fit(iris_x, iris_y)
23
24 # Test and predict
25 score_x = iris.data[:test_size, 2].reshape(-1, 1) # petal length
26 score_y = iris.data[:test_size, 3].reshape(-1, 1) # petal width
27
28 predictions = model.predict(score_x)
29
30 # Mean squared error
31 mean_sq = mean_squared_error(score_y, predictions)
32 cdsw.track_metric("mean_sq_err", mean_sq)
33 print("Mean squared error: %.2f" % mean_sq)
34
35 # Explained variance
36 r2 = r2_score(score_y, predictions)
37 cdsw.track_metric("r2", r2)
38 print('Variance score: %.2f' % r2)
39
40 # Output
41 filename = 'petalWidthModel.pkl'
42 pickle.dump(model, open(filename, 'wb'))
43 cdsw.track_file(filename)

I train
> iris_x = iris.data[test_size:, 2].reshape(-1, 1) # petal length
> iris_y = iris.data[test_size:, 3].reshape(-1, 1) # petal width
> model = linear_model.LinearRegression()
> model.fit(iris_x, iris_y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

Test and predict
> score_x = iris.data[:test_size, 2].reshape(-1, 1) # petal length
> score_y = iris.data[:test_size, 3].reshape(-1, 1) # petal width
> predictions = model.predict(score_x)

Mean squared error
> mean_sq = mean_squared_error(score_y, predictions)
> cdsw.track_metric("mean_sq_err", mean_sq)
⚠ This output feature is only available within an Experiment.
For more details, please see the documentation.
> print("Mean squared error: %.2f" % mean_sq)
Mean squared error: 0.01

Explained variance
> r2 = r2_score(score_y, predictions)
> cdsw.track_metric("r2", r2)
⚠ This output feature is only available within an Experiment.
For more details, please see the documentation.
> print('Variance score: %.2f' % r2)
Variance score: 0.05

Output
> filename = 'petalWidthModel.pkl'
> pickle.dump(model, open(filename, 'wb'))
> cdsw.track_file(filename)
```

3. Before we deploy this model, let's test it in the workbench. Open file **8_predict.py** and run it. Then, to confirm that the predict() function works as expected, run this line of code by copying and pasting into the workbench prompt, see below screenshot:

```
predict({"petal_length": 19})
```

You should see the output like this:

Read the fitted model from the file model.pkl and define a function that uses the model to predict petal width from petal length

```
> import pickle
> model = pickle.load(open('petalWidthModel.pkl', 'rb'))
> def predict(args):
    iris_x = float(args.get('petal_length'))
    result = model.predict([[iris_x]])
    return result[0][0]
> predict({"petal_length": 19})
7.5427050856062694
```

> |

Now let's deploy the add function to a REST endpoint.

4. Stop your Python 3 Session used in step 3 above.
5. Go to the project overview page and select **Models->New Model** from the left hand bar.
6. Enter a name, description, select the python file **8_predict.py**, the **predict** function, and Example Input as per below screenshot:

user15

cdsw-demo - user15

Models

New Model

Name *

Petal

Description *

petal

Build

File *

8_predict.py

Function *

predict

Example Input ?

```
{
  "petal_length": 19
}
```

Example Output ?

7.542

Kernel

- ☐ Python 2
- ☒ Python 3
- ☐ R

Comment

Enter comment for this build

Deployment

Engine Profile

2 vCPU / 4 GiB Memory

Replicas

1

[Set Environmental Variables](#)

- Click on the model you just added and go to its **Build and Monitoring** pages to watch the build process until it completes.

The screenshot shows the 'Petal' model page in the CDW interface. The top navigation bar includes 'user15', 'cdsw-demo - user15', 'Models', 'Petal', and 'Builds'. A search bar on the right says 'Project quick find'. The left sidebar has icons for Overview, Sessions, Experiments, Models, and Jobs. The main content area has tabs for Overview, Deployments, Builds (selected), Monitoring, and Settings. Below the tabs is a table with columns: Build, Status, File, Function, Kernel, Engine Image, Created By, Created At, Comment, and Actions. The table contains one row with the following data:

Build	Status	File	Function	Kernel	Engine Image	Created By	Created At	Comment	Actions
1	Building	8_predict.py	predict	python3	Base Image v6	user15	Jan 4, 2019, 2:00 PM	Initial revision.	Delete

- Once the model has been built and **deployed**, go back to the model **Overview** page and use the **Test Model** widget to make sure the model works as expected.

You can test using Shell, Python, and R to see the various ways the model can be accessed using a rest a REST interface:

Description

d

Sample Code

Shell

Python

R

```
curl -H "Content-Type: application/json" -X POST http://cdsw.cloudera-pa
s/call-model -d '{"accessKey":"menqhyrhaanj86cd8wa9z4iy8kmsrme","request"
```

Test Model

Input

```
{
  "petal_length": 19
}
```

Test

Reset

Result

Status	● success
Response	7.542705085606269
Replica ID	pp-9-14-686559ddb5-cnr66

Each model in Cloudera Data Science Workbench has a unique access key associated with it. This access key serves two purposes: 1) it is a unique identifier for the model, and 2) it serves as an authentication token that allows you to make calls to the model.

9. Note the access key used in the test examples. You would need this access key to use the Model from a REST API. Click on the Setting tab in your Model to see your key and the option for Regenerating a new Model Access Key.
10. Now try on your own to see if you can figure out how to use the model from your environment.
11. Stop your Model deployment when finished with this lab.

Lab 11 - Scheduling Jobs

It's often the case that you need to execute tasks on a periodic basis, and to execute one or more tasks once some other task has succeeded. Obviously there are sophisticated workflow engines many customers are already using. For simple workflows CDSW has a handy jobs system built in. Jobs can also be exposed for use from other scheduling systems.

This lab goes through the mechanics of creating a simple multi-step job process.

1. Open up your '**cdsw demo**' project to get to this screen:

The screenshot shows the Cloudera Data Science Workbench (CDSW) interface. The top navigation bar includes 'Account', 'cdsw workshop', and a search bar. The left sidebar contains links for 'Overview', 'Jobs', 'Sessions', 'Files', 'Team', and 'Settings'. The main content area is titled 'cdsw workshop' and includes a 'Jobs' section with the text 'This project has no jobs yet. Create a new job to document your analytics pipelines.' Below this is a 'Files' section with a table of files. The 'Workbench' section at the bottom provides information about the scripts available for interactive use.

Name	Size	Last Modified
data	-	15 minutes ago
R	-	11 minutes ago
1_python.py	3.00 kB	15 minutes ago
2_pyspark.py	1.65 kB	15 minutes ago
3_tensorflow.py	3.39 kB	15 minutes ago
4_sparklyt.R	2.52 kB	15 minutes ago
5_shiny.R	769 B	15 minutes ago
README.md	1.51 kB	15 minutes ago
requirements.txt	2.77 kB	15 minutes ago
server.R	536 B	15 minutes ago
ui.R	689 B	15 minutes ago
utils.py	1.09 kB	15 minutes ago

Cloudera Data Science Workbench demos
Basic tour of Cloudera Data Science Workbench.

Workbench

There are 4 scripts provided which walk through the interactive capabilities of Cloudera Data Science Workbench.

1. Basic Python visualizations (Python 2). Demonstrates:
 - Markdown via comments
 - Jupyter-compatible visualizations

- 2.
3. Select the 'new job' link in the middle of the page, and you'll get to the following screen (there are other ways of getting to this next screen - its an exercise for the student to figure out what they might be):

4.

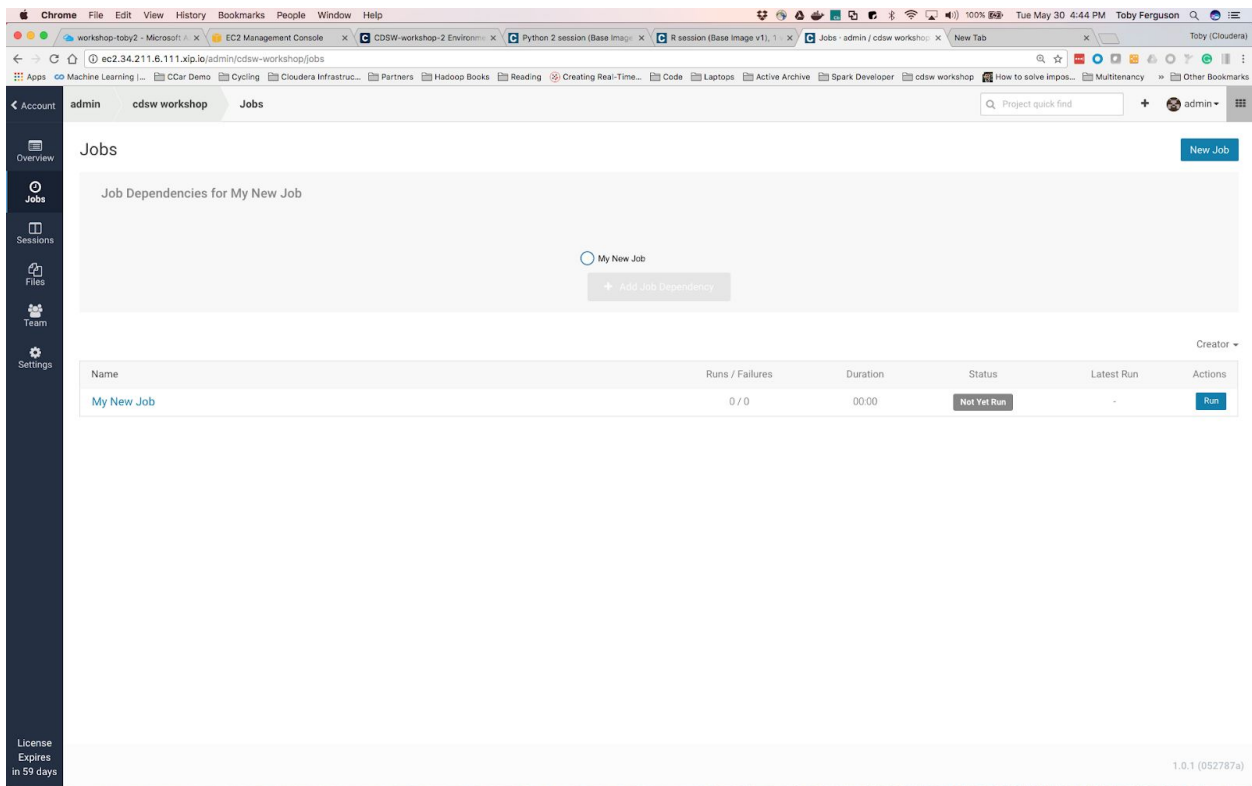
5. Create a job that will be triggered manually and will execute the 1_python.py. Here are the parameters to do that:

6.

Name	My New Job
Script	1_python.py
Engine Kernel	Python 3

7.

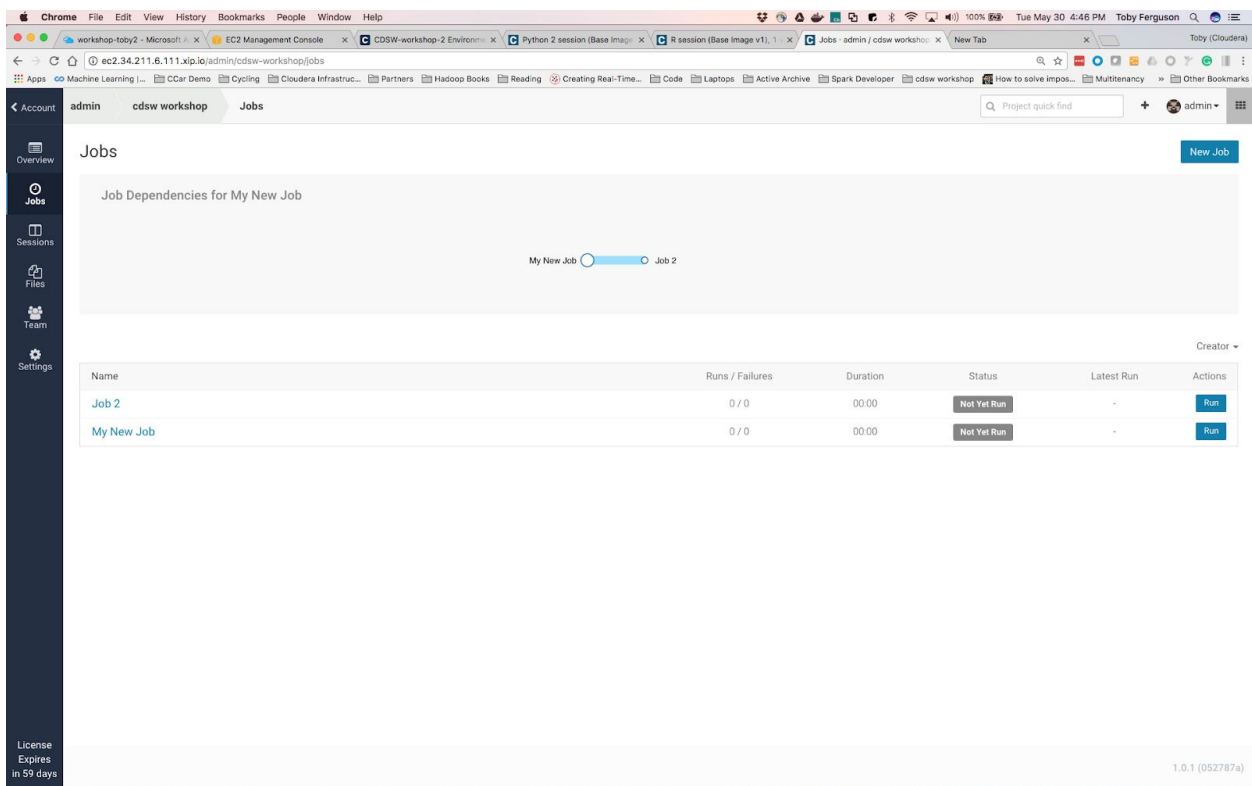
8. Leave everything else as default. Scroll down and hit 'Create Job'. You should get to this screen:



9. Here you can see that you have a job ('My New Job'). It's never been run, and it has no dependencies.
10. Let's make other jobs depend on this one: Click the '+ Add Job Dependency' grayed out button and add a new job that has a dependency on 'My New Job'. The parameters are:
- 11.

Name	Job 2
Script	2_pyspark.py
Engine Kernel	Python 2

12. Scroll down and 'Create Job'. You'll now see a page like this:



13. So here we can see that 'Job 2' depends upon 'My New Job' (although you can run each manually, if you so choose).
14. Lets add another job that will run in parallel with Job2:
15. Click 'New Job' in the top right corner and create another job that depends upon 'My New Job'. The parameters you'll need are:
- 16.

Name	R Job
Script	4_sparklyr.R
Engine Kernel	R
Schedule	Dependent / My New Job

17. Create the job and you'll see this:

The screenshot shows the CDW Jobs interface. At the top, there's a 'Jobs' tab. Below it, a section titled 'Job Dependencies for My New Job' shows a dependency graph where 'My New Job' depends on 'Job 2' and 'R job'. Below this is a table of jobs:

Name	Runs / Failures	Duration	Status	Latest Run	Actions
R job	0 / 0	00:00	Not Yet Run	-	Run
Job 2	0 / 0	00:00	Not Yet Run	-	Run
My New Job	0 / 0	00:00	Not Yet Run	-	Run

The interface also shows a sidebar with navigation options like Overview, Jobs, Sessions, Files, Team, and Settings. At the bottom, it indicates the license expires in 59 days.

18. Lets run it all - hit the 'Run' button next to 'My New Job' (bottom of the list of jobs). You should see the job get scheduled, run, complete, and then the next two jobs should likewise get scheduled, run and complete:

The screenshot shows the CDW Jobs interface after running the jobs. The table now shows that all three jobs have completed successfully:

Name	Runs / Failures	Duration	Status	Latest Run	Actions
Job 2	1 / 0	00:29	Success	just now	Run
R job	1 / 0	01:19	Success	just now	Run
My New Job	1 / 0	00:03	Success	just now	Run

The interface remains the same as the previous screenshot, with the same sidebar and navigation options.

Question: How will a job scheduler reduce the effort required for you to build simple pipelines?

Question: What other facilities surrounding a job did we not explain? What do you think those other parameters might do?

That's It Folks

Thanks for working your way through the labs and we hope you learned how CDSW might make your Data Science journey easier. Please fill out the survey form!

Appendix

Using your own CDSW cluster

If you're using these instructions as part of a Self Service training then you'll have constructed your own CDSW cluster and the fully qualified domain name (FQDN) of your CDSW instance is what you configured in the [Set Up a Wildcard DNS Subdomain](#) section of the installation instructions. If you have root access to the CDSW instance you can execute the following, and the FQDN is the value of the 'DOMAIN' key/value pair¹:

```
grep '^DOMAIN=.*' /etc/cdsw/config/cdsw.conf
```

To install a demo CDSW cluster in MS Azure, you can follow the example how to guide <https://github.com/fabiog1901/cdsw-install>

The original project was taken from <https://github.com/fabiog1901/cdsw-demo>

¹ Only true for CDSW versions installed via packages and not using Cloudera Manager and parcels.

Troubleshooting

DNS Issue

If you see something like this:

Python 2 session (Base Image v1), 1 vCPU (burstable), 2 GiB memory, on 6/2 at 13:38   Collapse  Share Running
By Toby Ferguson — Python 2 Session — just now

Getting Started

This is your **Python 2 session**. Your **editor** is on the left and your **input prompt** is on the bottom.

To install a package type: `!pip install [package_name]` at the input prompt.

To execute code from the editor, select the code and execute it with `Command-Enter` on Mac or `Ctrl-Enter` on Windows. You can also enter code at the prompt below.

Use `?command` to get help on a particular command.



Or (when you open a terminal), like this:

Then just refresh your browser.

On Windows you might have to flush your dns cache. To do that open up a command prompt and then execute

```
ipconfig /flushdns
```

The root cause of this is that the DNS database on our DNS provider (xip.io) are being actively updated but that takes time (a few seconds) and you're requesting a DNS record that hasn't yet been added. So requesting again after a little while is a sensible thing to do ... and might need to be repeated, depending on how quickly the records are updated.

Spark R backend might have failed

```
✖ Error in invoke_method.spark_shell_connection(sc, TRUE, class, method, :  
  No status is returned. Spark R backend might have failed.
```

Or:

```

✖ Error in invoke_method.spark_shell_connection(sc, TRUE, class, method, :
  No status is returned. Spark R backend might have failed.
⚠ Engine exhausted available memory, consider a larger engine size.

```

Or

```

✖ Error in invoke_method.spark_shell_connection(spark_connection(jobj), :
  No status is returned. Spark R backend might have failed.
In addition: Warning messages:
1: Translator is missing window functions:
cor, count, cov, n_distinct, sd
2: Translator is missing window functions:
cor, count, cov, n_distinct, sd
3: Translator is missing window functions:
cor, count, cov, n_distinct, sd

```

Then you likely chose a 2G engine - stop that session and start another one, this time with a 4G session

At times we are seeing connections problems with the dynamic DNS configuration we are using specifically for this lab setup. If you experience ongoing connection problems, try stopping your sessions and restarting. That appears to re-establish connections through the dynamic DNS.

ImportError: No module named google.protobuf

If you see an error like this:

```

> import tensorflow as tf
✖ ImportError: No module named google.protobuf
✖ ImportErrorTraceback (most recent call last)
in engine
----> 1 import tensorflow as tf

/home/cdsw/.local/lib/python2.7/site-packages/tensorflow/__init__.py in <module>()
22
23 # pylint: disable=wildcard-import
----> 24 from tensorflow.python import *
25 # pylint: enable=wildcard-import
26

```

And you're running Lab 6, then you've probably forgotten to stop your current python session and start a new one. There's a bug in the Tensorflow libraries in the way that the installed packages get resolved. The simplest workaround is to start a new session!