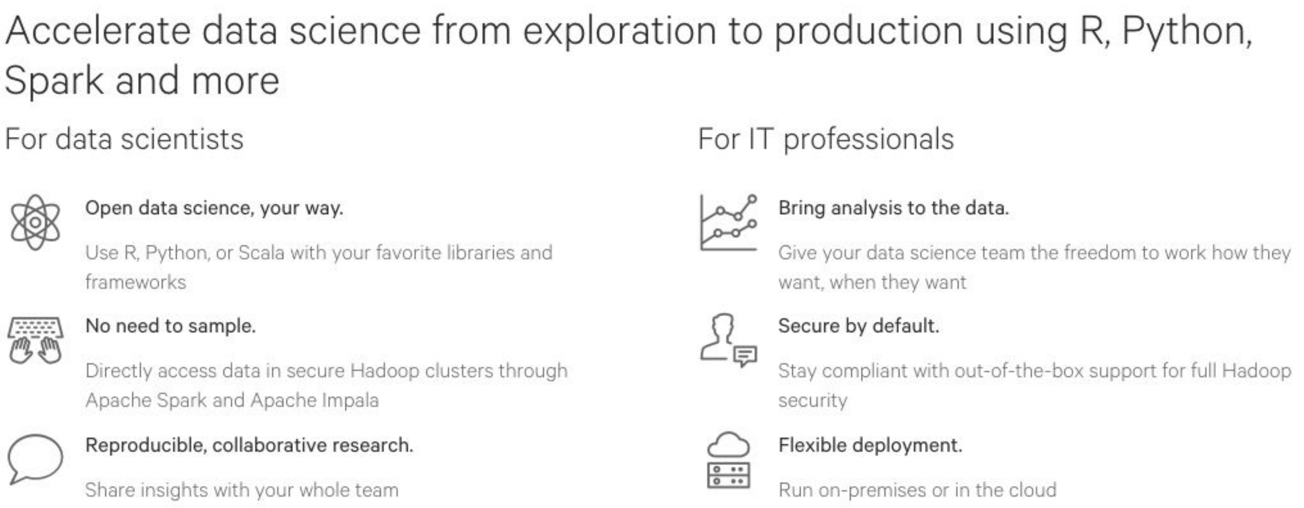
Document Title: Cloudera Data Science lab guide
Date: January 2019
Version: 3.0
Authors: Johannes (Joep) Muselaers - Sales Engineer - Nordics

Cloudera Data Science Workbench Labs

Table of Contents

Introduction

Cloudera Data Science workbench is a new product from Cloudera launched in May 2017. It is based on the acquisition of Sense.io that we made in March 2016. Cloudera has taken this product enhanced it and ensures that all workloads can be pushed down to Cloudera.



Cloudera Data Science workbench supports the R, Python, Scala programming languages. That capability could certainly be useful to Cloudera; the software could enable companies to make the most of their data scientists, who can then be more efficient with their use of company time and infrastructure.

Cloudera's goal with Cloudera Data Science workbench is to Help more data scientists use the power of Hadoop, make it easy and secure to add new users, use cases.

Why Hadoop for Data Science well here are the reasons:
High volume, low cost shared storage – More data more kinds of data
Parallel compute local to the data – more experiments, better results
Scalable, fault tolerant – easy to scale out, not just scale up
Flexible multipurpose data platform – easier path to production
Superior flexibility and price / performance to any other data platform

Lab 1 - Login to Cloudera Data Science Workbench (CDSW)

In this lab you'll learn how to:

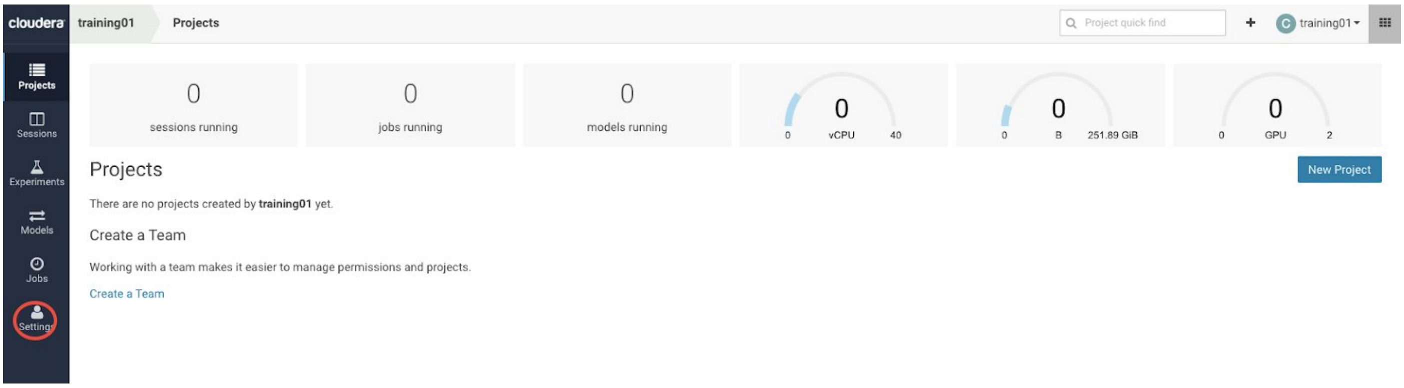
- Login to a Cloudera Data Science Workbench instance
- Set your Hadoop Authentication
- Navigate the Cloudera Data Science Workbench application

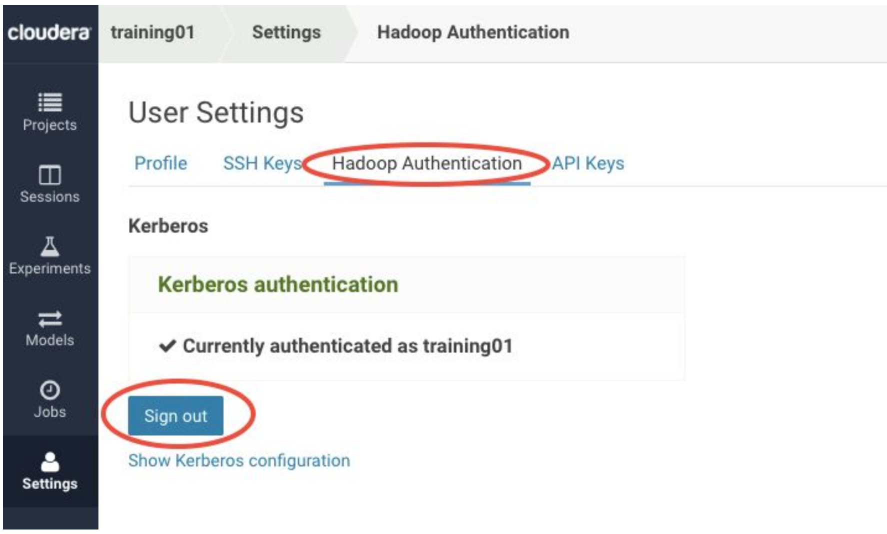
First thing you need to do is register onto Cloudera Data Science Workbench

URL: **`https://cdsw.cloudera.tellarius.eu/`**

Login with the provided credentials, like user trainingxx and a password.

When you are logged in, you need to specify the credentials to connect to the Cloudera cluster. Click on the 'Settings' button, and then the "Hadoop Authentication" link.

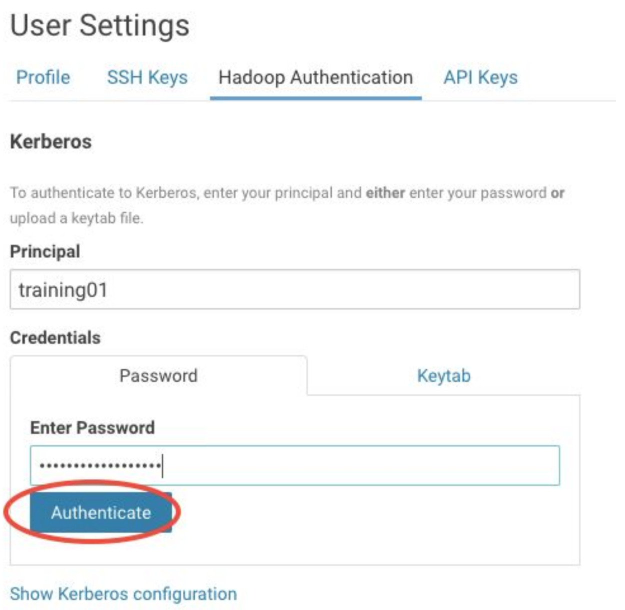




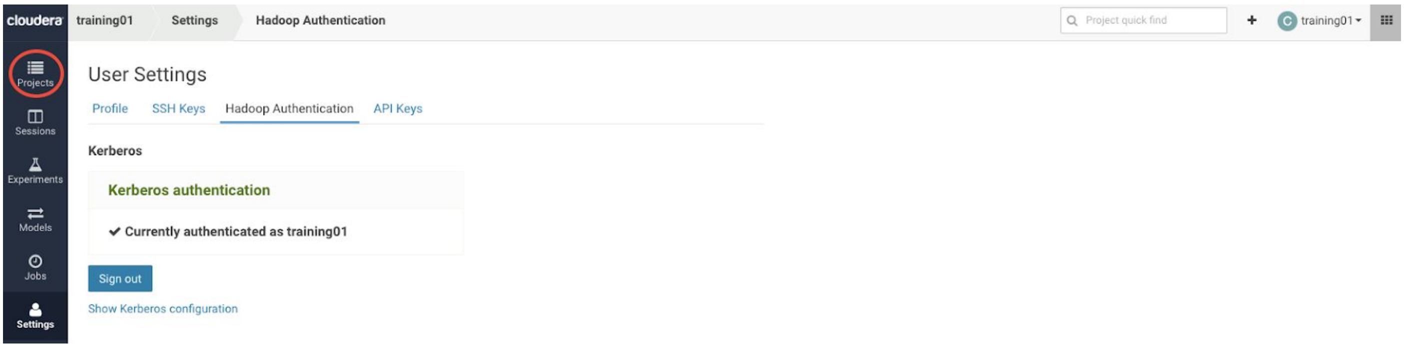
Since the password of your hadoop authentication most probably has changed, you need to hit the 'Sign out' button.

cloudera®

Now, authenticate to the cluster using the same credentials that you used when you logged in into the Data Science Workbench.

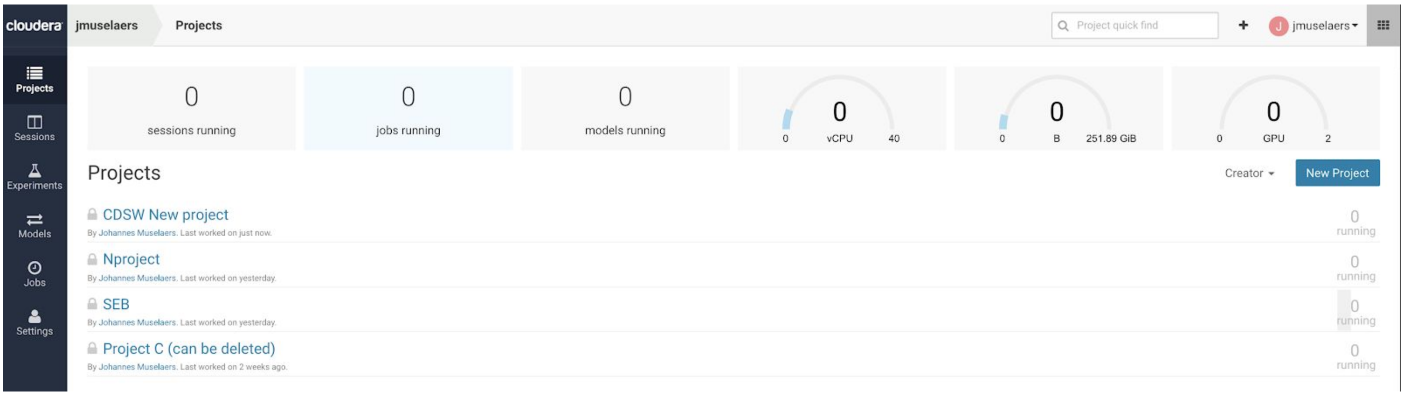


If you authenticated successfully, navigate back to the project space by clicking on the 'Projects' button.



Lab 2 - Creating a new project

You will see the project window as follows



You have on the left hand panel

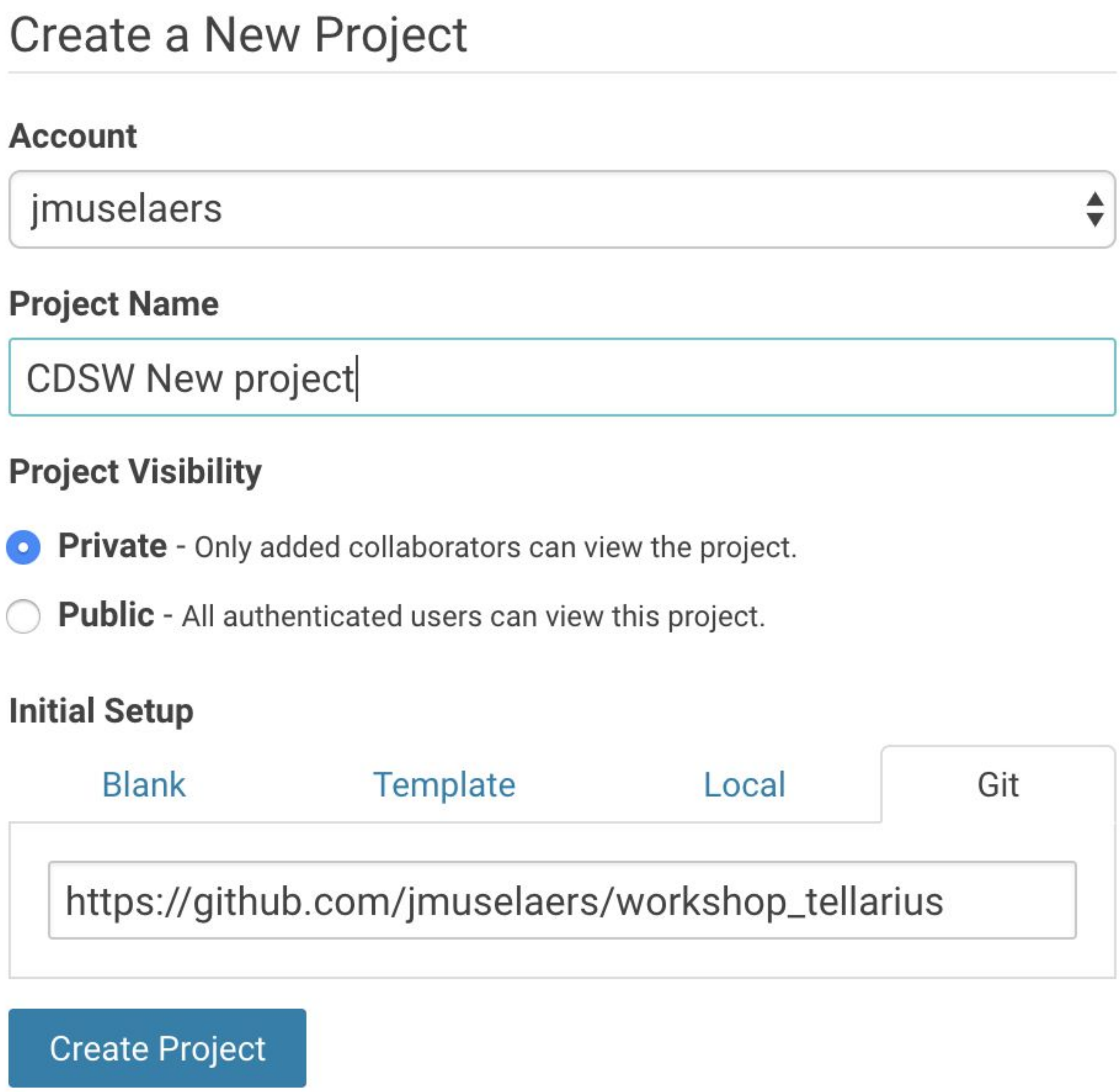
Projects - where you create data science projects
Jobs - Run and schedule jobs and add dependencies
Sessions - Python, Scala or R sessions
Experiments - batch experiments
Models - build, deploy, and manage models as REST APIs to serve predictions
Settings - User, Hadoop Authentication, SSH Keys and permission settings

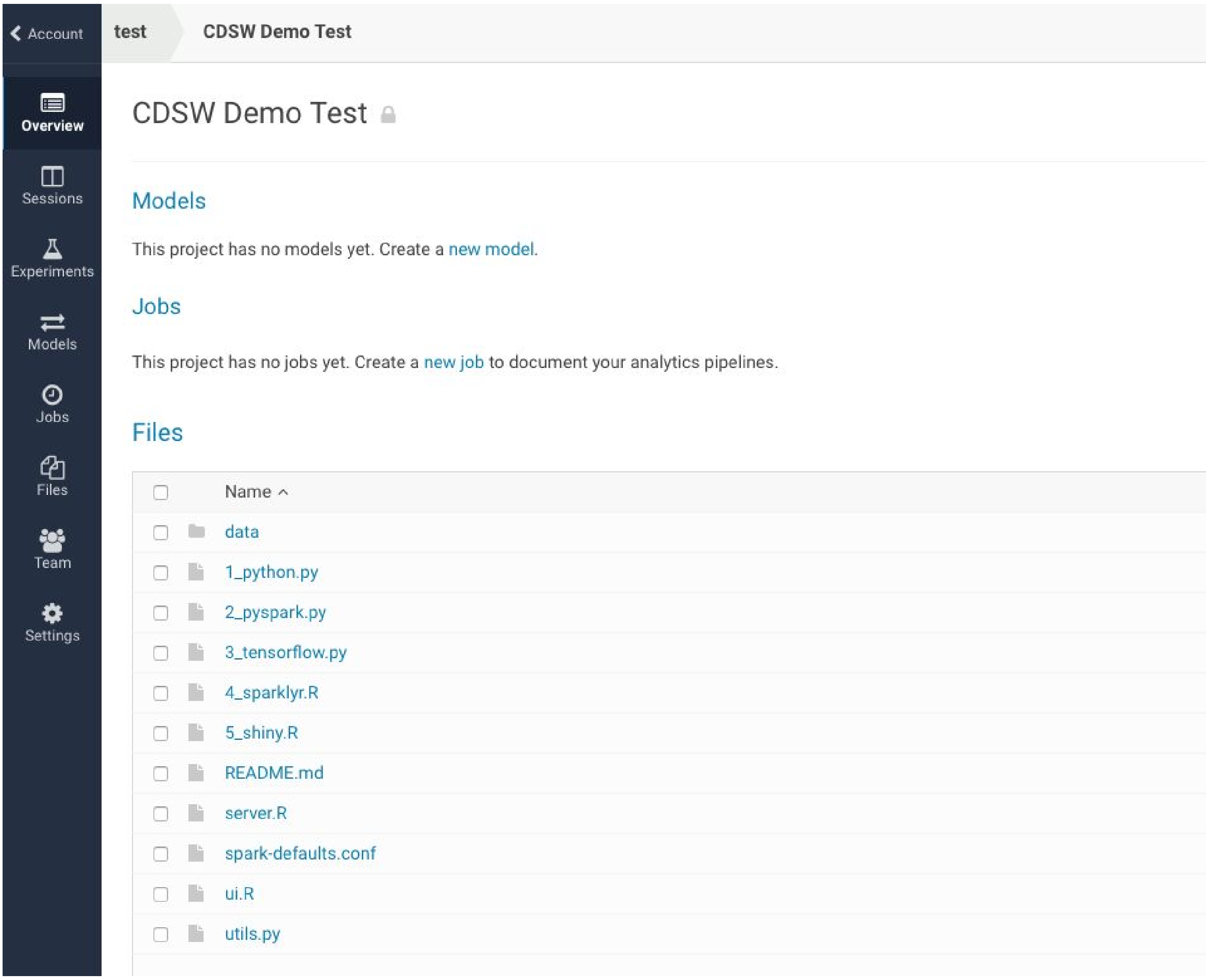
In the top right hand corner you have
Search bar - for search for projects
+ adding new projects or new teams
User name - Account settings and Sign out - Same as settings in home screen

Let’s create a new Project

Copy and paste this GitHub URL into the Git tab
https://github.com/jmuselaers/workshop_tellarius.git

Project_name = your user name and labs
You should see this

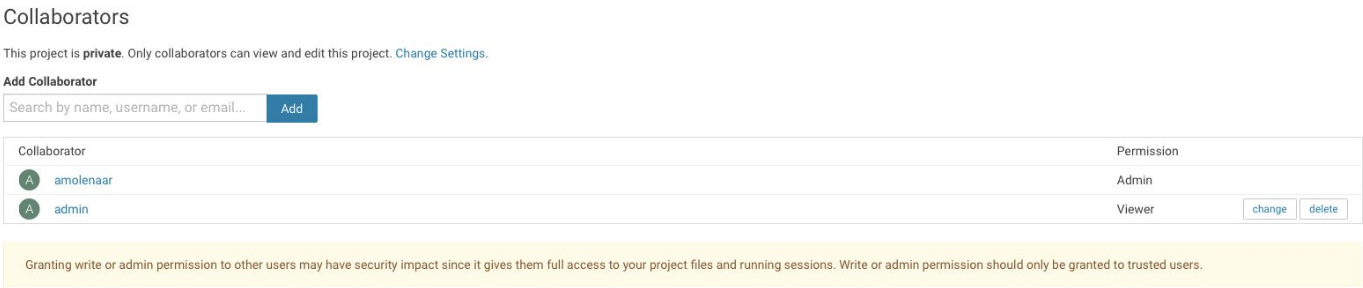

Create a New Project
Account
jmuselaers
Project Name
CDSW New project
Project Visibility
Private - Only added collaborators can view the project.
Public - All authenticated users can view this project.
Initial Setup
Blank
Template
Local
Git
https://github.com/jmuselaers/workshop_tellarius
Create Project



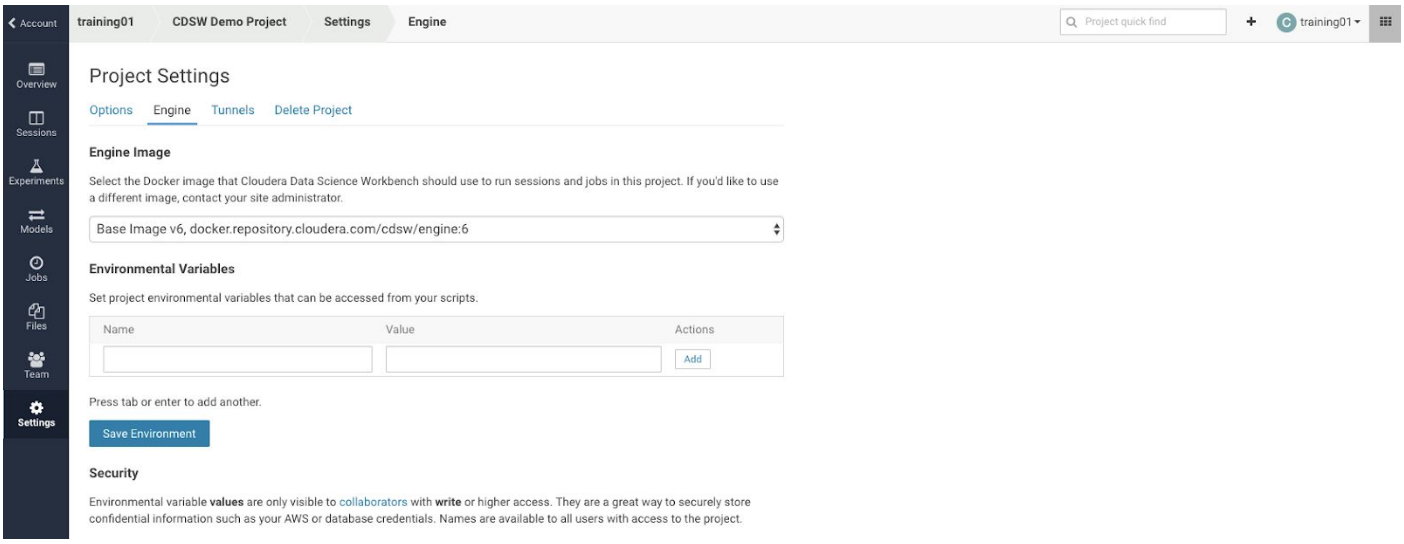
On the left hand side panel you will see a new menu items, among them Team where you can add team members to your project. Ask your neighbor for his or her username, and start typing in the search box.

As an example

You can add anybody to your project. If you cannot find your neighbor, you can use the 'admin' user to share your project with.



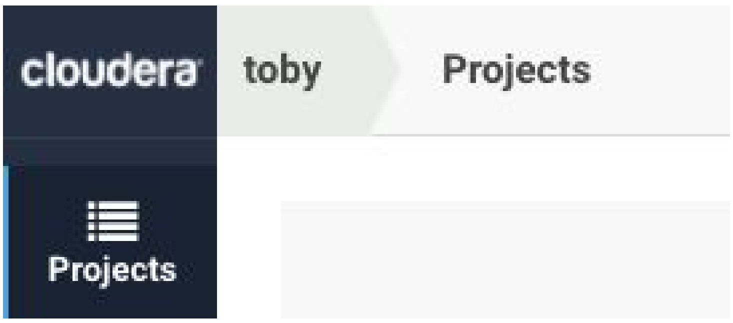
Click on the Settings icon and then the Engine tab:



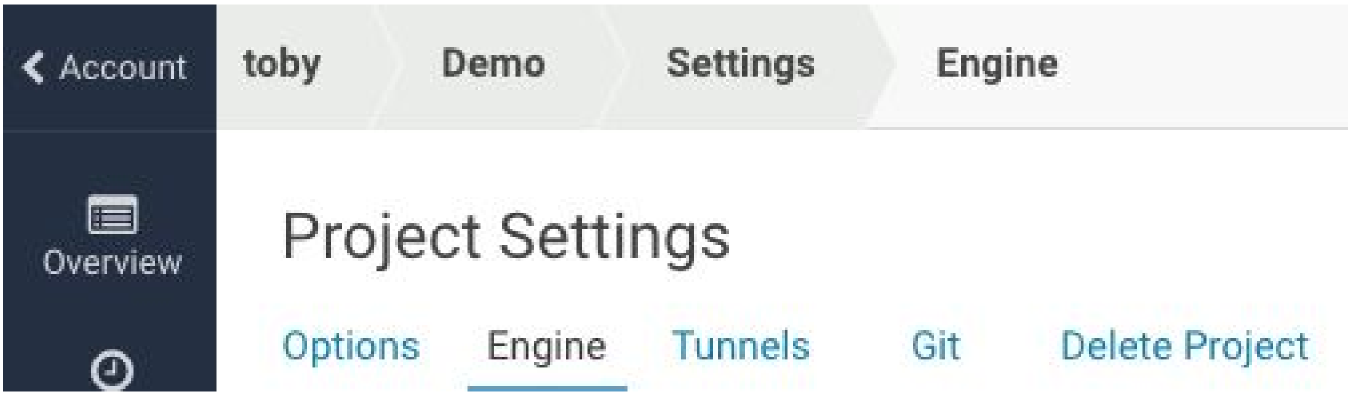
Under Project Settings you will see:
Options - Project Name and Description, Private or Public
Engine - Engine image and environment variables
Tunnels - SSH tunnels allow you to easily connect to firewalled resources such as databases or Hadoop

And this is where if you have to (please don't!) **Delete Project**

Cloudera - will appear if you are at the top level

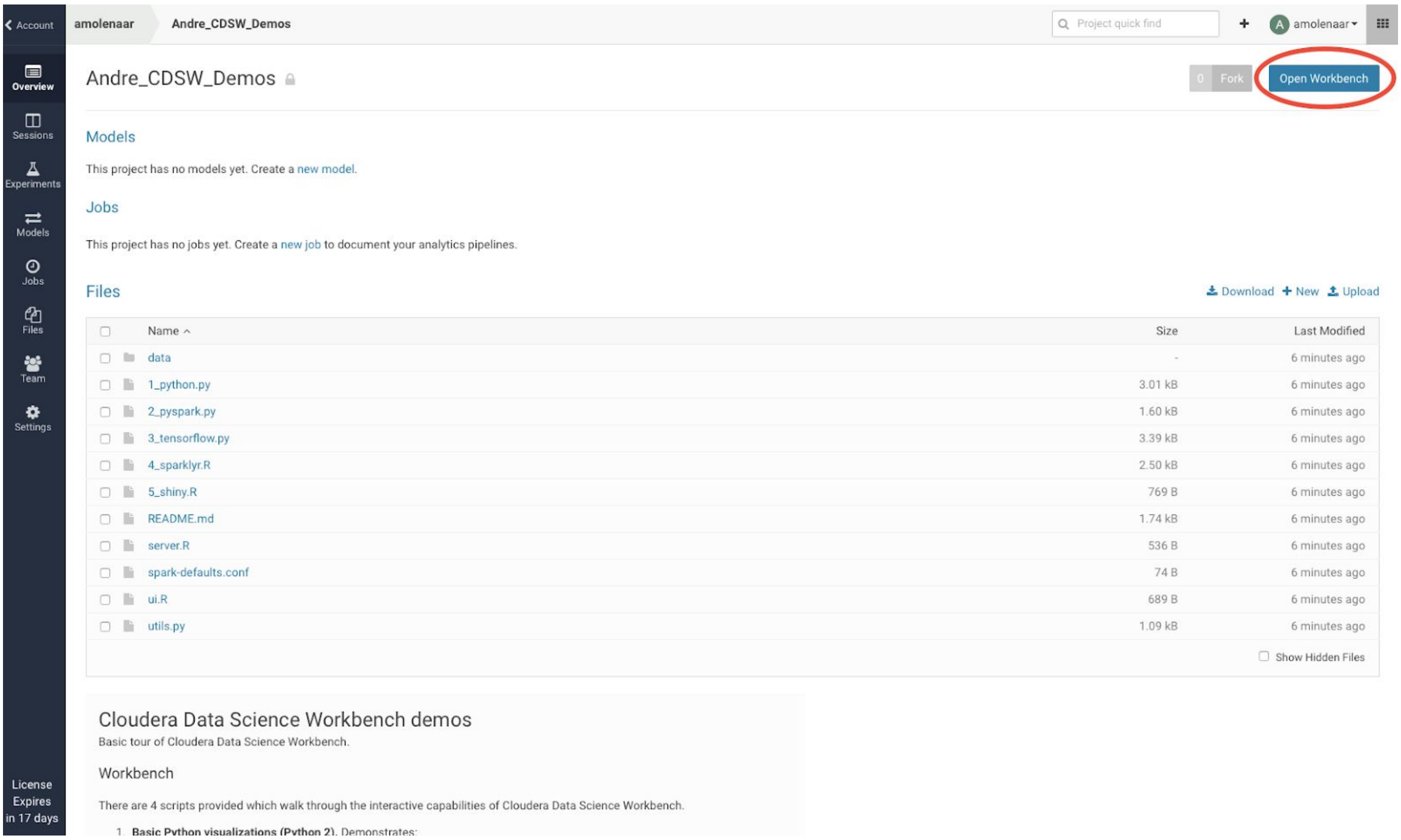


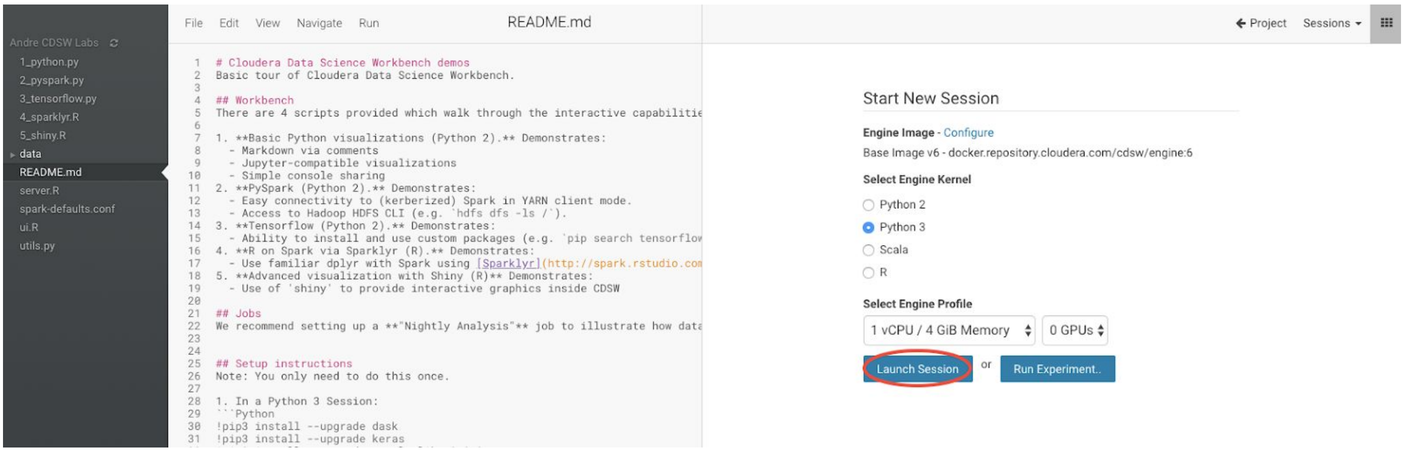
< Account - will appear if you are at the project level

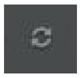


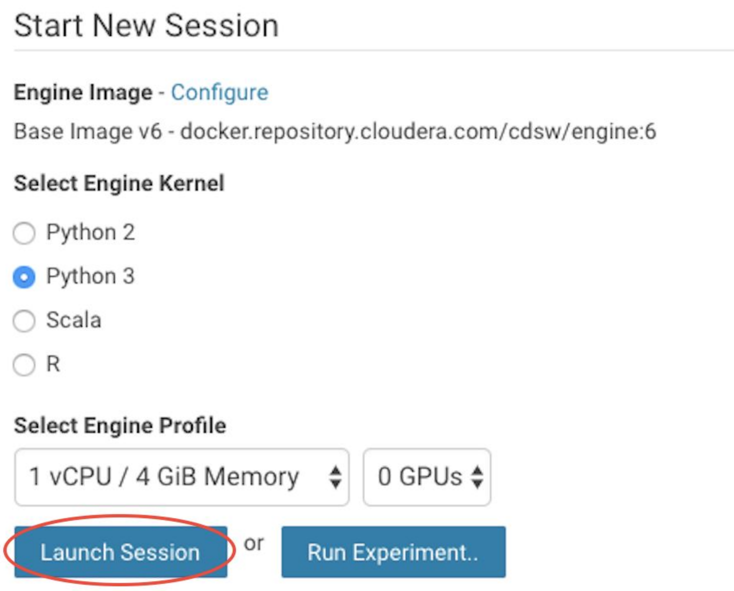
Click on the 'Overview' button.

Now, Launch a Python 3 Session as shown below:





1. On the far left is a file browser (note the little 'refresh' icon at the top: )
2. In the middle is an editor open on the file that you selected - in this case the README.md file.
3. 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).
4. Select the Python 3 kernel; select the 1 vCPU/4GiB Engine (and use this size engine for all your Python sessions in this workshop) and then the Launch Session button.



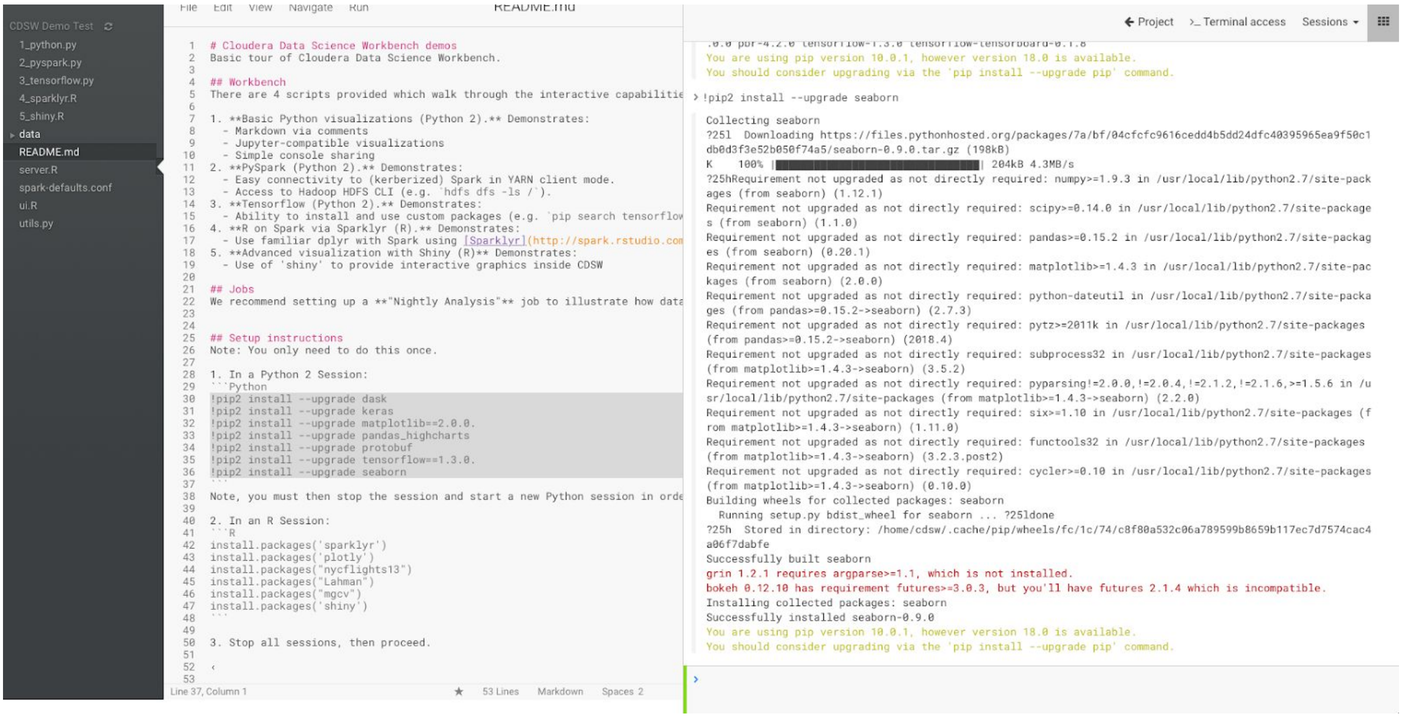
5. This will startup a Python engine and the right hand side will become two tiles. The top one is an output tile and the bottom one (with a red, then green left hand border) is a shell input window.

6. Look at the line 30 to 36 of README.md:

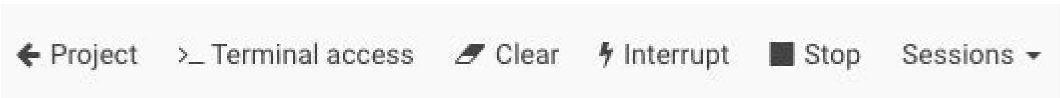
```
!pip2 install --upgrade dask
!pip2 install --upgrade keras
!pip2 install --upgrade matplotlib==2.0.0.
!pip2 install --upgrade pandas_highcharts
!pip2 install --upgrade protobuf
!pip2 install --upgrade tensorflow==1.3.0.
!pip2 install --upgrade seaborn
```

7. 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 to install or upgrade the listed python packages
8. Make a block selection of these lines, and select ‘Run Line(s)’:

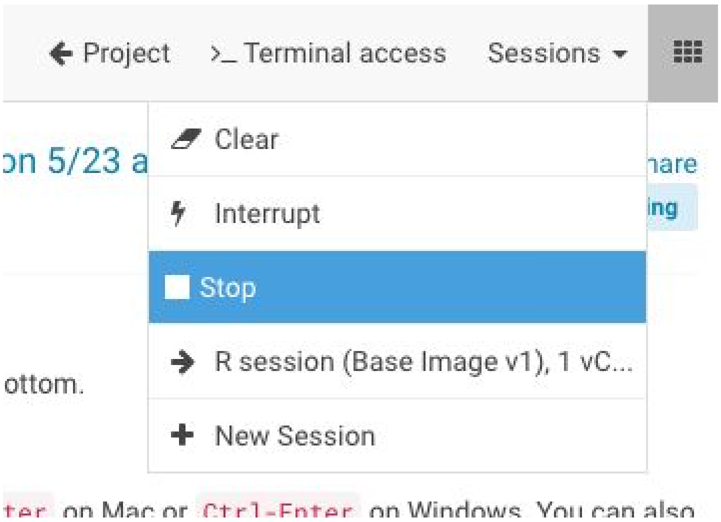
9. The cursor on the left should turn to red and, after a few seconds, you should see output like this:



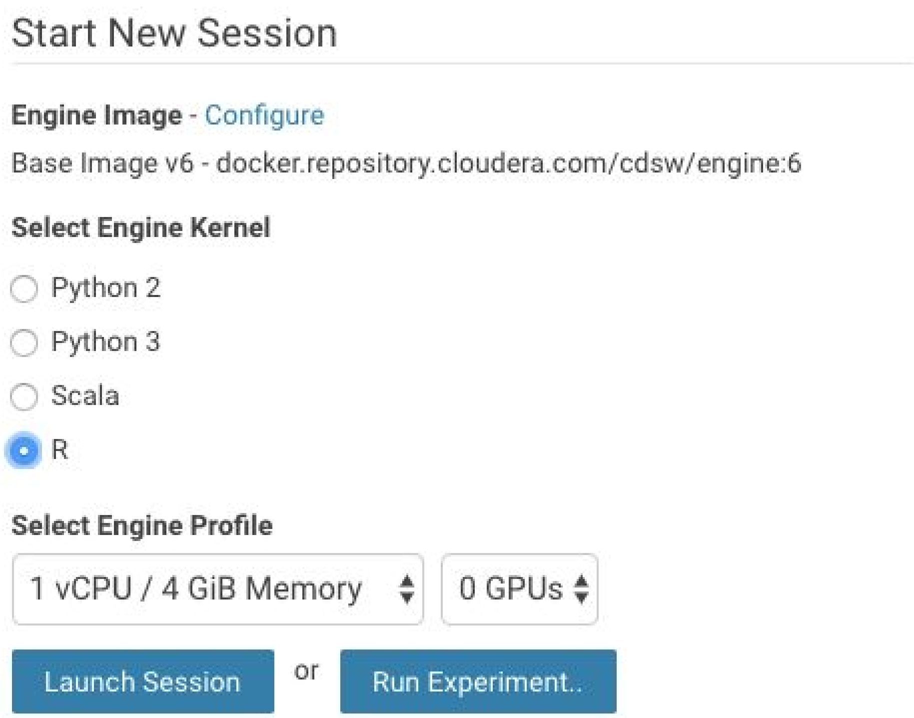
10. What's happening here is that the code in the file is being executed in the console (did you notice the left hand edge turned red?) and now you can see the output in the right hand screen.
11. Stop this session. The Stop button is either on the top menu bar (when there's sufficient room for it):



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



13. Now launch an R session. **Use a 1vCPU, 4GiB Engine**. Use this size engine for **all** your R sessions during this workshop.



14. This time we're going to execute lines 42 through 47 in the R session (these numbers could be off by 1 or so … look at the images below and figure out what you need to select). Select and highlight just these lines then select Run Lines to execute as in prior step:

```
2. In an R Session:
```R
install.packages('sparklyr')
install.packages('plotly')
install.packages("nycflights13")
install.packages("Lahman")
install.packages("mgcv")
install.packages('shiny')
```

This process will take 2 or 3 minutes because code is being downloaded, compiled and installed into your project's workbench.

Note that this workbench is independent of any other project's workbench. You want to try out different and conflicting libraries? Go for 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!
```

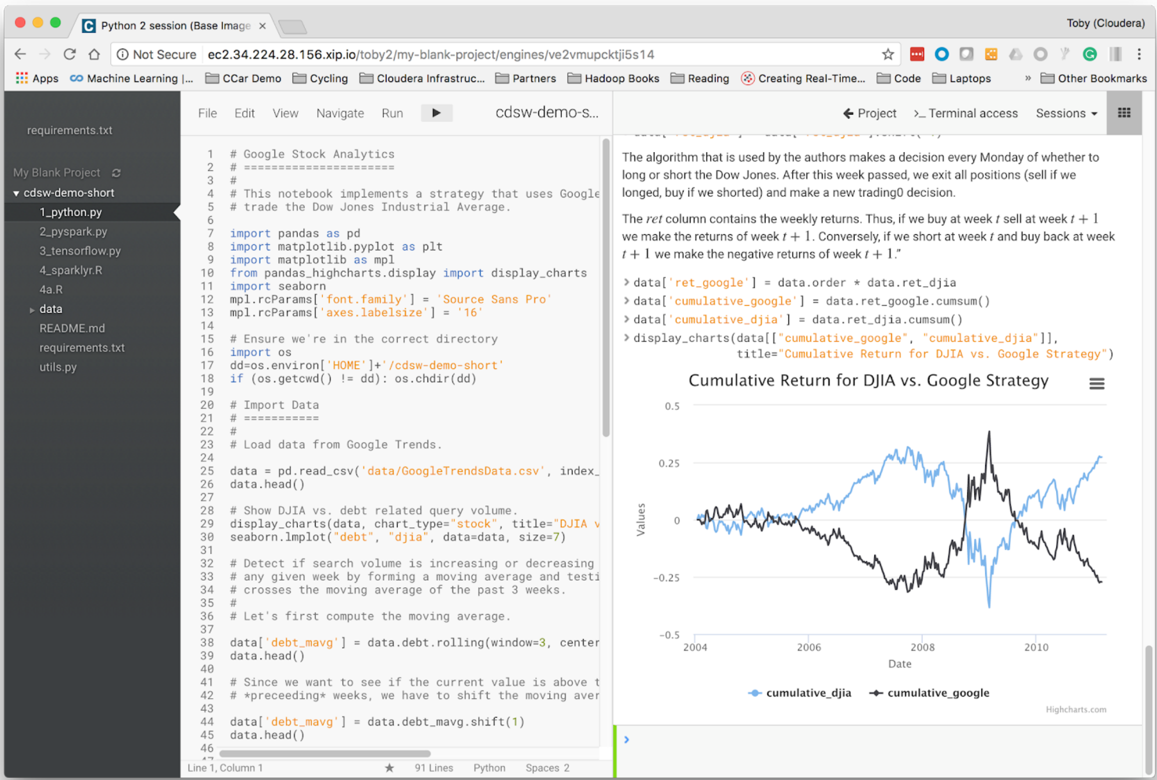
Lab 3 - Visualization and Sharing

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:

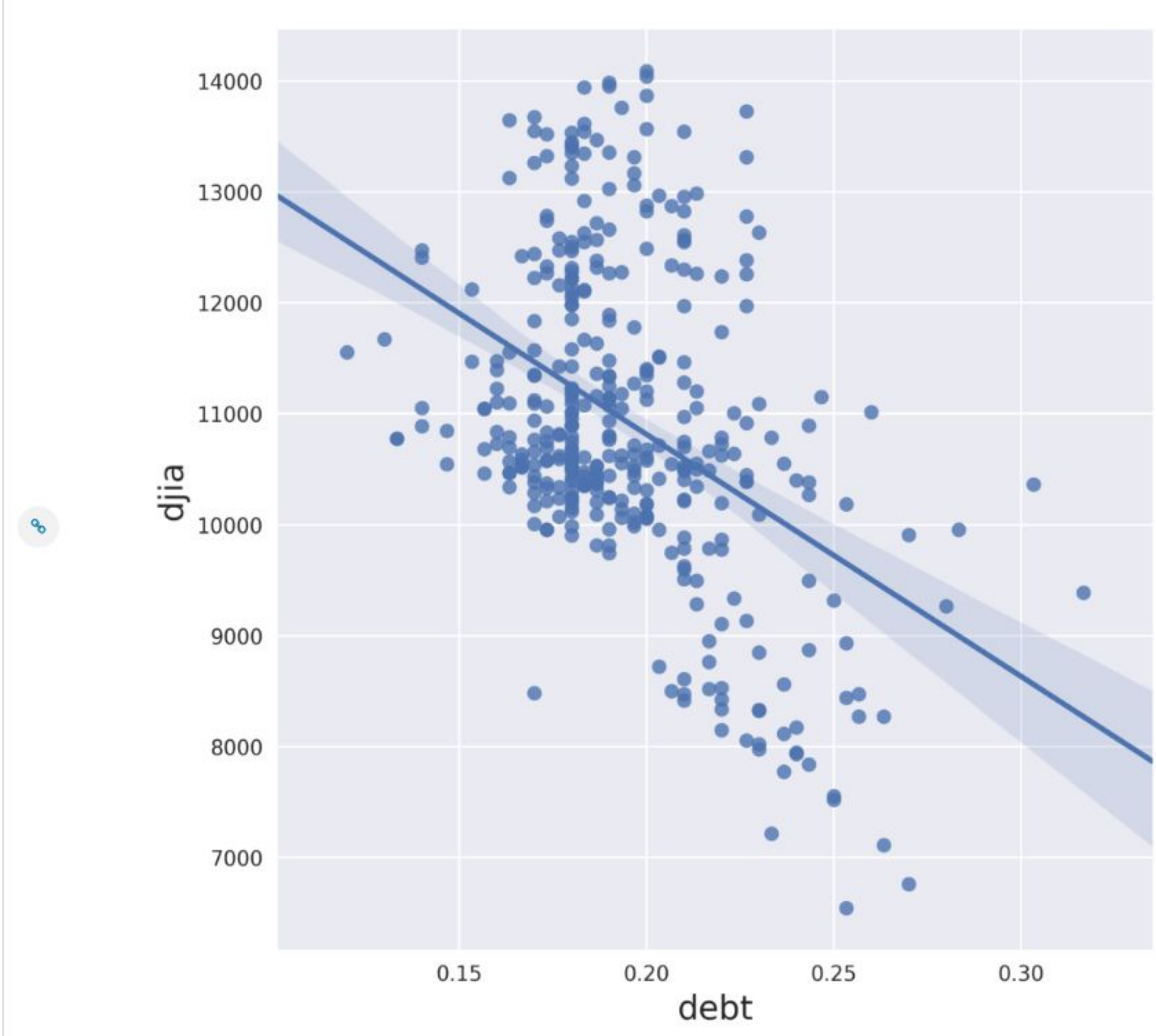
1. Start up a new Python 3 session (1vCPU, 4GiB) in the same manner you did before.
2. Select 1_python.py in the file browser
3. Run the entire file (multiple ways of doing that - try to figure out more than one way. It should be pretty obvious!).

4. You should end up with some nice graphs in the output window:



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 web site:

Copy and paste to embed this cell in your website!

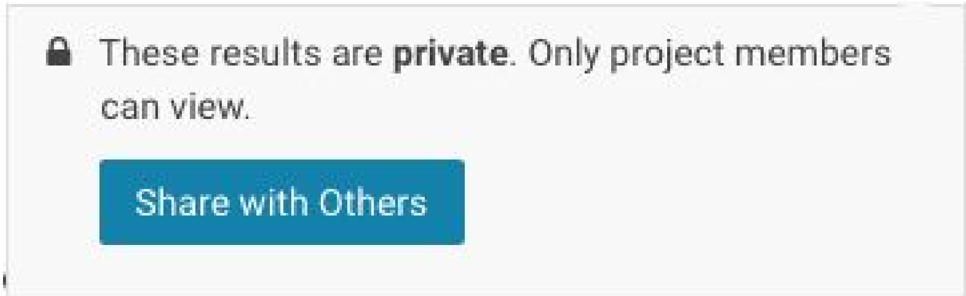
width (optional)

height (optional)

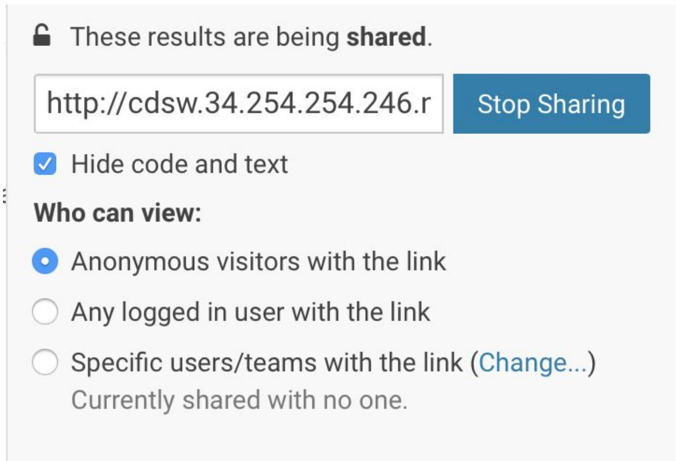
```
<div id='sense-widget-s7dkodjq' class="sense-embed-container" style="width:100%;"><script id='embedding-script-21' src='http://consoles.cdsw.52.214.150.212.nip.io/js/sense-embed.js' data-cell-url='http://livelog.cdsw.52.214.150.212.nip.io/topics/xeupfj2oeyaktdpa_output/21' data-auth-token='eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRoIjp7InJlYWQiOlsieGV1cGZqMm9leWFrdGRwYV8qIl0sIndyaXRlIjpb
data-assets-cdn-root='http://consoles.cdsw.52.214.150.212.nip.io/0/7/xeupfj2oeyaktdpa/' data-widget='sense-widget-s7dkodjq' data-width="" data-height=""></script></div>
```

9. Scroll to the top of the window and you'll see this on the far right (the exact layout depends upon the real estate available – you might have to expand your browser window to see the following links and they might be laid out vertically or horizontally):

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:



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



15. Cut and paste that link and put it into some other browser (best to be a completely different browser 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!)

cdsw.52.214.150.212.nip.io

Python 2 session (Base Image v2), 1 vCPU / 2 GiB Memory, on 9/6 at 9:26

Expand

Running

By Colm — Python 2 Session (Base Image v2) — 4 minutes ago for running

Google Stock Analytics

This notebook implements a strategy that uses Google Trends data to trade the Dow Jones Industrial Average.

Ensure we're in the correct directory

Import Data

Load data from Google Trends.

	djia	debt
Date		
2004-01-14	10485.18	0.210000
2004-01-22	10528.66	0.210000
2004-01-28	10702.51	0.210000
2004-02-04	10499.18	0.213333
2004-02-11	10579.03	0.200000

Show DJIA vs. debt related query volume.

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 4 - Hadoop Integration

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

Execute the following instructions.

1. Clean up your Python session by hitting the 'clear' button and the 'expand' link (if available)

2. Select the 2_pyspark.py file
3. If you don't have a Python 3 workbench session open yet, Launch a Python 3 session 1 vCPU / 4 GB RAM
4. Run line 32:
5. `!hdfs dfs -put -f $HOME/data/kmeans_data.txt /user/$HADOOP_USER_NAME`
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!)

```
Welcome to Cloudera Data Science Workbench

Kernel: python2

Project workspace: /home/cdsw

No Kerberos principal or Hadoop username detected

Runtimes:
  R: R version 3.4.1 (--) -- "Single Candle"
  Python 2: Python 2.7.11
  Python 3: Python 3.6.1
  Java: java version "1.8.0_121"

Git origin: https://github.com/andremolenaar/CDSW-Demo-Short

cdsw@6mzzmuu4vvgsvbjy:~$ hdfs dfs -ls
Found 5 items
drwx------   - cdsw cdsw          0 2018-10-05 13:00 .Trash
drwxr-xr-x   - cdsw cdsw          0 2018-10-12 11:23 .sparkStaging
-rw-r--r--   3 cdsw cdsw         71 2018-10-12 11:22 kmeans_data.txt
drwxr-xr-x   - cdsw cdsw          0 2018-10-12 11:18 models
drwxr-xr-x   - cdsw cdsw          0 2018-09-04 13:52 output
cdsw@6mzzmuu4vvgsvbjy:~$
```

or

```
> !hdfs dfs -ls

Found 5 items
drwx------   - cdsw cdsw          0 2018-10-05 13:00 .Trash
drwxr-xr-x   - cdsw cdsw          0 2018-10-12 11:23 .sparkStaging
-rw-r--r--   3 cdsw cdsw         71 2018-10-12 11:22 kmeans_data.txt
drwxr-xr-x   - cdsw cdsw          0 2018-10-12 11:18 models
drwxr-xr-x   - cdsw cdsw          0 2018-09-04 13:52 output

>
```

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

Lab 5 - Pushing the Boundaries

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 customer 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:

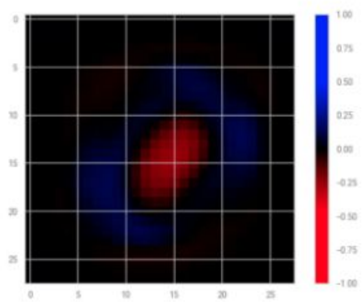
1. Select 3_tensorflow.py
2. Lines 2 through 6 show the imports of the various libraries (we installed them in Step 3)
3. Run 3_tensorflow.py in your current Python session (you might like to 'clear' your output screen first)- the input handwritten number images are shown, along with some images of feature maps for particular numbers

Examine layers

A red/black/blue colormap

```
> cdict = {'red':   [(0.0,  1.0, 1.0),
                    (0.25,  1.0, 1.0),
                    (0.5,  0.0, 0.0),
                    (1.0,  0.0, 0.0)],
         'green': [(0.0,  0.0, 0.0),
                    (1.0,  0.0, 0.0)],
         'blue':  [(0.0,  0.0, 0.0),
                      (0.5,  0.0, 0.0),
                      (0.75, 1.0, 1.0),
                      (1.0,  1.0, 1.0)]}
> redblue = matplotlib.colors.LinearSegmentedColormap('red_black_blue',cdict,256)
> wts = W.eval(sess)
> for i in range(0,5):
      im = wts.flatten()[i::10].reshape((28,-1))
      plt.imshow(im, cmap = redblue, clim=(-1.0, 1.0))
      plt.colorbar()
      print("Digit %d" % i)
      plt.show()
```

```
Digit 0
```



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.

STOP all your Python sessions now (you should only have one but sometimes people get carried away). This will help ensure there are plenty of resources for you and the others in the workshop.

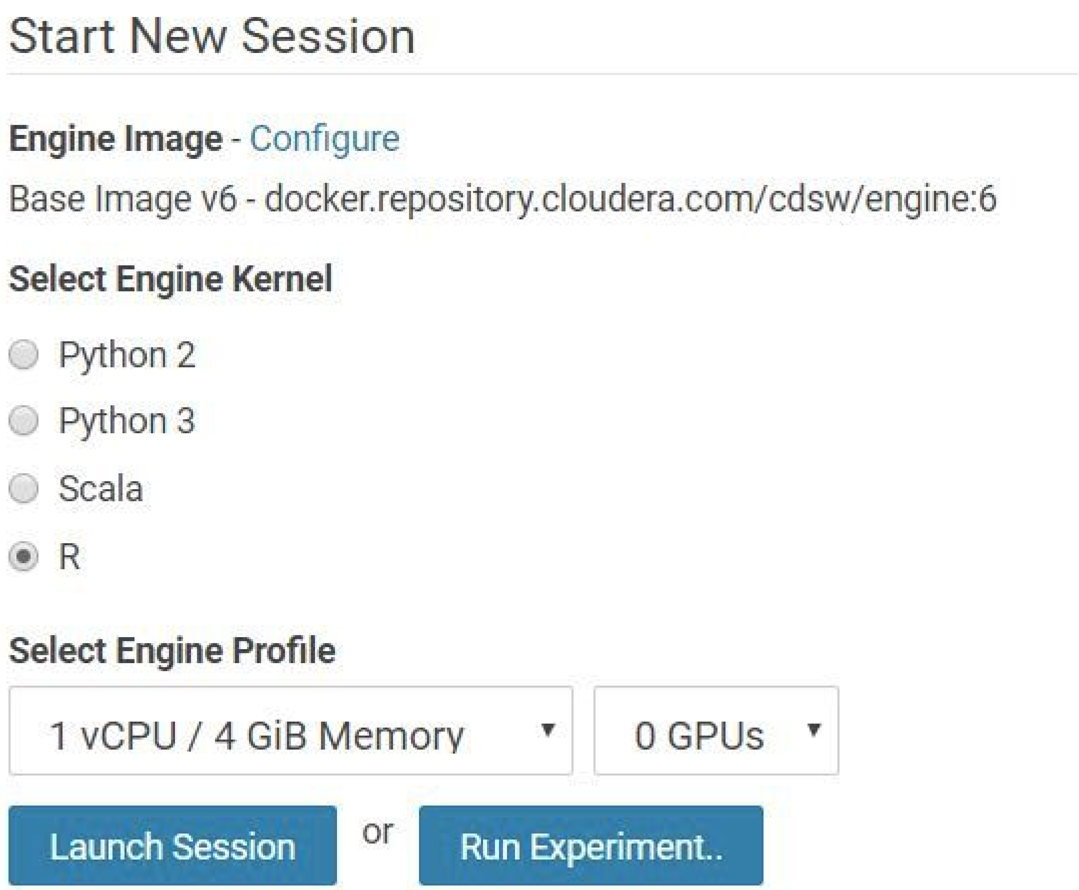
Lab 6 - SparklyR

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.

This lab requires that R is setup correctly. We provided instructions earlier, but if you skipped them then do this:

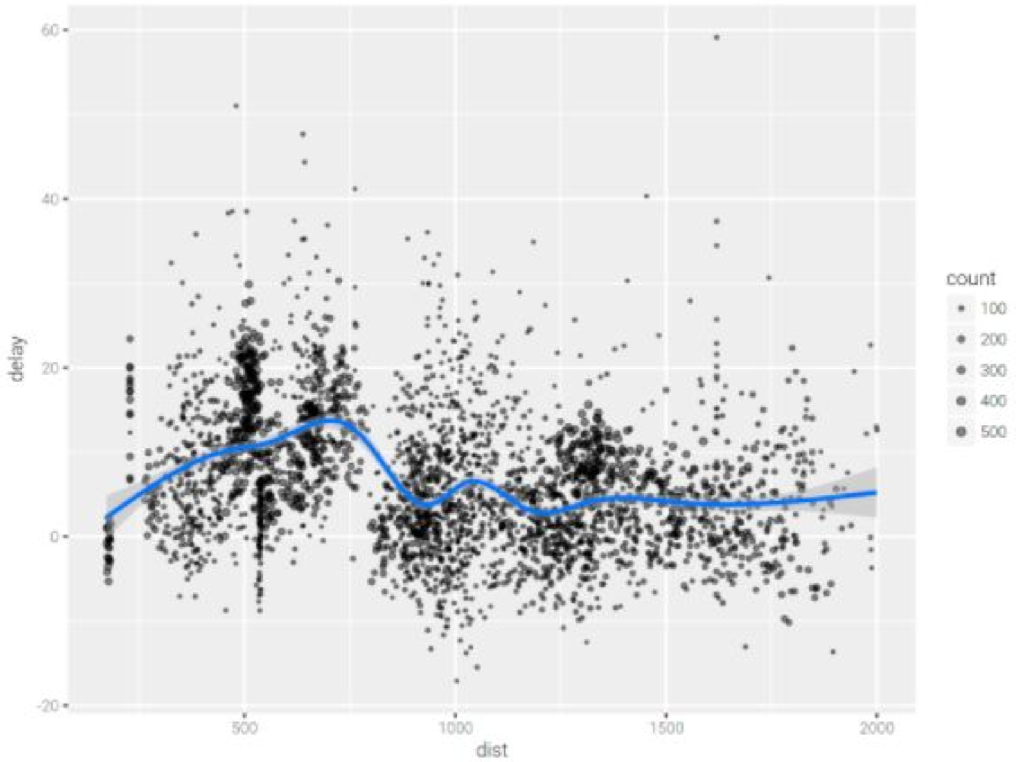
Ensure that you've started up an R session (2vCPU, 4GiB engine) and executed lines 37 through 42 from the README.md file. You'll only have to do this once for this project so if you've already done it don't do it again.

1. Create a new R Session.



2. Select (and run) 4_sparklyr.R
   The right hand side output window should (eventually) look like this (more or less - depending on your screen real-estate):

```
`geom_smooth()` using method = 'gam'
```



Machine Learning

You can orchestrate machine learning algorithms in a Spark cluster via the machine learning functions within sparklyr. connect to a set of high-level APIs built on top of DataFrames that help you create and tune machine learning workflow

In this example we'll use ml_linear_regression to fit a linear regression model. We'll use the built-in mtcars dataset, and predict a car's fuel consumption (mpg) based on its weight (wt) and the number of cylinders the engine contains (cyl). each case that the relationship between mpg and each of our features is linear.

copy mtcars into spark

```
> mtcars_tbl <- copy_to(sc, mtcars, overwrite = TRUE)
```

transform our data set, and then partition into 'training', 'test'

```
> partitions <- mtcars_tbl %>%
>
```

Have a read through the Machine Learning section

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!

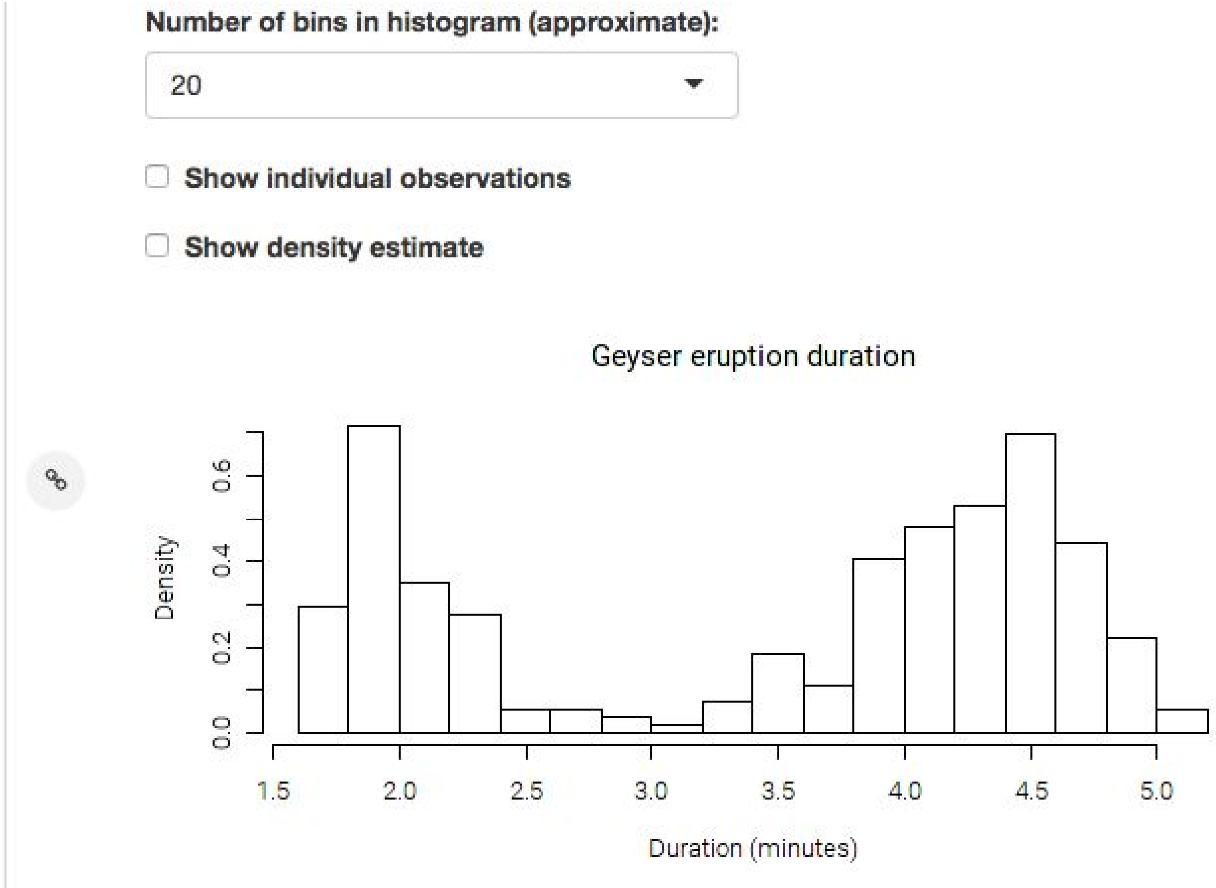
Lab 7 - Shiny

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!

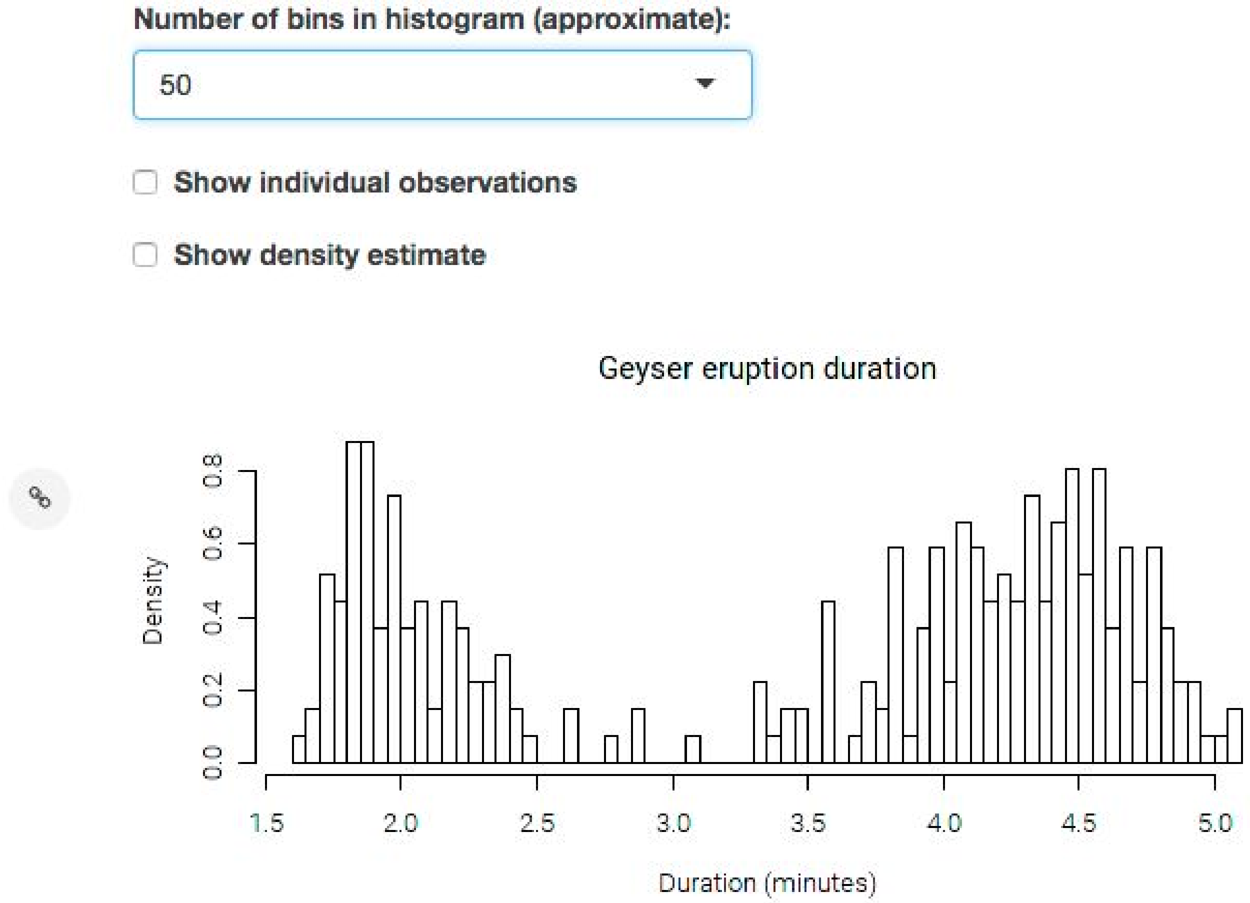
1. clear the R session screen and then run the 5_shiny.R application

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!



2. You can also change the number of bins in histogram to 50



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!

When you've finished here then stop the R session just to free up some resources.

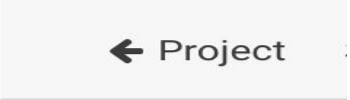
Have you Stopped the R Session ?

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?

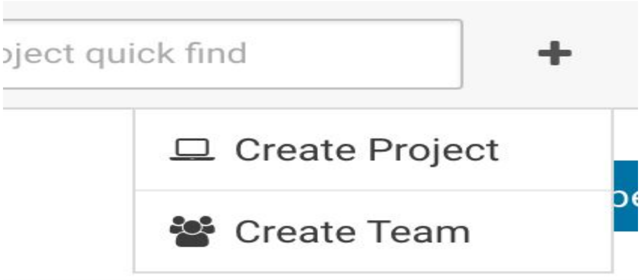
Lab 8 - Scala

In this lab we show how you can use the 'Template' mechanism to get started with a simple Scala example. Note that the built in templates and example code aren't written with multiple users in mind, so you might see file access and permission errors due to the fact that other students might've created or deleted files before you!:

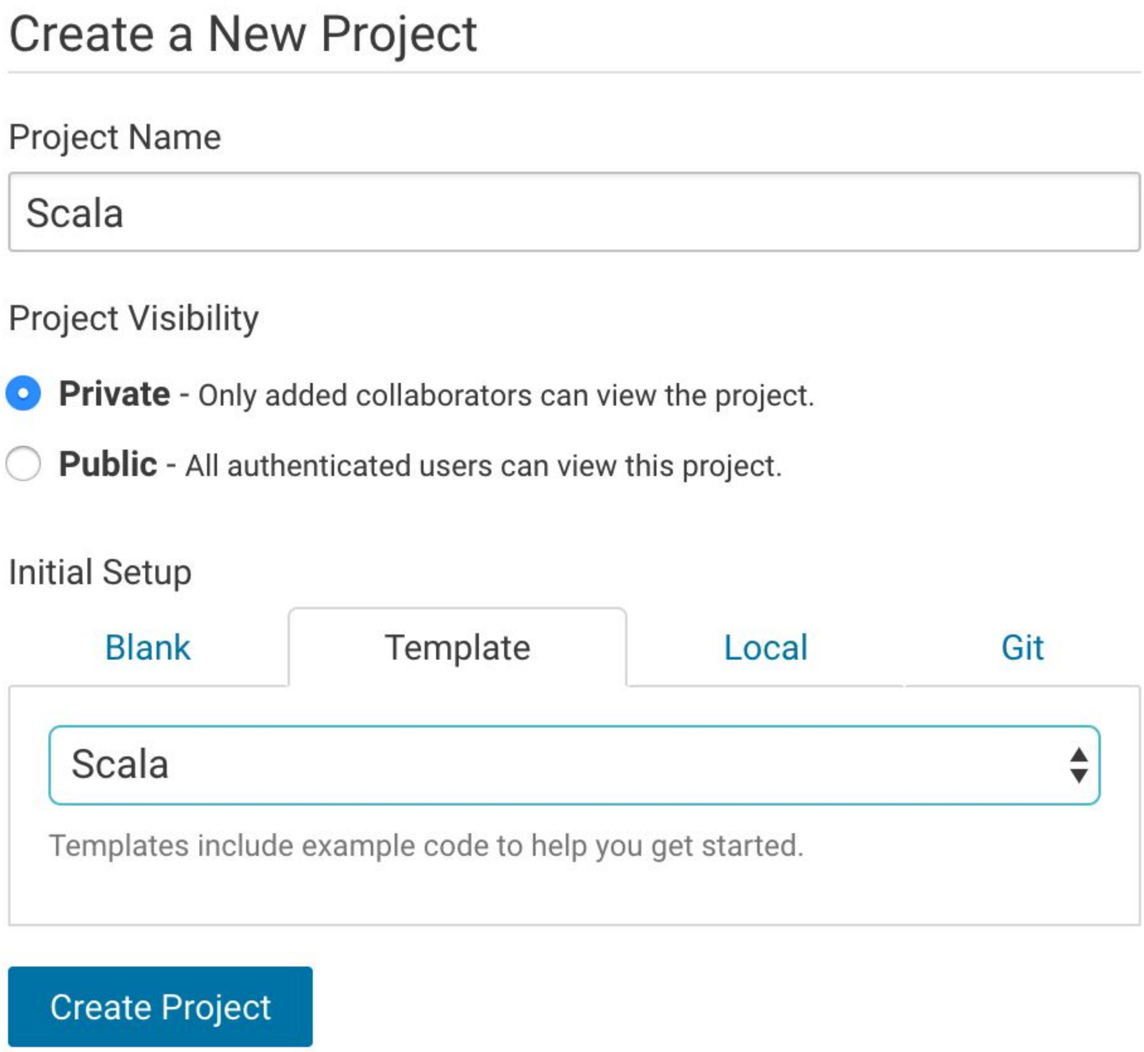
1. Navigate to the project space by selecting "project":



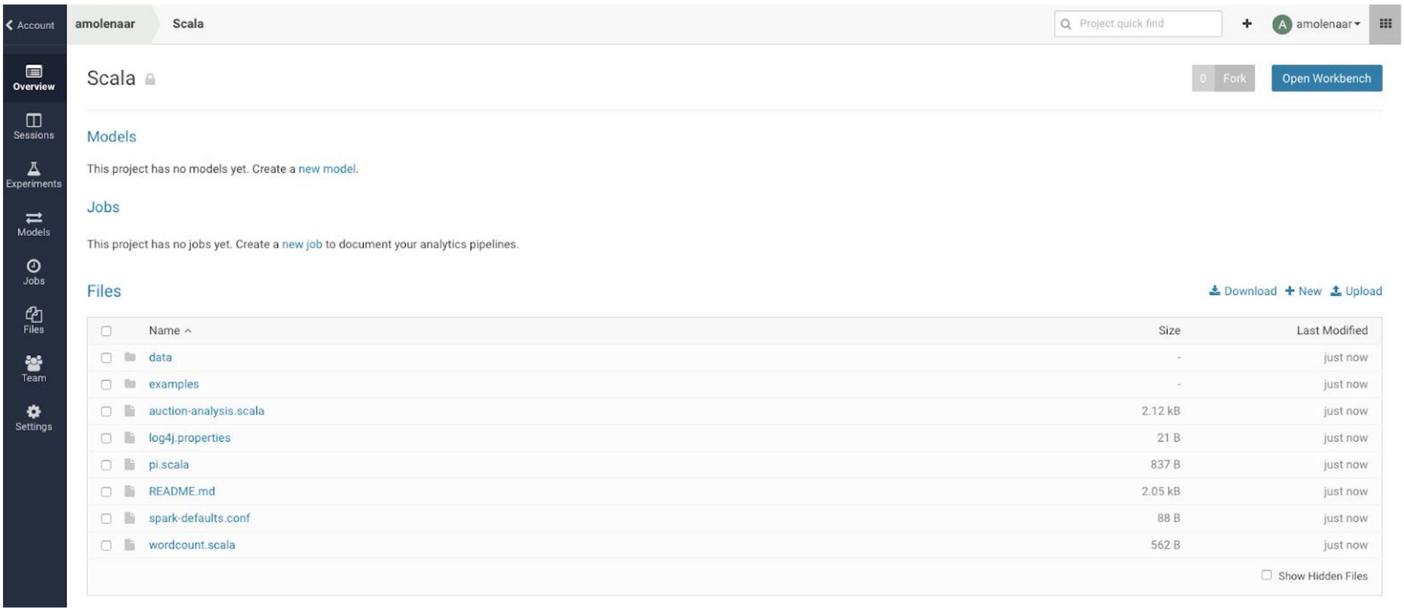
2. Create a new project by hitting the '+' button on the top right and selecting 'create project':



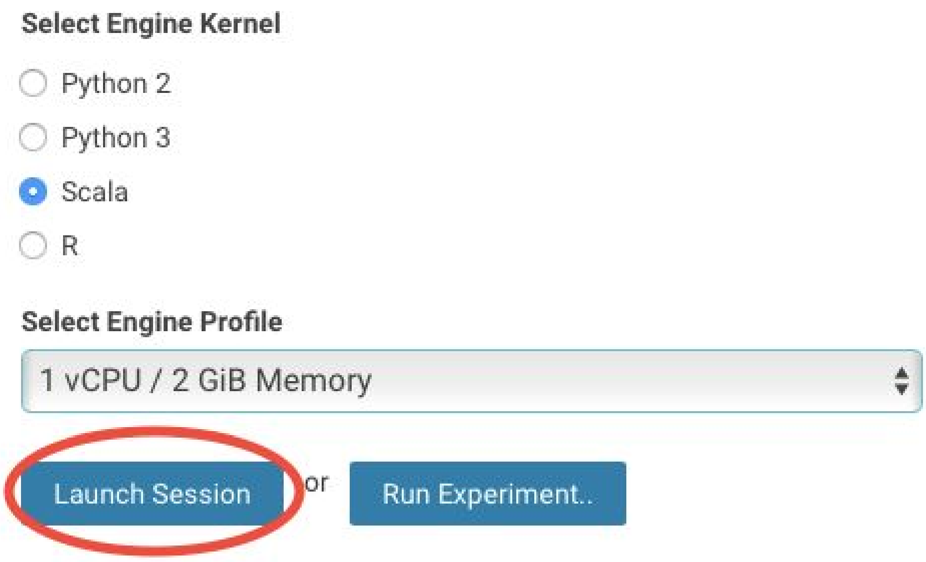
3. In the Create new Project window that comes up provide a name for your new project ('Scala', for example), and then choose the Scala template in the Initial Setup drop down menu:



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 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:
8. Open Terminal
9. `hdfs dfs -put -f data /user/$HADOOP_USER_NAME`

```
cdsw@44bi8vk24giecnf6:~$ hdfs dfs -put -f data /user/$HADOOP_USER_NAME
cdsw@44bi8vk24giecnf6:~$
```

10. Open example -> kmeans.scala, and edit the file as follows:

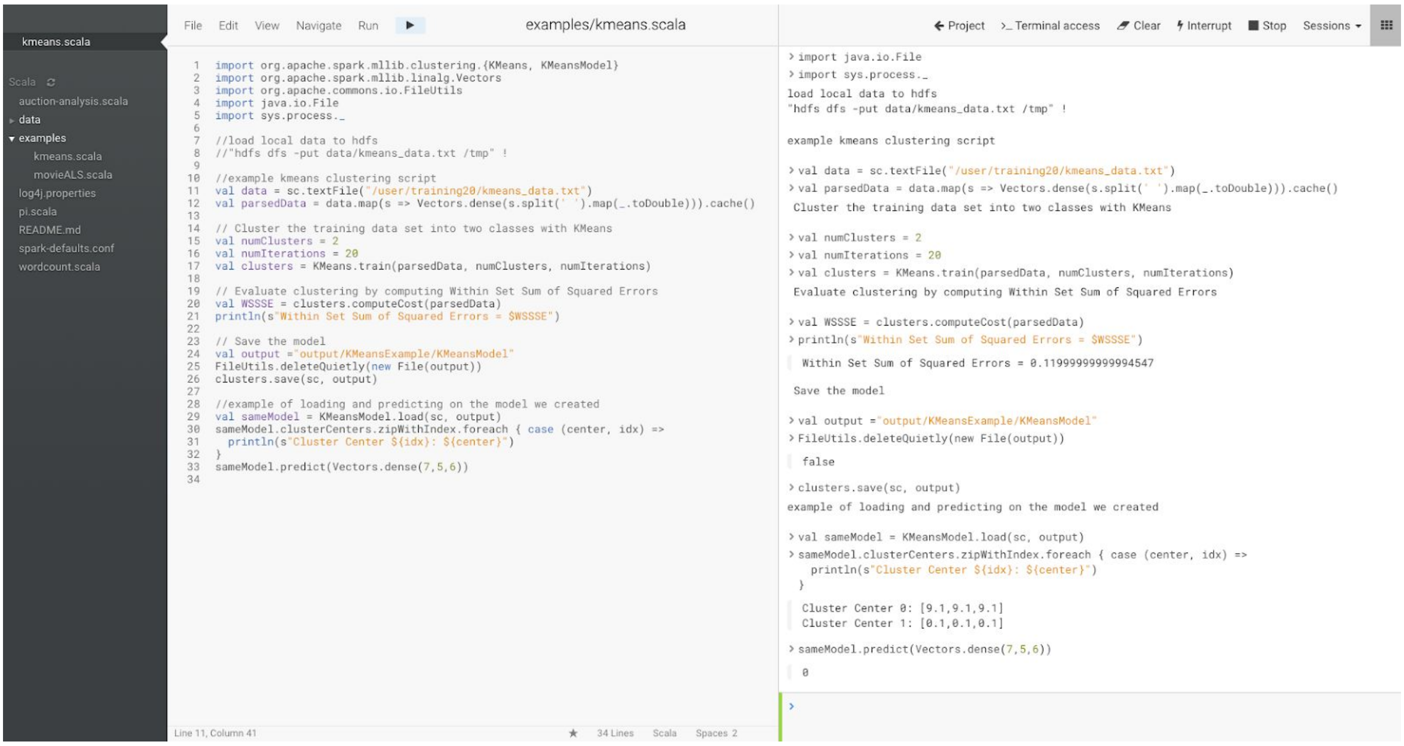
Remove line 8, by adding // in front. This will mark it as comments and not execute it.

```
7  //load local data to hdfs
8  //"hdfs dfs -put data/kmeans_data.txt /tmp" !
9
```

Modify line 11. The location of the input files is in your training directory. Make sure you refer to your training user number.

```
10  //example kmeans clustering script
11  val data = sc.textFile("/user/training20/kmeans_data.txt")
```

11. When the file is modified, run the program.



If you get a FileAlreadyExistsException executing ‘clusters.save(sc,output)’ then you can ignore the error and finish this lab by just executing the lines after that one:

```
> clusters.save(sc, output)
Name: org.apache.hadoop.mapred.FileAlreadyExistsException
Message: Output directory hdfs://ip-10-0-100-220.eu-west-1.compute.internal:8020/user/hdfs_super/output/KM
StackTrace:    at org.apache.hadoop.mapred.FileOutputFormat.checkOutputSpecs(FileOutputFormat.java:131)
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopDataset$1.apply$mcV$sp(PairRDDFunctions.sc
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopDataset$1.apply(PairRDDFunctions.scala:109
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopDataset$1.apply(PairRDDFunctions.scala:109
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:362)
  at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopDataset(PairRDDFunctions.scala:1096)
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$4.apply$mcV$sp(PairRDDFunctions.scala
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$4.apply(PairRDDFunctions.scala:1035)
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$4.apply(PairRDDFunctions.scala:1035)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:362)
  at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopFile(PairRDDFunctions.scala:1035)
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$1.apply$mcV$sp(PairRDDFunctions.scala
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$1.apply(PairRDDFunctions.scala:961)
  at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopFile$1.apply(PairRDDFunctions.scala:961)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:362)
  at org.apache.spark.rdd.PairRDDFunctions.saveAsHadoopFile(PairRDDFunctions.scala:960)
  at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply$mcV$sp(RDD.scala:1489)
  at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply(RDD.scala:1468)
  at org.apache.spark.rdd.RDD$$anonfun$saveAsTextFile$1.apply(RDD.scala:1468)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:362)
  at org.apache.spark.rdd.RDD.saveAsTextFile(RDD.scala:1468)
  at org.apache.spark.mllib.clustering.KMeansModel$SaveLoadV1_0$.save(KMeansModel.scala:128)
  at org.apache.spark.mllib.clustering.KMeansModel.save(KMeansModel.scala:94)

> val sameModel = KMeansModel.load(sc, output)
> sameModel.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
    println(s"Cluster Center ${idx}: ${center}")
  }

 Cluster Center 0: [0.1,0.1,0.1]
 Cluster Center 1: [9.1,9.1,9.1]

> sameModel.predict(Vectors.dense(7,5,6))

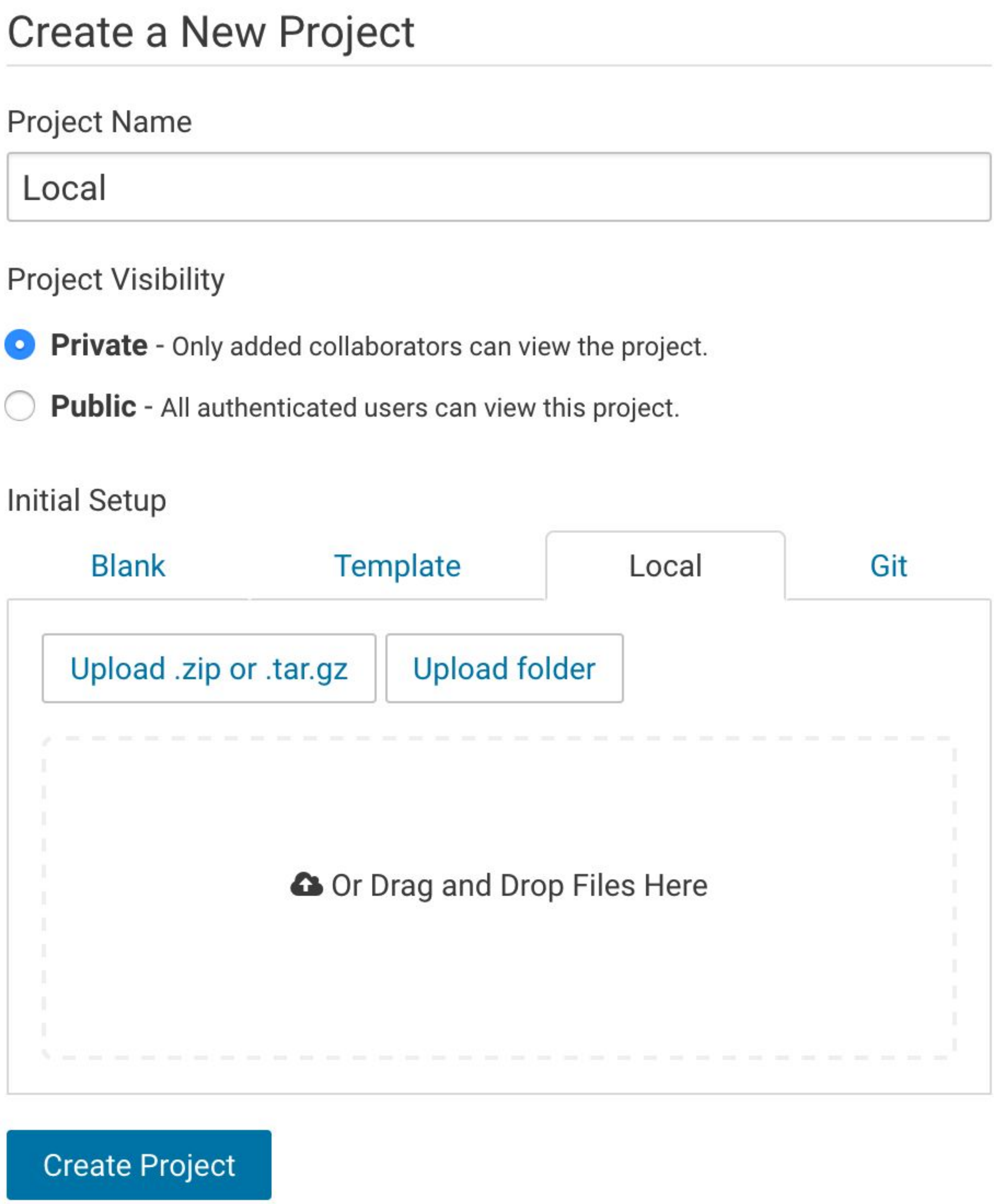
 1
```

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

Remember to stop your scala Session

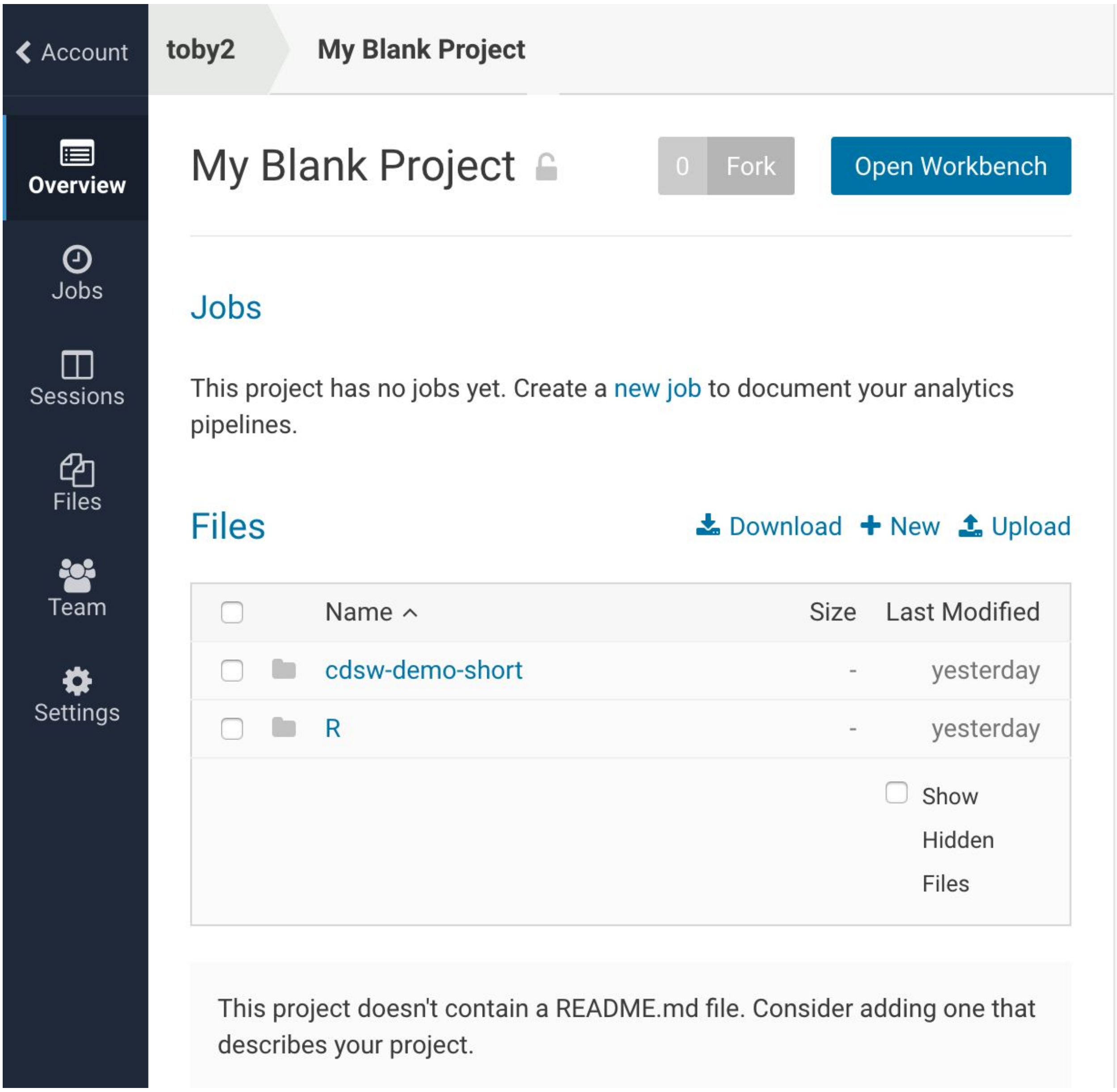
Lab 9 - Project Creation using Local Files

1. Create a new project, naming it 'Local', and select the 'Local tab':



2. Try adding a file or folder and then create your project
3. If you have some code you want to try then select that - otherwise just note that the project was created from the file(s) you selected.

4. You might've noticed in the File browser window that the ability to upload and download files is there for a project, no matter how you started the project:



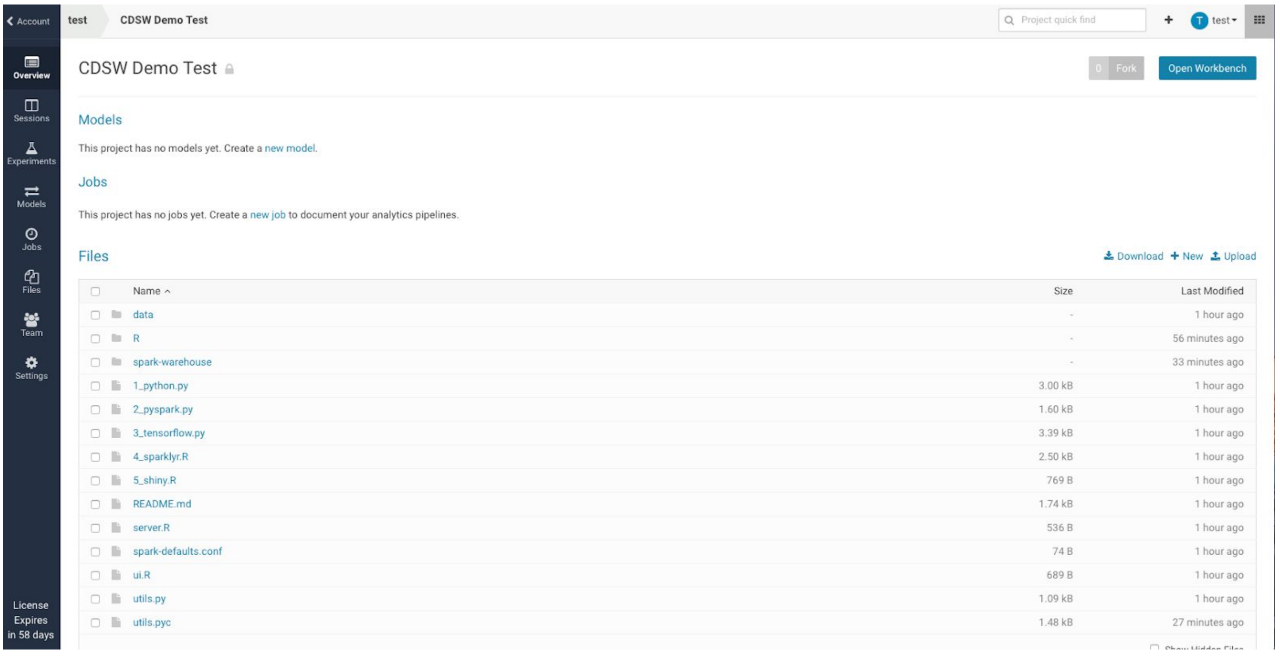
Question: With the combination of git, blank projects and uploading from a local file system on your laptop do you feel pretty confident you can get the data and code you want into the CDSW environment?

Lab 10 - Scheduling Jobs

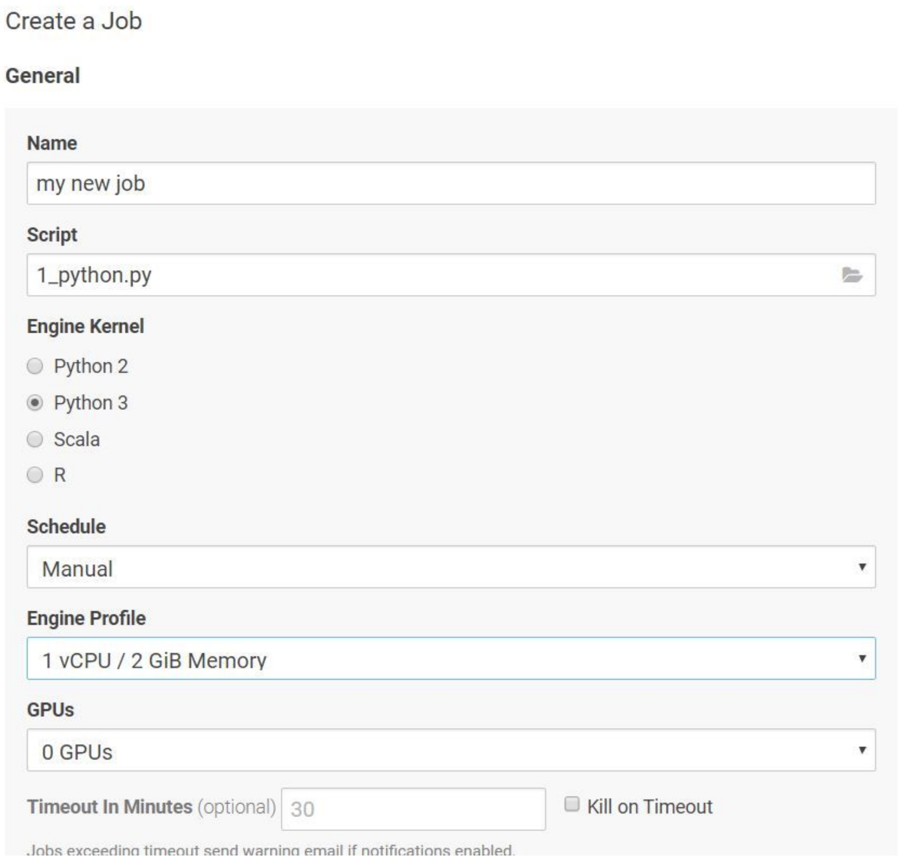
Its often the case that you need to execute tasks on a periodic basis, and to execute one or more tasks once some other task has succeeded. Obviously there are sophisticated workflow engines but for simple workflows CDSW has a jobs system built in.

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

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



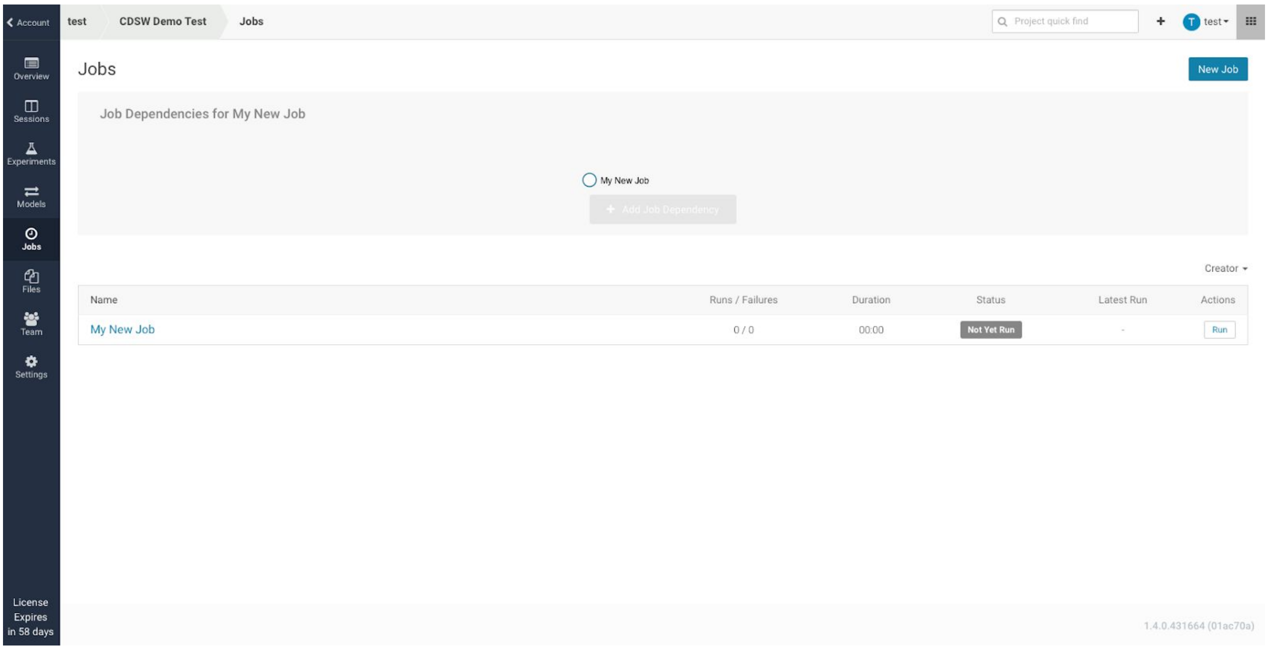
2. You need to be in a project to create a Job
3. Select the 'new job' link in the middle of the page, and you'll get to the following screen (there are other ways of getting to this next screen - its an exercise for the student to figure out what they might be):



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

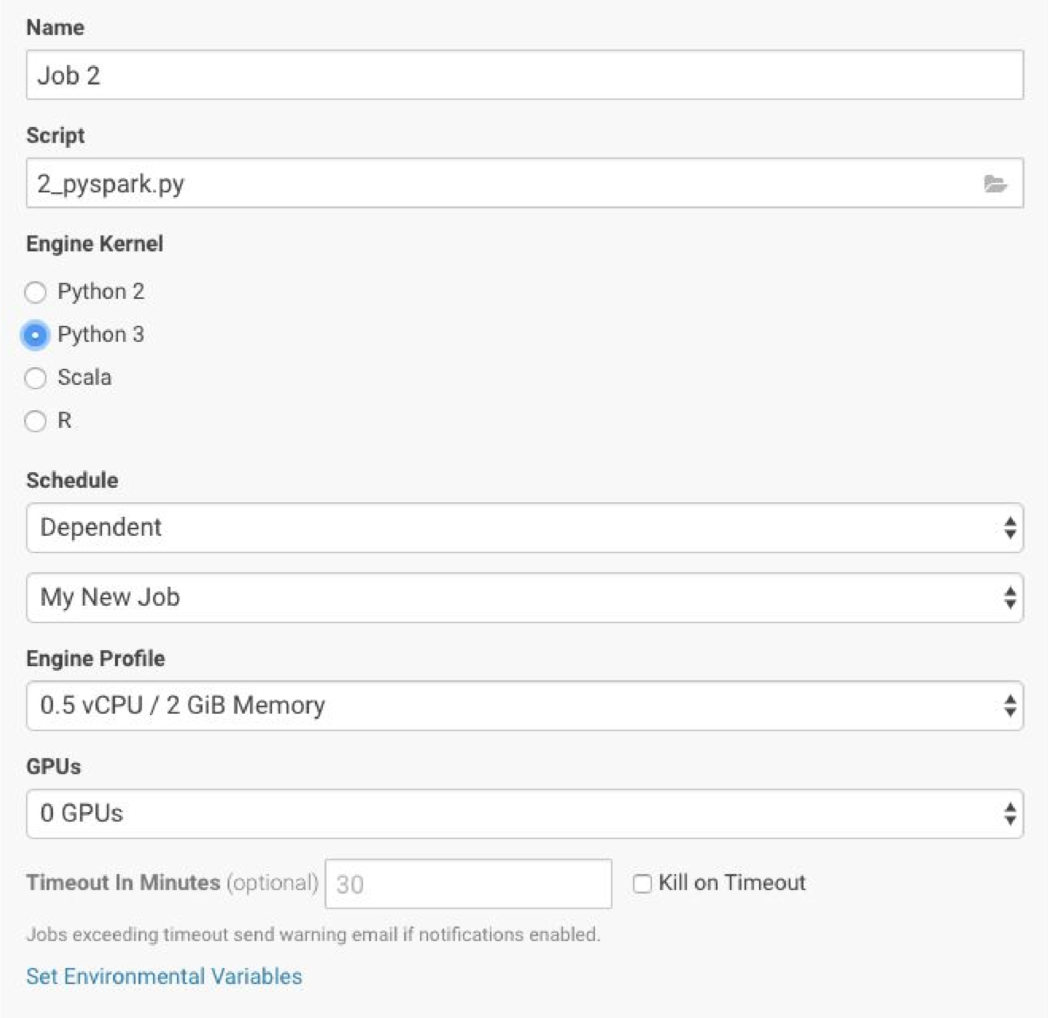
Name	My New Job
Script	1_python.py
Engine Kernel	Python 3

5. Leave everything else as **default**. Scroll down and hit ‘Create Job’. You should get to this screen:

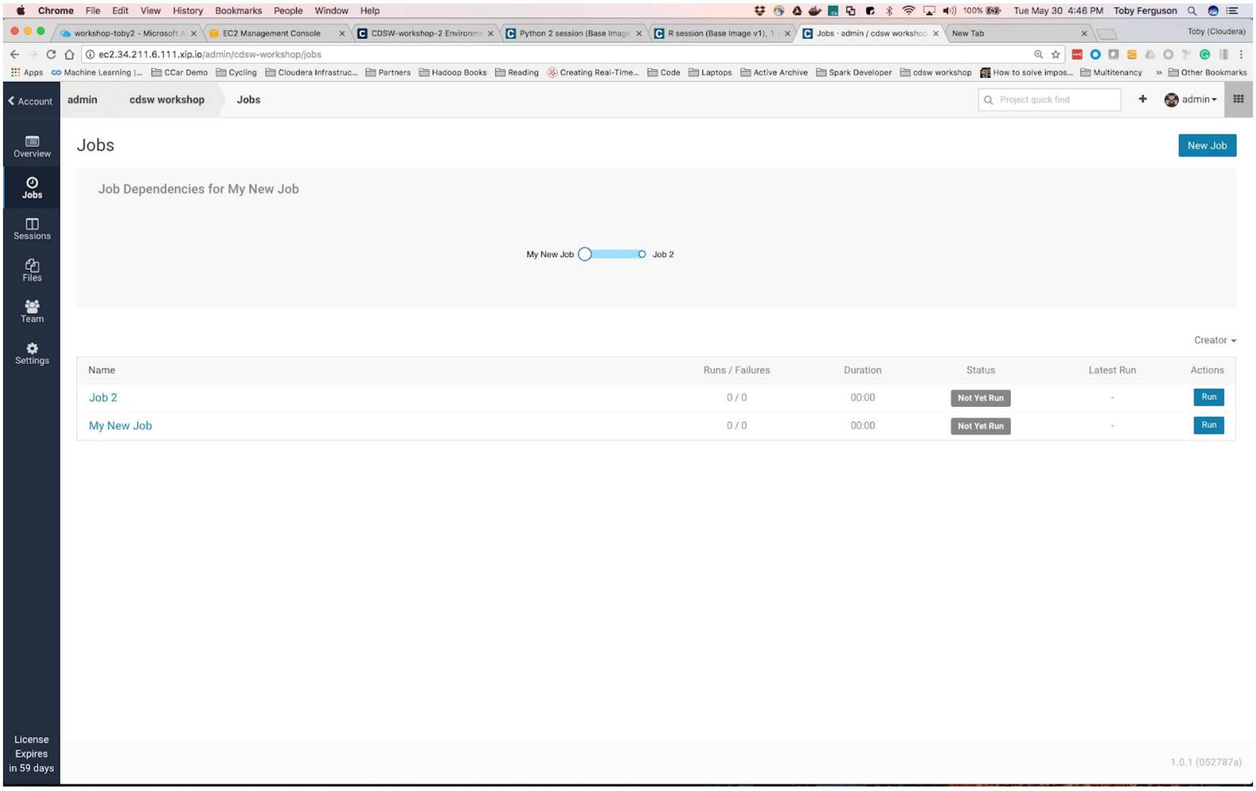


6. Here you can see that you have a job (‘My New Job’). It’s never been run, and it has no dependencies.
7. Let’s make other jobs depend on this one: Click the ‘+ Add Job Dependency’ grayed out button and add a new job that has a dependency on ‘My New Job’. The parameters are:

Name	Job 2
Script	2_pyspark.py
Engine Kernel	Python 3

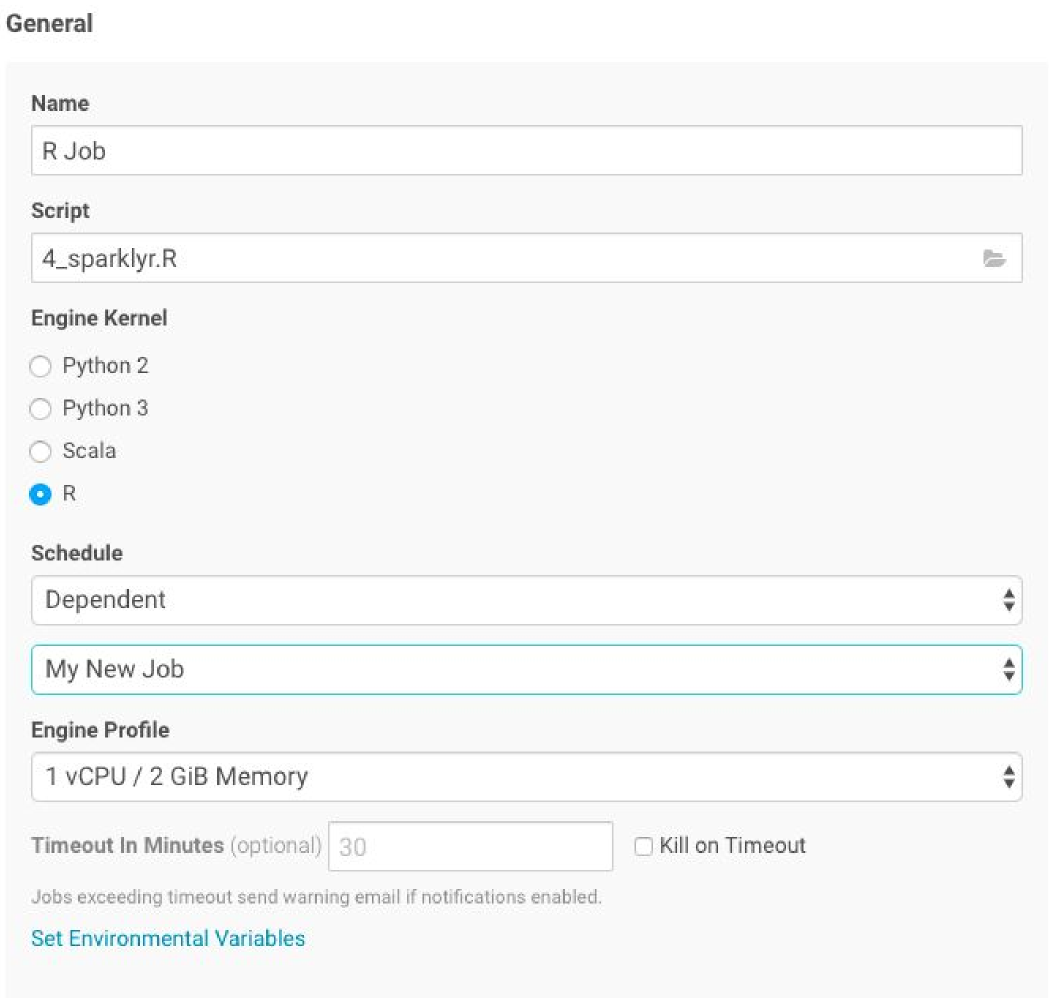


8. Scroll down and ‘Create Job’. You’ll now see a page like this:

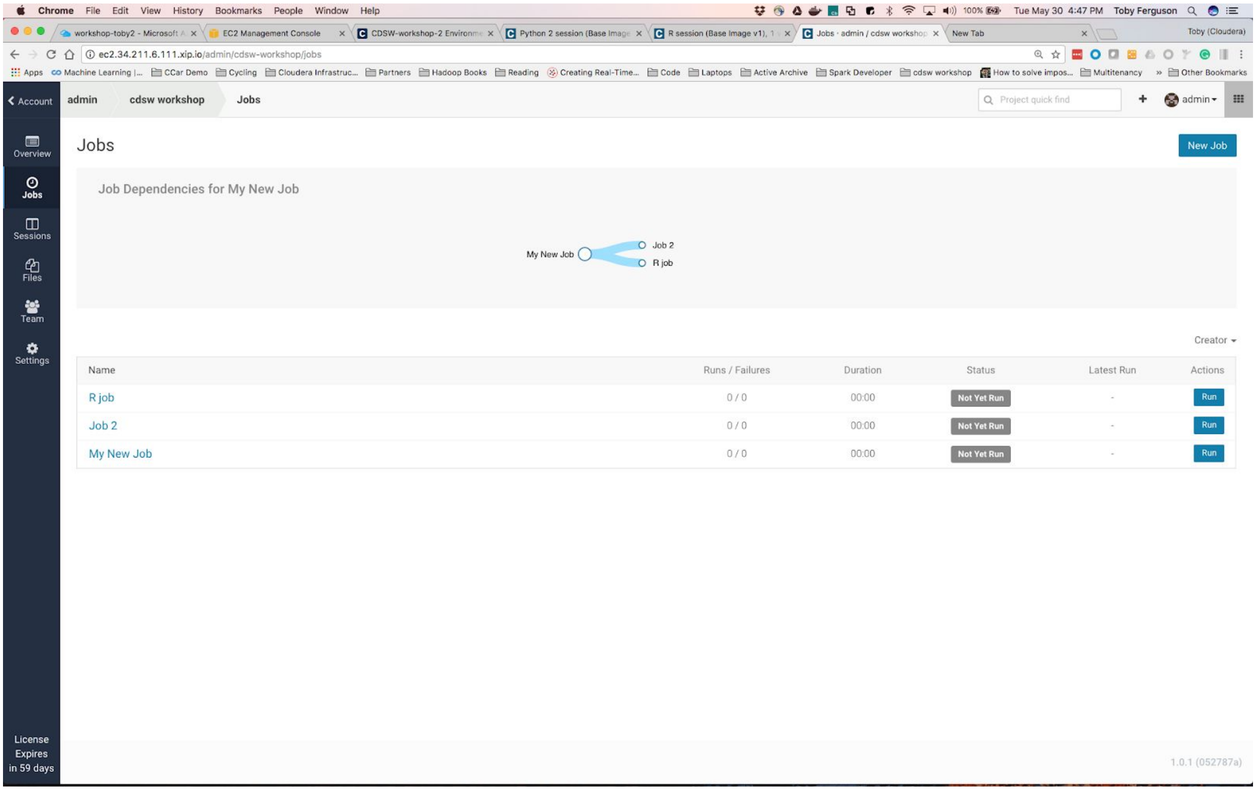


9. So here we can see that ‘Job 2’ depends upon ‘My New Job’ (although you can run each manually, if you so choose).
10. Lets add another job that will run in parallel with Job2:
11. Click ‘New Job’ in the top right corner and create another job that depends upon ‘My New Job’. The parameters you’ll need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

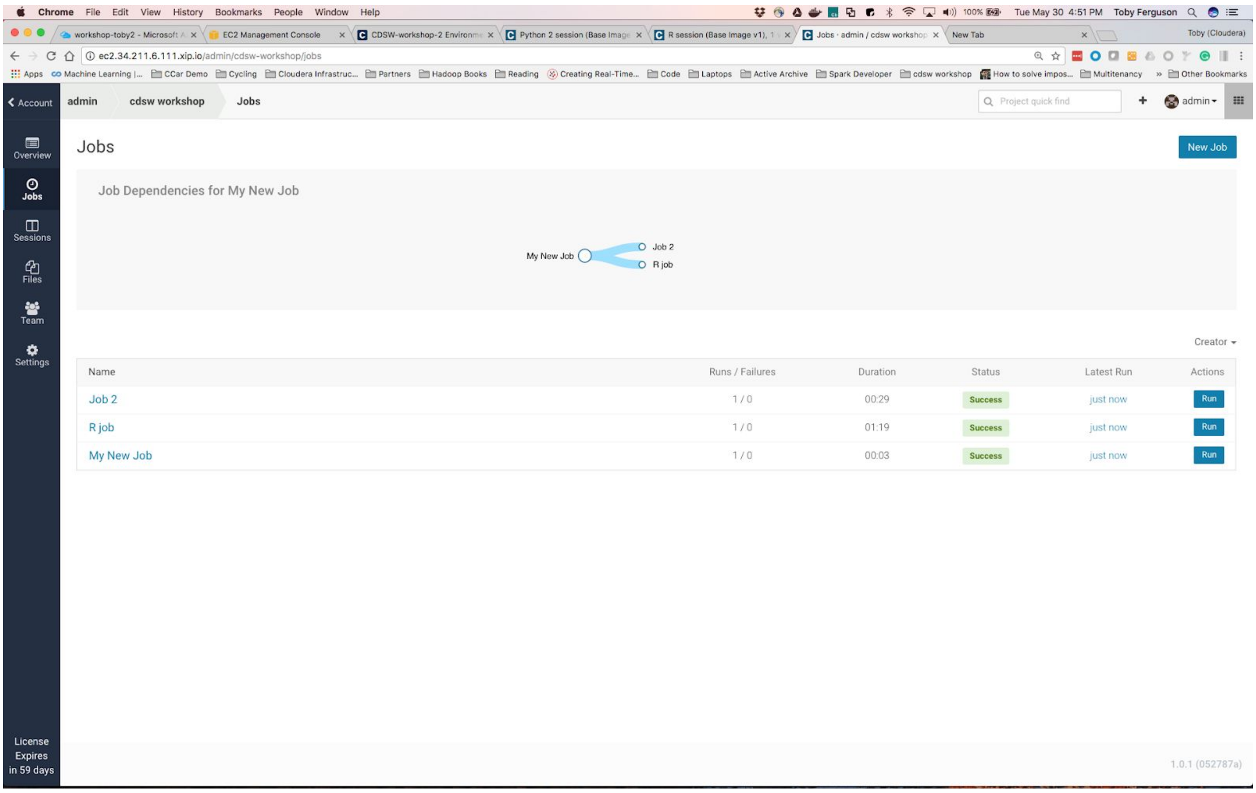
Name	R Job
Script	4_sparklyr.R
Engine Kernel	R
Schedule	Dependent / My New Job



12. Create the job and you’ll see this:



13. 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:



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?

Lab 11 – Experiments

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to run batch experiments that track different versions of code, input parameters, and output (both metrics and files).

Challenge

As data scientists iteratively develop models, they often experiment with datasets, features, libraries, algorithms, and parameters. Even small changes can significantly impact the resulting model. This means 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.

Without versioned experiments you would need intense process rigor to consistently track training artifacts (data, parameters, code, etc.), and even then it might be impossible to reproduce and explain a given result. This can lead to wasted time/effort during collaboration, not to mention the compliance risks introduced.

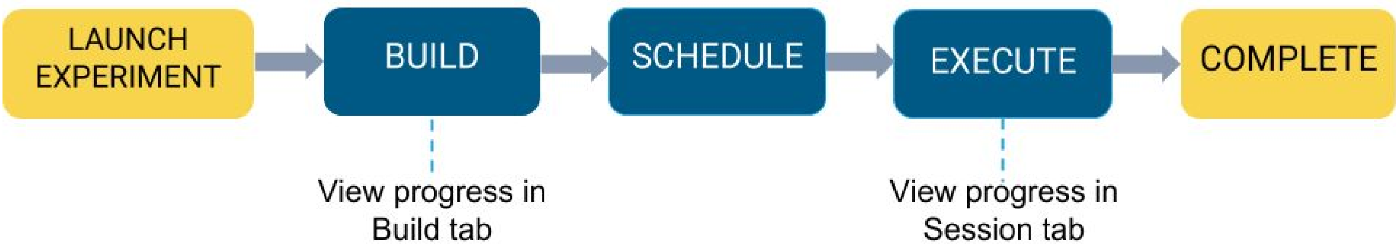
Solution

Starting with version 1.4, Cloudera Data Science Workbench uses experiments to facilitate ad-hoc batch execution and model training. Experiments are batch executed workloads where the code, input parameters, and output artifacts are versioned. This feature also provides a lightweight ability to track output data, including files, metrics, and metadata for comparison.

Concepts

The term experiment refers to a non interactive batch execution script that is versioned across input parameters, project files, and output. Batch experiments are associated with a specific project (much like sessions or jobs) and have no notion of scheduling; they run at creation time. To support versioning of the project files and retain run-level artifacts and metadata, each experiment is executed in an isolated container.

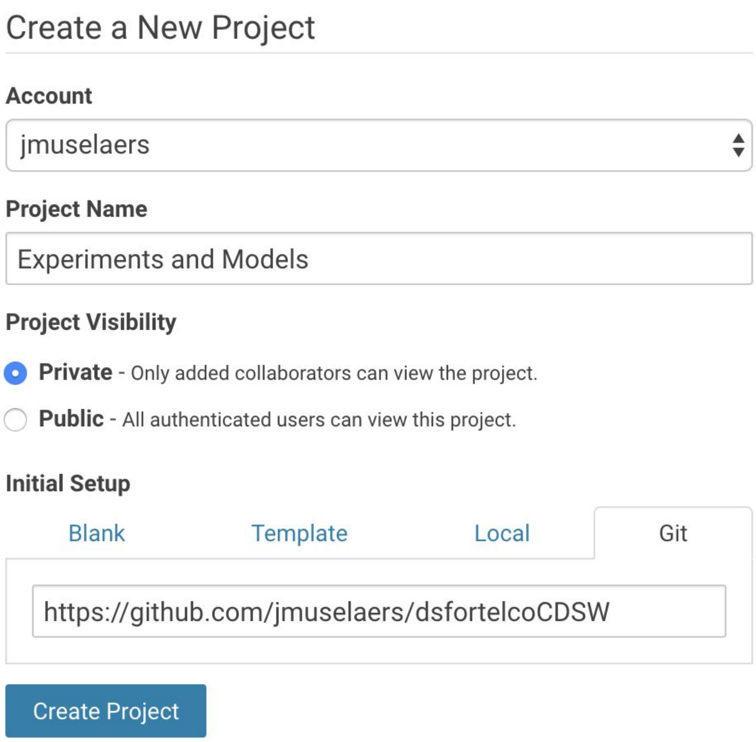
Lifecycle of an Experiment



Step 1: Create a new project

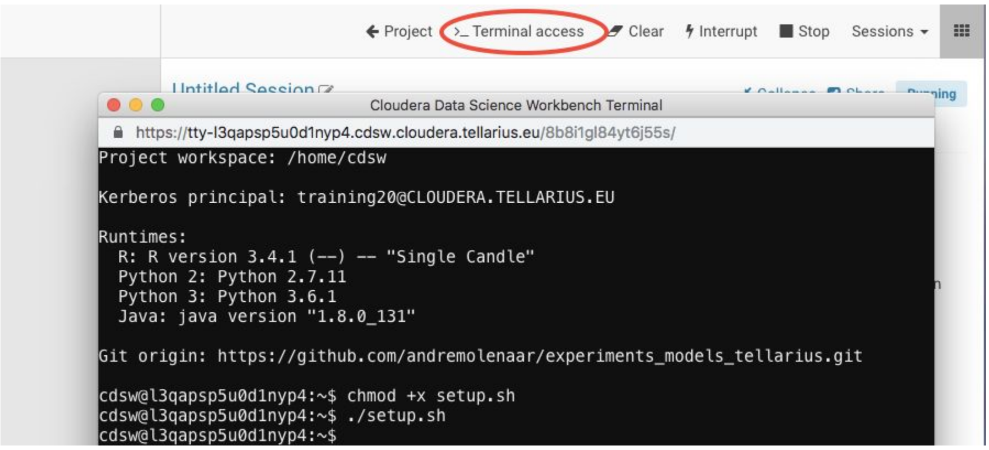
Go to the homepage of your Data Science workbench, and create a ‘New’ project.

Call the new repository something like Experiments and Models.
Create the repository as a clone of the github repository:
https://github.com/jmuselaers/dsfortelcoCDSW



Start a workbench with a Python 3 and 4 GB of memory.
When the workbench is available, open a terminal window and make the cdsw-build.sh program executable. Use the following command to do that:

```
chmod +x setup.sh
./setup.sh
```



Step 2: Examin dsfortelco_sklearn_exp.py

Open the file “dsfortelco_sklearn_exp.py”. This is a python program that builds a churn model to predict customer churn (the likelyhood that this customer is going to stop his subscription with his telecom operator). There is a dataset available on hdfs (/tmp/churn_all.csv), with customer data, including a churn indicator field.

The program is going to build a churn prediction model using the Random Forest algorithm. Random forests are ensembles of decision trees. Random forests are one of the most successful machine learning models for classification and regression. They combine many decision trees in order to reduce the risk of overfitting. Like decision trees, random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

spark.mllib supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features. spark.mllib implements random forests using the existing decision tree implementation. Please see the decision tree guide for more information on trees.

The Random Forest algorithm expects a couple of parameters:

- numTrees: Number of trees in the forest.
  Increasing the number of trees will decrease the variance in predictions, improving the model’s test-time accuracy.
  Training time increases roughly linearly in the number of trees.
- maxDepth: Maximum depth of each tree in the forest.
  Increasing the depth makes the model more expressive and powerful. However, deep trees take longer to train and are also more prone to overfitting.
  In general, it is acceptable to train deeper trees when using random forests than when using a single decision tree. One tree is more likely to overfit than a random forest (because of the variance reduction from averaging multiple trees in the forest).

In the dsfortelco_pyspark_exp.py program, these parameters can be passed to the program at runtime. In the lines 38 and 39, these parameters are passed to python variables:

```
param_numTrees=int(sys.argv[1])
param_maxDepth=int(sys.argv[2])
```

Also note that at the lines 69 and 70, the quality indicator for the Random Forest model, are written back to the Data Science Workbench repository:

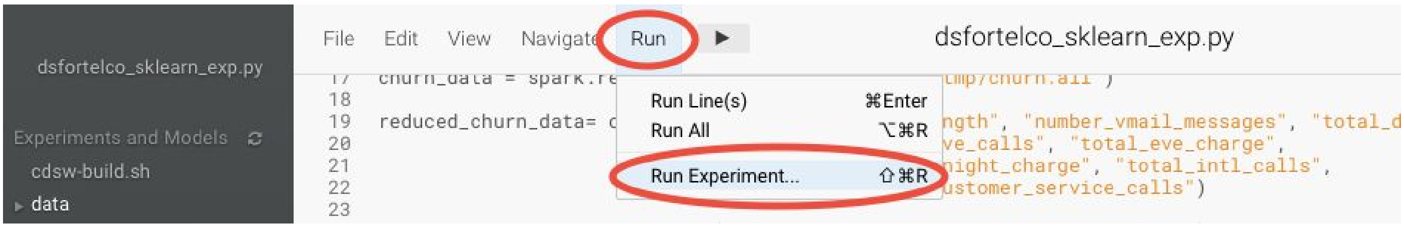
```
cdsw.track_metric("auroc", auroc)
cdsw.track_metric("ap", ap)
```

These indicators will show up later in the Experiments dashboard.

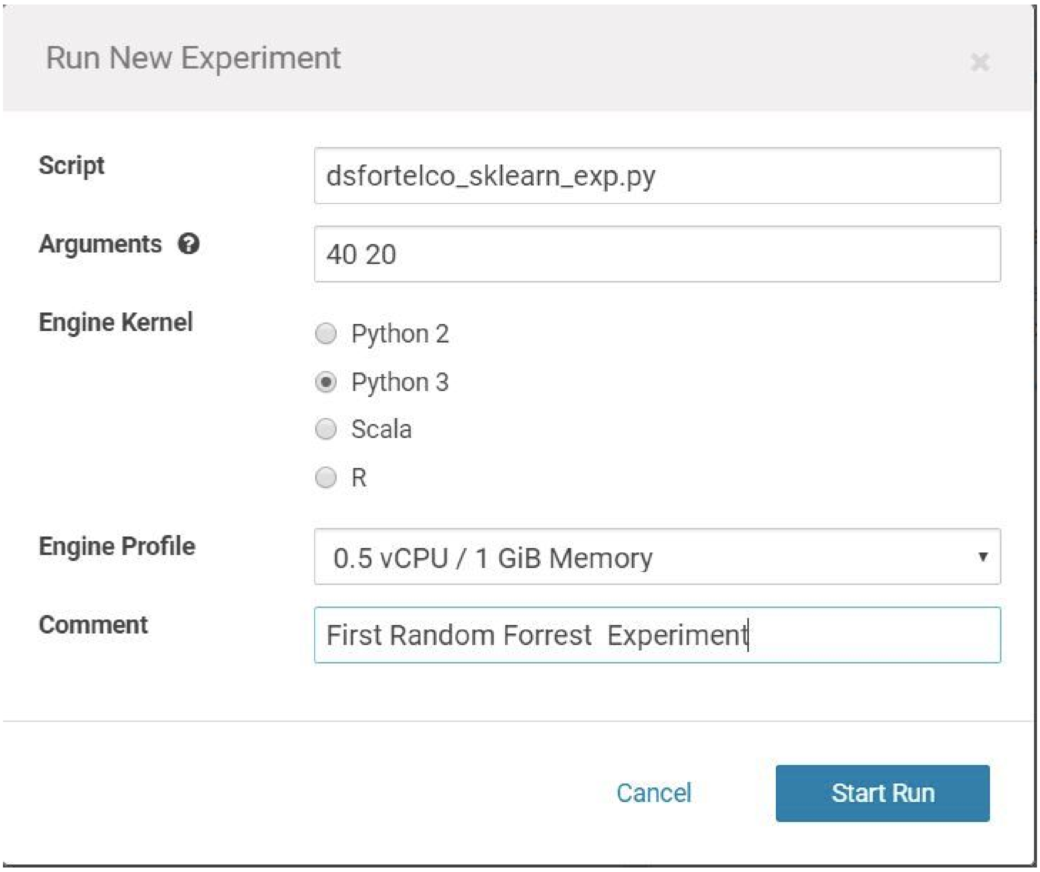
Step 3: Run the experiment for the first time

Now, run the experiment using the following parameters:
numTrees = 40
numDepth = 20

From the menu, select Run -> Experiments.



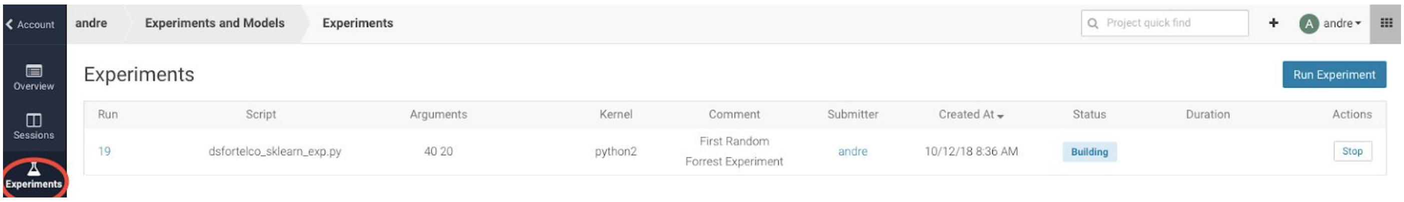
Specify the arguments for this run, by typing the numbers behind the arguments field. Note that these fields are separated by a space and that there is no comma (,)



Now, in the background, the Data Science Workbench environment will spin up a new docker container, where this program will run.

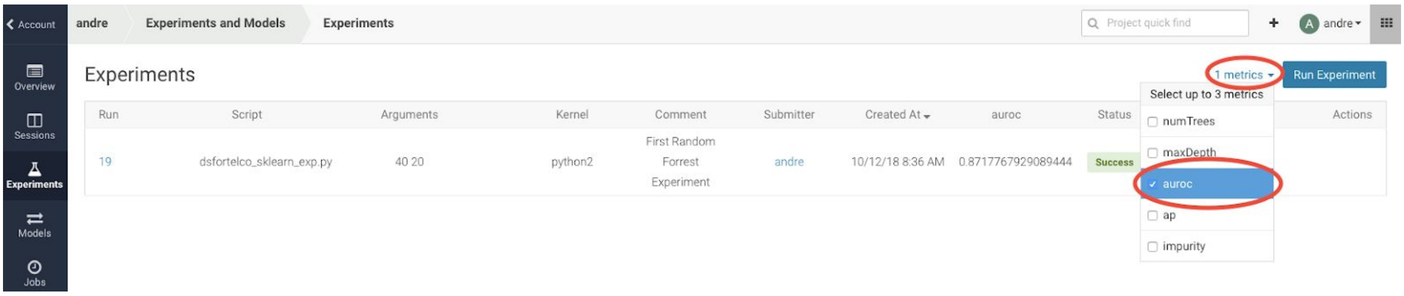
Step 4: Check the results for the first experiment

Go back to the ‘Projects’ page in CDSW, and hit the ‘Experiments’ button.

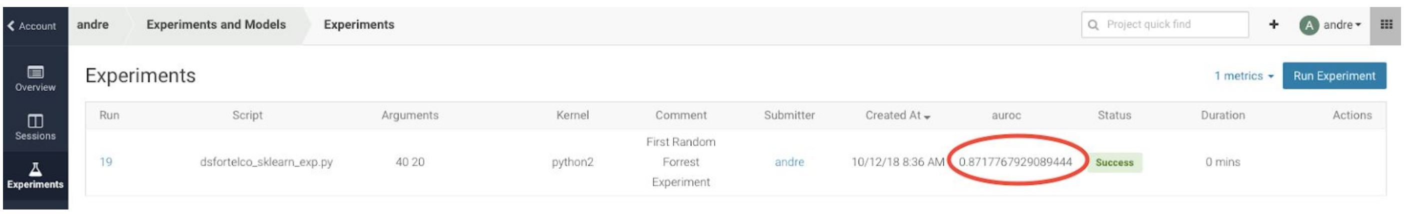


If the Status indicates ‘Running’, you have to wait till the run is completed.
In case the status is ‘Build Failed’ or ‘Failed’, check the log information. This is accessible by clicking on the run number of your experiments. There you can find the session log, as well as the build information.

In case your status indicates ‘Success’, you should be able to see the auroc (Area Under the Curve) model quality indicator. It might be that this value is hidden by the CDSW user interface. in that case, click on the ‘3 metrics’ links, and select the auroc field. It might be needed to de-select some other fields, since the interface can only show 3 metrics at the same time.



When the auroc metric is selected, you will be able to see the value.



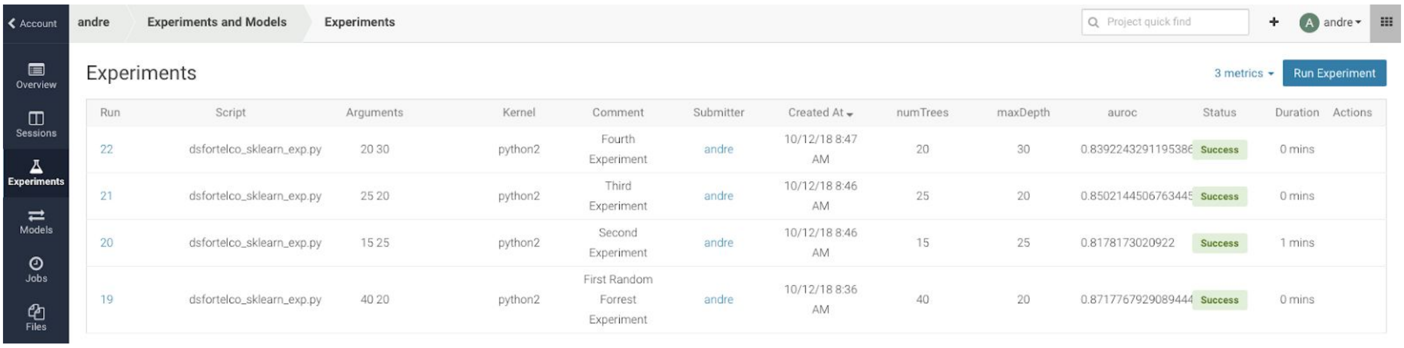
In this example, 0.871
Not bad, but maybe there are better hyper parameter values available.

Step 5: Re run the experiment several times

Now, re-run the experiment 3 more times and try different values for NumTrees and NumDepth.
Try the following values:

NumTrees	NumDepth
15	25
25	20
Try something yourself	Try something yourself

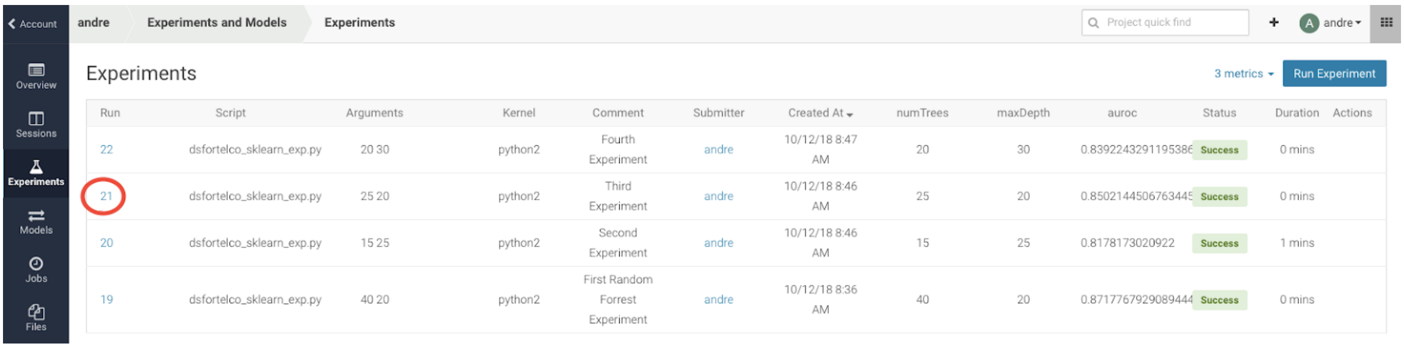
When all runs have completed successfully, check which parameters had the best quality (best predictive value). This is represented by the highest 'area under the curve', auroc metric.



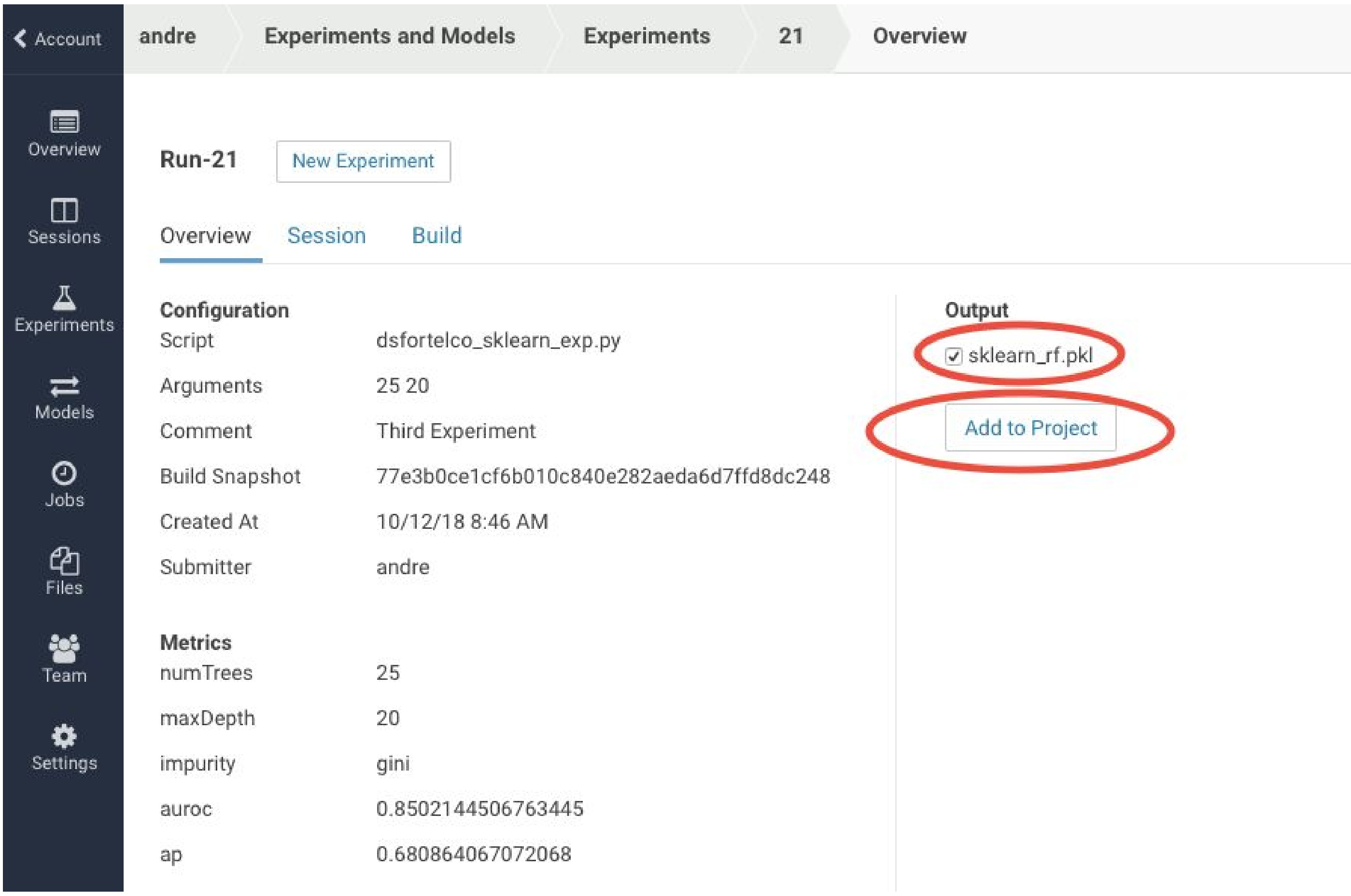
In this example, run 21 had the highest auroc value, so that is the model that you would want to use for your business.

Step 6: Save the best model to your environment

Select the run number with the best predictive value, in this example, run number 21.



In the Overview screen of the experiment, you can see that the model in spark format, is captured in the file 'sklearn_rf.pkl'. Select this file and hit the 'Add to Project' button. This will copy the model to your project directory.



Lab 12 – Working with Models

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to build, deploy, and manage models as REST APIs to serve predictions.

Challenge

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. In most organizations, the model deployment process will require assistance from a separate DevOps team who likely have their own policies about deploying new code.

For example, a model that has been developed in Python by data scientists might be rebuilt in another language by the devops team before it is actually deployed. This process can be slow and error-prone. It can take months to deploy new models, if at all. This also introduces compliance risks when you take into account the fact that the new re-developed model might not be even be an accurate reproduction of the original model.

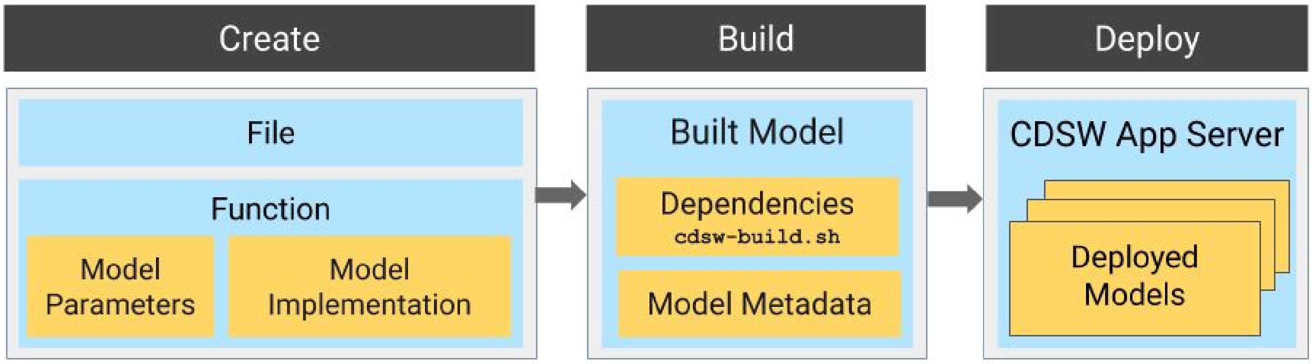
Once a model has been deployed, you then need to ensure that the devops team has a way to rollback the model to a previous version if needed. This means the data science team also needs a reliable way to retain history of the models they build and ensure that they can rebuild a specific version if needed. At any time, data scientists (or any other stakeholders) must have a way to accurately identify which version of a model is/was deployed.

Solution

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to build and deploy their own models as REST APIs. Data scientists can now select a Python or R function within a project file, and Cloudera Data Science Workbench will:

- Create a snapshot of model code, model parameters, and dependencies.
- Package a trained model into an immutable artifact and provide basic serving code.
- Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function's return type.
- Save the model along with some metadata.
- Deploy a specified number of model API replicas, automatically load balanced.

Stages of the Model Deployment Process



Step 1: Examine the program predic_churn_sklearn.py

Open the project you created in the previous lab, and examine the file.

```
import pickle
import numpy as np

model = pickle.load(open("models/sklearn_rf.pkl","rb"))

def predict(args):
  account=np.array(args["feature"].split(",")).reshape(1,-1)
  return {"result" : model.predict(account)[0]}

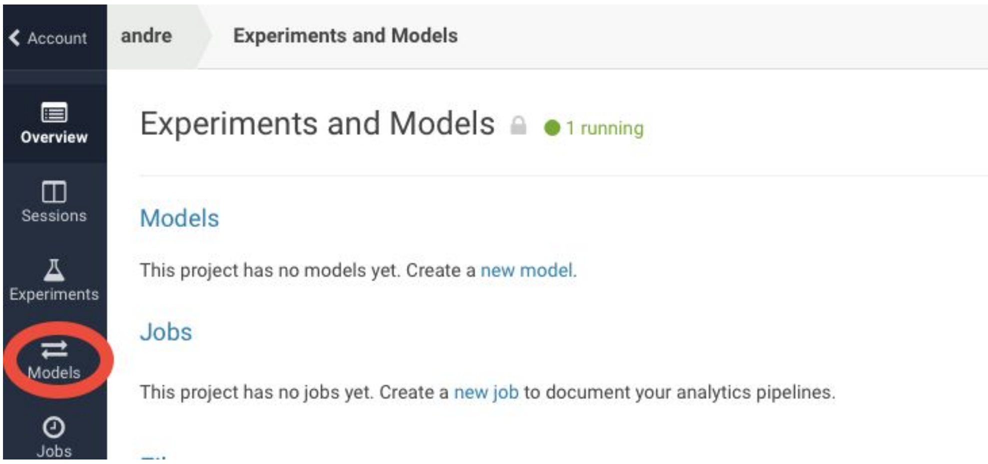

```

This PySpark program uses the pickle.load mechanism to deploy models.. The model it refers to the sklearn_rf.pkl file, was saved in the previous lab from the experiment with the best predictive model.

There is a predict definition which is the function that calls the model, using features, and will return a result variable.

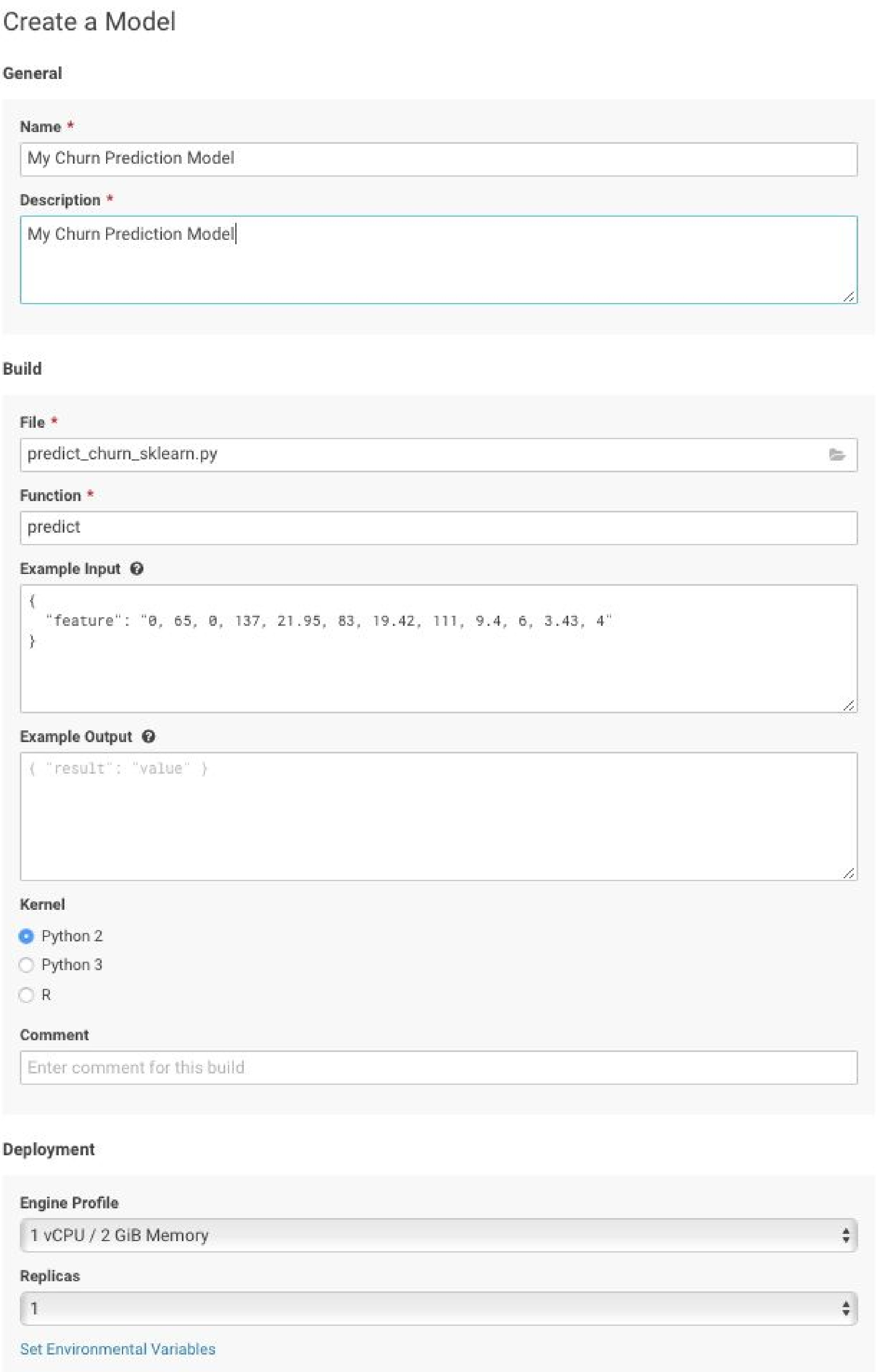
Step 2: Deploy the model

From the projects page of your project, select the ‘Models’ button.

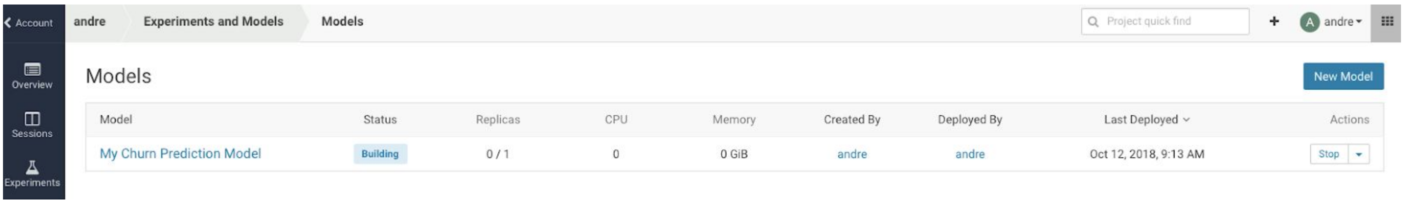


Select ‘New Model’, and populate specify the following configuration:

Name: something like “My Churn Prediction Model”
Description: Anything you want
File: predict_churn_sklearn.py
Function: predict
Example Inp: {
"feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4"
}
Kernal: Python 3
Engine: 0.5 vCPU / 2 GiB Memory
Replicas: 1

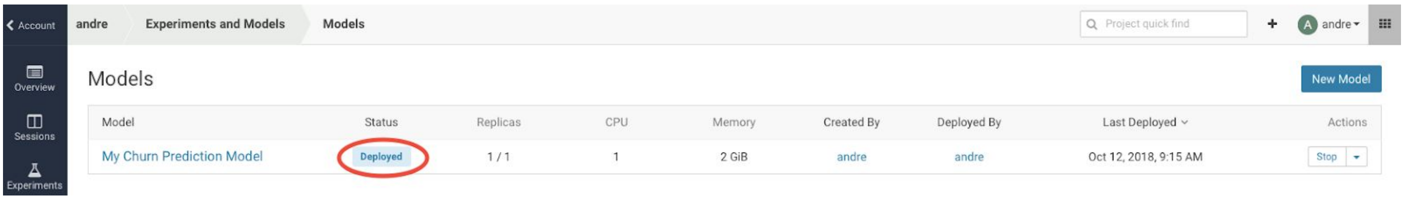


If all parameters are set, you can hit the ‘Deploy Model’ button. Wait till the model is deployed. This will take several minutes.

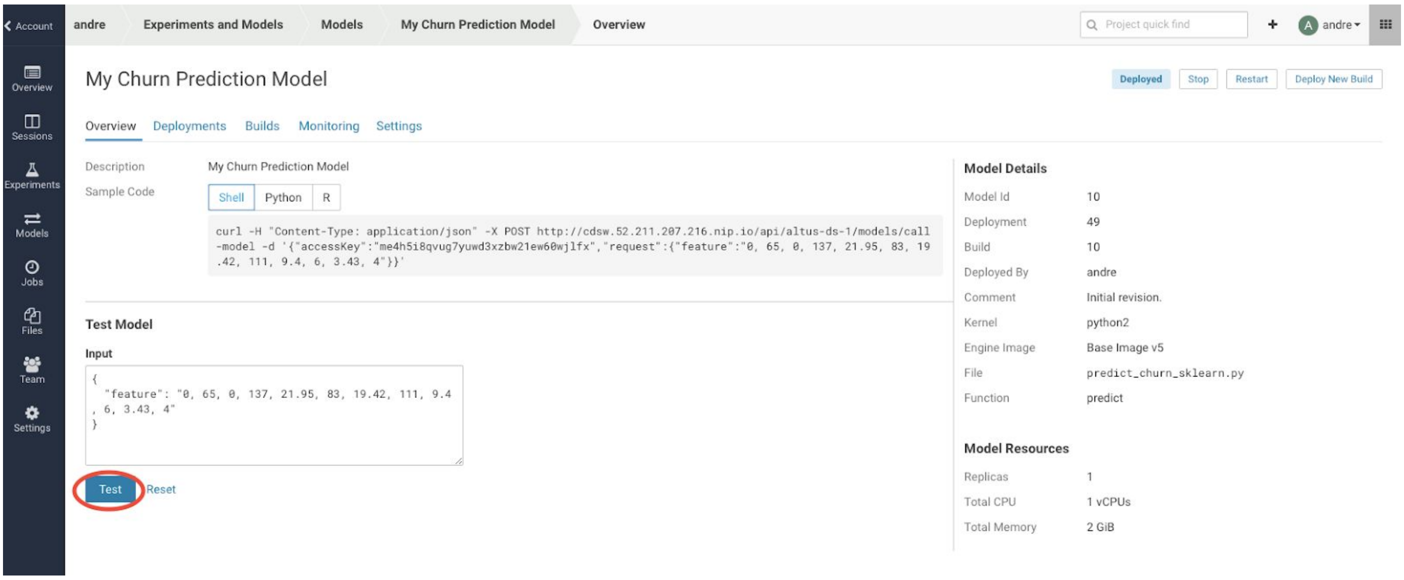


Step 3: Test the deployed model

After the several minutes, your model should get to the ‘Deployed’ state.



Now, click on the Model Name link, to go to the Model Overview page. From the that page, hit the ‘Test’ button to check if the model is working.



If your model is working, you should receive an output similar like this:

The green color with success is telling that our REST call to the model is technically working. And if you examine the response: {“result”: 1}, it returns a 1, which mean that customer with these features is likely to churn.

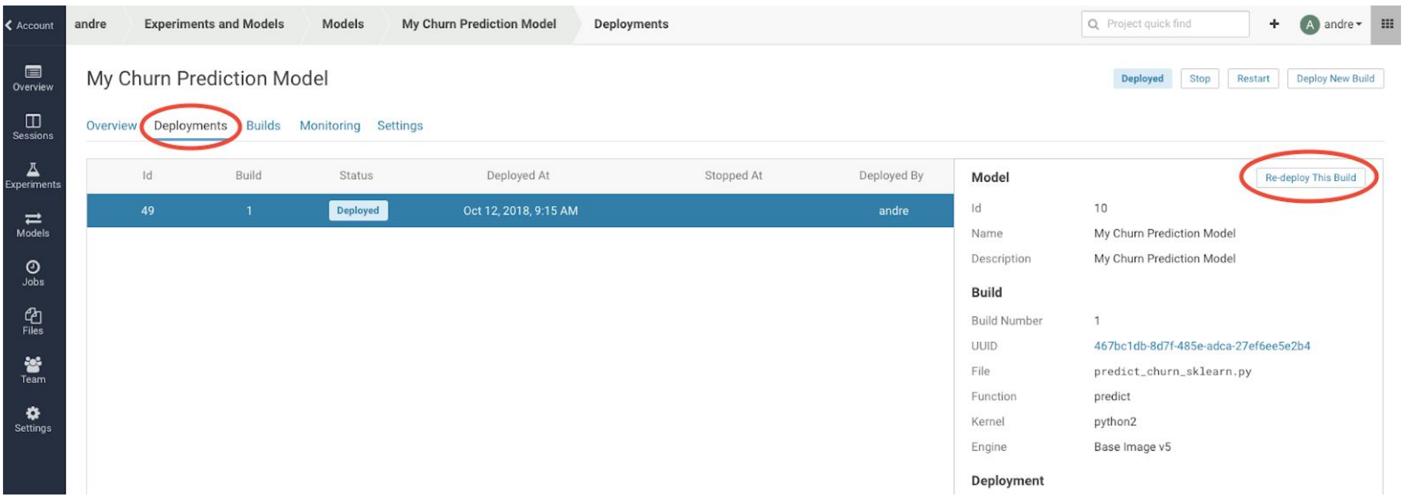
Now, lets change the input parameters and call the predict function again. Put the following values in the Input field:

```
{
  "feature": "0, 95, 0, 88, 26.62, 75, 21.05, 115, 8.65, 5, 3.32, 3"
}
```

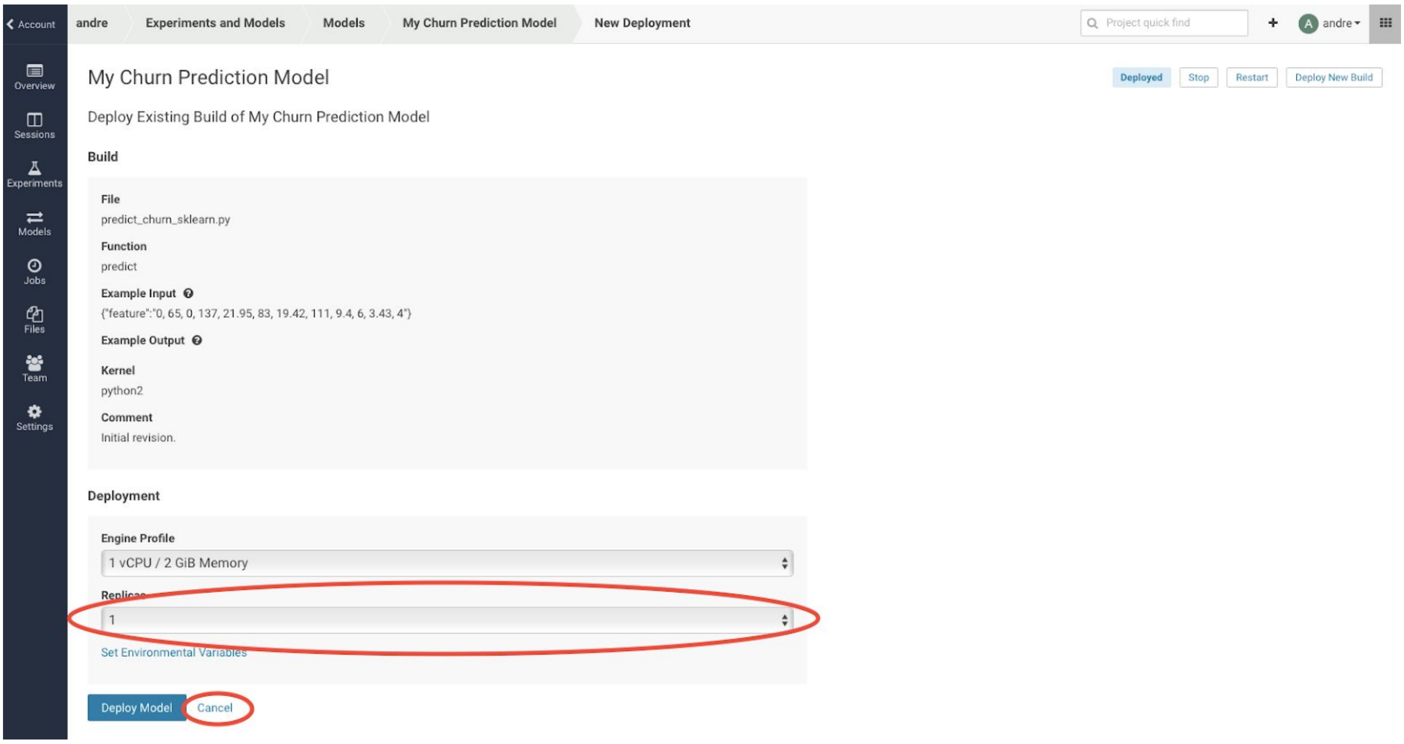
With these input parameters, the model returns 0, which mean that the customer is not likely to churn.

Step 4: Model Administration

When a model is deployed, Cloudera Data Science Workbench allows you to specify a number of replicas that will be deployed to serve requests. For each active model, you can monitor its replicas by going to the model's Monitoring page. On this page you can track the number of requests being served by each replica, success and failure rates, and their associated stderr and stdout logs. Depending on future resource requirements, you can increase or decrease the number of replicas by re-deploying the model.

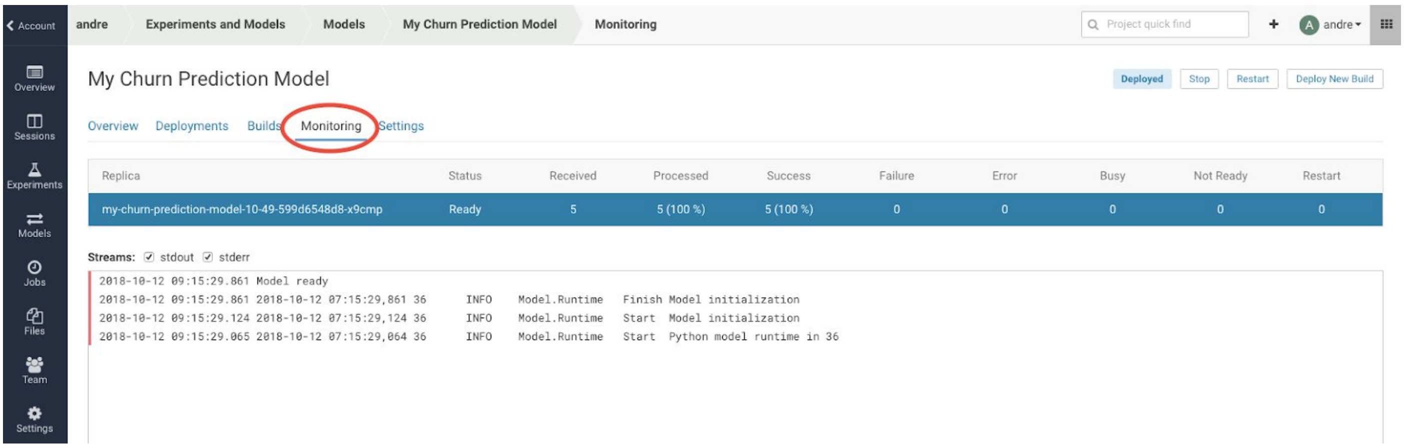


When you get to the re-deployment page, you can increase the number of replica's.



In order not to overload the cluster, hit the 'Cancel' button to return to the running model page.

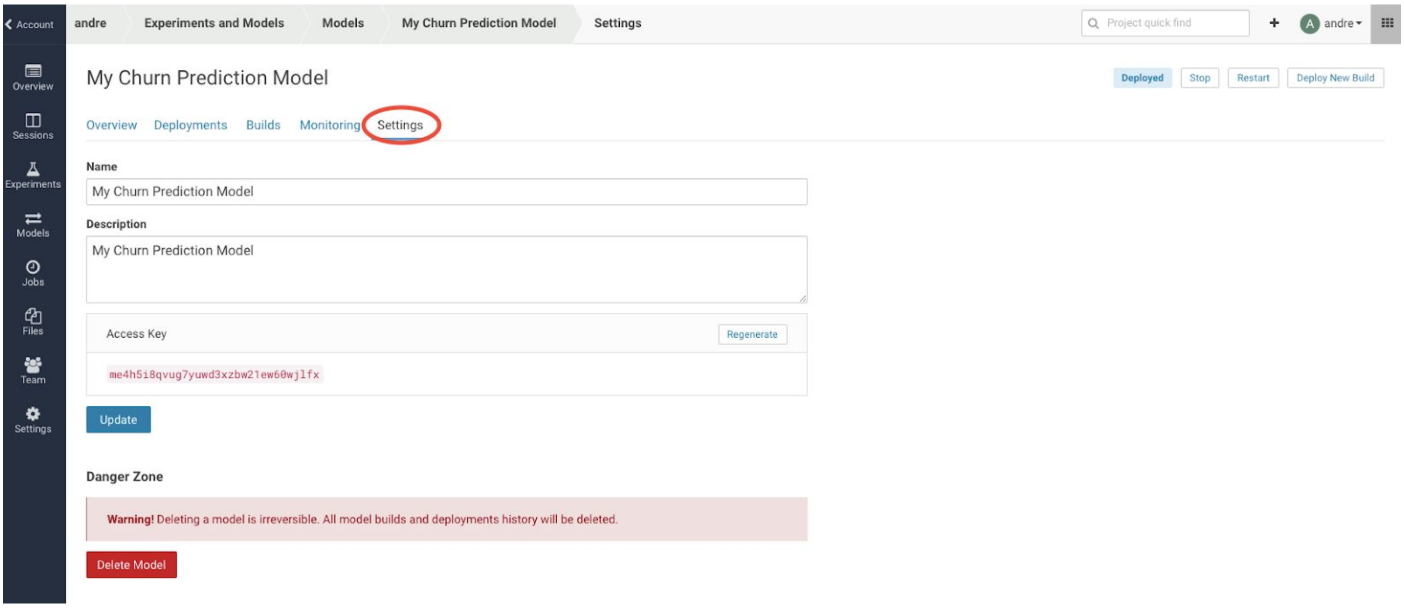
Now, navigate to the 'Monitoring' tab.



Several statistics of the model are displayed, like the number of times the model has been called, have been processed, etc.

Logfile information is also available here. The most recent logs are at the top of the pane (see image). stderr logs are displayed next to a red bar while stdout logs are by a green bar. Note that model logs and statistics are only preserved so long as the individual replica is active. When a replica restarts (for example, in case of bad input) the logs also start with a clean slate.

Now, navigate to the 'Settings' tab.

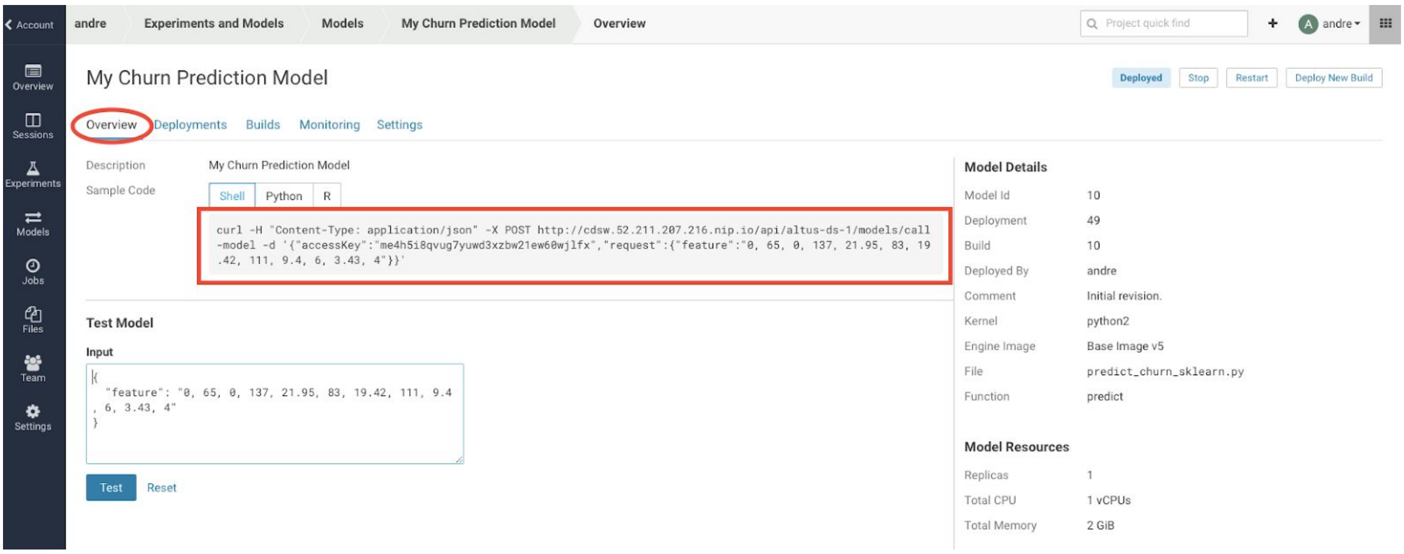


On the settings tab, you will be able to find the "Access Key" that is needed in order to call the model with a REST webservice call.

Step 5: Test the rest service from a commandline.

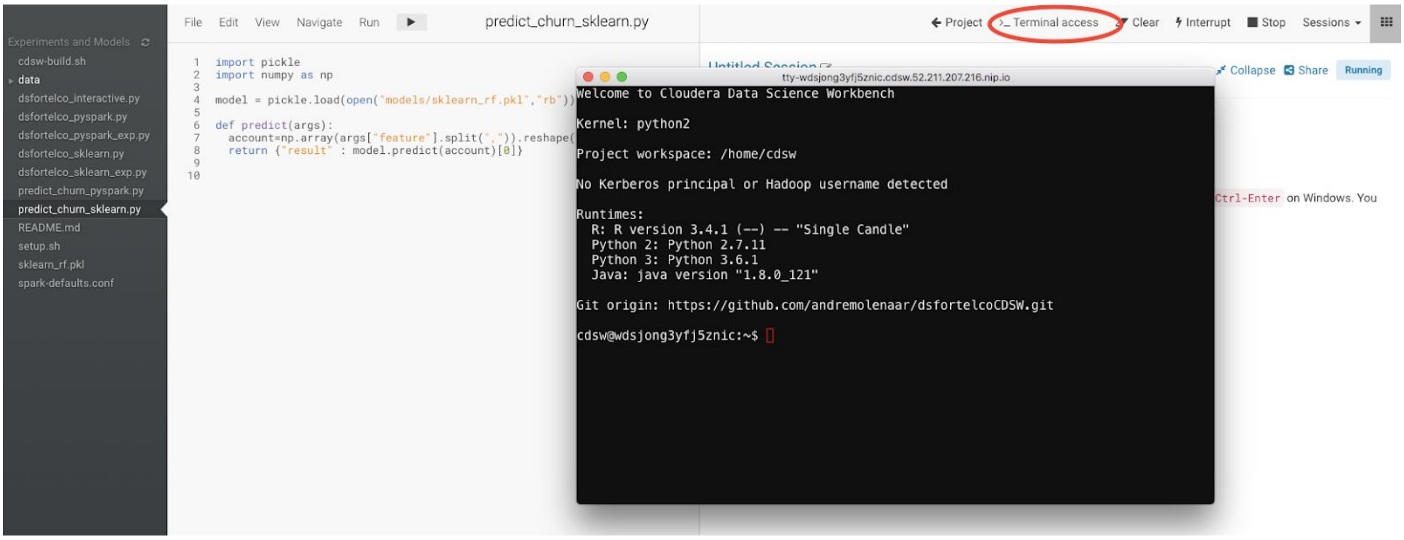
The last step in this workshop, is to test the predict function from another (virtual) machine, using the "curl" tool.

Navigate to the Overview tab of your running model.



Copy the whole shell statement, starting with 'curl -H ....”

Open a workbench session, running python 2 with 2 GB of memory. When the session is available, open a Terminal.



Now, paste the curl statement to the command prompt, and run the statement.

```
Welcome to Cloudera Data Science Workbench

Kernel: python2

Project workspace: /home/cdsw

No Kerberos principal or Hadoop username detected

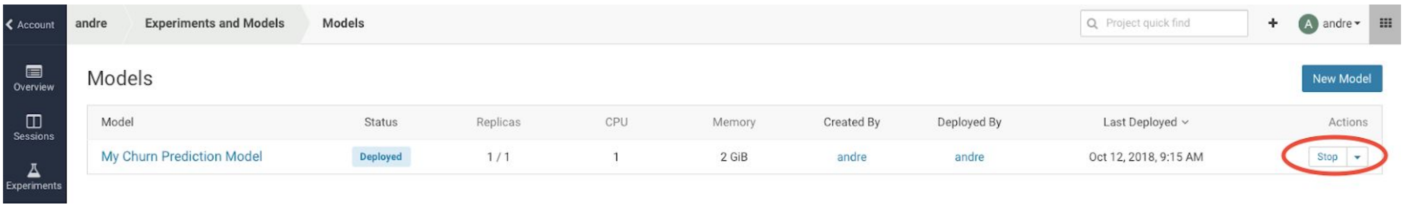
Runtimes:
  R: R version 3.4.1 (--) -- "Single Candle"
  Python 2: Python 2.7.11
  Python 3: Python 3.6.1
  Java: java version "1.8.0_121"

Git origin: https://github.com/andremolenaar/dsfortelcoCDSW.git

cdsw@wdsjong3yfj5znic:~$ curl -H "Content-Type: application/json" -X POST http://cds
w.52.211.207.216.nip.io/api/altus-ds-1/models/call-model -d '{"accessKey":"me4h5i8qv
ug7yuwd3xzbw21ew60wjlfx","request":{"feature":"0, 65, 0, 137, 21.95, 83, 19.42, 111,
 9.4, 6, 3.43, 4"}}'
{
  "success": true,
  "response": {
    "result": 1
  }
}cdsw@wdsjong3yfj5znic:~$
```

The response shows that the model is still running and making predictions.

That completes our lab with models. Please, freeup some resources for other people and new projects. So stop your workbench session. And from the Models page, also stop your deployed model.



Lab 13 – Face recognition with Python: Where is Lego?

In this lab, you will work with images in Python. Using a predefined library, you will try to locate faces on a photo. And once a face is found, you will try to match a specific face on the photo. The instructions for this lab are less detailed, so you might need to browse back in this document (or use google) to find the exact syntax for some statements.

The instructions for this lab will be using a new library, face_recognition.

Step 1:

Create a new project in CDSW with a Python template, and start a Python 3 workbench with 16GB of RAM

Step 2:

Install the face_recognition library using the "pip3 install" command.
As soon as the library is installed, stop your CDSW session. As soon as it is stopped, open a new CDSW session with 4 GB of memory, to allow other students resources to install the library.
Also, install the opencv-python library, to use some graphic processing capabilities.

Step 3:

Open a command prompt from your CDSW workbench, and copy all .jpg files from the hdfs directory /tmp/photos to the home directory of your CDSW session.
To copy files, you can use the command:

```
hdfs dfs -get
```

Step 4:

Try to display a photo using the Image command. Before you can use this command, you need to import some display libraries, with the statement:

```
from IPython.display import Image, display
```

As soon as the library is imported, try to display a photo. Use the command:

```
display(Image('<filename>'))
```

In the directory with .jpg files, search for the image with your name. This should be your photo as was found on LinkedIn.

Step 5:

Detect the exact location of the face on the image, using the face_recognition library.
First, load the library:

```
import face_recognition
```

Now you can use the following statement to find where the face is located on the photo:

```
my_photo = face_recognition.load_image_file("<filename>")
my_face_locations = face_recognition.face_locations(my_photo)
```

Check if your algorithm has detected a face, by showing the value.

```
my_face_locations
```

You will see a list of tuples, with the locations of where a face can be found on the photo. Most probably, you will only see 1 tuple, with 1 face on the photo. The output should be something like this:

```
[(32L, 107L, 94L, 45L)]
```

This is the representation of 1 tuple, with the values [(top, right, bottom, left)], which represent the pixel in the top right corner, as well as the bottom left corner. The face on the photo is located in the square between these 2 corners.

Step 6:
Cut the face out of the picture. To do this, use the corners found in step 4. First, extract the corners form the array of tuples. You can do this with the following command:

```
top, right, bottom, left = my_face_locations[0]
```

Now we can extract the face out of the photo, using the following statement:

```
from PIL import Image, ImageDraw
my_face=my_photo[top:bottom, left:right]
my_face_img=Image.fromarray(my_face)
```

And now visualise the image, to check if it worked.
We use some libraries from PIL for that, so we need to import that first.

```
display(my_face_img)
```

You should see the face only now.

Step 7:
Use the face_recognition library to encode the face to a 'match_code'.

```
my_face_encoding = face_recognition.face_encodings(my_photo)[0]
```

The face encoding is an array of numbers that represent the face from the picture. This array is used for matching to find a face that is similarly looking.

Step 8:
Encode the faces on group photo.
First, show the group photo with the statement:

```
from IPython.display import Image, display
display(Image(filename="teamphoto.jpg"))
```

Now, find the face locations on the group photo:

```
group_photo = face_recognition.load_image_file("teamphoto.jpg")
```

Step 9:
Write a loop that makes an encoding of each face in the photo, and that matches the face on the group photo with the encoding created in step 6.

A piece of example code is here:

```
group_photo = face_recognition.load_image_file("teamphoto.jpg")
group_face_locations = face_recognition.face_locations(group_photo)
print len(group_face_locations)
for face_location in group_face_locations:
  top, right, bottom, left = face_location
  print top, right, bottom, left
```

Your hints are:

- Find the number of faces in the group photo. Use the statement from Step 4.
- Create a loop to an encoding for each face. Encoding is done using the statement in Step 6.
- Match the encoding of my_photo and the face of the group_photo.
- `face_recognition.compare_faces([my_face_encoding], group_face_encoding)`
- If matching is True, then draw a box around the face on the group photo.
- Draw a box statement:
- `import cv2`
- `cv2.rectangle(<image>, (left, top), (right, bottom), (0,0,255), 2)`

If you don't feel like coding, you can also clone the github repository https://github.com/jmuselaers/face_recognition

Appendix

Cloudera Documentation
http://www.cloudera.com/documentation.html

Cloudera Data Science Workbench
https://www.cloudera.com/products/data-science-and-engineering/data-science-workbench.html

CDSW User Guide
https://www.cloudera.com/documentation/data-science-workbench/latest/topics/cdsw_user_guide.html

Troubleshooting Guide
https://www.cloudera.com/documentation/data-science-workbench/latest/topics/cdsw_troubleshooting.html

Recordings
Part 1 - Introduction
https://www.cloudera.com/content/dam/www/marketing/resources/webinars/introducing-cloudera-data-science-workbench-part1-recorded-webinar.png.landing.html

Part 2 – A Visual Dive into machine Learning
https://www.cloudera.com/content/dam/www/marketing/resources/webinars/part-2-visual-dive-into-machine-learning-and-deep-learning.png.landing.html

Part 3 – Data Science Models into production from beginner to end
https://www.cloudera.com/content/dam/www/marketing/resources/webinars/models-in-production-a-look-from-beginning-to-end-part3.png.landing.html