

Author: André Molenaar, Partner SE, DS Specialist

Version: 4.1



Data Science at Scale Lab Guide

Cloudera Data Science Workbench Labs

TABLE OF CONTENTS

INTRODUCTION	2
Lab 1 - Login to Cloudera Data Science Workbench (CDSW)	4
Lab 2 - Creating a new project	7
Lab 3 - Visualization and Sharing	16
Lab 4 - Hadoop Integration	21
Lab 5 - Pushing the Boundaries	23
Lab 6 - Running R	25
Lab 8 - Scala	27
Lab 9 - Project Creation using Local Files	32
Lab 10 - Scheduling Jobs	34
Lab 11 - Working with Models	42
Lab 12 - Face Recognition with Python: Where is Filippo?	54
APPENDIX	57
Recordings	57

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 ensured that all workloads can be pushed down to Cloudera.

Accelerate data science from exploration to production using R, Python, Spark and more

For data scientists



Open data science, your way.

Use R, Python, or Scala with your favorite libraries and frameworks



No need to sample.

Directly access data in secure Hadoop clusters through Apache Spark and Apache Impala



Reproducible, collaborative research.

Share insights with your whole team

For IT professionals



Bring analysis to the data.

Give your data science team the freedom to work how they want, when they want



Secure by default.

Stay compliant with out-of-the-box support for full Hadoop security



Flexible deployment.

Run on-premises or in the cloud

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.



Programming language and software environment for statistical computing and graphics.

Best known in: **Academia and statistics community.**



High-level programming language for general-purpose programming.

Best known in: **Machine learning and data engineering community.**



General-purpose functional programming language with a strong static type system.

Best known in: **Data engineering community, due to Spark.**

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. Use the link that is provided to you. The url should look something like:

`http://cdsw.138.91.159.140.nip.io/`

Select the "Sign Up for New Account" link.

Sign In to Data Science Workbench

Username

Password

Sign In

Forgot Password?

Sign Up for a New Account

Provide your name, surname, email address and password. Make sure to use the provided userXX which you received by email, as well as a valid and accessible email address. In order to make things easy, it is best to use the same password here (Cloudera1), as provided in the email.

Sign Up for Data Science Workbench

Andre Molenaar

user01

Your personal profile will be located at:
<http://cdsw.138.91.159.140.nip.io/user01>

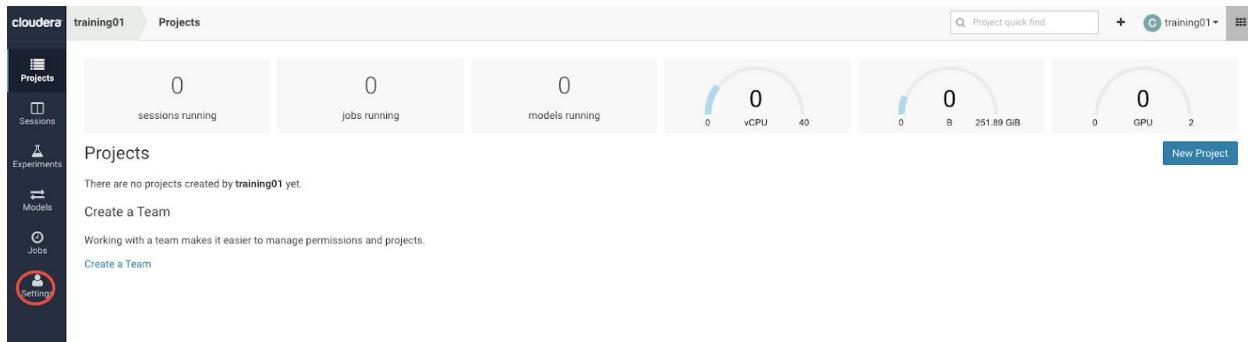
andre.molenaar@cloudera.com

Sign Up

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

As soon as you hit ‘Sign Up’, your account will be created and you will login to the Cloudera Data Science Workbench.

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.



Now, authenticate to the cluster using the hdfs credentials that you received by email.

User Settings

Profile SSH Keys Hadoop Authentication API Keys

Kerberos

To authenticate to Kerberos, enter your principal and either enter your password or upload a keytab file.

Principal

user01

Credentials

Password Keytab

Enter Password

Authenticate

Show Kerberos configuration

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

training01 Settings Hadoop Authentication

Project quick find + C training01

User Settings

Profile SSH Keys Hadoop Authentication API Keys

Kerberos

Kerberos authentication

✓ Currently authenticated as training01

Sign out

Show Kerberos configuration

Lab 2 - Creating a new project

You will see the project window as follows:

The screenshot shows the Cloudera Data Science Workbench Labs interface. At the top, there's a header with the Cloudera logo, the user name 'amolenaar', and a 'Projects' tab. Below the header are several performance metrics: 'sessions running' (0), 'jobs running' (0), 'models running' (0), 'vCPU' usage (0/32), and 'B' usage (0/125.49 GB). A search bar and a '+' icon for adding new projects are also at the top. On the left, a sidebar lists navigation options: Projects (selected), Experiments, Models, Jobs, and Settings. The main content area shows a message: 'There are no projects created by amolenaar yet.' It also includes links to 'Create a Team' under both 'Jobs' and 'Experiments'.

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

New Project

Copy and paste this GitHub URL into the Git tab

https://github.com/andremolenaar/cdsw_workshop_azure.git

Project_name = your user name and labs.

You should see this:

Create a New Project

Project Name

Andre Molenaar CDSW Labs

Project Visibility

- Private** - Only added collaborators can view the project.
- Public** - All authenticated users can view this project.

Initial Setup

Blank

Template

Local

Git

`://github.com/andremolenaar/cdsw_workshop_azure.git`

Create Project

The screenshot shows the Cloudera Data Science Workbench interface. At the top, there's a navigation bar with 'Account' and 'user01'. Below it, the project name 'Andre Molenaar CDSW Labs' is displayed. On the left, a sidebar menu includes 'Overview', 'Sessions', 'Experiments', 'Models', 'Jobs', 'Files', 'Team', and 'Settings'. The 'Files' section is currently active, showing a list of files in the project directory. The files listed are:

Name	Size	Last Modified
data	-	just now
1_python.py	2.97 kB	just now
2_pyspark.py	1.61 kB	just now
3_tensorflow.py	3.31 kB	just now
4_basket_analysis.r	1.33 kB	just now
5_shiny.R	769 B	just now
groceries.csv	498.73 kB	just now
README.md	1.73 kB	just now
server.R	536 B	just now
spark-defaults.conf	74 B	just now
ui.R	689 B	just now
utils.py	1.07 kB	just now

On the left hand side panel you will see 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:

User	Role
fil	Admin
Filippo (flambiente)	Admin
Sofie (Gundersen)	Admin
Sofia Thorén (sofiathoren)	Admin

You can add anybody to your project. If you cannot find your neighbor, you can use the administrator user00 to share your project with:

Collaborators

This project is **private**. Only collaborators can view and edit this project. [Change Settings.](#)

Add Collaborator

Collaborator	Role
amolenaar	Admin
admin	Viewer

Granting write or admin permission to other users may have security impact since it gives them full access to your project files and running sessions. Write or admin permission should only be granted to trusted users.

Click on the Settings icon and then the Engine tab:

Project Settings

Engine Image
Select the Docker image that Cloudera Data Science Workbench should use to run sessions and jobs in this project. If you'd like to use a different image, contact your site administrator.

Environmental Variables
Set project environmental variables that can be accessed from your scripts.

Name	Value	Actions
		Add

Press tab or enter to add another.

Save Environment

Security
Environmental variable **values** are only visible to collaborators with **write** or higher access. They are a great way to securely store confidential information such as your AWS or database credentials. Names are available to all users with access to the project.

Under Project Settings you will see:

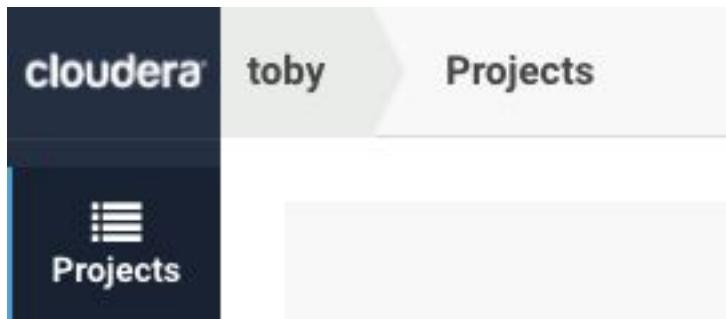
Options - Project Name and Description, Private or Public

Engine - Engine image and environment variables

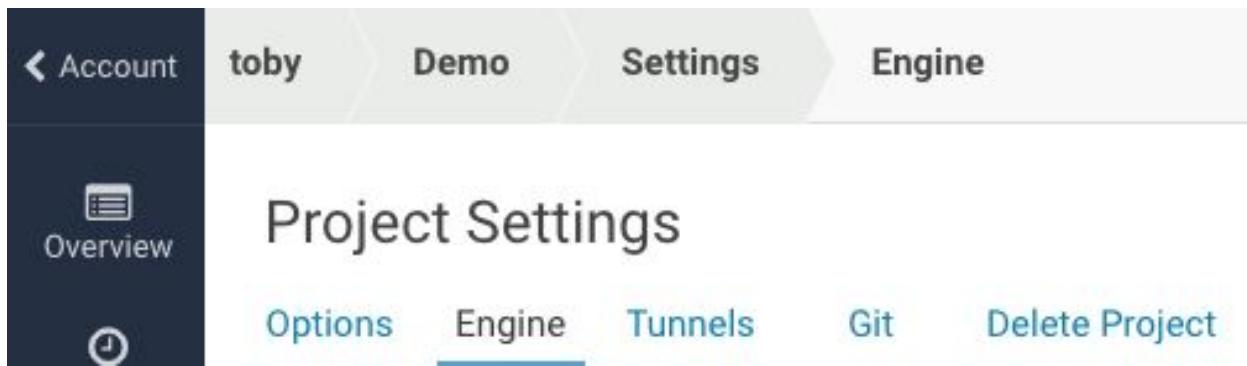
Tunnels - SSH tunnels allow you to easily connect to firewalled resources such as databases or Hadoop

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:

The screenshot shows the 'Andre_CDSW_Demos' project page. On the left is a sidebar with links for Account, Overview, Sessions, Experiments, Models, Jobs, Files, Team, and Settings. A license notice at the bottom left states 'License Expires in 17 days'. The main area has tabs for 'Overview', 'Models', 'Jobs', and 'Files'. The 'Files' tab is active, showing a list of files with columns for Name, Size, and Last Modified. A red circle highlights the 'Open Workbench' button in the top right corner.

Name	Size	Last Modified
-	-	6 minutes ago
data	3.01 kB	6 minutes ago
1.python.py	1.60 kB	6 minutes ago
2.pyspark.py	3.39 kB	6 minutes ago
3.tensorflow.py	2.50 kB	6 minutes ago
4.sparklyr.R	769 B	6 minutes ago
5.shiny.R	1.74 kB	6 minutes ago
README.md	536 B	6 minutes ago
server.R	74 B	6 minutes ago
spark-defaults.conf	689 B	6 minutes ago
ui.R	1.09 kB	6 minutes ago
utils.py	-	-

The screenshot shows the 'Start New Session' page. It includes fields for 'Engine Image - Configure' (set to 'Base Image v7 - docker.repository.cloudera.com/cdsw/engine:7'), 'Select Engine Kernel' (Python 3 selected), 'Select Engine Profile' (1 vCPU / 2 GiB Memory), and two buttons: 'Launch Session' (circled in red) and 'Run Experiment..'

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/2GiB Engine (and use this size engine for all your Python sessions in this workshop) and then the Launch Session button.

Start New Session

Engine Image - Configure
Base Image v7 - docker.repository.cloudera.com/cdsw/engine:7

Select Engine Kernel

- Python 2
- Python 3
- Scala
- R

Select Engine Profile

1 vCPU / 2 GiB Memory

Launch Session or **Run Experiment..**

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.

```

File Edit View Navigate Run README.md
Andre Molenaar CDSW Labs
1 # Cloudera Data Science Workbench demos
2 Basic tour of Cloudera Data Science Workbench.
3
4 ## Workbench
5 There are 4 scripts provided which walk through the interactive capabilities of Cloudera Data Science Workbench:
6 1. **Basic Python visualizations (Python 2).** Demonstrates:
7     - Markdown via comments
8     - Jupyter-compatible visualizations
9     - Accessing local files via dfs -ls /.
10 2. **PySpark (Python 2).** Demonstrates:
11     - Ease connectivity to (kerberized) Spark in YARN client mode.
12     - Accessing local files via dfs -ls /.
13 3. **TensorFlow (Python 2).** Demonstrates:
14     - Ability to install and use custom packages (e.g. pip search tensorflow)
15     - Using TensorFlow in Python
16 4. **R.** Demonstrates:
17     - Run R code on CDSW, showing arules library
18 5. **Advanced visualization with Shiny (R).** Demonstrates:
19     - Use of "shiny" to provide interactive graphics inside CDSW
20
21 ## Jobs
22 We recommend setting up a **Nightly Analysis** job to illustrate how data scientists can
23 automatically trigger analysis and reporting.
24
25 ## Setup instructions
26 Note: You only need to do this once.
27
28 1. In a Python 3 Session:
29     1. Python 3
30     !pip3 install --upgrade dask
31     !pip3 install --upgrade keras
32     !pip3 install --upgrade matplotlib==2.0.0
33     !pip3 install --upgrade numpy
34     !pip3 install --upgrade protobuf
35     !pip3 install --upgrade tensorflow==1.3.0
36     !pip3 install --upgrade seaborn
37
38 Note: you must then stop the session and start a new Python session in order for all the packages to be available.
39
40 2. In an R Session:
41     1. R
42     install.packages('sparklyr')
43     install.packages('pliofly')
44     install.packages('nycflights13')
45     install.packages('Lahman')
46     install.packages('ggplot2')
47     install.packages('shiny')
48     install.packages('arules')
49     install.packages('readr')
50
51
52 3. Stop all sessions, then proceed.
53
54
55

```

Untitled Session By Administrator – Python 3 Session – 1 vCPU / 2 GiB Memory – just now

Getting Started

This is your Python 3 session. Your editor is on the left and your input prompt is on the bottom. To install a package type: `!pip3 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.

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

```
!pip3 install --upgrade dask
!pip3 install --upgrade keras
!pip3 install --upgrade matplotlib==2.0.0.
!pip3 install --upgrade pandas_highcharts
!pip3 install --upgrade protobuf
!pip3 install --upgrade tensorflow==1.3.0.
!pip3 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)':

1. In a Python 3 Session:

```
```Python
!pip3 install --upgrade dask
!pip3 install --upgrade keras
!pip3 install --upgrade matplotlib==2.0.0.
!pip3 install --upgrade pandas_highcharts
!pip3 install --upgrade protobuf
!pip3 install --upgrade tensorflow==1.3.0.
!pip3 install --upgrade seaborn
```

```

Run Line(s)
⌘Enter

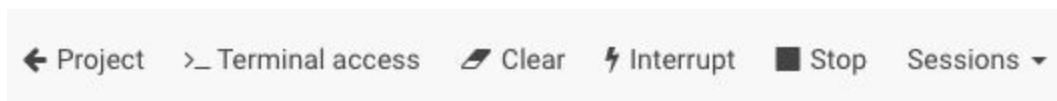
Select All
⌘A

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

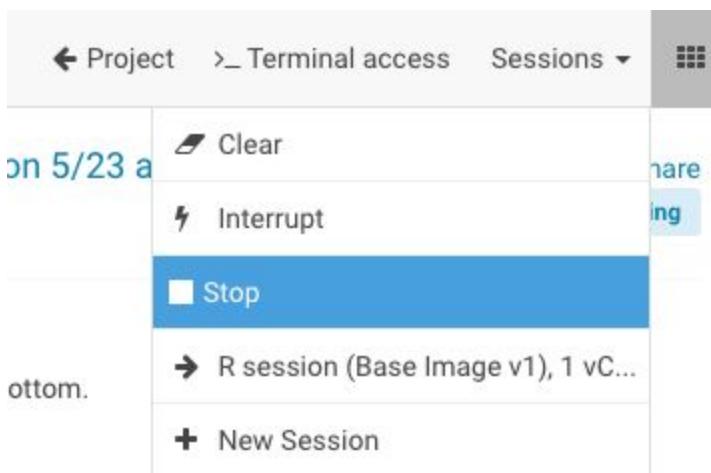
```
...@... python3 -m pip install --upgrade dask
...@... python3 -m pip install --upgrade keras
...@... python3 -m pip install --upgrade matplotlib==2.0.0.
...@... python3 -m pip install --upgrade pandas_highcharts
...@... python3 -m pip install --upgrade protobuf
...@... python3 -m pip install --upgrade tensorflow==1.3.0.
...@... python3 -m pip install --upgrade seaborn
Requirement not upgraded as not directly required: numpy>=1.9.3 in /usr/local/lib/python2.7/site-packages (from seaborn) (1.12.1)
Requirement not upgraded as not directly required: scipy>=0.14.0 in /usr/local/lib/python2.7/site-packages (from seaborn) (1.1.0)
Requirement not upgraded as not directly required: pandas>=0.15.2 in /usr/local/lib/python2.7/site-packages (from seaborn) (0.20.1)
Requirement not upgraded as not directly required: matplotlib>=1.4.3 in /usr/local/lib/python2.7/site-packages (from seaborn) (2.0.0)
Requirement not upgraded as not directly required: python-dateutil in /usr/local/lib/python2.7/site-packages (from pandas>=0.15.2>seaborn) (2.7.3)
Requirement not upgraded as not directly required: pytz>=2011k in /usr/local/lib/python2.7/site-packages (from pandas>=0.15.2>seaborn) (2018.4)
Requirement not upgraded as not directly required: subprocess32 in /usr/local/lib/python2.7/site-packages (from matplotlib>=1.4.3>seaborn) (3.5.2)
Requirement not upgraded as not directly required: pyparsing!=2.0.0,!>2.0.4,!>2.1.2,!>2.1.6,>=1.5.6 in /usr/local/lib/python2.7/site-packages (from matplotlib>=1.4.3>seaborn) (2.2.0)
Requirement not upgraded as not directly required: six>=1.10 in /usr/local/lib/python2.7/site-packages (from matplotlib>=1.4.3>seaborn) (1.11.0)
Requirement not upgraded as not directly required: functools32 in /usr/local/lib/python2.7/site-packages (from matplotlib>=1.4.3>seaborn) (3.2.3.post2)
Requirement not upgraded as not directly required: cycler>=0.10 in /usr/local/lib/python2.7/site-packages (from matplotlib>=1.4.3>seaborn) (0.10.0)
Building wheels for collected packages: seaborn
    Running setup.py bdist_wheel for seaborn ... ?251done
?251 Stored in directory: /home/cdsaw/.cache/pip/wheels/fc/1c/74/c8f80a532c06a789599b8659b117ec7d7574cac4a06f7dabfe
Successfully built seaborn
grin 1.2.1 requires argparse>=1.1, which is not installed.
bokeh 0.12.10 has requirement futures>=3.0.3, but you'll have futures 2.1.4 which is incompatible.
Installing collected packages: seaborn
Successfully installed seaborn-0.9.0
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

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 Session drop down (when there's little room for buttons):



ter on Mac or `Ctrl-Enter` on Windows. You can also

13. Now launch an R session. **Use a 1vCPU, 2GiB Engine.** Use this size engine for **all** your R sessions during this workshop.

Start New Session

Engine Image - Configure

Base Image v7 - docker.repository.cloudera.com/cdsw/engine:7

Select Engine Kernel

- Python 2
- Python 3
- Scala
- R

Select Engine Profile

1 vCPU / 2 GiB Memory

Launch Session

or Run Experiment..

14. This time we're going to execute lines 42 through 49 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:

```
install.packages('sparklyr')
install.packages('plotly')
install.packages("nycflights13")
install.packages("Lahman")
install.packages("mgcv")
install.packages('shiny')
install.packages("arules")
install.packages("readr")
```



A screenshot of an R session window. The code in the editor is:

```
40 2. In an R Session:
41 ````R
42 install.packages('sparklyr')
43 install.packages('plotly')
44 install.packages("nycflights13")
45 install.packages("Lahman")
46 install.packages("mgcv")
47 install.packages('shiny')
48 install.packages("arules")
49 install.packages("readr")
```

To the right of the code, there is a toolbar with the following options:

| | |
|-------------|---------|
| Run Line(s) | ⌘ Enter |
| Select All | ⌘ A |

This process will take 10 to 15 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.

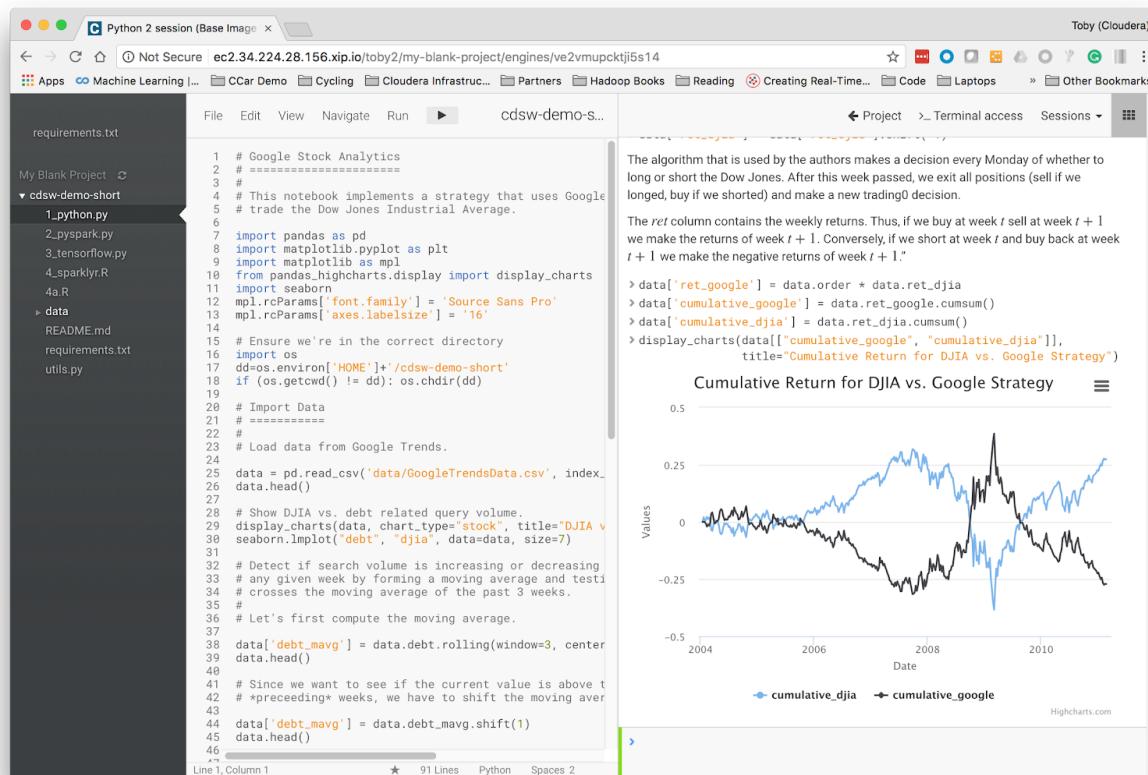
Data Science is often about visualizing ideas, and then sharing them to persuade others to take action. CDSW lets you use the visualization tools you'd use naturally, and adds a neat twist to the whole idea of sharing.

Let's get started:

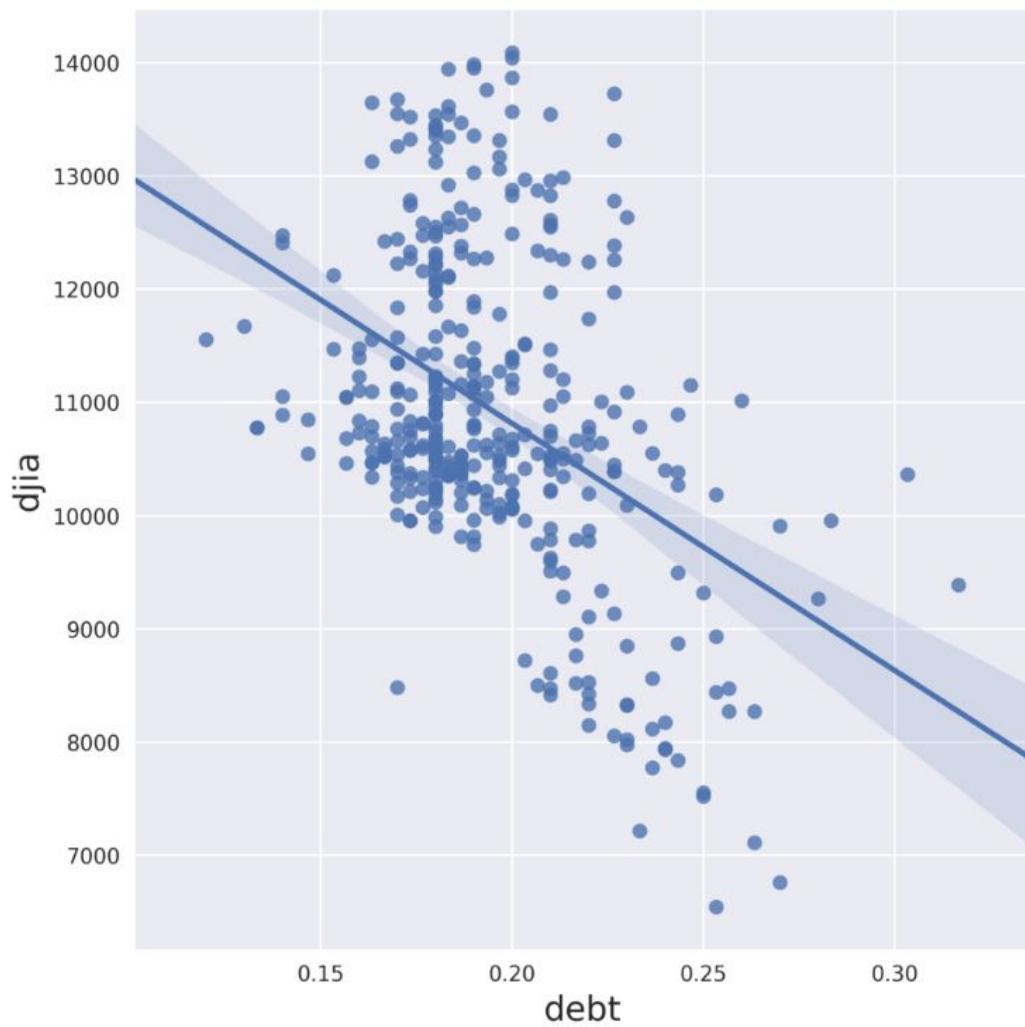
1. Start up a new Python 3 session (1vCPU, 2GiB) in the same manner you did before.
2. Select 1_python.py in the file browser
3. Run the entire file (multiple ways of doing that - try to figure out more than one way. It should be pretty obvious!).



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:

- 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-s7dkodjq' class="sense-embed-container" style="width:100%;"><script id='embedding-script-21' src='http://consoles.cdsweb52.214.150.212.nip.io/js/sense-embed.js' data-cell-url='http://livelog.cdsweb52.214.150.212.nip.io/topics/xeupfj2oeyaktdpa_output/21' data-auth-token='eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRoIjp7InJlYWQiOlsieGV1cGZqMm9leWFrdGRwYV8qIl0sIndyaXRlIjpbdashAssetsCdnRoot='http://consoles.cdsweb52.214.150.212.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):

 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:

 These results are **private**. Only project members can view.

Share with Others

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

The screenshot shows a sharing dialog box. At the top left is a lock icon followed by the text "These results are being **shared**". To the right is a blue "Stop Sharing" button. Below this is a checked checkbox labeled "Hide code and text". Under the heading "Who can view:" is a radio button group. The first option, "Anonymous visitors with the link", is selected (indicated by a blue dot) and has a tooltip "Currently shared with no one.". The other two options are "Any logged in user with the link" and "Specific users/teams with the link (Change...)".

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!)

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

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!

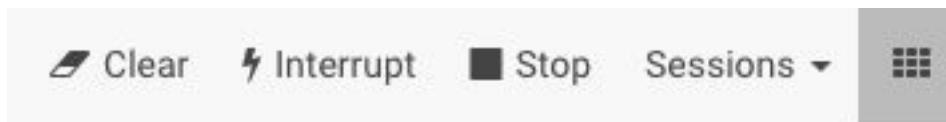
Lab 4 - Hadoop Integration

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

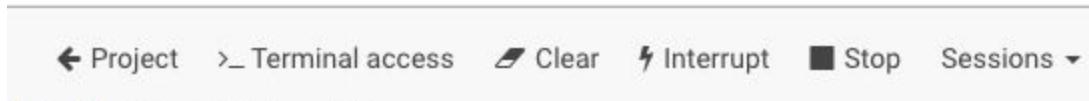
Execute the following instructions.

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



6 at 10:51 Expand Share Running

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 / 2 GB RAM
4. Run line 34:
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/cds
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@6mzzmuu4vvgsbjy:~$ 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@6mzzmuu4vvgsbjy:~$
```

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:

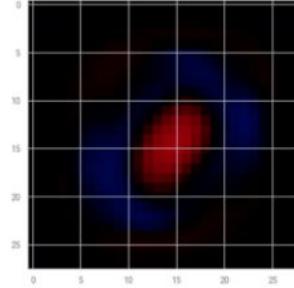
1. Select 3_tensorflow.py
2. Lines 2 through 6 show the imports of the various libraries (we installed them in Step 3)
3. Run 3_tensorflow.py in your current Python session (you might like to 'clear' your output screen first)- the input handwritten number images are shown, along with some images of feature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



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 - Running R

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 (1vCPU, 2GiB engine) and executed lines 42 through 50 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.

Start New Session

Engine Image - [Configure](#)

Base Image v7 - docker.repository.cloudera.com/cdsw/engine:7

Select Engine Kernel

- Python 2
- Python 3
- Scala
- R

Select Engine Profile

1 vCPU / 2 GiB Memory

[Launch Session](#)

or

[Run Experiment..](#)

2. Select (and run) 4_basket_analysis.r

The right hand side output window should (eventually) look like this (more or less - depending on your screen real-estate):

File Edit View Navigate Run ▶ 4_basket_analysis.r

```

2 # Running a Basket Analysis
3 # For small datasets, the R package arules can be used:
4
5 library(arules)
6
7 # Read the example dataset groceries
8 groceries <- read.transactions("groceries.csv", sep = ",")
9
10 library(readr)
11
12
13 # Show some statistics of the dataset
14 summary(groceries)
15 inspect(groceries[1:5])
16 itemFrequency(groceries[, 1:3])
17 itemFrequencyPlot(groceries, support = 0.1)
18 itemFrequencyPlot(groceries, topN = 20)
19 image(groceries[1:5])
20 image(sample(groceries, 100))
21
22 # Run an association model rules model
23 groceryrules <- apriori(groceries,
24   parameter = list(support = 0.006,
25     confidence = 0.25,
26     minlen = 2))
27
28 # Show the rules
29 groceryrules
30 summary(groceryrules)
31 inspect(groceryrules[1:3])
32 inspect(sort(groceryrules, by = "lift")[1:10])
33
34 # Finding rules with Berries
35 berryrules <- subset(groceryrules, items %in% "berries")
36 inspect(berryrules)
37
38 # Writing rules to a .csv file
39 write(groceryrules, file = "groceryrules.csv",
40   sep = ";", quote = TRUE, row.names = FALSE)
41
42 # Now, convert the dataset to an R dataframe file, for later usage.
43 groceryrules_df <- as(groceryrules, "data.frame")
44 str(groceryrules_df)
45
46 # For bigger datasets, Spark with the MLlib library can be used.
47

```

[5] {other vegetables, tropical fruit} => {root vegetables} 0.007930859 0.4028619 3.68692 78
[6] {beef, whole milk} => {root vegetables} 0.008032537 0.3779904 3.467851 79
[7] {other vegetables, pip fruit} => {tropical fruit} 0.009456024 0.3618677 3.448613 93
[8] {pip fruit, yogurt} => {tropical fruit} 0.006405694 0.3559322 3.392048 63
[9] {citrus fruit, other vegetables} => {root vegetables} 0.010371124 0.3591549 3.295045 102
[10] {other vegetables, whole milk, yogurt} => {tropical fruit} 0.007625826 0.3424658 3.263712 75

Finding rules with Berries

```

> berryrules <- subset(groceryrules, items %in% "berries")
> inspect(berryrules)

lhs rhs support confidence lift count
[1] {berries} => {whipped/sour cream} 0.009049314 0.2721713 3.796886 89
[2] {berries} => {yogurt} 0.010574479 0.3180428 2.279848 104
[3] {berries} => {other vegetables} 0.010269446 0.3088685 1.596288 101
[4] {berries} => {whole milk} 0.011794611 0.3547401 1.388328 116

```

Writing rules to a .csv file

```

> write(groceryrules, file = "groceryrules.csv",
      sep = ";", quote = TRUE, row.names = FALSE)
Now, convert the dataset to an R dataframe file, for later usage.
> groceryrules_df <- as(groceryrules, "data.frame")
> str(groceryrules_df)

'data.frame': 463 obs. of 5 variables:
 $ rules : Factor w/ 463 levels "(baking powder) => {other vegetables},...: 340 302 207 206 208
 341 402 21 139 148 ...
 $ support : num 0.00691 0.0061 0.00702 0.00773 0.00773 ...
 $ confidence: num 0.4 0.405 0.431 0.475 0.475 ...
 $ lift : num 1.57 1.59 3.96 2.45 1.86 ...
 $ count : num 68 60 69 76 69 70 67 63 88 ...

```

For bigger datasets, Spark with the MLlib library can be used.

>

Line 34, Column 1 ★ 47 Lines R Spaces 2

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!

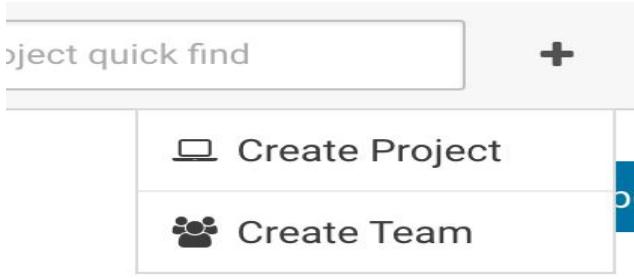
Lab 7 - Scala

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:

Create a New Project

Project Name

Scala

Project Visibility

- Private** - Only added collaborators can view the project.
- Public** - All authenticated users can view this project.

Initial Setup

Blank

Template

Local

Git

Scala

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:

The screenshot shows the Cloudera Data Science Workbench interface. On the left is a sidebar with icons for Account, Sessions, Experiments, Models, Jobs, Files, Team, and Settings. The main area shows a 'Scala' project. At the top, there's a search bar, a 'Fork' button, and an 'Open Workbench' button. Below the search bar, it says 'amolenaar Scala'. Under 'Models', it says 'This project has no models yet. Create a new model.' Under 'Jobs', it says 'This project has no jobs yet. Create a new job to document your analytics pipelines.' Under 'Files', there's a table listing files:

Name	Size	Last Modified
data	-	just now
examples	-	just now
auction-analysis.scala	2.12 kB	just now
log4j.properties	21 B	just now
pi.scala	837 B	just now
README.md	2.05 kB	just now
spark-defaults.conf	88 B	just now
wordcount.scala	562 B	just now

Show Hidden Files

5. Hit 'Open Workbench' in the top right and let's go run some Scala code:
6. Start a Scala session

Select Engine Kernel

- Python 2
- Python 3
- Scala
- R

Select Engine Profile

1 vCPU / 2 GiB Memory

Launch Session

or

Run Experiment...

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/user01/kmeans_data.txt")
```

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

The screenshot shows the Cloudera Data Science Workbench interface. On the left, the file tree shows a directory structure with files like kmeans.scala, auction-analysis.scala, data, examples, movieALS.scala, log4j.properties, pi.scala, README.md, spark-defaults.conf, and wordcount.scala. The main area displays the contents of kmeans.scala:

```

1 import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
2 import org.apache.spark.mllib.linalg.Vectors
3 import org.apache.commons.io.FileUtils
4 import java.io.File
5 import sys.process.-
6
7 //load local data to hdfs
8 //hdfs dfs -put data/kmeans_data.txt /tmp" !
9
10 //example kmeans clustering script
11 val data = sc.textFile("/user/training20/kmeans_data.txt")
12 val parsedData = data.map(s => Vectors.dense(s.split(" ").map(_.toDouble))).cache()
13
14 // Cluster the training data set into two classes with KMeans
15 val numClusters = 2
16 val numIterations = 20
17 val clusters = KMeans.train(parsedData, numClusters, numIterations)
18
19 // Evaluate clustering by computing Within Set Sum of Squared Errors
20 val WSSSE = clusters.computeCost(parsedData)
21 println(s"Within Set Sum of Squared Errors = $WSSSE")
22
23 // Save the model
24 val output = "output/KMeansExample/KMeansModel"
25 FileUtils.deleteQuietly(new File(output))
26 clusters.save(sc, output)
27
28 //example of loading and predicting on the model we created
29 val sameModel = KMeansModel.load(sc, output)
30 sameModel.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
31   println(s"Cluster Center $idx: ${center}")
32 }
33 sameModel.predict(Vectors.dense(7,5,6))
34

```

Below the code editor, it says "Line 11, Column 41". At the bottom, there are tabs for ★, 34 Lines, Scala, and Spaces 2. To the right, the terminal window shows the execution of the script:

```

> import java.io.File
> import sys.process.-
Load local data to hdfs
>hdfs dfs -put data/kmeans_data.txt /tmp" !

example kmeans clustering script

> val data = sc.textFile("/user/training20/kmeans_data.txt")
> val parsedData = data.map(s => Vectors.dense(s.split(" ").map(_.toDouble))).cache()
Cluster the training data set into two classes with KMeans

> val numClusters = 2
> val numIterations = 20
> val clusters = KMeans.train(parsedData, numClusters, numIterations)
Evaluate clustering by computing Within Set Sum of Squared Errors

> val WSSSE = clusters.computeCost(parsedData)
> println(s"Within Set Sum of Squared Errors = $WSSSE")
Within Set Sum of Squared Errors = 0.1199999999994547

Save the model

> val output = "output/KMeansExample/KMeansModel"
> FileUtils.deleteQuietly(new File(output))
false

> clusters.save(sc, output)
example of loading and predicting on the model we created

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

Cluster Center 0: [9.1,9.1,9.1]
Cluster Center 1: [0.1,0.1,0.1]
> sameModel.predict(Vectors.dense(7,5,6))
0

```

If you get a `FileNotFoundException` 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/KMeansModel$SaveLoadV1_0$ already exists
  StackTrace:   at org.apache.hadoop.mapred.FileOutputFormat.checkOutputSpecs(FileOutputFormat.java:131)
               at org.apache.spark.rdd.PairRDDFunctions$$anonfun$saveAsHadoopDataset$1.apply$mcV$sp(PairRDDFunctions.scala:109)
               at org.apache.spark.rdd.PairRDDFunctions$$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:1035)
               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$$anonfun$saveAsHadoopFile$1.apply$mcV$sp(PairRDDFunctions.scala:961)
               at org.apache.spark.rdd.PairRDDFunctions$$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 8 - Project Creation using Local Files

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

Create a New Project

Project Name

Local

Project Visibility

- Private** - Only added collaborators can view the project.
- Public** - All authenticated users can view this project.

Initial Setup

Blank

Template

Local

Git

Upload .zip or .tar.gz

Upload folder

Or Drag and Drop Files Here

Create Project

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:

The screenshot shows the Cloudera Data Science Workbench interface. On the left is a sidebar with icons for Account, Overview (selected), Jobs, Sessions, Files, Team, and Settings. The main area is titled "My Blank Project". It displays a "Jobs" section with a message: "This project has no jobs yet. Create a [new job](#) to document your analytics pipelines." Below that is a "Files" section with a table showing two items: "cdsw-demo-short" and "R". There are "Download", "New", and "Upload" buttons above the table. A checkbox menu on the right allows switching between "Show", "Hidden", and "Files". At the bottom, a message encourages adding a README.md file.

	Name	Size	Last Modified
<input type="checkbox"/>	cdsw-demo-short	-	yesterday
<input type="checkbox"/>	R	-	yesterday

This project doesn't contain a README.md file. Consider adding one that describes your 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 9 - 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 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 first project to get to this screen:

Name	Size	Last Modified
__pycache__	-	54 minutes ago
data	-	2 hours ago
R	-	39 minutes ago
1.python.py	2.97 kB	2 hours ago
2.pyspark.py	1.61 kB	2 hours ago
3.tensorflow.py	3.31 kB	2 hours ago
4_basket_analysis.r	1.29 kB	39 minutes ago
5_shiny.R	769 B	2 hours ago
groceries.csv	498.73 kB	2 hours ago
groceryrules.csv	44.83 kB	39 minutes ago
README.md	1.73 kB	2 hours ago
server.R	536 B	2 hours ago
spark-defaults.conf	74 B	2 hours ago
ui.R	689 B	2 hours ago
utils.py	1.07 kB	2 hours ago

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):

Create a Job

General

Name

my new job

Script

1_python.py



Engine Kernel

- Python 2
- Python 3
- Scala
- R

Schedule

Manual



Engine Profile

1 vCPU / 2 GiB Memory



GPUs

0 GPUs

Timeout In Minutes (optional) Kill on Timeout.Jobs exceeding timeout send warning email if notifications enabled.

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

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

The screenshot shows the Cloudera Data Science Workbench interface. On the left is a sidebar with icons for Account, Overview, Sessions, Experiments, Models, Files, Team, and Settings. The main area has tabs for 'test', 'CDSW Demo Test', and 'Jobs'. A search bar at the top right contains 'Project quick find' and a dropdown for 'test'. Below the tabs is a section titled 'Jobs' with a sub-section 'Job Dependencies for My New Job'. A button '+ Add Job Dependency' is visible. The main table lists one job:

Name	Runs / Failures	Duration	Status	Latest Run	Actions
My New Job	0 / 0	00:00	Not Yet Run	-	<button>Run</button>

At the bottom left, it says 'License Expires in 58 days'. At the bottom right, it shows '1.4.0.431664 (01ac70a)'.

- Here you can see that you have a job ('My New Job'). It's never been run, and it has no dependencies.
- 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

Name

Script

Engine Kernel

Python 2

Python 3

Scala

R

Schedule

Dependent

My New Job

Engine Profile

0.5 vCPU / 2 GiB Memory

GPUs

0 GPUs

Timeout In Minutes (optional) Kill on Timeout

Jobs exceeding timeout send warning email if notifications enabled.

[Set Environmental Variables](#)

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

The screenshot shows the Cloudera Data Science Workbench interface. On the left, a sidebar includes 'Overview', 'Jobs' (selected), 'Sessions', 'Files', 'Team', and 'Settings'. A message at the bottom left says 'License Expires in 59 days'. The main area has tabs for 'Overview', 'Jobs' (selected), and 'New Job'. Below the tabs, it says 'Job Dependencies for My New Job'. A horizontal dependency bar shows 'My New Job' (green circle) followed by a blue progress bar and 'Job 2' (orange circle). A table lists two jobs:

Name	Runs / Failures	Duration	Status	Latest Run	Actions
Job 2	0 / 0	00:00	Not Yet Run	-	<button>Run</button>
My New Job	0 / 0	00:00	Not Yet Run	-	<button>Run</button>

At the bottom right, it says '1.0.1 (052787a)'.

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. Let's 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_basket_analysis.r
Engine Kernel	R
Schedule	Dependent / My New Job

user01

Andre Molenaar CDSW Labs

Jobs

New Job

Create a Job

General

Name

R Job

Script

4_basket_analysis.r



Engine Kernel

- Python 2
- Python 3
- Scala
- R

Schedule

Dependent

my new job

Engine Profile

1 vCPU / 2 GiB Memory

Timeout In Minutes (optional) Kill on Timeout

Jobs exceeding timeout send warning email if notifications enabled.

[Set Environmental Variables](#)

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

The screenshot shows the Cloudera Data Science Workbench interface. The left sidebar has a dark theme with icons for Overview, Jobs (selected), Sessions, Files, and Team. The main content area shows 'Job Dependencies for My New Job' with three nodes: 'My New Job', 'Job 2', and 'R job'. A blue arrow points from 'My New Job' to 'Job 2', and another blue arrow points from 'My New Job' to '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	-	<button>Run</button>
Job 2	0 / 0	00:00	Not Yet Run	-	<button>Run</button>
My New Job	0 / 0	00:00	Not Yet Run	-	<button>Run</button>

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:

The screenshot shows the Cloudera Data Science Workbench interface. On the left is a sidebar with icons for Overview, Jobs (selected), Sessions, Files, Team, and Settings. A message at the bottom left says "License Expires in 59 days". The main area has tabs for Overview, Jobs (selected), and a "New Job" button. Below that is a "Job Dependencies for My New Job" section with a diagram showing "My New Job" connected to "Job 2" and "R job". Below the diagram is a table of completed jobs:

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

At the bottom right of the main area, it says "1.0.1 (052787a)".

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 10 - 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.

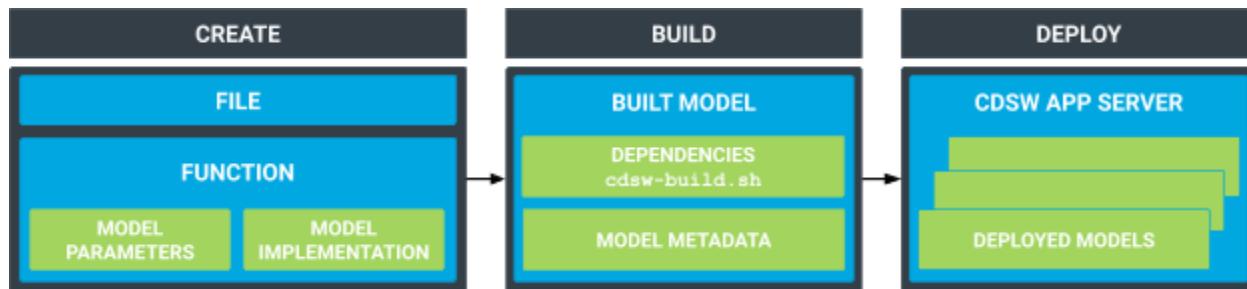
Once a model has been deployed, you then need to ensure that the devops team has a way to rollback the model to a previous version if needed. This means the data science team also needs a reliable way to retain history of the models they build and ensure that they can rebuild a specific version if needed. 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: Telco churn prediction 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/andremolenaar/experiments_models_tellarius.git

Create a New Project

Project Name

Telco Churn Prediction 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

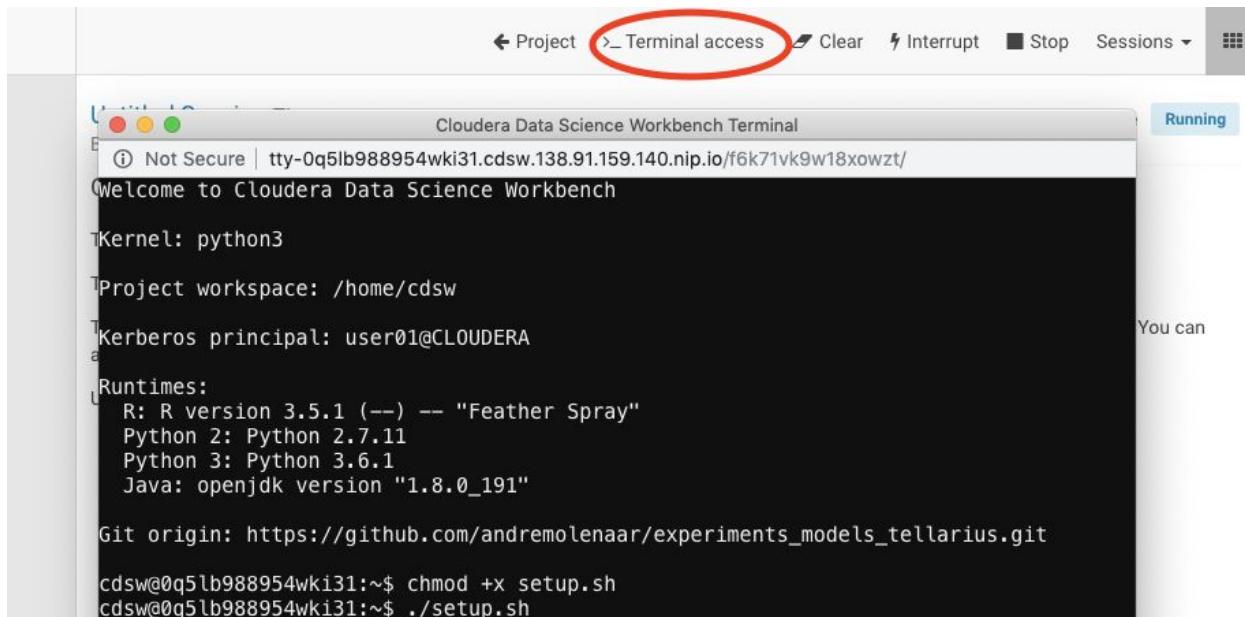
https://github.com/andremolenaar/experiments_models_tellarius.git

Create Project

Start a workbench with a Python 3 and 2 GB of memory.

When the workbench is available, open a terminal window and make the `setup.sh` program executable. Use the following command to do that:

```
chmod +x setup.sh
./setup.sh
./cdsw-build.sh
```



Step 2: Run the program `dsfortelco_sklearn.py`

When you run this program, it will create a predictive RandomForrest model. The Python predictive model object will be saved as a pickle file `models/sklearn_rf.pkl`

```
> ap = average_precision_score (y_true, y_scores)
> print(auroc, ap)
0.832449833751 0.6776227265

> cdsw.track_metric("auroc", auroc)
⚠ This output feature is only available within an Experiment.
  For more details, please see the documentation.

> cdsw.track_metric("ap", ap)
⚠ This output feature is only available within an Experiment.
  For more details, please see the documentation.

> pickle.dump(randF, open("models/sklearn_rf.pkl", "wb"))
> cdsw.track_file("/models/sklearn_rf.pkl")
```

Step 3: Examine the program predict_churn_sklearn.py

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

```
1 import pickle
2 import numpy as np
3
4 model = pickle.load(open("models/sklearn_rf.pkl", "rb"))
5
6 def predict(args):
7     account=np.array(args["feature"].split(",")).reshape(1,-1)
8     return {"result": model.predict(account)[0]}
9
10
```

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 step.

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.

The screenshot shows the 'Experiments and Models' page. On the left, there is a sidebar with several buttons: 'Overview' (selected), 'Sessions', 'Experiments', 'Models' (circled in red), and 'Jobs'. The main content area displays the title 'Experiments and Models' with a lock icon and '1 running'. Below this, there are sections for 'Models' (with a message 'This project has no models yet. Create a new model.'), 'Jobs' (with a message 'This project has no jobs yet. Create a new job to document your analytics pipelines.'), and a '...' button.

Select 'New Model', and 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"
 }
Kernel: Python 3
Engine: 1 vCPU / 2 GiB Memory
Replicas: 1

Create a Model

General

Name *

My Churn Prediction Model

Description *

My Churn Prediction Model

Build

File *

predict_churn_sklearn.py

**Function ***

predict

Example Input ⓘ

```
{  
    "feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4"  
}
```

Example Output ⓘ

```
{ "result": "value" }
```

Kernel

- Python 2
- Python 3
- R

Comment

Enter comment for this build

Deployment

Engine Profile

1 vCPU / 2 GiB Memory

**Replicas**

1

[Set Environmental Variables](#)

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

The screenshot shows the 'Models' section of the Cloudera Data Science Workbench interface. On the left sidebar, 'Experiments' is selected. The main table lists one model:

Model	Status	Replicas	CPU	Memory	Created By	Deployed By	Last Deployed	Actions
My Churn Prediction Model	Building	0 / 1	0	0 GiB	andre	andre	Oct 12, 2018, 9:13 AM	<button>Stop</button>

Step 3: Test the deployed model

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

The screenshot shows the 'Models' section again, but this time the 'My Churn Prediction Model' is listed with a status of 'Deployed'. A red circle highlights the 'Status' column for this row.

Model	Status	Replicas	CPU	Memory	Created By	Deployed By	Last Deployed	Actions
My Churn Prediction Model	Deployed	1 / 1	1	2 GiB	andre	andre	Oct 12, 2018, 9:15 AM	<button>Stop</button>

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

The screenshot shows the 'Overview' tab for the 'My Churn Prediction Model'. The 'Test Model' section contains an 'Input' field with the following JSON:

```
{
  "feature": [0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4]
}
```

A red circle highlights the 'Test' button at the bottom of the input field.

Model Details

- Model Id: 10
- Deployment: 49
- Build: 10
- Deployed By: andre
- Comment: Initial revision.
- Kernel: python2
- Engine Image: Base Image v5
- File: predict_churn_sklearn.py
- Function: predict

Model Resources

- Replicas: 1
- Total CPU: 1 vCPUs
- Total Memory: 2 GiB

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

Test Model**Input**

```
{  
  "feature": "0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4  
, 6, 3.43, 4"  
}
```

Test **Reset****Result****Status**

success

Response{
 "result": 1
}**Replica ID**

my-churn-prediction-model-10-49-599d6548d8-x9cmp

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 means that customer with these features is likely to churn.

Now, let's 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"  
}
```

Test Model**Input**

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

Test **Reset****Result****Status**

success

Response{
 "result": 0
}**Replica ID**

my-churn-prediction-model-10-49-599d6548d8-x9cmp

With these input parameters, the model returns 0, which means 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.

My Churn Prediction Model

Overview Deployments Builds Monitoring Settings

ID	Build	Status	Deployed At	Stopped At	Deployed By
49	1	Deployed	Oct 12, 2018, 9:15 AM		andre

Model
 Id: 10
 Name: My Churn Prediction Model
 Description: My Churn Prediction Model

Build
 Build Number: 1
 UUID: 467bc1db-8d7f-485e-adca-27ef6ee5e2b4
 File: predict_churn_sklearn.py
 Function: predict
 Kernel: python2
 Engine: Base Image v5

Deployment

Re-deploy This Build

When you get to the re-deployment page, you can increase the number of replicas.

My Churn Prediction Model

Deploy Existing Build of My Churn Prediction Model

Build

- File: predict_churn_sklearn.py
- Function: predict
- Example Input: ("feature":0.65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4)
- Example Output: 0
- Kernel: python2
- Comment: Initial revision.

Deployment

Engine Profile: 1 vCPU / 2 GiB Memory

Replicas: 1

Set Environmental Variables

Deploy Model Cancel

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

Now, navigate to the ‘Monitoring’ tab.

Replica	Status	Received	Processed	Success	Failure	Error	Busy	Not Ready	Restart
my-churn-prediction-model-10-49-599d6548d8-x9cmp	Ready	5	5 (100 %)	5 (100 %)	0	0	0	0	0

Streams: stdout stderr

```

2018-10-12 09:15:29.861 2018-10-12 07:15:29.861 INFO Model.Runtime Finish Model initialization
2018-10-12 09:15:29.124 2018-10-12 07:15:29.124 INFO Model.Runtime Start Model initialization
2018-10-12 09:15:29.065 2018-10-12 07:15:29.064 INFO Model.Runtime Start Python model runtime in 36

```

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.

Name: My Churn Prediction Model

Description: My Churn Prediction Model

Access Key: me4h518qvug7ywd3xbw21ew60w1fx

Update

Danger Zone: Warning! Deleting a model is irreversible. All model builds and deployments history will be deleted.

Delete Model

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 command line.

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.

My Churn Prediction Model

Overview Deployments Builds Monitoring Settings

Description My Churn Prediction Model

Sample Code Shell Python R

curl -H "Content-Type: application/json" -X POST http://cdsw.52.211.207.216.nip.io/api/altus-ds-1/models/call-model -d '{"accessKey":"me4h5i8qvug7yuwd3xzbw21ew6@wj1fx","request":{"feature":[0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4]}'

Test Model

Input

```
[{"feature": [0, 65, 0, 137, 21.95, 83, 19.42, 111, 9.4, 6, 3.43, 4]}
```

Test Reset

Model Details

Model Id	10
Deployment	49
Build	10
Deployed By	andre
Comment	Initial revision.
Kernel	python2
Engine Image	Base Image v5
File	predict_churn_sklearn.py
Function	predict

Model Resources

Replicas	1
Total CPU	1 vCPUs
Total Memory	2 GiB

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.

File Edit View Navigate Run Project Terminal access Clear ⌘ Interrupt Stop Sessions

predict_churn_sklearn.py

1 import pickle
2 import numpy as np
3
4 model = pickle.load(open("models/sklearn_rf.pkl", "rb"))
5
6 def predict(args):
7 account=np.array(args["feature"].split(",")).reshape(1,-1)
8 return {"result": model.predict(account)[0]}
9
10

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:~\$

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

```
Welcome to Cloudera Data Science Workbench

Kernel: python2

Project workspace: /home/cds

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://cdsw.52.211.207.216.nip.io/api/altus-ds-1/models/call-model -d '{"accessKey":"me4h5i8qvug7yuwd3xzbw21ew60wjlf","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, free up some resources for other people and new projects. So stop your workbench session. And from the Models page, also stop your deployed model.

The screenshot shows the 'Models' section of the Cloudera Data Science Workbench interface. On the left, there's a sidebar with 'Overview', 'Sessions', and 'Experiments'. The main area has tabs for 'Experiments and Models' (which is selected) and 'Models'. A search bar at the top right says 'Project quick find' with a user icon 'andre'. Below the tabs is a 'New Model' button. The 'Models' table lists one entry:

Model	Status	Replicas	CPU	Memory	Created By	Deployed By	Last Deployed	Actions
My Churn Prediction Model	Deployed	1 / 1	1	2 GiB	andre	andre	Oct 12, 2018, 9:15 AM	<button>Stop</button>

The 'Actions' column for the first row contains a 'Stop' button, which is highlighted with a red oval.

Lab 12 - Face Recognition with Python: Where is Filippo?

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 from 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/andremolenaar/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/cds_w_user_guide.html

Troubleshooting Guide

https://www.cloudera.com/documentation/data-science-workbench/latest/topics/cds_w_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 beginning 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>