ML for SparkR: Just Add Water

Vincent D. Warmerdam, GoDataDriven, @fishnets88, koaning.io

Spark and R: There's Joy Now

Vincent D. Warmerdam, GoDataDriven, @fishnets88, koaning.io

Today

— I'll explain what spark is and why to care
— I'll quickly show the `sparklyr` setup.
— I'll demo a sessionizing usecase (WoW).
— I'll demo how to do H2o on Spark from R.
— I'll explain the why and the benefits.
— I'll hint at a bright future.

What do you do when you want to blow up a building?

Use a bomb.

What do you do when you want to blow up a building?

Use a bomb.

What do you do when you want to blow up a bigger building?

Use a bigger, way more expensive, bomb

What do you do when you want to blow up a building?

Use a bomb.

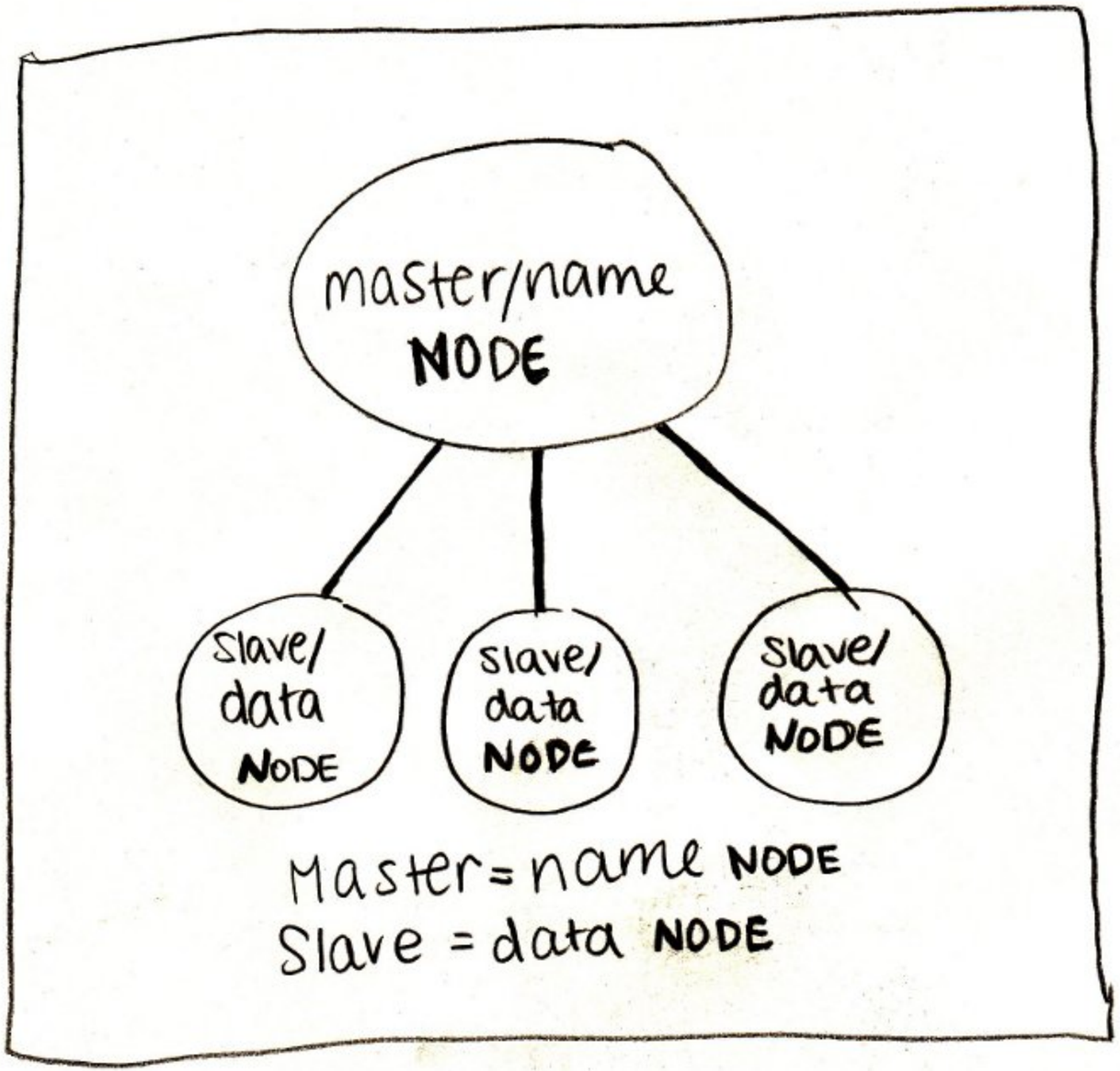
What do you do when you want to blow up a bigger building?

~~**Use a bigger, way more expensive, bomb**~~

Use many small ones.

Distributed computation

- connect machines
- store the data on multiple machine (memory)
- compute word in parallel
- bring code to data
- not the other way around



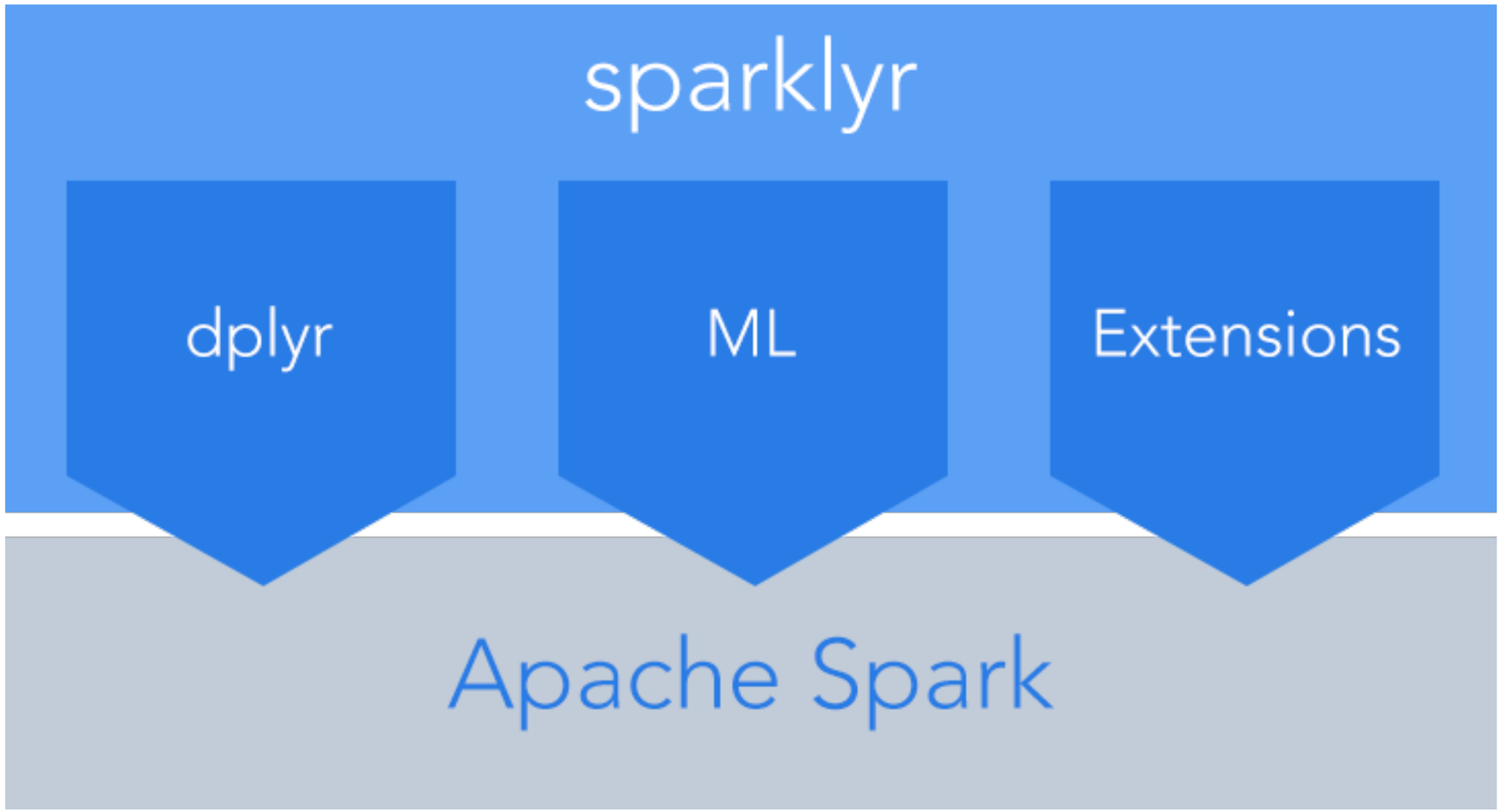
Spark is parallel

Even locally: perfect for the >2GB blobs!

```
Processes: 228 total, 3 running, 3 stuck, 222 sleeping, 1345 threads                 21:04:
Load Avg: 3.24, 2.29, 1.87  CPU usage: 96.94% user, 2.76% sys, 0.29% idle
SharedLibs: 90M resident, 0B data, 14M linkedit.
MemRegions: 83992 total, 7019M resident, 76M private, 13G shared.
PhysMem: 13G used (2546M wired), 632M unused.
VM: 608G vsize, 1312M framework vsize, 3013284(0) swapins, 3316559(0) swapouts.
Networks: packets: 29603472/34G in, 11073080/2276M out.
Disks: 3185216/85G read, 3042468/109G written.

PID    COMMAND      %CPU      TIME     #TH    #WQ  #PORT #MREGS MEM    RPRVT  PURG   CMPRS
48026  java         775.5     11:21.01 95/8   0    236-  2339   941M-  947M-  0B     138M
36104  top          18.9      42:47.01 1/1    0    45    56     7904K  7748K  0B     172K
118    WindowServer 2.4       02:45:02 4      0    732   6561-  581M-  120M-  29M    242M
```

What so special now?



Example: WoW Churn!

Spark + R = Joy

Installation is super easy.

```
devtools::install_github("rstudio/sparklyr")
spark_install(version = "1.6.2")
```

Don't underestimate how useful this can be for a local setup. My mac has 4-8 cores available for spark and 16 Gb of memory.

Csv to Parquet

You can read .csv or .parquet files.

```
ddf_pq <- spark_read_parquet(sc,
    name="main_pq",
    "<path>/wowah_data.parquet"
)
```

You can give the dataframe a table-name, which can be seen from the Spark UI.

Example Log Bit

	char	level	race	charclass	zone	guild	date	ts
1	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:02:20
2	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:12:07
3	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:22:40
4	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:32:29
5	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:42:18
6	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:52:47
7	9	70	Orc	Hunter	Ashenvale	79	2008-01-01	2008-01-01 13:02:29
8	9	70	Orc	Hunter	Ashenvale	79	2008-01-01	2008-01-01 13:12:18
9	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 13:22:44
10	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 13:32:32
11	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:02:31
12	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:12:18
13	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:22:44
14	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:32:32
15	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:42:20
16	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:52:08
17	9	70	Orc	Hunter	Shattrath City	79	2008-01-01	2008-01-01 17:02:43

Example Log Bit

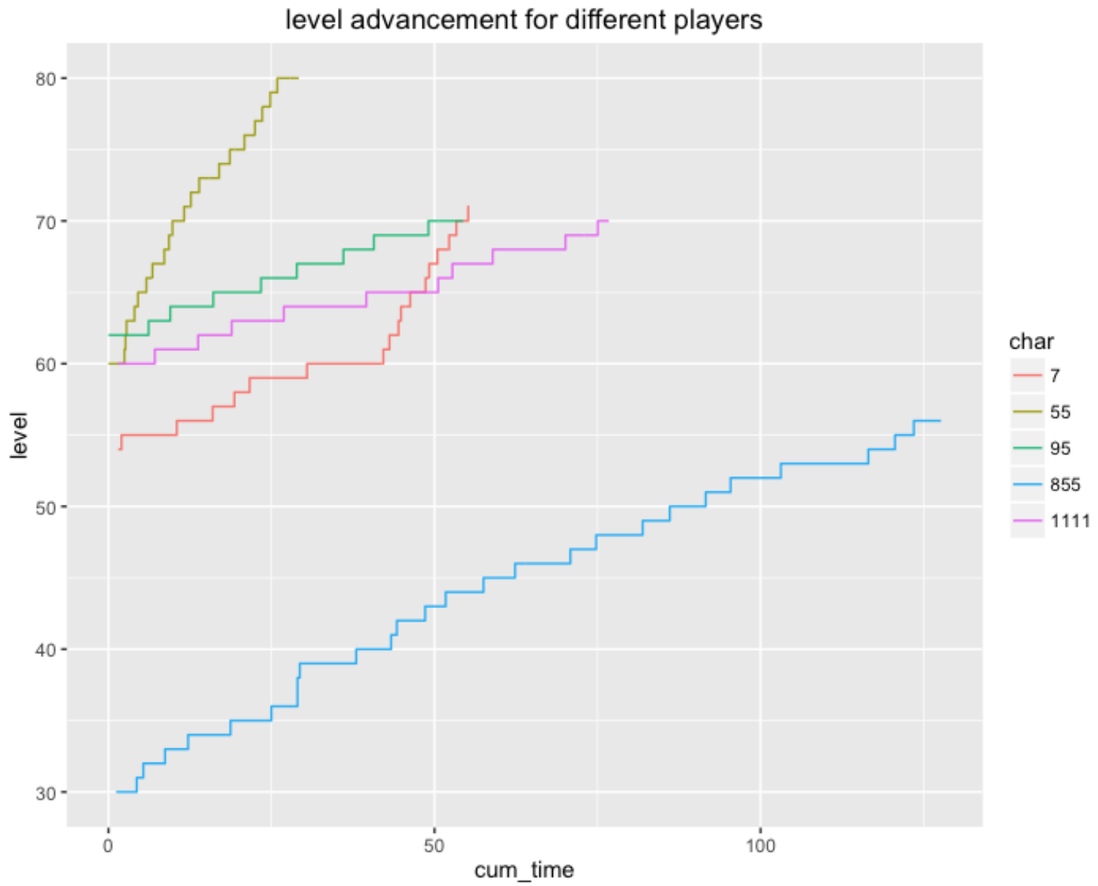
	date	ts	diff_mins	new_session	session_id
1	2008-01-01	2008-01-01 12:02:20	NA	TRUE	1
2	2008-01-01	2008-01-01 12:12:07	10	FALSE	1
3	2008-01-01	2008-01-01 12:22:40	10	FALSE	1
4	2008-01-01	2008-01-01 12:32:29	10	FALSE	1
5	2008-01-01	2008-01-01 12:42:18	10	FALSE	1
6	2008-01-01	2008-01-01 12:52:47	10	FALSE	1
7	2008-01-01	2008-01-01 13:02:29	10	FALSE	1
8	2008-01-01	2008-01-01 13:12:18	10	FALSE	1
9	2008-01-01	2008-01-01 13:22:44	10	FALSE	1
10	2008-01-01	2008-01-01 13:32:32	10	FALSE	1
11	2008-01-01	2008-01-01 16:02:31	10	FALSE	1
12	2008-01-01	2008-01-01 16:12:18	120	TRUE	2
13	2008-01-01	2008-01-01 16:22:44	10	FALSE	2
14	2008-01-01	2008-01-01 16:32:32	10	FALSE	2
15	2008-01-01	2008-01-01 16:42:20	10	FALSE	2
16	2008-01-01	2008-01-01 16:52:08	10	FALSE	2
17	2008-01-01	2008-01-01 17:02:43	10	FALSE	2

The code

Window functions FTW!

```
sessionized_df <- df %>%
  arrange(char, ts) %>%
  group_by(date, char) %>%
  mutate(time_since = ts - lag(ts),
         timegap = ifelse(is.na(time_since), TRUE,  time_since > 700)) %>%
  ungroup() %>%
  arrange(ts) %>%
  group_by(char) %>%
  mutate(session_id = cumsum(as.numeric(timegap)))
```

GGplot is one `ddf %>% collect()` away



Sparklyr: ML

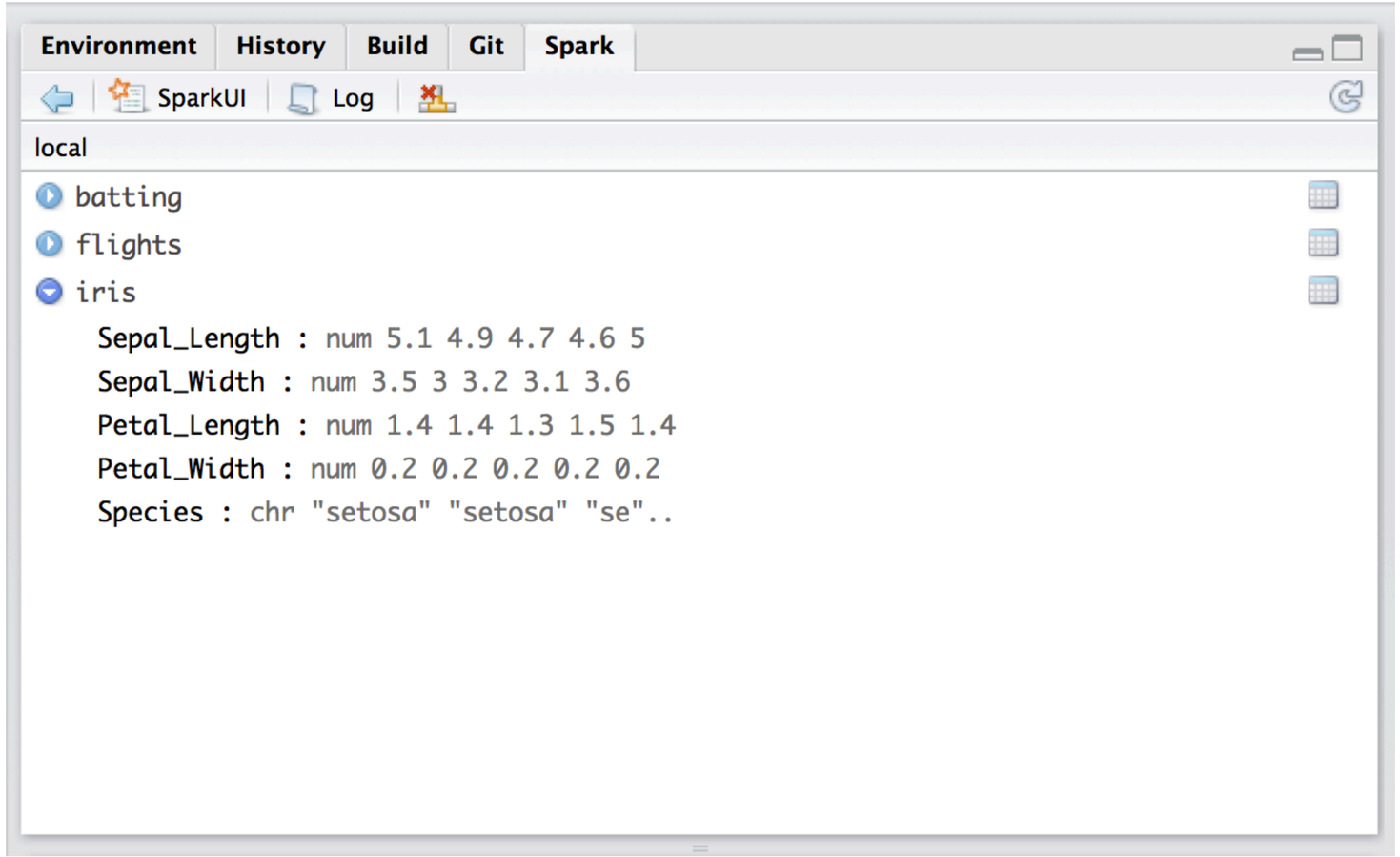
A lot of popular ML libraries are now available.

Much more than in SparkR.

- **kmeans**
- **trees/forests/naive bayes**
- **logistic regression**
- **basic feed forward neural network**

Plenty of examples listed in the docs.

SparklyR heart Rstudio



But you cannot do *everything* with just base Spark.

But you cannot do *everything* with just base Spark.

Sometimes it's more about the availability of a certain model because a lot of other things can be done with SQL (this is even true for SparkR).

For this next bit I'll demo H2o; which is a JVM based machine learning library that plays well with Spark.

Note that you do not need Hadoop/Spark to use H2o. You especially don't need a cluster of machines, even locally there's a speedup.

H2o for R

Step by Step

Copy code from blogpost

Step 1

Download Spark and Sparkling Water.

Step 2

Start the Sparkling Shell

```
export SPARK_HOME="/<path>/spark-1.6.0-bin-hadoop1"
export MASTER="local[*]"
./bin/sparkling-shell
```

This shell can start up Spark and you can easily import an H2o context from here.

Step 3

Import `H2o` from this sparkling-shell.

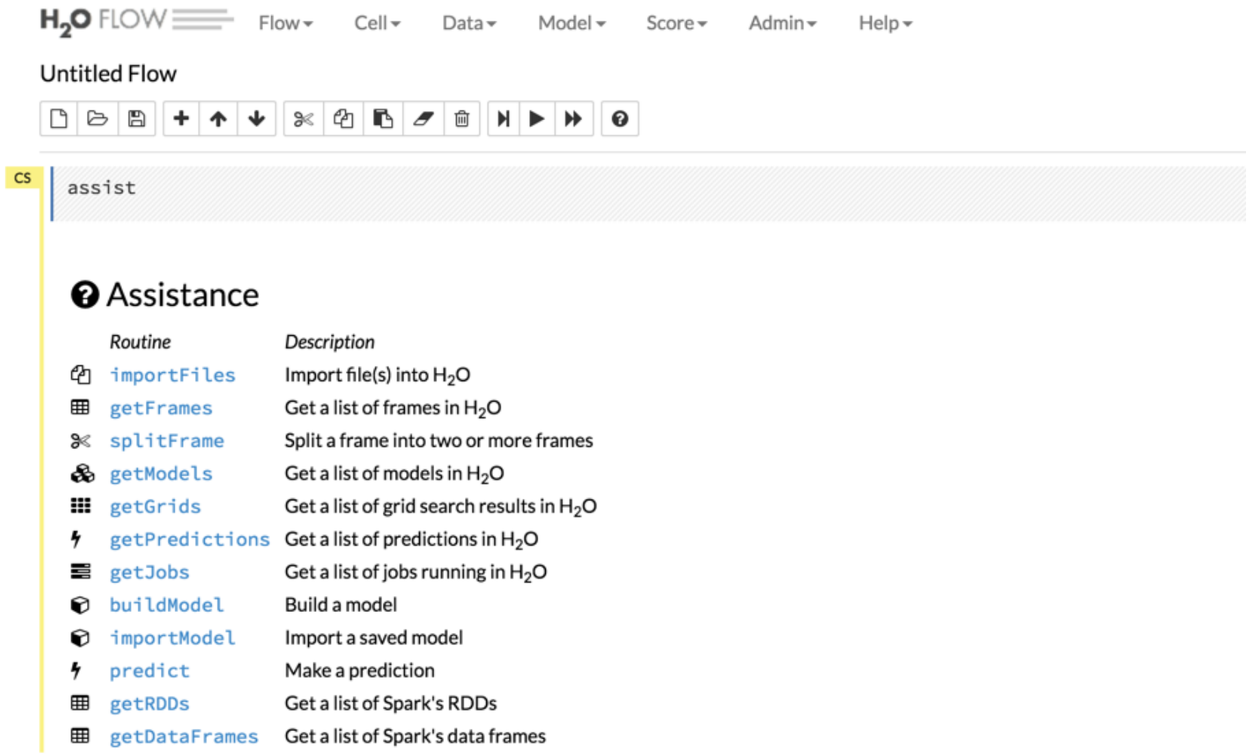
```
import org.apache.spark.h2o._
val h2oContext = H2OContext.getOrCreate(sc)
```

H2o can now make use of Spark resources.

Step 3

With this connection made, H2o can make use of Spark as a compute engine as well as access its dataframes. It should also prompt you with an ip adress and a port number. You can visit this endpoint in the browser to see the h2o notebook.

Step 3, confirm the UI.



Step 3, continued

What we'll do next:

- generate a dataframe in Sparkling-Shell
- pass it to H2o
- connect H2o from R to the same instance
- apply machine learning libraries from Rstudio

Step 4

```
import org.apache.spark.mllib.random.{RandomRDDs => r}
import org.apache.spark.sql.{functions => sf}

def gen_blob() = {
  if(scala.util.Random.nextDouble() > 0.5){
    (0,
     scala.util.Random.nextDouble()*2,
     scala.util.Random.nextDouble(),
     scala.util.Random.nextDouble()*2)
  }else{
    (1,
     1 + scala.util.Random.nextDouble(),
     1 + scala.util.Random.nextDouble()*2,
     1 + scala.util.Random.nextDouble())
  }
}
```

Step 4

With that function, let's actually create a DataFrame.

```
val n = 10000
val rdd = sc.parallelize(1 to n).map(x => gen_blob())

val ddf = rdd.toDF()
val hdf = h2oContext.asH2OFrame(ddf, frameName = "foobar")
```

This last step is crucial, this H2o frame can be accessed from Rstudio.

Step 5

Start Rstudio with the following libraries.

```
library(dplyr)
library(ggplot2)
library(h2o)
```

Install these packages if you didn't have them before.

Step 5

Use the same <localhost>:<port> combination as the one that was prompted from the sparkling-shell.

```
client <- h2o.init(ip = 'localhost', port=54321)
h2o.ls()
rddf <- h2o.getFrame("foobar")
```

Note that we're getting the frame that we've created before.

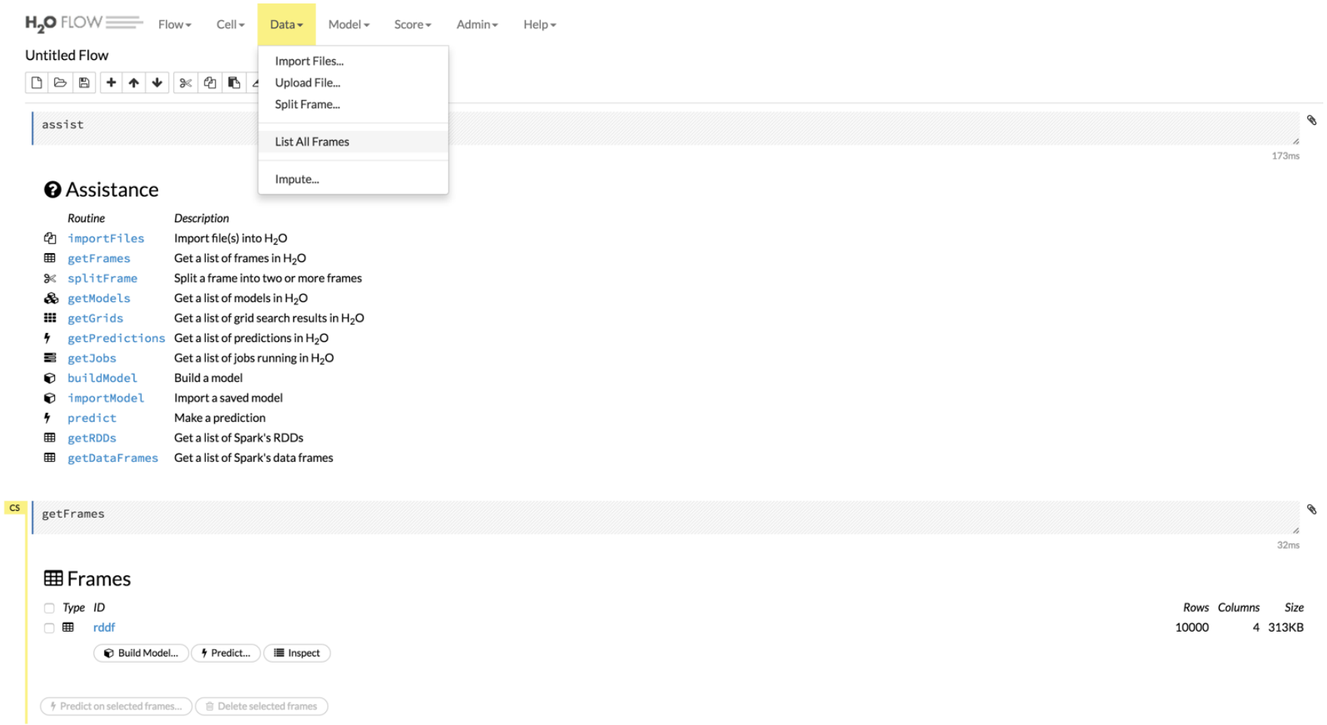
Step 5

This `rddf` is not a Spark DataFrame or a normal R DataFrame.

```
> typeof(rddf)
[1] "environment"
```

Step 5

You can also reach this H2oFrame from the UI.



Step 5

What can H2o offer us now next to Sparklyr?

- **wider range of ML models**
- **decent tools for hyperparameter tuning**
- **output of model is an actual downloadable .jar**

Step 6

Let's apply an autoencoder from H2o.

```
mod_nn <- h2o.deeplearning(
  x = c("_2", "_3", "_4"),
  training_frame = rddf,
  hidden = c(4,2),
  epochs = 100,
  activation = 'Tanh',
  autoencoder = TRUE
)
```

Step 6

Apply this model and visualize.

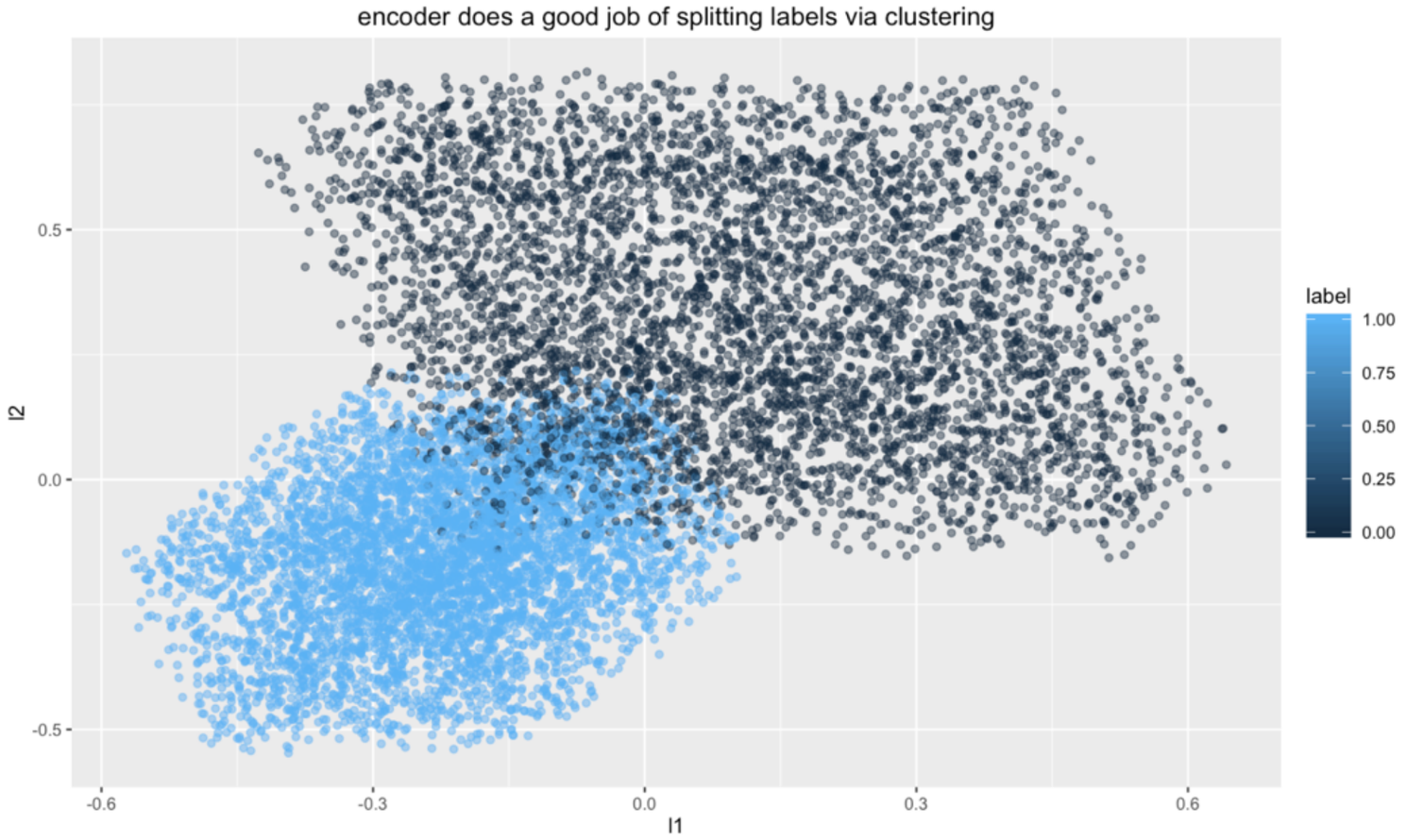
```
features <- h2o.deepfeatures(mod_nn, rddf, layer=2)

pltr_nn <- features %>%
  as.data.frame %>%
  cbind(rddf %>% as.data.frame %>% .[1])

colnames(pltr_nn) <- c("l1", "l2", 'label')

ggplot() +
  geom_point(data=pltr_nn, aes(l1, l2, colour = label), alpha = 0.5) +
  ggtitle("encoder does a good job of splitting labels via clustering")
```

Step 6: Output



Step 6: Gradient Boosted Search

```
hyper_params = list(ntrees = c(100, 1000),
                    max_depth = 1:4,
                    learn_rate = seq(0.001,0.01),
                    sample_rate = seq(0.3,1))

search_criteria = list(strategy = "RandomDiscrete",
                       max_runtime_secs = 600,
                       max_models = 100,
                       stopping_metric = "AUTO",
                       stopping_tolerance = 0.00001,
                       stopping_rounds = 5,
                       seed = 123456)
```

```
gbm_grid <- h2o.grid("gbm", grid_id = "mygrid",
                     x = c("_2", "_3", "_4"),
                     y = c("_1"),
                     training_frame = rddf, nfolds = 5,
                     distribution="gaussian",
                     score_tree_interval = 100,
                     seed = 123456,
                     hyper_params = hyper_params,
                     search_criteria = search_criteria)

gbm_sorted_grid <- h2o.getGrid(grid_id = "mygrid", sort_by = "mse")
```

You can view the results via;

```
gbm_sorted_grid@summary_table %>% View
```

Select the best model

```
best_model <- h2o.getModel(gbm_sorted_grid@model_ids[[1]])
```

POJO

Making Friends with Engineers

```
h2o.download_pojo(
    best_model,
    path = "/tmp/",
    getjar=TRUE
)
```

The Future?

In practice, you want the engineer and the scientist to be friends and this h2o + sparklyr stack really makes a lot of sense for production.

You can google around and get an impression that I'm not the only person who is considering this path to be valueable.

The Future?

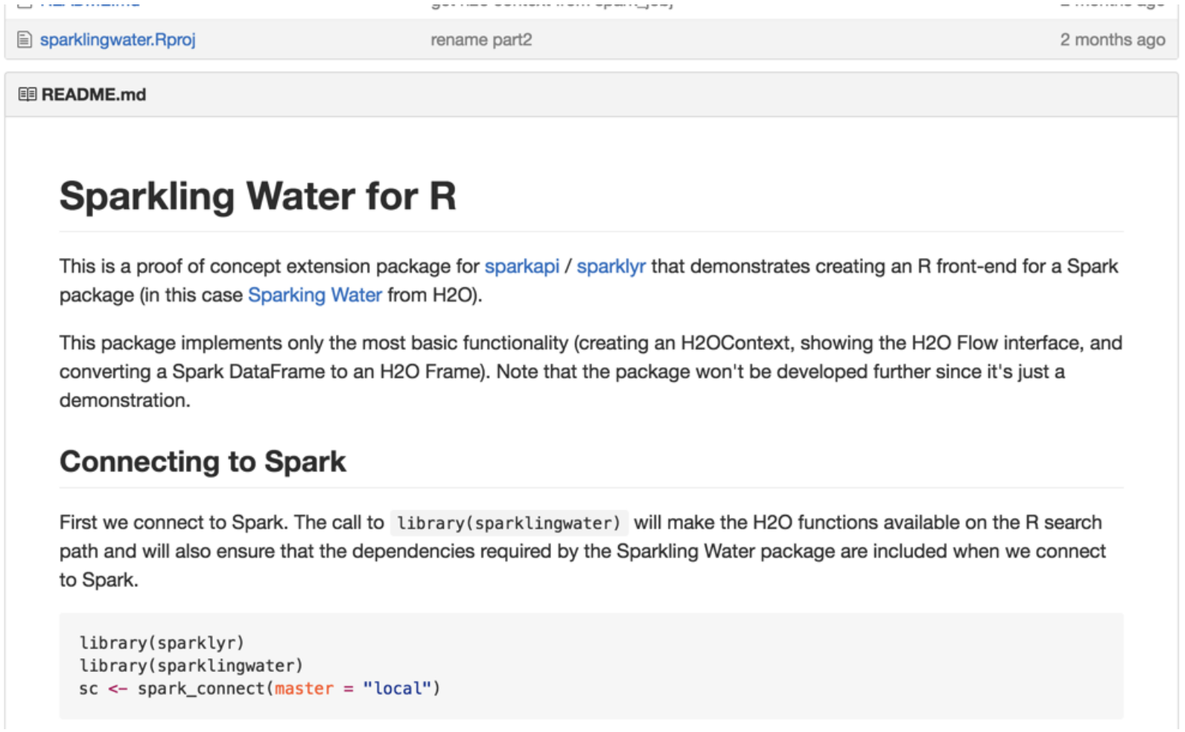
Github: `jjallaire/sparklingwater`.

```
library(sparklyr)
library(sparklingwater)
sc <- spark_connect(master = "local")

h2o_context(sc)
mtcars_tbl <- copy_to(sc, mtcars, overwrite = TRUE)
mtcars_hf <- h2o_frame(mtcars_tbl)
```

The Future?

Github: `jjallaire/sparklingwater`.



sparklingwater.Rproj — rename part2 — 2 months ago

README.md

Sparkling Water for R

This is a proof of concept extension package for sparkapi / sparklyr that demonstrates creating an R front-end for a Spark package (in this case Sparking Water from H2O).

This package implements only the most basic functionality (creating an H2OContext, showing the H2O Flow interface, and converting a Spark DataFrame to an H2O Frame). Note that the package won't be developed further since it's just a demonstration.

Connecting to Spark

First we connect to Spark. The call to `library(sparklingwater)` will make the H2O functions available on the R search path and will also ensure that the dependencies required by the Sparkling Water package are included when we connect to Spark.

```
library(sparklyr)
library(sparklingwater)
sc <- spark_connect(master = "local")
```

Conclusion

Sparklyr is a clear win, H2o can help fill in some gaps. I expect the two to become better at talking to eachother in the future.

Don't give me any credit as I'm just a user. Be sure to high-five Rstudio today.

I'll be around the conf, AMA!

Thanks for listening! Code is on the blog.