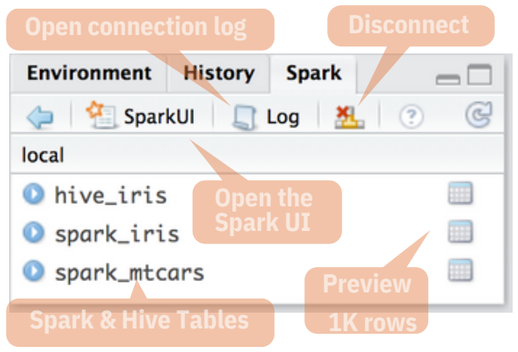
# Data Science in Spark with Sparklyr : :

## IntroData Science Toolchain with Spark + sparklyr

sparklyr is an R interface for Apache Spark™, it provides a complete **dplyr** backend and the option to query directly using **Spark SQL** statement. With sparklyr, you can orchestrate distributed machine learning using either **Spark's MLlib** or **H2O** Sparkling Water.

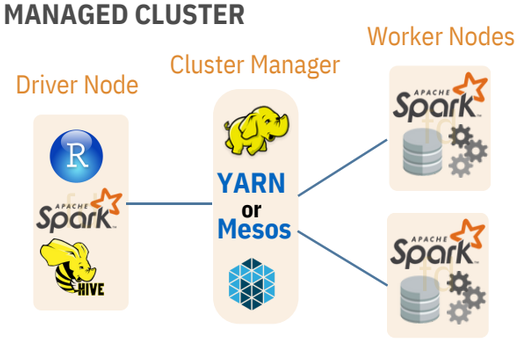
Starting with **version 1.044, RStudio Desktop, Server and Pro include integrated support for the sparklyr package**. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

**RStudio Integrates with sparklyr**

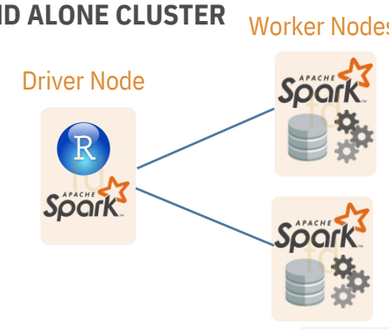


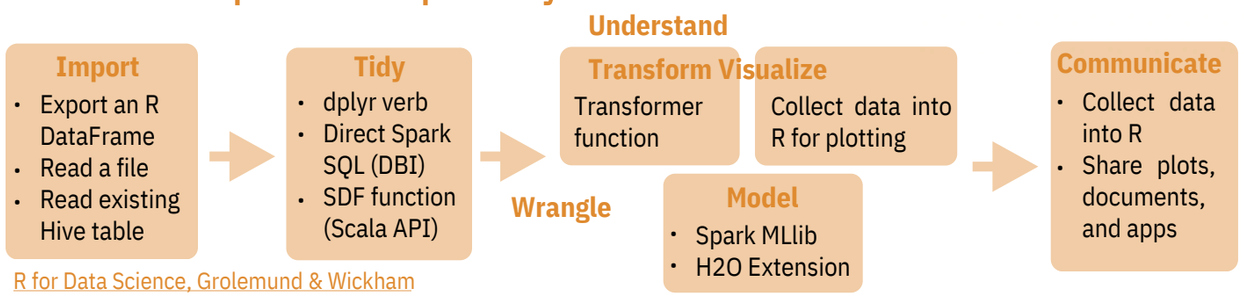

## Cluster Deployment

**MANAGED CLUSTER**




**STAND ALONE CLUSTER**




R for Data Science, Grolemund & Wickham

## Getting Started

**LOCAL MODE** (No cluster required)

1.Install a local version of Spark:
**spark_install ("2.0.1")**

2.Open a connection

**sc <- spark_connect (master = "local")**

**ON A MESOS MANAGED CLUSTER**

1.Install RStudio Server or Pro on one of the existing nodes

2.Locate path to the cluster's Spark directory

3.Open a connection
**spark_connect(master="[mesos URL]", version = "1.6.2", spark_home = [Cluster's Spark path])**

**USING LIVY** (Experimental)

1.The Livy REST application should be running on the cluster

2.Connect to the cluster

**sc <- spark_connect(method = "livy", master = "http://host:port")**

**ON A YARN MANAGED CLUSTER**

1.Install RStudio Server or RStudio Pro on one of the existing nodes, preferably an edge node

2.Locate path to the cluster's Spark Home Directory, it normally is "/usr/lib/spark"

3.Open a connection

**spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])**

**ON A SPARK STANDALONE CLUSTER**

1.Install RStudio Server or RStudio Pro on one of the existing nodes or a server in the same LAN

2.Install a local version of Spark:
**spark_install (version = "2.0.1")**

3.Open a connection

**spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())**

## Tuning Spark

**EXAMPLE CONFIGURATION**

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect (master="yarn-client",
config = config, version = "2.0.1")
```

**IMPORTANT TUNING PARAMETERS** with defaults

- spark.yarn.am.cores
- spark.yarn.am.memory *512m*
- spark.network.timeout *120s*
- spark.executor.memory *1g*
- spark.executor.cores *1*
- spark.executor.instances
- spark.executor.extraJavaOptions
- spark.executor.heartbeatInterval *10s*
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

## Using sparklyr

A brief example of a data analysis using Apache Spark, R and sparklyr in local mode

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyr);
set.seed(100)
spark_install("2.0.1")
sc <- spark_connect(master = "local")

import_iris <- copy_to(sc, iris, "spark_iris",
overwrite = TRUE)

partition_iris <- sdf_partition(
 import_iris,training=0.5, testing=0.5)

sdf_register(partition_iris,
c("spark_iris_training","spark_iris_test"))

tidy_iris <- tbl(sc,"spark_iris_training") %>%
select(Species, Petal_Length, Petal_Width)

model_iris <- tidy_iris %>%
 ml_decision_tree(response="Species",
features=c("Petal_Length","Petal_Width"))

test_iris <- tbl(sc,"spark_iris_test")

pred_iris <- sdf_predict(
model_iris, test_iris) %>%
collect

pred_iris %>%
inner_join(data.frame(prediction=0:2,
lab=model_iris$model.parameters$labels)) %>%
ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
geom_point()
spark_disconnect(sc)
```

# Reactivity

### COPY A DATA FRAME INTO SPARK

**sdf_copy_to**(sc, iris, "spark_iris")

**sdf_copy_to(**sc, x, name, memory, repartition, over write**)**

### IMPORT INTO SPARK FROM A FILE

**Arguments that apply to all functions:**
**sc, name, path, options = list(), repartition = 0, memory = TRUE, overwrite = TRUE**

CSV **spark_read_csv(** header = TRUE, columns = NULL, infer_schema = TRUE, delimiter = ",", quote = "\"", escape = "\\", charset = "UTF-8", null_value = NULL**)**

JSON **spark_read_json()**

PARQUET **spark_read_parquet()**

### SPARK SQL COMMANDS

DBI::**dbWriteTable**(sc, "spark_iris", iris)

DBI::**dbWriteTable(**conn, name, value**)**

### FROM A TABLE IN HIVE

my_var <- **tbl_cache**(sc, name= "hive_iris")

**tbl_cache(**sc, name, force = TRUE**)**
Loads the table into memory

my_var <- **dplyr::tbl**(sc, name= "hive_iris")

dplyr::**tbl(**scr, ...**)**
Creates a reference to the table without loading it into memory

# Wrangle

### SPARK SQL VIA DPLYR VERBS

Translates into Spark SQL statements

```
my_table <- my_var %>%
    filter(Species=="setosa") %>%
sample_n(10)
```

### DIRECT SPARK SQL COMMANDS

my_table <- DBI::**dbGetQuery**( sc , "SELECT * FROM iris LIMIT 10")

DBI::**dbGetQuery(**conn, statement**)**

### SCALA API VIA SDF FUNCTIONS

**sdf_mutate**(.data**)**
Works like dplyr mutate function

**sdf_partition(**x, ..., weights = NULL, seed = sample (.Machine$integer.max, 1**)**)
**sdf_partition**(x, training = 0.5, test = 0.5)

**sdf_register(**x, name = NULL**)**
Gives a Spark DataFrame a table name

**sdf_sample(**x, fraction = 1, replacement = TRUE, seed = NULL**)**

**sdf_sort(**x, columns**)**
Sorts by >=1 columns in ascending order

**sdf_with_unique_id(**x, id = "id"**)**

**sdf_predict()** object, newdata
Spark DataFrame with predicted values

### ML TRANSFORMERS

**ft_binarizer**(my_table,input.col="Petal_Length", output.col="petal_large", threshold=1.2)

**Arguments that apply to all functions:**
**x, input.col = NULL, output.col = NULL**

**ft_binarizer(**threshold = 0.5**)**
Assigned values based on threshold

**ft_bucketizer(**splits**)**
Numeric column to discretized column

**ft_discrete_cosine_transform(**inverse = FALSE**)**
Time domain to frequency domain

**ft_elementwise_product(**scaling.col**)**
Element-wise product between 2 cols

**ft_index_to_string()**
Index labels back to label as strings

**ft_one_hot_encoder()**
Continuous to binary vectors

**ft_quantile_discretizer(**n.buckets=5L**)**
Continuous to binned categorical values

**ft_sql_transformer(**sql**)**

**ft_string_indexer(** params = NULL**)**
Column of labels into a column of label indices.

**ft_vector_assembler()**
Combine vectors into single row-vector

# Visualize & Communicate

### DOWNLOAD DATA TO R MEMORY

r_table <- **collect**(my_table)
plot(Petal_Width~Petal_Length, data=r_table)

dplyr ::**collect(**x**)**
Download a Spark DataFrame to an R DataFrame

**sdf_read_column(**x, column**)**
Returns contents of a single column to R

### SAVE FROM SPARK TO FILE SYSTEM

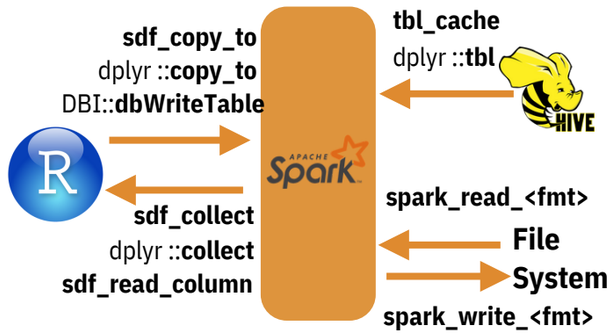
**Arguments that apply to all functions: x, path**

CSV **spark_read_csv(** header = TRUE, delimiter = ",", quote = "\"", escape = "\\", charset = "UTF-8", null_value = NULL**)**

JSON **spark_read_json(**mode = NULL**)**

PARQUET **spark_read_parquet(**mode = NULL**)**

# Reading & Writing from Apache Spark




# Extensions

**Create an R package that calls the full Spark API & provide interfaces to Spark packages.**

### CORE TYPES

**spark_connection()**
Connection between R and the Spark shell process
**spark_jobj()**
Instance of a remote Spark object
**spark_dataframe()**
Instance of a remote Spark DataFrame object

### CALL SPARK FROM R

**in v o k e ( )** Call a method on a Java object
**in v o k e _ n e w ( )** Create a new object by invoking a constructor
**in v o k e _ s t a t i c ( )** Call a static method on an object

### MACHINE LEARNING EXTENSIONS

**ml_create_dummy_variables() ml_options()**
**ml_prepare_dataframe() ml_model()**
**ml_prepare_response_features_intercept()**

# Model (MLlib)

**ml_decision_tree(**
response = "Species", features = c("Petal_Length" , "Petal_Width"))

**ml_als_factorization(**x, user.column = "user", rating.column = "rating", item.column = "item", rank = 10L, regularization.parameter = 0.1, iter.max = 10L, **)** ml.options = ml_options()

**ml_decision_tree(**x, response, features, max.bins = 32L, max.depth = 5L, type = c("auto", "regression", "classification"), ml.options = ml_options() **)** Same options for: **ml_gradient_boosted_trees**

**ml_generalized_linear_regression(**
x, response, features, intercept = TRUE, family = gaussian(link = "identity"), iter.max = 100L, ml.options = ml_options()**)**

**ml_kmeans(**
x, centers, iter.max = 100, features = dplyr::tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options()**)**

**ml_lda(**
x, features = dplyr::tbl_vars(x), k = length(features), alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options() **)**

**ml_linear_regression(**
x, response, features, intercept = TRUE, alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options() **)**
Same options for: **ml_logistic_regression**

**ml_multilayer_perceptron(**x, response, features, layers, iter.max = 100, seed = sample(.Machine$integer.max, 1), ml.options = ml_options()**)**

**ml_naive_bayes(**x, response, features, lambda = 0, ml.options = ml_options()**)**

**ml_one_vs_rest(**
x, classifier, response, features, ml.options = ml_options() **)**

**ml_pca(**
x, features = dplyr::tbl_vars(x), ml.options = ml_options()**)**

**ml_random_forest(**
x, response, features, max.bins = 32L, max.depth = 5L, num.trees = 20L, type = c("auto", "regression", "classification"), ml.options = ml_options() **)**

**ml_survival_regression(**
x, response, features, intercept = TRUE,censor = "censor", iter.max = 100L, ml.options = ml_options()**)**

**ml_binar y_classification_eval(**predicted_tbl_spark, label, score, metric = "areaUnderROC" **)**

**ml_classification_eval(**
predicted_tbl_spark, label, predicted_lbl, metric = "f1"**)**

**ml_tree_feature_importance(** sc, model**)**

sparklyr is an R interface for Apache Spark