Spark & R with sparklyr

Doug Ashton – Senior Data Scientist
Owen Jones – Data Scientist

#ODSC

Agenda

- R and the Big Data Landscape
- The dplyr package for Spark
- Machine Learning with sparklyr

About Mango

www.mango-solutions.com
@MangoTheCat

MANGO SOLUTIONS

R and the Big Data Landscape

R & Big Data

What is R?

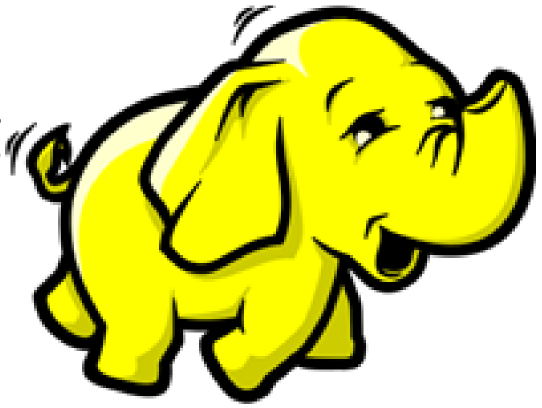
- Fast: In memory calculations
- Easy:
  - Built for Data Science
  - Vibrant community building tools
  - Flexible, fast to develop

- But …
  - Single Threaded
  - Poor scaling

The Yellow Elephant

Apache Hadoop

- HDFS (Hadoop Distributed File system)
  - Distributed, scalable file system
  - Store (and access) large files across machines
  - Written in Java
- MadReduce
  - Programming Model for processing parallelizable problems
  - Map > Shuffle > Reduce steps
  - Slow

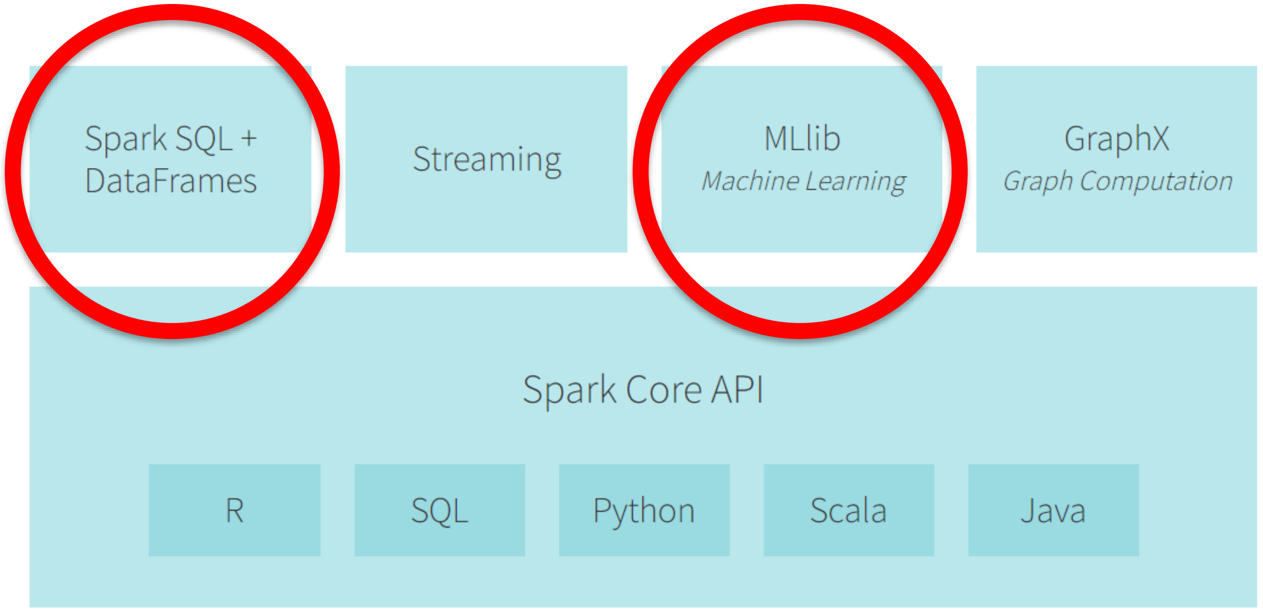


Spark

Apache Spark

- Fast: In-memory processing
- Easy: High level API designed for Data Science
- Scales: Distributed computation

Spark Ecosystem



Spark and R

Spark and R

- Originally supported Scala, Java and Python
- SparkR integrated into Spark as of v1.4
- Lets you create and work with MASSIVE data frames over a cluster of machines
- https://spark.apache.org/docs/latest/sparkr.html

SparkR

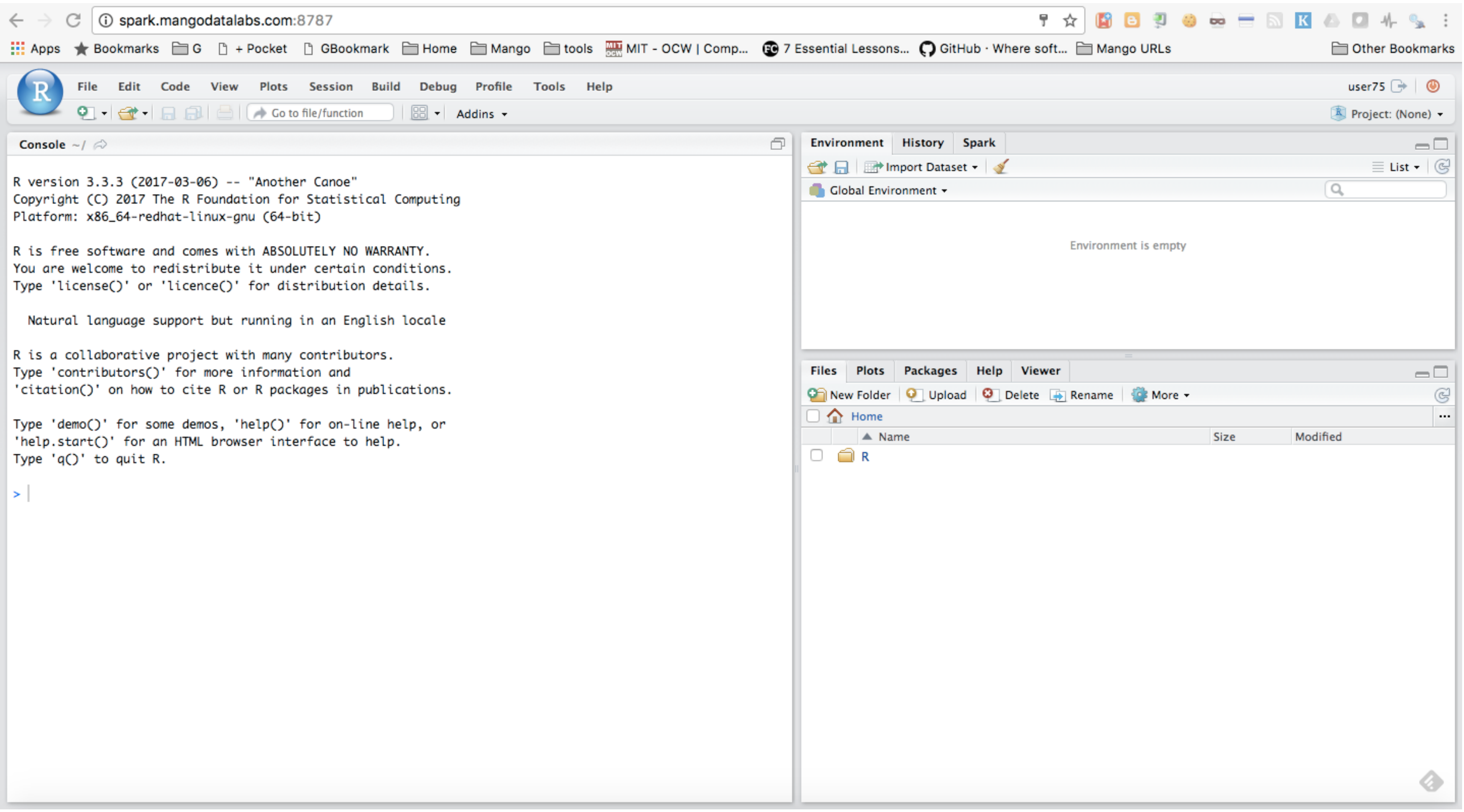
- Limited capability
- Unfamiliar code structures
- New implementations (e.g. of lm)

Sparklyr

- Rstudio with IDE integration
- Installer built in
- SparkSQL backend to **dplyr**
- Use Spark distributed ML library from R

What’s in it for you?

- If you’ve ever run out of memory in R or need to interface with a Hadoop cluster, Spark and Sparklyr may be the solution for you.
- Allows you to create and work with vast datasets
- You can work interactively or create batch jobs, all using a familiar R/dplyr syntax



```
R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Creating a spark context

```
library(sparklyr)

config <- spark_config()
config$spark.ui.port <- "1234"

sc <- spark_connect(master = "spark://…:7077",
                    config = config,
                    app_name = "<your_name>")
```

MANGO SOLUTIONS

The dplyr Package for Spark

Overview

- The Data
- Connect & Read
- The Core dplyr Functions
- SQL Functions, Joins and Sampling Data
- Save and Disconnect

Airlines Data

- Arrival and departure details for all commercial flights in US between October 1987 and April 2008.
- 120,000,000 records. **12 GB**
- stat-computing.org/dataexpo/2009/

MANGO SOLUTIONS

Connect & Read

Connecting to Spark

```
sc <- spark_connect(
          master = "…",
          app_name = "my_name"
     )
```

Importing the Data

- Multiple options for connecting to data:
  - Copy data from R data frame
  - Import from file (CSV, JSON, PARQUET)
  - Using Spark SQL
  - From a Hive table

Importing the Data - Today

- We are going to read from a parquet file
  - Column store data format
- Can be read with

```
spark_read_parquet()
```

Read a Parquet File

```
airlines <-
  spark_read_parquet(
          sc,
          name = "name_in_spark",
          path = "path/to/file"
    )
```

Importing the Data - Today

- We are also going to read from csv
- Can be read with

```
spark_read_csv()
```

The Data Object in R

- We don't work with data in R
- We work with a connection to the data
- dplyr will create Spark SQL to run
- The data will come to R (if we ask for it) after running the dplyr code

Exercise

- Ensure you have a connection to Spark
- Read the data and create a connection object to the table
- Read in the airports, carriers and plane-data CSV files
- What columns are in the airports data?
- Print the object in the console

  (DO NOT `View` THE DATA YET)

Core dplyr Functions

The Core dplyr Functions

Function	Usage
`filter`	Filter the rows of a data set
`select`	Select columns from a data set
`mutate`	Add or manipulate columns in a data set
`arrange`	Sort the data
`summarise`	Generate summaries for columns in the data

A Quick Example

```
friday <- filter(flights, DayOfWeek == 5)

friday <- select(friday, -DayOfWeek)

friday <- mutate(friday,
   Date = paste(Year, Month, DayofMonth,
            sep = "-"))

select(friday, Year,
        Month, DayofMonth, Date)
```

The Pipe Operator

- We can use dplyr as usual when working with spark
- This includes using the pipe operator (`%>%`) to simplify operations

Example with Pipes

```
flights %>%
   filter(DayOfWeek == 5) %>%
   select(-DayOfWeek) %>%
   mutate(Date = paste(Year,
   Month, DayofMonth, sep = "-"))
        %>%
   select(Year, Month,
            DayofMonth, Date)
```

Exercise

- Create a query for the flights data where:
  - depdelay > 15, depdelay < 240, dayofmonth == 15
  - Columns: year, month, arrdelay, depdelay, distance, uniquecarrier
- Ensure your query is using the pipe operator

Lazy Execution

- The SQL query will be run at the last possible moment
- If you simply print results only 10 rows will be retrieved

How Do You Get All The Data?

```
flights %>%
  filter(year > 2007) %>%
  filter(depdelay == 240) %>%
  collect()
```

What is the Code Doing?

- The dplyr functions create a Spark SQL statement
- We can see the SQL statement by using

```
show_query()
```

Exercise

- Using the code you wrote in the last exercise
  - What does the Spark SQL query look like?
  - What are the dimensions of the data?

MANGO SOLUTIONS

SQL Functions, Joins & Sampling

Translating to SQL

Operators	`+, -, *, /, %%, ^`
Mathematical functions	`abs, acos, cosh, sin, asinh, atan, atan2, atanh, ceiling, cos, cosh, cot, coth, exp, floor, log, log10, round, sign, sin, sinh, sqrt, tan, tanh`
Comparisons	`<, <=, !=, >=, >, ==, %in%`
Booleans	`&, &&, |, ||, !`
Aggregations	`mean, sum, min, max, sd, var, cor, cov, n`
Characters	`paste, tolower, toupper, nchar`
Casting	`as.double, as.integer, as.logical, as.character, as.date`

Joining and Sampling

Function	Usage
`group_by`	Apply a grouping to the data
`left_join,` `right_join,` `...`	Perform data joins (left, right, outer, ...)
`sample_n`	Sample n rows from the data

Joining

```
friday <- flights %>%
  filter(DayOfWeek == 5) %>%
  left_join(airports,
    by = c(Dest = "iata")) %>%
  filter(Origin %in%
                c("SFO", "BOS"))
```

Sampling

```
friday %>%
  sample_n(5)
```

Creating New Spark Data Frames

- When you join or sample you create a new data frame
- You may want to save this to use later
- We can do this with

```
sdf_register(r_obj, "spark_name")
```

Exercise

- Join the airport data to the flights data
- Filter the data to retain only flights originating in San Francisco and Boston
- Only retain data for Fridays
- Create a new Spark data frame containing this data

MANGO SOLUTIONS

Save & Disconnect

Saving Your Results

- When we quit Spark our data won't be saved
- If we want to keep results we need to save them
- All of our import functions have equivalent write functions

```
spark_write_parquet()
spark_write_csv()
```

Disconnecting (Don't Do This Now)

- Once we are done we need to disconnect from the spark instance

```
spark_disconnect(sc)
```

MANGO SOLUTIONS

Machine Learning with sparklyr

Overview

- Mllib
- Data Preparation
- Model Training
- Model Evaluation

MLlib

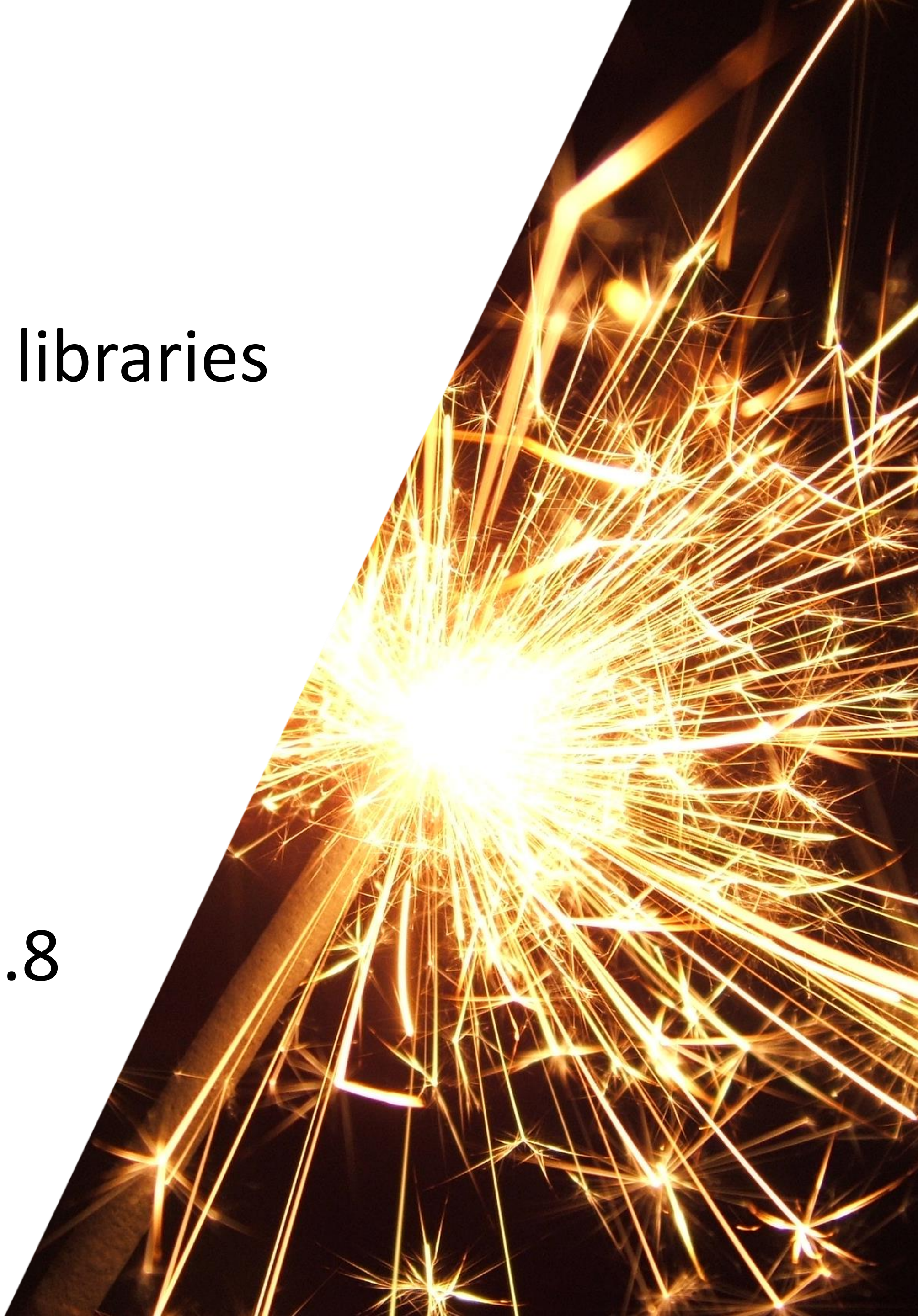
MLlib Overview

“Apache Spark's scalable machine learning library”

- Data Preparation
  - Feature transformations, data partition
- Machine Learning Algorithms:
  - glm, Random Forest, GBM, K-means, Naive Bayes...
- Model Evaluation

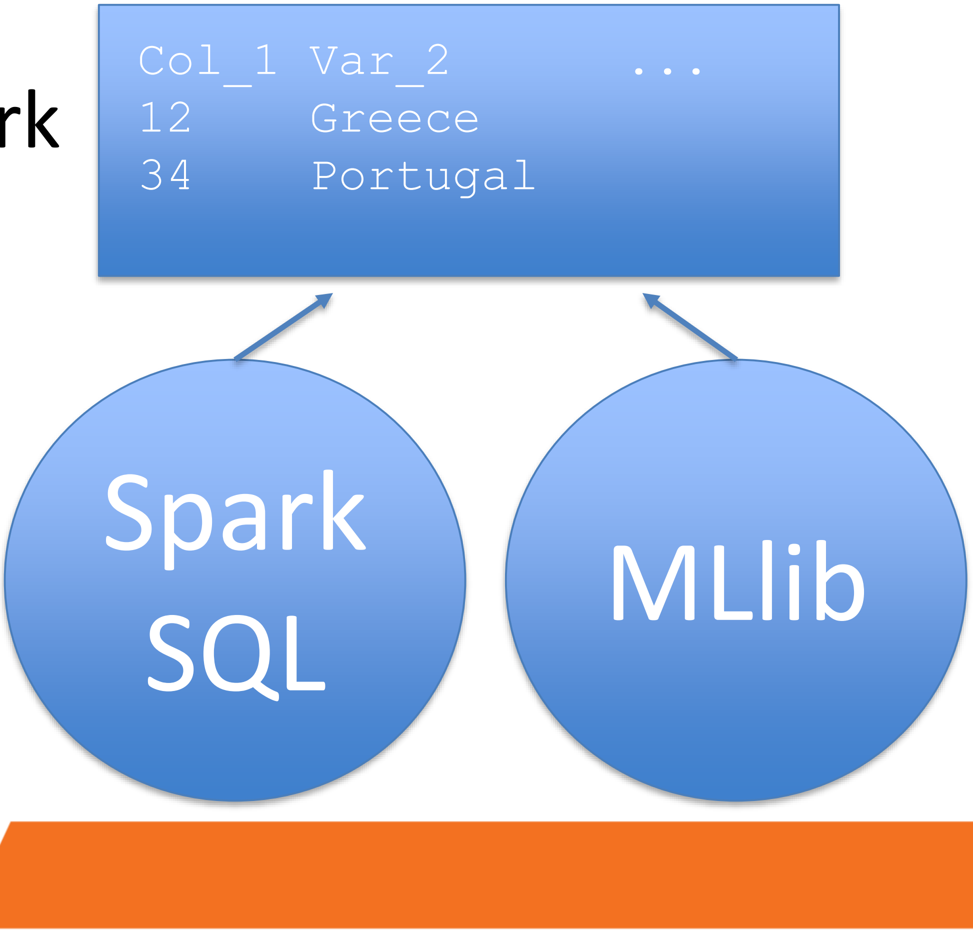
MLlib in Context

- One of the four main libraries on top of core Spark
  - **Spark SQL**
  - Spark Streaming
  - **MLlib**
  - GraphX
- Shipped with Spark 0.8 (2013)



MLlib & Spark SQL

- Completely separate way to interact with DataFrames from Spark SQL.
- We can use them together!



Aside: Spark Core Structures

- RDD: Resilent Distributed Datasets
  - The original Spark data structure
  - Immutable general purpose structure
- Dataset:
  - New in Spark 1.6
  - More performant than RDDs
- DataFrame: A Dataset of rows
  - The only structure we'll be using

MLlib in sparklyr

- Three families of functions
- Feature Transformers: `ft_`
- Spark DataFrame Manipulation: `sdf_`
- Machine Learning: `ml_`

Data Preparation

Target Creation

- To demonstrate ft_ functions try the binarizer
- For classification can use threshold a continuous variable

```
iris_tbl <- copy_to(sc, iris)

ft_binarizer(iris_tbl,
            input.col = "Sepal_Length",
            output.col = "SL_Threshold",
            threshold = 6)
```

Using MLlib in dplyr chain

- Where `mutate` maps to Hive (Spark SQL) functions, `sdf_mutate` uses MLlib
- Easier on the eye than raw `ft_` functions

```
flights %>%
  mutate(ArrDelayDb = as.numeric(ArrDelay)) %>%
  sdf_mutate(Late = ft_binarizer(ArrDelayDb,
                                 threshold = 30)) %>%
  select(ArrDelay, Late)
```

Categorical (Factor) Data

- There is no “Factor” data type.
- Character data interpreted as factor
  - No need to convert for most ML algorithms
- Can converted using one hot encoding:
  - `ft_one_hot_encoder`
  - Or `ml_create_dummy_variables`

```
ml_create_dummy_variables(iris_tbl ,
                          input = "Species")
```

Note: Need to drop reference var

Cut (bucketizer) 1

- `ft_bucketizer` works like R's cut

```
# In R
iris %>%
  mutate(SL_Group = cut(Sepal.Length,
                        breaks = 5))
# In Spark
iris_tbl %>%
  sdf_mutate(SL_Group =
ft_bucketizer(Sepal_Length,
                           splits = 4:8))
```

Cut (bucketizer) 2

```
# In R
iris %>%
  mutate(SL_Group = cut(Sepal.Length,
                        breaks = 4:8))
# In Spark
iris_tbl %>%
  sdf_mutate(SL_Group =
ft_quantile_discretizer(Sepal_Length,
                            n.levels = 5))
```

Other `ft_` functions

- `ft_one_hot_encoder`: Encoding categoricals
- `ft_tokenizer`: Splitting text into words
- `ft_string_to_index`: Category -> index and back again
- `ft_quantile_discretizer`: Faster, non-deterministic bucketizer
- `ft_vector_assembler`: Bring various vector cols back together

Exercise

Create a categorical feature in the friday dataset for morning and afternoon departures.

(hint DepTime > 1200)

Model Training

Available Algorithms

Function	Description
ml_kmeans	K-Means Clustering
ml_linear_regression	Linear Regression
ml_logistic_regression	Logistic Regression
ml_survival_regression	Survival Regression
ml_generalized_linear_regression	Generalized Linear Regression
ml_decision_tree	Decision Trees
ml_random_forest	Random Forests
ml_gradient_boosted_trees	Gradient-Boosted Trees
ml_pca	Principal Components Analysis
ml_naive_bayes	Naive-Bayes
ml_multilayer_perceptron	Multilayer Perceptron
ml_lda	Latent Dirichlet Allocation
ml_one_vs_rest	One vs Rest

Source: spark.rstudio.com

Final Data Prep

- Select **only** columns you want to use

```
select(friday , ArrDelay, DepTime)
```

- Last thing to do is partition into test and train

```
fl_part <- friday %>%
  select(ArrDelay, DepTime) %>%
  na.omit() %>%
  sdf_partition(train=0.8, test=0.2)
```

- Can have as many partitions as you like. Full cross validation not implement in R yet (cf H2O sparkling water).

Train the Model

- Similar to R. Can use formula interface.

```
model <- fl_part$train %>%
   ml_linear_regression(ArrDelay ~ DepTime)
```

- Can't use interaction (`y ~ x1:x2`) terms yet
- Alternatively, specify **Response** and **Features**

```
model <- fl_part$train %>%
 ml_linear_regression(response = "ArrDelay",
                      features = "DepTime")
```

Model Object

```
> class(model)
[1] "ml_model_linear_regression" "ml_model"
```

Works much like any other model

```
> summary(model)

Deviance Residuals::
     Min        1Q    Median        3Q      Max
 -81.260   -19.136    -8.682     5.231 1245.612

Coefficients:
               Estimate  Std. Error t value  Pr(>|t|)
(Intercept) -1.0703e+01  5.2566e-01 -20.361 < 2.2e-16 ***
DepTime      1.5007e-02  3.7059e-04  40.495 < 2.2e-16 ***
---
```

Scoring

- Use the `sdf_predict` function

```
scores <- sdf_predict(model,
                      newdata = fl_part$test) %>%
                      collect()
```

Elastic Net

- `ml_linear_regression` and `ml_logistic_regression` have lasso and ridge regression built in.
- Alpha: 0=ridge, 1=lasso
- Lambda: Sets strength of penalisation (use small numbers ~ 10^{-3} (0 for normal lm)
- `ml_generalized_linear_regression` works like glm

Exercise

Fit a ml_logisitic regression on the Friday dataset with a response of “Cancelled” predicted by departure time and destination state

MANGO SOLUTIONS

Model Evaluation

Cross Validation

- MLlib does have cross validation routines but not yet available from sparklyr
- Can use H2O’s cross validation with rsparkling

Area Under Curve

```
pred <- sdf_predict(model_class,
                    fl_part$test)

ml_binary_classification_eval(pred,
   label = "Cancelled",
   score = "probability",
   metric = "areaUnderROC")
```

MANGO SOLUTIONS

R on Spark

R on Spark Overview

- New in sparklyr 0.6
- Run arbitrary R code on all nodes
- Must have installed on all nodes
  - Packages are copied on first use (each session)
- Function with data frame in, data frame out

spark_apply

- A bit like lapply **over partitions**

```
iris2_tbl <- sdf_copy_to(
  sc, iris, repartition = 2)

spark_apply(iris2_tbl,
            function(e) {
              data.frame(N=nrow(e))
            },
            names = "n",
            packages = FALSE
)
```

spark_apply

- Partitions are independent
- R packages are copied over once per session
  - This is super slow!
- If you add columns you have to declare them
- DataFrame -> function -> DataFrame

spark_apply

- Lets you use R only functions

```
spark_apply(
  iris2_tbl,
  function(e) {
    dplyr::mutate(e,
      Rpois = rpois(nrow(e), 3))
  },
  names = c(names(iris), "Rpois"),
  packages = FALSE  )
```

MANGO SOLUTIONS

Concluding

Workshop Takeaways

- sparklyr evolving rapidly, things may change!
- Great tool for exploratory analysis on big datasets without needing to learn Scala/Java
- Built into Rstudio and Cloudera
- Can mix MLlib and Hive commands through dplyr
- Expect lots of new features soon